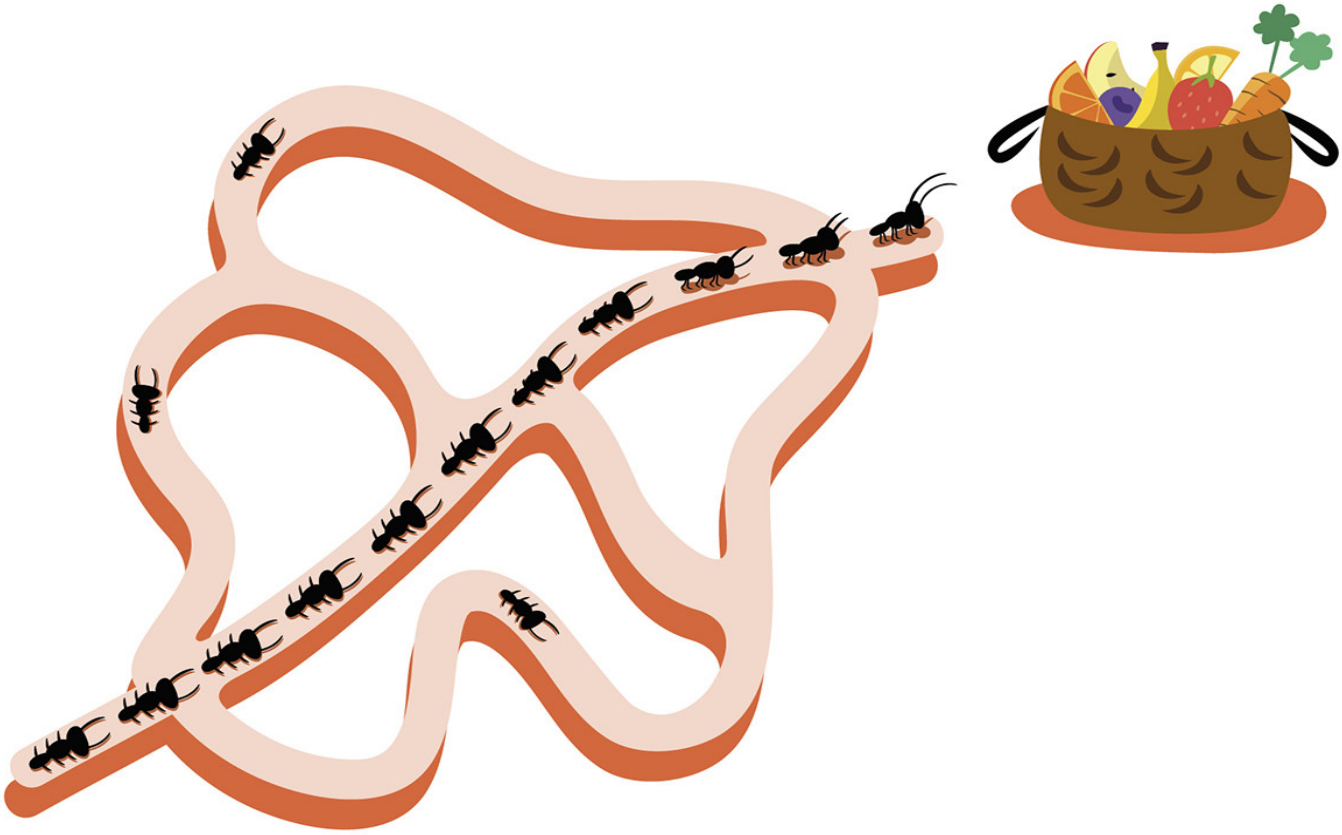
RISHAL HURBANS

Intelligenza Artificiale

spiegata in modo facile



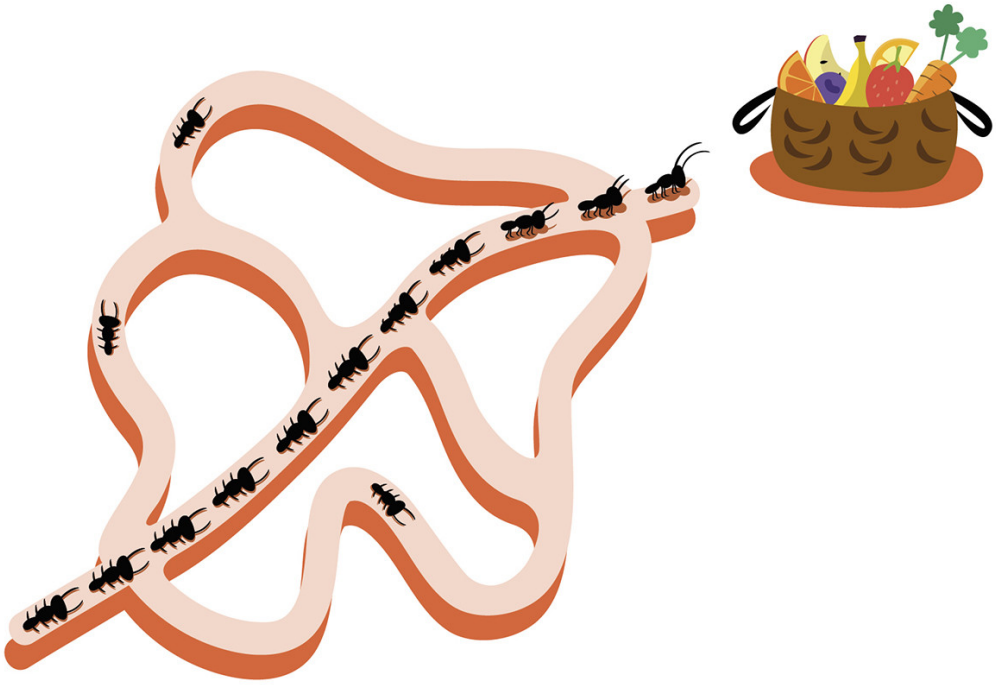
Guida illustrata per programmatori curiosi

APOGEO

RISHAL HURBANS

Intelligenza Artificiale

spiegata in modo facile



Guida illustrata per programmatori curiosi

APOGEO

Intelligenza Artificiale spiegata in modo facile
Guida illustrata per programmatori curiosi

Rishal Hurbans

APOGEO

ISBN ebook: 9788850319893

Original English–language edition published with the title “Grokking Artificial Intelligence Algorithms: Understand and apply the core algorithms of deep learning and artificial intelligence in this friendly illustrated guide including exercises and examples”. Authorized translation of the English edition (c) 2020 Manning Publications. This translation is published and sold by permission of Manning Publications, the owner of all rights to publish and sell the same.

Guarda cosa stiamo facendo su Instagram

Rimani aggiornato iscrivendoti alla nostra newsletter

~

Sai che ora facciamo anche CORSI? Dai un’occhiata al calendario.

Ai miei genitori, Pranil e Rekha. Che hanno fatto la differenza.

Questa Premessa ha lo scopo di descrivere l'evoluzione della tecnologia, la nostra necessità di automatizzare le cose e la nostra responsabilità di prendere decisioni etiche utilizzando l'intelligenza artificiale per costruire il nostro futuro.

La nostra ossessione per la tecnologia e l'automazione

Nel corso della storia, abbiamo avuto la necessità di risolvere i problemi riducendo il lavoro manuale e la fatica. Abbiamo sempre lottato per la sopravvivenza e puntato alla conservazione delle energie attraverso lo sviluppo di nuovi strumenti e l'automazione delle nostre attività. Alcuni potrebbero osservare che siamo menti meravigliose, che cercano l'innovazione attraverso la risoluzione creativa di problemi o opere creative di letteratura, musica e arte, ma questo libro non è stato scritto per discutere questioni filosofiche sul genere umano. Questa è una panoramica degli approcci di intelligenza artificiale (IA) che possono essere sfruttati per affrontare problemi concreti. Risolviamo problemi difficili per renderci la vita più facile, più sicura, più sana, più appagante e più piacevole. Tutti i progressi che vedete oggi nella storia e in tutto il mondo, inclusa l'intelligenza artificiale, rispondono a specifici bisogni di individui, comunità e nazioni.

Per plasmare il nostro futuro, dobbiamo comprendere alcune pietre miliari chiave del nostro passato. In molte rivoluzioni, l'innovazione ha cambiato il modo in cui viviamo e ha modellato il modo in cui interagiamo con il mondo e il modo stesso in cui lo concepiamo. Continuiamo a farlo mentre facciamo evolvere e miglioriamo gli strumenti che utilizziamo, i quali aprono sempre nuove possibilità per il futuro (Figura 0.1).

Questa breve visione di alto livello sulla storia e la filosofia ha esclusivamente lo scopo di offrire una comprensione di base delle tecnologie e dell'intelligenza artificiale e per stimolare una riflessione su processi decisionali responsabili quando si intraprendono i propri progetti.

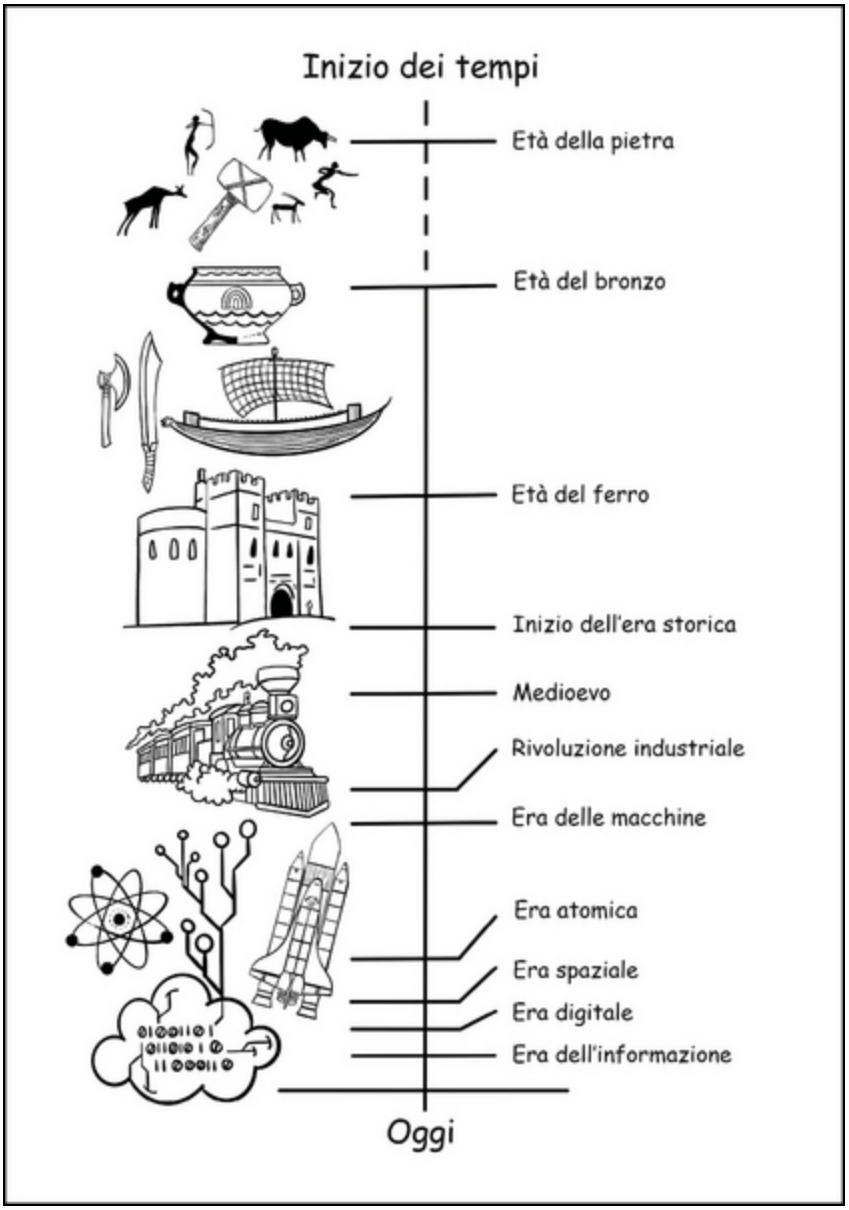


Figura P.1 Una breve cronologia dei miglioramenti tecnologici nella storia.

Nella sequenza temporale rappresentata in figura, notate la compressione delle pietre miliari nei tempi più recenti. Negli ultimi trent'anni, i progressi più notevoli sono stati l'evoluzione dei microchip, la diffusione dei personal computer, il boom dei dispositivi connessi in rete e la digitalizzazione delle industrie, per abbattere i confini fisici e connettere il mondo. Questi sono anche i motivi per cui

l’intelligenza artificiale è diventata un’area sempre più attuabile e opportuna.

- Internet ha connesso il mondo e ha reso possibile la raccolta di enormi quantità di dati su quasi tutto.
- I progressi nell’hardware informatico ci hanno dato i mezzi per applicare gli algoritmi già noti utilizzando però l’enorme quantità di dati che abbiamo raccolto, scoprendo nel frattempo nuovi algoritmi.
- Le industrie hanno visto la necessità di sfruttare i dati e gli algoritmi per prendere decisioni migliori, per risolvere problemi più difficili, per offrire soluzioni migliori e per ottimizzare la nostra vita, come abbiamo sempre fatto dall’inizio dell’umanità.

Sebbene tendiamo a considerare il progresso tecnologico come lineare, esaminando la nostra storia, scopriamo che è più probabile che il nostro progresso sia e sarà esponenziale (Figura P.2). I progressi tecnologici procederanno più velocemente ogni anno che passa. È necessario apprendere nuovi strumenti e tecniche, ma alla base di tutto ci sono i *fondamenti della risoluzione dei problemi*.

Questo libro include concetti di base per la risoluzione di problemi difficili, ma punta anche a rendere più facile l’apprendimento dei concetti più complessi.

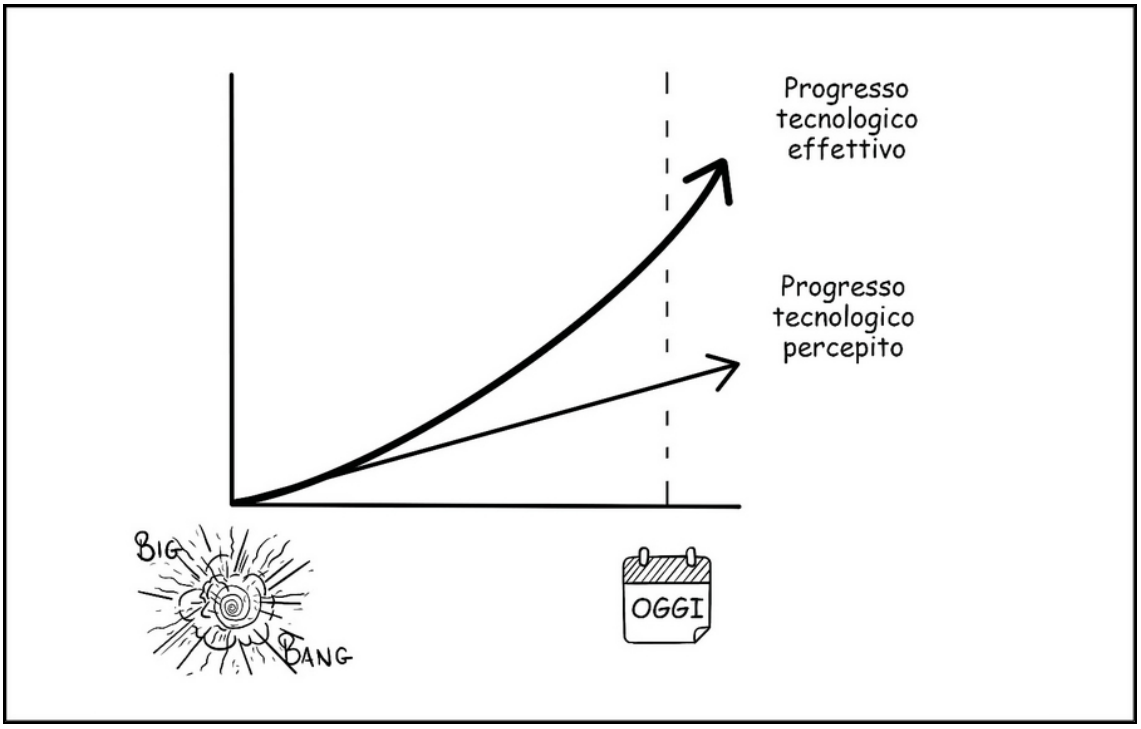


Figura P.2 Progresso tecnologico percepito rispetto al progresso tecnologico effettivo.

L'automazione può essere percepita in modo differente. Per chi si occupa di tecnologie, l'automazione può significare scrivere script che rendano più fluidi e "sicuri" (meno soggetti a errori) lo sviluppo, l'implementazione e la distribuzione del software. Per un ingegnere, può significare semplificare una linea di produzione per garantire una maggiore produttività o meno difetti. Per un agricoltore, può significare utilizzare dei mezzi per ottimizzare la resa dei raccolti attraverso trattori e sistemi di irrigazione automatici. L'automazione è qualsiasi soluzione che riduca la necessità di energie (umane) per favorire la produttività o per ottenere un valore aggiunto rispetto a quello del lavoro manuale (Figura P.3).

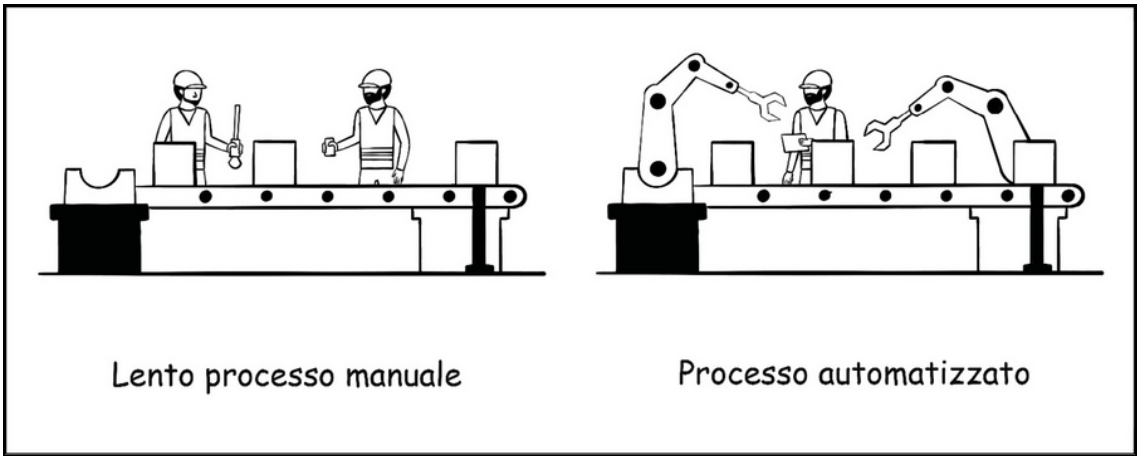


Figura P.3 Processi manuali vs. processi automatizzati.

Se pensiamo alle ragioni per *non* automatizzare, una di queste è il fatto che una persona può svolgere meglio un compito, con minori possibilità di fallimento e maggiore accuratezza, se è necessaria capacità d'intuito, quando è richiesto un pensiero creativo astratto o quando è importante tenere conto delle interazioni sociali e della natura delle persone.

Gli infermieri non si limitano a espletare compiti, ma creano legami e si prendono cura dei pazienti. Gli studi dimostrano che l'interazione umana da parte di persone premurose è un fattore importante nel processo di guarigione. Gli insegnanti non si limitano a travasare conoscenza, ma trovano modi creativi per presentare la conoscenza, per fare da mentori e per guidare gli studenti in base ai loro interessi, capacità, personalità. Detto questo, pensiamo che ci sia un posto per l'automazione tecnologica e un posto per le persone. Con le innovazioni di oggi, l'automazione tecnologica ci affiancherà qualsiasi sia la nostra occupazione.

Etica, questioni legali e responsabilità

Forse vi starete chiedendo perché inserire in un libro tecnico un paragrafo dedicato all'etica e alla responsabilità. Bene, mentre avanziamo verso un mondo in cui la tecnologia si intreccia con il nostro modo di vivere, coloro che creano tecnologie hanno nelle loro mani più potere di quanto immaginino. Piccoli contributi possono avere enormi conseguenze. È pertanto importante che le nostre intenzioni siano benevole e che il risultato del nostro lavoro non sia dannoso (Figura P.4).

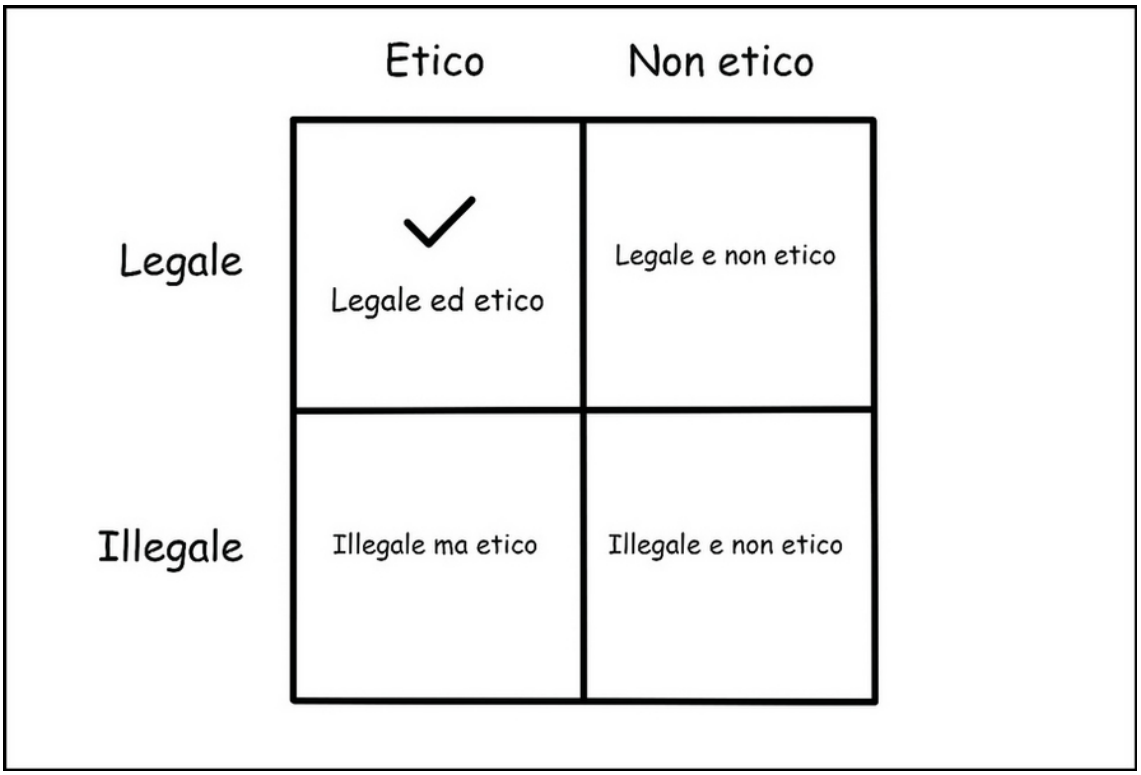


Figura P.4 L'obiettivo è un uso etico e legale delle tecnologie.

Intenzione e impatto: quale visione e quali obiettivi

Quando sviluppate qualcosa, per esempio un nuovo prodotto fisico, servizio o software, c'è sempre una domanda sull'*intenzione* che c'è

dietro. State sviluppando un software che influisce positivamente sul mondo o la vostra intenzione è malevola? Avete pensato all'impatto più ampio di ciò che state sviluppando? Le aziende trovano sempre nuovi modi per diventare più redditizie e potenti, che è il focus del loro sviluppo. Usano strategie per determinare i modi migliori per battere la concorrenza, per acquisire più clienti e per diventare ancora più influenti. Detto questo, le aziende devono chiedersi se le loro intenzioni sono pure, legate non solo alla sua stessa sopravvivenza, ma anche al bene dei loro clienti e della società in generale. Molti famosi scienziati, ingegneri e chi si occupa di tecnologie hanno espresso la necessità di controllare l'utilizzo dell'intelligenza artificiale per prevenirne ogni uso improprio. Ma anche come individui abbiamo l'obbligo etico di fare ciò che è giusto e di definire un solido insieme di valori fondamentali. Quando vi viene chiesto di fare qualcosa che vìola i vostri principi, è importante che diate voce a questi principi.

Uso improprio: protezione contro un uso dannoso

È importante identificare e proteggersi contro ogni uso improprio. Anche se può sembrare ovvio e facile da realizzare, è difficile capire come gli altri utilizzeranno la "cosa" che state creando e, ancora più difficile, prevedere se è in linea con i vostri valori e quelli della vostra azienda.

Un esempio è l'altoparlante, inventato da Peter Jensen nel 1915. L'altoparlante era originariamente chiamato Magnavox, inizialmente utilizzato per riprodurre musica d'opera a grandi folle a San Francisco, il che è un uso davvero benevolo della tecnologia. Il regime nazista, in Germania, tuttavia, aveva altre idee: collocò degli altoparlanti nei luoghi pubblici, in modo tale che tutti potessero ascoltare i discorsi e gli annunci di Hitler. Poiché i suoi monologhi erano diventati non evitabili, sempre più persone divennero recettive alle idee di Hitler e,

da allora, il regime nazista ottenne grande sostegno in Germania. Questo non è ciò per cui Jensen immaginava di usare la sua invenzione, ma non avrebbe potuto fare molto al riguardo.

I tempi sono cambiati e abbiamo un maggiore controllo sulle cose che costruiamo, in particolare sul software. Rimane ancora difficile immaginare come possa essere utilizzata la tecnologia che costruite, ma è quasi sicuro che qualcuno riuscirà a utilizzarla in un modo che non avevate previsto, con conseguenze positive o negative. Alla luce di questo fatto, noi, come professionisti del settore tecnologico e le aziende per le quali lavoriamo, dobbiamo pensare a dei modi per mitigare il più possibile ogni uso malevolo.

Pregiudizi involontari: creare soluzioni per tutti

Quando creiamo sistemi di intelligenza artificiale, utilizziamo la nostra conoscenza dei contesti e dei domini. Inoltre, utilizziamo algoritmi che cercano modelli nei dati e li sfruttano. Non si può negare che i pregiudizi siano "tutti intorno a noi". Può trattarsi di pregiudizi nei confronti di una persona o di un gruppo, fra cui, a titolo esemplificativo e non esaustivo, di genere, di razza e di convinzioni. Molti di questi pregiudizi derivano da comportamenti che emergono nelle interazioni sociali, da eventi storici e da opinioni culturali e politiche. Questi pregiudizi influenzano i dati che raccogliamo. Poiché gli algoritmi di intelligenza artificiale operano su questi dati, è solo inevitabile che la macchina "apprenda" questi pregiudizi. Da un punto di vista tecnico, possiamo anche progettare perfettamente il sistema, ma in fin dei conti, sono esseri umani quelli che interagiscono con questi sistemi ed è nostra responsabilità ridurre al minimo i pregiudizi. Gli algoritmi che utilizziamo valgono solo quanto i dati che forniamo loro. Comprendere i dati e il contesto in cui vengono utilizzati è il primo passo per combattere i pregiudizi, e questa comprensione vi

aiuterà a costruire soluzioni migliori, perché sarete divenuti esperti nello spazio del problema. Fornendo dati equilibrati con la minor quantità possibile di pregiudizi otterrete soluzioni migliori.

Le leggi, la privacy e il consenso: l'importanza dei valori fondamentali

L'aspetto legale di ciò che facciamo è estremamente importante. Le leggi disciplinano ciò che possiamo e non possiamo fare, nell'interesse della società nel suo complesso. A causa del fatto che molte leggi sono state scritte in un'epoca in cui i computer e Internet non erano così cruciali per la nostra vita come sono oggi, troviamo molte aree grigie nel modo in cui sviluppiamo le tecnologie e in ciò che ci è permesso di fare con quelle tecnologie. Detto questo, le leggi stanno lentamente cambiando, per adattarsi alle rapide innovazioni tecnologiche.

Per esempio, tramite le nostre interazioni su computer, telefoni cellulari e altri dispositivi stiamo compromettendo la nostra privacy quasi ogni istante. Stiamo trasmettendo una grande quantità di informazioni su noi stessi, alcune più personali di altre. Come vengono elaborati e archiviati i dati? Dovremmo considerare questi fatti quando costruiamo soluzioni. Le persone dovrebbero poter scegliere quali dati possono essere acquisiti, elaborati e archiviati su di loro; come vengono utilizzati; chi può potenzialmente accedere a tali dati. Nella mia esperienza, le persone generalmente accettano quelle soluzioni che utilizzano i loro dati per migliorare i prodotti che esse utilizzano e aggiungere benefici alla loro vita. Ma è importante capire che le persone accettano di più quando viene data loro una possibilità di scelta, e poi quella scelta viene rispettata.

Singolarità: esplorare l'ignoto

La *singolarità* è l'idea che siamo in grado di creare un'intelligenza artificiale a tal punto intelligente da essere in grado di migliorare se stessa ed espandere la propria intelligenza fino a uno stadio superiore. Il timore è che qualcosa di tale portata non possa essere compreso da noi esseri umani, al punto che potrebbe cambiare la civiltà così come la conosciamo per ragioni che non possiamo nemmeno comprendere. Alcune persone temono che questa intelligenza artificiale possa finire per considerare gli esseri umani come una minaccia; altri pensano che per una superintelligenza noi potremmo essere ciò che le formiche sono per noi. Non prestiamo particolare attenzione alle formiche, né ci preoccupiamo di come vivono, ma se ci danno fastidio, le combattiamo.

Indipendentemente dal fatto che queste ipotesi siano rappresentazioni accurate del futuro o meno, dobbiamo essere responsabili e riflettere sulle decisioni che prendiamo, poiché alla fine esse influenzano una persona, un gruppo di persone o il mondo in generale.

Scrivere questo libro è stata una delle cose più impegnative ma gratificanti che abbia mai fatto fino a oggi. Avevo bisogno di trovare il tempo, e non ne avevo, trovare il giusto spazio mentale mentre mi destreggiavo fra molti contesti e trovare la motivazione mentre ero coinvolto nella realtà della mia vita. Non avrei potuto farlo senza alcune persone straordinarie. Ho imparato e sono cresciuto grazie a questa esperienza. Grazie a Bert Bates, per essere stato un fantastico editor e mentore per me. Da te ho imparato moltissimo sull'insegnamento efficace e sulla comunicazione scritta. Le nostre discussioni e dibattiti e la tua empatia durante tutto il processo di scrittura hanno contribuito a plasmare questo libro, per farlo diventare quello che è. Ogni progetto ha bisogno di qualcuno organizzato, con il polso della situazione per assicurarsi che giunga a termine. Per questo, vorrei ringraziare Elesha Hyde, il mio editor di sviluppo. Lavorare con te è stato un vero piacere. Fornisci sempre indicazioni e spunti interessanti per il mio lavoro. Abbiamo sempre bisogno di persone sulle quali far "rimbalzare" le idee e chi meglio dei nostri amici possiamo infastidire? Vorrei ringraziare in particolare Hennie Brink per essere sempre stata una grande cassa di risonanza e un pilastro di supporto. Poi, vorrei ringraziare Frances Buontempo e Krzysztof Kamyczek per aver fornito critiche costruttive e feedback oggettivi sulla scrittura e sugli aspetti tecnici. Il vostro intervento ha contribuito a colmare le lacune e a rendere l'insegnamento più accessibile. Vorrei

anche ringraziare Deirdre Hiam, il mio project manager; il mio revisore, Ivan Martinovic; il mio redattore Kier Simpson; il mio correttore di bozze, Jason Everett.

Infine, vorrei ringraziare tutti i revisori che si sono presi il tempo di leggere il mio manoscritto durante lo sviluppo e hanno fornito un prezioso feedback che ha migliorato il libro, in un modo o nell'altro: Andre Weiner, Arav Agarwal, Charles Soetan, Dan Sheikh, David Jacobs, Dhivya Sivasubramanian, Domingo Salazar, GandhiRajan, Helen Mary Barrameda, James Zhijun Liu, Joseph Friedman, Jousef Murad, Karan Nih, Kelvin D. Meeks, Ken Byrne, Krzysztof Kamyczek, Kyle Peterson, Linda Ristevski, Martin Lopez, Peter Brown, Philip Patterson, Rodolfo Allendes, Tejas Jain e Weiran Deng.

Intelligenza artificiale spiegata in modo facile è stato scritto e illustrato per rendere più accessibili la comprensione e l'implementazione degli algoritmi di intelligenza artificiale e dei loro usi nella risoluzione dei problemi, avendo in mente una persona mediamente preparata nel campo tecnologico. Per questo abbiamo impiegato analogie riconoscibili, esempi pratici e spiegazioni visuali.

A chi è rivolto questo libro

Intelligenza artificiale spiegata in modo facile è dedicato agli sviluppatori e a chiunque nel settore del software desideri scoprire i concetti e gli algoritmi che sono alla base dell'intelligenza artificiale, privilegiando gli esempi pratici e le spiegazioni visuali rispetto agli approfondimenti teorici e alle dimostrazioni matematiche.

Questo libro è rivolto a chiunque abbia una comprensione dei concetti di base della programmazione: variabili, tipi di dati, array, istruzioni condizionali, iteratori, classi e funzioni. È sufficiente avere una certa esperienza in un qualsiasi linguaggio di programmazione, e avere una conoscenza di base di concetti matematici come le variabili dei dati, la rappresentazione delle funzioni e la rappresentazione di dati e funzioni su un grafico.

Come è organizzato questo libro

Questo libro è formato da dieci capitoli, ognuno dei quali è incentrato su un diverso algoritmo di intelligenza artificiale o approccio algoritmico. Il materiale copre inizialmente algoritmi e concetti fondamentali, che costituiscono la base per l'apprendimento degli algoritmi più sofisticati, trattati più avanti.

- Capitolo 1: introduce i concetti fondamentali che circondano i dati, i tipi di problemi, le categorie di algoritmi e i paradigmi e i casi d'uso per gli algoritmi di intelligenza artificiale.
- Capitolo 2: tratta i concetti fondamentali delle strutture di dati e gli approcci degli algoritmi di ricerca primitivi, più i loro usi.
- Capitolo 3: va oltre gli algoritmi di ricerca primitivi e introduce algoritmi di ricerca più avanzati, per trovare soluzioni in modo più ottimale e anche in un ambiente competitivo.
- Capitolo 4: approfondisce il funzionamento degli algoritmi genetici, nei quali le soluzioni ai problemi vengono generate in modo iterativo e migliorate imitando l'evoluzione naturale.
- Capitolo 5: è una continuazione del capitolo sugli algoritmi genetici, ma affronta concetti avanzati che riguardano il modo in cui i passaggi dell'algoritmo possono essere adattati per risolvere diversi tipi di problemi in modo più ottimale.
- Capitolo 6: scava nel concetto di intelligenza di sciame e analizza il modo in cui l'algoritmo di ottimizzazione a colonia di formiche utilizza una teoria sulla vita e il lavoro delle formiche per risolvere problemi difficili.
- Capitolo 7: continua con gli algoritmi a sciame mentre approfondisce i problemi di ottimizzazione e come vengono risolti utilizzando l'ottimizzazione particellare, poiché cerca buone soluzioni in ampi spazi di ricerca.

- Capitolo 8: esamina un flusso di lavoro di machine learning per realizzare la preparazione, l'elaborazione, la modellazione e il collaudo dei dati, con lo scopo di risolvere i problemi di regressione con la regressione lineare e i problemi di classificazione con gli alberi decisionali.
- Capitolo 9: tratta l'intuizione, i passaggi logici e i calcoli matematici nell'addestramento e nell'utilizzo di una rete neurale artificiale in grado di trovare schemi nei dati e fare previsioni; evidenziando la sua collocazione in un flusso di lavoro di machine learning.
- Capitolo 10: tratta l'idea dell'apprendimento per rinforzo, derivata dalla psicologia comportamentale e impiega l'algoritmo Q-Learning per consentire agli agenti di apprendere le decisioni buone e cattive da impiegare in un ambiente.

I capitoli sono pensati per essere letti dall'inizio alla fine, in sequenza. I concetti e le conoscenze si sviluppano di capitolo in capitolo. È utile fare riferimento al codice Python nel repository dopo aver letto ogni capitolo, per sperimentare e ottenere informazioni pratiche utili per implementare il rispettivo algoritmo.

Il codice

Questo libro contiene frammenti di pseudocodice per permettervi di concentrarvi sui concetti e sul pensiero logico che è alla base degli algoritmi, nonché per garantire che il codice sia accessibile a chiunque, indipendentemente dal linguaggio di programmazione scelto. Lo pseudocodice è un modo informale per descrivere le istruzioni. È pensato per essere più leggibile e comprensibile, fondamentalmente più umano.

Detto questo, tutti gli algoritmi descritti nel libro hanno esempi di codice Python disponibili su Github (http://mng.bz/Vgr0) e sul sito di Apogeo all'indirizzo https://bit.ly/apo-aisf. Le istruzioni e i commenti di installazione presenti nel codice sorgente hanno lo scopo di supportare l'utente nell'apprendimento. Un possibile approccio all'apprendimento sarebbe quello di leggere ogni capitolo e poi fare riferimento al codice per consolidare la comprensione dei rispettivi algoritmi.

Il codice sorgente Python vuole essere solo un'indicazione sul modo in cui gli algoritmi potrebbero essere implementati. Questi esempi sono quindi *finalizzati all'apprendimento e NON sono adatti all'uso in produzione*. Il codice è stato scritto come strumento per l'insegnamento. Per i progetti che entreranno in produzione consiglio l'utilizzo di librerie e framework ben consolidati, poiché di solito sono ottimizzati in termini di prestazioni, ben collaudati e ben supportati.

L'autore

Rishal è affascinato fin dall'infanzia dai computer, dalla tecnologia e dalle idee folli. Nel corso della sua carriera è stato coinvolto nella leadership di team e progetti, nell'ingegneria del software, nella pianificazione strategica e nella progettazione end-to-end di soluzioni per varie aziende di livello internazionale. È stato anche responsabile della crescita attiva di una cultura di pragmatismo, apprendimento e sviluppo delle competenze all'interno della sua azienda, comunità e settore.

Rishal ha una passione per le dinamiche e le strategie aziendali, lo sviluppo di persone e team, il pensiero progettuale, l'intelligenza artificiale e la filosofia. Rishal ha fondato vari prodotti digitali per aiutare le persone e le aziende a essere più produttive e a concentrarsi

su ciò che è più importante. Ha tenuto decine di conferenze in tutto il mondo, sempre con lo scopo di rendere più accessibili i concetti più complessi e di aiutare le persone a elevarsi.

Potete visitare il suo sito all'indirizzo https://rhurbans.com

Concetti di intelligenza artificiale

Che cos'è l'intelligenza artificiale?

L'intelligenza è un mistero, un concetto che non ha una definizione comune. Filosofi, psicologi, scienziati e ingegneri hanno tutti opinioni diverse su che cosa sia e come emerga. Vediamo l'intelligenza all'opera nella natura che ci circonda, nei gruppi di creature che collaborano, e vediamo l'intelligenza nel modo in cui gli esseri umani pensano e si comportano. In generale, ciò che è autonomo e adattivo è considerato intelligente. *Autonomo* significa che non ha bisogno di ricevere costantemente istruzioni; *adattivo* significa che può cambiare il proprio comportamento al variare dell'ambiente o dello spazio del problema. Osservando all'opera gli organismi viventi e le macchine, vediamo che l'elemento centrale per il loro funzionamento sono i dati. Le immagini che vediamo sono dati; i suoni che udiamo sono dati; le misurazioni che effettuiamo delle cose che ci circondano sono dati. Consumiamo dati, elaboriamo dati e prendiamo decisioni sulla base di dati; quindi, una comprensione fondamentale dei concetti che circondano i dati è importante per comprendere gli algoritmi di intelligenza artificiale.

Definizione di intelligenza artificiale

Alcune persone sostengono che non capiamo che cosa sia l'intelligenza artificiale perché facciamo fatica a definire l'intelligenza stessa. Salvador Dalí credeva che un attributo dell'intelligenza fosse

l'ambizione; disse: "L'intelligenza senza ambizione è come un uccello senza ali". Albert Einstein credeva che nell'intelligenza un fattore importante fosse l'immaginazione; disse: "Il vero segno dell'intelligenza non è la conoscenza, ma l'immaginazione". E Stephen Hawking ha detto: "L'intelligenza è la capacità di adattamento", che si concentra sulla capacità di adattarsi ai cambiamenti nel mondo. Queste tre grandi menti avevano visioni diverse sull'intelligenza. Quindi, ancora privi di una vera risposta definitiva su che cosa sia l'intelligenza, sappiamo almeno che basiamo la nostra comprensione dell'intelligenza sugli esseri umani come specie dominante (la più intelligente).

Per il bene della nostra salute mentale e per attenerci ad applicazioni pratiche, in questo libro definisco vagamente l'intelligenza artificiale come un sistema sintetico che esibisce un comportamento "intelligente". Invece di cercare di definire qualcosa come dotato o meno di intelligenza artificiale, facciamo riferimento alla sua somiglianza con l'intelligenza. Qualcosa potrebbe mostrare alcuni aspetti dell'intelligenza perché ci aiuta a risolvere problemi difficili e ci fornisce valore e utilità. Di solito, le implementazioni dell'intelligenza artificiale che simulano la vista, l'udito e altri sensi sono viste come esempi di intelligenza artificiale. Anche le soluzioni in grado di apprendere autonomamente adattandosi a nuovi dati e ambienti sono considerate esempi di intelligenza artificiale.

Ecco alcuni esempi di cose che esibiscono un'intelligenza artificiale:

- un sistema che riesce a giocare a più giochi complessi;
- un sistema in grado di individuare un tumore;
- un sistema che genera opere d'arte sulla base di pochi input;
- un'auto a guida autonoma.

Douglas Hofstadter ha detto: "L'intelligenza artificiale è tutto ciò che non è stato ancora fatto". Negli esempi appena citati, un'auto a guida

autonoma può sembrare intelligente, perché non è stata ancora realizzata. Allo stesso modo, un computer che somma numeri era considerato intelligente qualche tempo fa, mentre ora la cosa è data per scontata.

Il fatto è che *intelligenza artificiale* è un termine ambiguo, che significa cose differenti per persone, settori e discipline differenti. Gli algoritmi contenuti in questo libro sono stati classificati come algoritmi di intelligenza artificiale nel passato o nel presente; se poi rientrino o meno in una definizione specifica di intelligenza artificiale ciò non ha molta importanza. Quello che conta è che siano utili per risolvere problemi difficili.

I dati come elementi fondamentali per gli algoritmi di intelligenza artificiale

I dati sono l'input dei meravigliosi algoritmi che sono capaci di imprese quasi magiche. Ma con dati errati, mal rappresentati o mancanti, gli algoritmi funzionano male. In pratica il risultato è direttamente proporzionale alla qualità dei dati forniti. Il mondo è pieno di dati e quei dati esistono in forme che non possiamo nemmeno immaginare. I dati possono rappresentare valori misurati numericamente, come la temperatura attuale nell'Artico, il numero di pesci in uno stagno o l'età di una persona in giorni. Tutti questi esempi implicano l'acquisizione di valori numerici accurati, basati su fatti. È difficile interpretare male questi dati. La temperatura di un luogo specifico in un momento specifico è assolutamente oggettiva e non è soggetta ad alcuna distorsione. Questi dati sono chiamati *dati quantitativi*.

I dati possono anche rappresentare valori dati da osservazioni, come il profumo di un fiore o la propria concordanza con le idee di un politico. Questi dati sono chiamati *dati qualitativi*, e a volte sono

difficili da interpretare, perché non descrivono una verità oggettiva, ma una percezione della verità da parte di qualcuno. La Figura 1.1 illustra alcuni esempi di dati quantitativi e qualitativi.

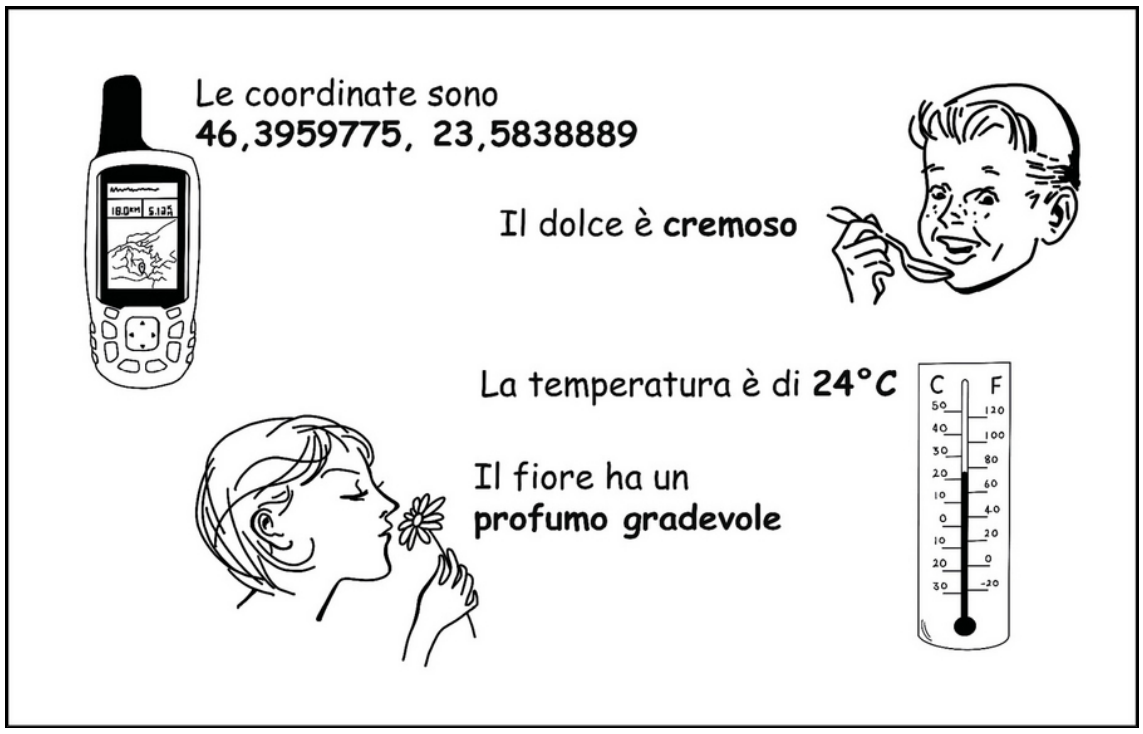


Figura 1.1 Esempi di diversi tipi di dati.

I dati sono fatti grezzi sulle cose, quindi le registrazioni di solito non contengono pregiudizi. Nel mondo reale, tuttavia, i dati vengono raccolti, registrati e correlati da persone in base a un determinato contesto, con una comprensione specifica di come i dati potranno essere utilizzati. L'atto di trarre idee significative per rispondere a domande basate sui dati crea *informazioni*. Inoltre, l'atto di utilizzare le informazioni e le esperienze, per applicarle consapevolmente crea *conoscenza*. Questo è, almeno in parte, ciò che cerchiamo di simulare con gli algoritmi di intelligenza artificiale.

La Figura 1.2 mostra come possono essere interpretati i dati quantitativi e qualitativi. Per misurare dati quantitativi vengono solitamente usati strumenti standardizzati, come orologi, calcolatrici e

bilance, mentre per creare dati qualitativi sono solitamente usati i sensi (l'olfatto, l'udito, del gusto, il tatto e la vista), o anche i nostri pensieri.

Dati, informazioni e conoscenza possono essere interpretati in modo differente da persone differenti, in base al loro livello di comprensione di quel dominio e alla loro visione del mondo, e questo fatto ha conseguenze sulla qualità delle soluzioni, cosa che rende estremamente importante l'aspetto scientifico della creazione di una tecnologia. Seguendo processi scientifici ripetibili per acquisire dati, condurre esperimenti e riportare accuratamente le misurazioni, possiamo garantire risultati più accurati e soluzioni migliori ai problemi che richiedono l'elaborazione di dati tramite algoritmi.

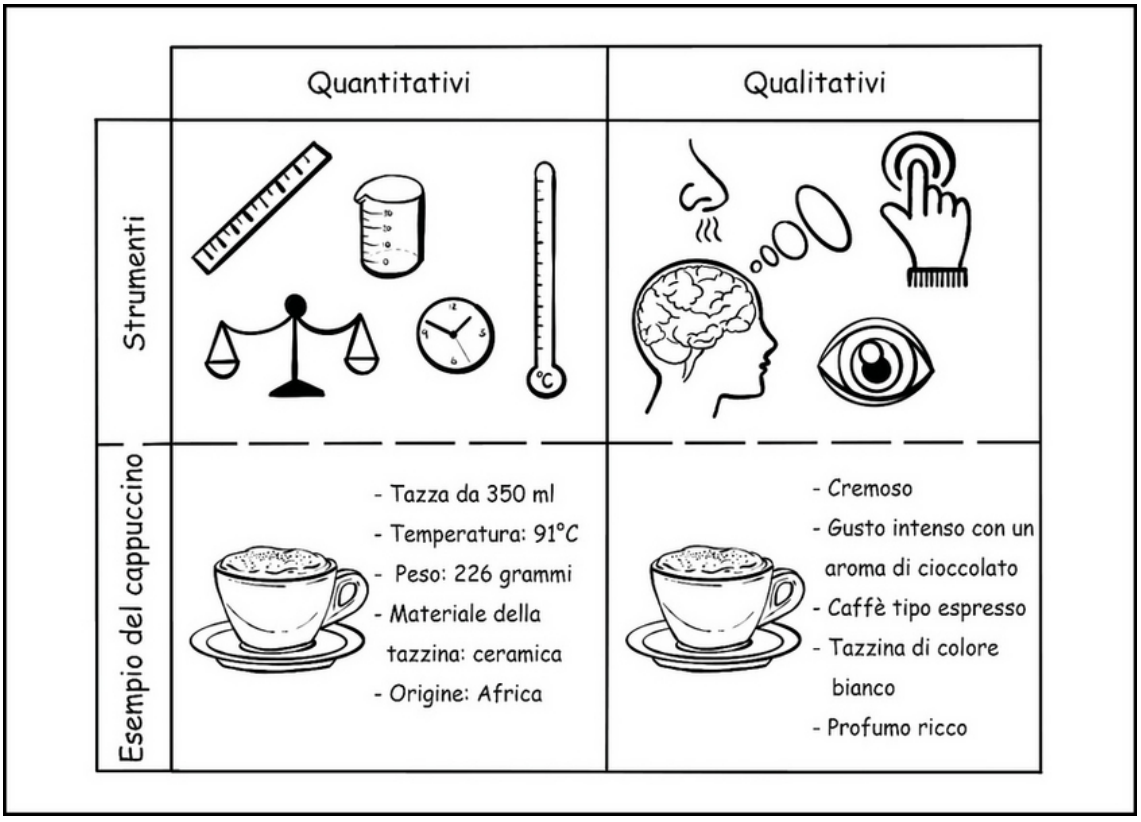


Figura 1.2 Dati qualitativi e quantitativi.

Gli algoritmi come le istruzioni di una ricetta

Ora abbiamo una definizione almeno vaga di che cos'è l'intelligenza artificiale e ci siamo resi conto della grande importanza dei dati. Poiché in questo libro esploreremo diversi algoritmi di intelligenza artificiale, ora è utile capire esattamente che cos'è un algoritmo. Un *algoritmo* è una sequenza di istruzioni e regole che consentono di raggiungere un determinato obiettivo. Gli algoritmi, in genere, accettano degli input e, dopo un numero finito di passaggi, nei quali l'algoritmo progredisce attraverso vari stati, produce un output.

Anche qualcosa di semplice come leggere un libro può essere rappresentato come un algoritmo. Ecco un esempio dei passaggi coinvolti nella lettura di questo libro.

1. Trova il libro *Intelligenza artificiale spiegata in modo facile*.
2. Apri il libro.
3. Fintantoché vi sono pagine ancora da leggere,
   a. leggi la pagina;
   b. va alla pagina successiva;
   c. rifletti su quello che hai imparato.
4. Pensa a come applicare nel mondo reale quanto hai appreso.

Un algoritmo può essere considerato come una ricetta (Figura 1.3). Dati alcuni ingredienti e strumenti di input e le istruzioni per creare un piatto, l'output è il pasto.

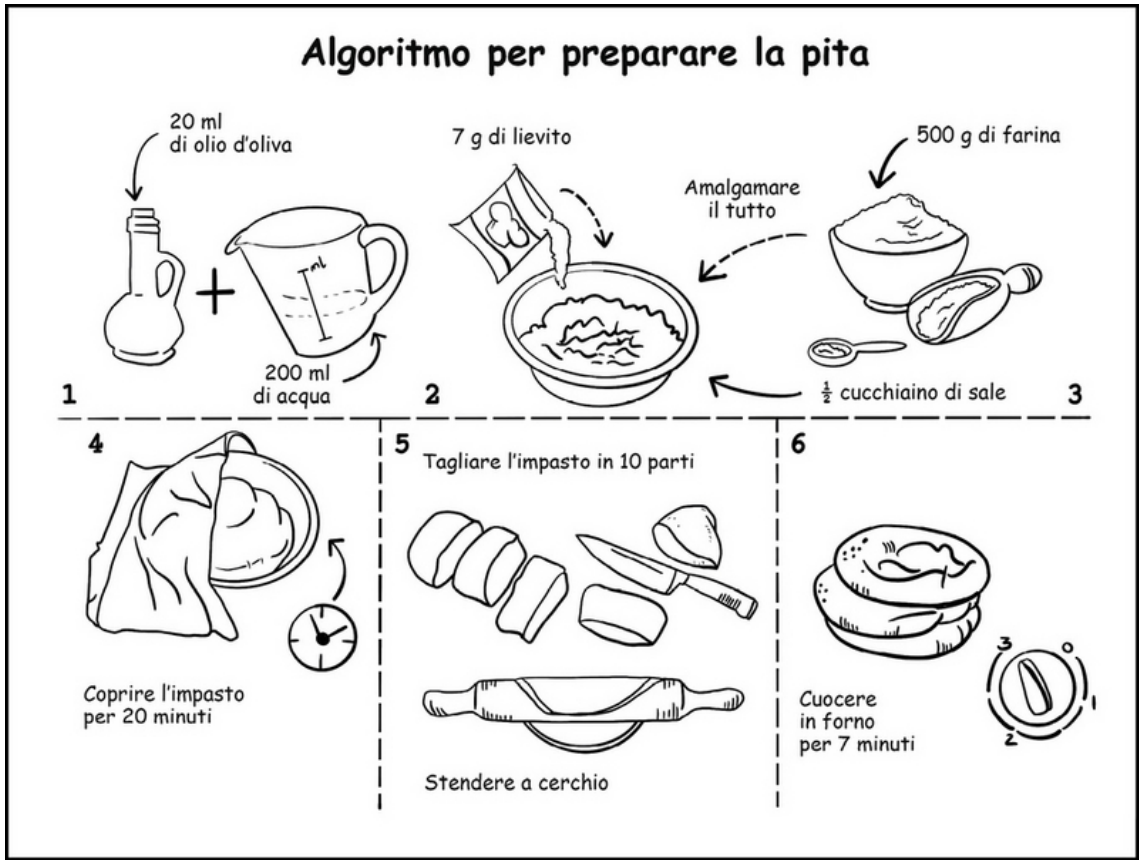


Figura 1.3 Un algoritmo è come una ricetta.

Gli algoritmi sono utilizzati per produrre vari tipi di soluzioni. Per esempio, possiamo attivare una chat video dal vivo con l'altro capo del mondo tramite algoritmi di compressione e possiamo navigare nelle strade di un Paese tramite applicazioni che utilizzano algoritmi di calcolo dei percorsi in tempo reale. Anche un semplice programma "Hello World" prevede molti algoritmi per tradurre le istruzioni in linguaggio di programmazione di alto livello in codice macchina eseguibile dall'hardware. Se fate attenzione, siamo tutti circondati da moltissimi algoritmi, ovunque.

Per illustrare qualcosa di più strettamente correlato agli algoritmi che presenterò in questo libro, la Figura 1.4 mostra un algoritmo per indovinare un numero, rappresentato come un diagramma di flusso. Il

computer genera un numero casuale entro un determinato intervallo e il giocatore deve tentare di indovinare quel numero. Notate che l'algoritmo ha passaggi discreti, che eseguono un'azione o prendono una decisione prima di passare all'operazione successiva.

Data la nostra comprensione di che cosa sono la tecnologia, i dati, l'intelligenza e gli algoritmi, diciamo che gli algoritmi di intelligenza artificiale sono sequenze di istruzioni che utilizzano i dati per creare sistemi che esibiscono un comportamento intelligente e risolvono problemi difficili.

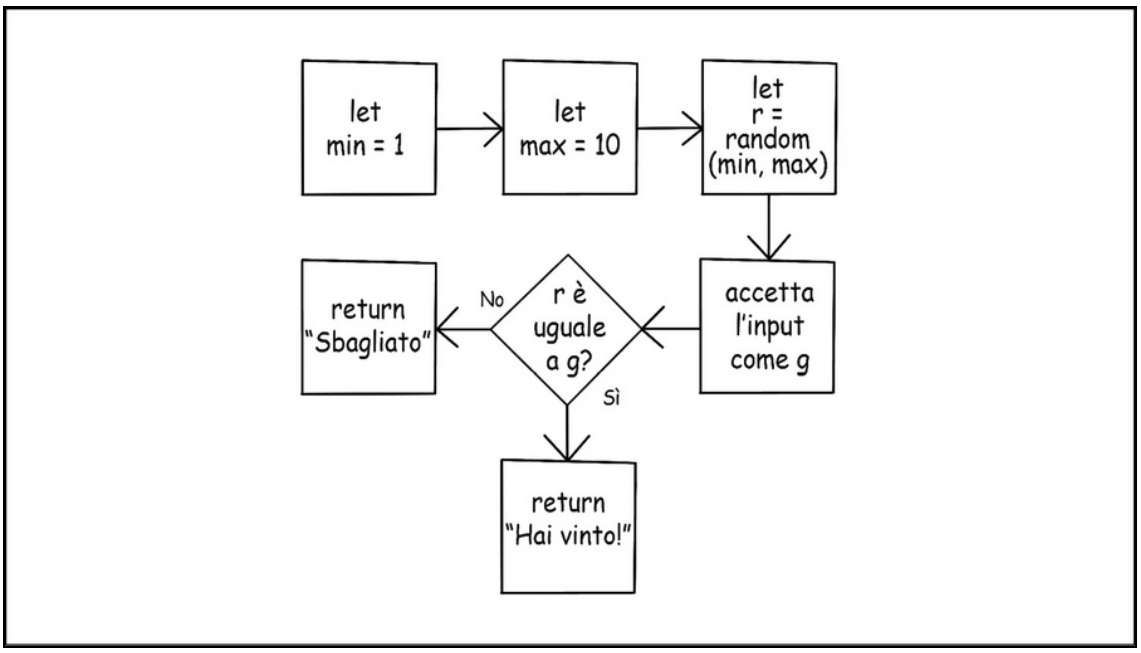


Figura 1.4 Diagramma di flusso dell'algoritmo di un gioco di indovinelli.

Breve storia dell'intelligenza artificiale

Un breve sguardo indietro ai progressi fatti nel campo dell'intelligenza artificiale è utile per capire come le vecchie tecniche possono essere combinate con nuove idee per risolvere i problemi in modi innovativi. L'intelligenza artificiale non è un'idea nuova. La storia

è piena di miti di uomini meccanici e macchine “pensanti”. Guardando indietro, scopriamo di ergerci sulle spalle di giganti. Forse noi stessi possiamo contribuire nel nostro piccolo ad aumentare le conoscenze dell’umanità.

L’esame degli sviluppi passati evidenzia l’importanza di comprendere i fondamenti dell’intelligenza artificiale; algoritmi nati decenni fa sono ancora fondamentali in molte moderne implementazioni dell’intelligenza artificiale. Questo libro inizia con alcuni algoritmi fondamentali che aiutano a intuire come avviene la risoluzione dei problemi e passa gradualmente a esplorare approcci più interessanti e moderni.

La Figura 1.5 è un elenco non esaustivo dei risultati raggiunti nel campo dell’intelligenza artificiale: è semplicemente un piccolo insieme di esempi, ma la storia è piena di molte altre scoperte!

Tipi di problemi e paradigmi di risoluzione dei problemi

Gli algoritmi di intelligenza artificiale sono potenti, ma non possono essere considerati “soluzioni magiche” in grado di risolvere qualsiasi problema. Ma quali sono questi problemi? Questo paragrafo esamina i diversi tipi di problemi che di solito incontriamo, iniziando a ragionarci. I concetti presentati possono aiutarci a identificare questi problemi nel mondo reale e a guidare la scelta degli algoritmi da utilizzare per trovare una soluzione.

Diversi termini in informatica e intelligenza artificiale sono usati per descrivere i problemi. I problemi sono classificati in base al *contesto* e *all’obiettivo*.

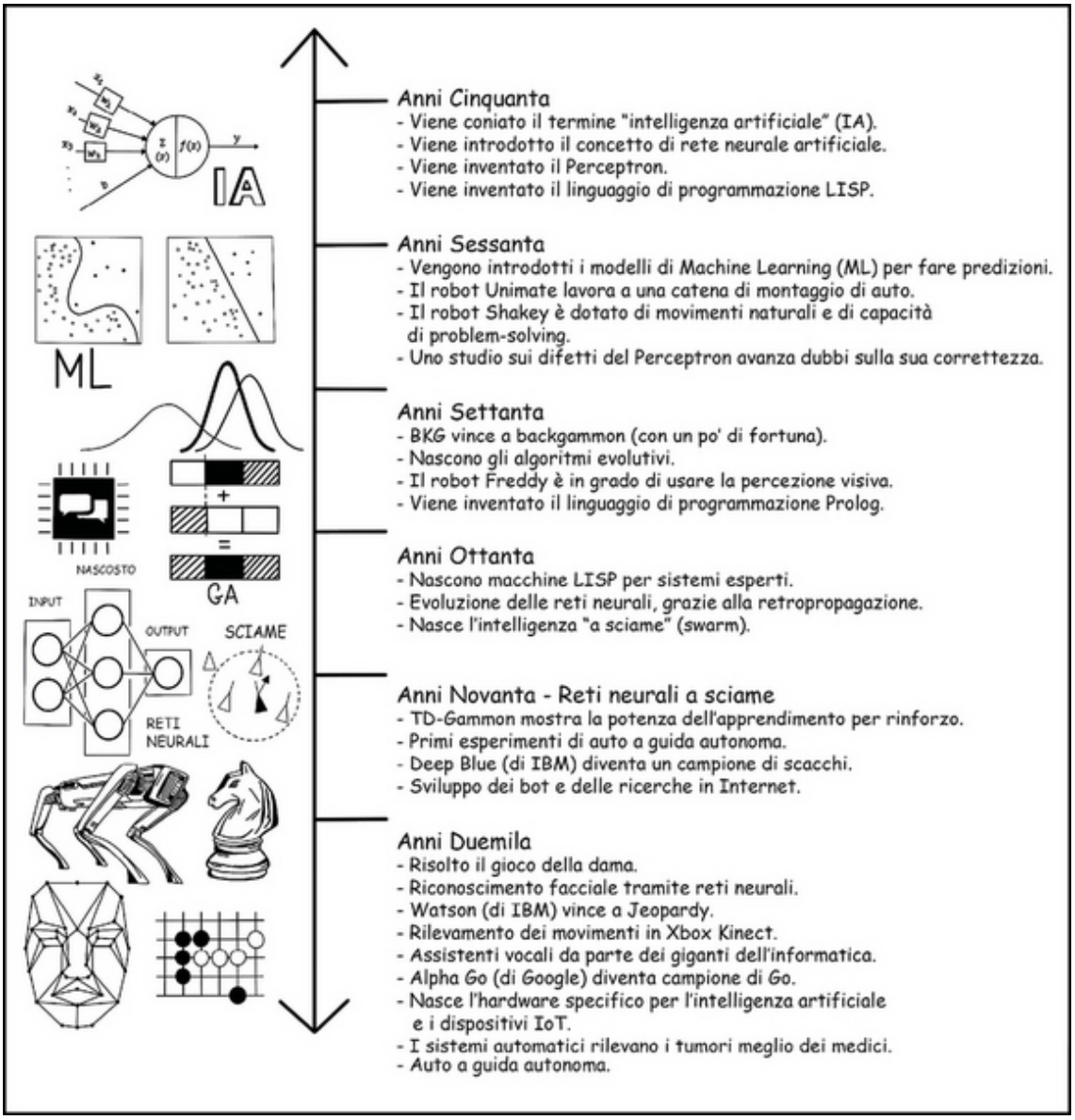


Figura 1.5 L'evoluzione dell'intelligenza artificiale.

Problemi di ricerca: trovare un percorso verso una soluzione

Un *problema di ricerca* implica una situazione che ha più soluzioni possibili, ognuna delle quali rappresenta una sequenza di passaggi (un "percorso", quindi) verso un obiettivo. Alcune soluzioni contengono

sottoinsiemi di percorsi sovrapposti; alcune sono migliori di altre; e alcune sono più economiche da raggiungere rispetto ad altre. Una soluzione "migliore" è determinata dal problema specifico in questione; con soluzione "più economica" si intende una soluzione computazionalmente più economica da trovare. Un esempio consiste nel determinare il percorso più breve fra due città su una mappa. Possono essere trovati molti percorsi, con distanze e condizioni di traffico differenti, ma alcuni percorsi sono migliori di altri. Molti algoritmi di intelligenza artificiale si basano sul concetto di ricerca di uno spazio delle soluzioni.

Problemi di ottimizzazione: trovare una buona soluzione

Un *problema di ottimizzazione* implica una situazione in cui esiste un vasto numero di soluzioni valide, e la soluzione migliore in assoluto è difficile da trovare. I problemi di ottimizzazione di solito hanno un numero enorme di possibilità, ognuna delle quali differisce in termini di efficacia nel risolvere il problema. Un esempio è quello dei bagagli nel bagagliaio di un'auto. Come massimizzare l'uso dello spazio? Sono disponibili molte combinazioni e, se il bagagliaio è riempito in modo efficace, può contenere più bagagli.

Migliore locale vs migliore globale

Poiché i problemi di ottimizzazione hanno molte soluzioni e poiché queste soluzioni esistono in punti differenti dello spazio di ricerca, entra in gioco il concetto di *local* e *global best*. Una soluzione *migliore locale* è la soluzione migliore all'interno di una determinata area nello spazio di ricerca; una soluzione *migliore globale* è la soluzione migliore nell'intero spazio di ricerca. Di solito, esistono molte migliori soluzioni locali e una sola migliore soluzione globale. Pensate, per esempio, alla ricerca del miglior ristorante. Potreste trovare il miglior ristorante in zona, ma difficilmente sarà anche il miglior ristorante del Paese o del mondo.

Problemi di previsione e classificazione: imparare dai modelli presenti nei dati

Nei *problemi di previsione* abbiamo dei dati riguardanti qualcosa e vogliamo provare a trovarvi degli schemi. Per esempio, potremmo avere dei dati su diversi veicoli e la relativa cilindrata, nonché il consumo di carburante di ciascun veicolo. Possiamo prevedere il consumo di carburante di un nuovo modello, data la sua cilindrata? Se troviamo una correlazione nei dati fra cilindrata e consumo di carburante, questa previsione è possibile.

I *problemi di classificazione* sono simili ai problemi di previsione, ma invece di cercare di trovare una previsione esatta, come il consumo di carburante, vogliamo trovare la categoria cui appartiene qualcosa, in base alle sue caratteristiche. Date le dimensioni di un veicolo, la sua cilindrata e il numero di posti, possiamo prevedere se quel veicolo è una motocicletta, una berlina o un SUV? I problemi di classificazione richiedono l'individuazione di modelli nei dati, per raggruppare gli esempi in più categorie. L'interpolazione è un concetto importante quando si trovano dei modelli nei dati, perché ci consente di stimare nuovi punti di dati in base ai dati noti.

Problemi di clustering: identificare i modelli nei dati

I *problemi di clustering* includono scenari nei quali le tendenze e le relazioni vengono tratte dai dati. Vari aspetti dei dati vengono utilizzati per raggruppare gli esempi in modi differenti. Dati i dati sui costi e sulla posizione dei ristoranti, per esempio, potremmo scoprire che i giovani tendono a frequentare luoghi nei quali il cibo è più economico.

Il clustering ha lo scopo di trovare relazioni nei dati anche quando non viene posta una domanda ben precisa. Questo approccio è utile

anche per ottenere una migliore conoscenza dei dati, per capire che cosa permettono di fare.

Modelli deterministici: lo stesso risultato ogni volta che viene calcolato

I *modelli deterministici*, dato un input specifico, restituiscono sempre un output coerente. Se è mezzogiorno in una data città, per esempio, possiamo sempre aspettarci che ci sia luce diurna; e se è mezzanotte, possiamo sempre aspettarci che sia buio. Ovviamente questo semplice esempio non tiene conto delle insolite durate del giorno nei pressi dei poli.

Modelli stocastici/probabilistici: risultati potenzialmente differenti ogni volta

I *modelli probabilistici*, dato un input specifico, restituiscono un risultato tratto da un insieme di risultati possibili. I modelli probabilistici di solito hanno un elemento di casualità, che contribuisce a guidare il possibile insieme di risultati. Se è mezzogiorno, per esempio, possiamo aspettarci che il tempo sia soleggiato, nuvoloso o piovoso; non esiste una risposta univoca a questo problema.

I concetti dell'intelligenza artificiale

L'intelligenza artificiale è un argomento "caldo", di questi tempi, così come il machine learning e il deep learning. Cercare di dare un senso a questi concetti diversi ma simili può essere un'esperienza scoraggiante. Inoltre, nel dominio dell'intelligenza artificiale, esistono distinzioni fra i diversi livelli di intelligenza.

In questo paragrafo, cercherò di demistificare alcuni di questi concetti. Il paragrafo è anche una roadmap degli argomenti trattati in

questo libro.

Entriamo nei diversi livelli di intelligenza artificiale, introdotti nella Figura 1.6.

Intelligenza ristretta: soluzioni per scopi specifici

I sistemi di *intelligenza ristretta* risolvono problemi in un contesto o dominio ben determinato. Questi sistemi di solito non sanno risolvere un problema in un contesto e applicare la stessa conoscenza a un altro. Un sistema sviluppato per comprendere le interazioni con i clienti e il loro comportamento di spesa, per esempio, non sarebbe in grado di identificare i gatti in un'immagine. Di solito, affinché qualcosa sia efficace nel risolvere un problema, deve essere abbastanza specializzato nel dominio del problema, il che rende difficile adattarlo ad altri problemi.

Diversi sistemi di intelligenza ristretta possono essere combinati in modi opportuni per creare qualcosa di più grande, che sembra esibire un'intelligenza più generale. Un esempio è un assistente vocale. Questo sistema è in grado di comprendere il linguaggio naturale, un problema ristretto, ma attraverso l'integrazione con altri sistemi di intelligenza ristretta, come le ricerche sul Web e i consigli musicali, può esibire qualità di intelligenza generale.

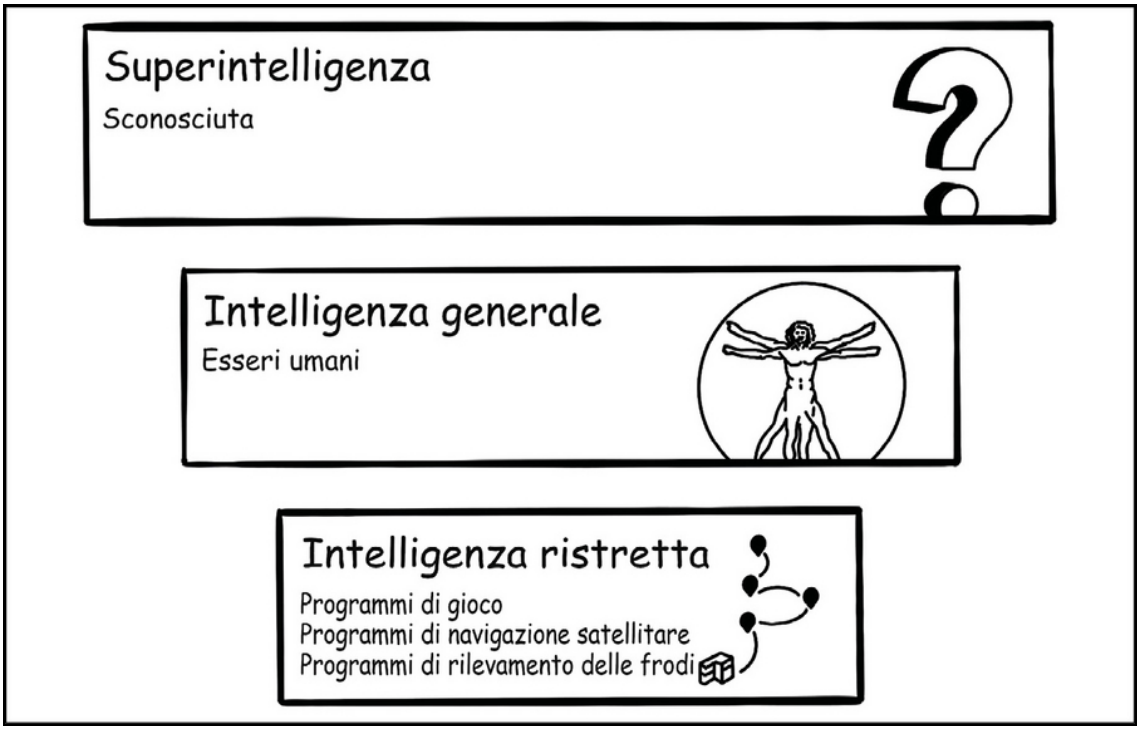


Figura 1.6 Livelli di intelligenza artificiale.

Intelligenza generale: soluzioni simili a quelle umane

L'*intelligenza generale* è l'intelligenza umana. Come esseri umani, siamo in grado di apprendere dalle esperienze e dalle interazioni con il mondo e applicare tale comprensione da un problema a un altro. Se avete provato dolore quando avete toccato qualcosa di bollente, per esempio, potete estrapolare l'esperienza e sapere che altre cose bollenti potrebbero farvi del male. L'intelligenza generale degli esseri umani, tuttavia, è molto più di un semplice ragionamento del tipo "Le cose calde possono farmi male". L'intelligenza generale comprende la memoria, il ragionamento spaziale attraverso input visivi, l'uso della conoscenza e altro ancora. Ottenere un'intelligenza generale in una macchina sembra essere un'impresa improbabile, a breve termine, ma i

progressi nel calcolo quantistico, nell'elaborazione dei dati e negli algoritmi di intelligenza artificiale potrebbero renderlo una realtà in futuro.

Superintelligenza: il grande sconosciuto

Alcune idee di *superintelligenza* compaiono nei film di fantascienza ambientati in mondi post-apocalittici, nei quali tutte le macchine sono connesse e in grado di ragionare su cose al di là della nostra comprensione e dominare gli esseri umani. Ci sono molte questioni filosofiche sul fatto che gli esseri umani siano in grado di creare qualcosa di più intelligente di loro: se potessimo farlo, sapremmo farlo. La superintelligenza è la grande sconosciuta e per molto tempo ogni definizione sarà una pura speculazione.

Intelligenza artificiale vecchia e nuova

A volte vengono utilizzati i concetti di vecchia e nuova intelligenza artificiale. La *vecchia intelligenza artificiale* spesso individua quei sistemi nei quali qualcuno ha programmato regole che fanno sì che un algoritmo esibisca un comportamento intelligente, grazie a una conoscenza approfondita del problema o per tentativi ed errori. Un esempio di vecchia intelligenza artificiale può essere la creazione manuale di un albero decisionale e delle sue regole e opzioni. La *nuova intelligenza artificiale* ha lo scopo di creare algoritmi e modelli che apprendono dai dati e creano le proprie regole, in grado di funzionare con la stessa precisione o meglio delle regole create dall'uomo. La differenza è che la nuova intelligenza artificiale può riuscire a trovare schemi importanti nei dati, che una persona potrebbe non riuscire a trovare mai o che impiegherebbe troppo tempo a trovare. Gli algoritmi di ricerca sono spesso considerati come esempi di vecchia intelligenza artificiale, ma una loro solida comprensione è propedeutica per

comprendere gli approcci più complessi. Questo libro si propone di introdurre gli algoritmi di intelligenza artificiale più popolari e sviluppare gradualmente ogni concetto. La Figura 1.7 illustra la relazione fra alcuni dei diversi concetti dell'intelligenza artificiale.

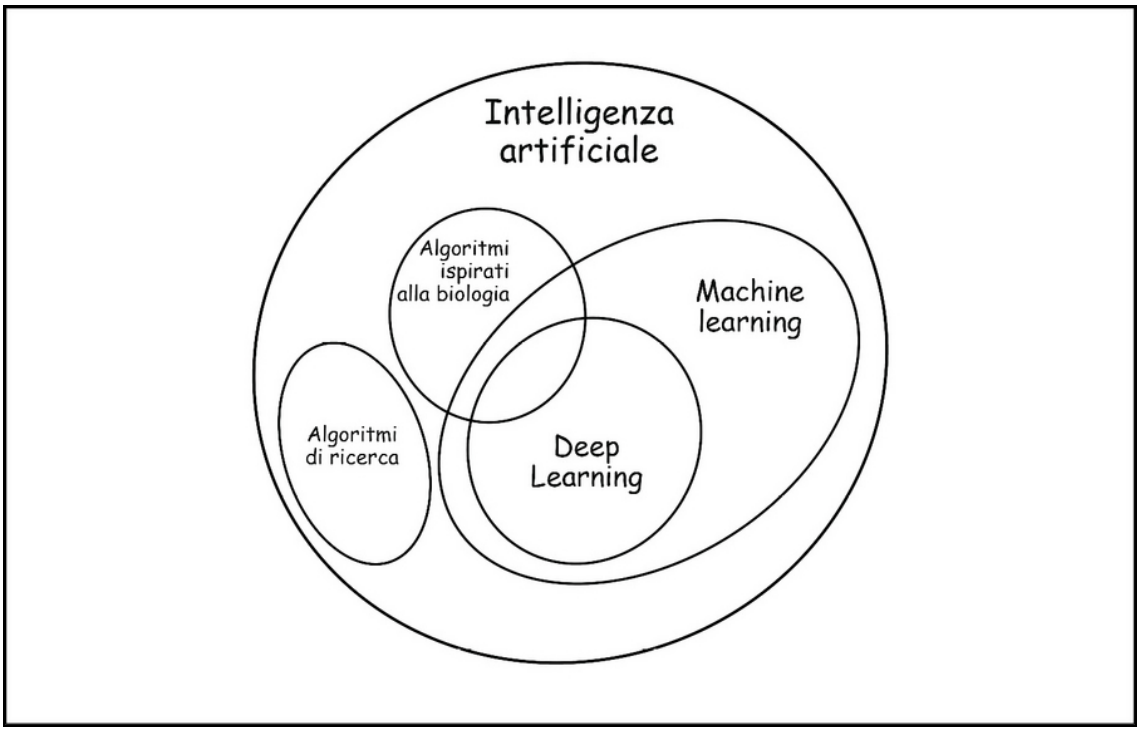


Figura 1.7 Categorizzazione dei concetti dell'intelligenza artificiale.

Algoritmi di ricerca

Gli *algoritmi di ricerca* sono utili per risolvere problemi nei quali sono necessarie diverse azioni per raggiungere un obiettivo, come trovare un percorso attraverso un labirinto o determinare la mossa migliore in un gioco. Gli algoritmi di ricerca valutano gli stati futuri e tentano di trovare il percorso ottimale per raggiungere l'obiettivo più ricco. In genere, abbiamo troppe soluzioni possibili per provare ciascuna di esse tramite la forza bruta. Anche piccoli spazi di ricerca potrebbero comportare migliaia di ore di elaborazione per trovare la

soluzione migliore. Gli algoritmi di ricerca forniscono modi intelligenti per esplorare lo spazio di ricerca. Gli algoritmi di ricerca vengono utilizzati nei motori di ricerca online, nelle applicazioni di mappatura e anche negli agenti di gioco.

Algoritmi ispirati alla biologia

Osservando il mondo che ci circonda, notiamo cose incredibili in varie creature, piante e altri organismi viventi. Fra gli esempi, cito la cooperazione delle formiche nella raccolta del cibo, lo stormo di uccelli in migrazione, le ipotesi sul funzionamento del cervello e l'evoluzione di diversi organismi per produrre una prole più resistente. Osservando e imparando da molti fenomeni, abbiamo acquisito conoscenze riguardanti il modo in cui funzionano questi sistemi organici e di come semplici regole possano portare a comportamenti intelligenti. Alcuni di questi fenomeni hanno ispirato algoritmi utili nell'intelligenza artificiale, come algoritmi gli evolutivi e algoritmi di intelligenza di sciame.

Gli *algoritmi evolutivi* si ispirano alla teoria dell'evoluzione definita da Charles Darwin. Il concetto è che una popolazione si riproduce per creare nuovi individui e che attraverso questo processo, il mix dei geni e le mutazioni producono individui che hanno prestazioni migliori rispetto ai loro antenati. L'*intelligenza di sciame* consiste nel fatto che un insieme di individui apparentemente "stupidi" esibisce un comportamento intelligente. L'ottimizzazione a colonia di formiche e a sciami di particelle sono due algoritmi popolari che esploreremo in questo libro.

Algoritmi di machine learning

Il machine learning adotta un approccio statistico per addestrare i modelli ad apprendere dai dati. Il termine "ombrello" di machine

learning prevede vari algoritmi che possono essere sfruttati per migliorare la comprensione delle relazioni insite nei dati, per prendere decisioni e fare previsioni basate su tali dati.

Esistono tre approcci principali al machine learning.

- L'*apprendimento con supervisione* prevede di addestrare i modelli con algoritmi quando i dati di addestramento contengono risultati noti per una certa domanda, come determinare il tipo di frutta se disponiamo di un dataset che include il peso, il colore, la consistenza e l'etichetta della frutta per ogni esempio.
- L'*apprendimento senza supervisione* scopre relazioni e strutture nascoste all'interno dei dati che ci guidano nel porre domande rilevanti per il dataset. Può trovare schemi nelle proprietà di frutti simili e raggrupparli di conseguenza, il che può fornirci anche le domande che vogliamo porre sui dati. Questi concetti e algoritmi fondamentali ci aiutano a creare una base per esplorare nuovi algoritmi avanzati futuri.
- L'*apprendimento per rinforzo* si ispira alla psicologia comportamentale. In breve, si premia un individuo se compie un'azione utile e lo si penalizza se compie un'azione sfavorevole. Per esplorare un esempio umano, quando un bambino ottiene buoni voti, di solito viene premiato, mentre le prestazioni scarse a volte si traducono in punizioni, rafforzando un comportamento volto a ottenere buoni risultati. L'apprendimento per rinforzo è utile per esplorare come i programmi o i robot interagiscono con ambienti dinamici. Un esempio è un robot che ha il compito di aprire le porte; viene penalizzato se non apre una porta e premiato quando lo fa. Nel tempo, dopo alcuni tentativi, il robot "impara" la sequenza delle azioni necessarie per aprire una porta.

Algoritmi di deep learning

Il *deep learning*, che deriva dal machine learning, è una famiglia più ampia di approcci e algoritmi utilizzati per ottenere un'intelligenza ristretta e tendere verso un'intelligenza generale. Il deep learning di solito implica che l'approccio stia tentando di risolvere un problema in un modo più generale, come il ragionamento spaziale, oppure viene applicato a problemi che richiedono una maggiore generalizzazione, come la visione artificiale e il riconoscimento vocale. I problemi generali sono cose che gli esseri umani sono bravi a risolvere. Per esempio, possiamo abbinare modelli visivi in quasi tutti i contesti. Anche il deep learning impiega l'apprendimento con supervisione, senza supervisione e per rinforzo. Gli approcci di deep learning di solito impiegano molti livelli di reti neurali artificiali. Vi sono diversi livelli di componenti intelligenti, ognuno dei quali risolve problemi specializzati; insieme, questi livelli risolvono problemi complessi, convergendo verso un obiettivo più grande. Identificare un oggetto in un'immagine, per esempio, è un problema generale, ma può essere suddiviso nell'individuazione del colore, nel riconoscimento della forma e nell'identificazione delle relazioni fra gli oggetti, con lo scopo di raggiungere un obiettivo.

Usi degli algoritmi di intelligenza artificiale

Gli usi delle tecniche di intelligenza artificiale sono potenzialmente infiniti. Dove ci sono dati e problemi da risolvere, ci sono potenziali applicazioni dell'intelligenza artificiale. Dato che il nostro ambiente è in continua evoluzione, cambiano anche le interazioni fra gli esseri umani e anche fra le persone e i produttori, e così l'intelligenza artificiale può essere applicata in modi innovativi per risolvere sempre

nuovi problemi. Questo paragrafo descrive l'applicazione dell'intelligenza artificiale in vari settori.

Agricoltura: crescita ottimale delle piante

L'agricoltura è uno dei settori più importanti che sostengono la vita umana. Dobbiamo essere in grado di coltivare raccolti di qualità e in modo economico per il consumo di massa. Sono molti gli agricoltori che coltivano su scala commerciale per consentirci di acquistare comodamente frutta e verdura nei negozi. Le colture crescono in modo differente in base al tipo, ai nutrienti nel suolo, all'irrigazione, ai batteri presenti nell'acqua e alle condizioni meteorologiche dell'area, fra le altre cose. L'obiettivo è quello di coltivare quanti più prodotti di alta qualità possibile entro la giusta stagione, perché le colture generalmente crescono bene solo in certe stagioni.

Gli agricoltori e le organizzazioni agricole hanno acquisito moltissimi dati sulle loro aziende e colture nel corso degli anni. Utilizzando questi dati, possiamo sfruttare i computer per individuare modelli e relazioni fra le variabili del processo di crescita delle colture, e identificare i fattori che contribuiscono maggiormente alla crescita. Inoltre, con i moderni sensori digitali, possiamo registrare in tempo reale le condizioni meteorologiche, le caratteristiche del suolo, le condizioni dell'acqua e la crescita delle colture. Questi dati, combinati con algoritmi intelligenti, possono consentire di emettere raccomandazioni e di eseguire aggiustamenti in tempo reale per garantire una crescita ottimale (Figura 1.8).

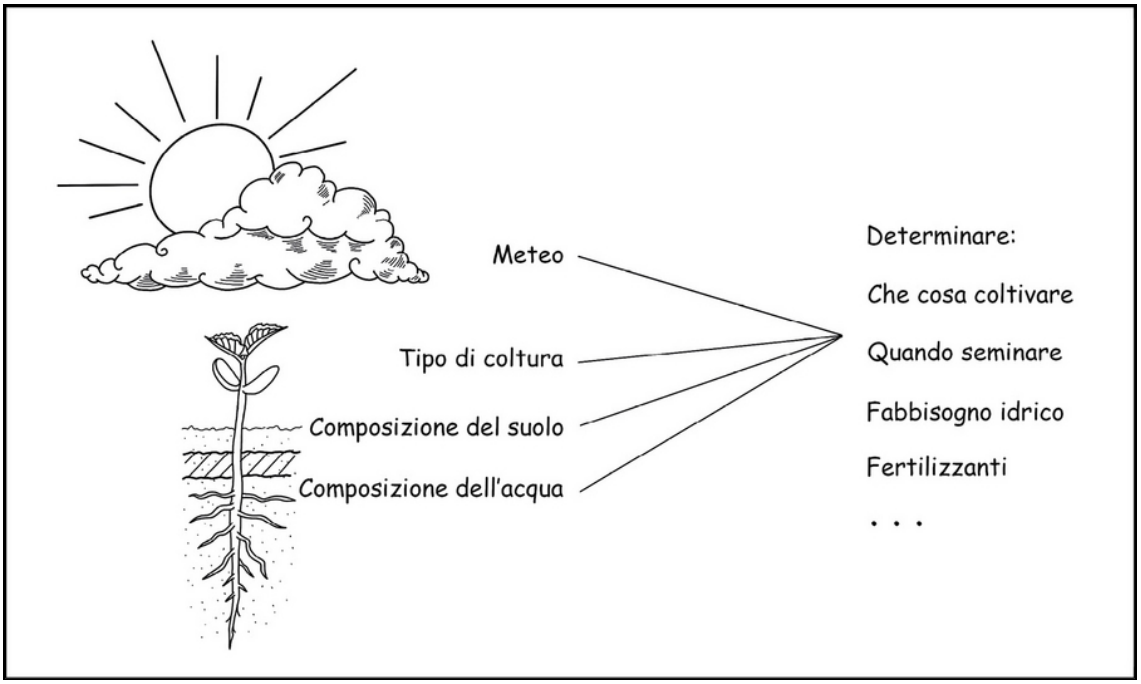


Figura 1.8 Utilizzo dei dati per ottimizzare l'agricoltura.

Banche: rilevamento delle frodi

La necessità di dotarci di un sistema bancario è diventata evidente quando abbiamo dovuto trovare una valuta comune per consentire il commercio di beni e servizi. Le banche sono cambiate nel corso degli anni per offrire diverse opzioni per conservare il denaro, investirlo ed effettuare pagamenti. Una cosa che non è cambiata nel tempo è la presenza di individui che trovano modi sempre nuovi per ingannare il sistema. Uno dei maggiori problemi, non solo del settore bancario ma anche della maggior parte degli istituti finanziari, come le compagnie assicurative, è la frode. Si ha una *frode* quando qualcuno fa qualcosa di disonesto o illegale per acquisire qualcosa che non gli appartiene. La frode di solito sfrutta scappatoie in un processo o, tramite una truffa, induce qualcuno a divulgare informazioni. Poiché il settore dei servizi finanziari è altamente connesso a Internet e ai dispositivi personali,

sempre più transazioni avvengono elettronicamente su una rete di computer e non di persona, con denaro fisico. Data la grande quantità di dati disponibili sulle transazioni, possiamo, in tempo reale, individuare specifici modelli di transazioni per il comportamento di spesa di un individuo, scoprendo quelli che potrebbero essere inusuali. Questi dati aiutano le istituzioni finanziarie a risparmiare enormi quantità di denaro e proteggono gli ignari consumatori dai furti.

Sicurezza informatica: rilevamento e gestione degli attacchi

Uno degli effetti collaterali interessanti del boom di Internet è la necessità di dotarsi di una protezione informatica. Inviamo e riceviamo continuamente informazioni sensibili: messaggi, dettagli della carta di credito, e-mail e altre informazioni importanti e riservate che potrebbero essere utilizzate in modo improprio se cadessero nelle mani sbagliate. Migliaia di server in tutto il mondo ricevono dati, li elaborano e li archiviano. Gli aggressori tentano di violare questi sistemi per ottenere l'accesso a dati, dispositivi e perfino intere strutture.

Utilizzando l'intelligenza artificiale, possiamo identificare e bloccare i potenziali attacchi ai server. Alcune grandi società archiviano dati sui modi in cui gli individui interagiscono con i loro servizi: codici identificativi dei dispositivi, dati di geolocalizzazione e schemi di utilizzo. Quando viene rilevato un comportamento anomalo, le misure di sicurezza bloccano l'accesso. Alcune società sono anche in grado di bloccare e reindirizzare il traffico dannoso durante un attacco DDoS (*Distributed Denial of Service*), che prevede il sovraccarico di un servizio con richieste fasulle, nel tentativo di saturarne l'operatività per impedire l'accesso da parte dei veri utenti. Queste richieste fasulle possono essere identificate e reindirizzate, per ridurre al minimo

l'impatto dell'attacco e proteggendo i dati di utilizzo degli utenti, i sistemi e la rete.

Assistenza sanitaria: diagnosi dei pazienti

L'assistenza sanitaria è stata una preoccupazione costante nel corso della storia umana. Abbiamo bisogno di accedere a una diagnosi e alle cure per i nostri disturbi in luoghi diversi in diverse finestre temporali, prima che un problema diventi più grave o addirittura fatale. Quando emettiamo la diagnosi di un paziente, possiamo contare sulla grande quantità di conoscenze sul corpo umano, problemi noti, esperienza nell'affrontare questi problemi e una miriade di scansioni del corpo. Tradizionalmente, i medici dovevano analizzare le immagini delle scansioni per rilevare la presenza di tumori, ma questo approccio ha portato al rilevamento solo dei tumori più grandi e in stadio avanzato. I progressi nel deep learning hanno migliorato il rilevamento dei tumori nelle immagini delle scansioni. Ora i medici possono rilevare un tumore prima, il che significa che un paziente può ottenere le cure richieste in tempo e avere maggiori possibilità di guarigione.

Inoltre, l'intelligenza artificiale può essere utilizzata per individuare modelli nei sintomi, nei disturbi, nei geni ereditari, nelle località geografiche e così via. Potremmo così sapere che qualcuno ha un'elevata probabilità di sviluppare un determinato disturbo e prepararci a trattarlo prima che si sviluppi. La Figura 1.9 illustra il riconoscimento delle caratteristiche di una scansione del cervello utilizzando il deep learning.

Logistica: percorsi e ottimizzazione

Il settore della logistica è un enorme mercato, dati i vari tipi di veicoli che consegnano vari tipi di merci in tanti luoghi diversi, con richieste e scadenze differenti. Immaginate quanto può essere

complessa la pianificazione della consegna di un grande sito di e-commerce. Indipendentemente dal fatto che i risultati finali siano beni di consumo, attrezzature per l'edilizia, parti di macchinari o carburanti, il sistema punta a ottimizzare i percorsi, per garantire che la domanda sia soddisfatta al minimo costo possibile.

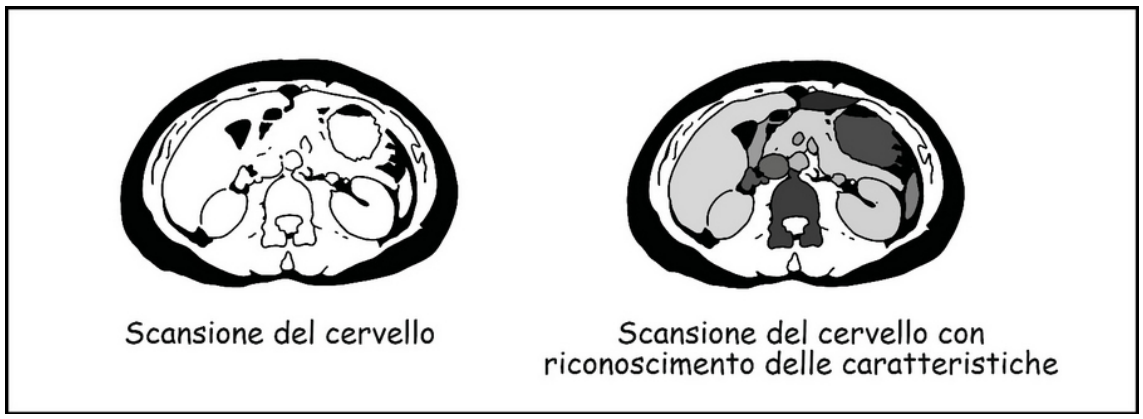


Figura 1.9 Utilizzo del machine learning per il riconoscimento delle caratteristiche nelle scansioni cerebrali.

Potreste aver sentito parlare del problema del commesso viaggiatore: un venditore deve visitare diverse località per completare il proprio lavoro e l'obiettivo è quello di trovare la distanza più breve per svolgere questo compito. I problemi logistici sono simili, ma di solito immensamente più complessi, a causa delle caratteristiche mutevoli del mondo reale. Attraverso l'intelligenza artificiale, possiamo trovare percorsi ottimali fra le località in termini di tempo e distanze. Inoltre, possiamo trovare i percorsi migliori in base a modelli di traffico, lavori stradali e perfino tipi di strade in base al veicolo utilizzato. Inoltre, possiamo calcolare il modo migliore per preparare ogni veicolo e che cosa collocare in ogni veicolo in modo tale che ogni singola consegna sia ottimizzata.

Telecomunicazioni: ottimizzazione delle reti

Il settore delle telecomunicazioni ha svolto un ruolo enorme nel collegare il mondo. Queste aziende posano e creano costose infrastrutture (cavi, torri e satelliti) per creare una rete che molti consumatori e organizzazioni possono poi utilizzare per comunicare tramite Internet o reti private. Il funzionamento di queste apparecchiature è costoso, quindi l'ottimizzazione di una rete permette di attivare più connessioni, il che consente a più persone di accedere a connessioni ad alta velocità. L'intelligenza artificiale può essere utilizzata per monitorare il comportamento su una rete e ottimizzare il routing. Inoltre, queste reti registrano le richieste e le risposte; questi dati possono essere utilizzati per ottimizzare le reti in base al carico tipico di determinati individui, aree e reti locali. I dati della rete possono anche essere utili per capire dove risiedono le persone e chi sono: informazioni preziose per la pianificazione urbana.

Giochi: creazione di agenti intelligenti

Da quando gli home e i personal computer si sono resi così ampiamente disponibili, i giochi sono stati un loro grande punto di forza. I giochi sono diventati popolari molto presto nella storia dei personal computer. Se ci ripensiamo, ricordiamo macchine da sala giochi, console per la tv e poi personal computer con capacità di gioco. Oggi gli scacchi, il backgammon e altri giochi sono dominati dalle macchine intelligenti. Se la complessità di un gioco è sufficientemente contenuta, un computer può potenzialmente esplorare tutte le possibilità e prendere una decisione basata sulla propria conoscenza più velocemente di qualsiasi essere umano. Di recente, un computer è riuscito a sconfiggere i campioni umani nel gioco di strategia Go. Go ha regole semplici per il controllo del territorio, ma ha un'enorme

complessità in termini di decisioni che devono essere prese per produrre uno scenario vincente. Questo significa che per battere i migliori giocatori umani un computer non può generare tutte le possibilità, perché lo spazio di ricerca è troppo ampio; al contrario, richiede di impiegare un algoritmo più generale, in grado di "ragionare" in modo astratto, sviluppare strategie e pianificare le mosse verso un obiettivo. Quell'algoritmo è stato inventato ed è riuscito a sconfiggere i campioni del mondo umani. È stato adattato anche ad altre applicazioni, come certi giochi Atari e i moderni giochi multiplayer. Questo sistema si chiama Alpha Go.

Diversi enti di ricerca hanno sviluppato sistemi di intelligenza artificiale in grado di giocare a giochi molto complessi meglio dei giocatori umani, anche in squadre. L'obiettivo di questo lavoro è quello di creare approcci generali in grado di adattarsi a più contesti differenti. A prima vista, questi algoritmi di intelligenza artificiale per il gioco possono sembrare frivoli, ma la conseguenza dello sviluppo di questi sistemi è che tale approccio può essere applicato efficacemente anche ad altri spazi dei problemi ben più importanti. La Figura 1.10 illustra come un algoritmo di apprendimento per rinforzo può imparare a giocare a un videogioco classico come *Mario*.

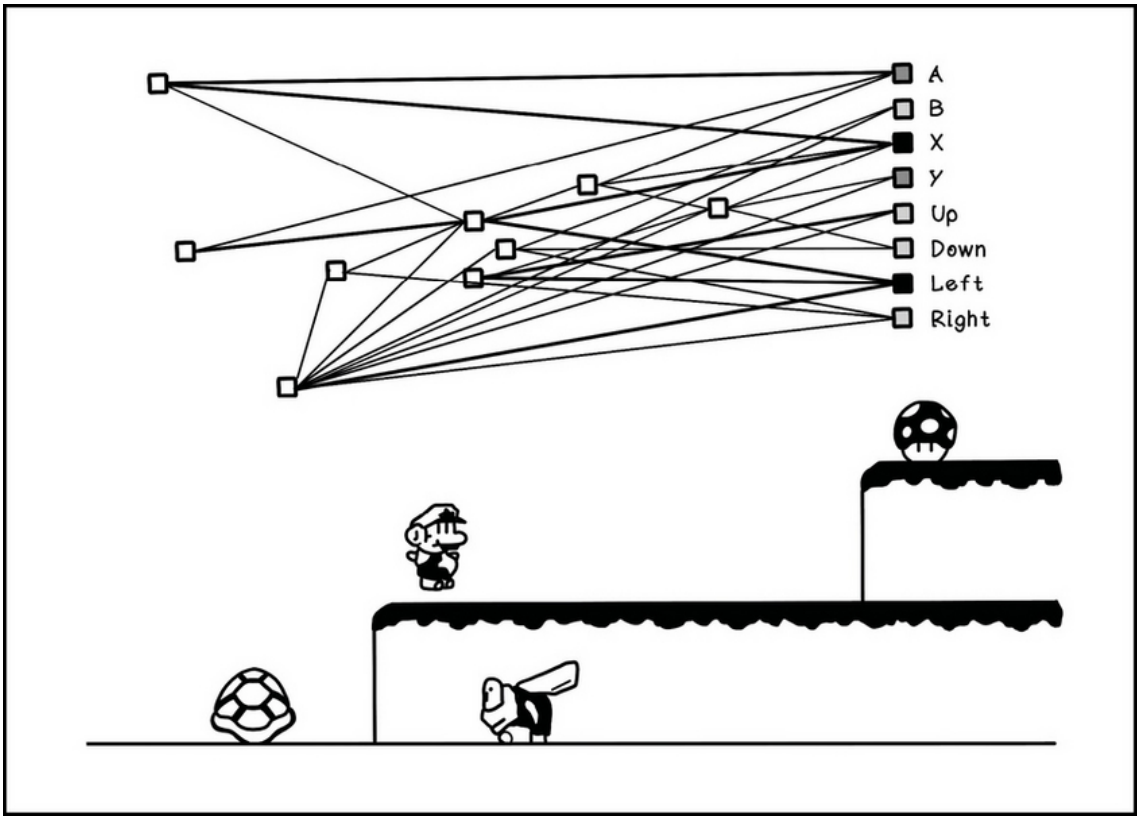


Figura 1.10 Le reti neurali possono imparare a giocare.

Arte: creazione di capolavori

Artisti unici e di enorme talento hanno creato bellissimi capolavori. Ogni artista ha il proprio modo di esprimere il mondo che lo circonda. Abbiamo anche composizioni musicali sorprendenti, che sono apprezzate da tutti. In entrambi i casi, il valore dell'arte non può essere misurato quantitativamente, ma qualitativamente: quanto l'opera viene apprezzata. I fattori coinvolti sono difficili da valutare e misurare; il gusto è guidato dall'emozione.

Molti progetti di ricerca puntano a costruire un'intelligenza artificiale in grado di generare arte. Il concetto stesso implica capacità di generalizzazione. Un algoritmo dovrebbe avere una comprensione ampia e generale dell'argomento per creare qualcosa che rientri in

determinati parametri. Un'intelligenza artificiale "Van Gogh", per esempio, dovrebbe comprendere tutto il lavoro di Van Gogh, estrarne lo stile e la "sensazione" in modo da poter poi applicare tali caratteristiche a nuove opere. Lo stesso pensiero può essere applicato all'estrazione di schemi nascosti in campi come l'assistenza sanitaria, la sicurezza informatica e la finanza.

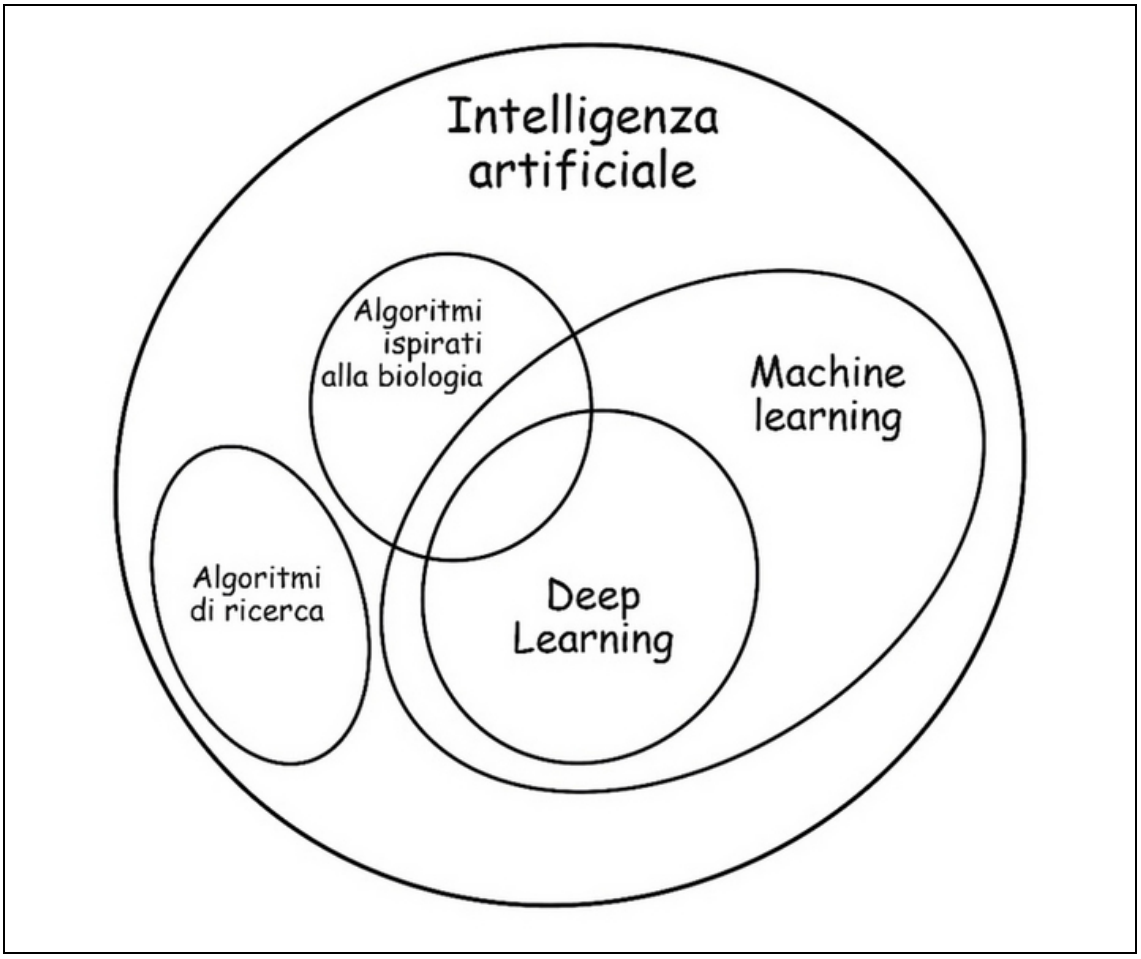
Ora che abbiamo una conoscenza almeno astratta su che cosa si intenda per intelligenza artificiale, la categorizzazione dei temi di cui si occupa, i problemi che punta a risolvere e alcuni casi d'uso, ci tufferemo in una delle forme più antiche e semplici di imitazione dell'intelligenza: gli algoritmi di ricerca, i quali costituiscono una buona base per alcuni dei concetti utilizzati da altri algoritmi di intelligenza artificiale più sofisticati, esplorati successivamente in questo libro.

Riepilogo

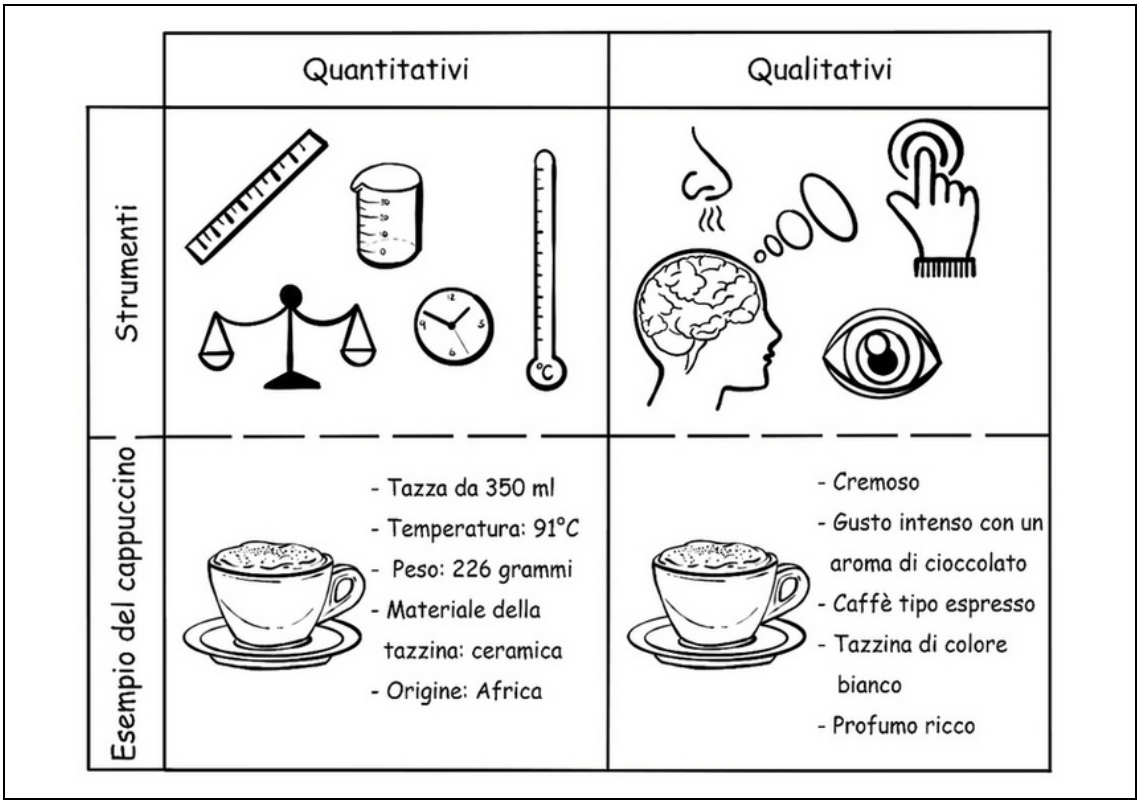
L'intelligenza artificiale non è facile da definire. Non vi è un consenso univoco.

Le implementazioni sono come agenti che esibiscono intelligenza.

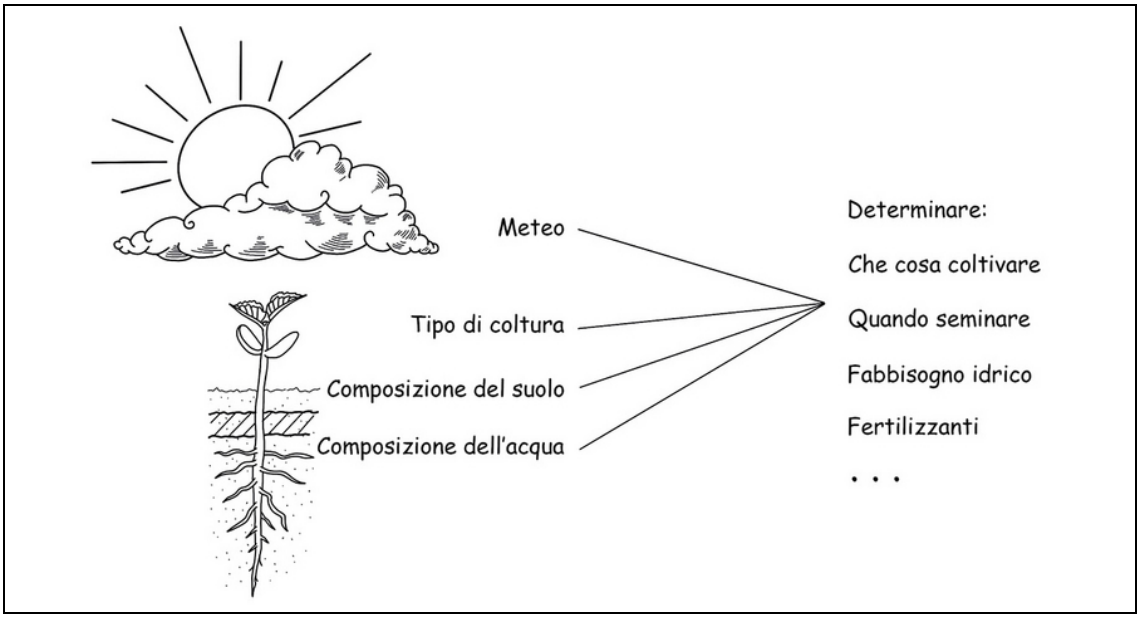
- L'intelligenza artificiale comprende molte discipline.

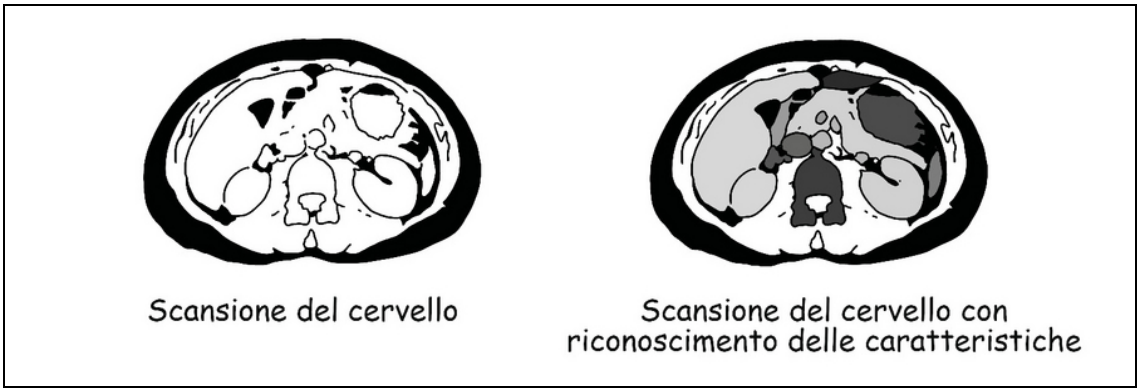


- Le implementazioni dell'intelligenza artificiale quasi sempre sono soggette a errori. Fate attenzione alle conseguenze di tali errori. La qualità e la preparazione dei dati sono importanti.



- L’intelligenza artificiale ha molte applicazioni. Applicate la fantasia.





Siate responsabili nello sviluppo di questa tecnologia.

I fondamenti della ricerca

Che cosa sono la pianificazione e la ricerca?

Quando pensiamo a ciò che ci rende intelligenti, consideriamo un attributo importante la capacità di pianificare prima di compiere azioni. Prima di intraprendere un viaggio in un altro Paese, prima di iniziare un nuovo progetto, prima di scrivere il codice di una funzione, svolgiamo una pianificazione. La *pianificazione* si attua a livelli di dettaglio differenti in contesti differenti per tendere al miglior risultato possibile dovendo svolgere dei compiti per raggiungere determinati obiettivi (Figura 2.1).

I piani raramente funzionano perfettamente nel modo in cui immaginiamo all'inizio di un'impresa. Viviamo in un mondo in cui la realtà cambia costantemente, quindi è impossibile tenere conto di tutte le variabili e le incognite che si presenteranno lungo il percorso. Indipendentemente dalla bontà del piano con cui abbiamo iniziato, deviamo quasi sempre, a causa dei cambiamenti intervenuti nello spazio del problema. Dobbiamo così predisporre un nuovo piano, deviando dal nostro punto attuale e andando avanti, poiché di norma, dopo aver fatto alcuni passi, si verificano eventi imprevisti che richiedono una nuova iterazione della pianificazione per raggiungere gli obiettivi prefissi. Di conseguenza, il piano finale che viene eseguito è solitamente diverso da quello originale.

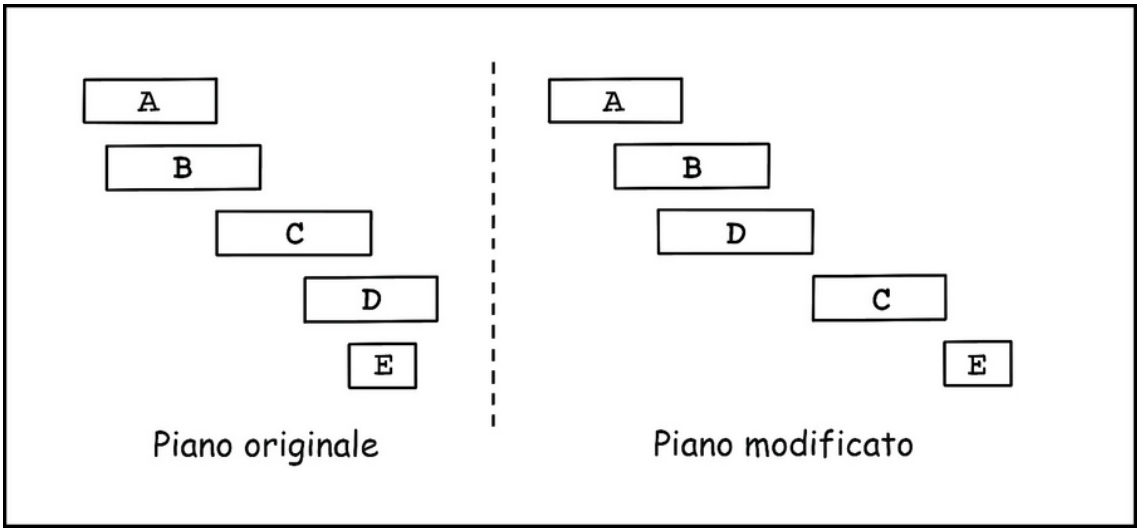


Figura 2.1 La pianificazione può modificare i progetti.

La *ricerca* è un modo per guidare la pianificazione creando i passaggi di un piano. Quando pianifichiamo un viaggio, per esempio, cerchiamo strade da percorrere, valutiamo le tappe lungo il percorso e quello che offrono, ricerchiamo alloggi e attività in linea con i nostri gusti e il nostro budget. A seconda dei risultati di queste ricerche, il nostro piano cambia.

Supponiamo di aver deciso di fare una gita al mare, a 500 chilometri, con due soste: una in una fattoria didattica e una in una trattoria. All'arrivo dormiremo in una casetta vicino alla spiaggia e parteciperemo a tre attività. Il viaggio verso la destinazione durerà circa otto ore. Prendiamo anche una scorciatoia per una strada panoramica, dopo la trattoria, ma è aperta solo fino alle 14:00.

Iniziamo il viaggio e tutto procede secondo i piani. Ci fermiamo alla fattoria didattica e vediamo alcuni animali meravigliosi. Proseguiamo e cominciamo ad avere fame; è l'ora della sosta in trattoria. Ma con nostra sorpresa, la trattoria ha recentemente cessato l'attività. Dobbiamo modificare il nostro piano e trovare un altro posto dove

mangiare, il che comporta la ricerca di un locale qui vicino che sia di nostro gradimento e quindi l'adeguamento del nostro piano.

Dopo aver girato un po', troviamo una pizzeria, ordiniamo una pizza e poi ci rimettiamo in viaggio. Avvicinandoci alla scorciatoia della strada panoramica, ci accorgiamo che sono le 14:20. La strada ora è chiusa; ancora una volta, dobbiamo modificare il nostro piano. Cerchiamo una deviazione e scopriamo che aggiungerà 120 chilometri al nostro viaggio, e dovremo trovare una sistemazione per la notte in un altro luogo, prima di arrivare alla spiaggia. Cerchiamo un posto dove dormire e tracciamo il nostro nuovo percorso. A causa del tempo perso, possiamo prendere parte solo a due attività, al mare. Il piano è cambiato notevolmente attraverso una ricerca delle diverse opzioni che soddisfacessero ogni nuova situazione, ma finiamo comunque per vivere una grande avventura nel nostro viaggio verso la spiaggia.

Questo esempio mostra come la ricerca venga utilizzata per la pianificazione e influenzi la pianificazione per produrre nuovi risultati desiderabili. A mano a mano che l'ambiente cambia, i nostri obiettivi possono cambiare leggermente, e il nostro percorso per raggiungerli deve inevitabilmente essere adattato (Figura 2.2). Gli aggiustamenti nei piani non possono quasi mai essere anticipati e devono essere apportati nel momento in cui si richiedono.

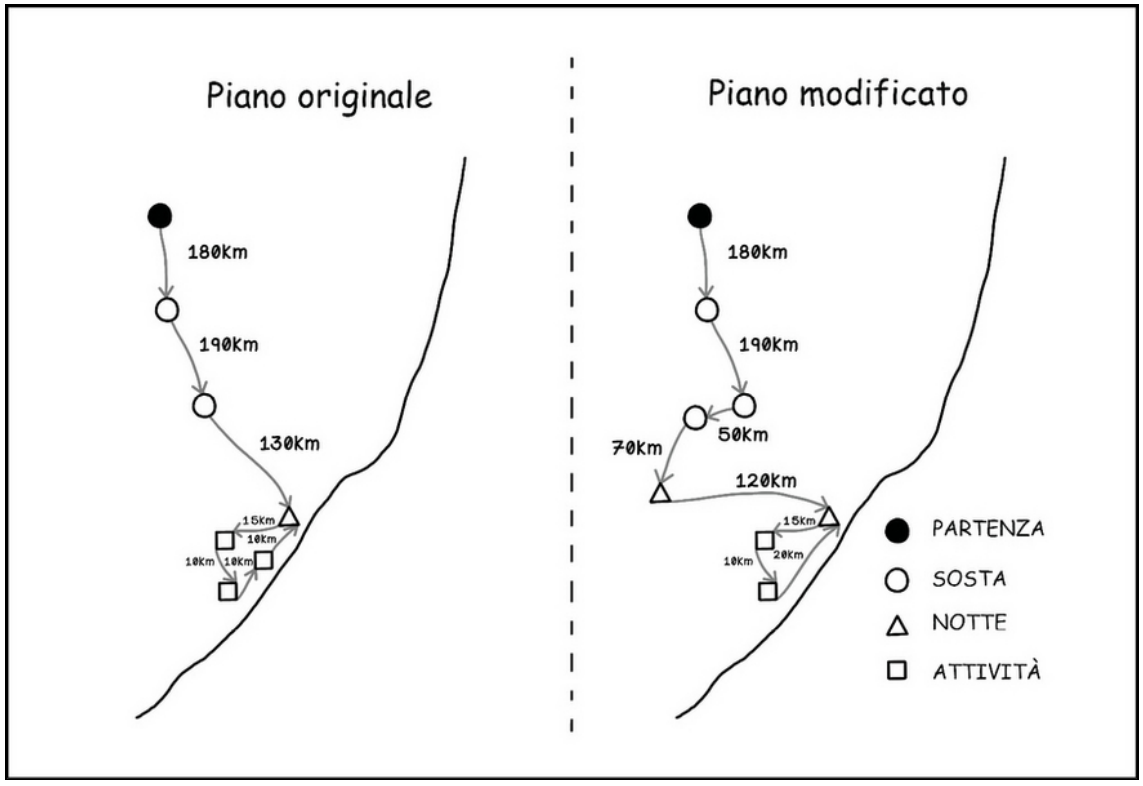


Figura 2.2 Piano originale e piano modificato per un viaggio.

La ricerca implica la valutazione di alcuni stati futuri in direzione di un obiettivo, con lo scopo di trovare un percorso ottimale di stati per raggiungere tale obiettivo. Questo capitolo è incentrato sui diversi approcci alla ricerca, a seconda dei diversi tipi di problemi. La ricerca è uno strumento antico ma potente per lo sviluppo di algoritmi intelligenti per la risoluzione dei problemi.

Costo del calcolo: lo scopo degli algoritmi intelligenti

In programmazione, le funzioni sono costituite da operazioni e, dato il modo in cui funzionano i computer, funzioni differenti impiegano tempi di elaborazione differenti. Maggiore è la quantità di calcolo

richiesta, più *costosa* è la funzione. Per descrivere la complessità di una funzione o di un algoritmo viene utilizzata la *notazione Big O*, che rappresenta il numero di operazioni richieste all'aumentare della dimensione dell'input. Ecco alcuni esempi di problemi e della relativa complessità.

- *Una singola operazione che stampa* `Hello World`: questa è una singola operazione, quindi il costo del calcolo è O(1).
- *Una funzione che esegue un'iterazione su una lista e mostra ciascun elemento*: il numero di operazioni dipende dal numero di elementi contenuti nella lista. Il costo è O(n).
- *Una funzione che confronta ogni elemento di una lista con ogni elemento di un'altra lista*: questa operazione costa $O(n^2)$.

La Figura 2.3 illustra i diversi costi degli algoritmi. Gli algoritmi che richiedono più operazioni all'aumentare della dimensione dell'input sono quelli con le prestazioni peggiori; molto migliori sono gli algoritmi che richiedono un numero il più possibile costante di operazioni all'aumentare del numero di input.

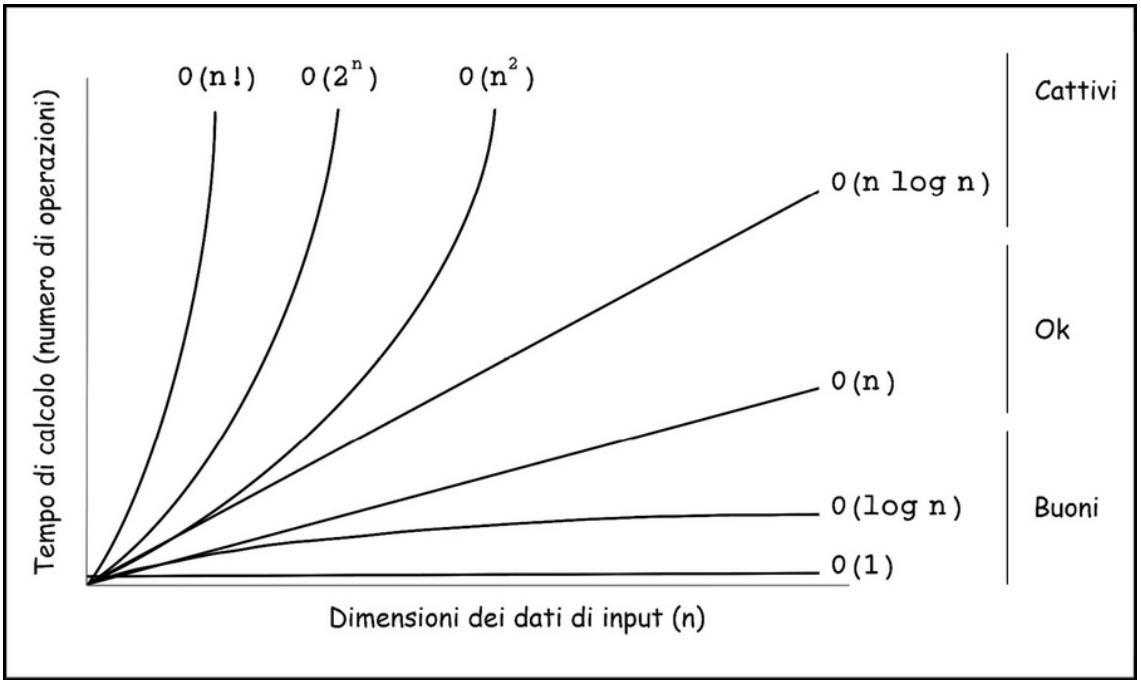


Figura 2.3 Complessità Big O.

Comprendere il fatto che algoritmi differenti prevedono costi di calcolo differenti è importante, perché esattamente questo è lo scopo di usare algoritmi intelligenti che risolvono i problemi in modo efficace e rapido. Teoricamente, possiamo risolvere quasi tutti i problemi esaminando "a forza bruta" ogni possibile opzione fino a trovare la migliore, ma così facendo il calcolo potrebbe richiedere ore o addirittura anni, il che lo renderebbe inutile per gli scenari del mondo reale.

Problemi risolvibili dagli algoritmi di ricerca

Quasi tutti i problemi che richiedono di prendere una serie di decisioni possono essere risolti con algoritmi di ricerca. A seconda del problema e delle dimensioni dello spazio di ricerca, possono essere

impiegati algoritmi differenti per tentare di risolverlo. A seconda dell'algoritmo di ricerca selezionato e della configurazione utilizzata, è possibile trovare la soluzione ottimale o la migliore soluzione disponibile. In altre parole, verrà trovata una buona soluzione, che tuttavia potrebbe non essere la soluzione migliore. Quando parliamo di "buona soluzione" o "soluzione ottimale", ci riferiamo alla capacità della soluzione di affrontare il problema in questione.

Uno scenario in cui gli algoritmi di ricerca sono particolarmente utili è la seguente: siamo in un labirinto e dobbiamo tentare di trovare il percorso più breve per raggiungere un obiettivo. Supponiamo di trovarci in un labirinto quadrato, costituito da un'area di 10×10 blocchi (Figura 2.4). Dobbiamo raggiungere un obiettivo, ma non possiamo attraversare le barriere. L'obiettivo è quello di trovare un percorso verso l'obiettivo, con il minor numero possibile di mosse spostandoci a nord, sud, est o ovest. In questo esempio, il giocatore non può muoversi in diagonale.

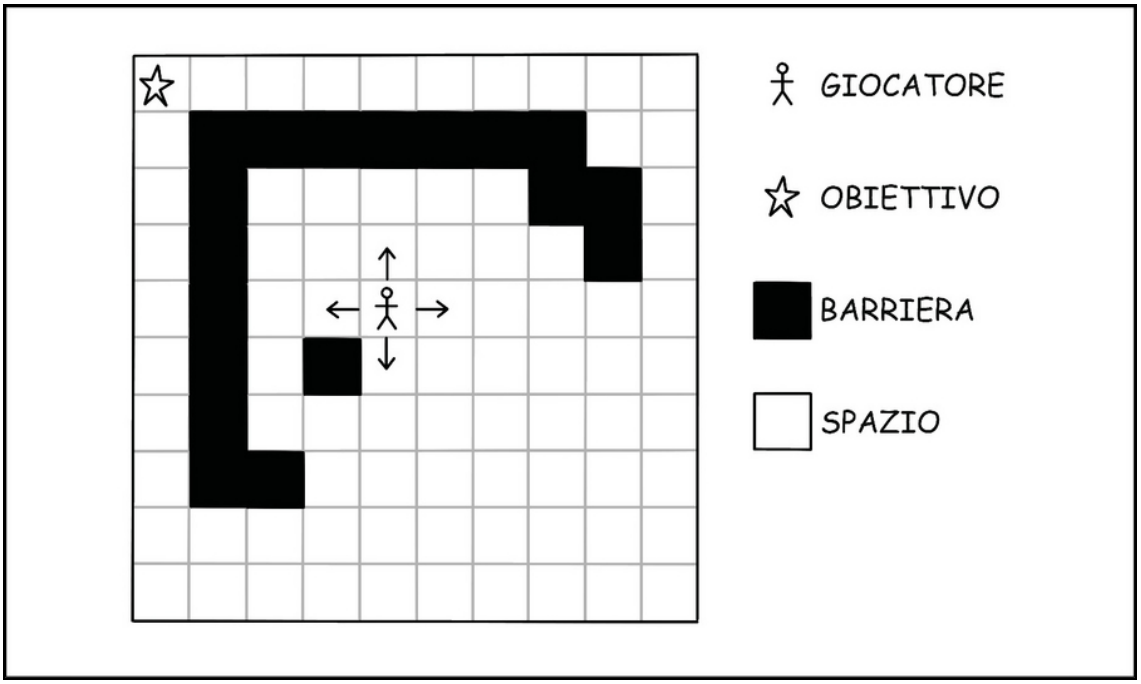


Figura 2.4 Un esempio del problema del labirinto.

Come possiamo trovare il percorso più breve verso l'obiettivo evitando le barriere? Valutando il problema come esseri umani, possiamo provare ogni possibilità e contare le mosse. Procedendo per tentativi ed errori, possiamo facilmente trovare i percorsi più brevi, dato che questo labirinto è relativamente piccolo.

Usando l'esempio del labirinto, la Figura 2.5 mostra alcuni possibili percorsi per raggiungere l'obiettivo, anche se notate che non raggiungiamo l'obiettivo nell'opzione 1.

Osservando il labirinto e contando i blocchi in direzioni differenti, possiamo trovare altre soluzioni del problema. Sono stati fatti cinque tentativi per trovare quattro soluzioni di successo su un numero imprecisato di soluzioni. Ci vorrà uno sforzo esaustivo per tentare di calcolare a mano tutte le soluzioni possibili.

- Il tentativo 1 non è una soluzione valida. Impiega 4 azioni e non trova l'obiettivo.

- Il tentativo 2 è una soluzione valida. Impiega 17 azioni per trovare l'obiettivo.
- Il tentativo 3 è una soluzione valida. Impiega 23 azioni per trovare l'obiettivo.
- Il tentativo 4 è una soluzione valida. Impiega 17 azioni per trovare l'obiettivo.
- Il tentativo 5 è la migliore soluzione valida. Impiega 15 azioni per trovare l'obiettivo. Sebbene questo tentativo sia il migliore, è stato trovato per caso.

Se il labirinto fosse molto più grande, come quello rappresentato nella Figura 2.6, ci vorrebbe un'enorme quantità di tempo per calcolare a mano il miglior percorso possibile. Gli algoritmi di ricerca possono aiutarci, in questi casi.

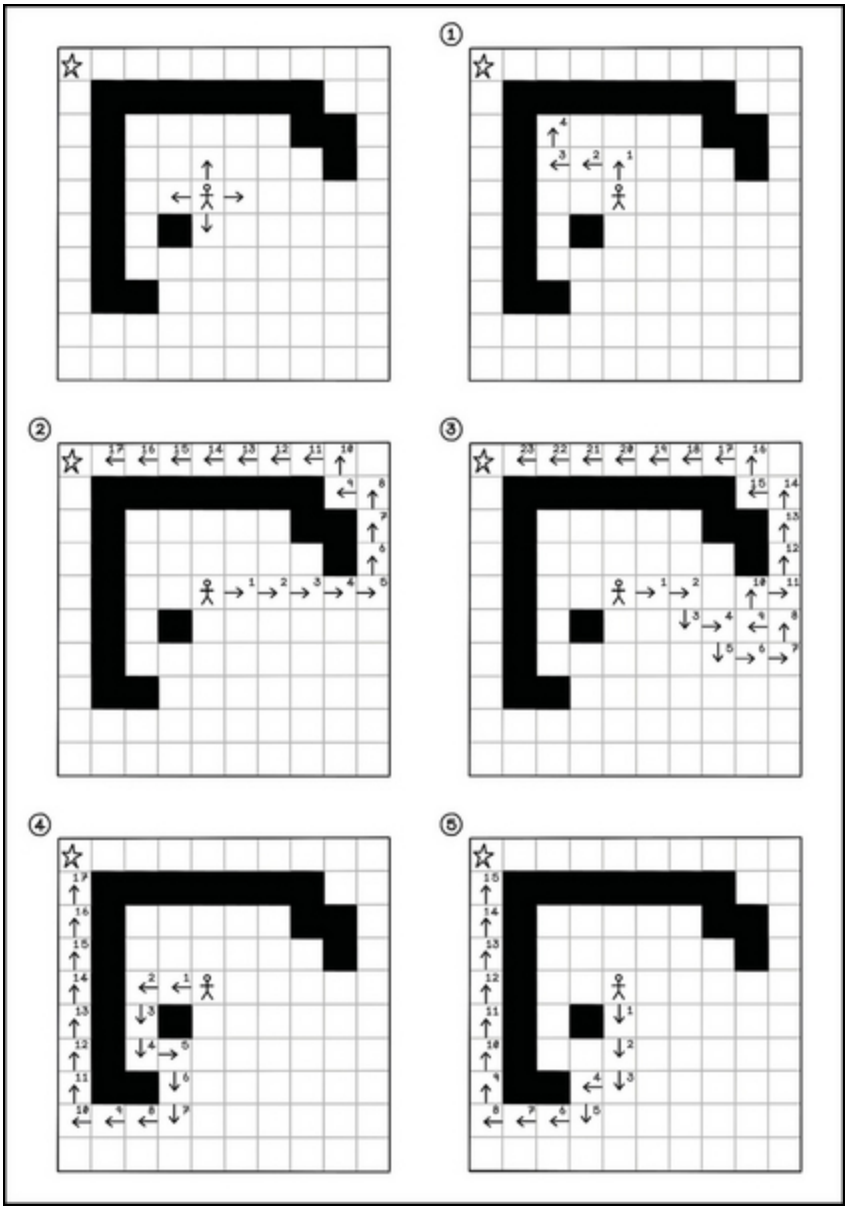


Figura 2.5 Esempi di possibili percorsi per risolvere il problema del labirinto.

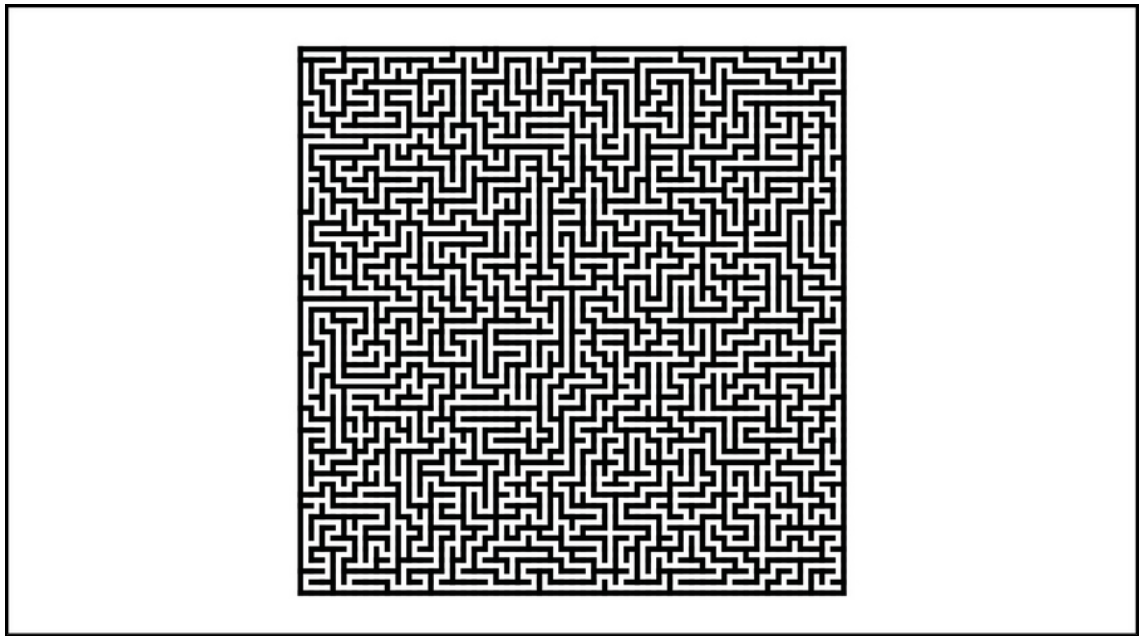


Figura 2.6 Un esempio più complesso del problema del labirinto.

La nostra capacità, come esseri umani, è quella di percepire visivamente il problema, studiarlo e trovare le soluzioni, dati i parametri. Come esseri umani, comprendiamo e interpretiamo i dati e le informazioni in modo astratto. Un computer non è ancora in grado di comprendere le informazioni generalizzate nella loro forma naturale, come facciamo noi. Lo spazio del problema deve essergli rappresentato in una forma che sia applicabile a un calcolo e che possa essere elaborata tramite algoritmi di ricerca.

Rappresentare lo stato: creare una struttura per rappresentare gli spazi dei problemi e le soluzioni

Nel rappresentare i dati e le informazioni in modi comprensibili a un computer, è necessario codificarli in modo che possano essere compresi oggettivamente. Sebbene i dati vengano codificati soggettivamente da

chi esegue tale attività, deve esserci un modo conciso e coerente per rappresentarli.

Chiariamo la differenza fra dati e informazioni. I *dati* sono fatti grezzi riguardanti qualcosa; le *informazioni* sono interpretazioni di quei fatti che forniscono informazioni sui dati in un determinato dominio. Le informazioni richiedono un contesto e l'elaborazione dei dati per fornire un significato. Per esempio, ogni distanza percorsa nell'esempio del labirinto è un dato, mentre la distanza totale percorsa è un'informazione. A seconda del punto di vista, del livello di dettaglio e del risultato desiderato, classificare qualcosa come un dato o un'informazione può essere una questione soggettiva, dipendente dal contesto e dalla persona/team (Figura 2.7).

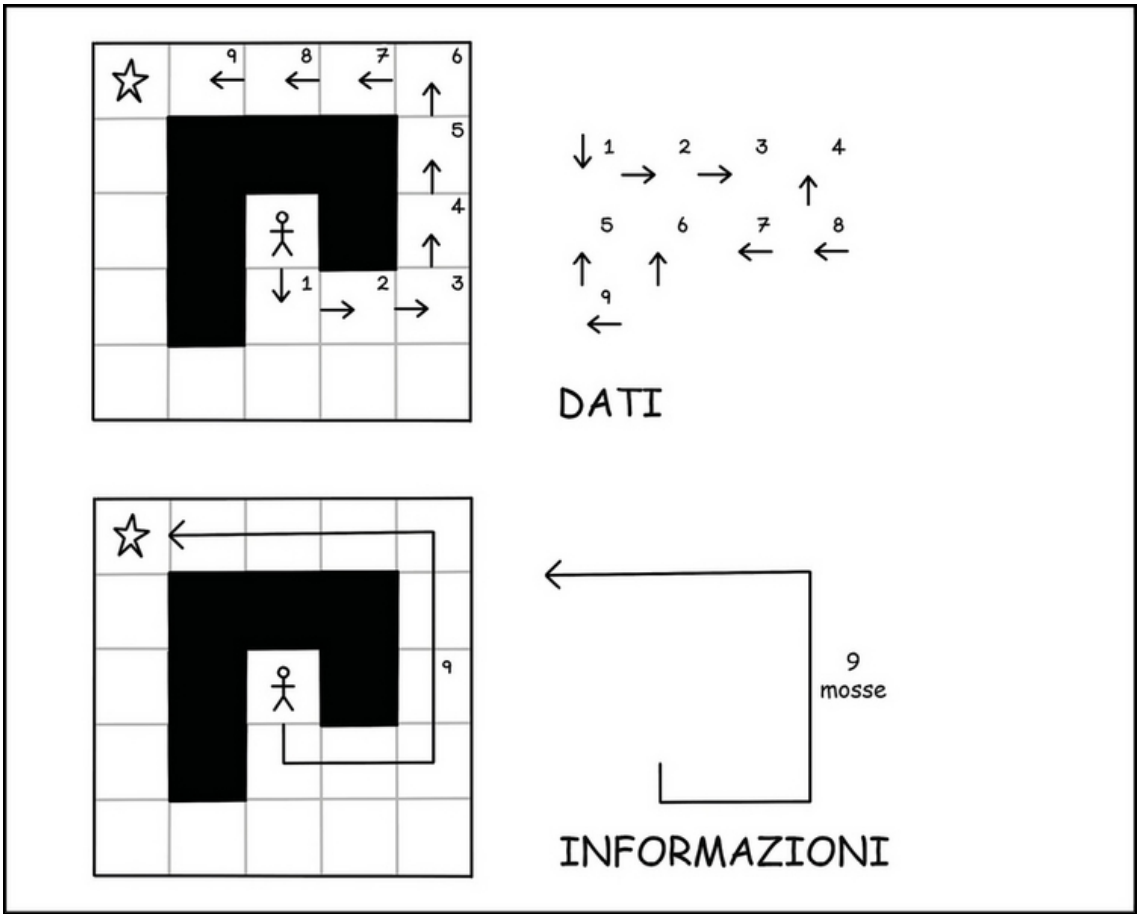


Figura 2.7 Dati e informazioni.

Le *strutture di dati* sono concetti informatici utilizzati per rappresentare i dati in un modo adatto per consentirne un'elaborazione efficiente da parte degli algoritmi. Una struttura di dati è un tipo di dati astratto, costituito dai dati e dalle operazioni eseguibili su di essi, organizzati in un modo ben preciso. La struttura dei dati che utilizziamo è influenzata dal contesto del problema e dall'obiettivo desiderato.

Un esempio di struttura di dati è l'*array*, che è semplicemente una sequenza di dati. Tipi di array differenti hanno proprietà differenti che li rendono efficienti per scopi differenti. A seconda del linguaggio di programmazione utilizzato, un array può ospitare valori di un tipo

differente o richiedere che ogni valore sia dello stesso tipo; in alcuni casi l'array potrebbe non consentire l'esistenza di valori duplicati. Questi diversi tipi di array di solito hanno nomi differenti. Le caratteristiche e i vincoli delle diverse strutture di dati consentono anche di eseguire i calcoli in modo più efficiente (Figura 2.8).

Altre strutture di dati sono utili per algoritmi di pianificazione e ricerca. Gli alberi e i grafi sono ideali per rappresentare i dati in un modo utilizzabile dagli algoritmi di ricerca.

Grafi: rappresentazione dei problemi di ricerca e delle soluzioni

Un *grafo* è una struttura di dati costituita da vari stati connessi fra loro. Ogni stato di un grafo è chiamato *nodo* (o talvolta *vertice*) e una connessione fra due stati è chiamata *arco* (o *lato* o *spigolo*). I grafi derivano dalla teoria dei grafi e vengono utilizzati per modellare le relazioni fra gli oggetti. I grafi sono strutture di dati facili da comprendere, grazie alla possibilità di rappresentarli visivamente e alla loro natura fortemente logica, che li rende ideali per l'elaborazione tramite vari algoritmi (Figura 2.9).

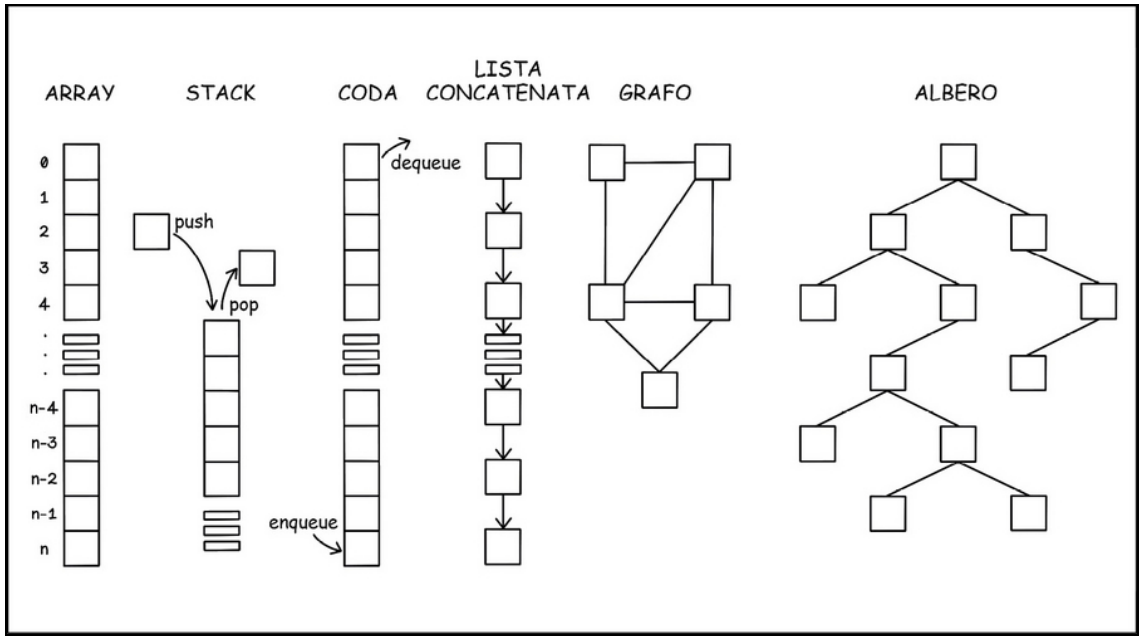


Figura 2.8 Strutture di dati utilizzate con gli algoritmi.

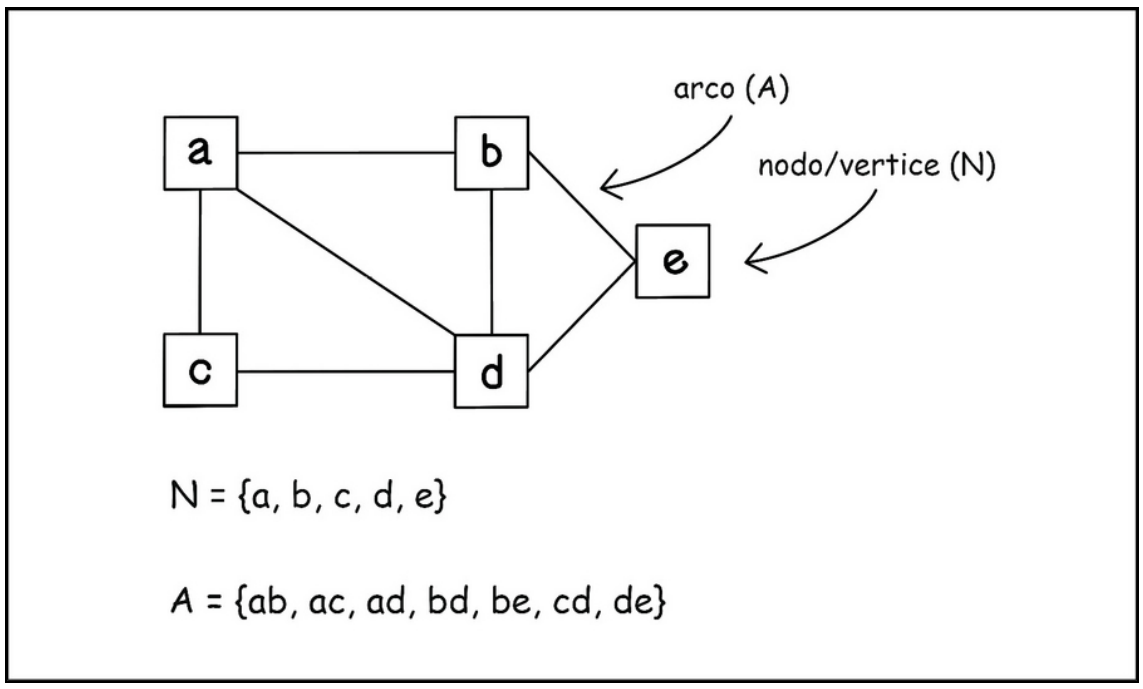


Figura 2.9 La notazione usata per rappresentare i grafi.

La Figura 2.10 mostra un grafo della gita in spiaggia discussa all'inizio di questo capitolo. Ogni sosta è un nodo del grafo; ogni arco fra i nodi rappresenta i punti attraversati; i pesi su ciascun arco indicano la distanza percorsa.

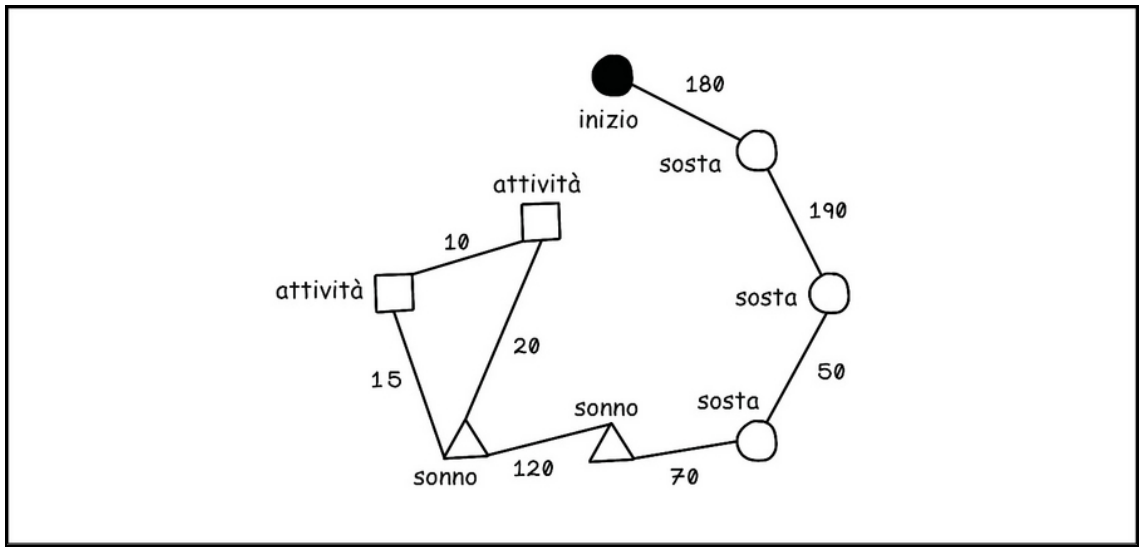


Figura 2.10 L'esempio di un percorso rappresentato come un grafo.

Rappresentare un grafo come una struttura di dati concreta

Un grafo può essere rappresentato in diversi modi per rendere possibile un'elaborazione efficiente da parte degli algoritmi. Fondamentalmente, un grafo può essere rappresentato da un array di array, che indica le relazioni fra i nodi, come mostrato nella Figura 2.11. A volte è utile avere un altro array che elenchi semplicemente tutti i nodi del grafo, in modo che i nodi non debbano essere dedotti dalle relazioni.

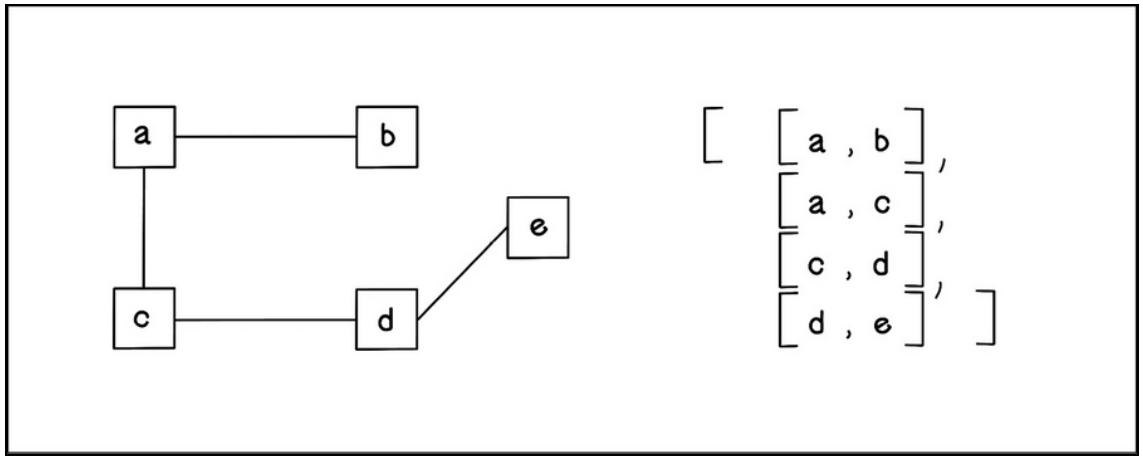
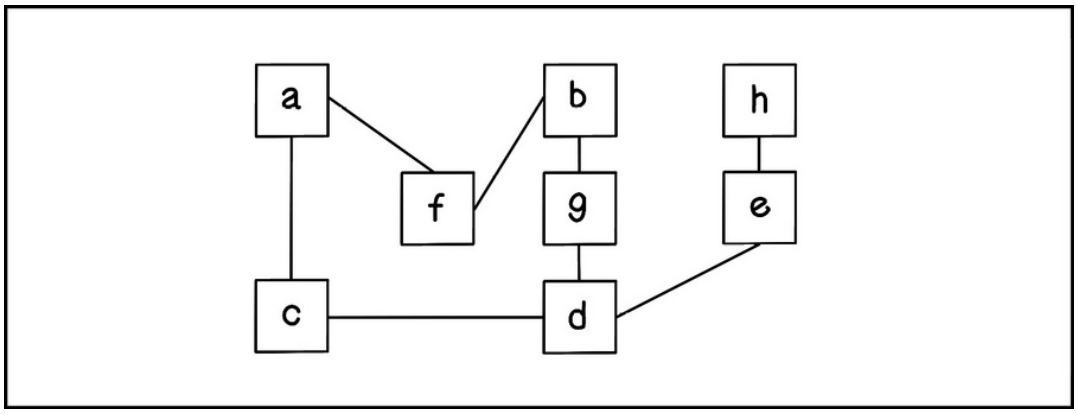


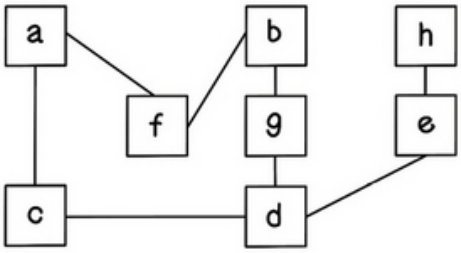
Figura 2.11 Rappresentazione di un grafo come un array di array.

Altre possibili rappresentazioni dei grafi sono la matrice di incidenza, la matrice di adiacenza e la lista di adiacenza. Osservando i nomi di queste rappresentazioni, si vede che l'adiacenza dei nodi di un grafo è importante. Un *nodo adiacente* è un nodo connesso direttamente a un altro nodo.

Esercizio: rappresentate un grafo come una matrice
Come rappresentereste il seguente grafo usando array di archi?



Soluzione




```
[ [ a, c ],
  [ a, f ],
  [ b, g ],
  [ b, f ],
  [ c, d ],
  [ d, g ],
  [ d, e ],
  [ e, h ]  ]
```

Array di archi

	a	b	c	d	e	f	g	h
a	0	0	1	0	0	1	0	0
b	0	0	0	0	0	1	1	0
c	1	0	0	1	0	0	0	0
d	0	0	1	0	1	0	1	0
e	0	0	0	1	0	0	0	1
f	1	1	0	0	0	0	0	0
g	0	1	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0

Matrice di adiacenza

Alberi: le strutture di dati utilizzate per rappresentare le soluzioni di una ricerca

Un *albero* è una struttura di dati molto utilizzata che simula una gerarchia di valori o oggetti. Una *gerarchia* è una disposizione di elementi in cui un singolo oggetto è correlato a più oggetti da esso dipendenti. Un albero è un *grafo aciclico connesso*: ogni nodo ha un arco che punta verso un altro nodo, e non esistono cicli.

In un albero, il valore o l'oggetto rappresentato in un determinato punto è chiamato *nodo*. Gli alberi, in genere, hanno un singolo nodo radice con zero o più nodi figli che a loro volta possono contenere sottoalberi. Fate un bel respiro ed entriamo nel dettaglio con un po' di terminologia. Quando a un nodo sono connessi altri nodi, il nodo radice è chiamato *genitore*. Potete applicare questo ragionamento in modo ricorsivo. Un nodo figlio può avere i propri nodi figli, i quali possono a loro volta contenere ulteriori sottoalberi. Ogni nodo figlio ha un singolo nodo genitore. Un nodo senza figli è detto nodo foglia.

Gli alberi hanno anche un'*altezza* totale. Nella discesa di livelli di nodi, si parla di *profondità*.

Quando si lavora con gli alberi, spesso si usa una terminologia legata alle relazioni fra i membri di una famiglia. Tenete a mente questa analogia, poiché vi aiuterà a memorizzare i concetti tipici della struttura di dati albero. Notate che nella Figura 2.12 l'altezza e la profondità sono indicizzate a partire da 0: il nodo radice.

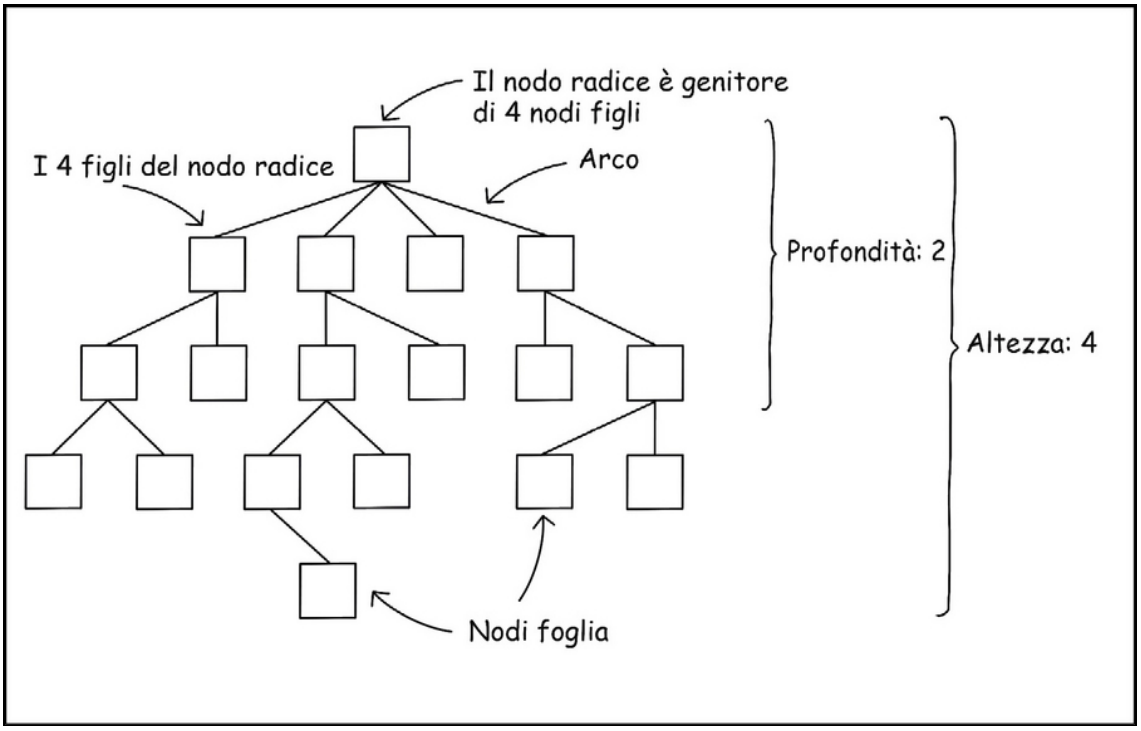


Figura 2.12 Gli attributi principali di un albero.

Il nodo più in alto di un albero è chiamato *nodo radice*. Un nodo dal quale dipendono uno o più altri nodi è chiamato *nodo padre* o *nodo genitore*, e i nodi connessi che discendono da un nodo genitore sono chiamati *nodi figli*. I nodi connessi allo stesso nodo genitore sono chiamati *nodi fratelli*. Una connessione fra due nodi è chiamata *arco*.

Un *percorso* è una sequenza di nodi e archi che collegano due nodi non direttamente connessi. Un nodo connesso a un altro nodo seguendo un percorso che parte dal nodo radice è chiamato *discendente*; un nodo connesso a un altro nodo seguendo un percorso che si dirige verso il nodo radice è chiamato *antenato*. Un nodo senza figli è chiamato *nodo foglia*. Per descrivere il numero di figli di un nodo si usa il termine *grado*; pertanto, un nodo foglia ha grado zero.

La Figura 2.13 rappresenta un percorso dal punto di partenza all'obiettivo per il problema del labirinto. Questo percorso contiene nove nodi, che rappresentano le diverse mosse compiute nel labirinto.

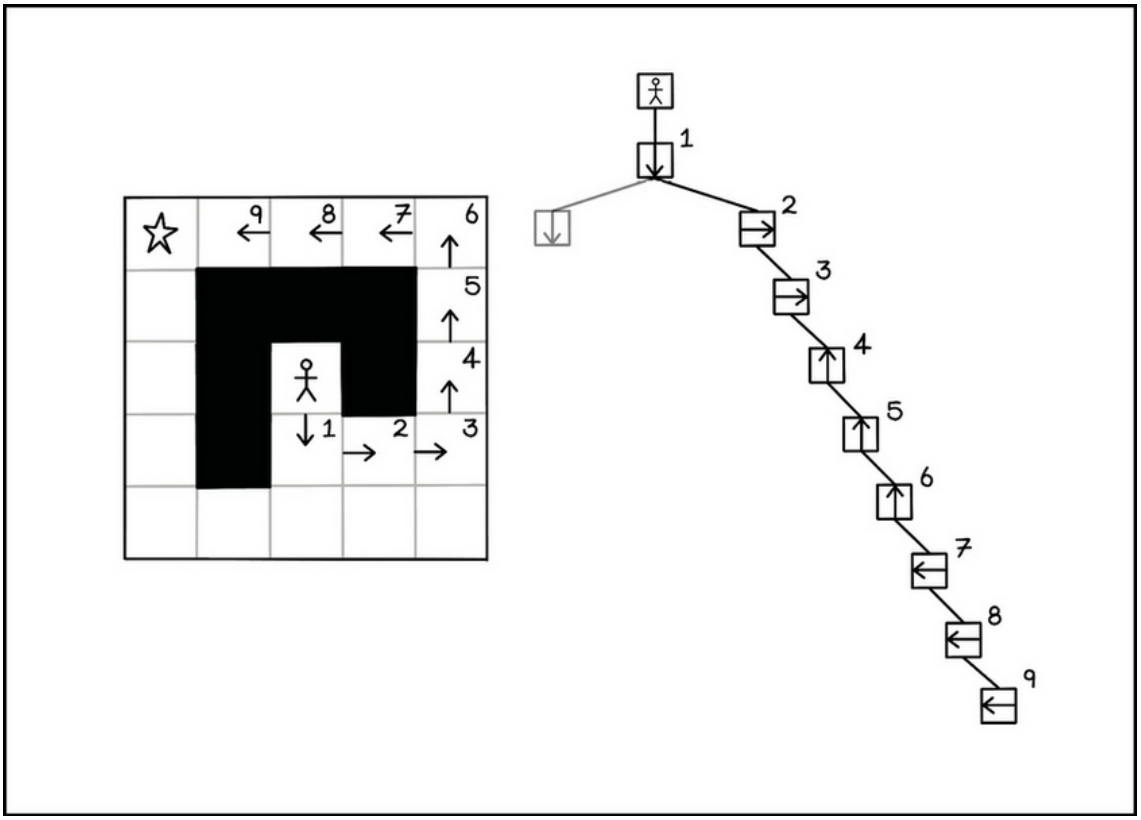


Figura 2.13 Una soluzione al problema del labirinto rappresentato come un albero.

Gli alberi sono la struttura di dati fondamentale per gli algoritmi di ricerca, che approfondiremo in seguito. Gli algoritmi di ordinamento sono utili anche per risolvere in modo più efficiente determinati problemi di calcolo. Se siete interessati a saperne di più sugli algoritmi di ordinamento, consultate *Algoritmi spiegati in modo facile* (Apogeo, 2022).

Ricerca non informata: ricerca cieca delle soluzioni

La *ricerca non informata* è detta anche *ricerca non guidata*, *ricerca cieca* o *ricerca a forza bruta*. Gli algoritmi di ricerca non informati non

dispongono di informazioni sul dominio del problema, a parte la rappresentazione del problema, che di solito è un albero.

Pensate a come si esplorano le cose da imparare. Alcuni potrebbero esaminare in ampiezza tutti gli argomenti e apprendere le basi di ciascuno di essi; altri potrebbero scegliere un determinato argomento ed esplorare tutti i suoi sottoargomenti, in profondità. Si parla quindi di ricerca in ampiezza e ricerca in profondità. Una *ricerca in profondità* (*Depth-First Search*, DFS) esplora un determinato percorso dall'inizio finché non trova un obiettivo alla massima profondità dell'albero. Una *ricerca in ampiezza* (*Breadth-First Search*, BFS) esplora prima tutte le opzioni a un livello di profondità, prima di passare a un livello più in profondità nell'albero.

Considerate lo scenario del labirinto (Figura 2.14). Nel tentativo di trovare un percorso ottimale verso l'obiettivo, applicate il seguente semplice vincolo per evitare di rimanere bloccati in un ciclo infinito, ovvero per impedire la formazione di cicli nel nostro albero: *il giocatore non può portarsi in un blocco che ha precedentemente occupato*. Poiché gli algoritmi non informati tentano ogni possibile opzione di ogni nodo, la creazione di un ciclo provocherebbe il fallimento catastrofico dell'algoritmo.

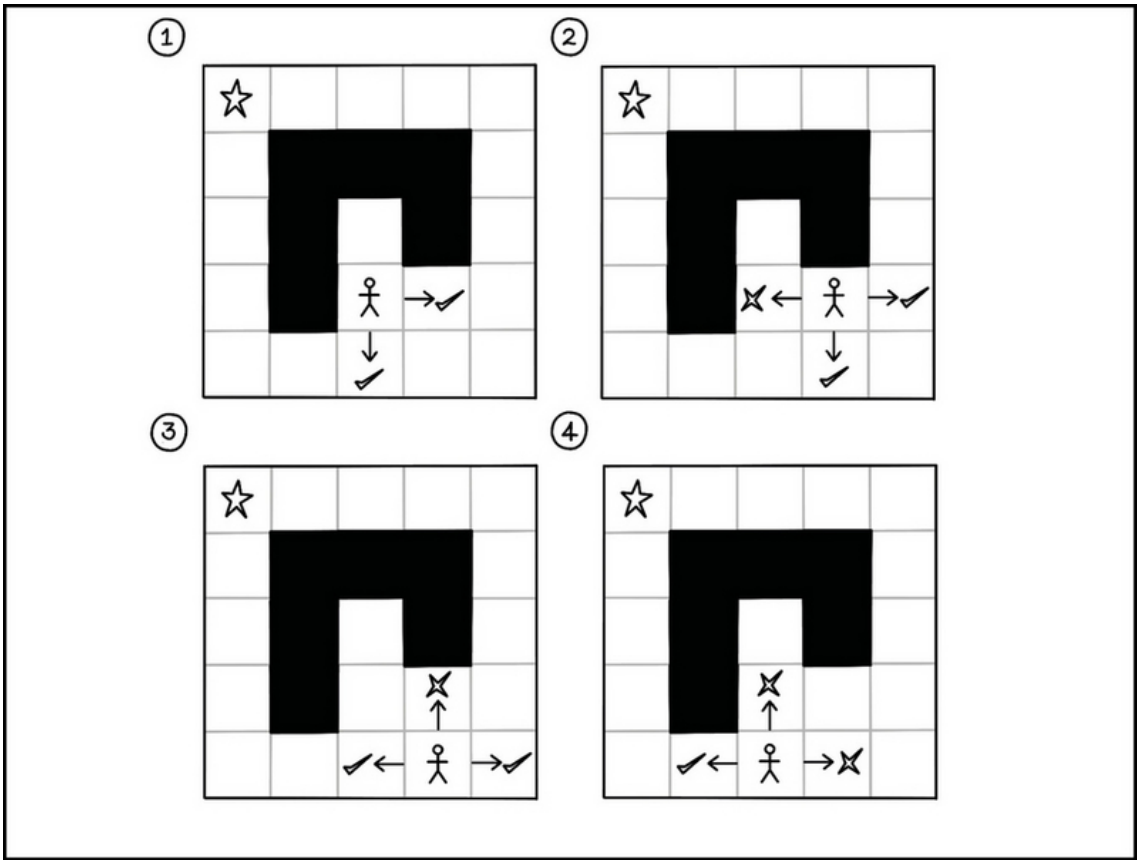


Figura 2.14 Il vincolo per il problema del labirinto.

Questo vincolo impedisce la formazione di cicli nel percorso verso l'obiettivo. Ma questo vincolo introdurrà problemi se, in un labirinto diverso con vincoli o regole differenti, sarà necessario portarsi più di una volta su un blocco per trovare la soluzione ottimale.

Nella Figura 2.15 sono rappresentati tutti i possibili percorsi nell'albero, per evidenziare le diverse opzioni disponibili. Questo albero contiene sette percorsi che portano all'obiettivo e un percorso che produce una soluzione non valida, dato il vincolo di non portarsi su blocchi precedentemente occupati. È importante capire che in questo piccolo labirinto, non è difficile rappresentare tutte le possibilità. Ma lo scopo degli algoritmi di ricerca è quello di cercare o generare questi alberi in modo iterativo, nei casi in cui generare l'intero

albero delle possibilità si inefficiente a causa del puro costo computazionale.

È anche importante notare che il termine *visitare* viene usato per indicare cose diverse. Il giocatore *visita* i blocchi del labirinto. L'algoritmo *visita* i nodi dell'albero. L'ordine delle scelte influenzerà l'ordine dei nodi visitati nell'albero. Nell'esempio del labirinto, l'ordine di priorità del movimento è nord, sud, est e poi ovest.

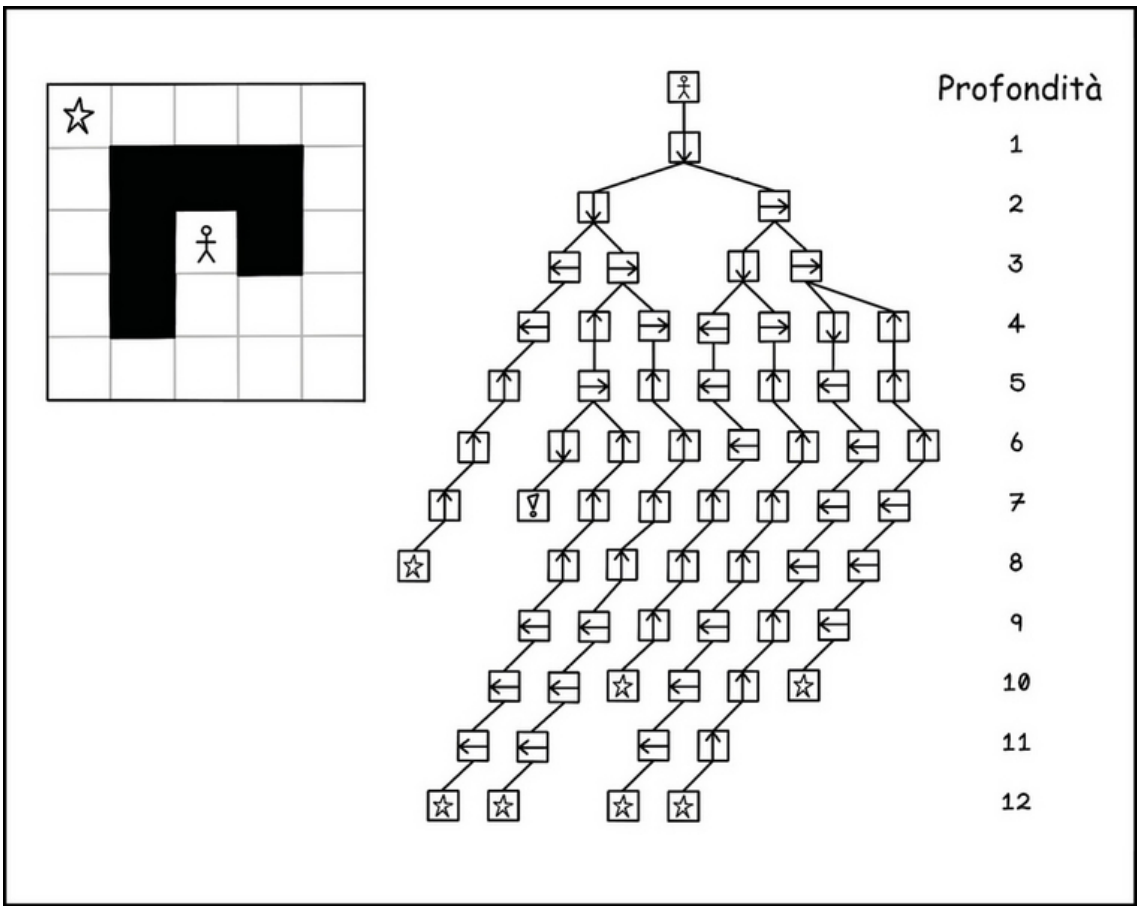


Figura 2.15 Tutte le possibili opzioni di movimento rappresentate come un albero.

Ora che conoscete le idee di base degli alberi e l'esempio del labirinto, esploriamo come gli algoritmi di ricerca possono generare alberi in grado di trovare i percorsi verso l'obiettivo.

Ricerca in ampiezza: prima in ampiezza e poi in profondità

La *ricerca in ampiezza* è un algoritmo utilizzato per attraversare o generare un albero. Questo algoritmo parte da un nodo, la radice o *root*, ed esplora ogni singolo nodo a quel livello di profondità, prima di passare ai nodi del livello successivo, e questo finché non trova un nodo foglia *obiettivo*.

L'algoritmo di ricerca in ampiezza viene implementato al meglio utilizzando una coda first-in, first-out in cui vengono elaborati i nodi a una certa profondità e i loro figli vengono messi in coda per essere elaborati successivamente. Questo ordine di elaborazione è esattamente ciò di cui abbiamo bisogno quando implementiamo questo algoritmo.

La Figura 2.16 è un diagramma di flusso che descrive la sequenza di passaggi coinvolti nell'algoritmo di ricerca in ampiezza.

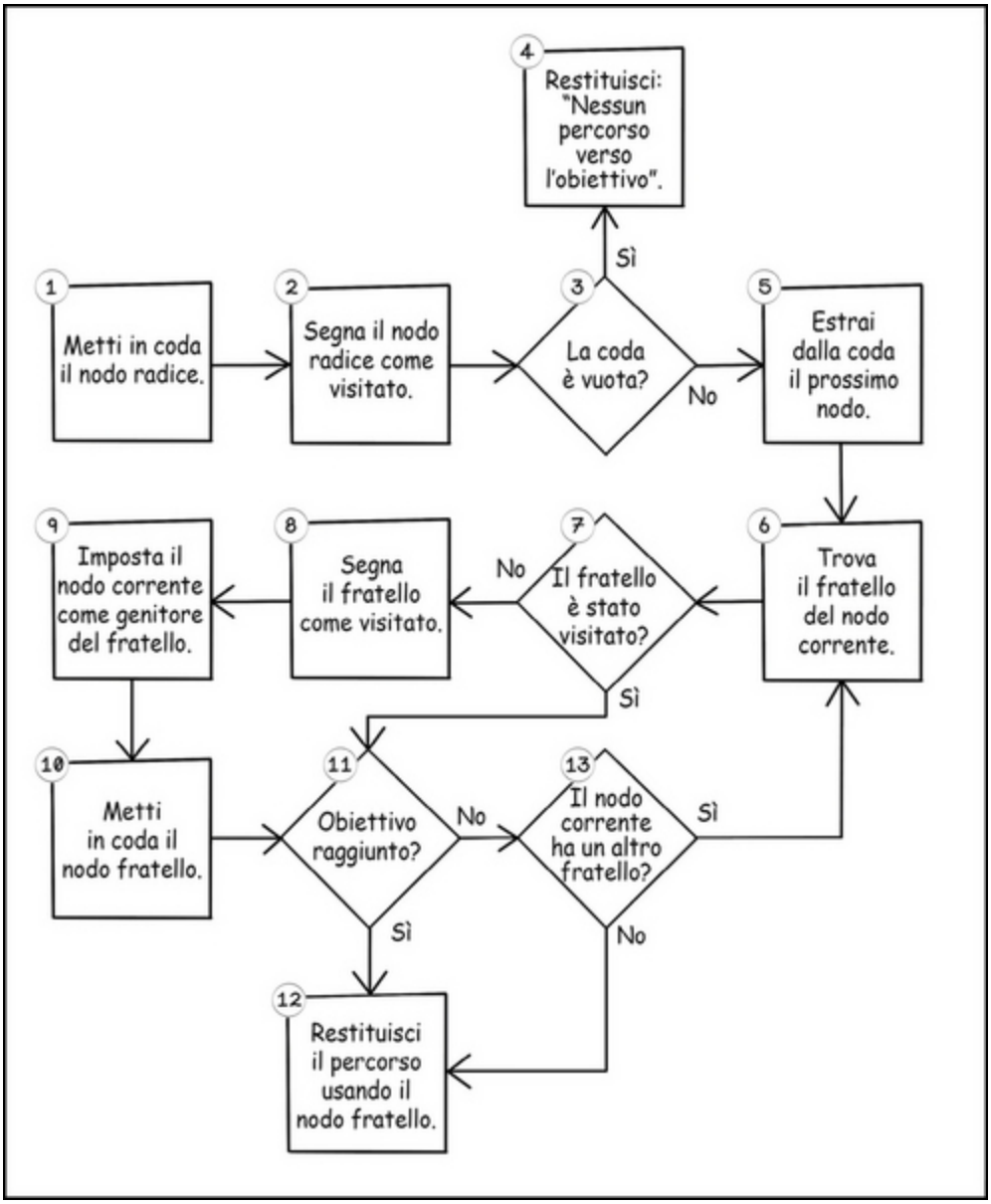


Figura 2.16 Flusso dell'algoritmo di ricerca in ampiezza.

Di seguito sono riportate alcune note e osservazioni aggiuntive per ciascuna fase del processo.

1. *Metti in coda il nodo radice*. L'algoritmo di ricerca in ampiezza viene implementato al meglio con una coda. Gli oggetti vengono elaborati nella sequenza in cui vengono aggiunti alla coda. Questo processo è noto anche come coda FIFO (*First In*, *First Out)*. Il

primo passaggio consiste nell'aggiungere il nodo radice alla coda. Questo nodo rappresenta la posizione di partenza del giocatore sulla mappa.

2. *Segna il nodo radice come visitato*. Ora che il nodo radice è stato aggiunto alla coda per l'elaborazione, viene contrassegnato come visitato, per evitare che venga visitato di nuovo senza motivo.
3. *La coda è vuota?* Se la coda è vuota (tutti i nodi sono stati elaborati dopo le iterazioni) e non è stato restituito alcun percorso nel Passaggio 12 dell'algoritmo, non esiste alcun percorso verso l'obiettivo. Se ci sono ancora nodi nella coda, l'algoritmo può continuare la sua ricerca per trovare l'obiettivo.
4. *Restituisci: "Nessun percorso verso l'obiettivo"*. Questo messaggio è l'unica possibile via d'uscita dall'algoritmo se non esiste alcun percorso verso l'obiettivo.
5. *Estrai dalla coda il prossimo nodo*. Estraendo l'oggetto successivo dalla coda e impostandolo come nodo corrente, possiamo esplorarne le caratteristiche. All'avvio dell'algoritmo, il nodo corrente sarà il nodo radice.
6. *Trova il fratello del nodo corrente*. Questo passaggio ottiene la prossima mossa possibile dalla posizione corrente, facendo riferimento al labirinto e determinando se è possibile un movimento a nord, sud, est o ovest.
7. *Il fratello è stato visitato?* Se il fratello corrente non è ancora stato visitato, non è stato ancora esplorato e può essere elaborato ora.
8. *Segna il fratello come visitato*. Questo passaggio indica che questo nodo fratello è stato visitato.
9. *Imposta il nodo corrente come genitore del fratello*. Imposta il nodo di origine come genitore del fratello. Questo passaggio è importante per tracciare il percorso dal fratello corrente al nodo radice. Dal punto di vista della mappa, l'origine è la posizione da

cui si è spostato il giocatore e il fratello corrente è la posizione in cui si è spostato il giocatore.

10. *Metti in coda il nodo fratello*. Il nodo adiacente viene messo in coda affinché possano essere esplorati i suoi figli. Questo meccanismo consente ai nodi di ogni singola profondità di essere elaborati in ordine.
11. *Obiettivo raggiunto?* Questo passaggio determina se il fratello corrente contiene l'obiettivo che l'algoritmo sta cercando.
12. *Restituisci il percorso usando il nodo fratello*. Facendo riferimento al genitore del nodo fratello, quindi al genitore di quel nodo e così via, si traccia il percorso dall'obiettivo alla radice. Il nodo radice sarà un nodo senza genitore.
13. *Il nodo corrente ha un altro fratello?* Se il nodo corrente ha altre mosse possibili nel labirinto, andate al Passaggio 6 per fare una mossa.

Esaminiamo come funzionerebbe la cosa in un semplice albero. Notate che, dato il modo in cui l'albero viene esplorato e i nodi vengono aggiunti alla coda FIFO, i nodi vengono elaborati nell'ordine desiderato sfruttando la coda (Figure 2.17 e 2.18).

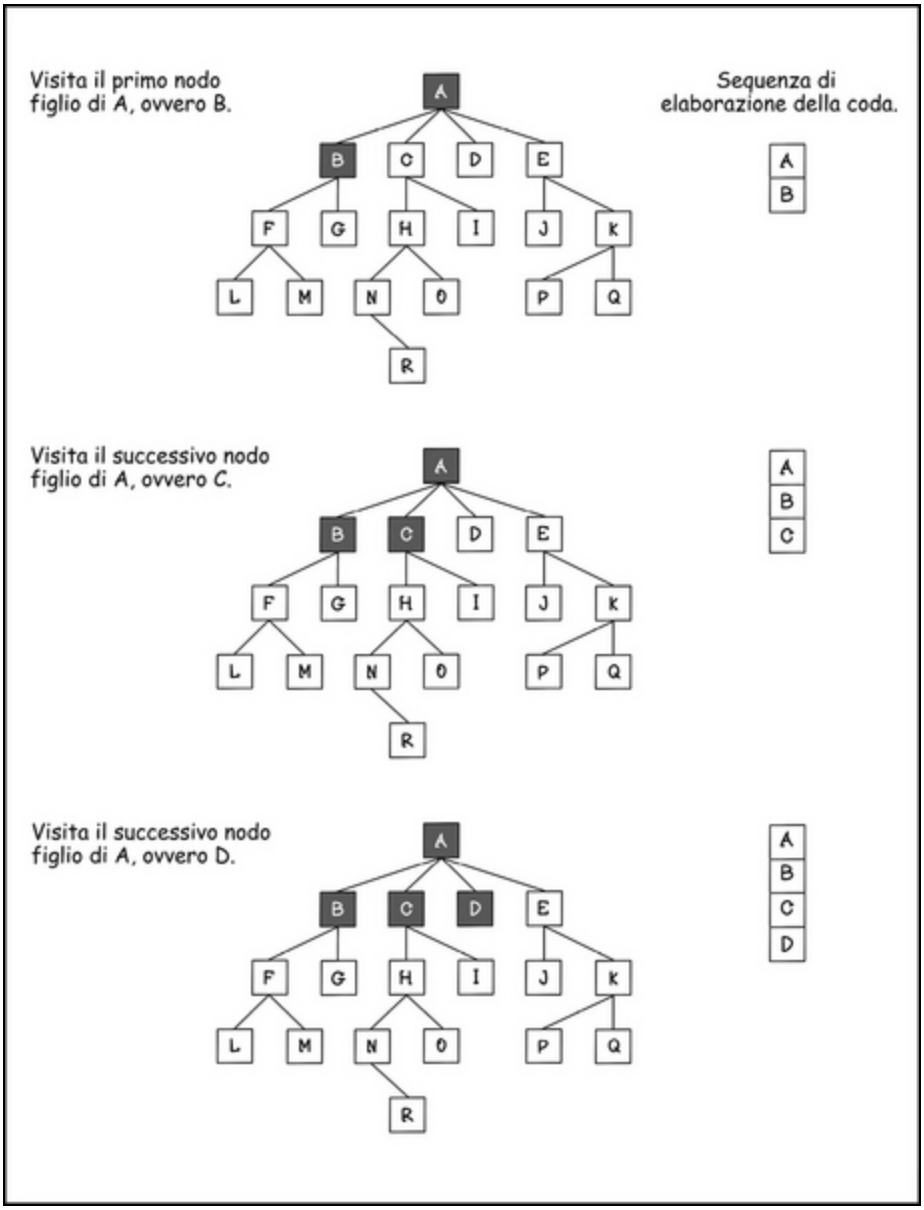


Figura 2.17 La sequenza di elaborazione dell'albero utilizzando la ricerca in ampiezza (Parte 1).

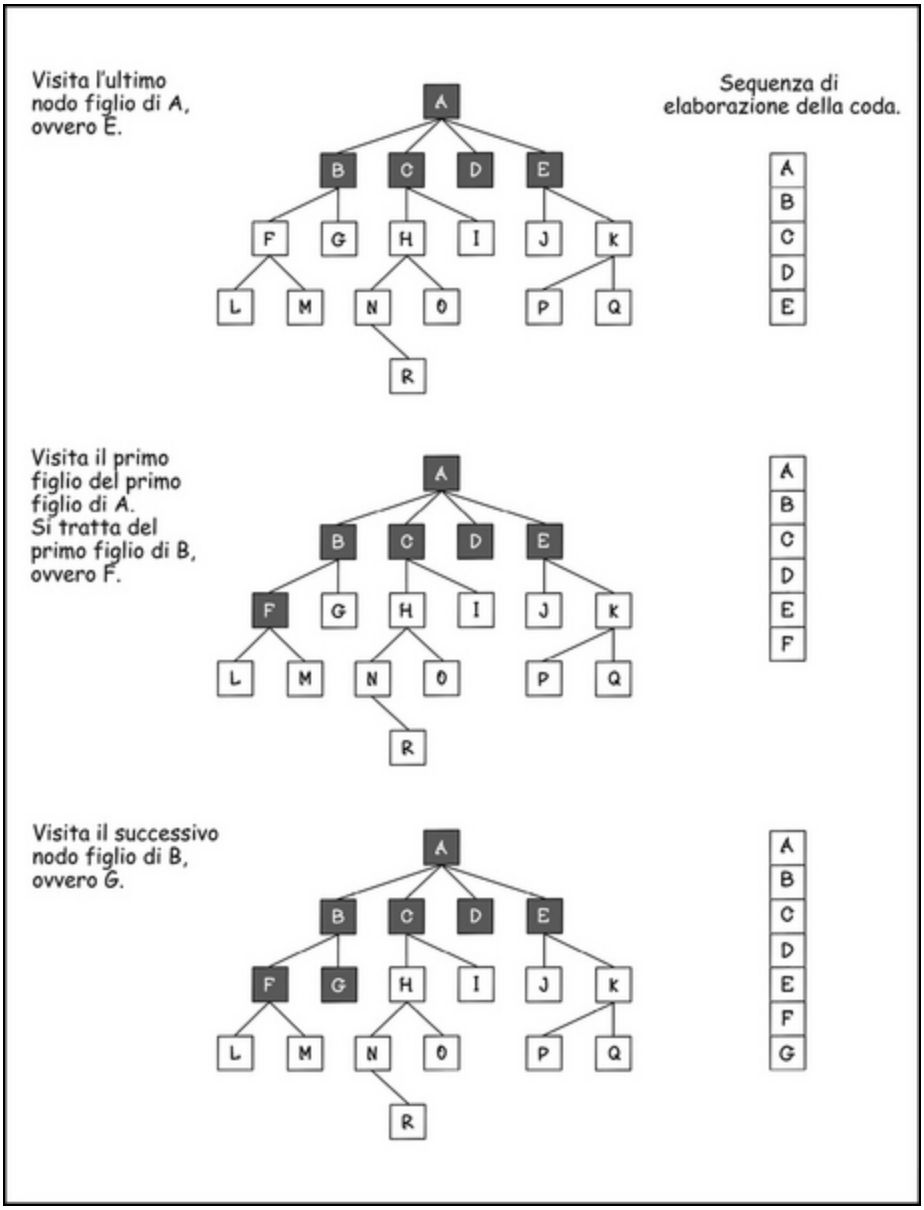
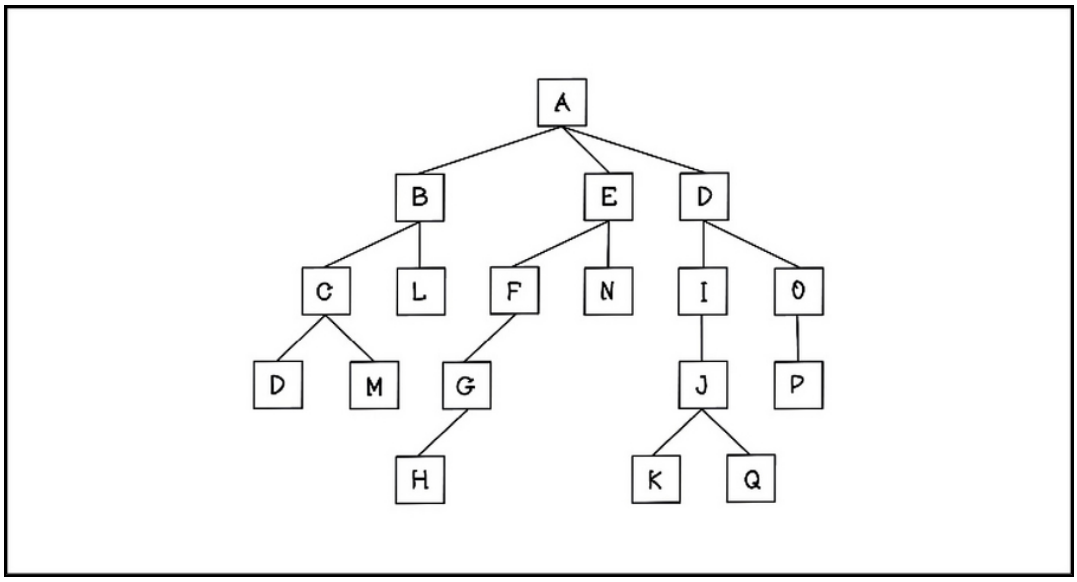


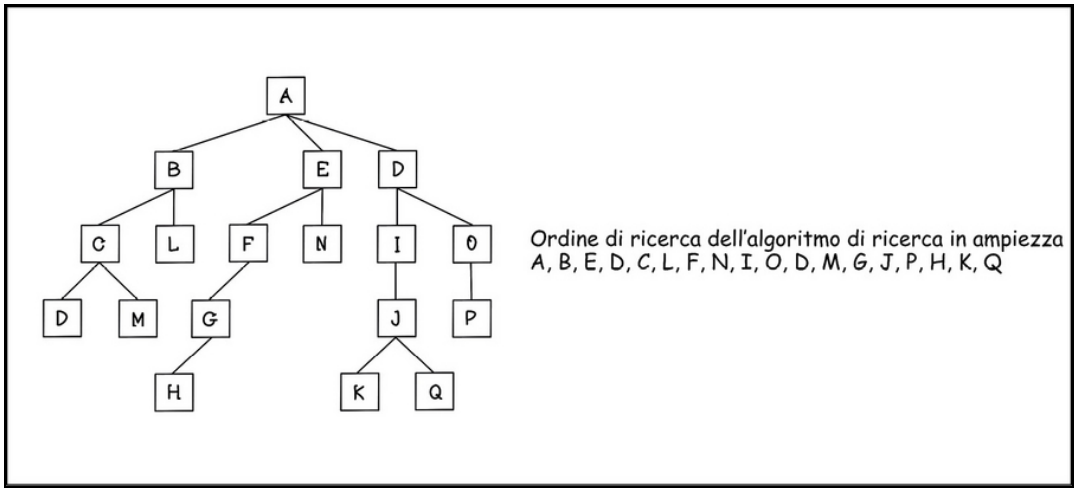
Figura 2.18 La sequenza di elaborazione dell'albero utilizzando la ricerca in ampiezza (Parte 2).

Esercizio: determinare il percorso verso la soluzione

Quale sarebbe l'ordine delle visite utilizzando la ricerca in ampiezza per il seguente albero?



Soluzione



Nell'esempio del labirinto, l'algoritmo parte dalla posizione attuale del giocatore nel labirinto, valuta tutte le possibili scelte delle mosse e ripete quella logica per ogni movimento effettuato, fino al raggiungimento dell'obiettivo. In questo modo, l'algoritmo genera un albero con un unico percorso verso l'obiettivo.

È importante capire che per generare il percorso vengono utilizzati i processi di visita dei nodi dell'albero. Stiamo trovando i nodi correlati utilizzando un meccanismo automatico.

Ogni percorso verso l'obiettivo consiste in una serie di mosse per raggiungere l'obiettivo. Il numero di mosse del percorso è la distanza per raggiungere l'obiettivo per quel percorso, un valore che chiameremo *costo*. Il numero di spostamenti è pari anche al numero di nodi visitati nel percorso, dal nodo radice al nodo foglia che rappresenta l'obiettivo. L'algoritmo si sposta lungo l'albero scendendo ogni volta a un nuovo livello di profondità, finché non trova un obiettivo; poi restituisce come soluzione il primo percorso che l'ha portato a tale obiettivo. Potrebbe esserci un percorso migliore verso l'obiettivo, ma poiché la ricerca in ampiezza procede alla cieca, non è garantito che lo trovi.

NOTA

Nell'esempio del labirinto, tutti gli algoritmi di ricerca utilizzati terminano quando hanno trovato l'obiettivo. È possibile consentire a questi algoritmi di trovare più soluzioni con una piccola modifica, ma i migliori casi d'uso per gli algoritmi di ricerca si limitano a trovare un unico obiettivo, poiché spesso è troppo costoso esplorare l'intero albero delle possibilità.

La Figura 2.19 mostra la generazione di un albero utilizzando i movimenti nel labirinto. Poiché l'esplorazione segue una ricerca in ampiezza, prima vengono esplorati i nodi a un dato livello di profondità e solo dopo viene iniziata l'esplorazione della profondità successiva (Figura 2.20).

Pseudocodice

Come ho accennato, l'algoritmo di ricerca in ampiezza utilizza una coda per generare un albero, un livello di profondità alla volta. Avere una struttura per archiviare i nodi visitati è fondamentale per evitare di rimanere bloccati in circuiti ciclici; e l'impostazione del genitore di ciascun nodo è importante per determinare un percorso dal punto di partenza nel labirinto all'obiettivo:

```
run_bfs(maze, current_point, visited_points):
 let q equal a new queue
  push current_point to q
  mark current_point as visited
  while q is not empty:
      pop q and let current_point equal the returned point
```

```
        add available cells north, east, south, and west to a list neighbors
        for each neighbor in neighbors:
            if neighbor is not visited:
                set neighbor parent as current_point
                mark neighbor as visited
                push neighbor to g
                if value at neighbor is the goal:
                    return path using neighbor
    return "No path to goal"
```

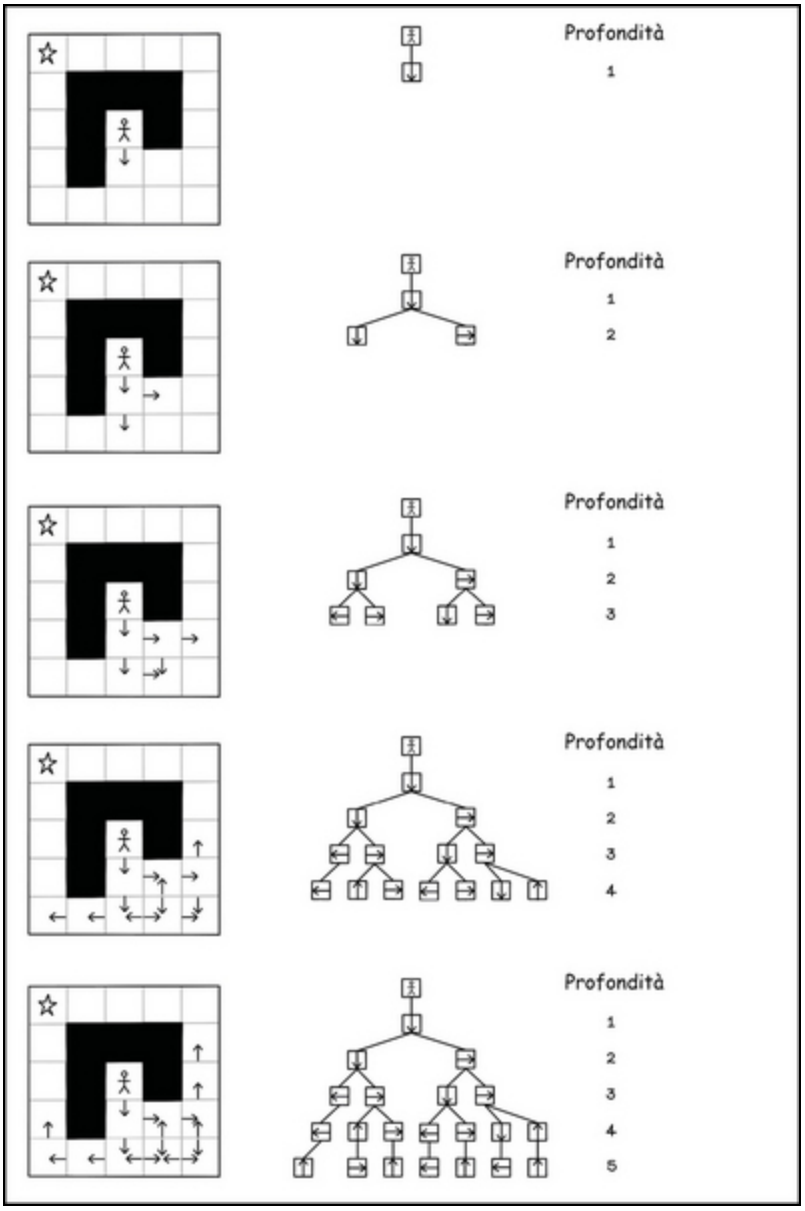


Figura 2.19 Generazione dell'albero dei movimenti nel labirinto usando la ricerca in ampiezza.

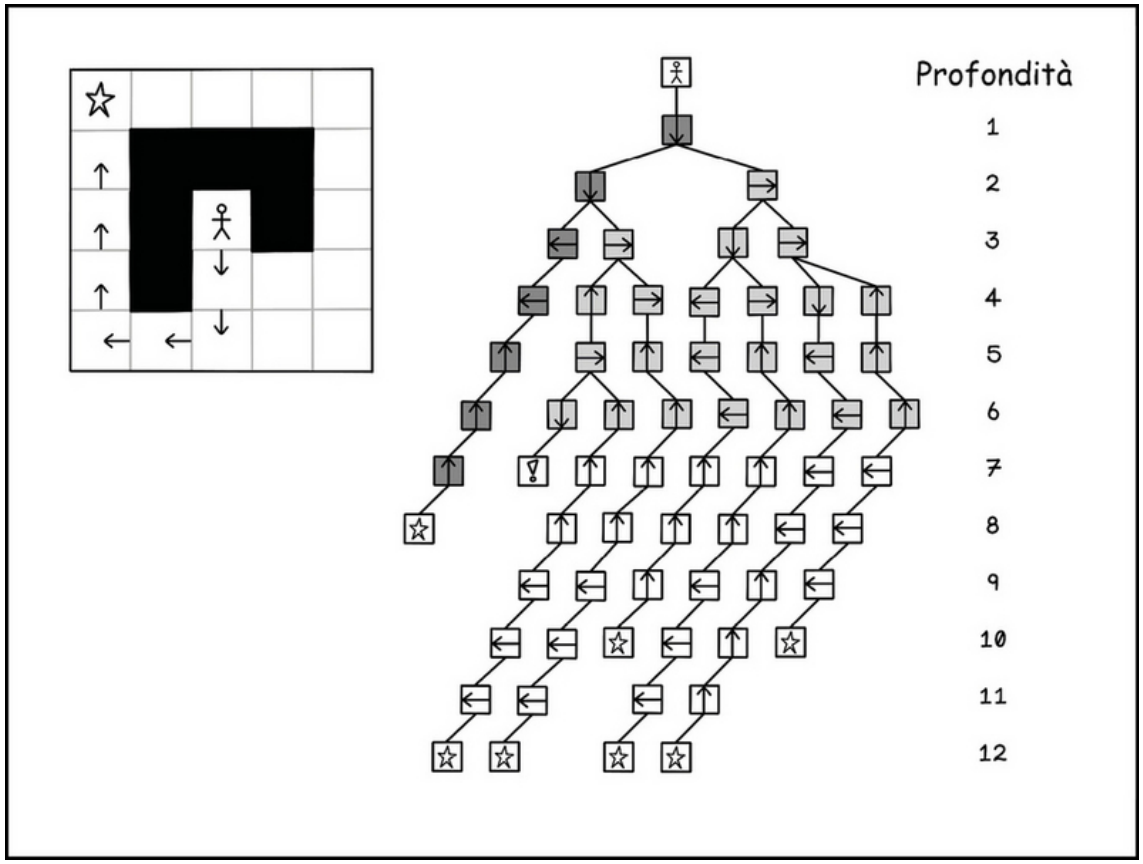


Figura 2.20 Nodi visitati nell'intero albero dopo la ricerca in ampiezza.

Ricerca in profondità: prima in profondità e poi in ampiezza

La *ricerca in profondità* è un altro algoritmo utilizzato per attraversare un albero o per generare nodi e percorsi in un albero. Questo algoritmo parte da un determinato nodo ed esplora in profondità i percorsi relativi ai nodi connessi al primo figlio, procedendo in modo ricorsivo fino a raggiungere il nodo foglia più lontano, per poi tornare indietro ed esplorare altri percorsi verso nodi foglia ma da altri nodi figli visitati. La Figura 2.21 illustra il flusso generale dell'algoritmo di ricerca in profondità.

Esaminiamo il flusso dell'algoritmo di ricerca in profondità.

- *Aggiungi il nodo radice allo stack*. L'algoritmo di ricerca in profondità può essere implementato utilizzando uno stack, in cui viene elaborato per primo l'ultimo oggetto inserito. Questo processo è noto come coda LIFO (*Last In*, *First Out)*. Il primo passaggio consiste nell'aggiungere allo stack il nodo radice.
- *Lo stack è vuoto?* Se lo stack è vuoto e non è stato restituito alcun percorso nel Passaggio 8 dell'algoritmo, non esiste alcun percorso verso l'obiettivo. Se invece ci sono ancora nodi nello stack, l'algoritmo può continuare la sua ricerca per trovare l'obiettivo.
- *Restituisci "Nessun percorso verso l'obiettivo"*. Questa è l'unica possibile via d'uscita dall'algoritmo se non esiste alcun percorso verso l'obiettivo.

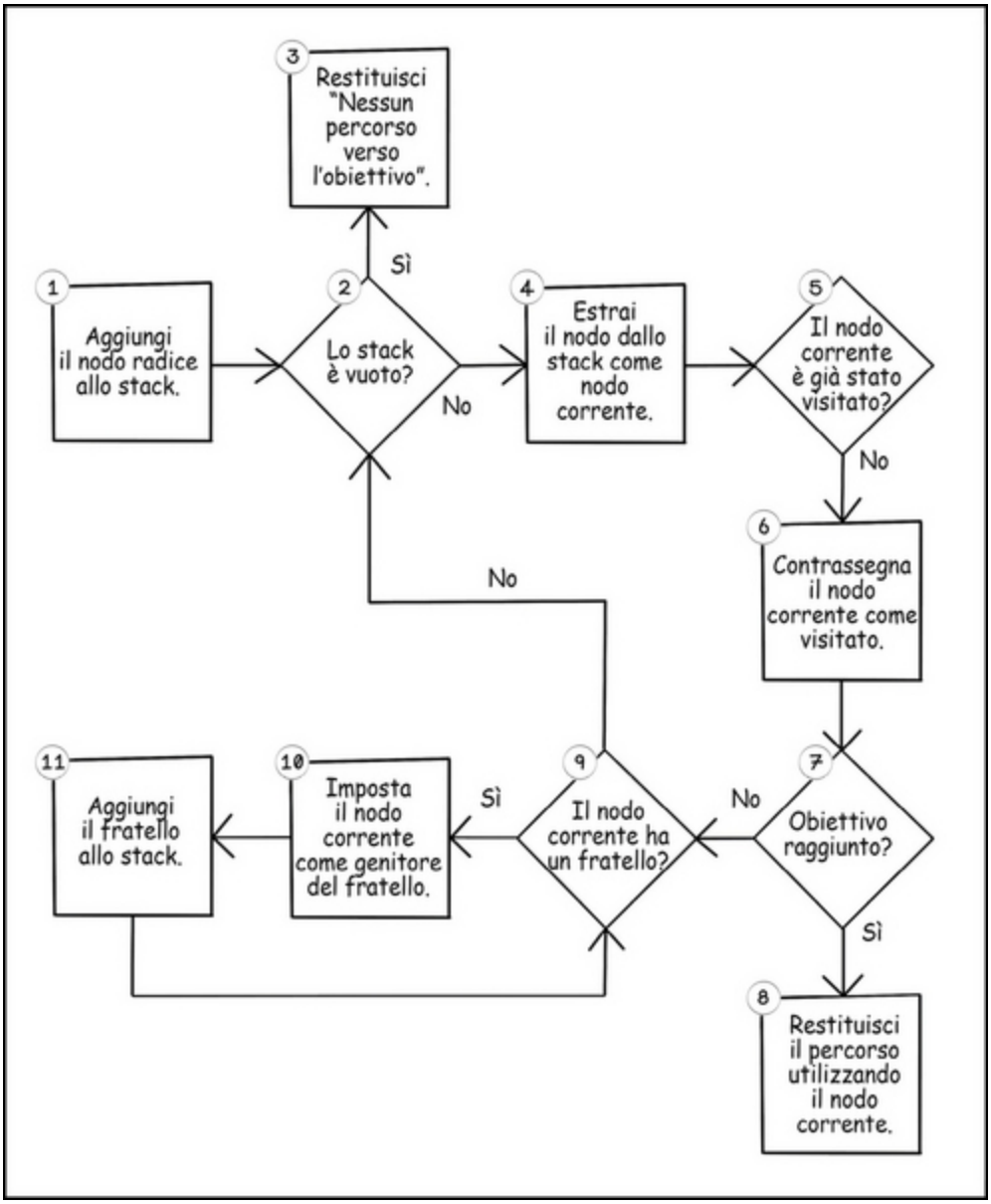


Figura 2.21 Flusso dell'algoritmo di ricerca in profondità.

- *Estrai il nodo dallo stack come nodo corrente*. Estraendo l'oggetto successivo dallo stack e impostandolo come nodo corrente, possiamo esplorarne le caratteristiche.
- *Il nodo corrente è già stato visitato?* Se il nodo corrente non è stato visitato, non è stato ancora esplorato e può essere elaborato ora.

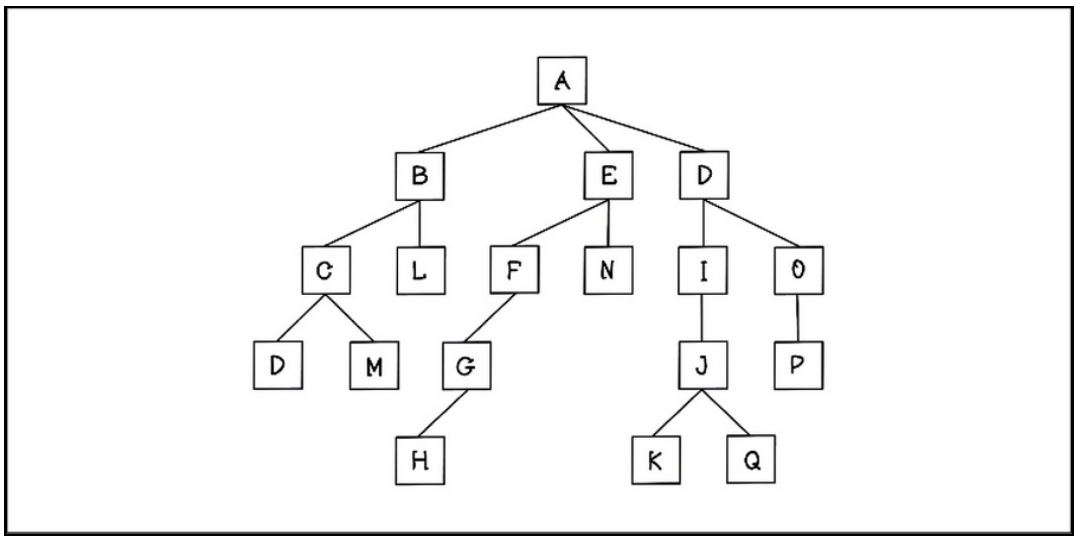
- *Contrassegna il nodo corrente come visitato*. Questo passaggio indica che questo nodo è stato visitato, per impedire che venga elaborato una seconda volta.
- *Obiettivo raggiunto?* Questo passaggio determina se il fratello corrente contiene l'obiettivo che l'algoritmo sta cercando.
- *Restituisci il percorso utilizzando il nodo corrente*. Facendo riferimento al genitore del nodo corrente, quindi al genitore di quel nodo e così via, viene creato il percorso dall'obiettivo alla radice. Il nodo radice sarà un nodo senza genitore.
- *Il nodo corrente ha un fratello?* Se il nodo corrente ha altre mosse possibili, tale mossa può essere aggiunta allo stack per essere elaborata. In caso contrario, l'algoritmo può tornare al Passaggio 2, in cui può essere elaborato l'oggetto successivo dello stack, se non è vuoto. La natura dello stack LIFO consente all'algoritmo di elaborare tutti i nodi fino a raggiungere la profondità di un nodo foglia, prima di tornare indietro per visitare altri figli del nodo radice.
- *Imposta il nodo corrente come genitore del fratello*. Imposta il nodo di origine come genitore del fratello corrente. Questo passaggio è importante per tracciare il percorso dal fratello corrente al nodo radice. Dal punto di vista della mappa, l'origine è la posizione da cui si è spostato il giocatore e il fratello corrente è la posizione in cui si è spostato il giocatore.
- *Aggiungi il fratello allo stack*. Il nodo adiacente viene aggiunto allo stack affinché possano essere esplorati i suoi figli. Ancora una volta, questo meccanismo a stack consente di elaborare i nodi alla massima profondità dell'albero prima di elaborare i fratelli a minori profondità.

Le Figure 2.22 e 2.23 illustrano come la ricerca in profondità utilizza lo stack LIFO per visitare i nodi nell'ordine desiderato. Notate

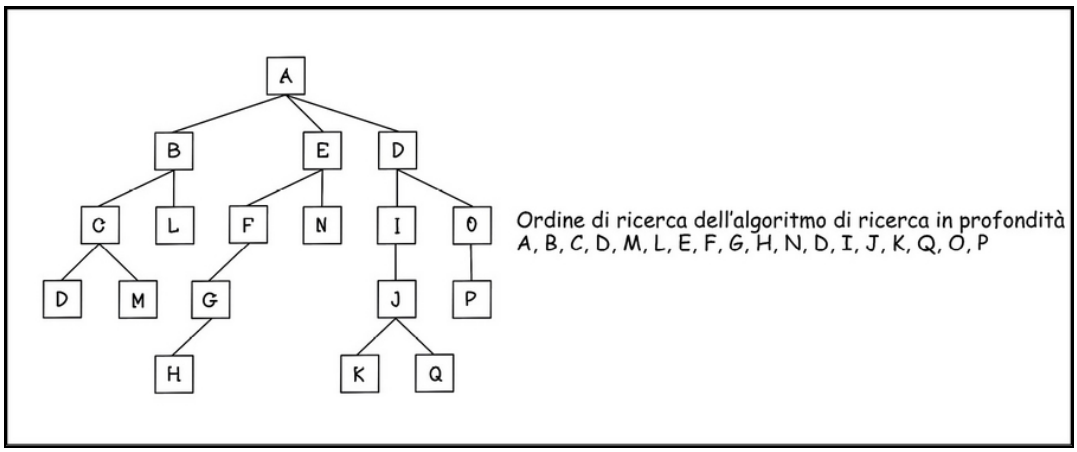
che i nodi vengono inseriti ed estratti dallo stack a mano a mano che progredisce la profondità dei nodi visitati. L'inserimento di oggetti nello stack è chiamato *push* e l'estrazione di oggetti dallo stack è chiamata *pop*.

Esercizio: determinare il percorso verso la soluzione

Quale sarebbe l'ordine delle visite nella ricerca in profondità del seguente albero?



Soluzione



È importante capire che l'ordine dei figli cambia molto quando si utilizza la ricerca in profondità, poiché l'algoritmo esplora sempre

prima il primo figlio, finché non trova un nodo foglia, e a quel punto arretra di un livello.

Nell'esempio del labirinto, l'ordine delle mosse (nord, sud, est e ovest) influenza il percorso verso l'obiettivo individuato dall'algoritmo. Un cambiamento nell'ordine produrrà una soluzione differente. Le direzioni rappresentate nelle Figure 2.24 e 2.25 non contano; ciò che conta è l'ordine delle scelte delle mosse.

Pseudocodice

Sebbene l'algoritmo di ricerca in profondità possa essere implementato con una funzione ricorsiva, esaminiamo un'implementazione ottenuta con uno stack, per chiarire meglio l'ordine in cui i nodi vengono visitati ed elaborati. È importante tenere traccia dei nodi visitati, in modo da non tornare a visitare inutilmente gli stessi nodi, creando circuiti ciclici:

```
run_dfs(maze, root_point, visited_points):
  let s equal a new stack
  add root_point to s
  while s is not empty
      pop s and let current_point equal the returned point
      if current_point is not visited:
          mark current_point as visited
          if value at current_node is the goal:
              return path using current_point
          else:
                add available cells north, east, south, and west to a list
neighbors
              for each neighbor in neighbors:
                  set neighbor parent as current_point
                  push neighbor to s
  return "No path to goal"
```

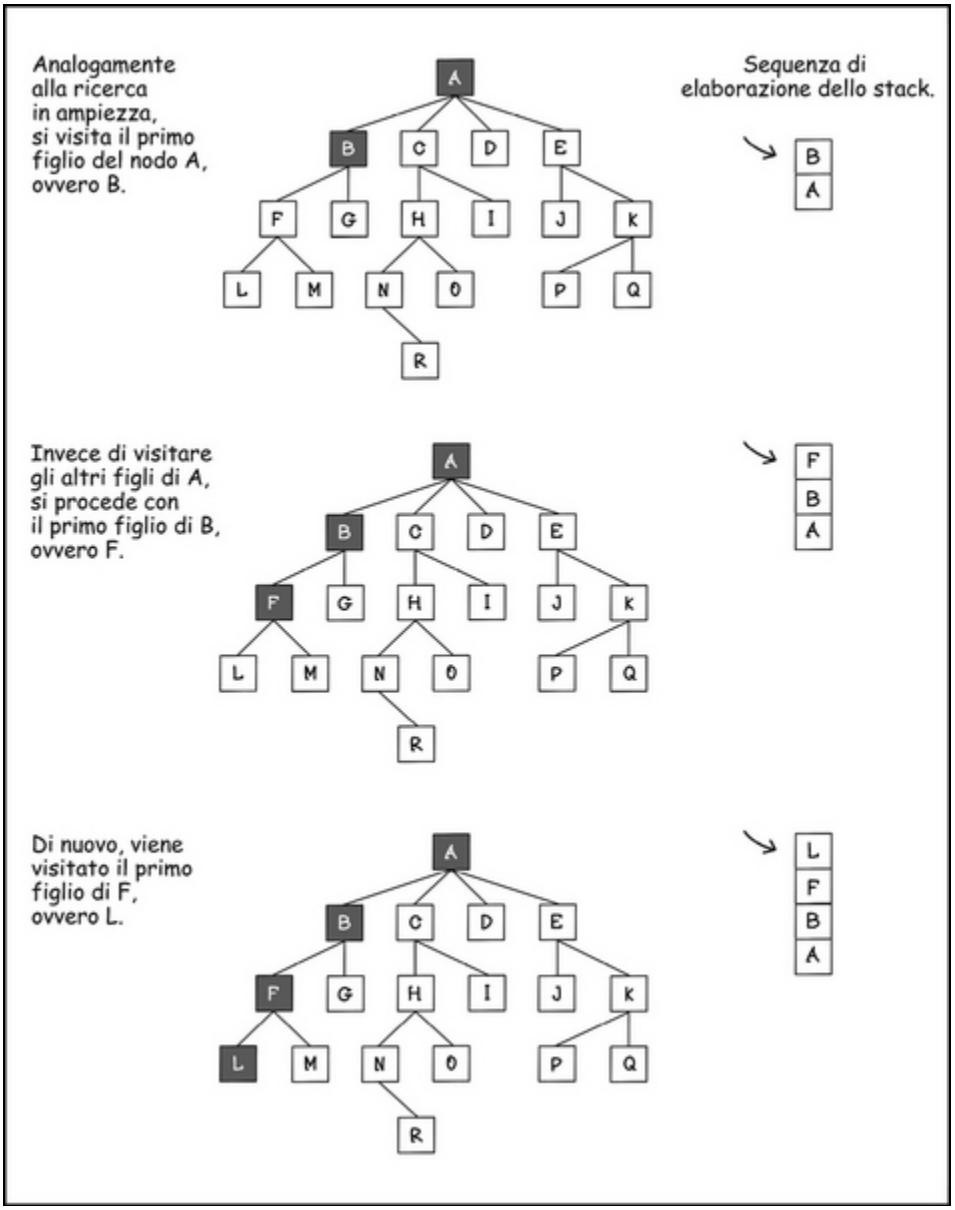


Figura 2.22 La sequenza di elaborazione dell'albero usando la ricerca in profondità (Parte 1).

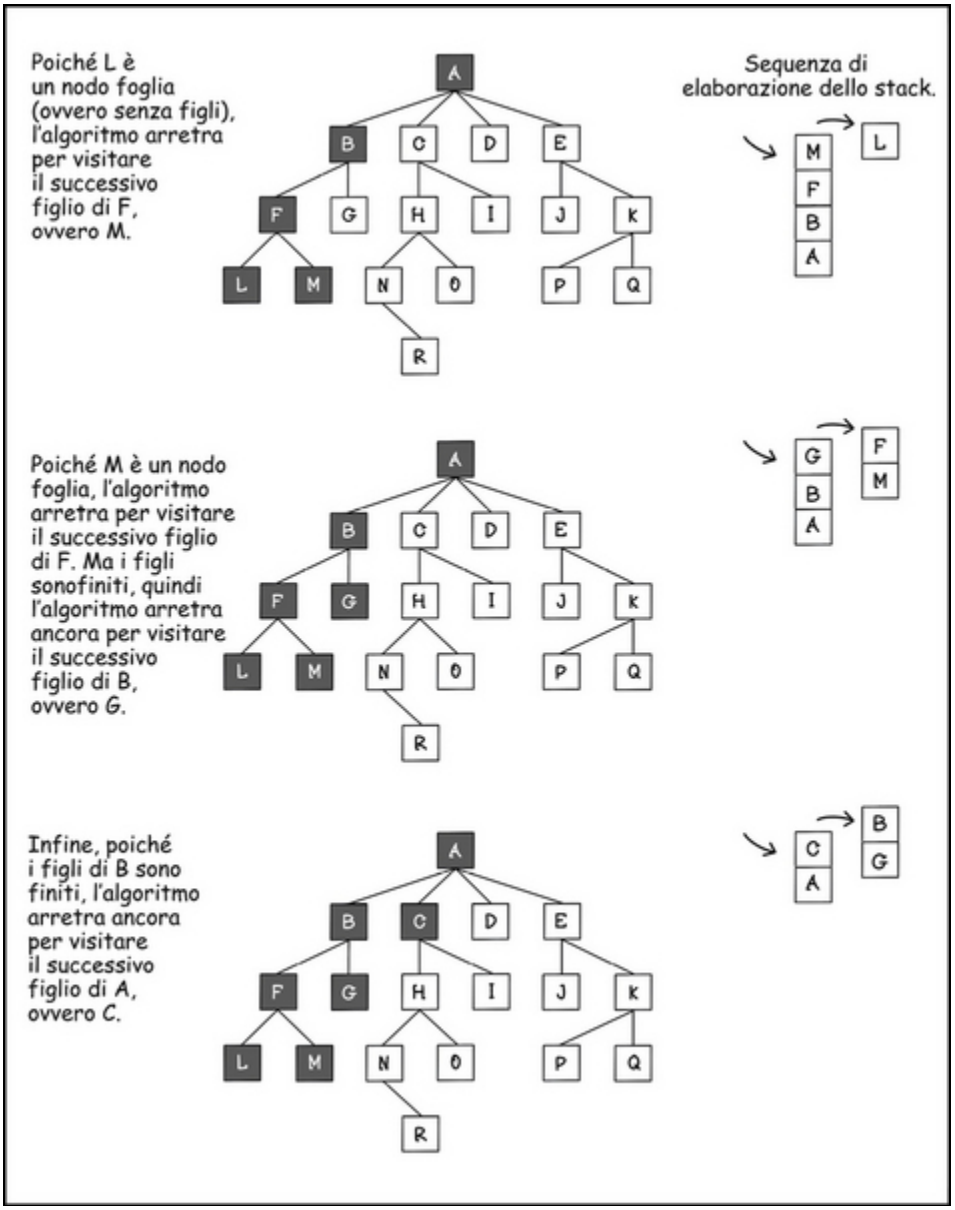


Figura 2.23 La sequenza di elaborazione dell'albero usando la ricerca in profondità (Parte 2).

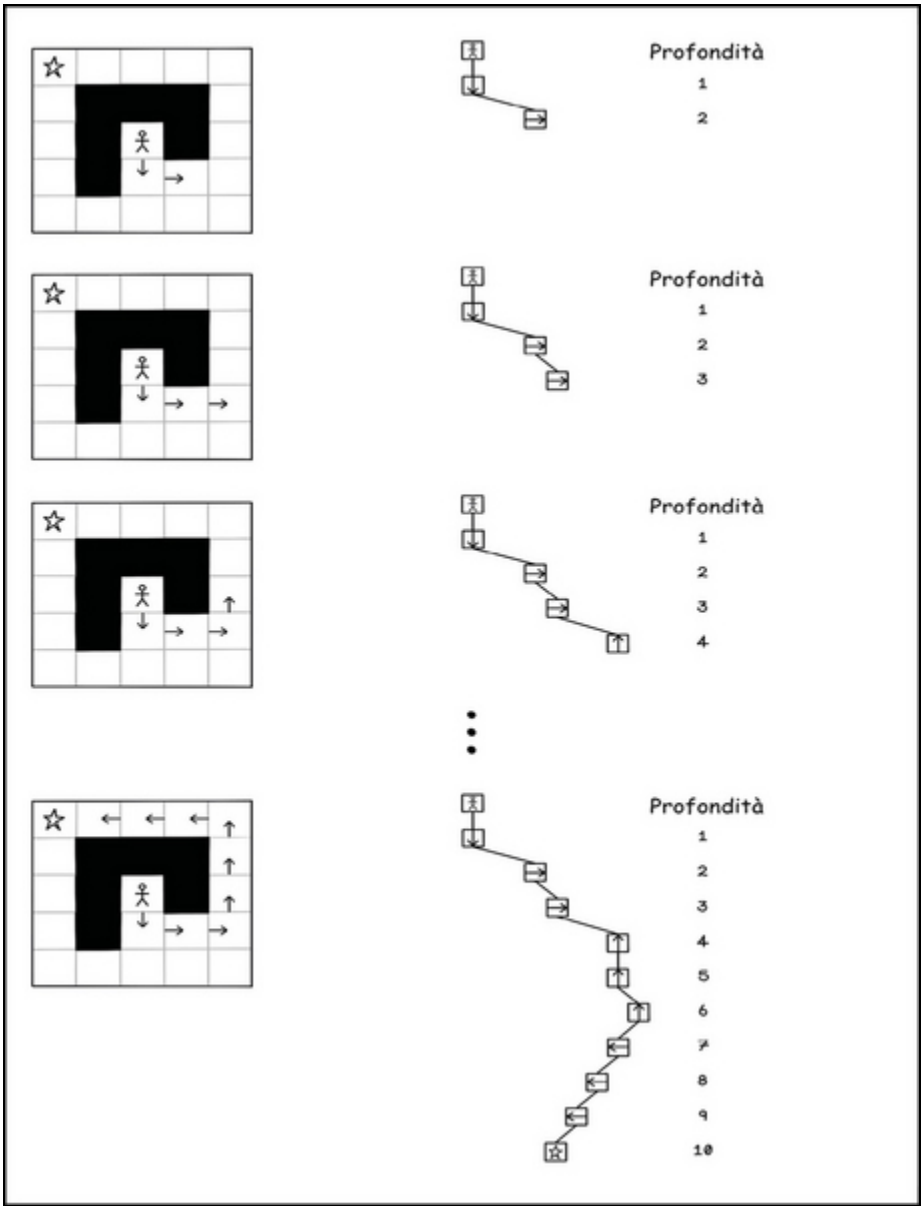


Figura 2.24 Generazione dell'albero dei movimenti nel labirinto usando la ricerca in profondità.

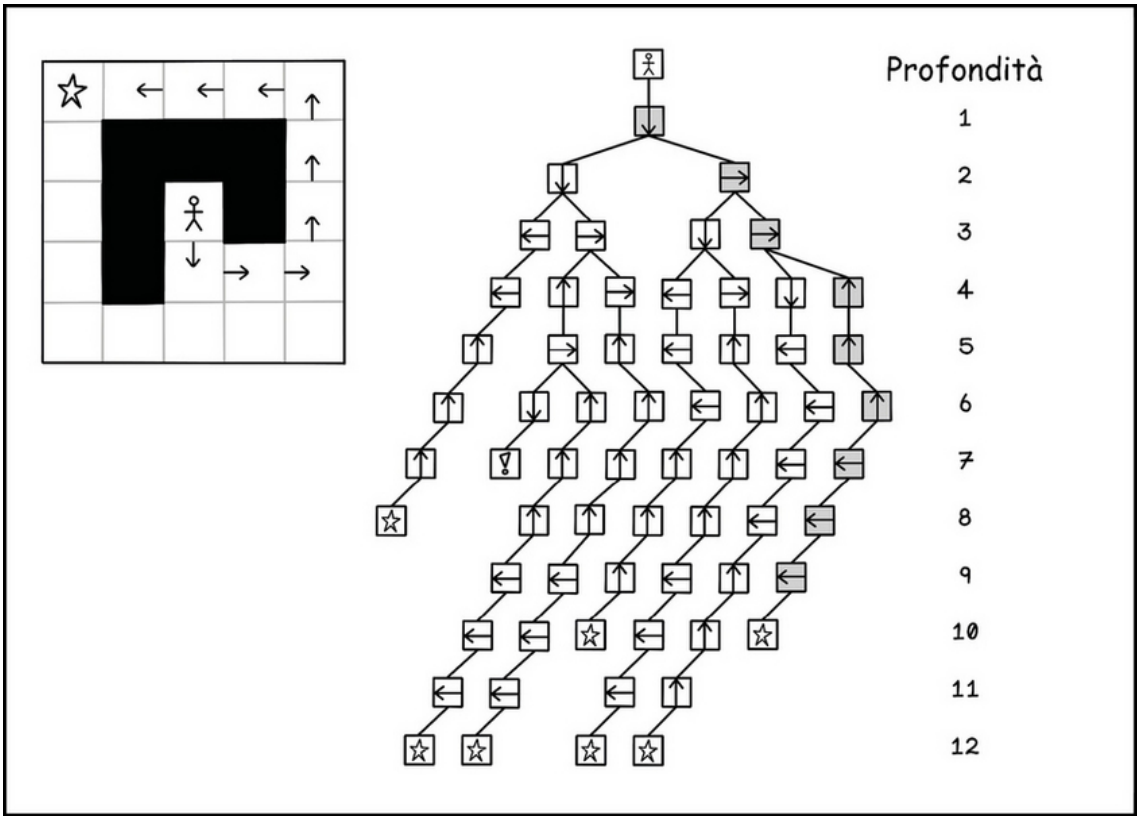


Figura 2.25 Nodi visitati nell'intero albero dopo la ricerca in profondità.

Casi d'uso per gli algoritmi di ricerca non informati

Gli algoritmi di ricerca non informati sono versatili e utili in diversi casi d'uso del mondo reale. Ecco alcuni esempi.

- *Ricerca di percorsi fra i nodi di una rete*: quando due computer devono comunicare su una rete, la connessione attraversa molti computer e dispositivi. Gli algoritmi di ricerca possono essere utilizzati per trovare in quella rete un percorso fra i due dispositivi.

- *Scansione di pagine web*: le ricerche web ci consentono di trovare informazioni in uno sterminato insieme di pagine web. Per indicizzare queste pagine web, i crawler in genere leggono le informazioni presenti in ciascuna pagina, oltre a seguire in modo ricorsivo ogni link presente in quella pagina. Gli algoritmi di ricerca sono utili per creare crawler, strutture di metadati e relazioni fra i contenuti.
- *Trovare connessioni nei social network*: le applicazioni social ospitano molte persone e le loro relazioni. Bob può essere amico di Alice, per esempio, la quale è amica di John; quindi Bob e John sono connessi solo indirettamente tramite Alice. Un'applicazione di social media può suggerire che Bob e John diventino amici, perché potrebbero conoscersi grazie alla reciproca amicizia con Alice.

Facoltativo: ulteriori informazioni sulle categorie di grafi

I grafi sono utili per molti problemi informatici e matematici e, a causa della natura dei diversi tipi di grafi, a determinate categorie di grafi possono essere applicati principi e algoritmi differenti. Un grafo può essere classificato in base alla sua struttura, al numero di nodi, al numero di archi e all'interconnettività fra i nodi.

È utile conoscere queste categorie di grafi, in quanto spesso sono citate nelle ricerche e in altri algoritmi di intelligenza artificiale.

- *Grafo non orientato*: il grafo non ha archi orientati. Le relazioni fra due nodi sono reciproche. Come le strade fra città, prevedono entrambe le direzioni di movimento.

- *Grafo orientato*: gli archi indicano la direzione. Le relazioni fra due nodi sono esplicite. Come in un grafo che rappresenta un figlio e un genitore, il nodo figlio non può essere "genitore del suo genitore".
- *Grafo disconnesso*: uno o più nodi non sono collegati da alcun arco. Come in un grafo che rappresenti i punti di contatto fra continenti, alcuni nodi non sono collegati. Come nel caso dei continenti, alcuni sono collegati dalla terra e altri sono separati dagli oceani.
- *Grafo aciclico*: un grafo che non contiene cicli. Come nel caso del tempo, il grafo non può tornare indietro a nessun punto del passato.
- *Grafo completo*: ogni nodo è connesso a ogni altro nodo da un arco. Come nelle linee di comunicazione di un piccolo team, tutti parlano con tutti gli altri.
- *Grafo bipartito completo*: una *partizione di archi* è un raggruppamento di archi. Data una partizione di archi, ogni nodo di una partizione è connesso a ogni nodo dell'altra partizione tramite archi. Come in una degustazione di formaggi, tipicamente, ogni singola persona assaggia ogni singolo tipo di formaggio.
- *Grafo pesato*: un grafo in cui gli archi fra i nodi hanno un peso. Come nella distanza fra le città, alcune città sono più distanti fra loro di altre. Le loro connessioni, quindi, "pesano" di più.

È utile comprendere i diversi tipi di grafi per descrivere al meglio il problema in questione e utilizzare l'algoritmo più efficiente per l'elaborazione (Figura 2.26). Alcune di queste categorie di grafi saranno trattate nei prossimi capitoli, come il Capitolo 6 sull'ottimizzazione a colonia di formiche e il Capitolo 8 sulle reti neurali.

Facoltativo: altri modi per rappresentare i grafi

A seconda del contesto, altre codifiche di grafi possono essere più efficienti per l'elaborazione o più facili da utilizzare, a seconda del linguaggio di programmazione e degli strumenti che state utilizzando.

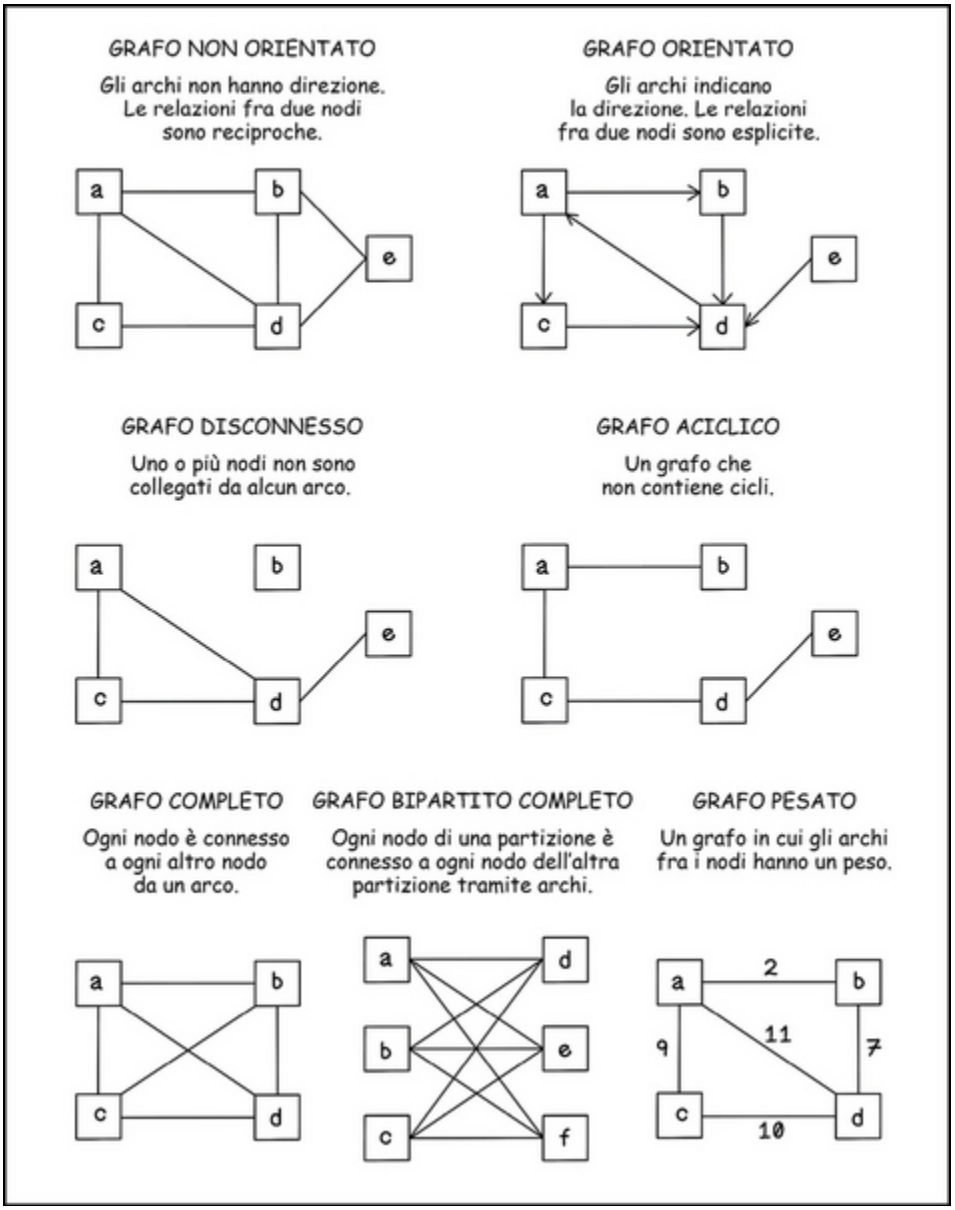


Figura 2.26 Tipi di grafi.

Matrice di incidenza

Una *matrice di incidenza* utilizza una matrice in cui l'altezza equivale al numero di nodi del grafo e la larghezza è il numero di archi. Ogni riga rappresenta le relazioni di un nodo. Se un nodo non è connesso da un arco, nella cella viene memorizzato il valore `0`. Se un nodo riceve una connessione da un altro nodo, nel caso di un grafo orientato, viene memorizzato il valore `-1`. Se un nodo è connesso in uscita a un altro nodo (o è connesso nel caso di un grafo non orientato), viene memorizzato il valore `1`. Una matrice di incidenza può essere utilizzata per rappresentare grafici orientati e non orientati (Figura 2.27).

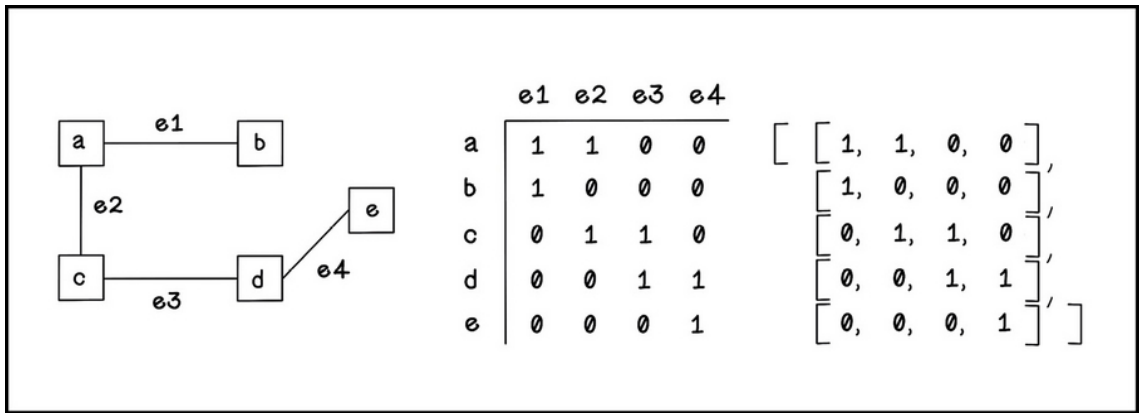


Figura 2.27 Rappresentazione di un grafo tramite una matrice di incidenza.

Lista di adiacenza

Una *lista di adiacenza* utilizza liste concatenate in cui la dimensione della lista iniziale è il numero di nodi presenti nel grafo e ogni valore rappresenta i nodi connessi a un determinato nodo. Una lista di adiacenza può essere utilizzata per rappresentare grafi sia orientati sia non orientati (Figura 2.28).

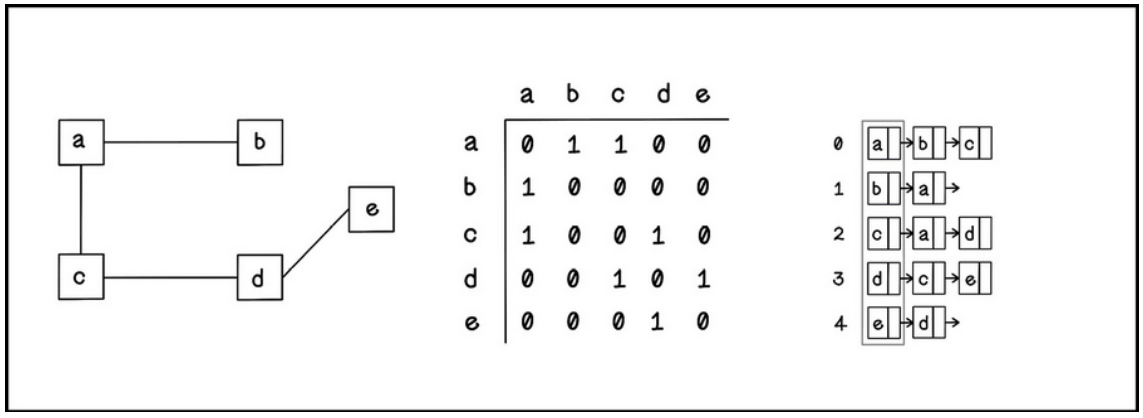


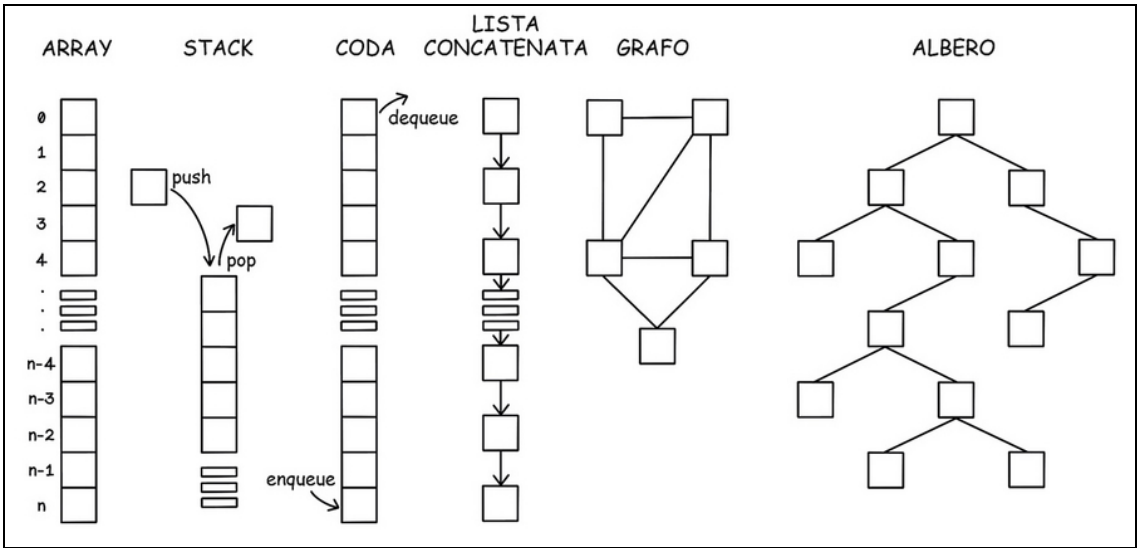
Figura 2.28 Rappresentazione di un grafo come una lista di adiacenza.

I grafi sono strutture di dati interessanti e utili, perché possono essere facilmente rappresentati come equazioni matematiche, offrendo quindi un supporto a tutti gli algoritmi che utilizziamo. Troverete maggiori informazioni su questo argomento un po' in tutto il libro.

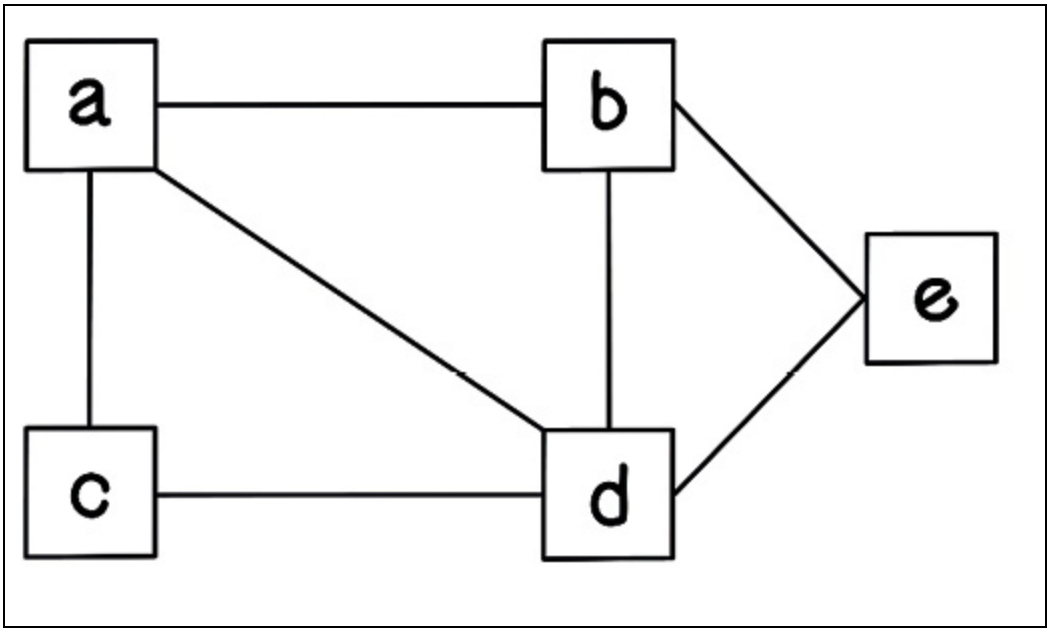
Riepilogo

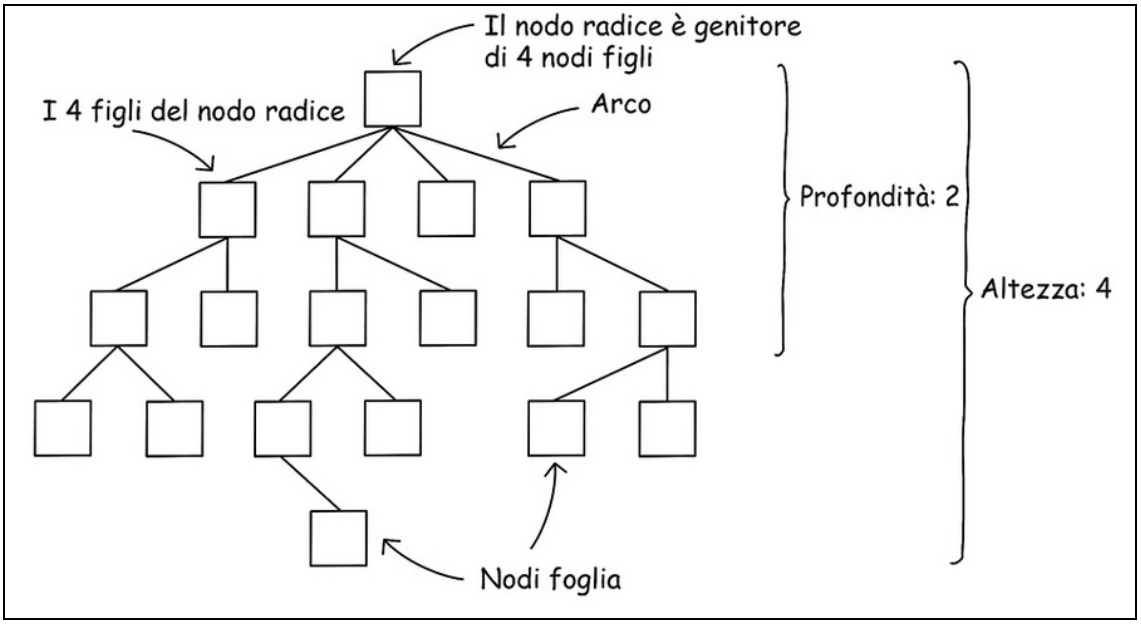
Le strutture di dati sono importanti per risolvere i problemi.

- Gli algoritmi di ricerca sono utili nella pianificazione e ricerca di soluzioni in alcuni ambienti soggetti a cambiamenti.



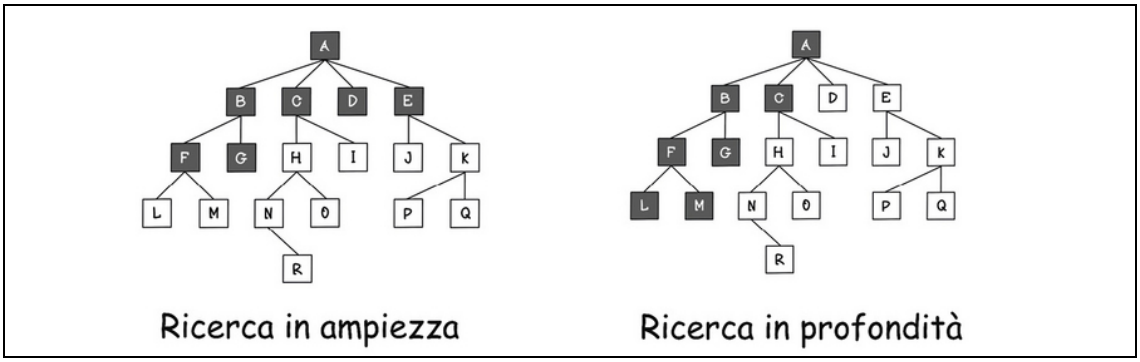
- I grafi e gli alberi sono strutture di dati utili nell'intelligenza artificiale.





- La ricerca non informata è cieca e può essere costosa dal punto di vista computazionale. Può essere utile l'impiego di strutture di dati corrette.
- La ricerca in ampiezza procede prima in ampiezza e poi in profondità. La ricerca in profondità procede prima in profondità e

poi in ampiezza.



Ricerca intelligente

Definizione di euristica: applicazione di scelte informate

Ora che abbiamo un'idea di come funzionano gli algoritmi di ricerca non informati, possiamo esplorare come questi possono essere migliorati potendo contare su maggiori informazioni sul problema. A tale scopo, utilizziamo la ricerca informata. Si ha una *ricerca informata* quando l'algoritmo può contare su un contesto relativo al problema da risolvere. Le euristiche sono un modo per rappresentare questo contesto. Spesso chiamata *regola empirica*, un'*euristica* è una regola (o un insieme di regole) utilizzata per valutare uno stato. Può essere utilizzata per definire i criteri che uno stato deve soddisfare o per misurare le prestazioni di un determinato stato. Un'euristica viene utilizzata quando non esiste un metodo ben definito per trovare una soluzione ottimale. Un'euristica è un po' come un'ipotesi, e dovrebbe essere considerata più come una linea guida che come una verità scientifica relativa al problema che si deve risolvere.

Quando ordinate una pizza, per esempio, la vostra euristica di quanto sarà buona può essere definita dagli ingredienti utilizzati. Se vi piace avere molta salsa, doppio formaggio, funghi e ananas su una base spessa e con crosta croccante, una pizza comprendente un maggior numero di questi attributi vi emozionerà di più e otterrà un punteggio migliore in questa particolarissima euristica. Una pizza comprendente

meno di questi attributi sarà meno attraente e otterrà un punteggio inferiore.

Un altro esempio è la scrittura di algoritmi per risolvere un problema di navigazione GPS. L'euristica potrebbe essere "I buoni percorsi riducono al minimo il tempo nel traffico e minimizzano la distanza percorsa" oppure "I buoni percorsi riducono al minimo i pedaggi e massimizzano le buone condizioni stradali". Una scarsa euristica per un programma di navigazione GPS potrebbe cercare di ridurre al minimo la distanza in linea retta fra due punti. Questa euristica potrebbe funzionare per gli uccelli o gli aerei, ma non per chi deve camminare o guidare: siamo legati a strade, vie e sentieri, che aggirano edifici e ostacoli. L'euristica deve quindi essere sensata per il contesto in cui viene impiegata.

Un esempio: occorre controllare se una clip audio caricata è tratta da una libreria di contenuti protetti da copyright. Poiché le clip audio sono costituite da frequenze sonore, un modo per raggiungere questo obiettivo consiste nel ricercare ogni frammento del clip caricato in ogni clip contenuto nella libreria. Questo compito richiederà un'enorme quantità di calcolo. Un approccio primitivo per realizzare una ricerca migliore potrebbe consistere nel definire un'euristica che minimizzi la differenza di distribuzione delle frequenze fra le due clip, come mostrato nella Figura 3.1. Notate che le frequenze sono identiche, a parte la differenza temporale; non vi sono differenze nelle distribuzioni di frequenze. Questa soluzione potrebbe non essere perfetta, ma è un buon inizio verso la realizzazione di un algoritmo meno costoso dal punto di vista computazionale.

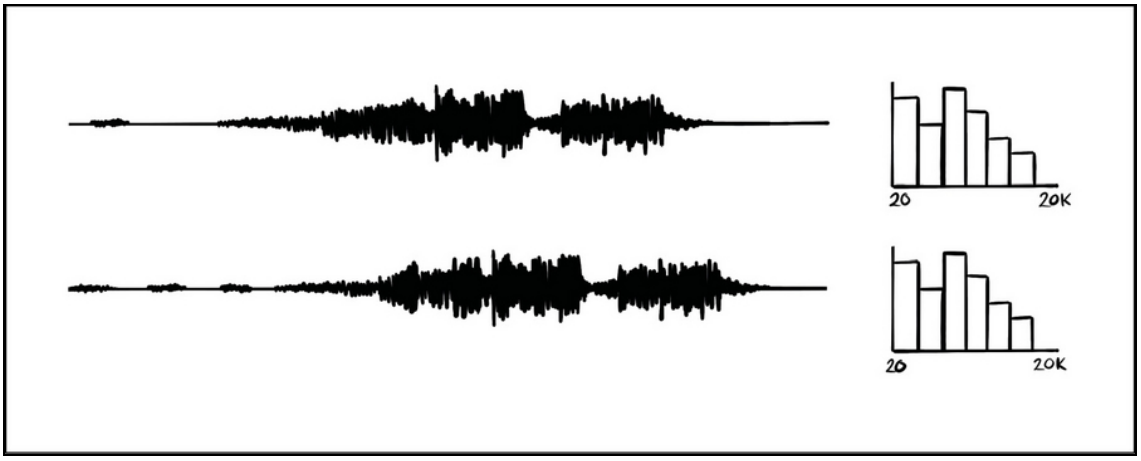


Figura 3.1 Confronto di due clip audio utilizzando la distribuzione di frequenze.

Le euristiche sono specifiche del contesto, e una buona euristica può aiutare a ottimizzare sensibilmente le soluzioni. Lo scenario del labirinto del Capitolo 2 verrà adattato per dimostrare il concetto di creazione di euristiche, introducendo una dinamica interessante. Invece di trattare tutti i movimenti allo stesso modo e valutare le migliori soluzioni esclusivamente in base ai percorsi con meno azioni (quindi con una profondità ridotta nell'albero), ai movimenti in direzioni differenti ora possiamo associare costi differenti. C'è stata come una strana alterazione della forza di gravità nel nostro labirinto, e spostarsi a nord o a sud ora costa cinque volte di più che spostarsi a est o a ovest (Figura 3.2).

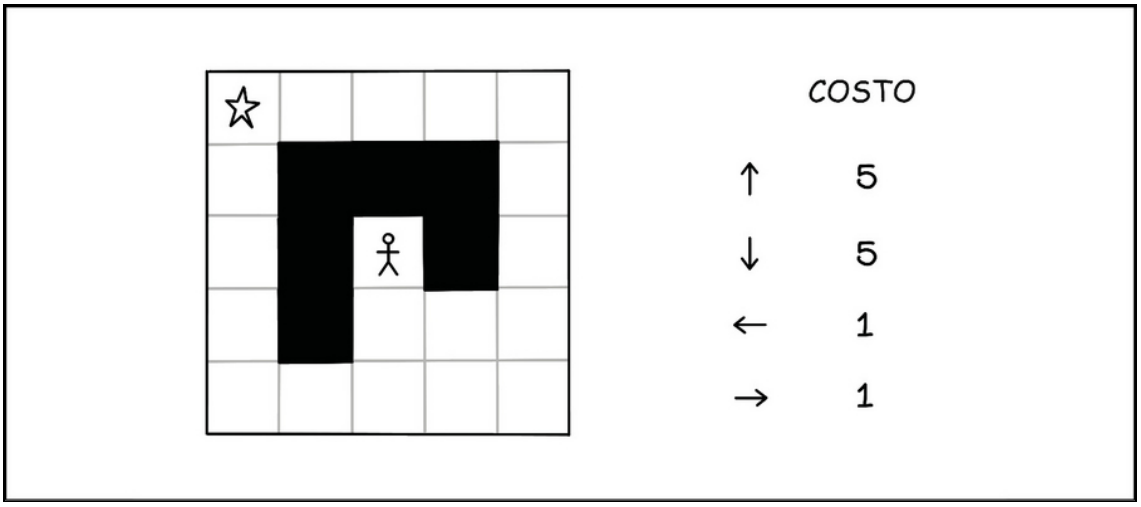


Figura 3.2 Alterazione della “gravità” nell’esempio del labirinto.

Nel nuovo scenario del labirinto, i fattori che influenzano il miglior percorso possibile verso l’obiettivo sono il numero di azioni intraprese e la somma dei costi di ciascuna azione di tale percorso.

Nella Figura 3.3 tutti i possibili percorsi sono rappresentati in un albero, per evidenziare le opzioni disponibili, indicando i costi delle rispettive azioni. Ancora una volta, questo esempio riguarda solo lo spazio di ricerca di questo specifico labirinto e non è applicabile a scenari della vita reale. L’algoritmo genererà l’albero come parte della sua ricerca.

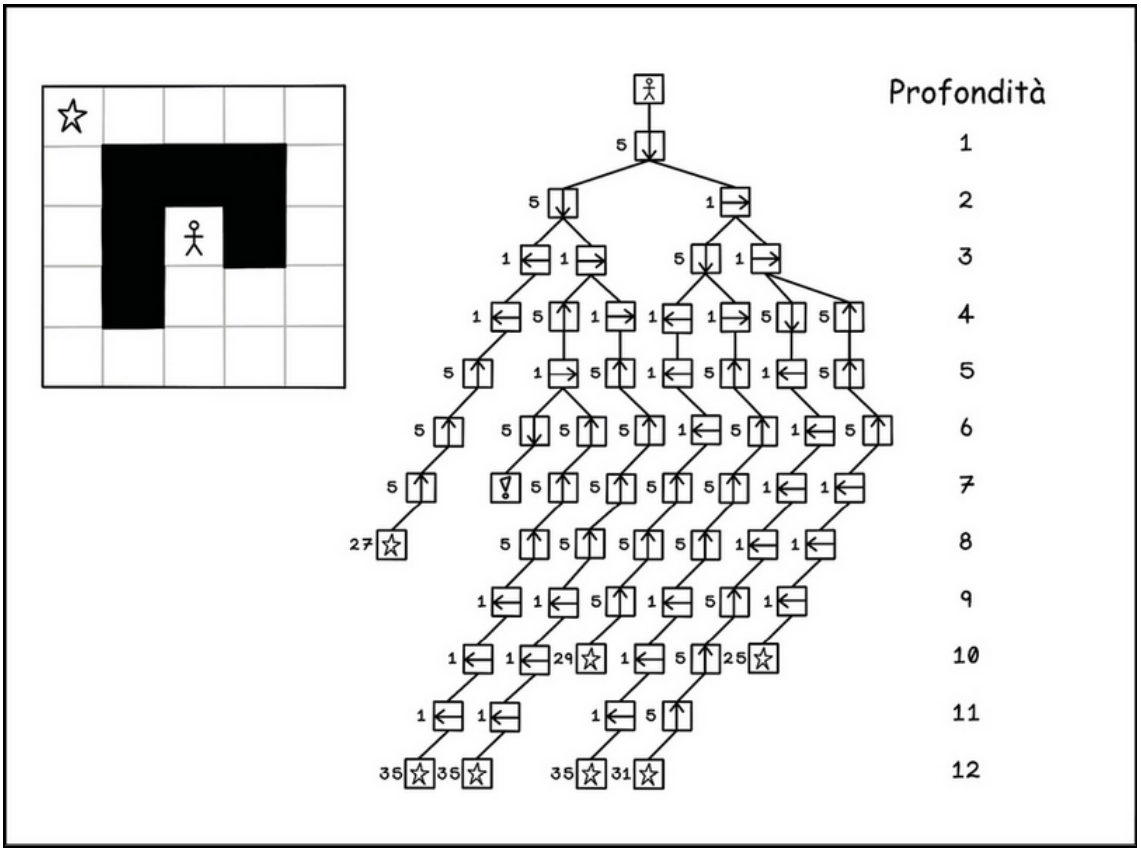


Figura 3.3 Tutte le possibili opzioni di movimento rappresentate come un albero.

Un'euristica per il problema del labirinto può essere definita come segue: "I buoni percorsi minimizzano il costo del movimento e minimizzano le mosse totali per raggiungere l'obiettivo". Questa semplice euristica aiuta a determinare quali nodi vengono visitati, perché stiamo applicando alcune conoscenze del dominio per risolvere il problema.

Esperimento mentale: dato il seguente scenario, quale euristica potete immaginare?

Minatori differenti sono specializzati in tipi di estrazione differenti: diamanti, oro, platino… Tutti i minatori sono produttivi in qualsiasi miniera, ma sono più efficaci nelle miniere in cui sono specializzati. In un'area vi sono varie miniere di diamanti, oro e platino, e i magazzini si trovano a distanze differenti fra le miniere. Se il problema è quello di distribuire i minatori per massimizzare la

loro efficienza e ridurre i tempi di spostamento, quale potrebbe essere un'euristica?

Esperimento mentale: una possibile soluzione

Un'euristica sensata includerebbe l'assegnazione di ogni minatore a una miniera in cui è specializzato e l'incarico di recarsi al magazzino più vicino a quella miniera. Ciò può anche essere interpretato come minimizzare le assegnazioni di minatori a miniere in cui non sono specializzati e minimizzare anche la distanza da percorrere per arrivare ai magazzini.

Ricerca informata: ricerca di soluzioni con una guida

La *ricerca informata*, o *ricerca euristica*, è un algoritmo che utilizza entrambi gli approcci di ricerca (in ampiezza e in profondità) combinati con una certa intelligenza. La ricerca è guidata dall'euristica, data una certa conoscenza del problema in questione.

Possiamo impiegare diversi algoritmi di ricerca informata, a seconda della natura del problema, fra i quali la ricerca Greedy (nota anche come Best-first). L'algoritmo di ricerca informata più utilizzato e utile, tuttavia, è A*.

La ricerca A*

L'algoritmo di *ricerca A** (pronunciato "A star") di solito migliora le prestazioni stimando l'euristica per minimizzare il costo del prossimo nodo visitato.

Il costo totale viene calcolato con due metriche: la distanza totale dal nodo iniziale al nodo corrente e il costo stimato del passaggio a un determinato nodo utilizzando un'euristica. Quando l'obiettivo è quello di ridurre al minimo i costi, un valore inferiore indica una soluzione che offre prestazioni migliori (Figura 3.4).

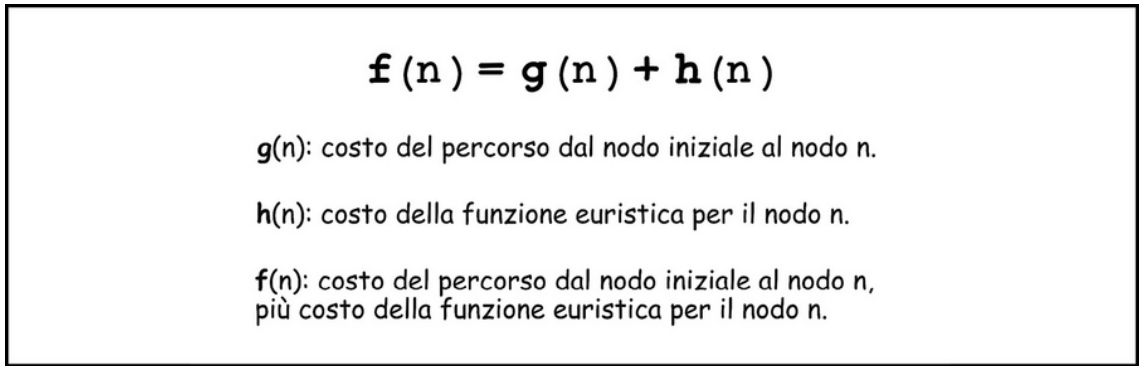


Figura 3.4 La funzione per l'algoritmo di ricerca A*.

Il seguente schema di elaborazione è un esempio astratto di come un albero venga visitato utilizzando l'euristica per guidare la nostra ricerca. L'attenzione si concentra sui calcoli euristici per i diversi nodi dell'albero.

La ricerca in ampiezza visita tutti i nodi a un certo livello di profondità, prima di passare alla profondità successiva. La ricerca in profondità visita tutti i nodi fino alla profondità massima, prima di tornare alla radice e visitare il percorso successivo. Una ricerca A* è differente, in quanto non ha uno schema predefinito da seguire; i nodi vengono visitati in un ordine basato sui loro costi euristici. Notate che l'algoritmo non conosce i costi di tutti i nodi. I costi vengono calcolati a mano a mano che l'albero viene esplorato o generato, e ogni nodo visitato viene aggiunto a uno stack. Questo ci permette di ignorare i nodi che costano più dei nodi già visitati, risparmiando tempo di calcolo (Figure 3.5, 3.6 e 3.7).

Esaminiamo il flusso dell'algoritmo di ricerca A*.

- *Aggiungi il nodo radice allo stack*. L'algoritmo di ricerca A* può essere implementato con uno stack in cui per primo viene elaborato l'ultimo oggetto aggiunto (Last-In, First-Out o LIFO). Il primo passaggio consiste nell'aggiungere allo stack il nodo radice.

- *Lo stack è vuoto?* Se lo stack è vuoto e non è stato restituito alcun percorso nel Passaggio 8 dell'algoritmo, non esiste alcun percorso verso l'obiettivo. Se ci sono ancora nodi nello stack, l'algoritmo può continuare la sua ricerca.
- *Restituisci "Nessun percorso verso l'obiettivo"*. Questo passaggio è l'unica possibile uscita dall'algoritmo se non esiste alcun percorso verso l'obiettivo.
- *Estrai il nodo dallo stack come nodo corrente*. Estraendo l'oggetto successivo dallo stack e impostandolo come nodo corrente, possiamo esplorarne le possibilità.
- *Il nodo corrente è già stato visitato?* Se il nodo corrente non è stato visitato, non è stato ancora esplorato e può essere elaborato ora.
- *Contrassegna il nodo corrente come visitato*. Questo passaggio indica che questo nodo è stato visitato, per impedire un'elaborazione ripetuta non necessaria.
- *Obiettivo raggiunto?* Questo passaggio determina se il fratello corrente contiene l'obiettivo che l'algoritmo sta cercando.
- *Restituisci il percorso utilizzando il nodo corrente*. Facendo riferimento al genitore del nodo corrente, quindi al genitore di quel nodo e così via, viene descritto il percorso dall'obiettivo alla radice. Il nodo radice sarà un nodo senza genitore.

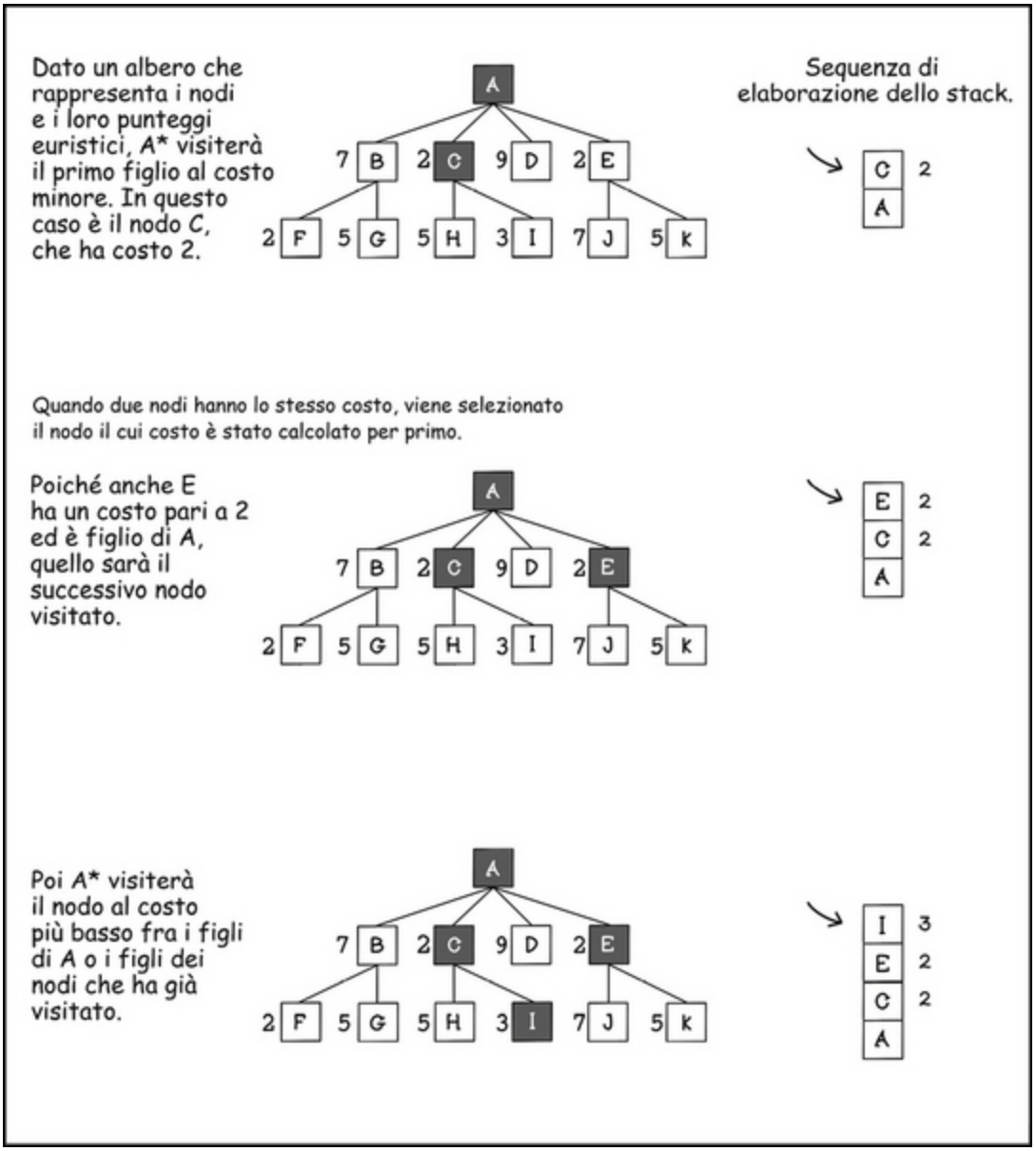


Figura 3.5 La sequenza di elaborazione dell'albero utilizzando la ricerca A* (Parte 1).

- *Il nodo corrente ha un altro fratello?* Se il nodo corrente ha più mosse possibili da fare nell'esempio del labirinto, quella mossa può essere aggiunta per essere elaborata. In caso contrario, l'algoritmo può tornare al Passaggio 2, in cui può essere elaborato l'oggetto successivo nello stack, se non è vuoto. La natura dello

stack LIFO consente all'algoritmo di elaborare tutti i nodi fino alla profondità di un nodo foglia, prima di tornare indietro per visitare altri figli del nodo radice.

- *Ordina lo stack per costo crescente*. Quando lo stack viene ordinato in base al costo di ciascun nodo nello stack ascendente, viene elaborato il nodo con il costo più basso, cosa che consente di visitare sempre il nodo più economico.

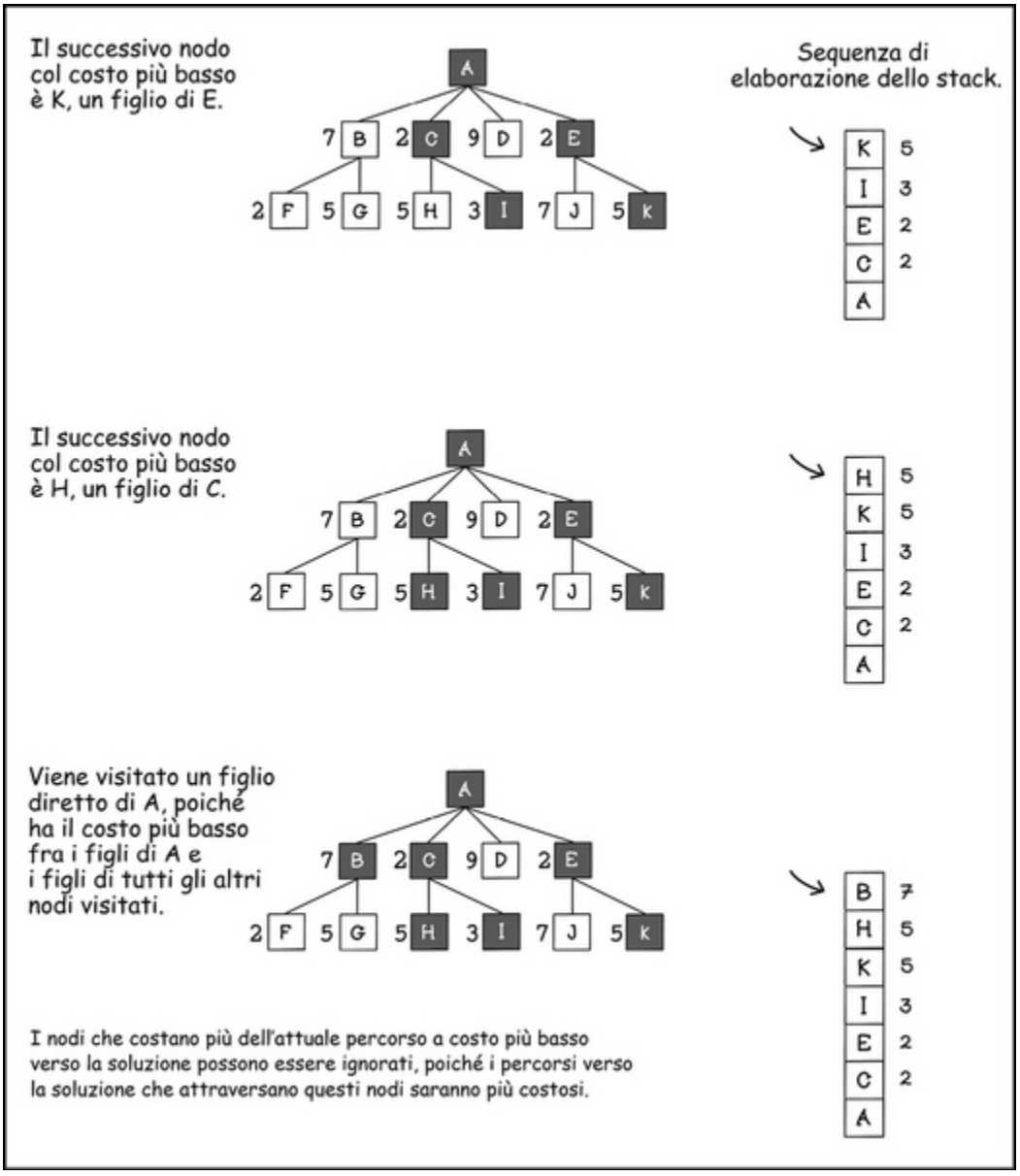


Figura 3.6 La sequenza di elaborazione dell'albero utilizzando la ricerca A* (Parte 2).

- *Imposta il nodo corrente come genitore del fratello*. Imposta il nodo di origine come genitore del fratello corrente. Questo passaggio è importante per tracciare il percorso dal fratello corrente al nodo radice. Dal punto di vista della mappa, l'origine è

la posizione da cui si è spostato il giocatore e il fratello corrente è la posizione in cui si è spostato il giocatore.

- *Calcola il costo per il fratello*. La funzione di costo è fondamentale per guidare l'algoritmo A*. Il costo viene calcolato sommando alla distanza dal nodo radice il punteggio euristico della mossa successiva. Un'euristica più intelligente influenzerà direttamente l'algoritmo A*, garantendogli prestazioni migliori.

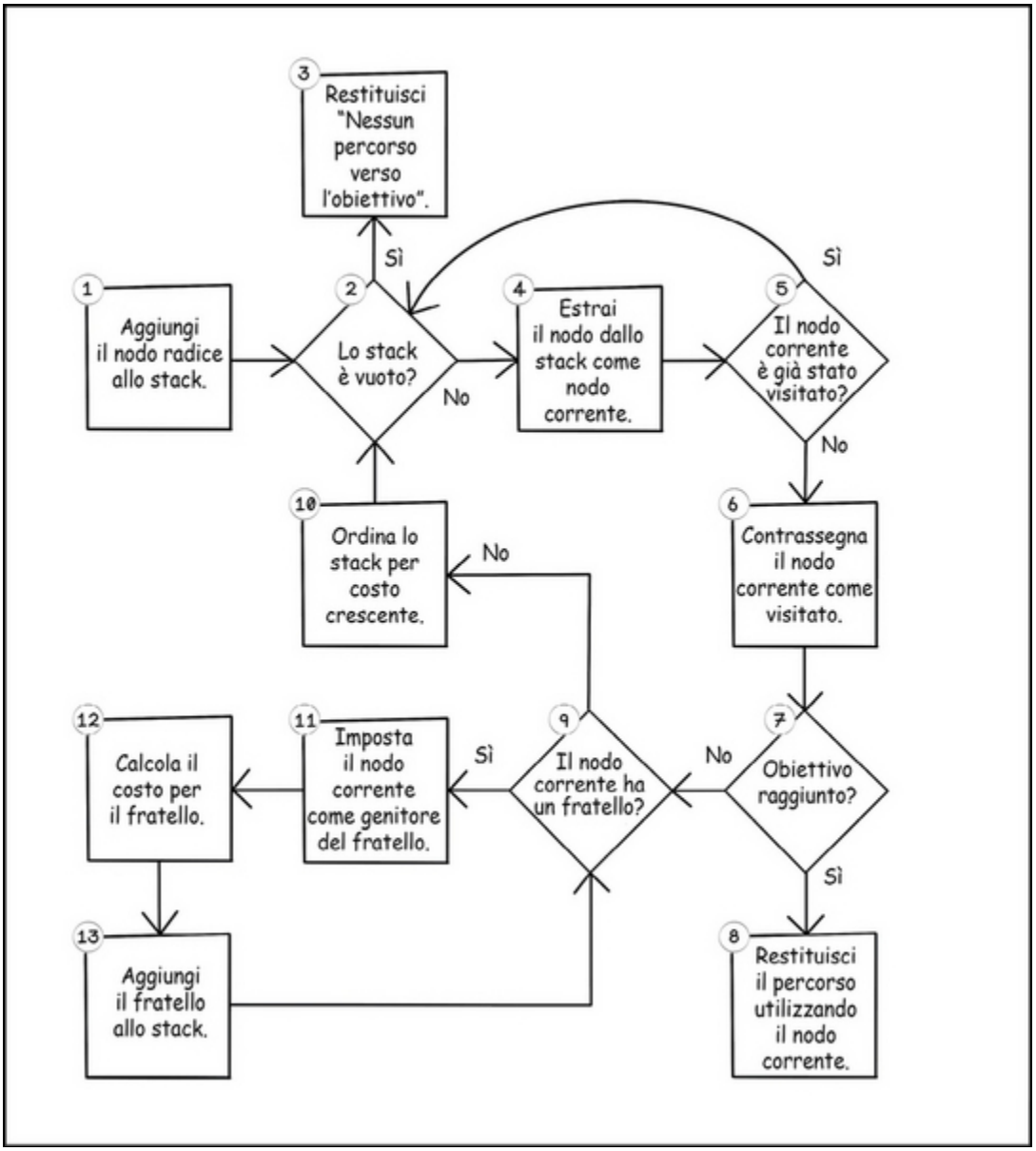


Figura 3.7 Flusso dell'algoritmo di ricerca A*.

- *Aggiungi il fratello allo stack*. Il nodo adiacente viene aggiunto allo stack affinché i suoi figli possano essere esplorati. Ancora una volta, questo meccanismo di funzionamento dello stack consente di elaborare i nodi alla massima profondità prima di elaborare i vicini a profondità minori.

Simile alla ricerca in profondità, l'ordine dei nodi figli influenza il percorso selezionato, ma in modo meno marcato. Se due nodi hanno lo stesso costo, viene visitato il primo nodo calcolato (Figure 3.8, 3.9 e 3.10).

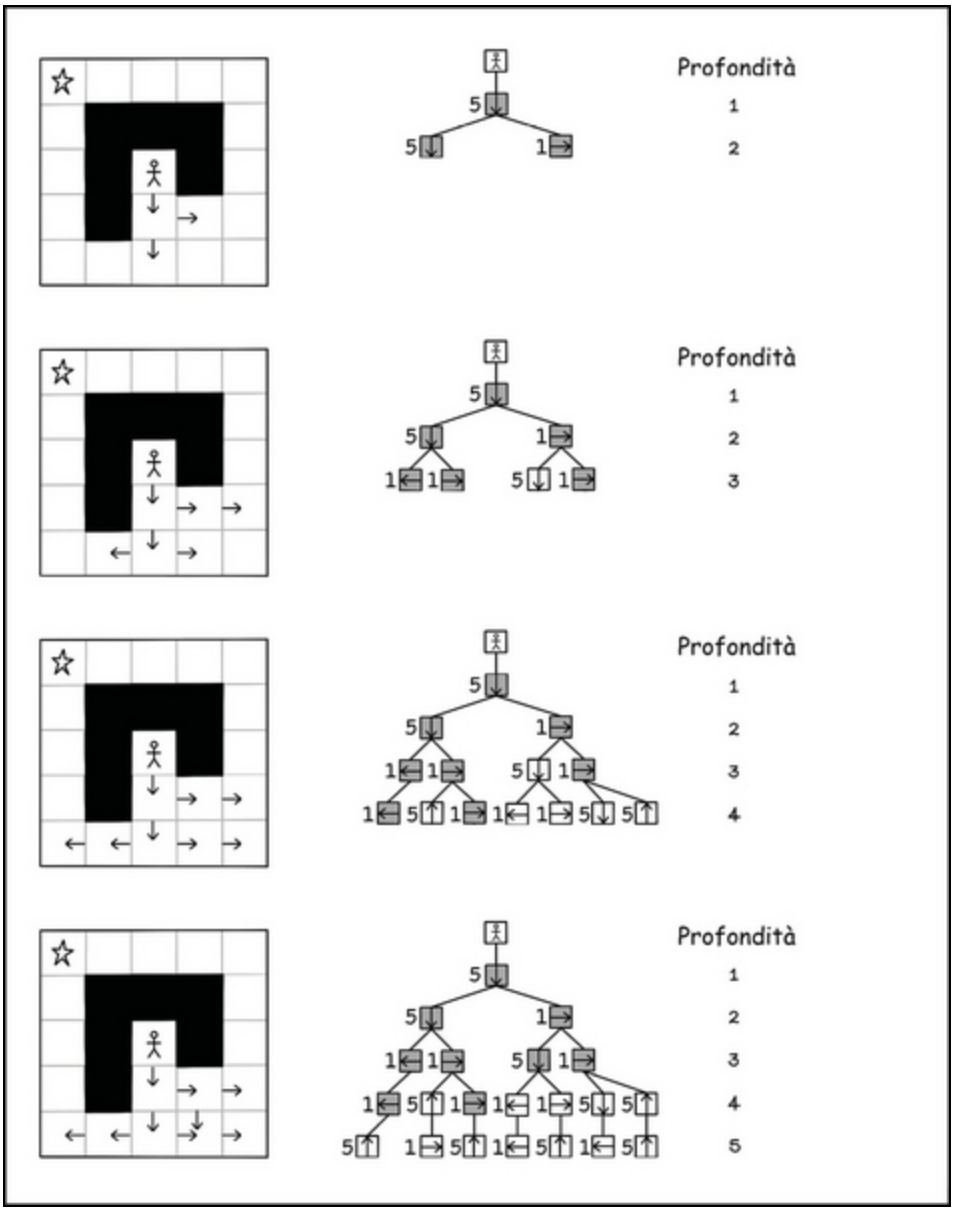


Figura 3.8 La sequenza di elaborazione dell'albero utilizzando la ricerca A* (Parte 1).

Notate che ci sono diversi percorsi verso l'obiettivo, ma l'algoritmo A* trova un percorso riducendo al minimo il costo per raggiungerlo, con meno mosse e costi di spostamento più economici, dato il fatto che le mosse a nord e a sud sono più costose.

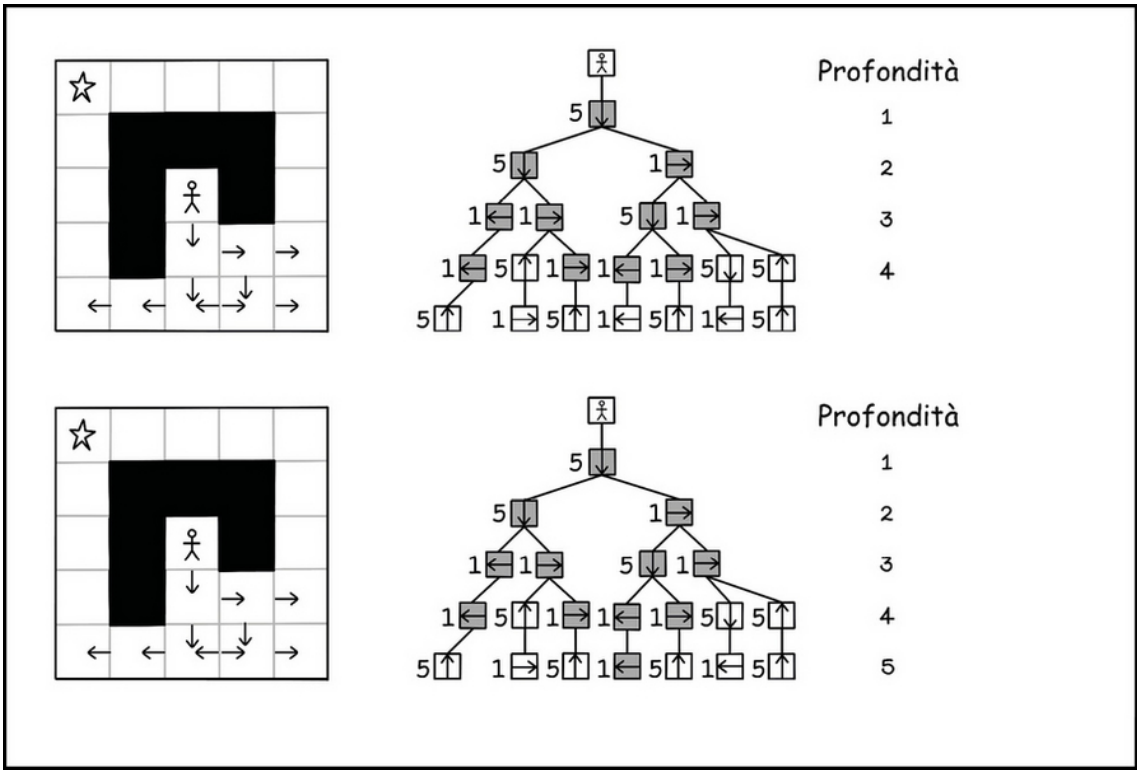


Figura 3.9 La sequenza di elaborazione dell'albero utilizzando la ricerca A* (Parte 2).

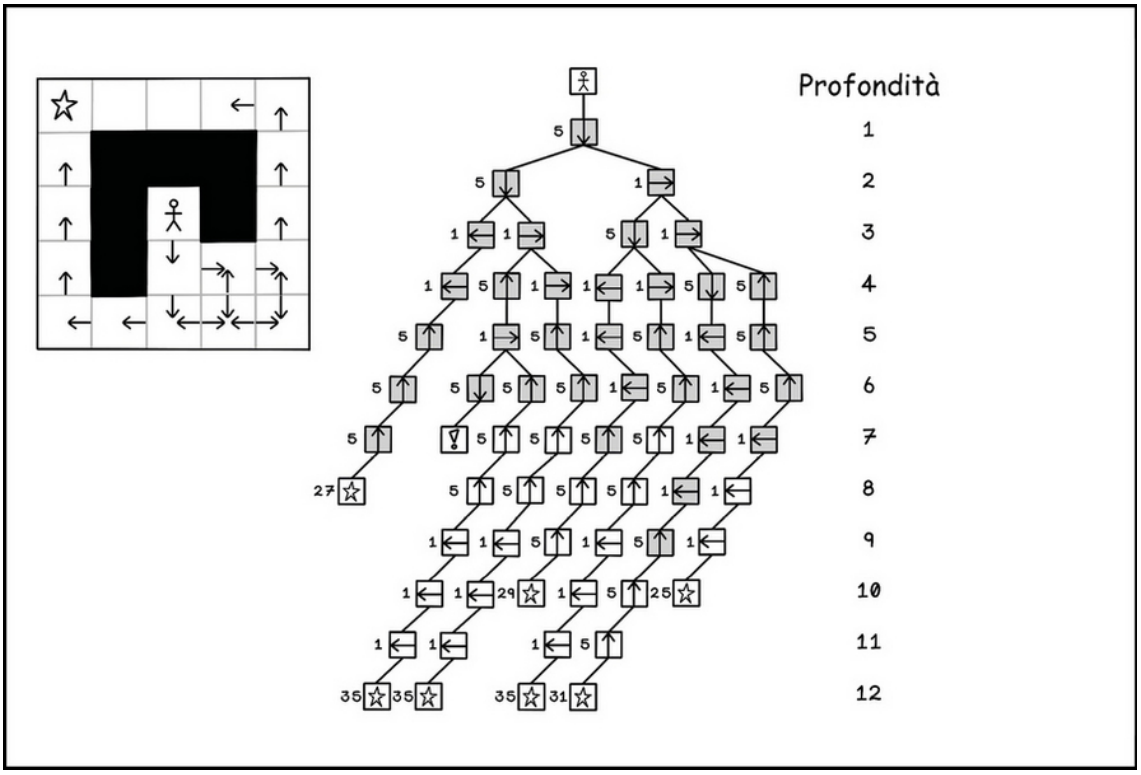


Figura 3.10 Nodi visitati nell'intero albero dopo la ricerca A*.

Pseudocodice

L'algoritmo A* utilizza un approccio simile all'algoritmo di ricerca in profondità, ma privilegia i nodi più economici da visitare. Per elaborare i nodi viene utilizzato uno stack, che tuttavia viene ordinato per costo crescente ogni volta che viene eseguito un nuovo calcolo. Questo ordine garantisce che l'oggetto estratto dallo stack sia sempre il più economico, in quanto dopo l'ordinamento è il primo nello stack:

```
run_astar(maze, root_point, visited_points):
  let s equal a new stack
  add root_point to s
  while s is not empty
      pop s and let current_point equal the returned point
      if current_point is not visited:
          mark current_point as visited
      if value at current_node is the goal:
          return path using current_point
      else:
              add available cells north, east, south, and west to a list
neighbors
          for each neighbor in neighbors:
              set neighbor parent as current_point
```

```
            set neighbor cost as calculate_cost(current_point, neighbor)
            push neighbor to s
        sort s by cost ascending
  return "No path to goal"
```

Le funzioni per il calcolo del costo sono fondamentali per il funzionamento della ricerca A*. La funzione di costo fornisce all'algoritmo le informazioni necessarie per cercare il percorso più economico. Nel nostro esempio di labirinto, un costo più elevato è associato a uno spostamento verso l'alto o verso il basso. Se la funzione di costo presenta un problema, l'algoritmo potrebbe non funzionare.

Le due funzioni seguenti descrivono come viene calcolato il costo. La distanza dal nodo radice viene aggiunta al costo del movimento successivo. Sulla base del nostro esempio ipotetico, il costo dello spostamento verso nord o verso sud influenza il costo totale della visita di quel nodo:

```
calculate_cost(origin, target):
  let distance_to_root equal length of path from origin to target
  let cost_to:move equal get_ move _cost (origin, target)
  return distance_to_root + cost_to_move
move_cost(origin,target):
  if target is north or south of origin:
      return 5
  else:
      return 1
```

Gli algoritmi di ricerca non informati, come la ricerca in ampiezza e in profondità, esplorano ogni possibilità in modo esaustivo e portano alla soluzione ottimale. La ricerca A* è un buon approccio quando è possibile creare un'euristica ragionevole per guidare la ricerca. Opera in modo più efficiente rispetto agli algoritmi di ricerca non informati, perché ignora i nodi che costano più dei nodi già visitati. Se però l'euristica è errata o non ha senso per il problema e il contesto, A* troverà soluzioni scadenti invece di quelle ottimali.

Casi d'uso per gli algoritmi di ricerca informata

Gli algoritmi di ricerca informata sono versatili e utili per diversi casi d'uso nei quali sia possibile definire un'euristica, come i seguenti.

- *Ricerca di percorsi per personaggi di gioco autonomi nei videogiochi*: gli sviluppatori di giochi utilizzano spesso questo algoritmo per controllare il movimento delle unità nemiche in

quei giochi nei quali l'obiettivo è quello di trovare il giocatore umano all'interno di un ambiente.

- *Analisi dei paragrafi nell'elaborazione del linguaggio naturale*: il significato di un paragrafo può essere suddiviso in una serie di frasi, che possono essere suddivise in una serie di parole di diverso tipo (nomi e verbi, per esempio), creando una struttura ad albero che può essere valutata. La ricerca informata può essere utile per estrarne il significato.
- *Routing in una rete di telecomunicazioni*: è possibile utilizzare algoritmi di ricerca informata per trovare i percorsi più brevi per il traffico di rete nelle reti di telecomunicazioni, in modo da migliorarne le prestazioni. Server/nodi di rete e connessioni possono essere rappresentati come grafi ricercabili di nodi e archi.
- *Giochi e puzzle a giocatore singolo*: gli algoritmi di ricerca informata possono essere utilizzati per risolvere giochi e puzzle a giocatore singolo, come il cubo di Rubik, perché ogni mossa è una decisione in un albero di possibilità, fino a quando non viene trovato lo stato che rappresenta l'obiettivo.

Ricerca avversaria: cercare soluzioni in un ambiente in evoluzione

L'esempio della ricerca nel gioco del labirinto coinvolge un solo attore: il giocatore. L'ambiente è influenzato solo dal giocatore singolo, quindi è solo quel giocatore a generare tutte le possibilità. L'obiettivo finora era quello di massimizzare il vantaggio per il solo giocatore: scegliere percorsi verso un obiettivo con la distanza e il costo minori.

La *ricerca avversaria* è caratterizzata dall'esistenza di un'opposizione o un conflitto. I problemi ad avversari ci richiedono di anticipare, comprendere e contrastare le azioni dell'avversario nel

perseguimento di un obiettivo. Esempi di problemi ad avversari includono giochi a turni per due giocatori come Tris (*Tic-Tac-Toe*) e Forza quattro (*Connect Four*). I giocatori, a turno, hanno l'opportunità di cambiare lo stato dell'ambiente di gioco a loro favore. Un insieme di regole determina come l'ambiente può essere modificato e quali sono gli stati vincenti e finali.

Un semplice problema ad avversari

Questo paragrafo utilizza il gioco Forza quattro per esplorare i problemi ad avversari. Forza quattro (Figura 3.11) è un gioco costituito da una griglia in cui i giocatori, a turno, rilasciano gettoni in una colonna. I gettoni si accumulano e vince il giocatore che riesca a creare sequenze di quattro gettoni adiacenti (in verticale, orizzontale o diagonale) del suo colore. Se la griglia è piena, senza vincitori, la partita termina in parità.

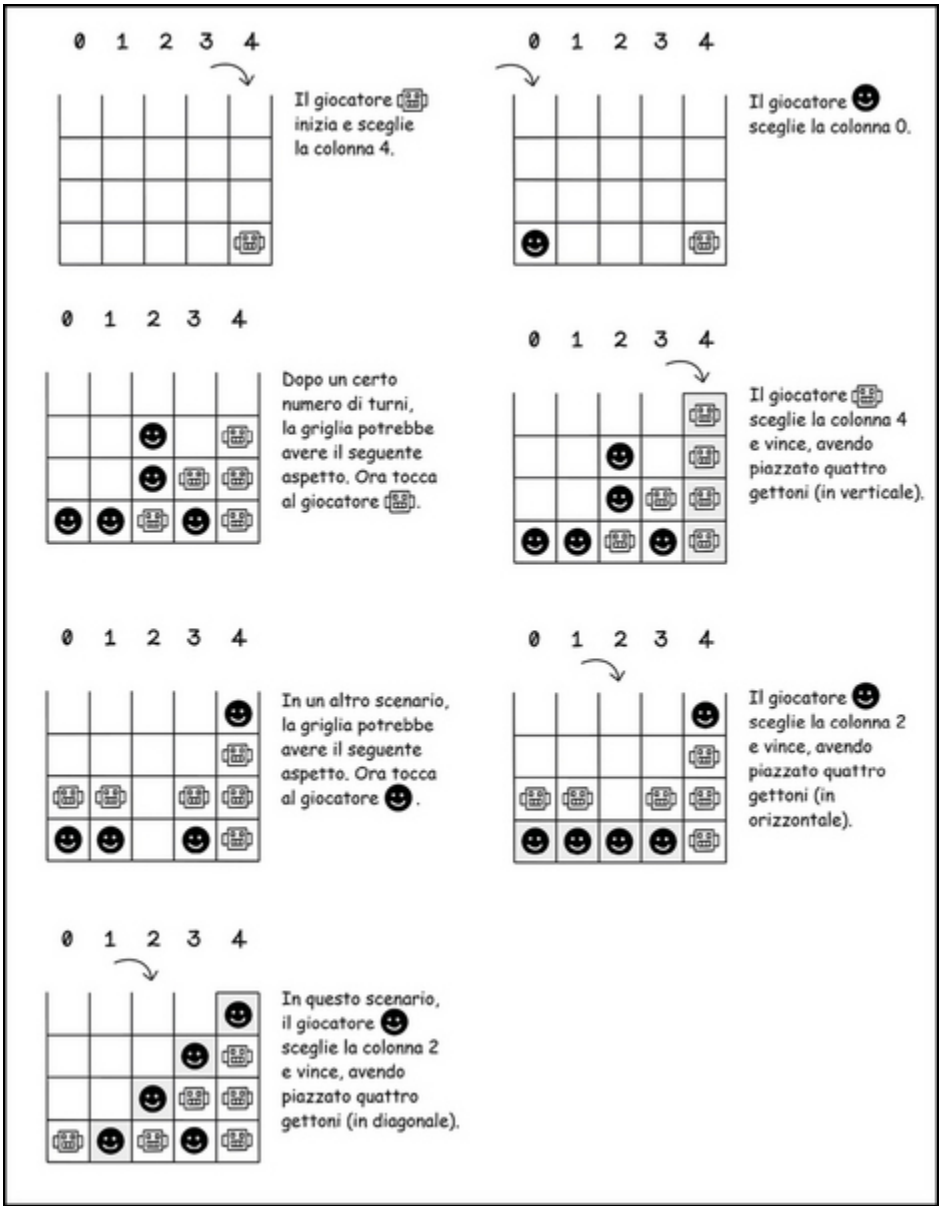


Figura 3.11 Il gioco Forza quattro.

Ricerca min-max: simula le azioni e sceglie il futuro migliore

La *ricerca min-max* ha lo scopo di costruire un albero dei risultati possibili in base alle mosse che ogni giocatore potrebbe fare, e favorire percorsi vantaggiosi per un giocatore evitando i percorsi favorevoli

all'avversario. Per fare ciò, questo tipo di ricerca simula le possibili mosse e assegna un punteggio a ogni stato in base a un'euristica dopo aver effettuato la rispettiva mossa. La ricerca min-max tenta di scoprire quanti più stati possibili in futuro; ma a causa dei limiti di memoria e calcolo, scoprire l'intero albero del gioco potrebbe non essere possibile, quindi limita la ricerca a una determinata profondità. La ricerca min-max simula i turni di ciascun giocatore, quindi la profondità specificata è direttamente legata al numero di turni fra i due giocatori. Una profondità di 4, per esempio, prevede due turni per ogni giocatore. Il giocatore A fa una mossa, il giocatore B fa una mossa, il giocatore A fa un'altra mossa e il giocatore B fa un'altra mossa.

Euristica

L'algoritmo min-max utilizza un punteggio euristico per prendere le sue decisioni. Questo punteggio è definito da una determinata euristica e non è appreso dall'algoritmo. Dato un determinato stato di gioco, ogni possibile risultato valido di una mossa a partire da quello stato sarà un nodo figlio dell'albero del gioco.

Supponiamo di avere un'euristica che fornisce un punteggio in cui i numeri positivi sono migliori dei numeri negativi. Simulando ogni possibile mossa valida, l'algoritmo di ricerca min-max cerca di minimizzare le mosse in cui l'avversario avrà un vantaggio o uno stato vincente e di massimizzare le mosse che danno all'agente un vantaggio o uno stato vincente.

La Figura 3.12 illustra un albero di ricerca min-max. In questa figura, i nodi foglia sono gli unici nodi nei quali viene calcolato il punteggio euristico, poiché questi stati indicano un vincitore o un pareggio. Gli altri nodi dell'albero indicano gli stati in corso. A partire dalla profondità in cui viene calcolata l'euristica e procedendo verso l'alto, viene scelto il nodo figlio che offre il punteggio minimo o il

nodo figlio con il punteggio massimo, a seconda di chi muoverà al prossimo turno. Partendo dall'alto, l'algoritmo tenta di massimizzare il punteggio dell'agente; a turni alterni l'intenzione cambia, perché l'obiettivo è quello di massimizzare il punteggio per l'agente e minimizzare il punteggio per l'avversario.

Poiché l'algoritmo di ricerca min-max simula i risultati possibili, nei giochi che offrono più scelte, l'albero del gioco esplode, e così diventa rapidamente troppo costoso, dal punto di vista computazionale, esplorare l'intero albero. Nel semplice esempio di Forza quattro giocato su un tabellone 5×4, il numero di possibilità rende già inefficiente l'esplorazione di ogni turno dell'intero albero di gioco (Figura 3.13).

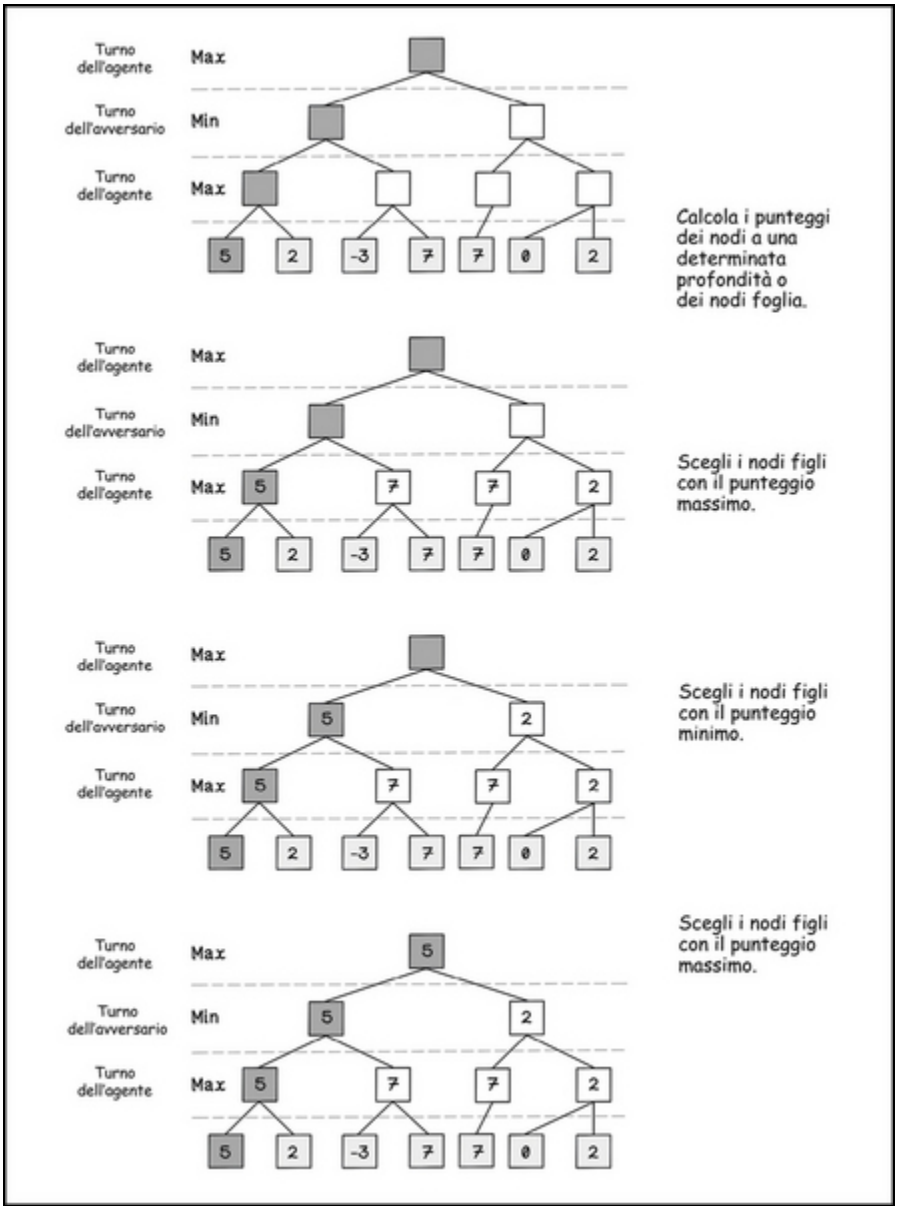
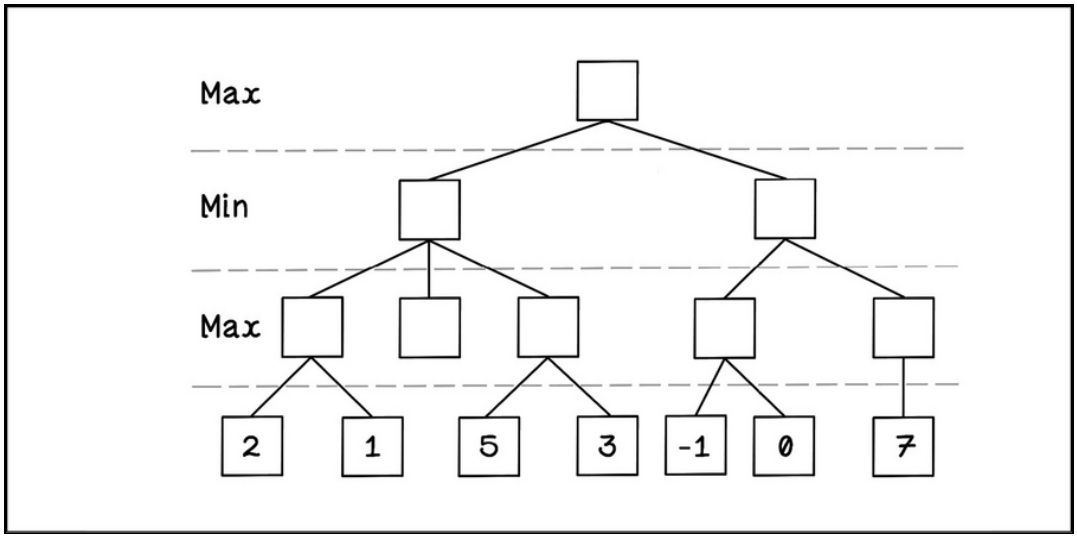


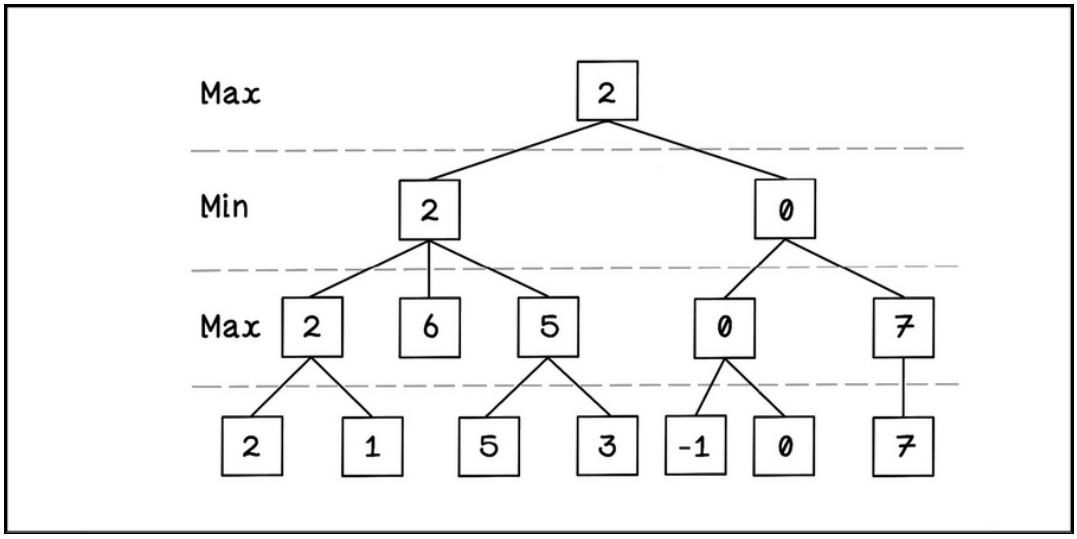
Figura 3.12 La sequenza di elaborazione dell'albero usando la ricerca min-max.

Esercizio: quali valori si propagherebbero nel caso del seguente albero min-

max?



Soluzione



Per utilizzare la ricerca min-max nell'esempio Forza quattro, l'algoritmo valuta essenzialmente tutte le mosse possibili da uno stato di gioco corrente; quindi determina tutte le possibili mosse da ciascuno di questi stati, finché non trova il percorso più favorevole. Gli stati del gioco che si traducono in una vittoria per l'agente restituiscono un punteggio 10 e gli stati che si traducono in una vittoria per l'avversario

restituiscono un punteggio -10. La ricerca min-max cerca di massimizzare il punteggio positivo per l'agente (Figure 3.14 e 3.15).

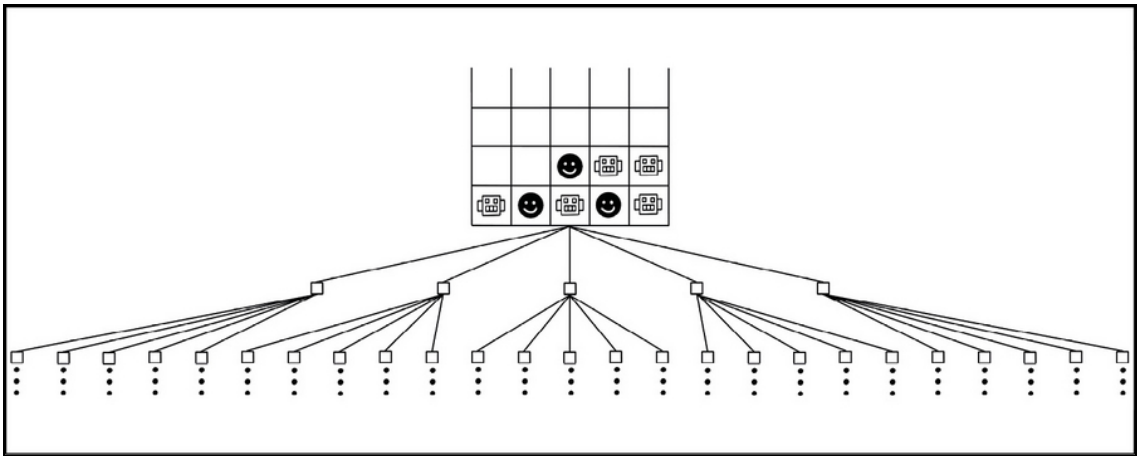


Figura 3.13 L'esplosione di possibilità nella ricerca nell'albero del gioco Forza quattro.

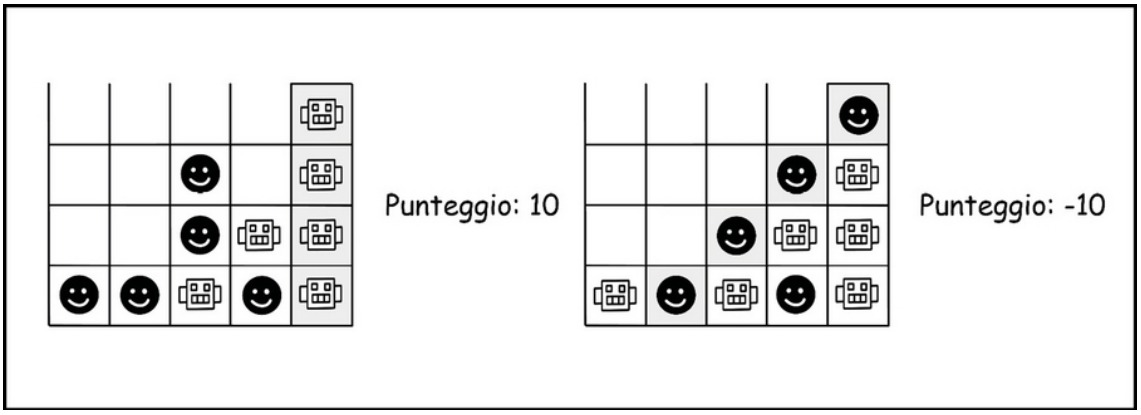


Figura 3.14 Punteggio per l'agente contro punteggio per l'avversario.

Sebbene il diagramma di flusso per l'algoritmo di ricerca min-max sembri complesso a causa delle sue dimensioni, in realtà non lo è. Quello che "gonfia" in grafico è il numero di condizioni che controllano se lo stato corrente deve essere massimizzato o minimizzato.

Esaminiamo il flusso dell'algoritmo di ricerca min-max.

- *Dato uno stato di gioco, indipendentemente dal fatto che la modalità corrente sia di minimizzazione o di massimizzazione e data una profondità corrente, l'algoritmo può iniziare*. È importante comprendere gli input dell'algoritmo, poiché l'algoritmo di ricerca min-max è ricorsivo. Un algoritmo ricorsivo richiama se stesso in uno o più passaggi. È importante che un algoritmo ricorsivo abbia una condizione di uscita per impedirgli di richiamare se stesso indefinitamente.
- *Lo stato attuale è finale o ha profondità 0?* Questa condizione determina se lo stato attuale del gioco è uno stato terminale o se è stata raggiunta la profondità desiderata. Uno stato è terminale se uno dei giocatori ha vinto o la partita è patta. Il punteggio 10 rappresenta una vittoria per l'agente, il punteggio -10 rappresenta una vittoria per l'avversario e il punteggio 0 indica un pareggio. Viene specificata una profondità, perché l'attraversamento dell'intero albero delle possibilità a coprire tutti gli stati finali è computazionalmente infattibile e richiederà troppo tempo su un normale computer. Specificando una profondità, l'algoritmo ha la possibilità di guardare al futuro per determinare se esiste uno stato terminale.

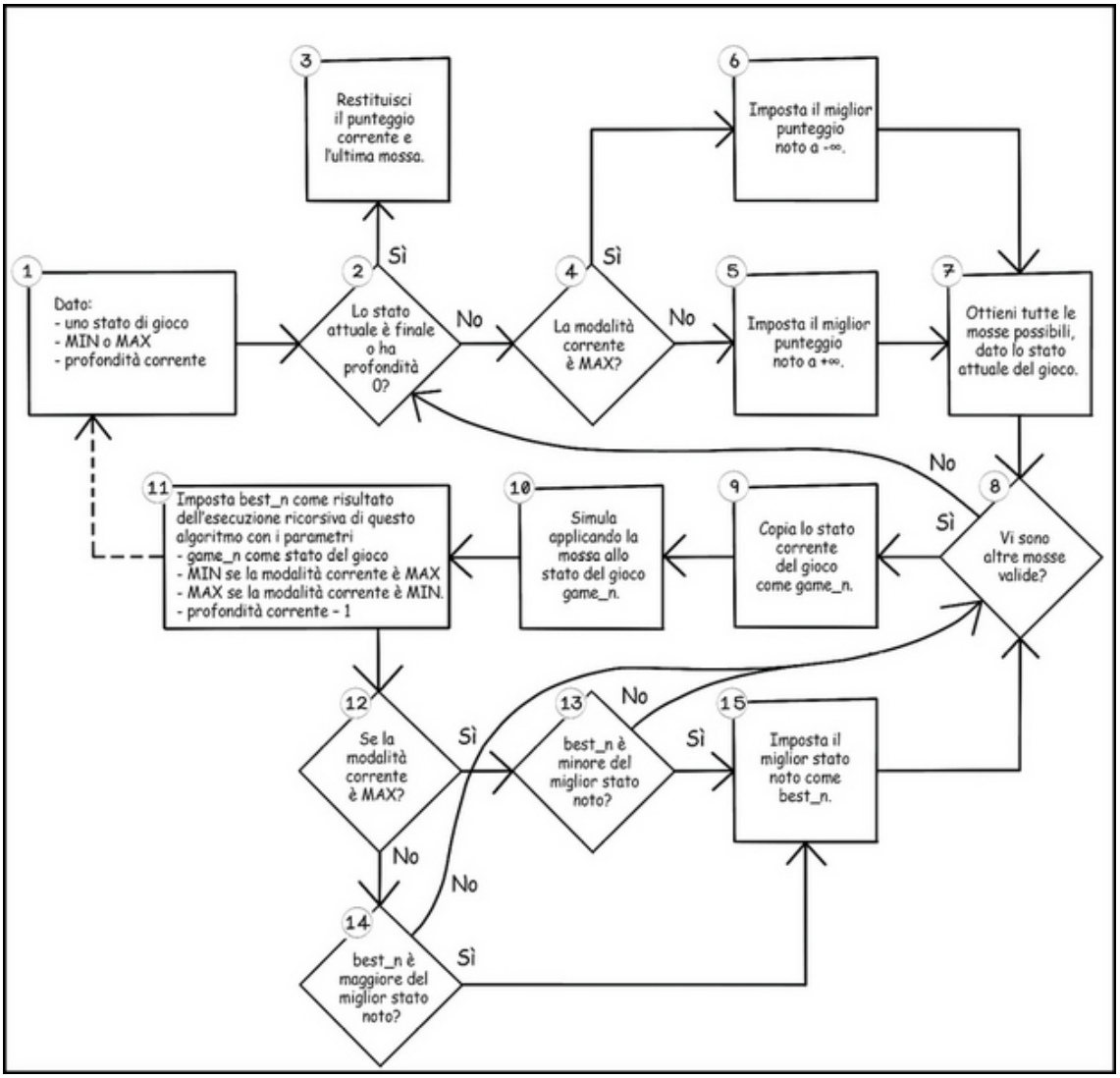


Figura 3.15 Diagramma di flusso dell'algoritmo di ricerca min-max.

- *Restituisci il punteggio corrente e l'ultima mossa*. Il punteggio per lo stato corrente viene restituito se lo stato corrente è uno stato di gioco terminale o se è stata raggiunta la profondità specificata.
- *La modalità corrente è MAX?* Se l'iterazione corrente dell'algoritmo è nello stato di massimizzazione, tenta di massimizzare il punteggio per l'agente.
- *Imposta il miglior punteggio noto a* $+\infty$. Se la modalità corrente è quella di minimizzare il punteggio, il punteggio migliore è

impostato a infinito positivo, perché sappiamo che i punteggi restituiti dagli stati del gioco saranno comunque inferiori. Nell'implementazione effettiva, viene utilizzato un numero molto elevato al posto dell'infinito.

- *Imposta il miglior punteggio noto a -∞*. Se la modalità corrente è quella di massimizzare il punteggio, il punteggio migliore è impostato a infinito negativo, perché sappiamo che i punteggi restituiti dagli stati del gioco saranno sempre maggiori. Nell'implementazione effettiva, viene utilizzato un numero negativo molto elevato al posto dell'infinito.
- *Ottieni tutte le mosse possibili, dato lo stato attuale del gioco*. Questo passaggio specifica un elenco di possibili mosse che possono essere effettuate, dato lo stato attuale del gioco. A mano a mano che il gioco procede, non tutte le mosse disponibili all'inizio potrebbero essere ancora disponibili. Nell'esempio Forza quattro, una colonna può essere già tutta piena; pertanto, una mossa che selezionasse quella colonna non sarebbe valida.
- *Vi sono altre mosse valide?* Se eventuali mosse possibili non sono ancora state simulate e non ci sono altre mosse valide da eseguire, l'algoritmo restituisce la mossa migliore in quell'istanza della chiamata a funzione.
- *Copia lo stato corrente del gioco come game_n*. È necessaria una copia dello stato di gioco corrente per eseguire simulazioni delle possibili mosse future.
- *Simula applicando la mossa allo stato del gioco game_n*. Questo passaggio applica l'attuale mossa allo stato di gioco copiato.
- *Imposta best_n come risultato dell'esecuzione ricorsiva di questo algoritmo*. Ecco dove entra in gioco la ricorsione. *best_n* è una variabile utilizzata per memorizzare la prossima mossa migliore:

stiamo facendo in modo che l'algoritmo esplori le possibilità future da questa mossa.

- *Se la modalità corrente è MAX?* Quando la chiamata ricorsiva restituisce un miglior candidato, questa condizione determina se la modalità corrente è quella di massimizzare il punteggio.
- *best_n è minore del miglior stato noto?* Questo passaggio determina se l'algoritmo ha trovato un punteggio migliore di quello trovato in precedenza, se la modalità è quella di massimizzare il punteggio.
- *best_n è maggiore di best_n conosciuto?* Questo passaggio determina se l'algoritmo ha trovato un punteggio migliore di quello trovato in precedenza, se la modalità è quella di minimizzare il punteggio.
- *Imposta il migliore stato noto come best_n*. Se viene trovato il nuovo miglior punteggio, lo imposta come punteggio migliore noto.

Dato l'esempio Forza quattro in un determinato stato, l'algoritmo di ricerca min-max genera l'albero mostrato nella Figura 3.16. Dallo stato iniziale, viene esplorata ogni possibile mossa. Quindi ogni mossa da quello stato viene esplorata fino a quando non viene trovato uno stato terminale, in quanto il tabellone è pieno o un giocatore ha vinto.

I nodi evidenziati nella Figura 3.17 sono nodi di stati terminali, in cui i pareggi sono valutati a 0, le sconfitte a -10 e le vittorie a 10. Poiché l'algoritmo ha lo scopo di massimizzare il suo punteggio, è necessario un numero positivo, mentre le vittorie dell'avversario sono segnate con un numero negativo.

Quando questi punteggi sono noti, l'algoritmo min-max parte dalla profondità più bassa e sceglie il nodo il cui punteggio è il valore minimo (Figura 3.18).

Quindi, alla profondità successiva, l'algoritmo sceglie il nodo il cui punteggio è il valore massimo (Figura 3.19).

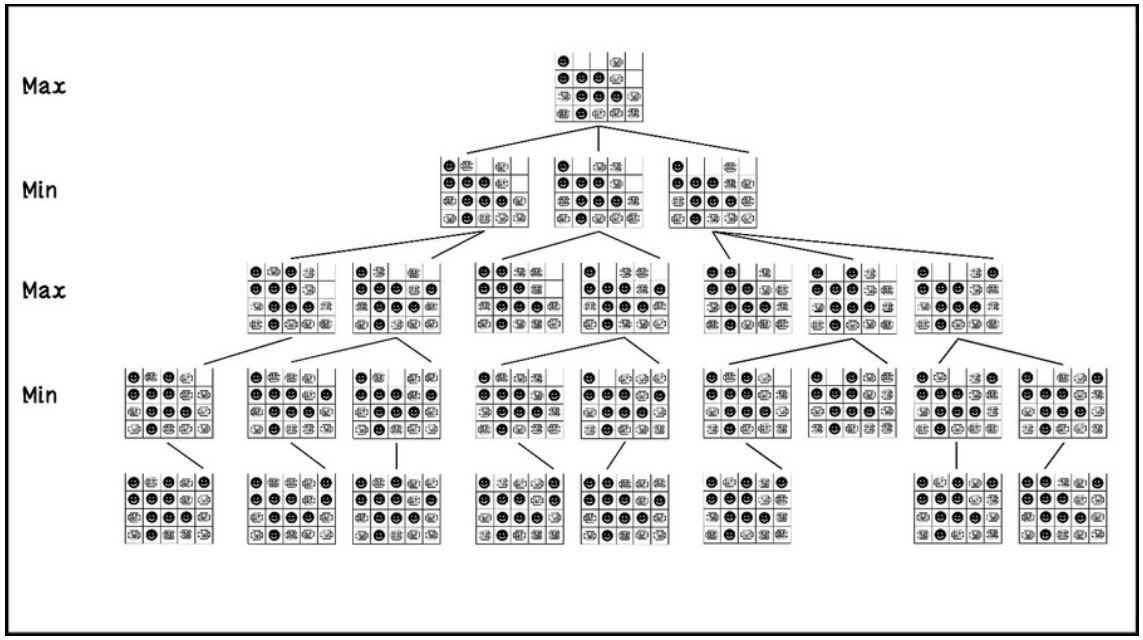


Figura 3.16 Una rappresentazione dei possibili stati in un gioco Forza quattro.

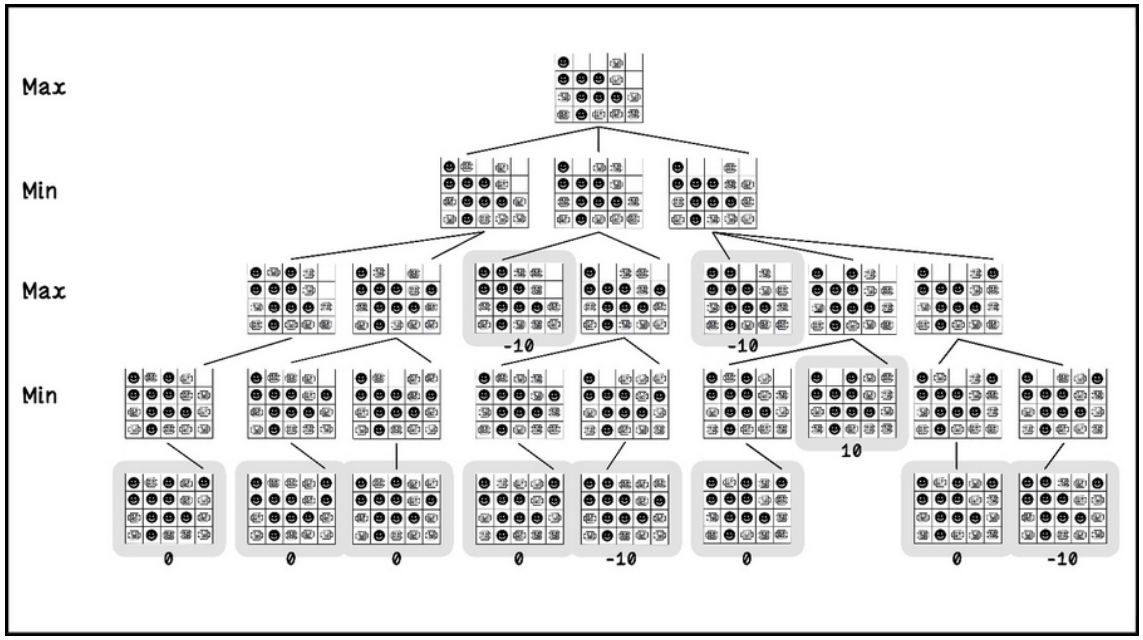


Figura 3.17 I possibili stati finali in una partita di Forza quattro.

Infine, alla profondità successiva, vengono scelti i nodi il cui punteggio è il minimo e il nodo radice sceglie l'opzione massima. Seguendo i nodi e il punteggio selezionati e applicando intuitivamente l'algoritmo al problema, vediamo che esso seleziona un percorso verso un pareggio, per evitare una sconfitta. Se l'algoritmo seleziona un percorso che conduce a una vittoria, c'è anche un'elevata probabilità di una sconfitta nel turno successivo.

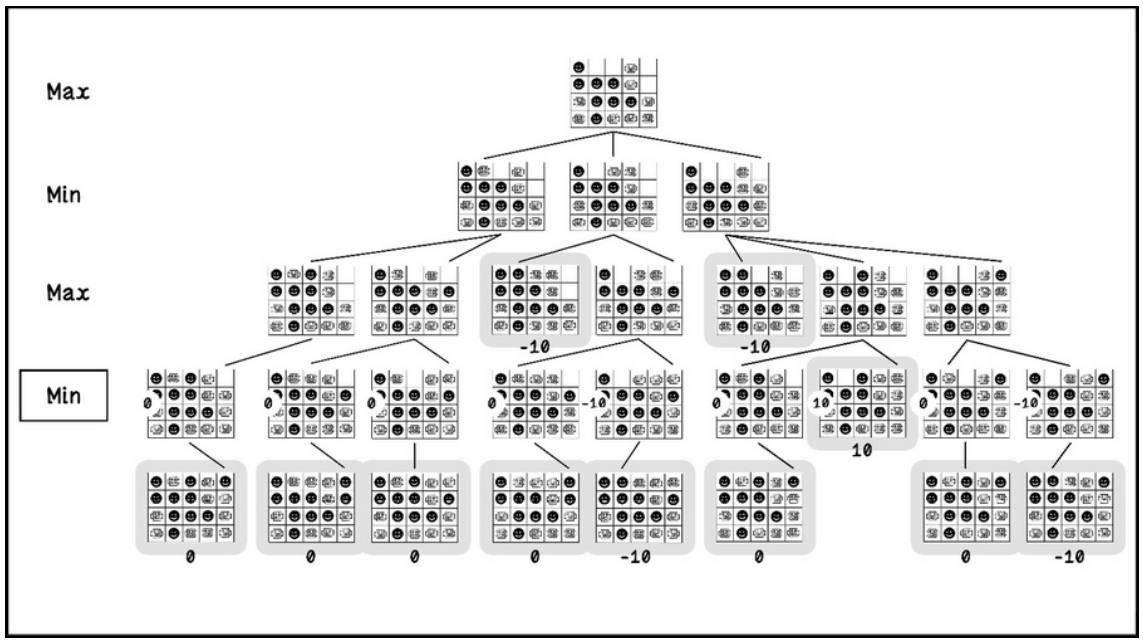


Figura 3.18 I possibili punteggi per gli stati finali in una partita di Forza quattro (Parte 1).

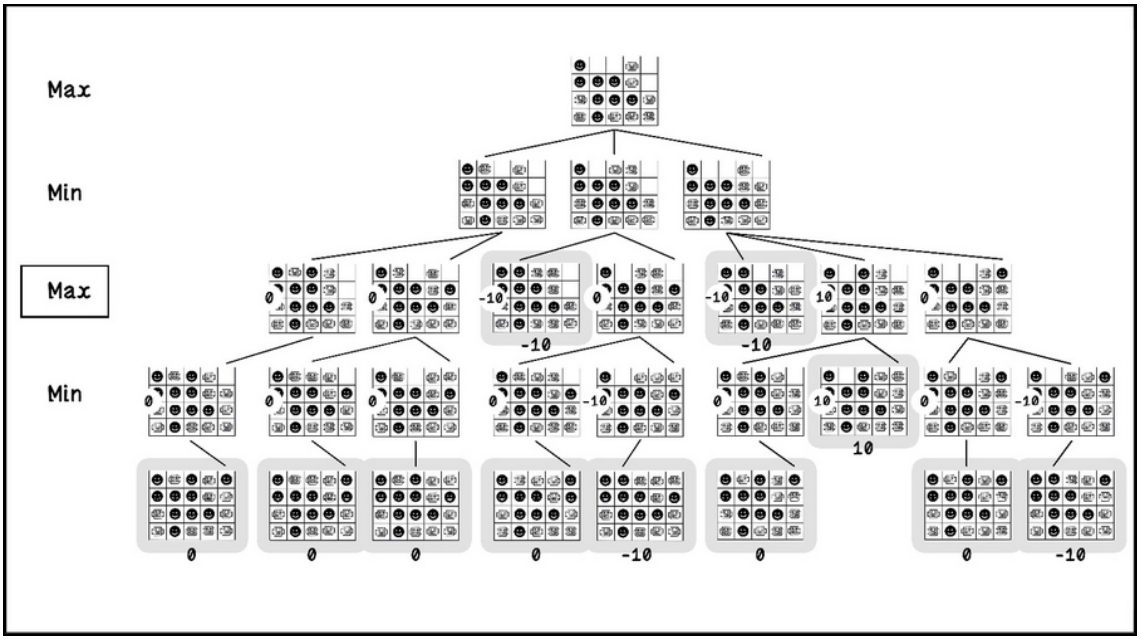


Figura 3.19 I possibili punteggi per gli stati finali in una partita di Forza quattro (Parte 2).

L'algoritmo presuppone che l'avversario farà sempre la mossa più intelligente per massimizzare le sue possibilità di vincita (Figura 3.20).

L'albero semplificato nella Figura 3.21 rappresenta il risultato dell'algoritmo di ricerca min-max per l'esempio di stato del gioco.

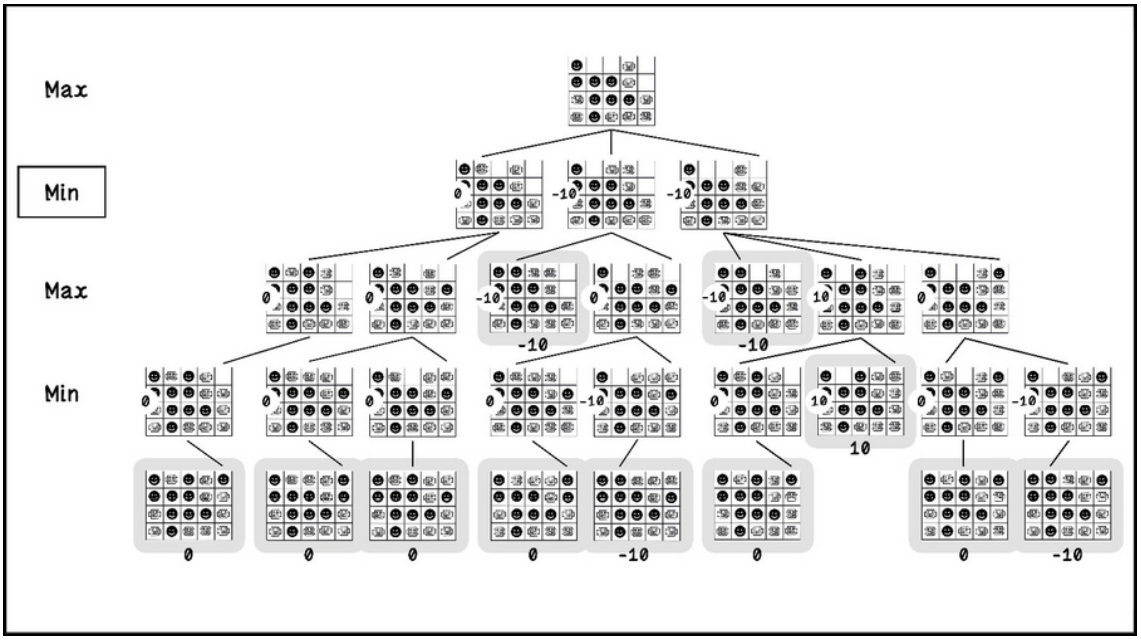


Figura 3.20 I possibili punteggi per gli stati finali in una partita di Forza quattro (Parte 3).

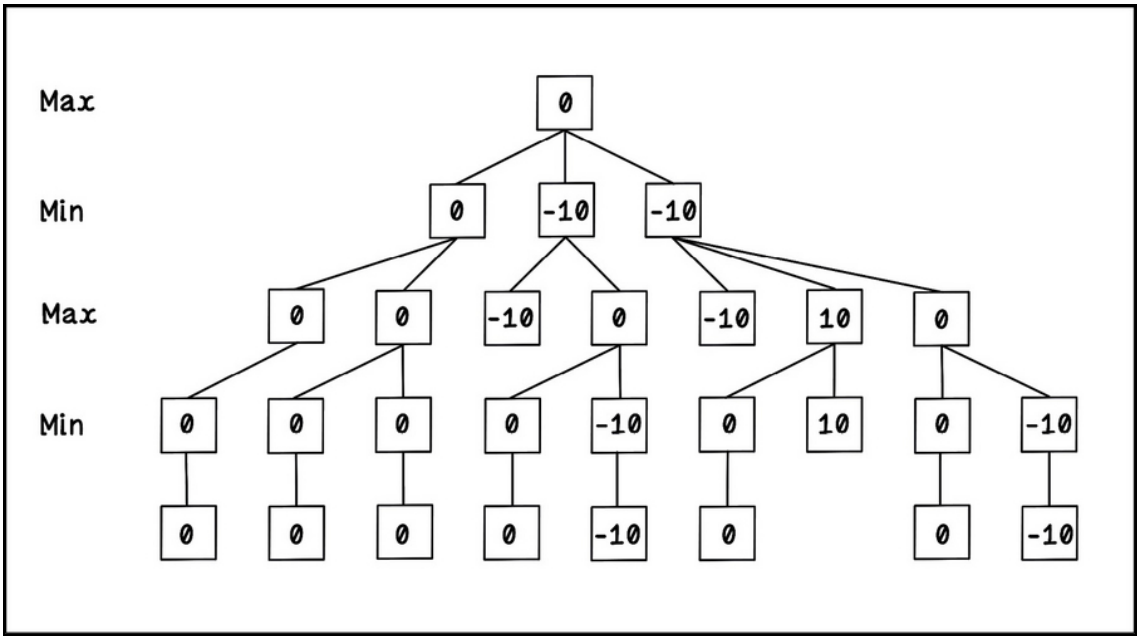


Figura 3.21 Albero di gioco semplificato con punteggio min-max.

Pseudocodice

L'algoritmo di ricerca min-max è implementato come una funzione ricorsiva. Alla funzione viene fornito lo stato corrente, la profondità desiderata per la ricerca, la modalità di minimizzazione o massimizzazione e l'ultima mossa. L'algoritmo termina restituendo la mossa e il punteggio migliori per ogni nodo figlio a ogni profondità dell'albero. Confrontando il codice con il diagramma di flusso rappresentato nella Figura 3.15, notiamo che le noiose condizioni di controllo se la modalità corrente è di massimizzazione o minimizzazione non sono così evidenti. Nello pseudocodice, 1 o -1 rappresentano rispettivamente l'intenzione di massimizzare o minimizzare. Utilizzando una logica intelligente, il punteggio, le condizioni e gli stati di commutazione migliori possono essere ottenuti tramite il principio della moltiplicazione negativa: un numero negativo moltiplicato per un altro numero negativo dà un risultato positivo. Quindi se -1 indica il turno dell'avversario, moltiplicandolo per -1 si ottiene 1, che indica il turno dell'agente. Per il turno successivo, 1 moltiplicato per -1 dà come risultato -1 per indicare nuovamente il turno dell'avversario:

```
minmax(state, depth, last_move):
  let current_score equal state.get_score
  if current_score is not equal to 0 or state.is_full or depth is equal to
0:
      return new Move(last_move, current_score)
  let best_score equal to min_or_max multiplied by −∞
  let best_move −1
  for each possible choice(0 to 4 in a 5x4 board) as move:
      let neighbor equal to a copy of state
      execute current move on neighbor
       let best_neighbor equal minmax(neighbor, depth −1, min_or_max * -1,
move)
      if(best_neighbor.score is greater than best_score and is MAX)
          or (best_neighbor.score is less than best_score and min_or_max is
MIN):
          let best_score = best_neighbor.score
          let best_move = best_neighbor.move
  return new Move(best_move, best_score)
```

Potatura alfa-beta: ottimizza esplorando solo i percorsi sensati

La *potatura alfa-beta* è una tecnica utilizzata con l'algoritmo di ricerca min-max per evitare di esplorare aree dell'albero del gioco che producono soluzioni notoriamente scadenti. Questa tecnica ottimizza

l'algoritmo di ricerca min-max per risparmiare calcoli, poiché i percorsi non sensati vengono ignorati. Poiché sappiamo che l'albero di gioco dell'esempio con Forza quattro esplode, è evidente che il fatto di ignorare selettivamente un maggior numero di percorsi migliorerà significativamente le prestazioni (Figura 3.22).

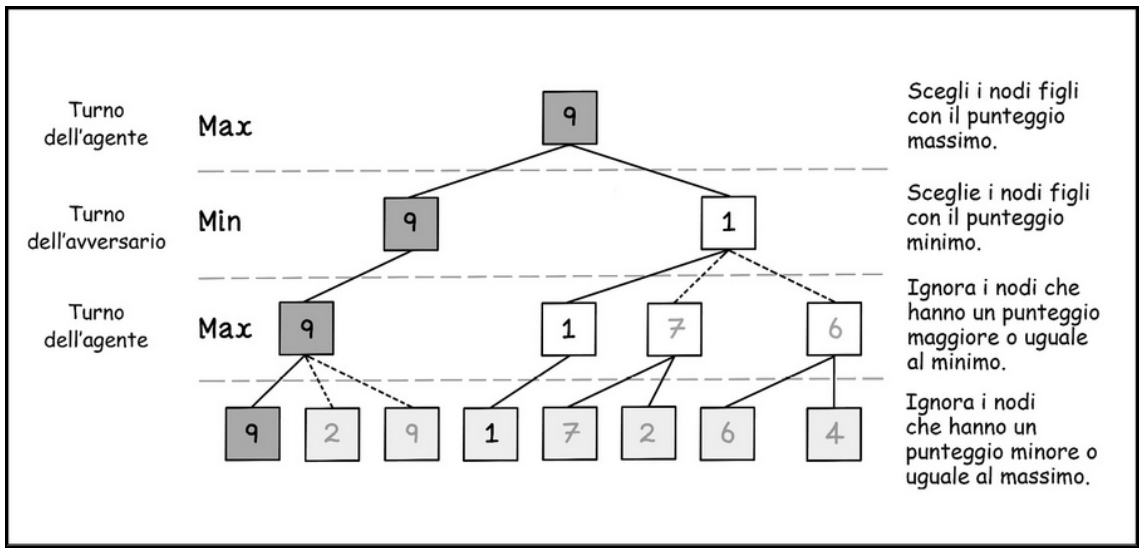


Figura 3.22 Un esempio di potatura alfa-beta.

L'algoritmo di potatura alfa-beta funziona memorizzando il miglior punteggio per il giocatore che massimizza come alfa e il miglior punteggio per il giocatore che minimizza come beta. Inizialmente, alfa è impostato a $-\infty$ e beta è impostato a ∞, il peggior punteggio per ogni giocatore. Se il miglior punteggio del giocatore che minimizza è minore del miglior punteggio del giocatore che massimizza, è logico che altri percorsi figli dei nodi già visitati non influenzeranno il miglior punteggio.

La Figura 3.23 illustra le modifiche apportate al flusso di ricerca min-max per consentire l'ottimizzazione della potatura alfa-beta. I blocchi evidenziati sono i passaggi aggiuntivi nel flusso dell'algoritmo di ricerca min-max.

- *La modalità corrente è MAX?* Di nuovo, determina se l'algoritmo sta attualmente tentando di massimizzare o minimizzare il punteggio.
- *best_n è maggiore o uguale ad alfa?* Se la modalità corrente è quella di massimizzare il punteggio e il miglior punteggio corrente è maggiore o uguale ad alfa, nei nodi figli di quel nodo non si troveranno punteggi migliori, quindi l'algoritmo può ignorare quel nodo.
- *Imposta alfa come best_n*. Imposta la variabile *alfa* come *best_n*.
- *alfa è maggiore o uguale a beta?* Il punteggio non è migliore degli altri punteggi trovati, e il resto dell'esplorazione di quel nodo può essere "potato".
- *best_n è minore o uguale a beta?* Se la modalità corrente è quella di minimizzare il punteggio e il miglior punteggio corrente è minore o uguale a beta, nei nodi figli di quel nodo non si troveranno punteggi migliori, quindi l'algoritmo può ignorare quel nodo.
- *Imposta beta come best_n*. Imposta la variabile *beta* come *best_n*.
- *alfa è maggiore o uguale a beta?* Il punteggio non è migliore degli altri punteggi trovati, e il resto dell'esplorazione di quel nodo può essere "potato".

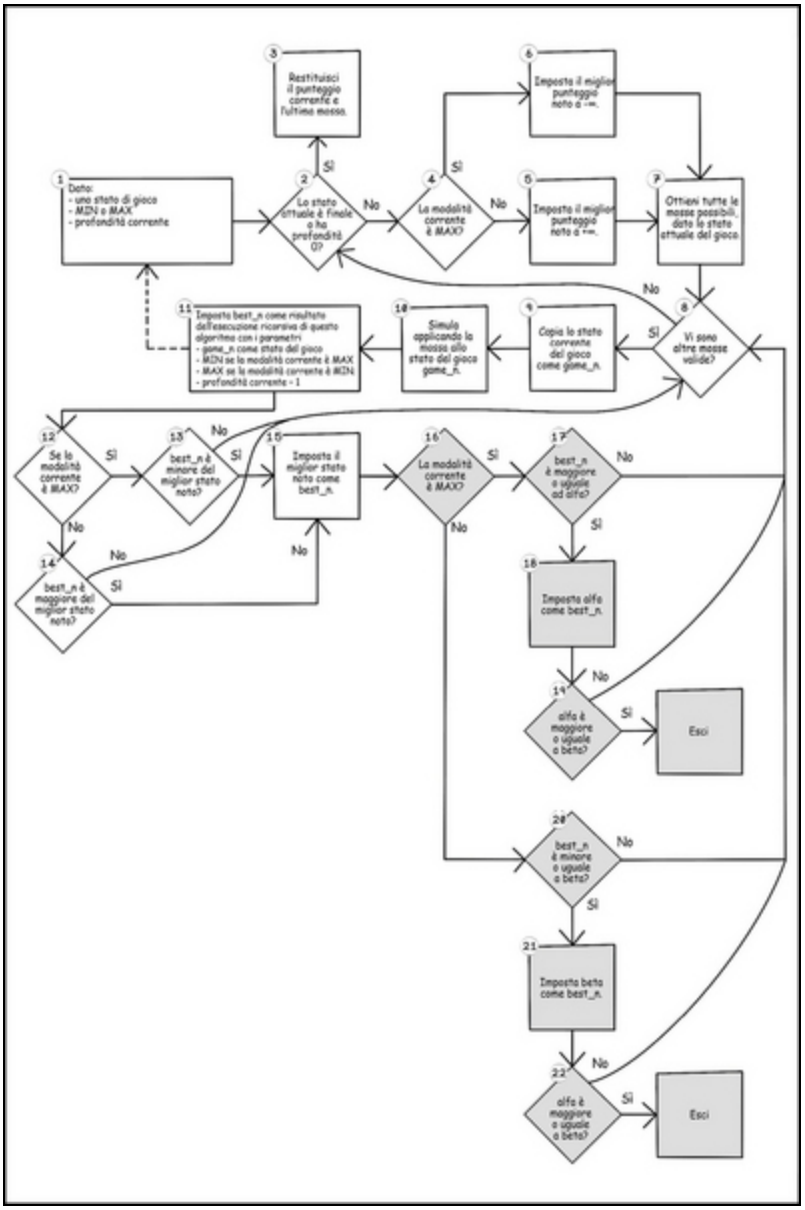


Figura 3.23 Diagramma di flusso dell'algoritmo di ricerca min-max con potatura alfa-beta.

Ecco i passaggi aggiunti all'algoritmo di ricerca min-max. Queste condizioni consentono di terminare l'esplorazione dei percorsi quando il miglior punteggio trovato non cambierebbe il risultato.

Pseudocodice

Lo pseudocodice per ottenere la potatura alfa-beta è in gran parte lo stesso del codice per la ricerca min-max, con l'aggiunta della registrazione e gestione dei valori alfa e beta mentre viene attraversato l'albero. Notate che quando viene selezionato il minimo (MIN) la variabile *min_or_max* è -1, e quando viene selezionato il massimo (MAX), la variabile *min_or_max* è 1:

```
minmax_ab_pruning(state, depth, min_or_max, last_move, alpha, beta):
  let current_score equal state.get_score
  if current_score is not equal to 0 or state.is_full or depth is equal to
0:
      return new Move(last_move, current_score)
  let best_score equal to min_or_max multiplied by −∞
  let best_move = −1
  for each possible choice(0 to 4 in a 5x4 board) as move:
      let neighbor equal to a copy of state
      execute current move on neighbor
        let best_neighbor equal                  minmax(neighbor, depth −1,
min_or_max * −1, move, alpha, beta)
       if (best_neighbor.score is greater than best_score and min_or_max is
MAX)
         or (best_neighbor.score is less than best_score and min_or_max is
MIN):
          let best_score = best_neighbor.score
          let best_move = best_neighbor.move
          if best_score >= alpha:
              alpha = best_score
          if best_score <= beta:
              beta = best_score
          if alpha >= beta:
              break
  return new Move(best_move, best_score)
```

Casi d'uso per gli algoritmi di ricerca avversaria

Gli algoritmi di ricerca informata sono versatili e utili in casi d'uso come i seguenti.

- *Creazione di agenti per giochi a turni con informazioni note*: in alcuni giochi, due o più giocatori operano nello stesso ambiente. Vi sono implementazioni di successo di scacchi, dama e altri giochi classici. I giochi con informazioni note non hanno informazioni nascoste o interventi del caso.
- *Creazione di agenti per giochi a turni con informazioni imperfette*: in questi giochi esistono opzioni future sconosciute,

come nel poker e in Scarabeo.

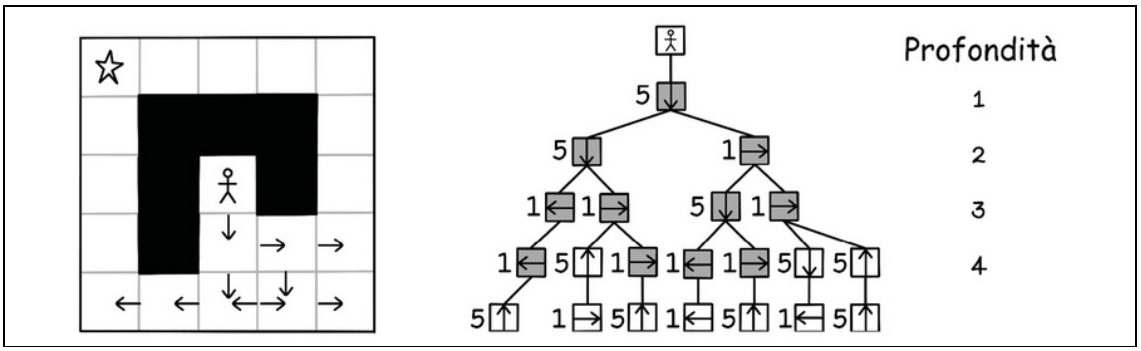
- *Ricerca avversaria e ottimizzazione a colonia di formiche (ACO) per l'ottimizzazione dei percorsi*: la ricerca avversaria viene utilizzata in combinazione con l'algoritmo ACO (vedi il Capitolo 6) per ottimizzare i percorsi di consegna dei pacchi.

Riepilogo

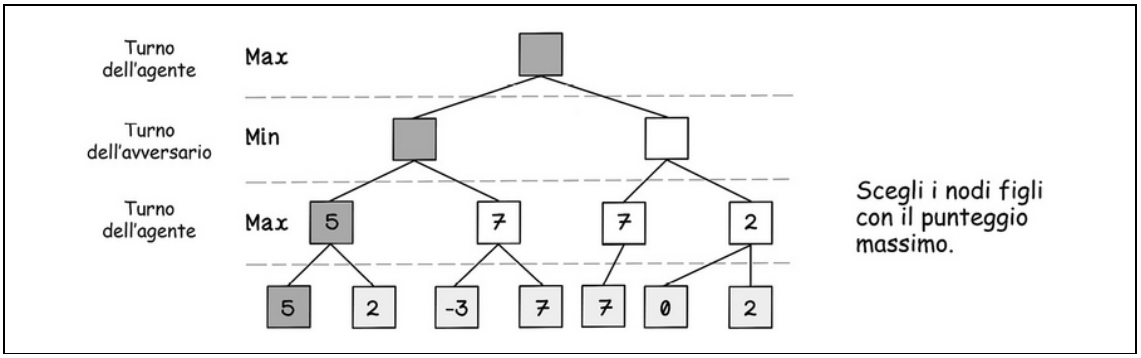
La ricerca informata dà agli algoritmi una certa intelligenza.

- L'euristica può essere difficile da definire, ma una buona euristica può essere potente per riuscire a trovare le soluzioni in modo efficiente.
- La ricerca A* utilizza l'euristica e la distanza dalla radice per trovare soluzioni ottimali.

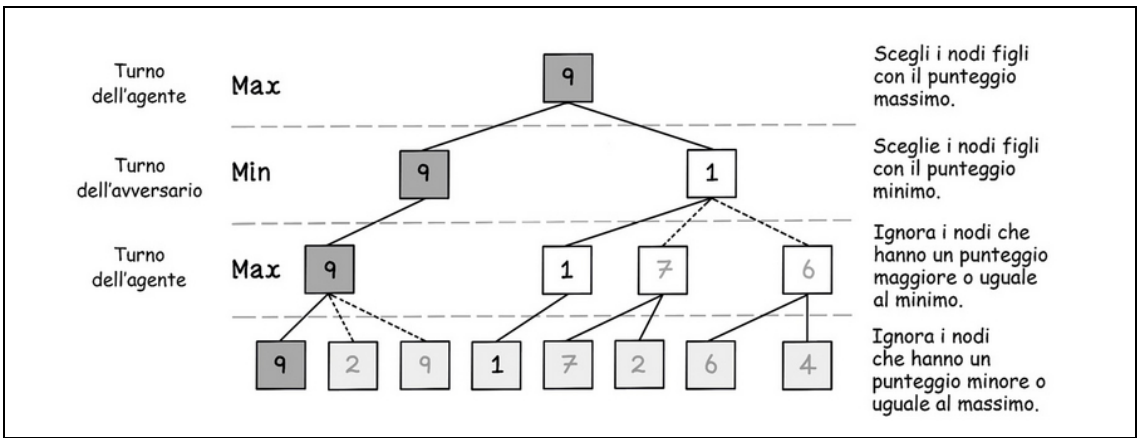
$$f(n) = g(n) + h(n)$$



- La ricerca avversaria, come min-max, è utile quando vi è qualche fattore che influenza l'ambiente.



- La potatura alfa-beta aiuta a ottimizzare l'algoritmo min-max eliminando i percorsi indesiderabili.



Algoritmi evolutivi

Che cosa si intente per evoluzione?

Quando osserviamo il mondo che ci circonda, può sorgerci la domanda di come sia nato tutto ciò che vediamo e con cui interagiamo. Un modo per spiegarlo è la teoria dell'evoluzione. La teoria dell'evoluzione suggerisce che gli organismi viventi di oggi non siano sorti all'improvviso in quel modo, ma che si siano evoluti in milioni di anni attraverso piccoli cambiamenti, con i quali ogni generazione si adattava al proprio ambiente. Ciò implica che le caratteristiche fisiche e cognitive di ciascun organismo vivente sono il risultato del più efficace adattamento al suo ambiente, con il fine della sopravvivenza. L'evoluzionismo suggerisce che gli organismi si evolvano attraverso la riproduzione, dove i figli mescolano i geni dei genitori. In base all'adattamento degli individui al loro ambiente, gli individui più forti hanno una maggiore probabilità di sopravvivere.

Possiamo pensare che l'evoluzione sia un processo lineare, con cambiamenti progressivi nei vari discendenti. In realtà, l'evoluzione è molto più caotica, con ampie divergenze all'interno di una specie. Attraverso la riproduzione e la mescolanza dei geni si genera una moltitudine di varianti di una specie. L'insorgere di differenze marcate in una specie potrebbe richiedere migliaia di anni, e realizzarsi solo confrontando vari "individui medi" attraverso vari punti temporali. La

Figura 4.1 rappresenta l'evoluzione percepita e l'evoluzione reale relativamente alla specie umana.

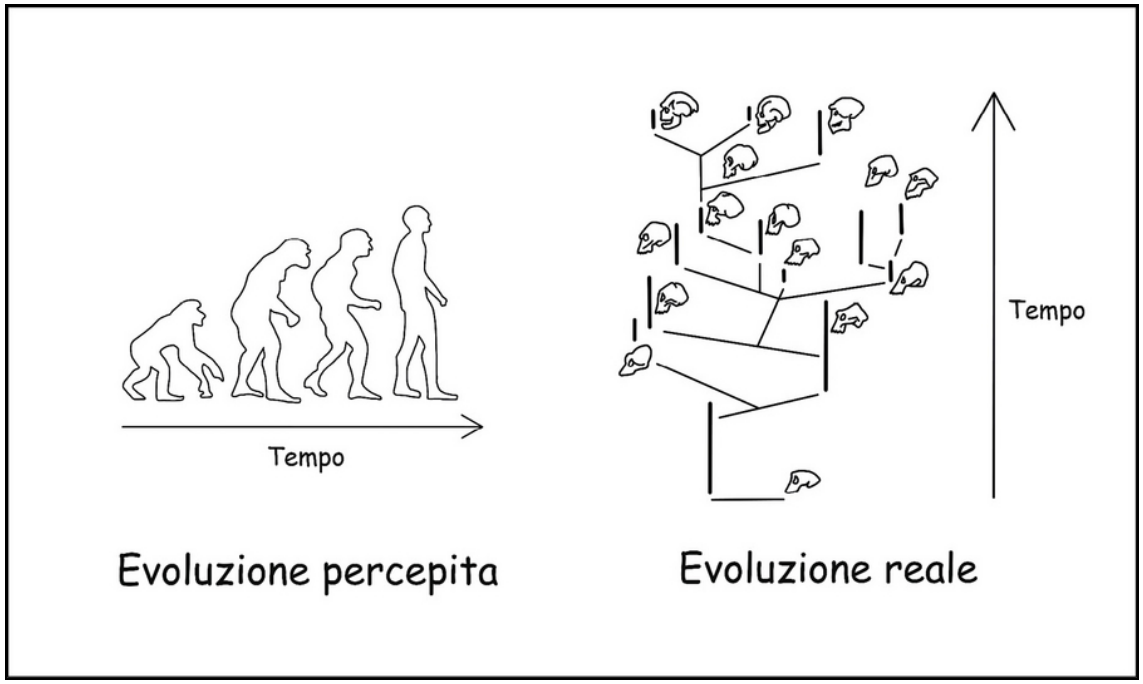


Figura 4.1 L'idea di evoluzione umana lineare rispetto all'evoluzione umana reale.

Charles Darwin propose una teoria dell'evoluzione incentrata sulla selezione naturale. La *selezione naturale* è il concetto secondo il quale i membri più forti di una popolazione hanno maggiori probabilità di sopravvivere perché sono più adatti al loro ambiente, il che fa sì che si riproducano di più e, quindi, che trasmettano alle generazioni future i loro tratti vantaggiosi per la sopravvivenza, che potrebbero potenzialmente funzionare meglio rispetto a quelli dei loro antenati.

Un classico esempio di evoluzione per adattamento è la falena delle betulle. Originariamente era di colore chiaro, il che costituiva un buon mimetismo contro i predatori, poiché la falena poteva mimetizzarsi con la corteccia chiara delle betulle e le superfici chiare. Solo il 2% circa della popolazione di falene era di colore scuro. Dopo la rivoluzione industriale, circa il 95% delle specie era della variante di colore scuro.

Una spiegazione è che le falene di colore chiaro non potevano più mimetizzarsi con tante superfici perché l'inquinamento le aveva scurite; quindi, le falene di colore chiaro venivano mangiate di più dai predatori, perché erano più visibili. Le falene scure avevano un vantaggio maggiore nel mimetizzarsi con le superfici scure, quindi sopravvissero più a lungo e si riprodussero di più, così le loro informazioni genetiche furono tramandate maggiormente ai successori.

Fra le falene delle betulle, l'attributo che cambiava a livello macroscopico era il colore. Tuttavia, questa proprietà non è cambiata magicamente. Perché il cambiamento avvenisse, i geni nelle falene con il colore scuro dovevano essere trasferiti ai successori.

In altri esempi di evoluzione naturale, possiamo vedere cambiamenti drammatici in qualcosa di più del semplice colore, ma in realtà questi cambiamenti sono influenzati da piccole differenze genetiche nel corso di molte generazioni (Figura 4.2).

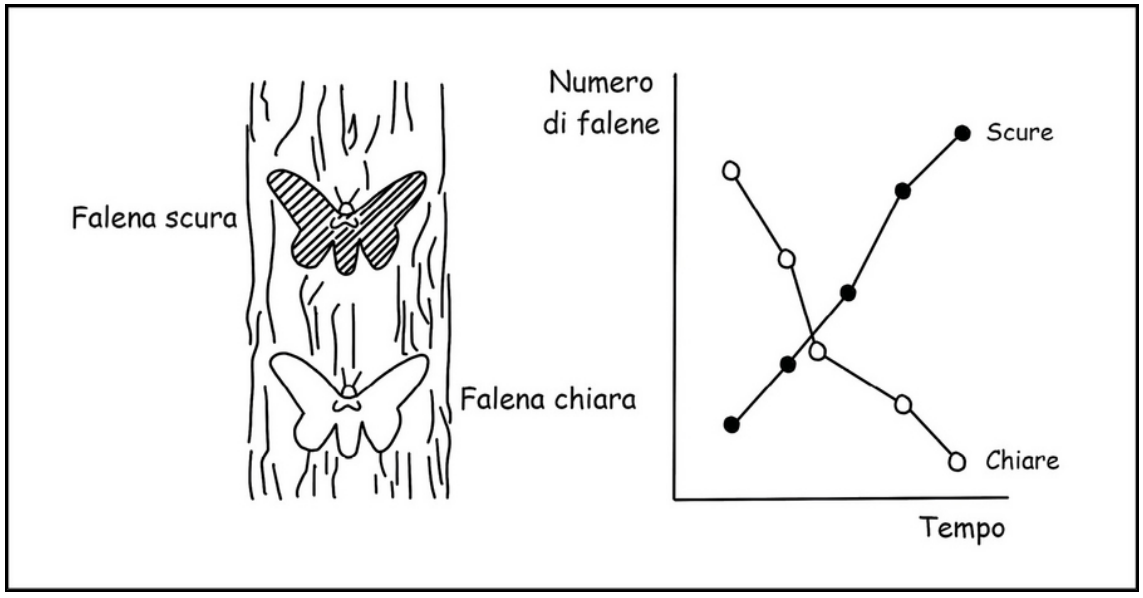


Figura 4.2 L'evoluzione della falena delle betulle.

In una popolazione di una specie, gli organismi si riproducono a coppie. La progenie impiega una combinazione dei geni dei genitori,

con piccoli cambiamenti attraverso un processo di *mutazione*. La prole fa comunque parte della popolazione. Tuttavia, non tutti i membri di una popolazione sopravvivono. Come è intuibile, malattie, lesioni e altri fattori possono causano la morte degli individui. Gli individui che risultano più adattabili all'ambiente che li circonda hanno maggiori probabilità di sopravvivere, una situazione che ha dato origine al termine *sopravvivenza del più adatto*. Sulla base della teoria dell'evoluzione darwiniana, una popolazione ha i seguenti attributi.

- *Varietà*: gli individui nella popolazione hanno tratti genetici differenti.
- *Ereditarietà*: un figlio eredita le proprietà genetiche dai suoi genitori.
- *Selezione*: un meccanismo misura l'idoneità degli individui. Gli individui più forti hanno maggiori probabilità di sopravvivere (la sopravvivenza del più adatto).

Queste proprietà implicano che, durante il processo di evoluzione, accadano le seguenti cose (Figura 4.3).

- *Riproduzione*: due individui si riproducono per dare origine a una prole.
- *Incroci e mutazioni*: la prole creata attraverso la riproduzione contiene un mix di geni dei genitori e presenta lievi modifiche casuali nel codice genetico.

In sintesi, l'evoluzione è un sistema meraviglioso e caotico che produce variazioni nelle forme di vita, alcune delle quali sono migliori di altre per determinati aspetti i determinati ambienti. Questa teoria si applica anche agli algoritmi evolutivi; i concetti dell'evoluzione biologica vengono sfruttati per trovare soluzioni ottimali a problemi pratici, generando varie soluzioni e convergendo su quelle che offrono prestazioni migliori nel corso di più generazioni.

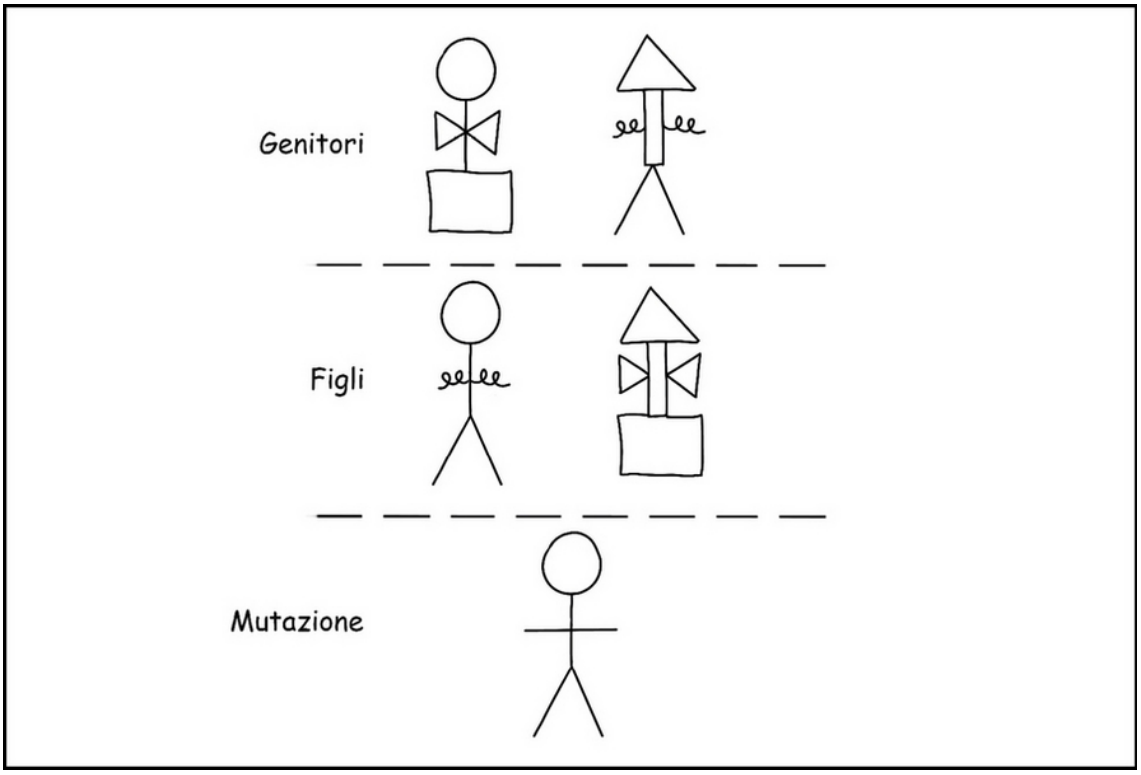


Figura 4.3 Un semplice esempio di riproduzione e mutazione.

Questo capitolo e il Capitolo 5 sono dedicati all'esplorazione degli algoritmi evolutivi, che sono approcci potenti ma sottovalutati per risolvere problemi complessi. Gli algoritmi evolutivi possono essere usati isolatamente o in costrutti come le reti neurali. Avere una solida comprensione di questi concetti apre molte possibilità per risolvere diversi nuovi problemi.

Problemi risolvibili tramite algoritmi evolutivi

Gli algoritmi evolutivi non sono applicabili alla risoluzione di tutti i problemi, ma sono molto efficaci nel risolvere i problemi di ottimizzazione, nei quali la soluzione consiste in un gran numero di

permutazioni o scelte. Questi problemi, in genere, offrono molte soluzioni valide, alcune delle quali sono però migliori di altre.

Considerate il problema dello zaino, un classico problema utilizzato per esplorare il funzionamento e l'efficienza degli algoritmi. Nel problema, uno zaino può sopportare un determinato peso massimo. Nello zaino possono essere inseriti vari articoli, ognuno di peso e valore differente. L'obiettivo è quello di inserire nello zaino il maggior numero possibile di oggetti, in modo da massimizzare il valore totale senza superare il peso totale che rappresenta il limite dello zaino. Nella variante più semplice del problema si ignorano la forma e le dimensioni fisiche degli elementi (Figura 4.4).

Come esempio banale, data la specifica del problema rappresentata nella Tabella 4.1, uno zaino può sopportare un peso totale di 9 kg e può contenere uno qualsiasi degli otto oggetti, di peso e valore variabili.

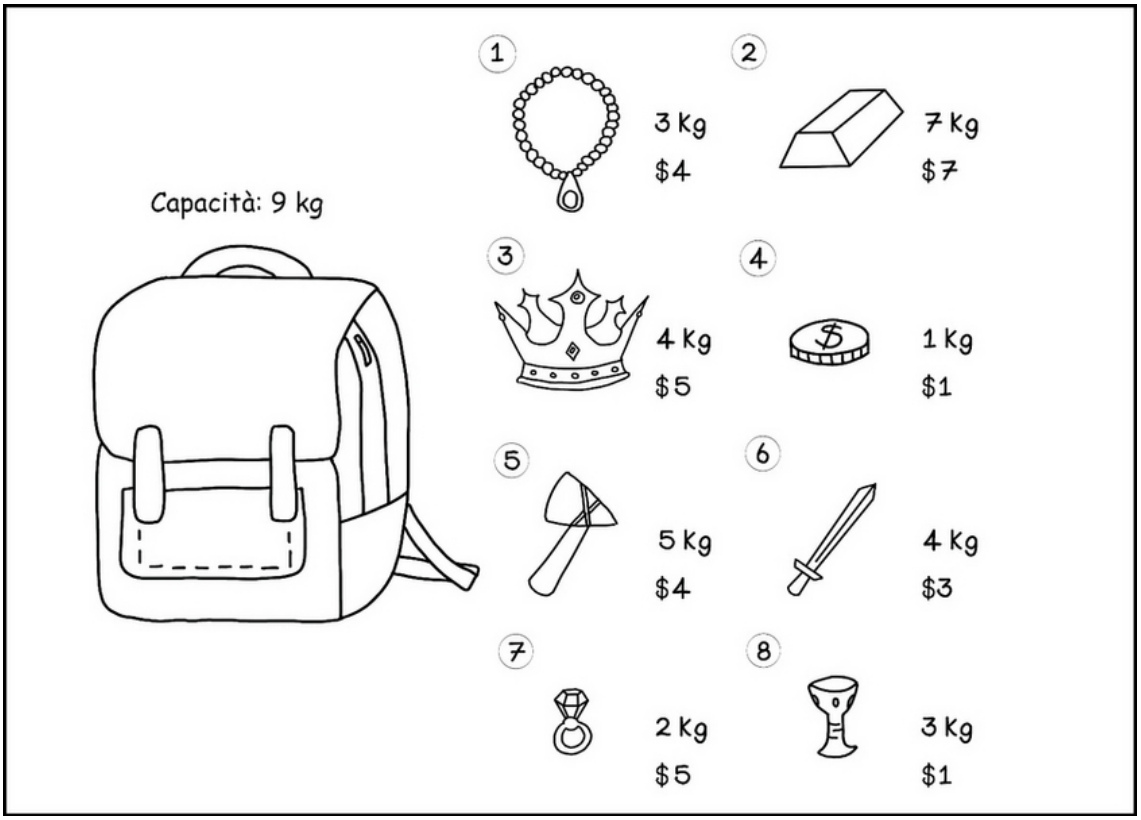


Figura 4.4 Un semplice esempio di problema dello zaino.

Tabella 4.1 Capacità dello zaino: 9 kg.

ID articolo	Nome articolo	Peso (kg)	Valore ($)
1	Perla	3	4
2	Oro	7	7
3	Corona	4	5
4	Monete	1	1
5	Ascia	5	4
6	Spada	4	3
7	Anello	2	5

8	Coppa	3	1

Questo problema ha 255 soluzioni possibili, incluse le seguenti (Figura 4.5).

- *Soluzione 1*: articoli 1, 4 e 6. Il peso totale è di 8 kg e il valore totale è di 8 dollari.
- *Soluzione 2* articoli 1, 3 e 7. Il peso totale è di 9 kg e il valore totale è di 14 dollari.
- *Soluzione 3* articoli 2, 3 e 6. Il peso totale è di 15 kg, superiore alla capacità dello zaino.

Chiaramente, la soluzione con il maggior valore è la *Soluzione 2*. Per ora non preoccupatevi troppo di come viene calcolato il numero di possibilità, ma considerate solo il fatto che le possibilità esplodono all'aumentare del numero di elementi.

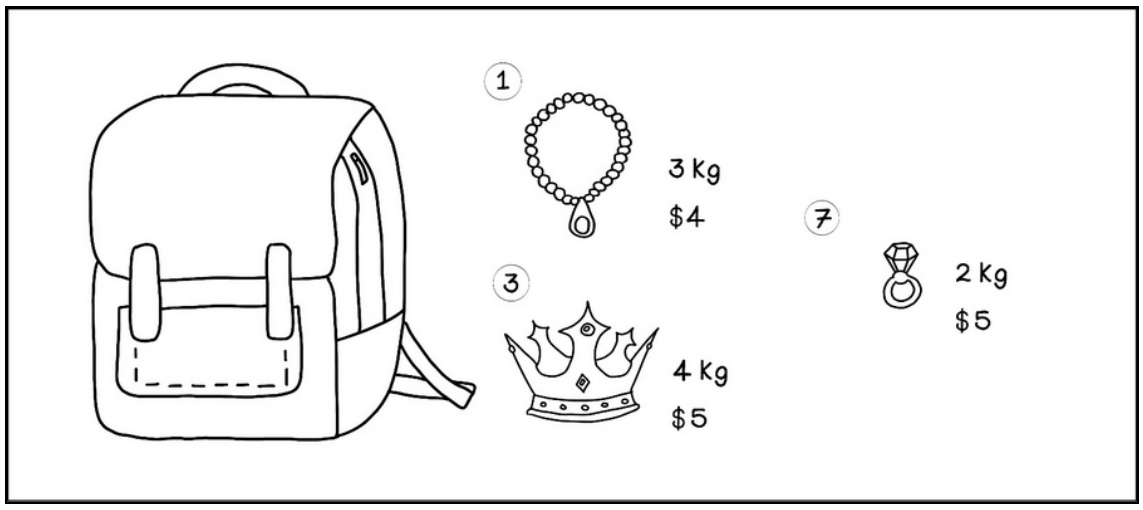


Figura 4.5 La soluzione ottimale per il semplice esempio del problema dello zaino.

Sebbene questo banale esempio possa essere risolto a mano, il problema dello zaino potrebbe avere particolari vincoli di peso, un numero variabile di elementi e pesi e valori variabili per ciascun elemento, cosa che rende impossibile la risoluzione manuale del

problema al crescere delle variabili. Sarà anche computazionalmente costoso provare a risolvere a forza bruta ogni combinazione di elementi all'aumentare delle variabili; quindi, cerchiamo degli algoritmi che siano efficienti nel trovare una soluzione desiderabile.

Notate che qualifichiamo la soluzione migliore che possiamo trovare come *soluzione desiderabile* e non *soluzione ottimale*. Sebbene alcuni algoritmi tentino di trovare *l'unica vera* soluzione ottimale del problema dello zaino, un algoritmo evolutivo *tenta* di trovare la soluzione ottimale, ma non è garantito che la trovi. Tuttavia, l'algoritmo troverà una soluzione desiderabile per il caso d'uso, dipendente da un'opinione soggettiva relativamente al problema. Per un sistema sanitario critico, per esempio, una soluzione "abbastanza buona" potrebbe non essere sufficiente; per un sistema di suggerimenti di canzoni, una soluzione "abbastanza buona" potrebbe essere accettabile.

Considerate ora il dataset più grande (con uno "zaino" gigantesco) della Tabella 4.2. Il numero di elementi, pesi e valori rende il problema difficile da risolvere a mano. Data la complessità di questo dataset, potete facilmente intuire perché molti algoritmi vengano misurati in base alle loro prestazioni nella risoluzione di tali problemi. Le prestazioni definiscono l'efficacia di una determinata soluzione nel risolvere un problema, non necessariamente le prestazioni computazionali. Nel problema dello zaino, è preferibile una soluzione che produce un valore totale più elevato. Gli algoritmi evolutivi forniscono un metodo per trovare soluzioni al problema dello zaino.

Tabella 4.2 Portata zaino: 6.404.180 kg.

ID articolo	Nome articolo	Peso (kg)	Valore ($)
1	Ascia	32.252	68.674
2	Moneta di bronzo	225.790	471.010

3	Corona	468.164	944.620
4	Statua di diamanti	489.494	962.094
5	Cintura di smeraldi	35.384	78.344
6	Fossile	265.590	579.152
7	Moneta d’oro	497.911	902.698
8	Elmo	800.493	1.686.515
9	Inchiostro	823.576	1.688.691
10	Portagioie	552.202	1.056.157
11	Coltello	323.618	677.562
12	Spada lunga	382.846	833.132
13	Maschera	44.676	99.192
14	Collana	169.738	376.418
15	Distintivo in opale	610.876	1.253.986
16	Perla	854.190	1.853.562
17	Faretra	671.123	1.320.297
18	Anello con rubino	698.180	1.301.637
19	Bracciale d’argento	446.517	859.835
20	Orologio	909.620	1.677.534
21	Uniforme	904.818	1.910.501
22	Pozione di veleno	730.061	1.528.646
23	Sciarpa di lana	931.932	1.827.477
24	Balestra	952.360	2.068.204

25	Annale	926.023	1.746.556
26	Coppa di zinco	978.724	2.100.851

Un modo per risolvere questo problema consiste nell'utilizzare un approccio a forza bruta. Questo approccio implica il calcolo di ogni possibile combinazione di elementi e la determinazione del valore di ciascuna combinazione che soddisfa il vincolo di peso dello zaino, fino a incontrare la soluzione migliore.

La Figura 4.6 mostra alcune analisi di riferimento per l'approccio a forza bruta. Notate che il calcolo si basa sull'hardware di un tipico PC.

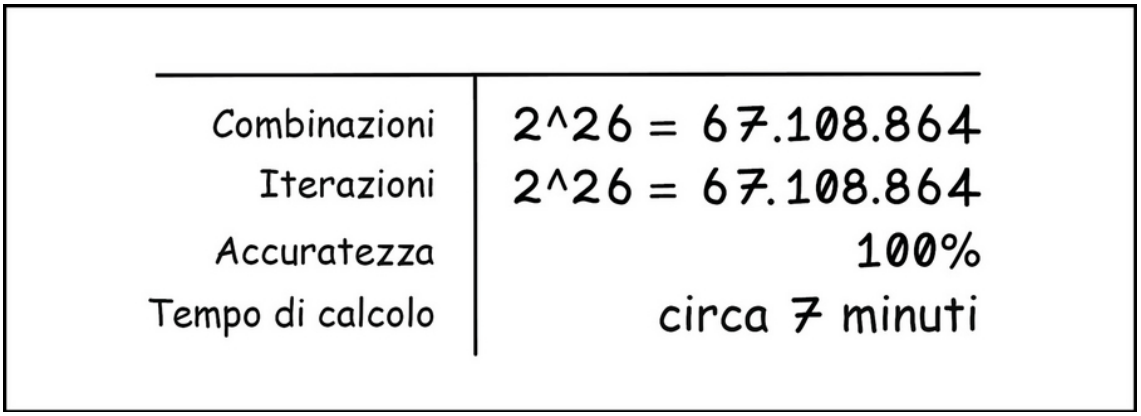


Figura 4.6 Analisi delle prestazioni dell'approccio a forza bruta nel problema dello zaino.

Tenete a mente il problema dello zaino, poiché verrà utilizzato in questo capitolo mentre tentiamo di comprendere, progettare e sviluppare un algoritmo genetico per trovare soluzioni accettabili a questo problema.

NOTA

Una nota sul termine *prestazioni*: dal punto di vista di una singola soluzione, le prestazioni valutano l'efficacia della soluzione nel risolvere il problema. Dal punto di vista dell'algoritmo, le prestazioni possono essere il modo in cui una determinata configurazione riesce a trovare una soluzione. Infine, le

prestazioni possono contare i cicli computazionali consumati. Tenete presente che questo termine viene utilizzato in modo differente in base al contesto.

I concetti di base dell'utilizzo di un algoritmo genetico per risolvere il problema dello zaino possono essere applicati a tutta una serie di problemi pratici. Se una società di logistica desiderasse ottimizzare la composizione dei camion in base alle loro destinazioni, per esempio, dovrebbe impiegare un algoritmo genetico. La stessa cosa se volesse trovare il percorso più breve fra più destinazioni. Se una fabbrica dovesse produrre articoli con determinate materie prime tramite un sistema di nastri trasportatori e l'ordine degli articoli influenzasse la produttività, un algoritmo genetico sarebbe utile per determinare tale ordine.

Mentre vi immergerete nei concetti, nell'approccio e nel ciclo di vita di un algoritmo genetico, dovrebbe risultarvi chiaro dove questo potente algoritmo può essere applicato nel vostro lavoro. È importante tenere presente che un algoritmo genetico è *stocastico*, il che significa che è probabile che il risultato dell'algoritmo sia diverso ogni volta che viene eseguito.

Il ciclo di vita dell'algoritmo genetico

L'algoritmo genetico è un membro della famiglia degli algoritmi evolutivi. Ogni algoritmo di questa famiglia funziona in base alla premessa evolutiva, ma con leggere varianti nelle diverse parti del ciclo di vita, per far fronte ai diversi problemi. Esploriamo alcuni di questi parametri nel Capitolo 5.

Gli algoritmi genetici vengono utilizzati per valutare ampi spazi di ricerca con lo scopo di fornire una buona soluzione. Attenzione: non è garantito che un algoritmo genetico trovi la soluzione migliore in assoluto; tenta di trovare la migliore soluzione globale evitando le migliori soluzioni locali.

Migliore globale è la migliore soluzione possibile; *migliore locale* è una soluzione meno ottimale. La Figura 4.7 rappresenta le "migliori soluzioni" possibili in un problema di minimizzazione, ovvero, di ricerca del valore minore. Quando l'obiettivo è la massimizzazione di una soluzione, si cerca il valore maggiore. Gli algoritmi di ottimizzazione come gli algoritmi genetici puntano a trovare in modo incrementale le migliori soluzioni locali, alla ricerca della migliore soluzione globale.

È necessario porre particolare attenzione nel configurare i parametri dell'algoritmo, in modo che all'inizio punti alla ricerca di soluzioni divergenti, e poi graviti gradualmente verso soluzioni migliori, generazione dopo generazione. All'inizio, le soluzioni potenziali dovrebbero variare ampiamente quanto ai singoli attributi genetici. Senza questa divergenza iniziale, aumenterebbe il rischio di rimanere bloccati in un migliore locale (Figura 4.8).

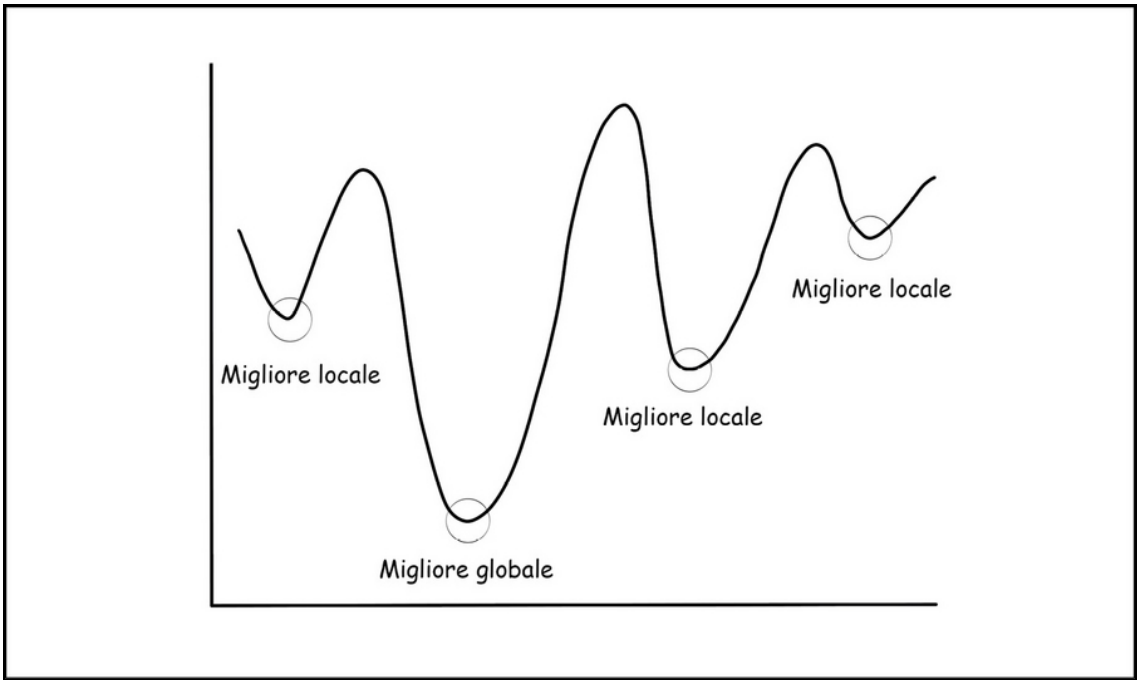


Figura 4.7 Migliore locale vs. migliore globale.

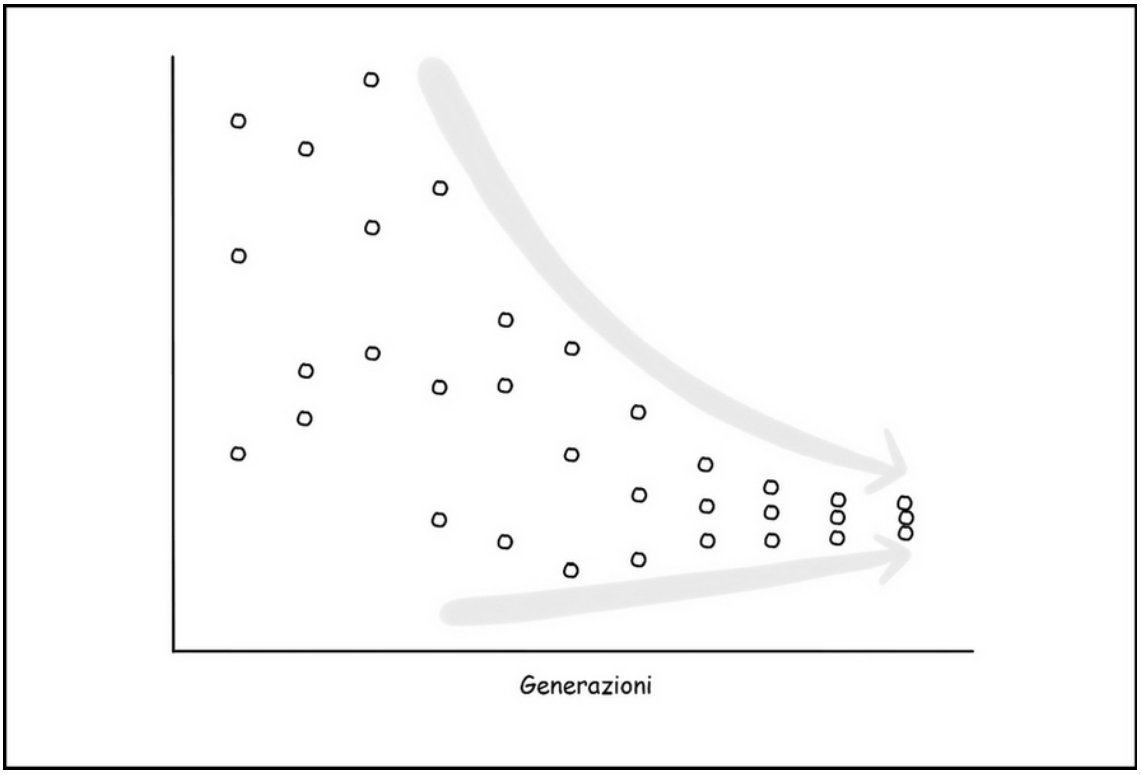


Figura 4.8 Dalla diversità alla convergenza.

La configurazione di un algoritmo genetico dipende dallo spazio del problema. Ogni problema ha un contesto unico e un dominio differente, e pertanto i dati sono rappresentati in modo differente e le soluzioni vengono valutate in modo differente.

Il ciclo di vita generale di un algoritmo genetico è il seguente.

- *Creazione di una popolazione*: occorre trovare una popolazione casuale delle soluzioni.
- *Misurazione della qualità degli individui nella popolazione*: occorre determinare quanto è valida una soluzione. Questo compito viene eseguito utilizzando una funzione che assegna un punteggio alle soluzioni, per determinarne la qualità.
- *Selezione dei genitori in base alla loro qualità*: sceglie coppie di genitori che si riprodurranno.

- *Riproduzione dei genitori*: occorre creare i figli a partire dai genitori, mescolando le informazioni genetiche e applicando lievi mutazioni alla prole.
- *Popolamento della generazione successiva*: ora occorre selezionare gli individui e i figli che sopravvivrà alla generazione successiva.

L'implementazione di un algoritmo genetico prevede diversi passaggi, che generano le fasi del ciclo di vita dell'algoritmo (Figura 4.9).

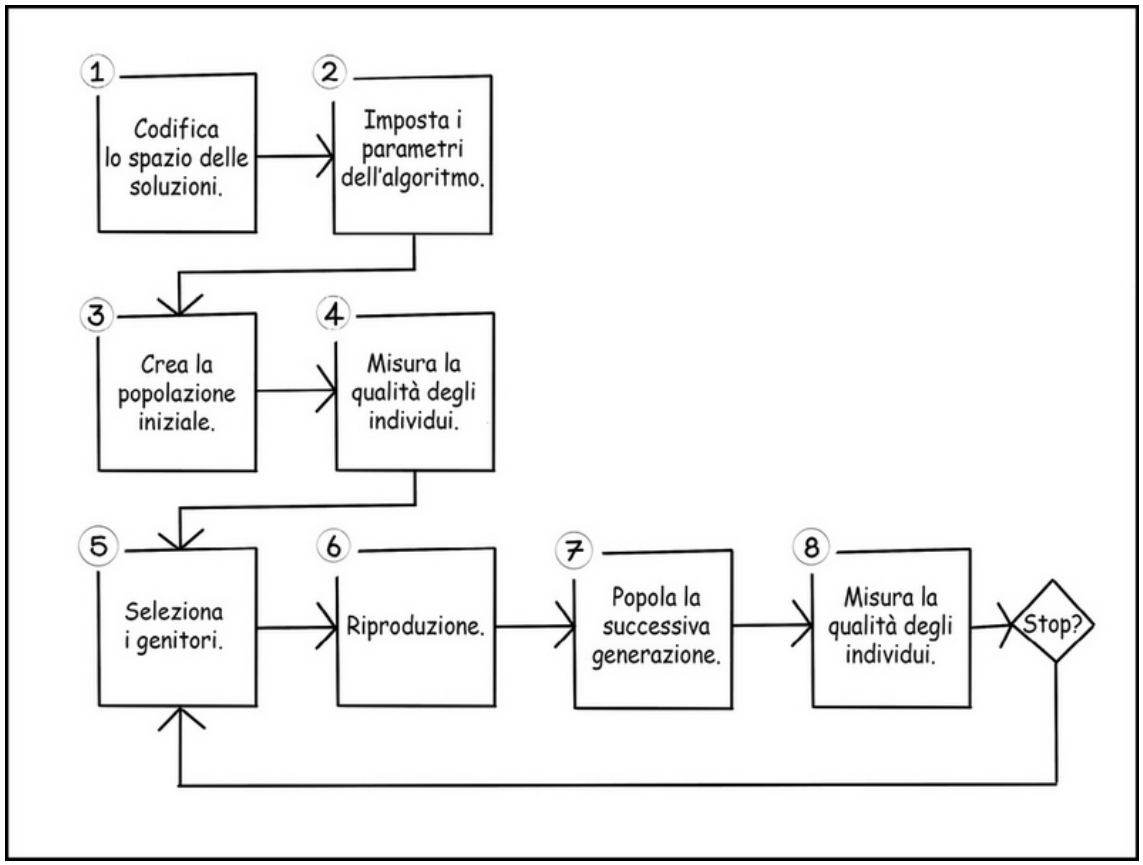


Figura 4.9 Ciclo di vita dell'algoritmo genetico.

Tenendo presente il problema dello zaino, come potremmo utilizzare un algoritmo genetico per trovare le soluzioni del problema? Lo vedremo nel prossimo paragrafo.

Codifica dello spazio delle soluzioni

Quando utilizziamo un algoritmo genetico, è fondamentale eseguire correttamente la fase di codifica, che richiede un'attenta progettazione della rappresentazione dei possibili stati. Uno *stato* è una struttura di dati con regole specifiche che rappresenta le soluzioni possibili di un problema. Un insieme di stati forma una *popolazione* (Figura 4.10).

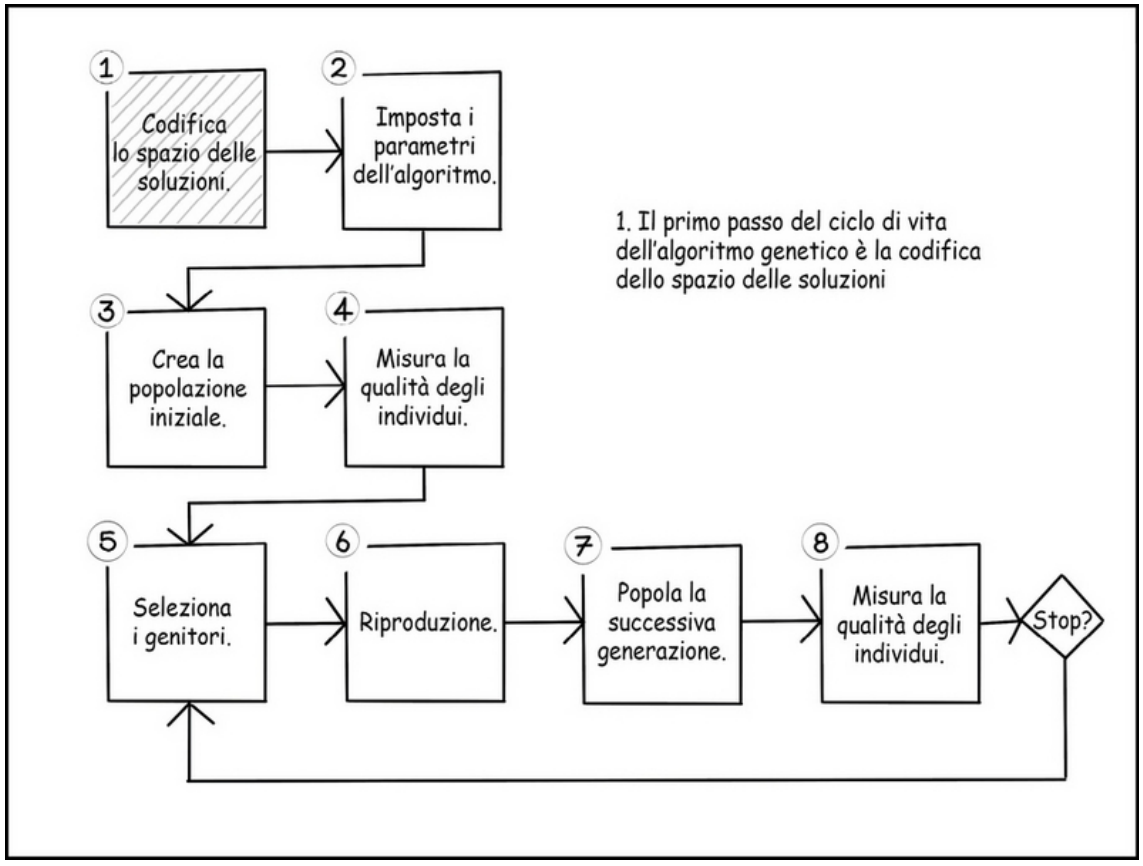


Figura 4.10 Codifica della soluzione.

Terminologia

In termini di algoritmi evolutivi, una singola soluzione candidata è chiamata *cromosoma*. Un cromosoma è formato da geni. *Gene* è il tipo logico dell'unità e *allele* è il valore effettivo memorizzato in tale unità. Un *genotipo* è una rappresentazione di una soluzione e un *fenotipo* è esso stesso una soluzione.

Ogni cromosoma ha sempre lo stesso numero di geni. Un insieme di cromosomi forma una *popolazione* (Figura 4.11).

Nel problema dello zaino, possono essere messi dentro lo zaino diversi oggetti. Un modo semplice per descrivere una possibile soluzione che contiene alcuni elementi ma non altri è la codifica binaria (Figura 4.12). La *codifica binaria* rappresenta gli elementi inclusi con 1 e gli elementi esclusi con 0. Per esempio, se il valore del gene all'indice 3 è 1, quell'elemento è contrassegnato per essere incluso nello zaino. La stringa binaria ha sempre la stessa dimensione, ovvero il numero totale di elementi selezionabili per l'inserimento nello zaino. Esistono tuttavia vari schemi di codifica alternativi, descritti nel Capitolo 5.

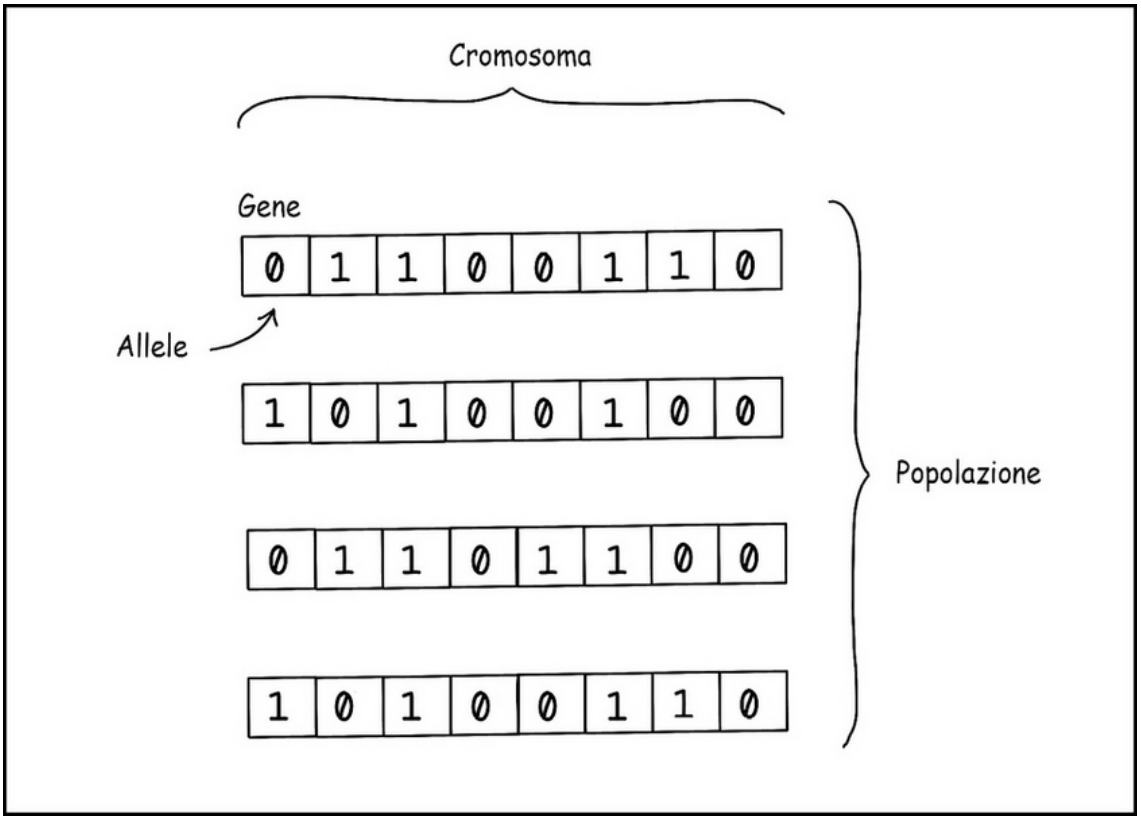


Figura 4.11 Terminologia delle strutture di dati per rappresentare una popolazione di soluzioni.

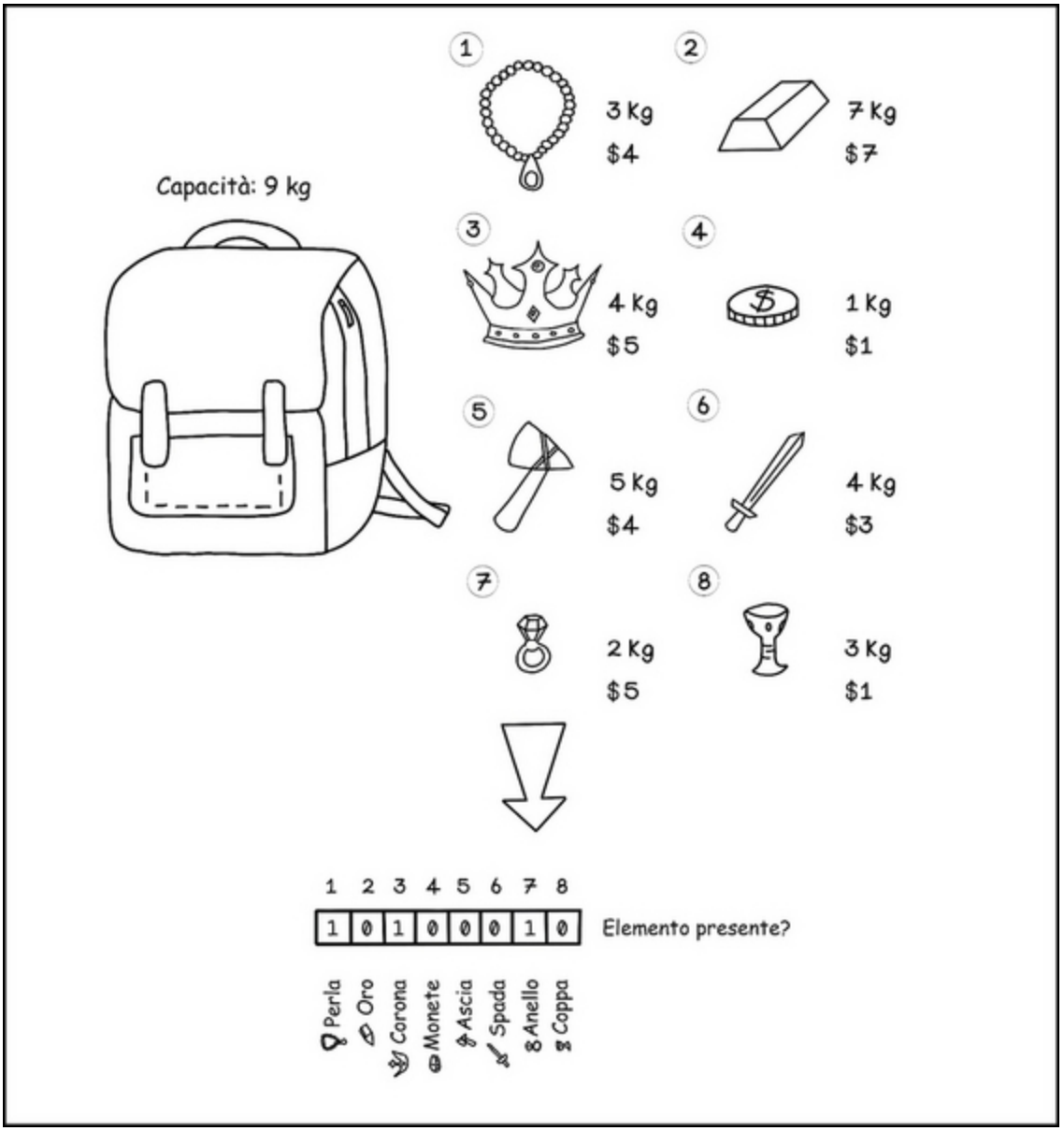


Figura 4.12 Codifica binaria del problema dello zaino.

Codifica binaria: rappresentazione delle soluzioni possibili tramite zeri e uni

La codifica binaria rappresenta un gene con 0 o 1, quindi un cromosoma è rappresentato da una stringa di bit binari. La codifica binaria può essere utilizzata in tanti modi, per esprimere la presenza di

un elemento o anche per codificare valori numerici come numeri binari. Il vantaggio della codifica binaria è che di solito è più performante grazie alla sua semplicità. L'uso della codifica binaria richiede meno memoria di lavoro e, a seconda del linguaggio utilizzato, le operazioni binarie sono computazionalmente più veloci. Ma occorre esercitare il buonsenso per assicurarsi che la codifica abbia senso per il problema in questione e rappresenti bene le soluzioni potenziali; in caso contrario, l'algoritmo potrebbe degradare (Figura 4.13).

Dato il problema dello zaino con un dataset costituito da 26 elementi di peso e valore variabili, è possibile utilizzare una stringa binaria per rappresentare l'inclusione di ciascun elemento. Il risultato è una stringa di 26 caratteri, in cui per ogni indice 0 indica che il rispettivo elemento è escluso e 1 indica che il rispettivo elemento è incluso.

Altri schemi di codifica, fra cui la codifica a valori effettivi, a ordine e ad albero, saranno trattati nel Capitolo 5.

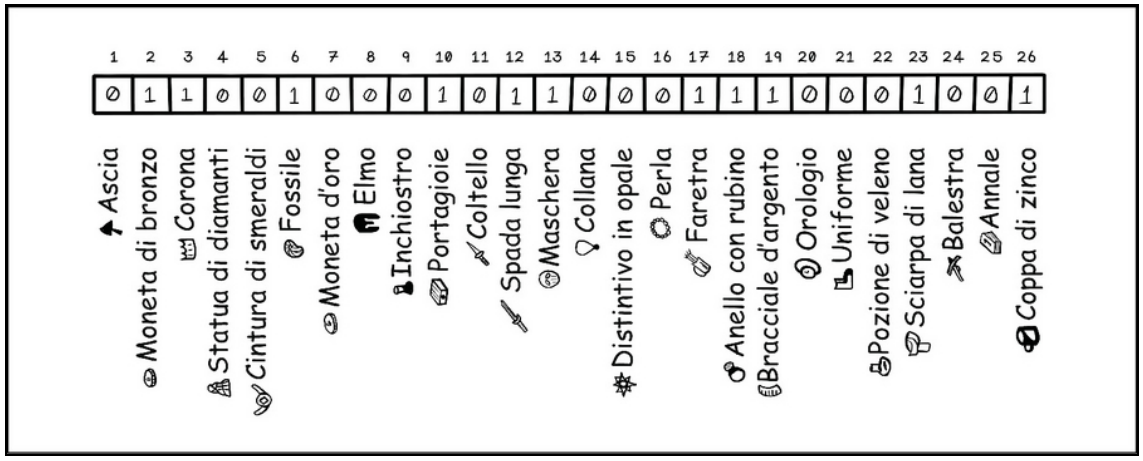
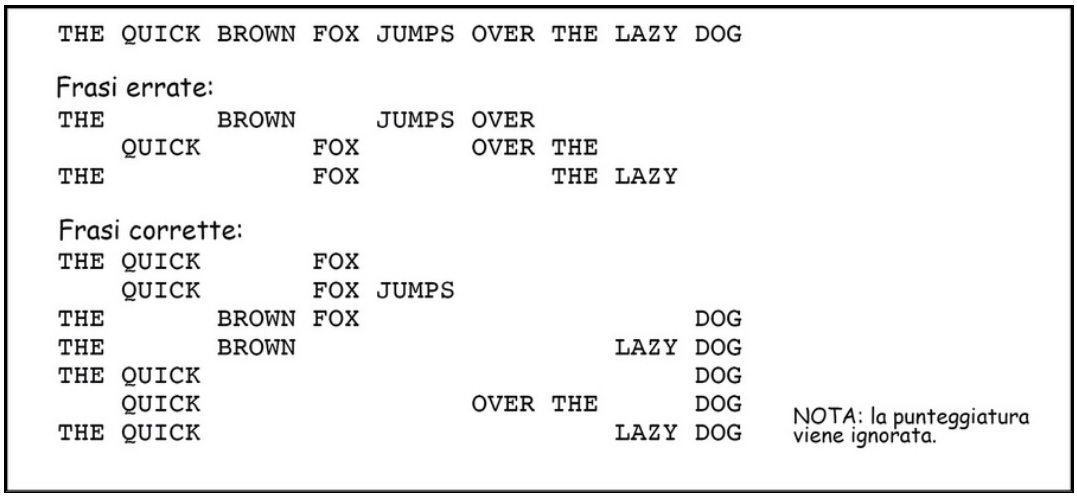


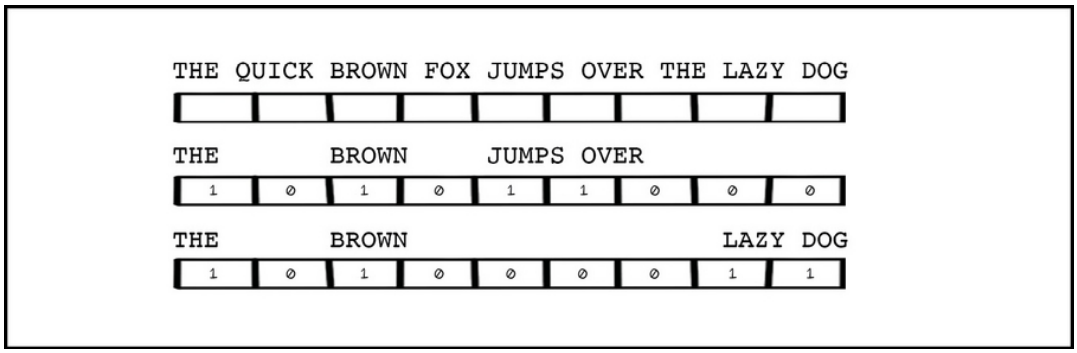
Figura 4.13 Codifica binaria del dataset del problema dello zaino più grande.

Esercizio: qual è una possibile codifica per il seguente problema?
Supponete di avere la seguente frase e di voler trovare quali parole possono essere escluse o incluse per mantenere comprensibile la frase utilizzando un algoritmo genetico:



Soluzione

Poiché il numero di parole possibili è sempre lo stesso e le parole sono sempre nella stessa posizione, è possibile utilizzare la codifica binaria per descrivere quali parole sono incluse e quali escluse. Il cromosoma consiste di 9 geni, ogni gene indica una parola della frase.



Creazione di una popolazione di soluzioni

All'inizio occorre creare la popolazione. Il primo passo di un algoritmo genetico è l'inizializzazione delle soluzioni potenziali casuali del problema in questione. Nel processo di inizializzazione della popolazione, sebbene i cromosomi vengano generati in modo

casuale, occorre prendere in considerazione i vincoli del problema, e le soluzioni potenziali devono essere valide o deve essere assegnato loro un pessimo punteggio se violano i vincoli. Ogni individuo della popolazione può anche non risolvere il problema in modo ottimale, ma deve rappresentare una soluzione valida. Come accennato nell'esempio precedente di inserimento degli oggetti in uno zaino, una soluzione che chieda di inserire lo stesso oggetto più volte non è valida e non dovrebbe far parte della popolazione delle soluzioni potenziali (Figura 4.14).

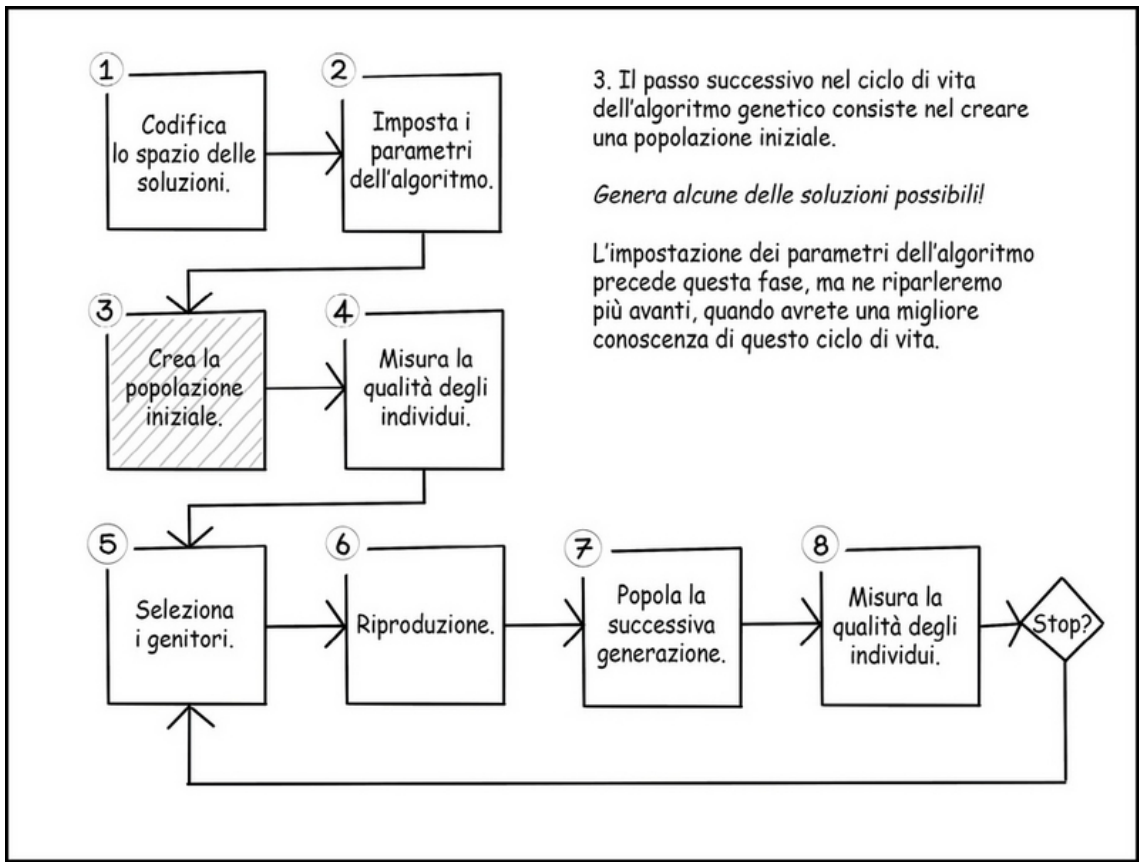


Figura 4.14 Creazione di una popolazione iniziale.

Dato come viene rappresentato lo stato delle soluzioni del problema dello zaino, questa implementazione decide in modo casuale se ciascun articolo deve essere incluso nello zaino. Detto questo, dovrebbero

essere prese in considerazione solo le soluzioni che soddisfano il vincolo del limite di peso. Il problema del semplice procedere da sinistra a destra e della scelta casuale dell'elemento incluso è che crea una propensione verso gli elementi all'estremità sinistra del cromosoma. Allo stesso modo, se partiamo da destra, saremo sbilanciati verso gli elementi a destra. Un modo per risolvere questo problema consiste nel generare un individuo con geni casuali, e poi determinare se la soluzione trovata è valida e non viola alcun vincolo. Assegnare un punteggio pessimo alle soluzioni non valide può risolvere questo problema (Figura 4.15).

Pseudocodice

Per generare una popolazione iniziale di soluzioni possibili, viene creato un array vuoto per contenere gli individui. Quindi, per ogni individuo della popolazione, viene creato un array per contenere i geni dell'individuo. Ogni gene è impostato in modo casuale a 1 o a 0, per indicare se l'elemento a quell'indice è presente o assente:

```
generate_initial_population(population_size, individual_size)
  let population be an empty array
  for individual in range 0 to population_size:
      let current_individual be an empty array
      for gene in range 0 to individual_size:
          let random_gene be 0 or 1 randomly
          append random_gene to current_individual
      append current_individual to population
  return population
```

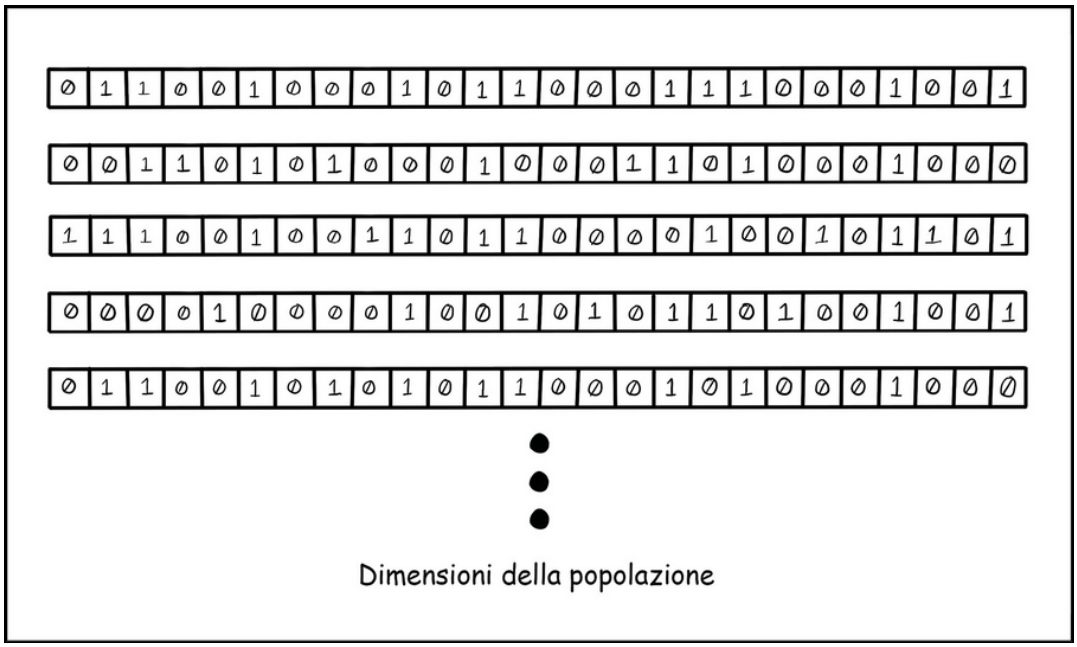


Figura 4.15 Un esempio di una popolazione di soluzioni.

Misurazione della qualità degli individui della popolazione

Dopo aver creato una popolazione, è necessario determinare la qualità di ogni suo individuo. La qualità definisce le prestazioni di una soluzione. La funzione che la valuta è fondamentale per il ciclo di vita di un algoritmo genetico. Se la qualità degli individui viene misurata in un modo errato o che non punta verso la soluzione ottimale, il processo di selezione dei genitori dei nuovi individui e delle nuove generazioni ne risentirà; l'algoritmo sarà imperfetto e faticherà a trovare la migliore soluzione possibile.

Le funzioni di qualità sono simili alle euristiche che abbiamo esplorato nel Capitolo 3. Sono linee guida per trovare buone soluzioni (Figura 4.16).

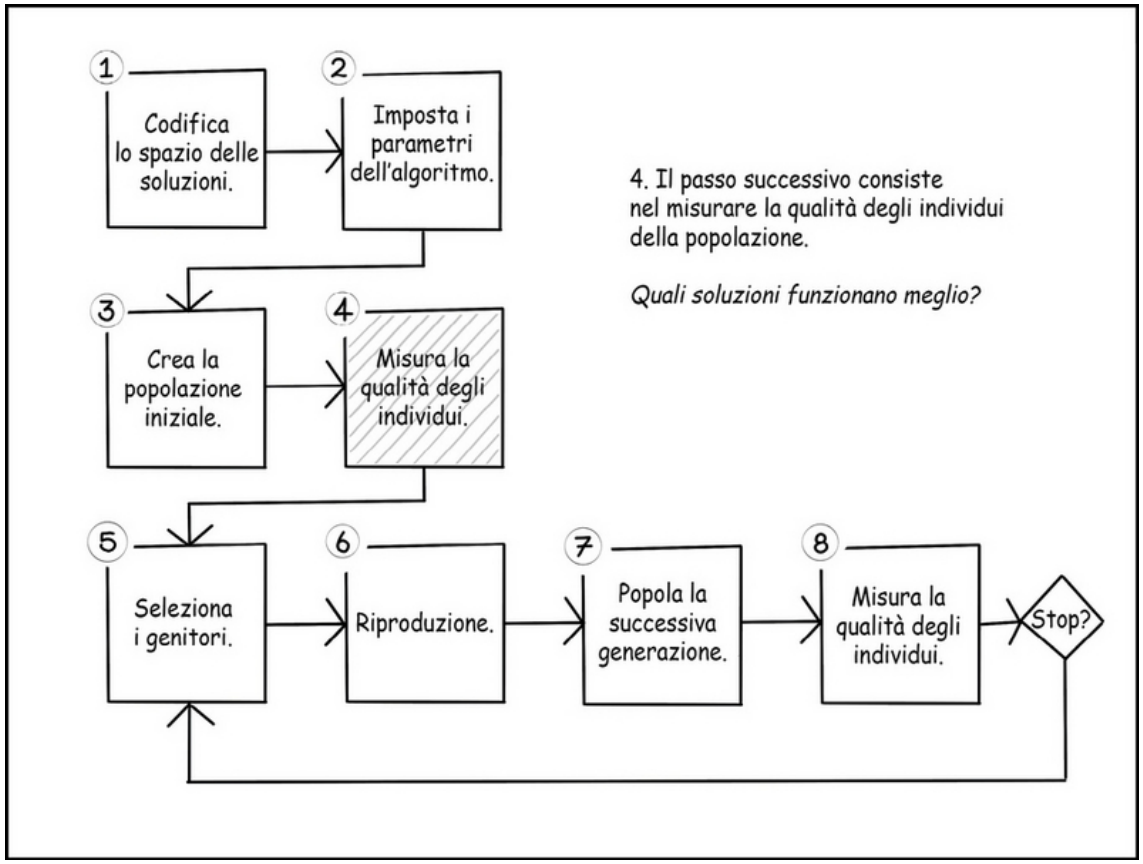


Figura 4.16 Misurazione della qualità degli individui.

Nel nostro esempio, la soluzione tenta di massimizzare il valore degli oggetti inseriti nello zaino rispettando i vincoli di peso. La funzione di valutazione della qualità calcola il valore totale degli oggetti inseriti nello zaino. Il risultato è che gli individui di valore più elevato sono i più adatti a essere inseriti. Notate che nella Figura 4.17 compare un individuo non valido, per evidenziare il fatto che il suo punteggio di qualità viene portato a 0, un punteggio terribile. Questo perché supera la capacità di peso per questa istanza del problema, che è 6.404.180.

A seconda del problema da risolvere, potrebbe essere necessario minimizzare o massimizzare il risultato della funzione di calcolo della qualità. Nel problema dello zaino, il contenuto può essere

massimizzato restando all'interno dei vincoli, ma in alternativa potrebbe essere minimizzato lo spazio vuoto rimasto nello zaino. L'approccio dipende dall'interpretazione del problema.

Pseudocodice

Per calcolare la qualità di un individuo nel problema dello zaino, devono essere calcolate le somme dei valori di ciascun elemento di ogni individuo. Questa attività viene eseguita impostando il valore totale a 0 e iterando su ciascun gene per determinare se sommare l'elemento che esso rappresenta. Se l'elemento è incluso, il valore dell'elemento rappresentato da quel gene deve essere aggiunto al valore totale. In tal modo viene calcolato il peso totale, il che consente anche di garantire che la soluzione sia valida. I concetti di calcolo della qualità e di controllo dei vincoli possono essere suddivisi, per una più chiara separazione degli ambiti:

```
calculate_individual_fitness(individual,                    knapsack_items,
knapsack_max_weight)
  let total_weight equal 0
  let total_value equal 0
  for gene_index in range 0 to length of individual:
let current_bit equal individual[gene_index]
if current_bit equals 1:
add weight of knapsack_items[gene_index] to total_weight
add value of knapsack_items [gene_index] to total_value
if total_weight is greater than knapsack_max_weight:
      return value as 0 since it exceeds the weight constraint
  return total_value as individual fitness
```

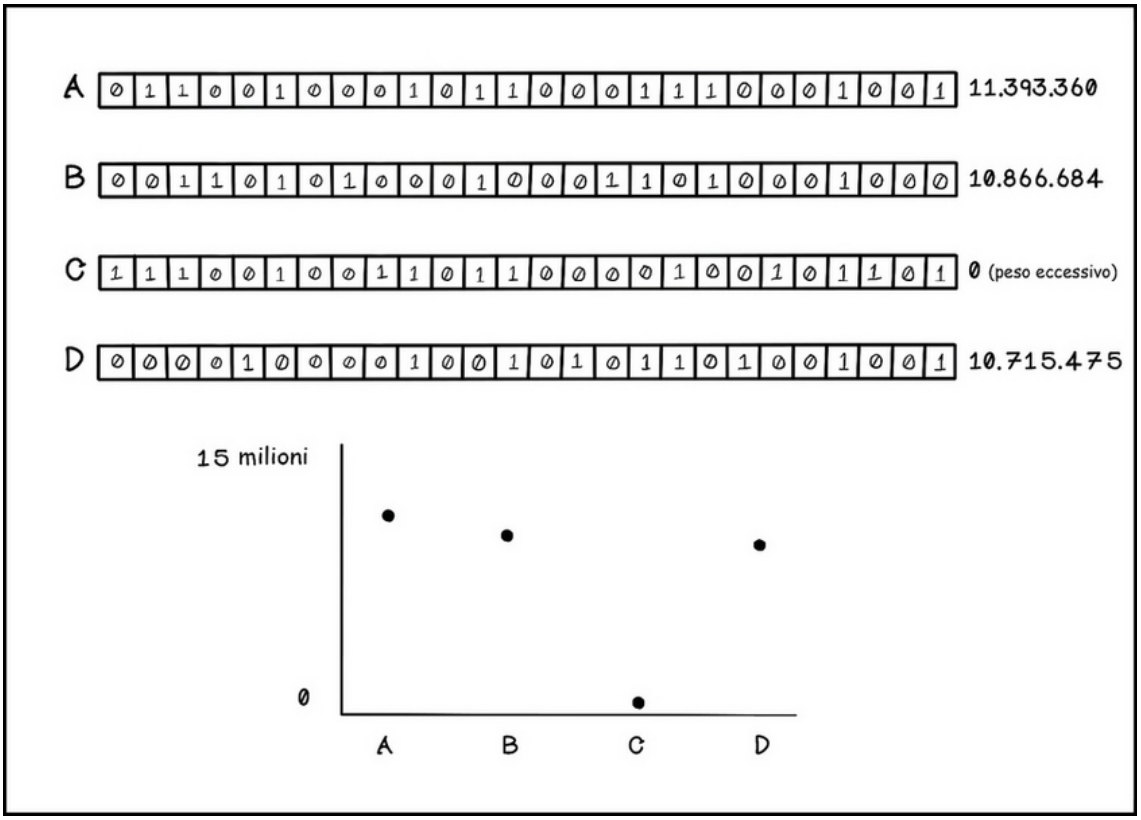


Figura 4.17 Misurazione della qualità degli individui.

Selezione dei genitori in base alla loro qualità

Il passo successivo in un algoritmo genetico consiste nel selezionare i genitori che produrranno nuovi individui. Nella teoria darwiniana, gli individui più adatti hanno maggiori probabilità di riprodursi rispetto agli altri, perché in genere vivono più a lungo. Inoltre, questi individui contengono attributi desiderabili per l'ereditarietà, a causa delle loro prestazioni superiori nel loro ambiente. Detto questo, è probabile che alcuni individui si riproducano anche se non sono i più adatti del gruppo: questi individui possono contenere tratti di qualità anche se non sono di alta qualità nella loro interezza.

Ogni individuo ha una sua qualità calcolata, che viene utilizzata per determinare la probabilità che venga selezionato per fungere da genitore di nuovi individui. Questo attributo fa sì che l'algoritmo genetico sia di natura stocastica (Figura 4.18).

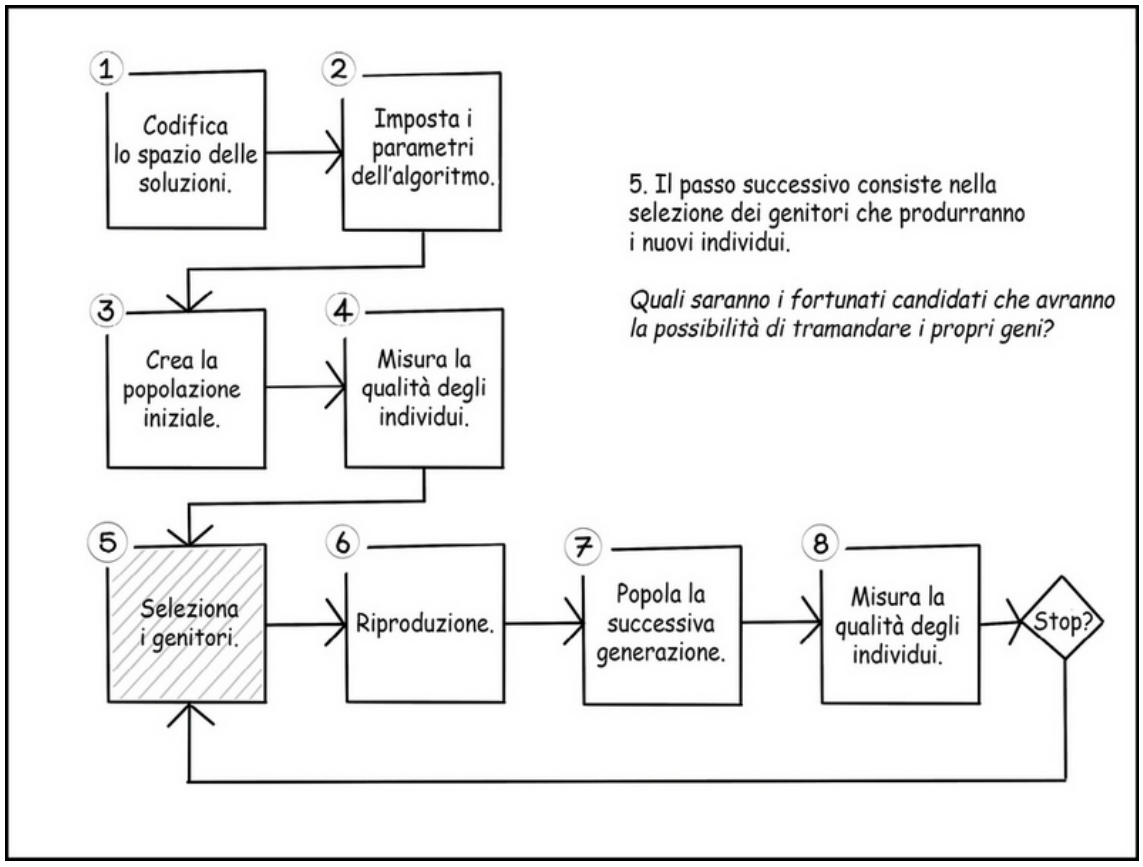


Figura 4.18 Selezione dei genitori.

Una tecnica molto utilizzata nella scelta dei genitori in base alla loro qualità è la *selezione a ruota della roulette*. Questa strategia offre ai diversi individui porzioni differenti di una ruota, in base alla loro qualità. La ruota viene "girata" e viene selezionato un individuo. Una maggiore qualità offre a un individuo una fetta più ampia della ruota. Questo processo viene ripetuto fino al raggiungimento del numero desiderato di genitori.

Calcolando le probabilità di 16 individui di diversa qualità, la ruota assegna una fetta a ciascuno. Poiché più individui hanno una qualità simile, la ruota avrà molte fette di dimensioni simili (Figura 4.19).

Il numero di genitori selezionati da utilizzare per la riproduzione di nuova prole è determinato dal numero totale previsto di figli, che è determinato dalla dimensione della popolazione desiderata per ciascuna generazione. Vengono selezionati due genitori e viene creata la prole. Questo processo si ripete selezionando ogni volta i genitori (con la possibilità che gli stessi individui siano genitori più di una volta) fino a quando non viene generato il numero desiderato di figli.

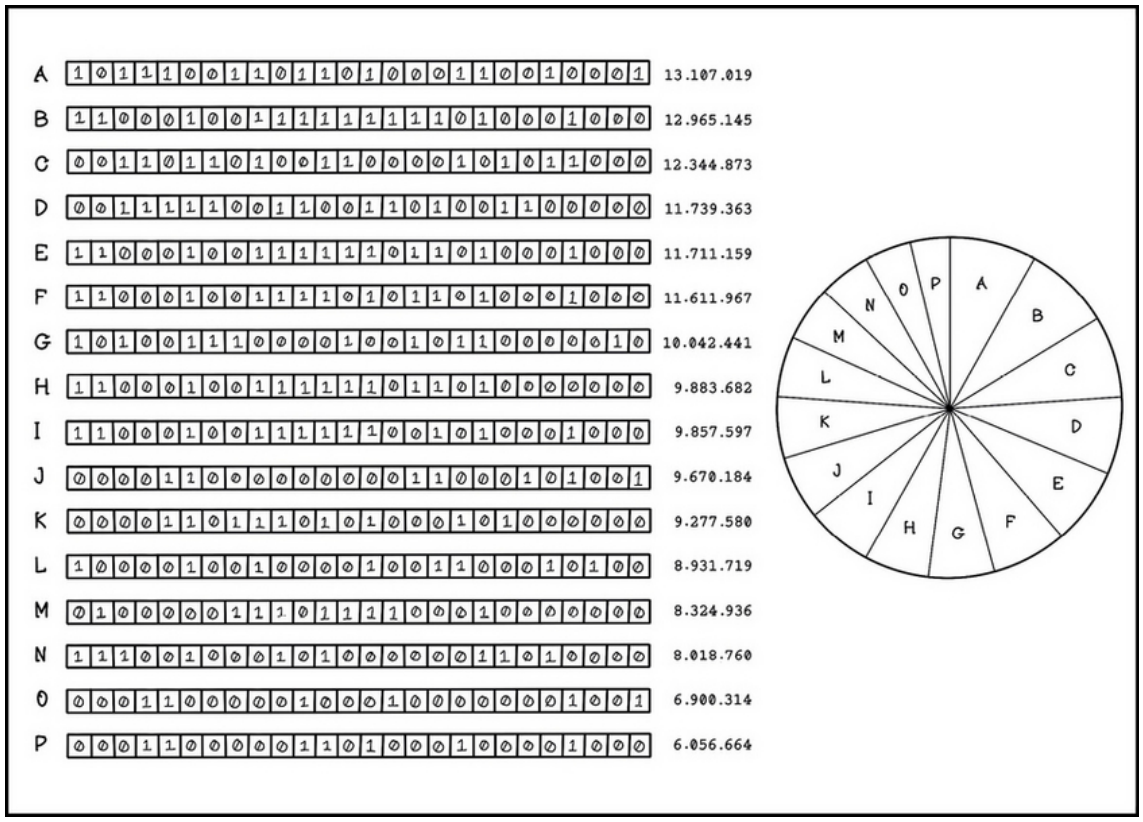


Figura 4.19 Determinazione della probabilità di selezione per ciascun individuo.

Due genitori possono riprodurre un solo figlio misto o due figli misti. Questo concetto sarà chiarito più avanti. Nel nostro esempio del problema dello zaino, gli individui con una maggiore qualità sono

quelli che riempiono lo zaino con il valore più elevato, rispettando il vincolo del limite di peso.

I modelli di popolazione sono modi per controllare la diversità della popolazione. Due modelli di popolazione che hanno i loro vantaggi e svantaggi sono quello a stato stazionario e generazionale.

Stato stazionario: sostituzione di una parte della popolazione a ogni generazione

Questo approccio alla gestione della popolazione non rappresenta un'alternativa alle altre strategie di selezione, ma uno schema che le utilizza. L'idea è che la maggior parte della popolazione venga mantenuta e che solo un piccolo gruppo di individui più deboli venga rimosso e sostituito con la nuova prole. Questo processo imita il ciclo della vita, in cui gli individui più deboli muoiono e i nuovi individui vengono creati attraverso la riproduzione. Se la popolazione fosse di 100 individui, una parte di essa sarebbe costituita da individui esistenti e una parte minore sarebbe costituita da nuovi individui creati per la riproduzione. Potrebbero quindi esserci 80 individui della generazione attuale più 20 nuovi individui.

Generazionale: sostituzione dell'intera popolazione a ogni generazione

Questo approccio alla gestione della popolazione è simile al modello a stato stazionario, ma non è un'alternativa alle strategie di selezione. Il modello generazionale crea un nuovo numero di individui pari alla dimensione della popolazione e sostituisce l'intera popolazione con la nuova prole. Se ci fossero 100 individui nella popolazione, ogni generazione produrrebbe 100 nuovi individui attraverso la

riproduzione. Stato stazionario e generazionale sono idee generali per configurare l'algoritmo.

Ruota della roulette: selezionare i genitori e i sopravvissuti

I cromosomi con punteggi di qualità più elevati hanno maggiori probabilità di essere selezionati, ma i cromosomi con punteggi di qualità più bassi hanno ancora una piccola possibilità di essere selezionati. Il termine *selezione a ruota della roulette* trae ispirazione dalla ruota della roulette di un casinò, divisa in fette. In genere, la ruota viene fatta girare e poi viene rilasciata una biglia. La fetta selezionata è quella nella quale si ferma la biglia quando la ruota smette di girare.

In base a questa analogia, i cromosomi vengono assegnati alle varie fette della ruota. I cromosomi con punteggi di qualità più elevati hanno fette più grandi della ruota, mentre i cromosomi con punteggi di qualità più bassi hanno fette più piccole. Un cromosoma viene poi selezionato in modo casuale, proprio come una pallina cade casualmente su una fetta.

Questa analogia è un esempio di selezione probabilistica. Ogni individuo ha una certa probabilità di essere selezionato, che può essere piccola o grande. La probabilità di selezione degli individui influenza la diversità della popolazione e i tassi di convergenza menzionati in precedenza in questo capitolo. La Figura 4.19, precedente, illustra questo concetto.

Pseudocodice

In primo luogo, è necessario determinare la probabilità di selezione per ogni individuo. Questa probabilità è calcolata per ogni individuo dividendo la sua qualità per la qualità totale della popolazione. A questo punto è possibile utilizzare la selezione a ruota della roulette. La "ruota" viene "girata" il numero di volte necessario a selezionare il numero desiderato di individui. Per ogni

selezione viene calcolato un numero decimale casuale compreso fra 0 e 1. Se la qualità di un individuo rientra in tale probabilità, questo viene selezionato. Possono essere utilizzati anche altri approcci probabilistici per determinare la probabilità di ogni individuo, inclusa la deviazione standard, in cui il valore di un individuo viene confrontato con il valore medio del gruppo:

```
set_probabilities_of_population(population):
  let total fitness equal the sum of fitness of the population
  for individual in population
            let the probability_of_selection of individual equal it's
fitness/total_fitness
roulette_wheel_selection(population, number_of_selections):
                      let           possible_probabilities           equal
set_probabilities_of_population(population)
  let slices equal empty array
  let total equal 0
  for i in range(0, number_of_selections):
      append [i, total, total + possible_probabilities[i]] to slices
      total += possible_probabilities[i]
  let spin equal random(0, 1)
  let result equal
```

Riproduzione degli individui dai genitori

Quando i genitori vengono selezionati, deve avvenire una riproduzione per creare la nuova prole dai genitori. In genere, due passaggi sono coinvolti nella creazione di figli a partire da due genitori. Il primo è quello di incrocio (*crossover*): mescolare parte del cromosoma del primo genitore con parte del cromosoma del secondo genitore, e viceversa. Questo processo produce due figli che contengono un mix dei geni dei loro genitori. Il secondo concetto è la *mutazione*: cambiare leggermente la progenie in modo casuale, per indurre variazioni nella popolazione (Figura 4.20).

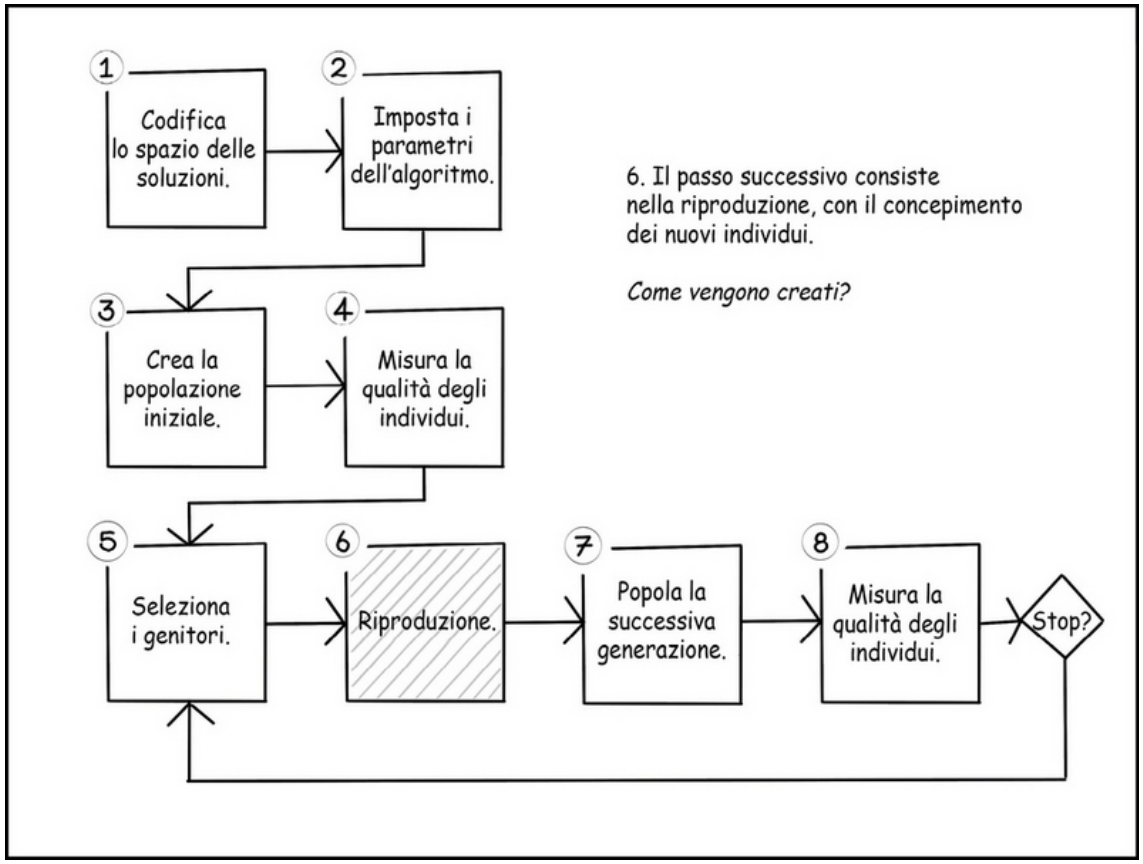


Figura 4.20 Riprodurre la prole.

Incrocio

Occorre miscelare i geni di due individui per creare uno o più individui discendenti. L'incrocio si ispira al concetto biologico di riproduzione. I figli traggono caratteri dai propri genitori, a seconda della strategia di incrocio utilizzata. La strategia di incrocio è fortemente influenzata dalla codifica utilizzata.

Incrocio a punto singolo: ereditare una parte da ciascun genitore

Viene selezionato un punto nella struttura del cromosoma. Quindi, facendo riferimento ai due genitori in questione, viene presa la prima parte dal primo genitore e la seconda parte dal secondo genitore.

Queste due parti combinate creano una nuova prole. Si può creare un secondo figlio utilizzando la prima parte del secondo genitore e la seconda parte del primo genitore.

L'incrocio a punto singolo è applicabile alla codifica binaria, alla codifica a ordine/permutazione e alla codifica a valori effettivi (Figura 4.21). Questi schemi di codifica saranno trattati nel Capitolo 5.

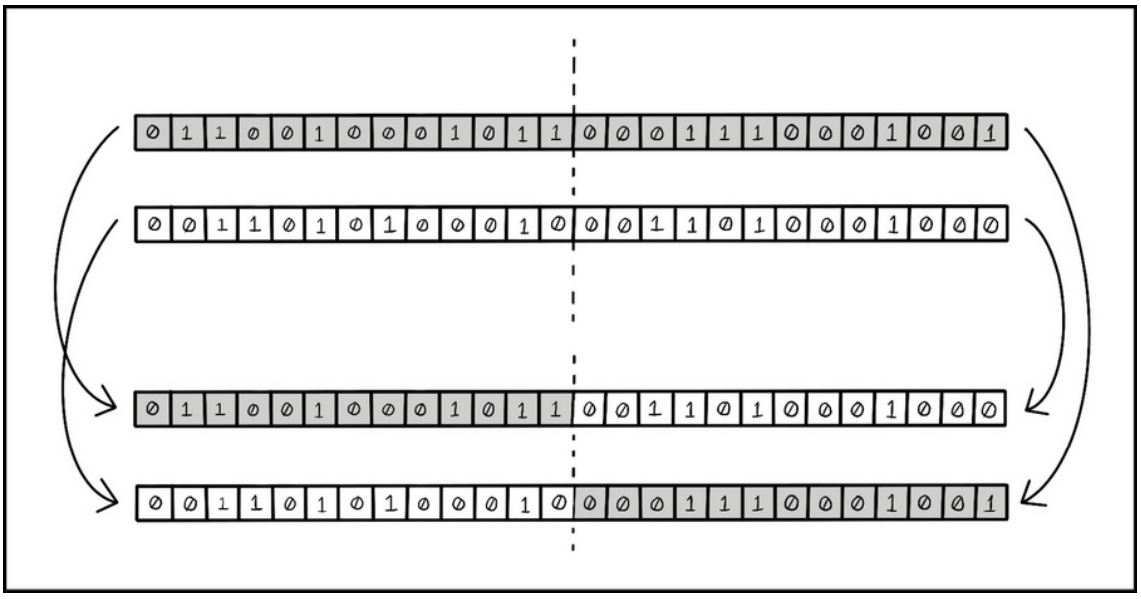


Figura 4.21 Incrocio a punto singolo.

Pseudocodice

Per creare due nuovi discendenti, viene creato un array per contenere i nuovi individui. Tutti i geni dall'indice 0 all'indice desiderato del genitore A vengono poi concatenati con tutti i geni dall'indice desiderato alla fine del cromosoma del genitore B, creando un individuo figlio. L'inverso crea un secondo individuo figlio:

```
one_point_crossover(parent_a, parent_b, xover_point)
  let children equal empty array
  let child_1 equal genes 0 to xover_point from parent_a plus         genes
xover_point to parent_b length from parent_b
  append child_1 to children
  let child_2 equal genes 0 to xover_point from parent_b plus         genes
xover_point to parent_a length from parent_a
  append child_2 to children
  return children
```

Incrocio a due punti: ereditare più parti da ciascun genitore

Vengono selezionati due punti nella struttura del cromosoma; poi, facendo riferimento ai due genitori in questione, si scelgono le parti in modo alternato, per formare un nuovo individuo completo. Questo processo è simile all'incrocio a punto singolo discusso in precedenza. In sostanza, il figlio è costituito dalla prima parte del primo genitore, dalla seconda parte del secondo genitore e dalla terza parte ancora del primo genitore. Anche in questo caso, può essere creato un secondo individuo utilizzando le parti inverse di ciascun genitore. L'incrocio a due punti è applicabile alla codifica binaria e alla codifica a valori effettivi (Figura 4.22).

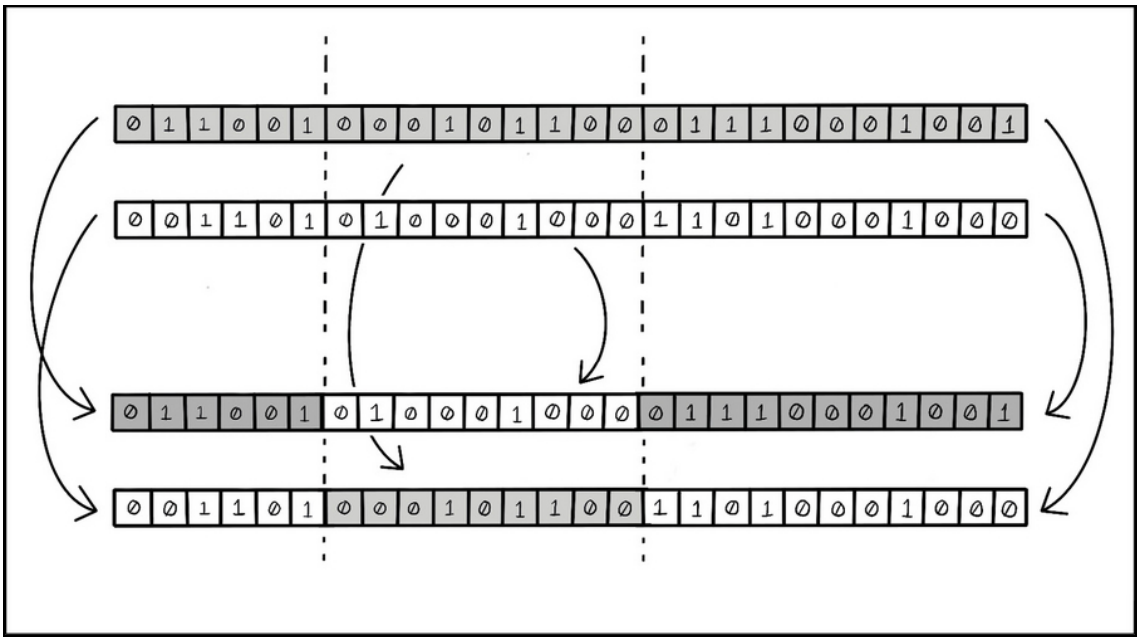


Figura 4.22 Incrocio a due punti.

Incrocio uniforme: ereditarietà di più parti da ciascun genitore

L'incrocio uniforme è un'evoluzione dell'incrocio a due punti. Nell'incrocio uniforme, viene creata una maschera che rappresenta quali geni di ciascun genitore verranno utilizzati per generare il figlio. Per creare un altro figlio può essere utilizzato il processo inverso. La maschera può essere generata ogni volta in modo casuale, per massimizzare la diversità. In generale, l'incrocio uniforme crea individui più differenti, perché gli attributi dei genitori vengono molto "mescolati" nella prole. L'incrocio uniforme è applicabile alla codifica binaria e alla codifica a valori effettivi (Figura 4.23).

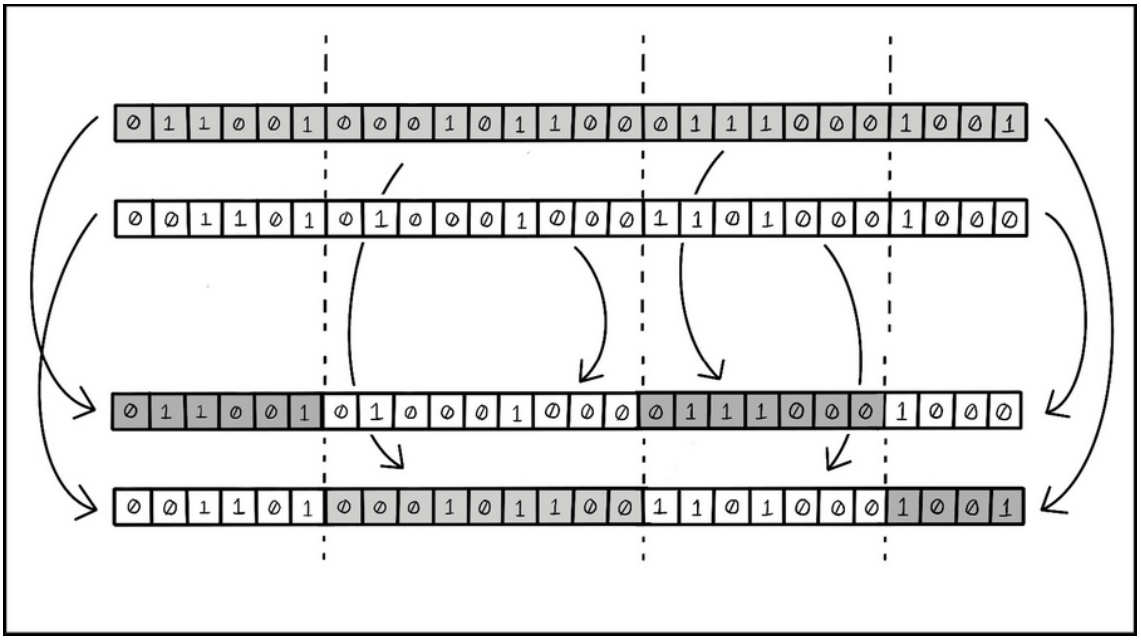


Figura 4.23 Incrocio uniforme.

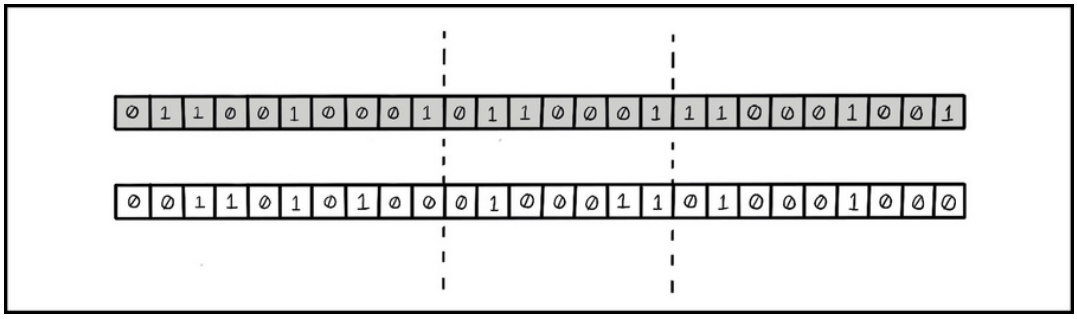
Mutazione

La mutazione comporta un leggero cambiamento degli individui figli per incoraggiare la diversità nella popolazione. Vengono utilizzati diversi approcci alla mutazione, in base alla natura del problema e al metodo di codifica.

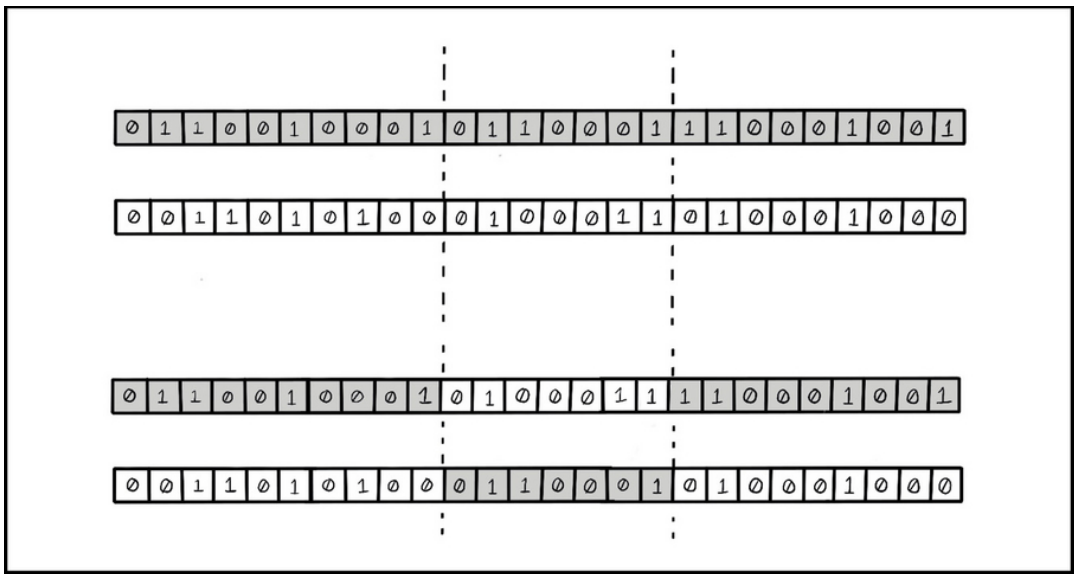
Un suo parametro è il tasso di mutazione: la probabilità che un cromosoma della progenie subisca una mutazione. Analogamente agli organismi viventi, alcuni cromosomi vengono mutati più di altri; una progenie non è quindi una combinazione diretta dei cromosomi dei genitori, ma contiene piccole differenze genetiche. La mutazione può essere fondamentale per favorire la diversità in una popolazione e impedire all'algoritmo di rimanere bloccato nelle migliori soluzioni locali.
Un alto tasso di mutazione può significare che gli individui hanno un'elevata probabilità di essere selezionati per essere mutati o che i geni nel cromosoma di un individuo hanno un'elevata probabilità di essere mutati, a seconda della strategia di mutazione scelta. Un elevato livello di mutazione crea più diversità, ma troppa diversità può comportare il deterioramento della convergenza verso buone soluzioni.

Esercizio: quale risultato genererebbe l'incrocio uniforme per questi

cromosomi?



Soluzione



Mutazione di stringhe di bit per la codifica binaria

Nella mutazione di stringhe di bit, un gene in un cromosoma con codifica binaria viene selezionato casualmente e modificato con un altro valore valido (Figura 4.24). Altri meccanismi di mutazione sono applicabili quando viene utilizzata una codifica non binaria. Il tema dei meccanismi di mutazione sarà esplorato nel Capitolo 5.

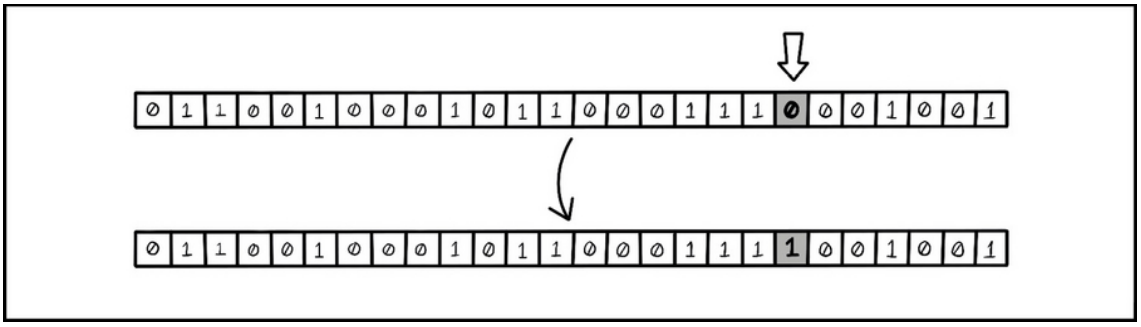


Figura 4.24 Mutazione di stringhe di bit.

Pseudocodice

Per mutare un singolo gene del cromosoma di un individuo, viene selezionato un indice genetico casuale. Se quel gene è a 1, viene cambiato in modo che divenga 0 (e viceversa):

```
mutate_individual(individual, chromosome_length):
  let random_index equal a random number between 0 and chromosome_length
  if gene at index random_index of individual is equal to 1:
      let gene at index random_index of individual equal 0
  else:
      let gene at index random_index of individual equal 1
  return individual
```

Mutazione flip-bit per la codifica binaria

Nella mutazione flip-bit, tutti i geni in un cromosoma con codifica binaria vengono invertiti. Al posto degli 1 si troveranno degli 0, e viceversa. Questo tipo di mutazione potrebbe degradare drasticamente le soluzioni, e di solito viene utilizzato quando nella popolazione deve essere costantemente introdotta una forte diversità (Figura 4.25).

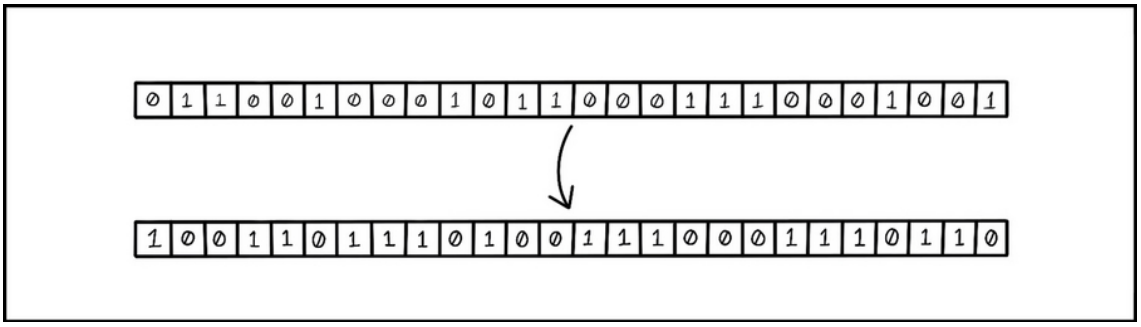


Figura 4.25 Mutazione flip-bit.

Popolare la generazione successiva

Dopo aver scelto gli individui da riproduzione e la prole è stata riprodotta, il passo successivo consiste nel selezionare quali individui vivranno fino alla generazione successiva. La dimensione della popolazione è solitamente fissa, e poiché sono stati introdotti nuovi individui attraverso la riproduzione, alcuni individui devono morire ed essere rimossi.

Può sembrare una buona idea tenere sempre i migliori individui ed eliminare i rimanenti. Questa strategia, tuttavia, potrebbe creare una stagnazione della diversità degli individui, se quelli che sopravvivono sono simili quanto a corredo genetico (Figura 4.26).

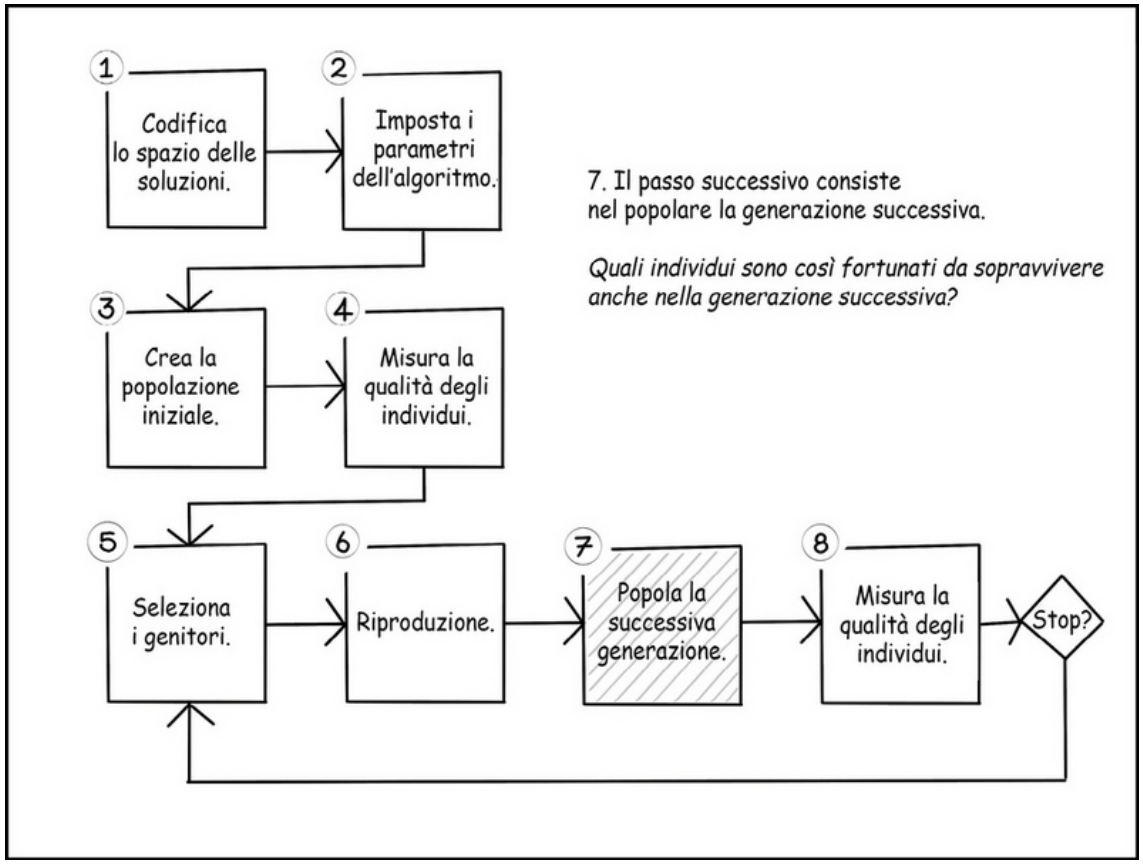


Figura 4.26 Popolare la generazione successiva.

Per determinare quali individui devono essere selezionati per entrare a far parte della popolazione della generazione successiva possono essere utilizzate le strategie di selezione menzionate di seguito.

Esplorare vs approfondire

L'esecuzione di un algoritmo genetico comporta sempre il raggiungimento di un equilibrio fra esplorazione e approfondimento. La situazione ideale prevede che vi sia una certa diversità negli individui e che la popolazione nel suo insieme cerchi soluzioni molto differenti nello spazio di ricerca; in tal modo, vengono sfruttati spazi di soluzioni locali più ricchi per trovare la soluzione più desiderabile. Il bello di questa situazione è che l'algoritmo da un lato *esplora* il più

possibile lo spazio di ricerca, *approfondendo* al contempo le soluzioni più forti a mano a mano che gli individui si evolvono (Figura 4.27).

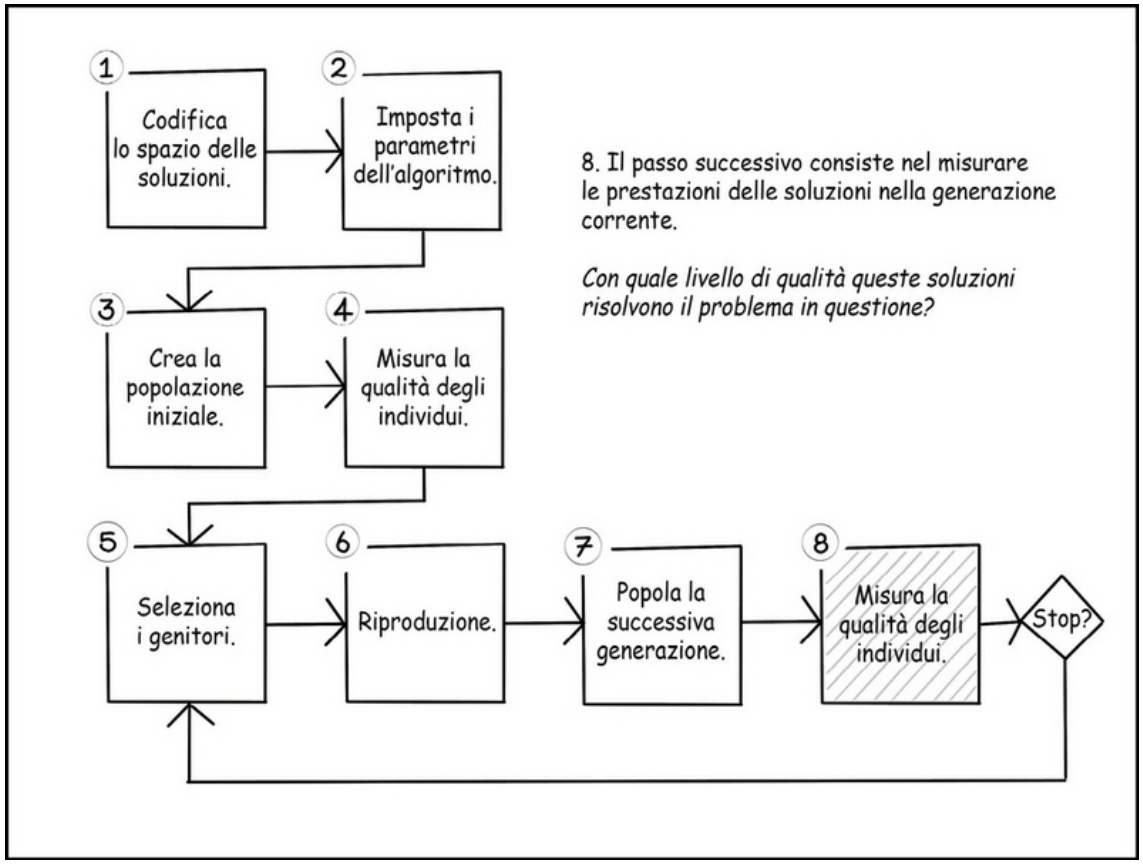


Figura 4.27 Misurazione della qualità degli individui.

Condizioni di arresto

Poiché un algoritmo genetico opera iterativamente nel trovare soluzioni migliori di generazione in generazione, è necessario stabilire una condizione di arresto; in caso contrario, l'algoritmo potrebbe procedere all'infinito nella sua ricerca. La *condizione di arresto* stabilisce il termine dell'algoritmo; come soluzione migliore viene scelto l'individuo più forte della popolazione di quella generazione.

La condizione di arresto più semplice è una *costante*, un valore che indica il numero fisso di generazioni per le quali l'algoritmo dovrà

procedere. Un altro approccio consiste nel fermarsi quando si raggiunge un certo livello di qualità. Questo metodo è utile quando è nota una qualità minima desiderata, e la soluzione è sconosciuta.

La *stagnazione* è un problema tipico degli algoritmi evolutivi: la popolazione produce soluzioni di qualità simile per diverse generazioni. Se una popolazione ristagna, si riduce la probabilità di generare soluzioni forti nelle generazioni future. Una condizione di arresto potrebbe valutare a ogni generazione i cambiamenti nella qualità del miglior individuo e, se la qualità cambia solo marginalmente, scegliere di fermare l'algoritmo.

Pseudocodice

I vari passaggi di un algoritmo genetico sono utilizzati in una funzione principale che delinea l'intero suo ciclo di vita. I parametri variabili comprendono la dimensione della popolazione, il numero di generazioni da calcolare e la capacità dello zaino per la funzione di valutazione della qualità, oltre alla posizione dell'incrocio e al tasso di mutazione:

```
run_ga(population_size, number_of_generations, knapsack_capacity):
  let best_global_fitness equal 0
  let global_population equal generate_initial_population(population_size)
  for generation in range(number_of_generations):
                                        let      current_best_fitness      equal
calculate_population_fitness(global_population,
            knapsack_capacity)
      if current_best_fitness is greater than best_global_fitness:
          let best_global_fitness equal current_best_fitness
          let  the_chosen  equal  roulette_wheel_selection(global_population,
population_size)
      let the_children equal reproduce_children(the_chosen)
      let the_children equal mutate_children(the_children)
                                          let      global_population      equal
merge_population_and_children(global_population,
             the_children)
```

Come accennato all'inizio di questo capitolo, il problema dello zaino potrebbe essere risolto utilizzando un approccio a forza bruta, che richiede la generazione e l'analisi di più di 60 milioni di combinazioni. Se confrontiamo gli algoritmi genetici che puntano a risolvere lo stesso problema, possiamo sperimentare una maggiore efficienza nel calcolo

se i parametri di esplorazione e approfondimento sono configurati correttamente. Ricordate che, in molti casi, un algoritmo genetico produce una soluzione “abbastanza buona”, che non è necessariamente la migliore possibile ma è comunque desiderabile. Ancora una volta, la scelta di utilizzare un algoritmo genetico per risolvere un determinato problema dipende dal contesto (Figura 4.28).

	Forza bruta	Algoritmo genetico
Iterazioni	2^26 = 67.108.864	10.000 - 100.000
Accuratezza	100%	100%
Tempo di calcolo	circa 7 minuti	circa 3 secondi
Valore migliore	13.692.887	13.692.887

Figura 4.28 Prestazioni del calcolo a forza bruta rispetto alle prestazioni dell’algoritmo genetico.

Configurazione dei parametri di un algoritmo genetico

Nel progettare e configurare un algoritmo genetico, è necessario prendere diverse decisioni che influenzano le prestazioni dell’algoritmo stesso. I ragionamenti sulle prestazioni ricadono in due aree: l’algoritmo dovrebbe sforzarsi di trovare buone soluzioni al problema; l’algoritmo dovrebbe funzionare in modo efficiente dal punto di vista computazionale. Sarebbe inutile progettare un algoritmo genetico per risolvere un problema se la soluzione fosse computazionalmente più costosa rispetto ad altre tecniche tradizionali. L’approccio utilizzato, la

funzione di calcolo della qualità utilizzata e gli altri parametri dell'algoritmo influenzano entrambi i tipi di prestazioni nel raggiungimento di una buona soluzione con il minor numero possibile di calcoli. Ecco alcuni parametri da considerare.

- *Codifica dei cromosomi*: il metodo di codifica dei cromosomi richiede una riflessione, per garantire che sia applicabile al problema e che le soluzioni potenziali convergano verso i massimi globali. Lo schema di codifica è fondamentale per il successo dell'algoritmo.
- *Dimensione della popolazione*: la dimensione della popolazione è configurabile. Una popolazione più numerosa incoraggia una maggiore diversità nelle soluzioni possibili. Popolazioni più grandi, tuttavia, richiedono un maggior numero di calcoli a ogni generazione. A volte, una popolazione più numerosa compensa la necessità di mutazioni, che si traduce in maggiore diversità iniziale ma minore diversità durante le generazioni. Un approccio valido consiste nell'iniziare con una popolazione più piccola e farla crescere in base alle prestazioni.
- *Inizializzazione della popolazione*: sebbene gli individui di una popolazione vengano inizializzati in modo casuale, è importante garantire che le soluzioni siano valide per ottimizzare il calcolo dell'algoritmo genetico e inizializzare gli individui con i vincoli corretti.
- *Numero di discendenti*: è possibile configurare il numero di discendenti creati in ciascuna generazione. Dato che dopo la riproduzione parte della popolazione viene uccisa per garantire che la dimensione della popolazione sia fissa, più figli danno maggiore diversità, ma c'è il rischio che le buone soluzioni vengano "uccise" per ospitare la nuova prole. Se la popolazione è dinamica, la dimensione della popolazione può cambiare dopo

ogni generazione, ma questo approccio richiede un maggior numero di parametri da configurare e controllare.

- *Metodo di selezione dei genitori*: è possibile configurare il metodo di selezione utilizzato per scegliere i genitori. Questo metodo deve essere basato sul problema e sull'esplorabilità desiderata rispetto all'utilizzabilità.
- *Metodo di incrocio*: il metodo di incrocio è associato al metodo di codifica utilizzato, ma può essere configurato per incoraggiare o scoraggiare la diversità nella popolazione. Gli individui discendenti devono ancora fornire una soluzione valida.
- *Tasso di mutazione*: il tasso di mutazione è un altro parametro configurabile che induce una maggiore diversità nella prole e nelle soluzioni potenziali. Un tasso di mutazione più elevato crea più diversità, ma troppa diversità può deteriorare la convergenza verso la soluzione. Il tasso di mutazione può cambiare nel tempo, per creare maggiore diversità nelle generazioni iniziali e meno nelle generazioni successive. Questo risultato può essere descritto in termini di esplorazione e poi approfondimento.
- *Metodo di mutazione*: il metodo di mutazione, come il metodo di incrocio, dipende dal metodo di codifica utilizzato. Un attributo importante del metodo di mutazione è che deve comunque fornire una soluzione valida o deve ricevere un pessimo punteggio di qualità.
- *Metodi di selezione della generazione*: proprio come il metodo di selezione utilizzato per scegliere i genitori, anche il metodo di selezione della generazione deve scegliere gli individui che sopravvivranno alla generazione. A seconda del metodo di selezione utilizzato, l'algoritmo potrebbe convergere troppo rapidamente e ristagnare o esplorare troppo a lungo.

- *Condizione di arresto*: la condizione di arresto per l'algoritmo deve essere sensata per il problema e il risultato desiderato. La complessità computazionale e il tempo sono le preoccupazioni principali per la condizione di arresto.

Casi d'uso per gli algoritmi evolutivi

Gli algoritmi evolutivi hanno un'ampia varietà di utilizzi. Alcuni algoritmi affrontano problemi isolati; altri combinano gli algoritmi evolutivi con altre tecniche per creare nuovi approcci alla risoluzione di problemi difficili, come i seguenti.

- *Previsione del comportamento degli investitori nel mercato azionario*: i consumatori che investono prendono decisioni ogni giorno sull'opportunità di acquistare determinate azioni, mantenere quelle che hanno o vendere le proprie azioni. Le sequenze di queste operazioni possono essere sviluppate e mappate sui risultati del portafoglio di un investitore. Gli istituti finanziari possono utilizzare questa conoscenza per fornire servizi e indicazioni proattive ai propri clienti.
- *Selezione delle funzionalità nel machine learning*: il machine learning sarà trattato nel Capitolo 8, ma un aspetto chiave del machine learning è che, dato un certo numero di caratteristiche di qualcosa, possiamo determinare come classificarlo. Se consideriamo le case, potremmo trovare molti attributi relativi a esse, come l'età, il materiale di costruzione, le dimensioni, il colore e la posizione. Ma per prevedere il valore di mercato, forse contano solo l'età, le dimensioni e la posizione. Un algoritmo genetico può scoprire quali caratteristiche contano di più.
- *Violazione del codice e cifratura*: una *cifratura* è una codifica di un messaggio e spesso viene utilizzata per rendere indecifrabili le

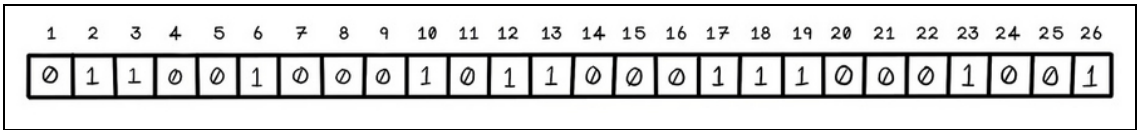
informazioni. Chi non sappia decifrare un messaggio, non potrà leggerlo. Gli algoritmi evolutivi possono generare molte possibilità per modificare il messaggio cifrato fino a scoprire il suo significato.

Il Capitolo 5 approfondisce alcuni concetti avanzati degli algoritmi genetici, per adattarli a diversi spazi dei problemi. Esploreremo diverse tecniche di codifica, come l'incrocio, la mutazione e la selezione, oltre a scoprire alternative efficaci.

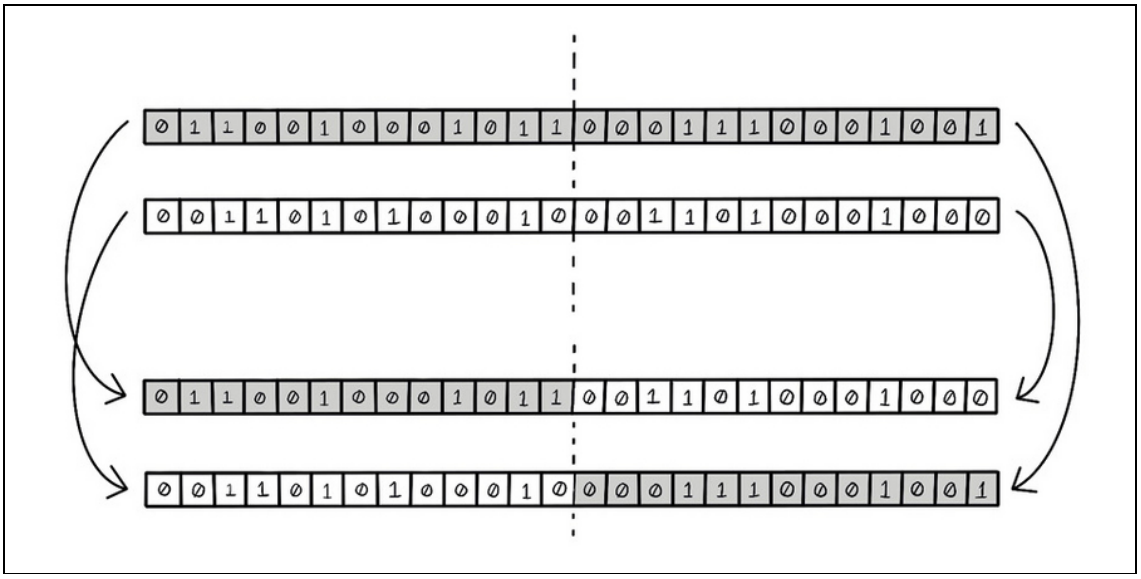
Riepilogo

Gli algoritmi genetici utilizzano intelligentemente la casualità per trovare rapidamente le migliori soluzioni.

- La codifica è fondamentale per l'algoritmo.

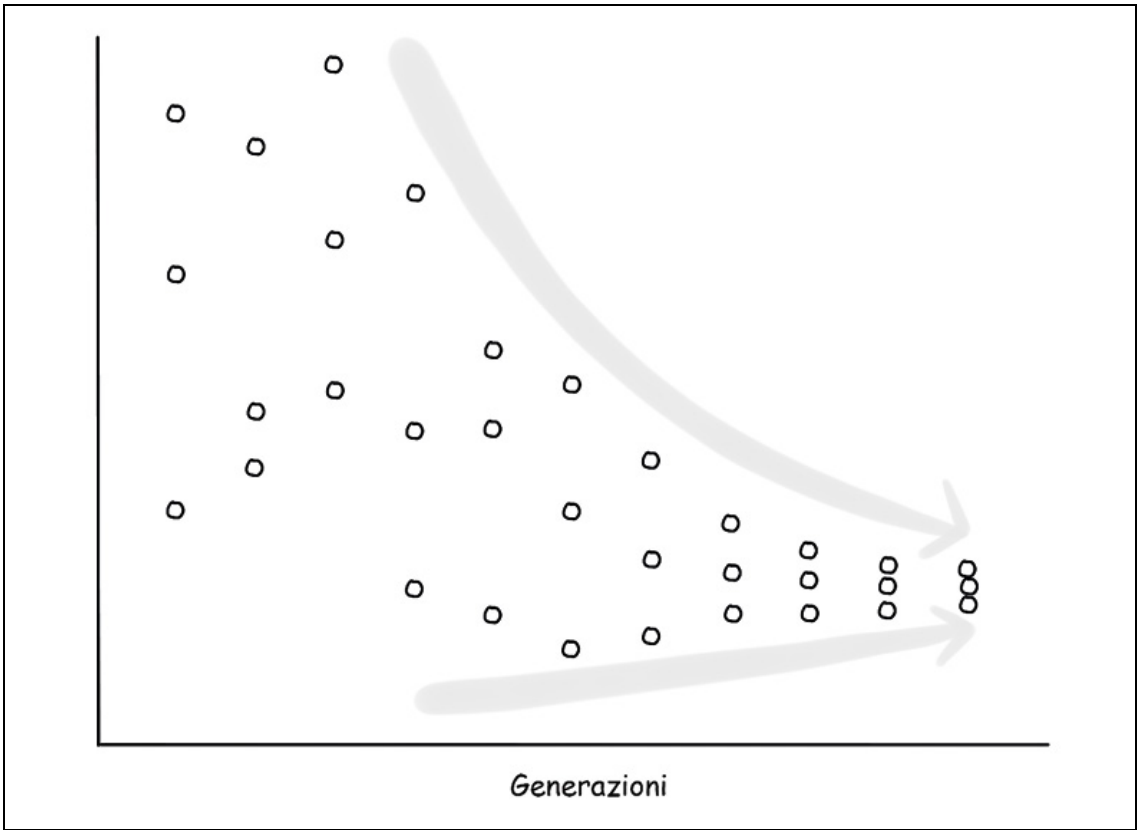


- La funzione di calcolo della qualità è importante per trovare le buone soluzioni al problema in questione.
  L'incrocio punta a ottenere a ogni generazione soluzioni sempre migliori.



- La selezione favorisce gli individui più forti, ma dà anche a quelli più deboli un'opportunità per riprodursi e produrre buone soluzioni in futuro.

- Esplorazione all'inizio e approfondimento verso la fine.


Generazioni

Approcci evolutivi avanzati

NOTA
Il Capitolo 4 è un prerequisito per questo capitolo.

Ciclo di vita dell'algoritmo evolutivo

Nel Capitolo 4 ho delineato il ciclo di vita generale di un algoritmo genetico. In questo capitolo consideriamo altri problemi che possono essere risolti con un algoritmo genetico, perché alcuni degli approcci trattati finora non funzionano e occorre ricorrere ad approcci alternativi.

Come promemoria, il ciclo di vita generale di un algoritmo genetico è il seguente.

- *Creazione di una popolazione*: si tratta di una popolazione casuale di soluzioni potenziali.
- *Misurazione della qualità degli individui della popolazione*: occorre determinare quanto è valida ogni soluzione. Questo compito viene eseguito utilizzando una funzione di calcolo della qualità che assegna un punteggio alle soluzioni.
- *Selezione dei genitori in base alla loro qualità*: si tratta delle coppie di genitori che potranno generare la prole.
- *Riproduzione di nuovi individui dai genitori*: i genitori generano figli mescolando le informazioni genetiche e applicando lievi mutazioni alla prole.

- *Popolamento della generazione successiva*: occorre selezionare gli individui (fra genitori e figli) che sopravvivranno alla generazione successiva.

Tenete presente il flusso del ciclo di vita (raffigurato nella Figura 5.1) mentre seguite questo capitolo.

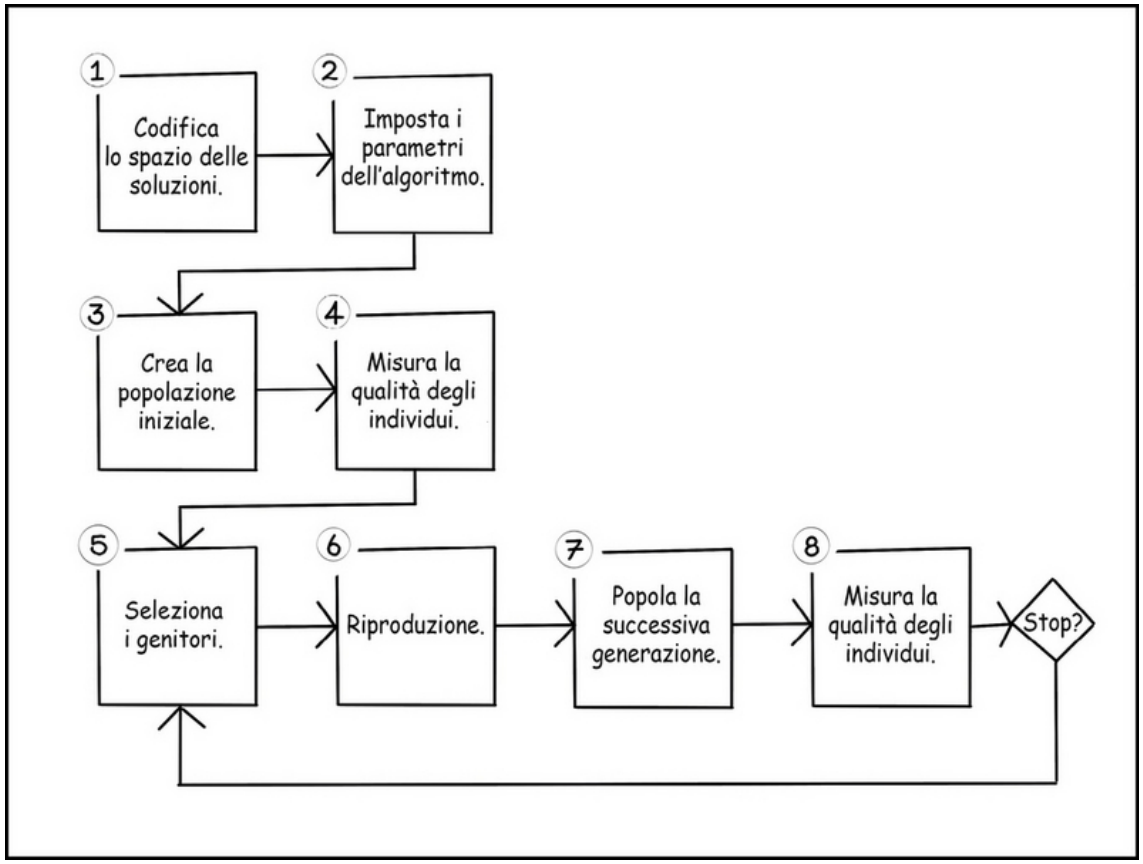


Figura 5.1 Ciclo di vita dell'algoritmo genetico.

Questo capitolo inizia esplorando nuove strategie di selezione; questi approcci possono essere applicati a qualsiasi algoritmo genetico. Di seguito trovate tre scenari che rappresentano altrettanti aggiustamenti del problema dello zaino (Capitolo 4) per chiarire l'utilità degli approcci alternativi di codifica, incrocio e mutazione (Figura 5.2).

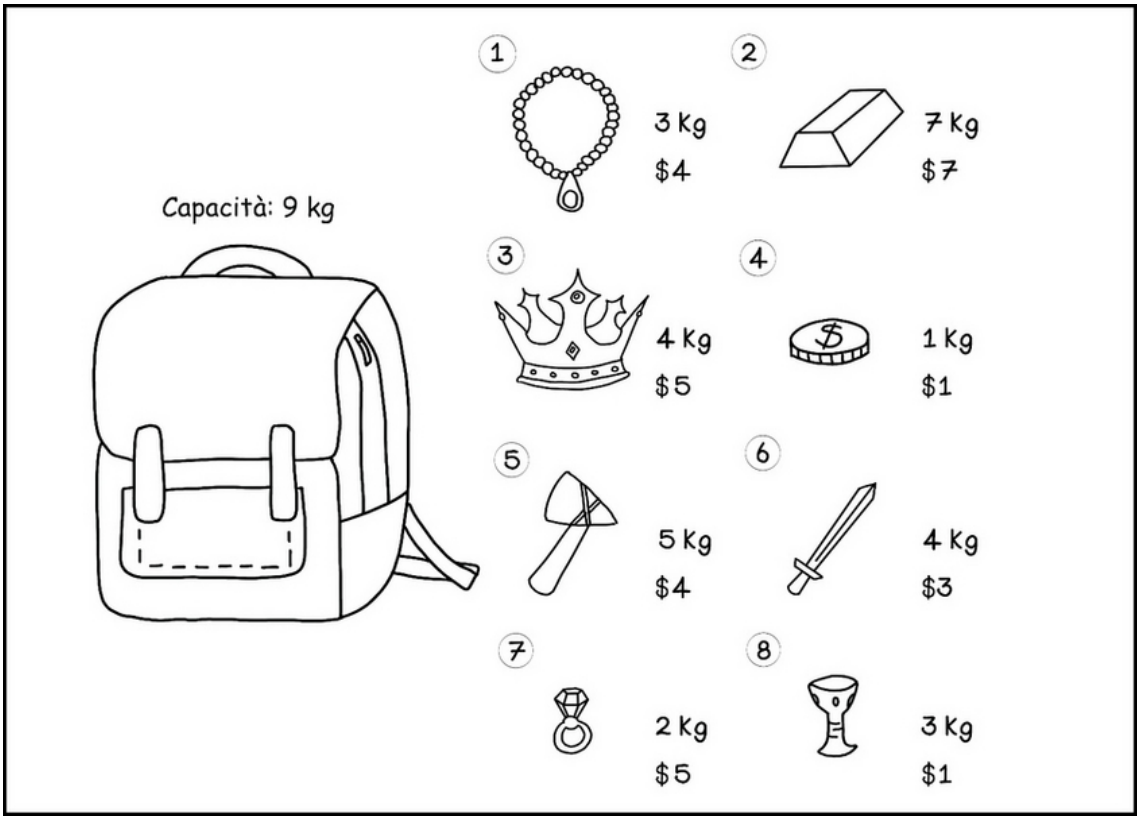


Figura 5.2 L'esempio del problema dello zaino.

Strategie alternative di selezione

Nel Capitolo 4, abbiamo esplorato una strategia di selezione: la selezione a roulette, che è uno dei metodi più semplici per selezionare gli individui. Le seguenti tre strategie di selezione aiutano a contenere i problemi di tale metodo di selezione; ognuno offre vantaggi e svantaggi, che hanno effetti sulla diversità della popolazione, che a sua volta influisce sul fatto che venga trovata una soluzione ottimale.

Selezione per grado: distribuire i punteggi

Un problema della selezione a ruota della roulette consiste nelle piccole differenze di grandezza attribuite in base alla qualità dei

cromosomi. Vorremmo distorcere la selezione verso la scelta di individui con punteggi di qualità elevati. Questo è vero che riduce la diversità della popolazione. Una maggiore diversità offrirebbe una maggiore esplorazione dello spazio di ricerca, ma un eccesso può far sì che la ricerca di soluzioni ottimali richieda troppe generazioni.

La selezione distribuita per grado ha lo scopo di risolvere questo problema, stilando una "classifica" degli individui in base alla loro qualità e poi utilizzando tale classifica per calcolare la dimensione della sua "fetta sulla ruota". Nel problema dello zaino, questo valore è un numero compreso fra 1 e 16, perché stiamo scegliendo fra 16 individui. Su una normale ruota, gli individui migliori non si distinguono molto da quelli "non così ottimali". Utilizzando il grado, invece, ogni individuo ha una probabilità di essere selezionato che dipende maggiormente dalla sua qualità. Nel classificare 16 individui, la ruota appare così un po' differente (Figura 5.3).

La Figura 5.4 mette a confronto la selezione a ruota della roulette e la selezione per grado. È evidente che la selezione per grado offre alle soluzioni con prestazioni migliori maggiori possibilità di selezione.

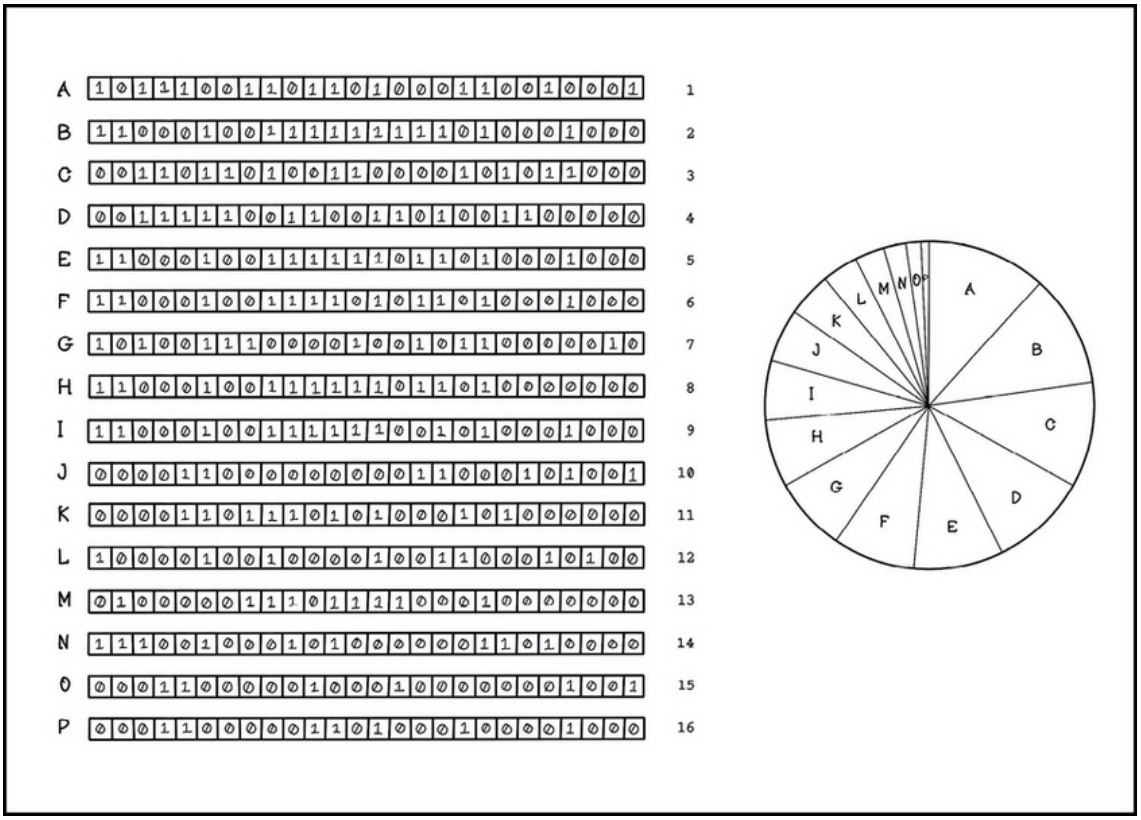


Figura 5.3 Esempio di selezione per grado.

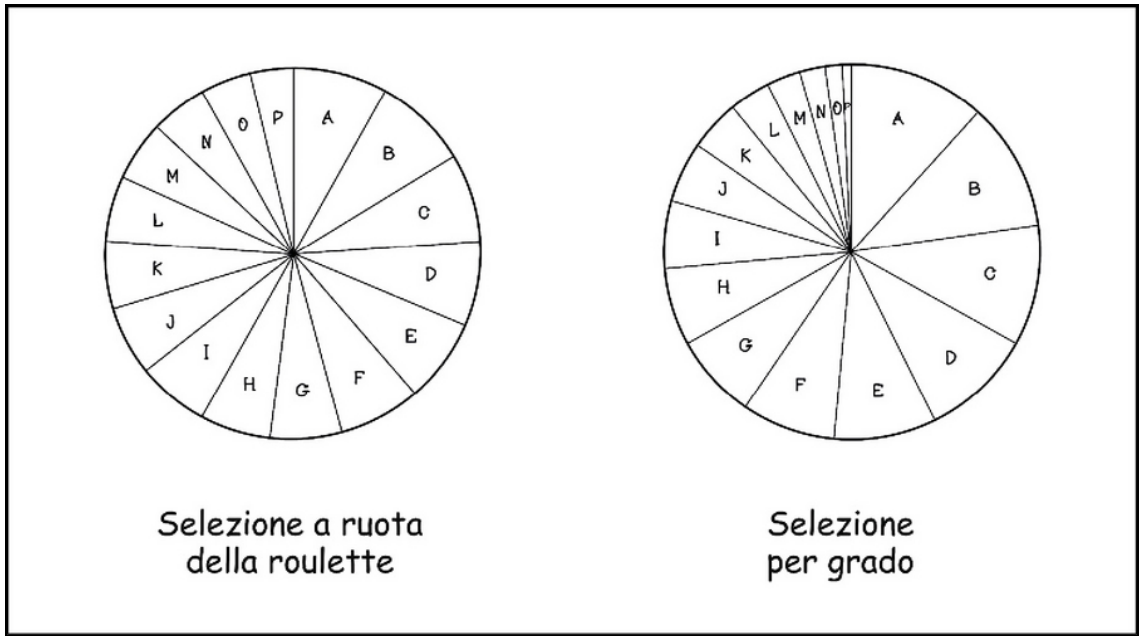


Figura 5.4 Selezione a ruota della roulette rispetto alla selezione per grado.

Selezione a torneo: facciamoli competere fra loro

La selezione a torneo mette i cromosomi l'uno contro l'altro: sceglie casualmente un determinato numero di individui dalla popolazione e li inserisce in un gruppo. Questo processo viene eseguito per formare un numero predeterminato di gruppi. Viene selezionato l'individuo con il punteggio di qualità più alto in ciascun rispettivo gruppo. Più grande è il gruppo, meno diversità si ottiene, perché viene selezionato un solo individuo per gruppo. L'effettivo punteggio di qualità di ogni individuo non è quindi il fattore chiave nella selezione degli individui.

Se i 16 individui vengono assegnati a 4 gruppi, selezionando un solo individuo da ciascun gruppo si ottengono i 4 individui più forti di quei gruppi. I 4 individui vincenti possono essere accoppiati per riprodursi (Figura 5.5).

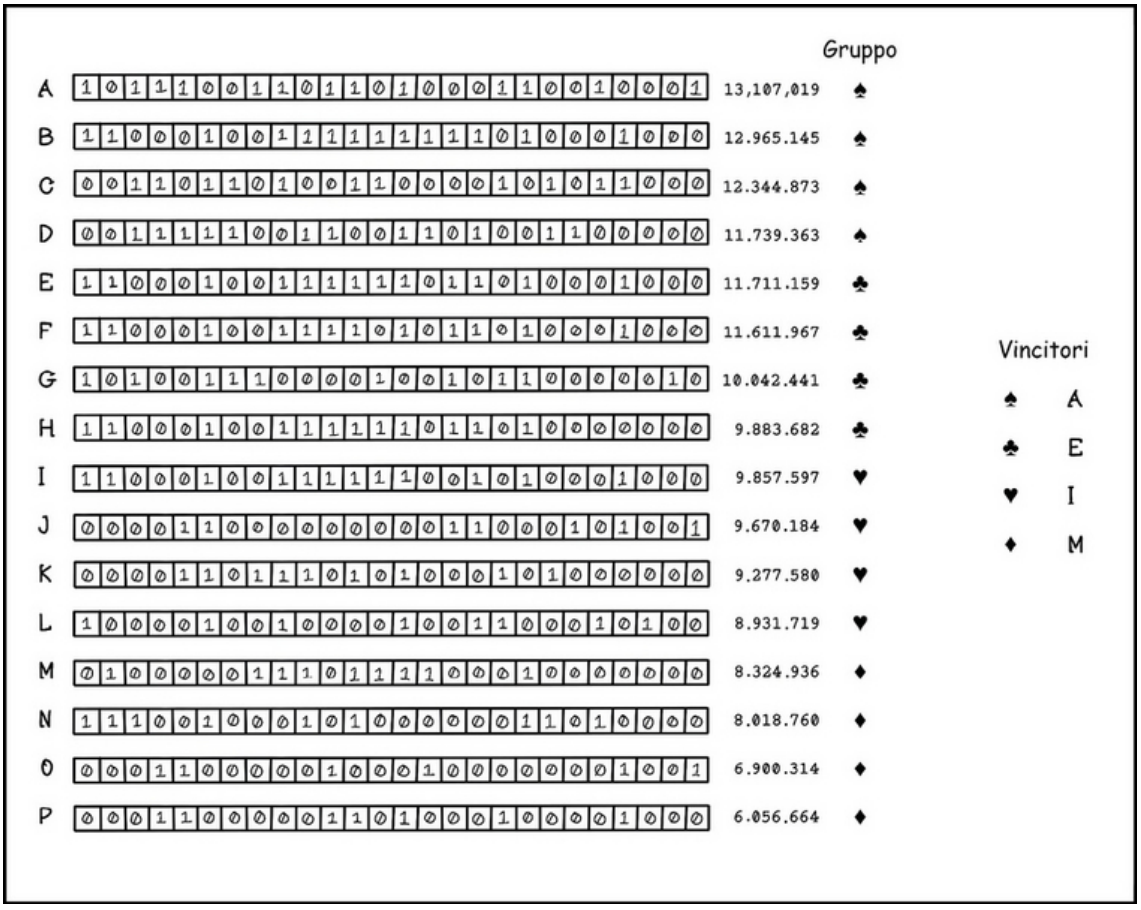


Figura 5.5 Esempio di selezione a torneo.

Selezione elitaria: scegli solo il meglio

L'approccio elitario seleziona i migliori individui della popolazione. Questo è utile per tenere gli individui con ottime prestazioni ed evitare il rischio che questi vengano persi attraverso altri metodi di selezione. Lo svantaggio è che può capitare che la popolazione rientri nello spazio delle migliori soluzioni locali e non essere mai abbastanza diversificata da trovare le migliori soluzioni globali.

L'elitarismo è spesso usato in combinazione con la selezione a ruota della roulette, la selezione per grado e la selezione a torneo. L'idea è che vengano selezionati per riprodursi certi individui di élite, e il resto

della popolazione venga riempito di altri individui, scelti in base a una delle altre strategie di selezione (Figura 5.6).

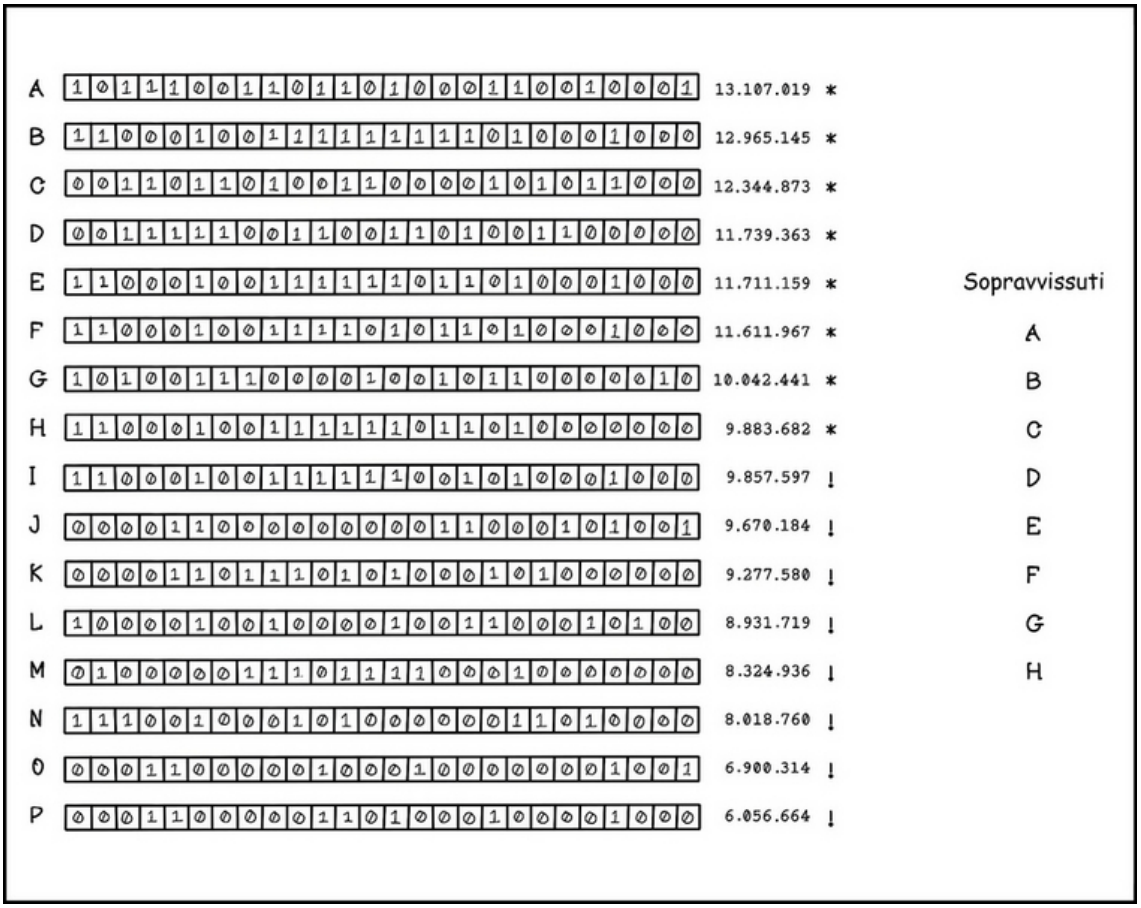


Figura 5.6 Esempio di selezione elitaria.

Il Capitolo 4 ha esplorato un problema in cui l'inserimento o meno di oggetti nello zaino era importante. Altri spazi dei problemi possono richiedere una codifica differente, perché una codifica puramente binaria non sarebbe sensata. I tre paragrafi seguenti descrivono questi scenari.

Codifica a valori effettivi: lavorare con i veri valori disponibili

Considerate una variante del problema dello zaino. Resta il problema di scegliere gli oggetti più preziosi per riempire la capacità di peso dello zaino. Ma la scelta riguarda più unità di ogni articolo. Come mostra la Tabella 5.1, i pesi e i valori rimangono gli stessi del dataset originale, ma è inclusa una quantità di ciascun articolo. Con questo leggero aggiustamento, è possibile combinare una miriade di nuove soluzioni, una o più delle quali potrebbero essere ottimali, perché un elemento può essere selezionato più volte. La codifica binaria non funzionerebbe in questo scenario. La codifica a valori effettivi è più adatta a rappresentare lo stato delle soluzioni potenziali.

Tabella 5.1 Portata zaino: 6.404.180 kg.

ID articolo	Nome articolo	Peso (kg)	Valore ($)	Quantità
1	Ascia	32.252	68.674	19
2	Moneta di bronzo	225.790	471.010	14
3	Corona	468.164	944.620	2
4	Statua di diamanti	489.494	962.094	9
5	Cintura di smeraldi	35.384	78.344	11
6	Fossile	265.590	579.152	6
7	Moneta d’oro	497.911	902.698	4
8	Elmo	800.493	1.686.515	10
9	Inchiostro	823.576	1.688.691	7
10	Portagioie	552.202	1.056.157	3

11	Coltello	323.618	677.562	5
12	Spada lunga	382.846	833.132	13
13	Maschera	44.676	99.192	15
14	Collana	169.738	376.418	8
15	Distintivo di opale	610.876	1.253.986	4
16	Perla	854.190	1.853.562	9
17	Faretra	671.123	1.320.297	12
18	Rubino	698.180	1.301.637	17
19	Bracciale d’argento	446.517	859.835	16
20	Orologio	909.620	1.677.534	7
21	Uniforme	904.818	1.910.501	6
22	Pozione di veleno	730.061	1.528.646	9
23	Sciarpa di lana	931.932	1.827.477	3
24	Balestra	952.360	2.068.204	1
25	Annale	926.023	1.746.556	7
26	Coppa di zinco	978.724	2.100.851	2

Funzionamento della codifica a valori effettivi

La codifica a valori effettivi rappresenta un gene in termini di valori numerici, stringhe o simboli, ed esprime le soluzioni potenziali nello stato naturale del problema. Questa codifica viene utilizzata quando le soluzioni potenziali contengono valori che non possono essere codificati facilmente con la codifica binaria. Per esempio, poiché in questo caso nello zaino può essere inserito più di un articolo, pertanto

ogni indice relativo a un articolo non può indicare solo se l'articolo è incluso; deve indicare anche la quantità di quell'articolo inserita nello zaino (Figura 5.7).

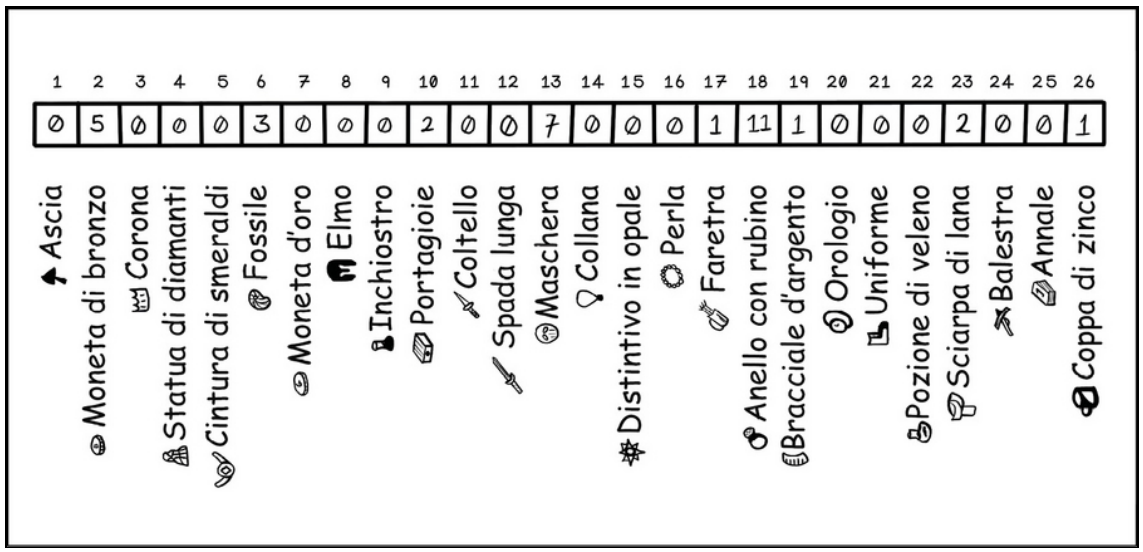


Figura 5.7 Esempio di codifica a valori effettivi.

Poiché lo schema di codifica è cambiato, diventano disponibili nuove opzioni di incrocio e mutazione. Gli approcci all'incrocio discussi per la codifica binaria rimangono ancora opzioni valide, ma la mutazione dovrebbe essere affrontata in modo differente.

Incrocio aritmetico: riproduzione matematica

L'incrocio aritmetico implica un'operazione aritmetica da calcolare utilizzando i genitori come variabili dell'espressione. Il risultato dell'applicazione di un'operazione aritmetica sui due genitori è la nuova progenie. Quando usiamo questa strategia con la codifica binaria, è importante assicurarsi che il risultato dell'operazione sia ancora un cromosoma valido. L'incrocio aritmetico è applicabile alla codifica binaria e alla codifica a valori effettivi (Figura 5.8).

NOTA

Attenzione: questo approccio può creare discendenti molto differenti, il che può essere problematico.

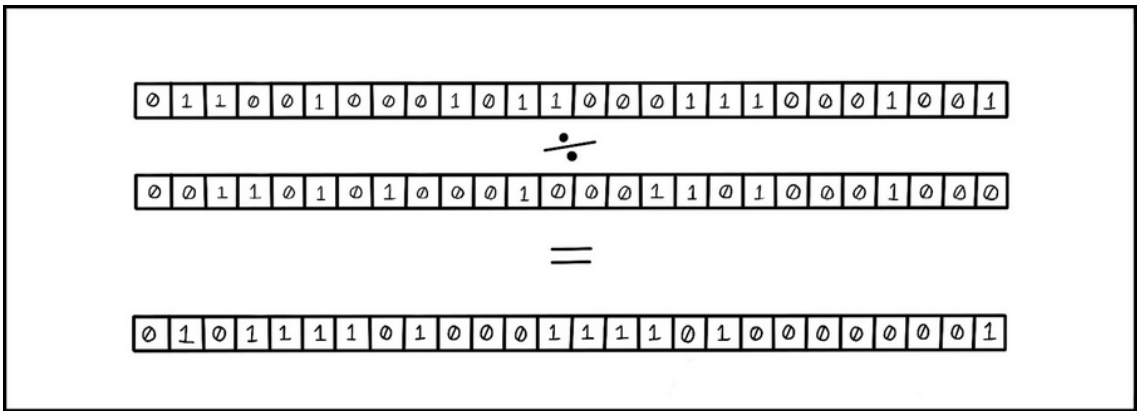


Figura 5.8 Esempio di incrocio aritmetico.

Mutazione a valore limite

In questo caso, un gene selezionato casualmente da un cromosoma codificato a valori effettivi viene impostato casualmente su un valore limite (inferiore o superiore). Dati 26 geni in un cromosoma, viene selezionato un indice casuale, e il suo valore viene impostato al minimo o al massimo. Nella Figura 5.9, il valore originale è 0 e sarà portato a 6, che è il massimo per quell'elemento. Il minimo e il massimo possono essere gli stessi per tutti gli indici o specificati in modo univoco per ciascun indice, se la conoscenza del problema informa tale decisione. Questo approccio tenta di esplorare l'impatto dei singoli geni sul cromosoma.

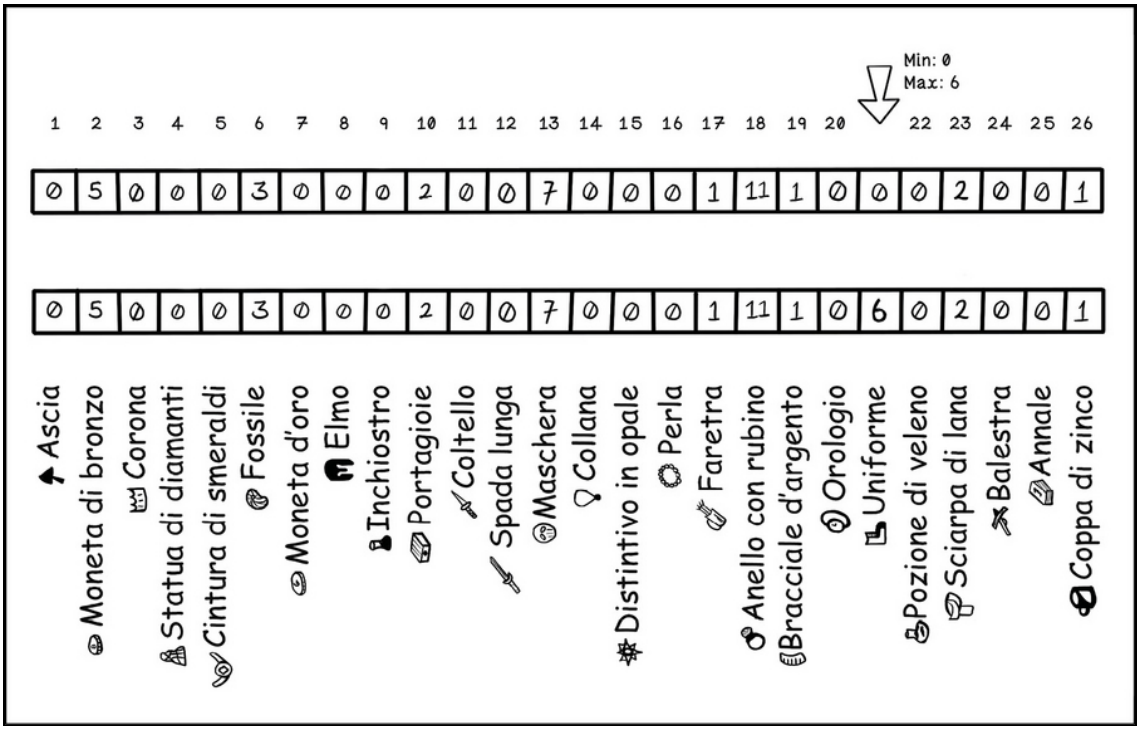


Figura 5.9 Esempio di mutazione a valore limite.

Mutazione aritmetica

In questo caso, un gene selezionato casualmente in un cromosoma codificato a valori effettivi viene modificato aggiungendo o sottraendo una piccola quantità. Notate che sebbene l'esempio nella Figura 5.10 includa numeri interi, i numeri potrebbero essere decimali, o anche frazionari.

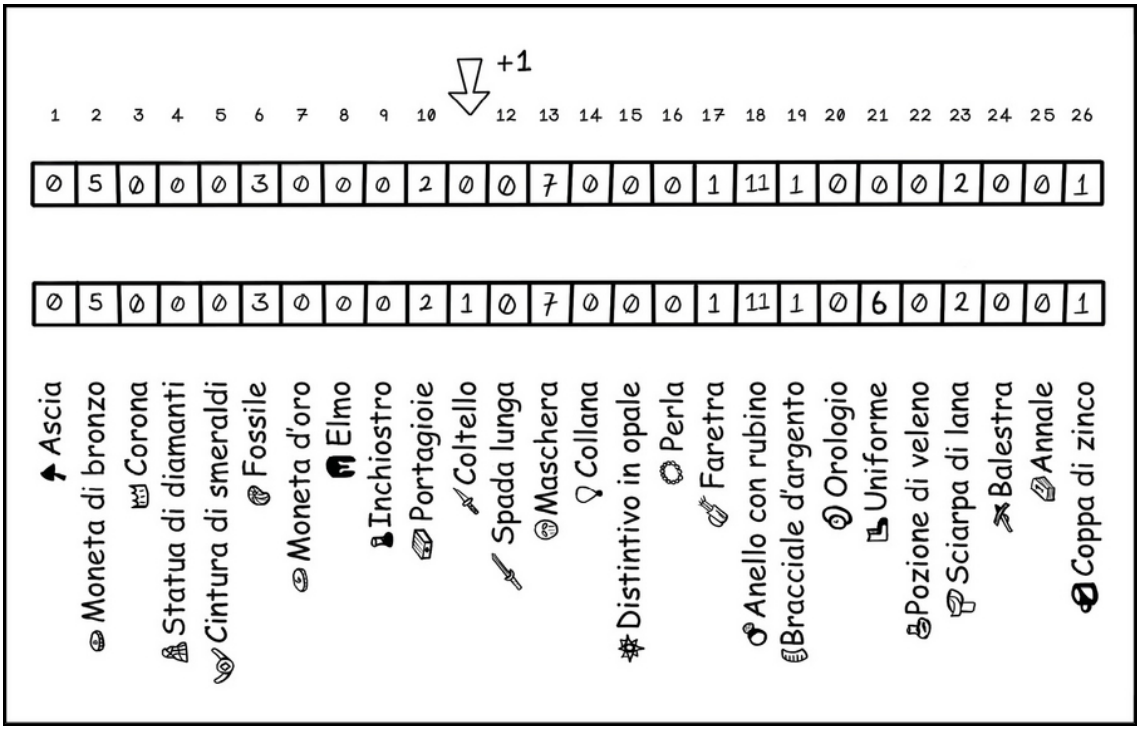


Figura 5.10 Esempio di mutazione aritmetica.

Codifica a ordine: lavorare con le sequenze

Abbiamo sempre gli stessi oggetti del problema dello zaino. Ma non determineremo gli oggetti che entreranno in uno zaino: gli articoli devono essere lavorati in un impianto in cui ogni articolo viene scomposto per estrarne il materiale di partenza.

La moneta d'oro, il braccialetto d'argento e altri oggetti verranno pertanto fusi per estrarre i composti di partenza. In questo scenario, gli elementi non vengono selezionati per essere inclusi, ma vengono inclusi tutti.

Per rendere le cose più interessanti, l'impianto richiede un tasso di estrazione costante, dato il tempo di estrazione e il valore dell'oggetto.

Si presume che il valore del materiale ricavato sia più o meno uguale al valore dell'articolo. Il problema si trasforma così in un problema di ordinamento. In quale ordine devono essere elaborati gli articoli per mantenere costante il valore? La Tabella 5.2 descrive gli articoli con i rispettivi tempi di estrazione.

Tabella 5.2 Valore ricavato all'ora: 600.000.

ID articolo	Nome articolo	Peso (kg)	Valore ($)	Tempo di estrazione
1	Ascia	32.252	68.674	60
2	Moneta di bronzo	225.790	471.010	30
3	Corona	468.164	944.620	45
4	Statua di diamanti	489.494	962.094	90
5	Cintura di smeraldi	35.384	78.344	70
6	Fossile	265.590	579.152	20
7	Moneta d'oro	497.911	902.698	15
8	Elmo	800.493	1.686.515	20
9	Inchiostro	823.576	1.688.691	10
10	Portagioie	552.202	1.056.157	40
11	Coltello	323.618	677.562	15
12	Spada lunga	382.846	833.132	60
13	Maschera	44.676	99.192	10
14	Collana	169.738	376.418	20
15	Distintivo in opale	610.876	1.253.986	60
16	Perla	854.190	1.853.56225	

17	Faretra	671.123	1.320.297	30
18	Rubino	698.180	1.301.637	70
19	Bracciale d’argento	446.517	859.835	50
20	Orologio	909.620	1.677.534	45
21	Uniforme	904.818	1.910.501	5
22	Pozione di veleno	730.061	1.528.646	5
23	Sciarpa di lana	931.932	1.827.477	5
24	Balestra	952.360	2.068.204	25
25	Annale	926.023	1.746.556	5
26	Coppa di zinco	978.724	2.100.851	10

Importanza della funzione di valutazione della qualità

Con il passaggio dal problema dello zaino al problema dell’impianto, una differenza fondamentale è la misurazione delle soluzioni di successo. Poiché l’impianto richiede di elaborare un livello minimo costante di valore all’ora, l’accuratezza della funzione di valutazione della qualità utilizzata diventa fondamentale per trovare soluzioni ottimali. Nel problema dello zaino, la qualità di una soluzione è banale da calcolare, in quanto coinvolge solo due elementi: assicurarsi di rispettare il limite di peso dello zaino e sommare il valore degli elementi selezionati. Nel problema dell’impianto, la funzione di calcolo della qualità deve valutare il valore ricavato, dato il tempo di estrazione e il valore di ogni articolo. Questo calcolo è più complesso, e un errore nella logica di questa funzione di calcolo della qualità influenza direttamente la qualità delle soluzioni.

Funzionamento della codifica a ordine

La codifica a ordine, nota anche come codifica a permutazione, rappresenta un cromosoma come una sequenza di elementi. La codifica a ordine di solito richiede che nel cromosoma siano presenti tutti gli elementi, il che implica che potrebbe essere necessario apportare correzioni durante l'esecuzione dell'incrocio e della mutazione, per garantire che nessun elemento sia mancante o duplicato. La Figura 5.11 illustra come un cromosoma rappresenta l'ordine di elaborazione degli elementi disponibili.

Un altro esempio in cui la codifica a ordine è sensata è la rappresentazione di soluzioni potenziali ai problemi di ottimizzazione del percorso. Dato un certo numero di destinazioni, ciascuna delle quali deve essere visitata almeno una volta minimizzando la distanza totale percorsa, il percorso può essere rappresentato come una stringa di destinazioni, nell'ordine in cui vengono visitate. Useremo questo esempio per trattare l'intelligenza di sciame nel Capitolo 6.

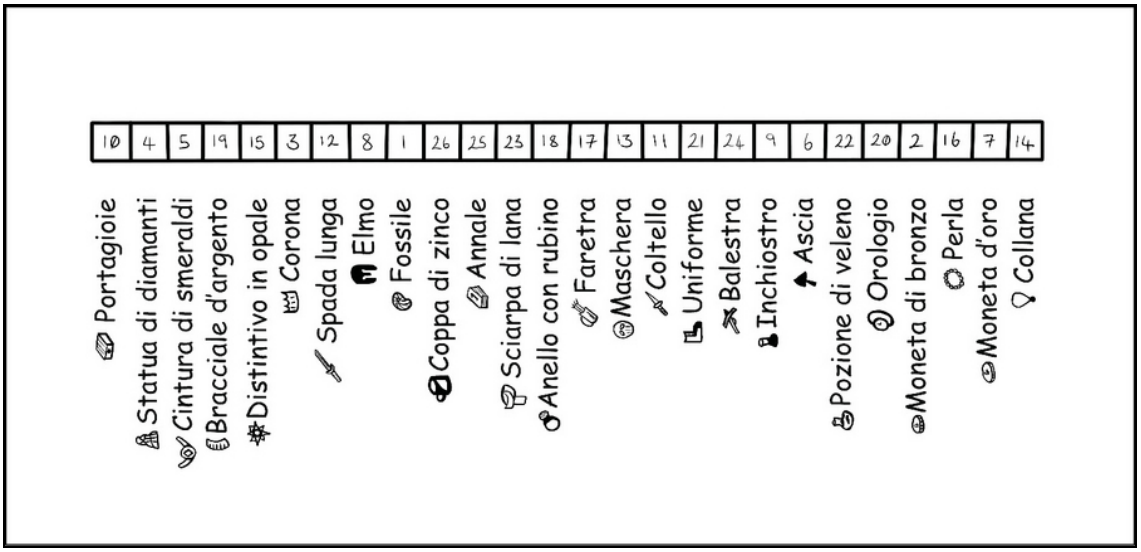


Figura 5.11 Esempio di codifica a ordine.

Mutazione a ordine: codifica a ordine/permutazione

Nella mutazione a ordine, due geni selezionati casualmente in un cromosoma codificato a ordine si scambiano di posizione, cosa che assicura che tutti gli elementi rimangano nel cromosoma pur introducendo diversità (Figura 5.12).

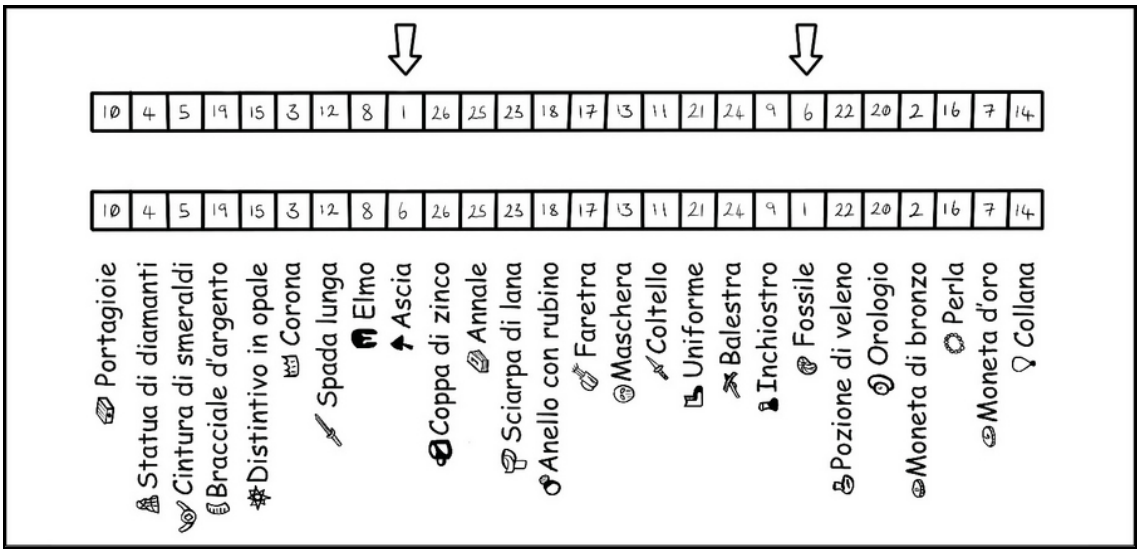


Figura 5.12 Esempio di mutazione a ordine.

Codifica ad albero: considerare le gerarchie

Nei paragrafi precedenti abbiamo visto che la codifica binaria è utile per selezionare gli elementi da un insieme, la codifica a valori effettivi è utile quando per la soluzione sono importanti i numeri disponibili e la codifica a ordine è utile per determinare priorità e sequenze. Supponiamo che gli oggetti del problema dello zaino vadano inseriti in pacchi, da spedire in vari punti della città. Ogni mezzo di trasporto può contenere un determinato volume. Il requisito è, quindi, determinare il

posizionamento ottimale dei colli per ridurre al minimo lo spazio vuoto in ciascun mezzo (Tabella 5.3).

Tabella 5.3 Capacità mezzo: 1000 di larghezza x 1000 di altezza.

ID articolo	Nome articolo	Peso (kg)	Valore ($)	L	A
1	Ascia	32.252	68.674	20	60
2	Moneta di bronzo	225.790	471.010	10	10
3	Corona	468.164	944.620	20	20
4	Statua di diamanti	489.494	962.094	30	70
5	Cintura di smeraldi	35.384	78.344	30	20
6	Fossile	265.590	579.152	15	15
7	Moneta d’oro	497.911	902.698	10	10
8	Elmo	800.493	1.686.515	40	50
9	Inchiostro	823.576	1.688.691	5	10
10	Portagioie	552.202	1.056.157	40	30
11	Coltello	323.618	677.562	10	30
12	Spada lunga	382.846	833.132	15	50
13	Maschera	44.676	99.192	20	30
14	Collana	169.738	376.418	15	20
15	Distintivo in opale	610.876	1.253.986	5	5
16	Perla	854.190	1.853.562	10	5
17	Faretra	671.123	1.320.297	30	70
18	Rubino	698.180	1.301.637	5	10

19	Bracciale d’argento	446.517	859.835	10	20
20	Orologio	909.620	1.677.534	15	20
21	Uniforme	904.818	1.910.501	30	40
22	Pozione di veleno	730.061	1.528.646	15	15
23	Sciarpa in lana	931.932	1.827.477	20	30
24	Balestra	952.360	2.068.204	50	70
25	Annale	926.023	1.746.556	25	30
26	Coppa di zinco	978.724	2.100.851	15	25

Per semplicità, supponiamo che il volume del mezzo sia un rettangolo bidimensionale e che i pacchi siano rettangolari anziché scatole tridimensionali.

Funzionamento della codifica ad albero

La codifica ad albero rappresenta un cromosoma come un albero di elementi ed è comoda per rappresentare le soluzioni potenziali in cui la gerarchia degli elementi è importante. La codifica ad albero può anche rappresentare funzioni, costituite da un albero di espressioni. Di conseguenza, può essere utilizzata per far evolvere funzioni che risolvono un determinato problema specifico; la soluzione potrebbe funzionare, ma sembrare bizzarra.

Ecco un esempio in cui ha senso impiegare la codifica ad albero. Abbiamo un mezzo con un’altezza e una larghezza e deve entrarvi un certo numero di pacchi. L’obiettivo è quello di inserire i pacchi nel vagone, in modo da ridurre al minimo lo spazio vuoto rimanente. Un approccio con codifica ad albero funzionerebbe bene nel rappresentare le soluzioni potenziali a questo problema.

Nella Figura 5.13, il nodo radice, A, rappresenta lo spazio nel mezzo, dall'alto verso il basso. Il nodo B rappresenta tutti i pacchi in file orizzontali, analogamente al nodo C e al nodo D. Il nodo E rappresenta i pacchi in una fila verticale, nella sua sezione del mezzo.

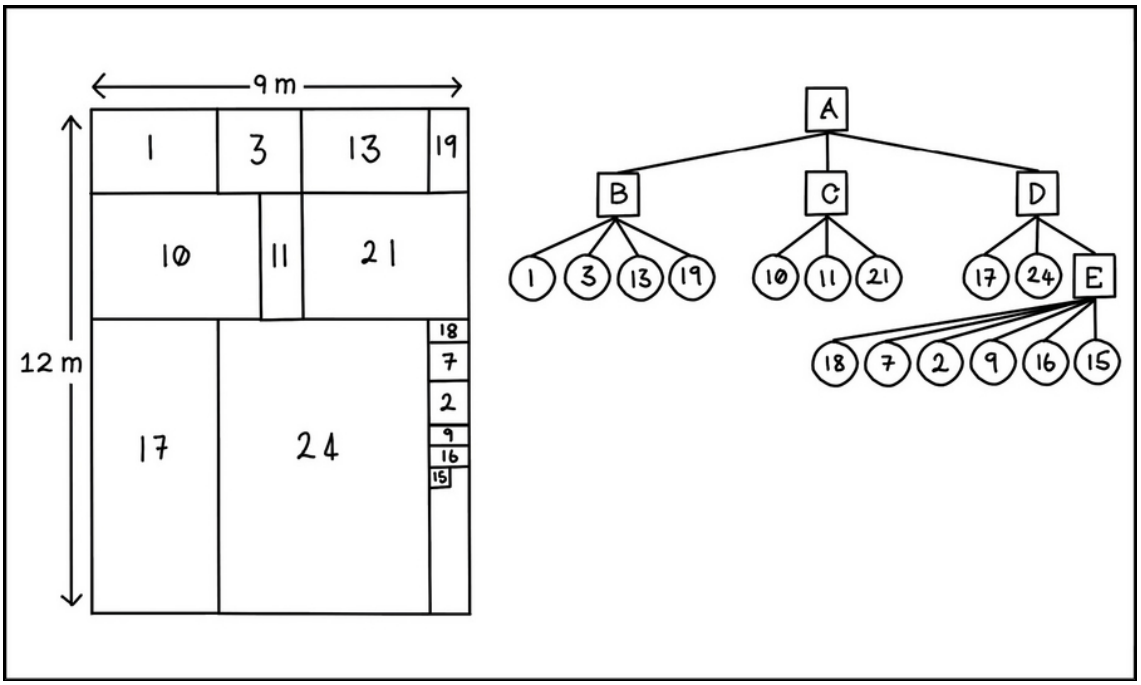


Figura 5.13 Esempio di un albero utilizzato per rappresentare il problema dell'imballaggio del mezzo.

Incrocio ad albero: ereditare porzioni di un albero

L'incrocio ad albero è simile all'incrocio a punto singolo (Capitolo 4) in quanto viene selezionato un singolo punto nella struttura ad albero e quindi le parti vengono scambiate e combinate con copie degli individui genitore, per creare un individuo figlio. Per creare un secondo figlio può essere utilizzato il processo inverso. È necessario verificare che i figli risultanti siano soluzioni valide, ovvero che obbediscono ai vincoli del problema. È possibile utilizzare più di un punto, per l'incrocio, se ciò ha senso per risolvere il problema (Figura 5.14).

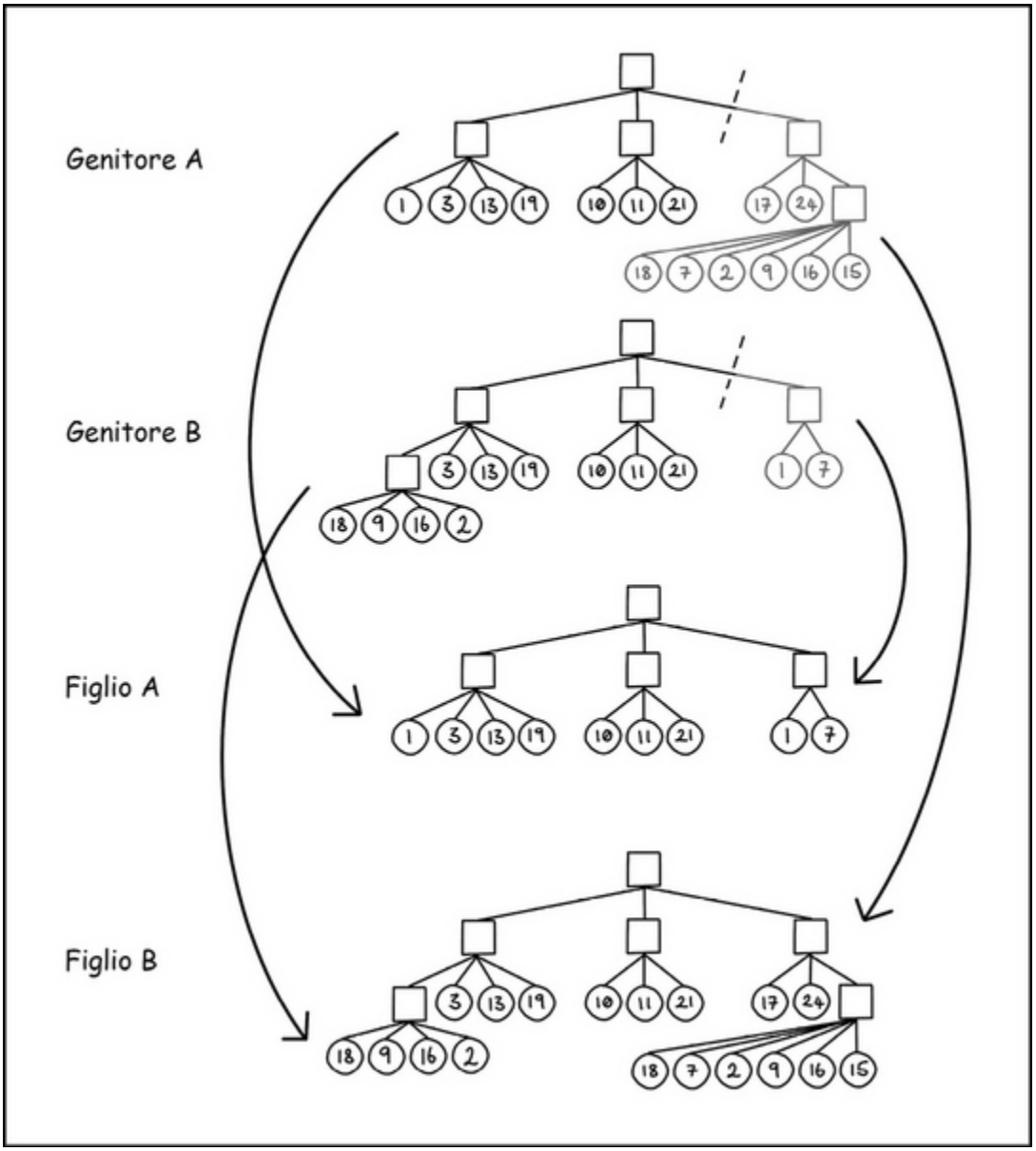


Figura 5.14 Esempio di incrocio ad albero.

Mutazione a cambio di nodo: alterazione del valore di un nodo

Nella mutazione a cambio di nodo, un nodo selezionato casualmente in un cromosoma codificato ad albero viene alterato, sostituendolo con un oggetto valido selezionato casualmente per quel nodo. Dato un

albero che rappresenta un'organizzazione di elementi, possiamo cambiare un elemento con un altro elemento valido (Figura 5.15).

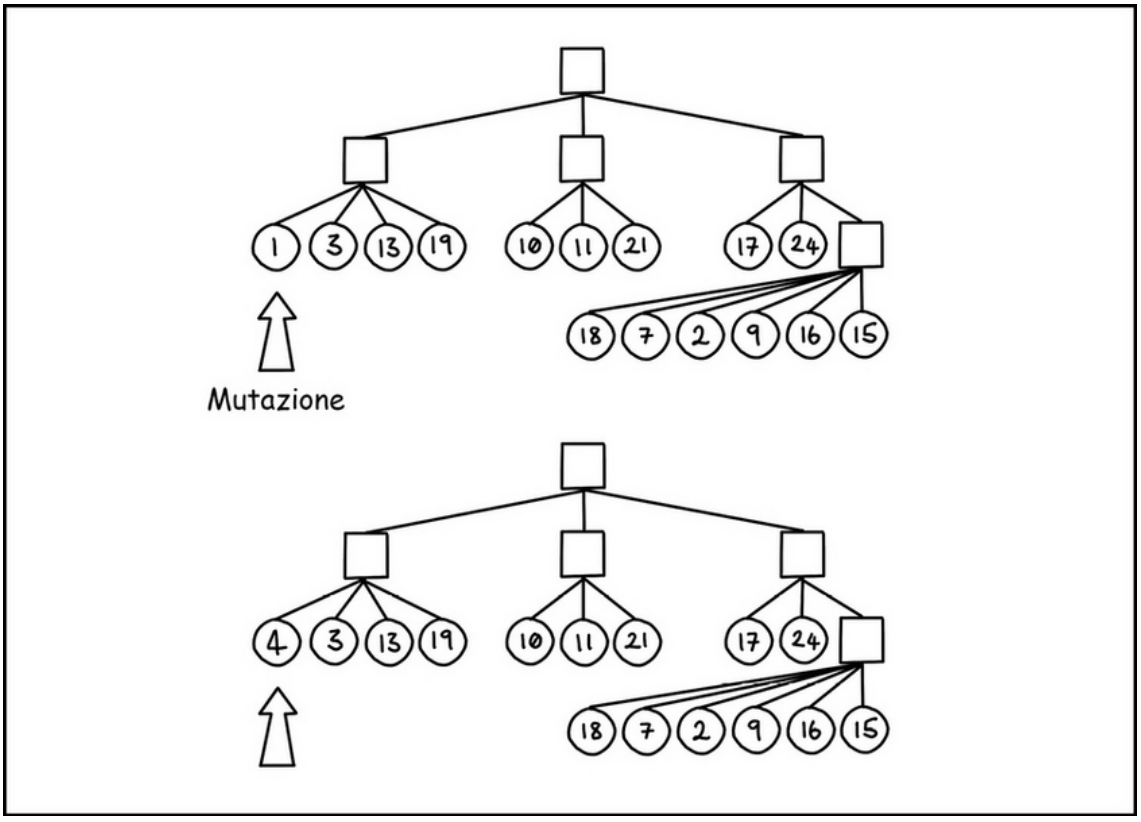


Figura 5.15 Mutazione a cambio di nodo in un albero.

Tipi comuni di algoritmi evolutivi

Questo capitolo si concentra sul ciclo di vita e sugli approcci alternativi per un algoritmo genetico. Le varianti dell'algoritmo possono essere utili per risolvere diversi tipi di problemi. Ora che sappiamo come funziona un algoritmo genetico, esamineremo queste varianti e i loro possibili casi d'uso.

Questo capitolo e il Capitolo 4 trattano vari schemi di codifica e di incrocio e strategie di selezione. Quale approccio impiegare dipende dal problema che state risolvendo.

Programmazione genetica

La programmazione genetica segue un processo simile a quello degli algoritmi genetici, ma viene utilizzata principalmente per generare programmi per risolvere problemi. Vale lo stesso processo descritto nel paragrafo precedente. La qualità, nelle soluzioni potenziali in un algoritmo di programmazione genetica, è data dall'efficacia con cui il programma generato risolve un problema computazionale. Detto questo, vediamo che il metodo di codifica ad albero funzionerebbe bene, perché la maggior parte dei programmi è costituita da grafi formati da nodi che indicano operazioni e processi. Questi alberi di elementi logici si possono far evolvere, quindi il programma subirà un'evoluzione per risolvere un problema specifico. Una cosa da notare: questi programmi di solito si evolvono fino a sembrare un groviglio di codice difficile da comprendere e correggere a mano.

Programmazione evolutiva

La programmazione evolutiva è simile alla programmazione genetica, ma la soluzione potenziale è costituita dai parametri di un programma fisso predefinito, non un programma. Se un programma richiede specifici dati di input e determinare una buona combinazione di questi input è difficile, è possibile utilizzare un algoritmo genetico per far evolvere questi input. La qualità delle soluzioni potenziali in un algoritmo di programmazione evolutiva è determinata dall'efficacia di comportamento di quel programma fisso in base ai parametri codificati in un individuo. Un approccio di programmazione evolutiva potrebbe essere utilizzato per trovare buoni parametri per una rete neurale artificiale (Capitolo 9).

Glossario dei termini usati per gli algoritmi evolutivi

Ecco un utile glossario dei termini impiegati parlando di algoritmi evolutivi.

- *Allele*: il valore di un determinato gene in un cromosoma.
- *Cromosoma:* una raccolta di geni che rappresenta una possibile soluzione.
- *Individuo*: un singolo cromosoma in una popolazione.
- *Popolazione*: una raccolta di individui.
- *Genotipo*: la rappresentazione artificiale della potenziale popolazione di soluzioni nello spazio di calcolo.
- *Fenotipo*: la rappresentazione effettiva della potenziale popolazione di soluzioni nel mondo reale.
- *Generazione*: una singola iterazione dell'algoritmo.
- *Esplorazione*: il processo che consiste nel trovare una varietà di soluzioni possibili, alcune delle quali possono essere buone e altre cattive.
- *Approfondimento*: il processo di perfezionamento di buone soluzioni e di perfezionamento iterativo.
- *Funzione di qualità*: un particolare tipo di funzione obiettivo.
- *Funzione obiettivo*: una funzione che tenta di massimizzare o minimizzare.

Altri casi d'uso per algoritmi evolutivi

Alcuni dei casi d'uso per gli algoritmi evolutivi sono stati elencati nel Capitolo 4, ma ne esistono molti altri. I seguenti casi d'uso sono particolarmente interessanti, perché utilizzano uno o più dei concetti discussi in questo capitolo.

- *Regolazione dei pesi nelle reti neurali artificiali*: le reti neurali artificiali saranno trattate più avanti, nel Capitolo 9, ma un concetto chiave del loro funzionamento è la regolazione dei pesi nella rete per apprendere modelli e relazioni insite nei dati. Vi sono diverse tecniche matematiche per aggiustare i pesi, ma gli algoritmi evolutivi sono alternative più efficienti in alcuni scenari.
- *Progettazione di circuiti elettronici*: i circuiti elettronici, dati certi componenti, possono essere progettati in varie configurazioni, alcune più efficienti di altre. Se due componenti che lavorano spesso insieme sono vicini fra loro, questa configurazione può migliorare l'efficienza. Gli algoritmi evolutivi possono essere utilizzati per far evolvere diverse configurazioni di circuiti, fino a trovare il design più ottimale.
- *Simulazione e progettazione di strutture molecolari*: come nella progettazione di circuiti elettronici, diverse molecole si comportano in modo differente e presentano vantaggi e svantaggi. Gli algoritmi evolutivi possono essere utilizzati per generare diverse strutture molecolari da simulare e studiare per determinarne le proprietà comportamentali.

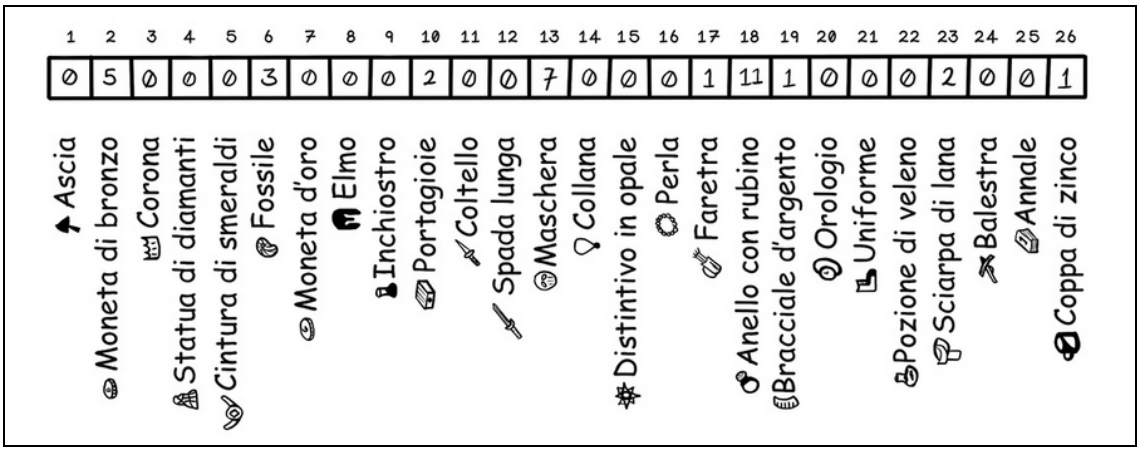
Ora che abbiamo esaminato il ciclo di vita dell'algoritmo genetico generale, nel Capitolo 4, e alcuni approcci avanzati, in questo capitolo, dovreste essere in grado di applicare gli algoritmi evolutivi nei vostri contesti e spazi delle soluzioni.

Riepilogo

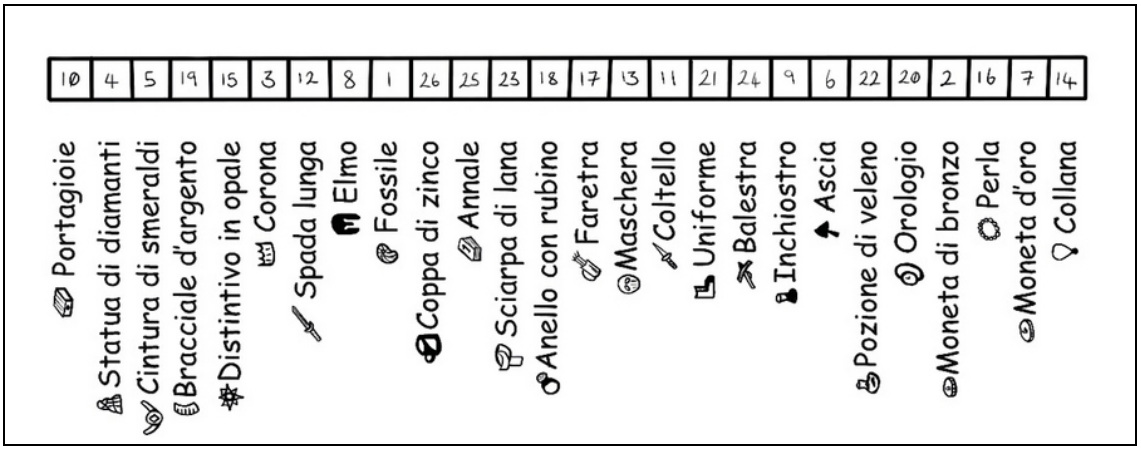
Gli algoritmi genetici possono essere utilizzati per risolvere una moltitudine di problemi.

Le diverse strategie di selezione presentano diversi vantaggi e svantaggi.

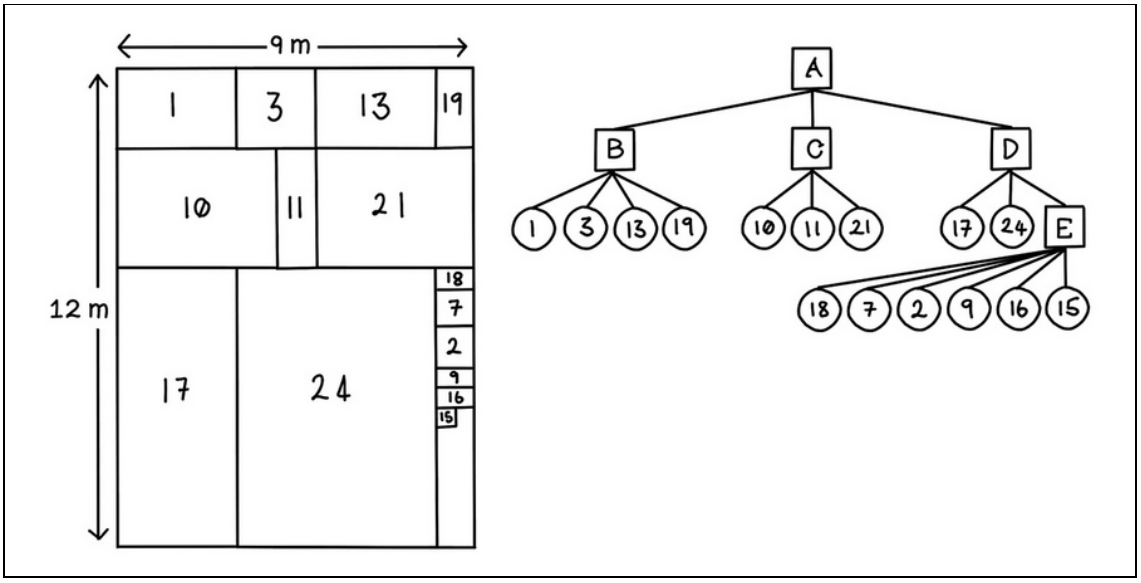
- La codifica a valori effettivi è utile in molti spazi dei problemi.



- La codifica a ordine è utile quando per risolvere i problemi è importante la priorità della sequenza.



- La codifica ad albero è utile quando per risolvere i problemi è importante la gerarchia delle relazioni.



Gli interventi su tutti i parametri dell'algoritmo sono importanti per trovare soluzioni e per individuarle in modo efficiente.

Intelligenza di sciame: le formiche

Che cos'è l'intelligenza di sciame?

Gli algoritmi di intelligenza di sciame costituiscono un sottoinsieme degli algoritmi evolutivi trattati nel Capitolo 5 e sono esempi di algoritmi ispirati dalla natura. Come abbiamo visto per la teoria dell'evoluzione, l'osservazione del comportamento delle forme di vita in natura è stata d'ispirazione per i concetti che sono alla base dell'intelligenza di sciame. Quando osserviamo il mondo che ci circonda, vediamo molte forme di vita apparentemente primitive e poco intelligenti come individui, ma che esibiscono un comportamento intelligente quando agiscono in gruppo.

Un esempio di queste forme di vita è costituito dalle formiche. Una singola formica può trasportare da dieci a cinquanta volte il proprio peso corporeo e percorrere in un minuto una distanza pari a settecento volte la sua lunghezza corporea. Sono misure già impressionanti; tuttavia, quando agisce in gruppo, quella singola formica può fare molto di più. In un gruppo, le formiche sono in grado di costruire colonie; trovare e recuperare il cibo e perfino avvertire altre formiche, illustrare i risultati di una ricognizione ad altre formiche e usare l'influenza dei pari per influenzare gli altri membri della colonia. Svolgono questi compiti per mezzo dei *feromoni*, essenzialmente odori che le formiche rilasciano ovunque vadano. Altre formiche possono percepire questi odori e, in base a essi, cambiare il proprio

comportamento. Le formiche possono secernere da dieci a venti tipi di feromoni, che possono essere utilizzati per comunicare vari tipi di informazioni. Poiché le singole formiche usano i feromoni per indicare le loro intenzioni e necessità, osserviamo che nel gruppo emerge un comportamento intelligente.

La Figura 6.1 mostra un esempio di formiche che lavorano in squadra per creare un ponte fra due punti, con lo scopo di consentire ad altre formiche di svolgere i loro compiti. Questi compiti possono essere recuperare cibo o materiali per la colonia.

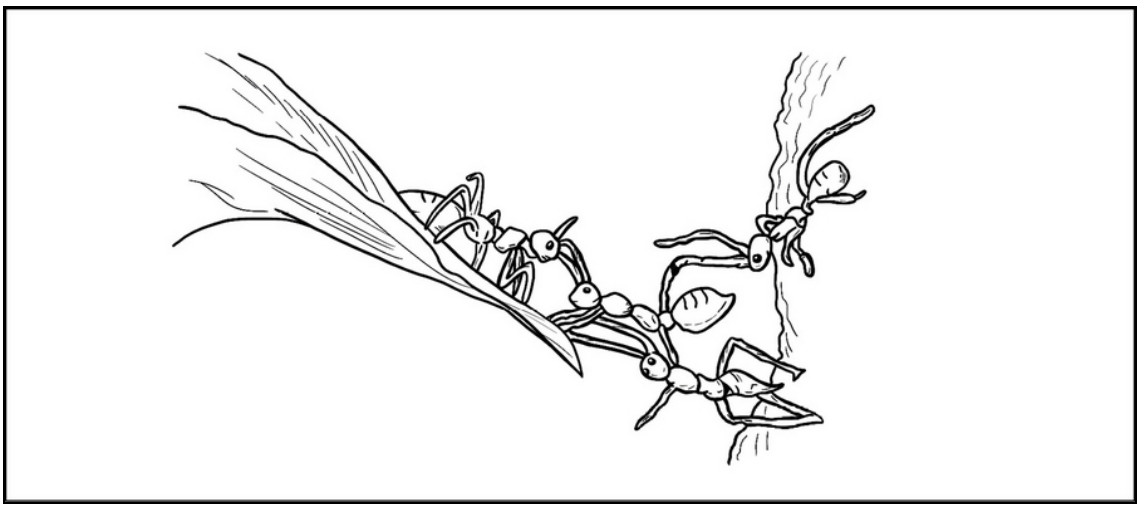


Figura 6.1 Un gruppo di formiche che collaborano per attraversare un baratro.

Un esperimento basato su formiche raccoglitrici ha mostrato che convergevano sempre verso il percorso più breve fra il formicaio e la fonte di cibo. La Figura 6.2 illustra la differenza nei movimenti della colonia dall'inizio fino a quando le formiche hanno esplorato i percorsi e aumentato l'intensità dei feromoni sui percorsi più efficienti. Questo risultato è stato osservato in un classico esperimento asimmetrico con formiche vere. Le formiche convergono sul percorso più breve dopo soli otto minuti.

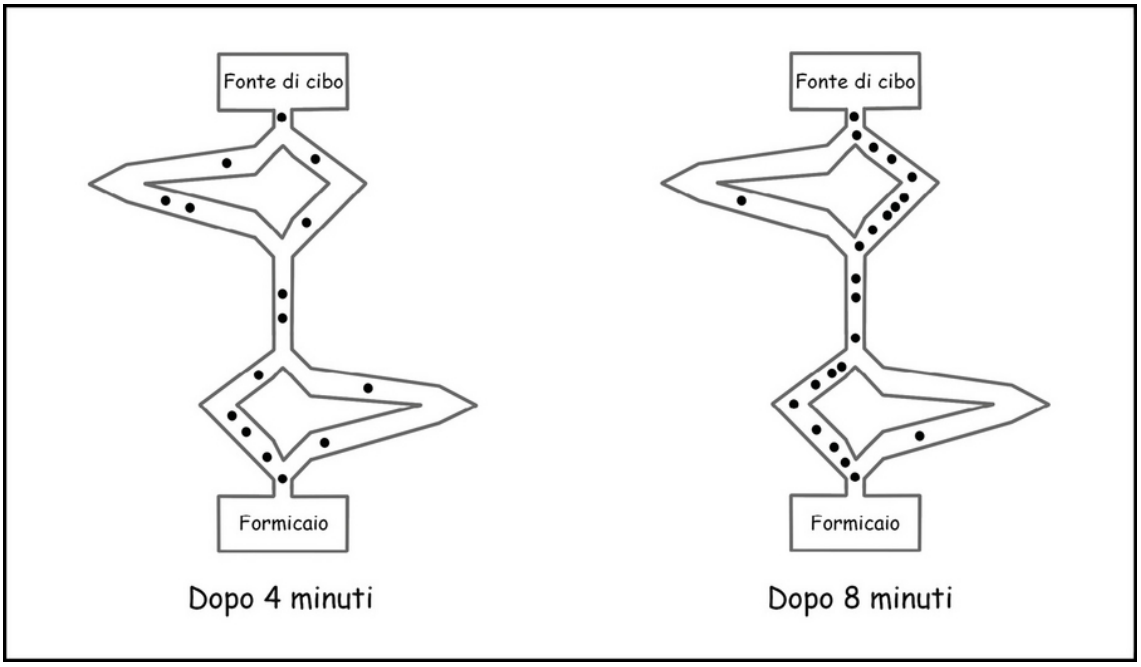


Figura 6.2 Esperimento a percorsi asimmetrici.

Gli algoritmi di ottimizzazione a colonia di formiche (ACO – *Ant Colony Optimization*) simulano il comportamento che emerge da questo esperimento. Nel trovare il percorso più breve, l'algoritmo converge in modo simile, come osservato con le formiche.

Gli algoritmi di intelligenza di sciame sono utili per risolvere problemi di ottimizzazione quando è necessario soddisfare più vincoli in uno specifico spazio del problema ed è difficile trovare la soluzione migliore in assoluto a causa del grande numero di soluzioni possibili, alcune migliori e altre peggiori. Questi problemi rientrano nella stessa classe di problemi cui sono dedicati gli algoritmi genetici; la scelta dell'algoritmo da impiegare dipende da come il problema può essere rappresentato ed elaborato. Nel Capitolo 7 ci addentreremo negli aspetti tecnici dei problemi di ottimizzazione a sciame di particelle. L'intelligenza di sciame è utile in diversi contesti, alcuni dei quali sono rappresentati nella Figura 6.3.

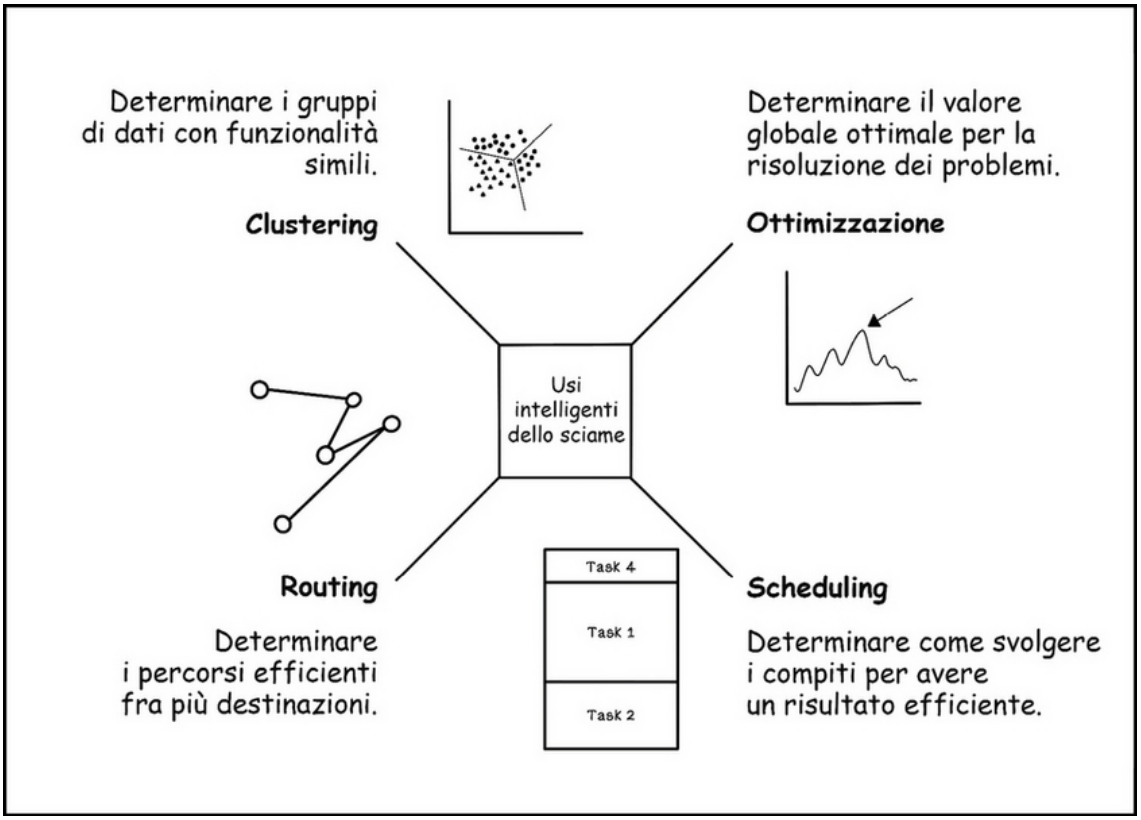


Figura 6.3 Problemi risolvibili dall'ottimizzazione a sciame.

Dato uno studio generale dell'intelligenza di sciame nelle formiche, i prossimi paragrafi esplorano alcune specifiche implementazioni ispirate a questi concetti. L'algoritmo di ottimizzazione a colonia di formiche si ispira al comportamento delle formiche che si spostano fra varie destinazioni, rilasciando feromoni e rispondendo ai feromoni che incontrano. Il comportamento che emerge è il fatto che le formiche convergono verso i percorsi più efficienti.

Problemi risolvibili dall'ottimizzazione a colonia di formiche

Immaginate di visitare un luna park che ha molte attrazioni da provare. Ogni attrazione si trova in una certa area, a distanze variabili.

Poiché non vogliamo perdere tempo camminando inutilmente, cercheremo di trovare i percorsi più brevi fra tutte le attrazioni.

La Figura 6.4 illustra le attrazioni di un piccolo luna park e le relative distanze. Notate che prendendo percorsi differenti per raggiungere le attrazioni le distanze percorse saranno differenti.

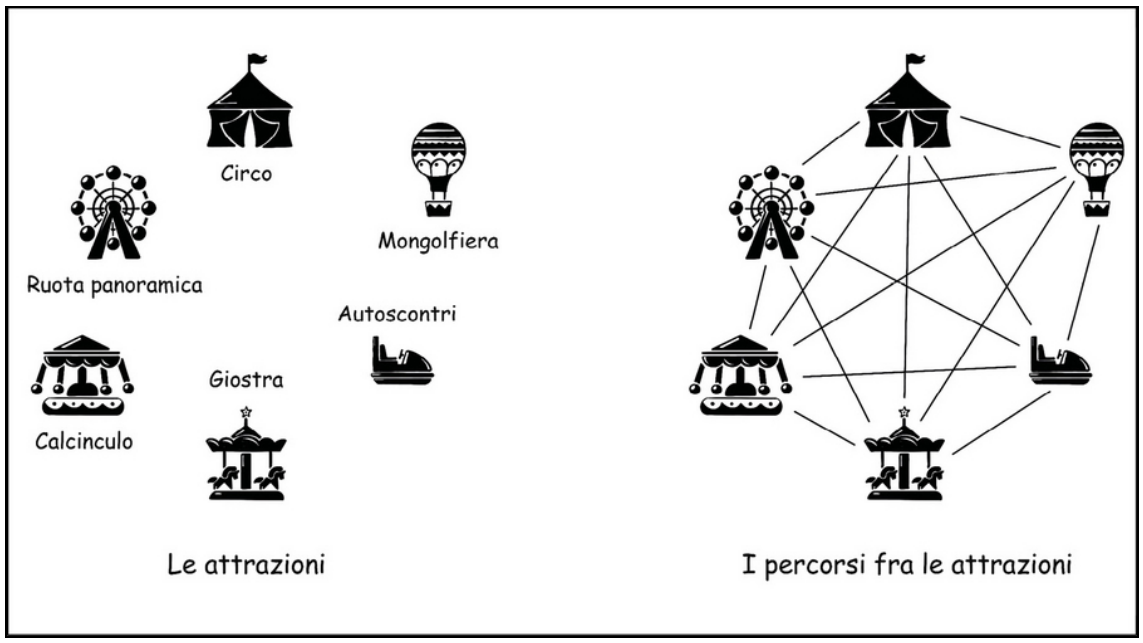


Figura 6.4 Attrazioni del luna park e percorsi.

La figura mostra sei attrazioni da visitare, con quindici percorsi. Questo esempio dovrebbe esservi familiare. Questo problema è rappresentato da un grafo completamente connesso, come descritto nel Capitolo 2. Le attrazioni sono i nodi (vertici), e i percorsi fra le attrazioni sono gli archi. La seguente formula calcola il numero di archi in un grafo completamente connesso. A mano a mano che il numero di attrazioni aumenta, il numero di archi esplode:

$$n \times (n - 1) / 2$$

Le attrazioni hanno distanze differenti fra loro. La Figura 6.5 mostra le distanze fra ogni attrazione e anche un possibile percorso fra tutte le

attrazioni. Notate che le linee rappresentate nella Figura 6.5 e che mostrano le distanze fra le attrazioni *non* sono disegnate in scala.

Se dedichiamo un po' di tempo ad analizzare le distanze fra le attrazioni, scopriamo che la Figura 6.6 mostra un percorso ottimale fra tutte le attrazioni. Visitiamo le attrazioni in questa sequenza: calcinculo, ruota panoramica, circo, giostra, mongolfiera e autoscontri.

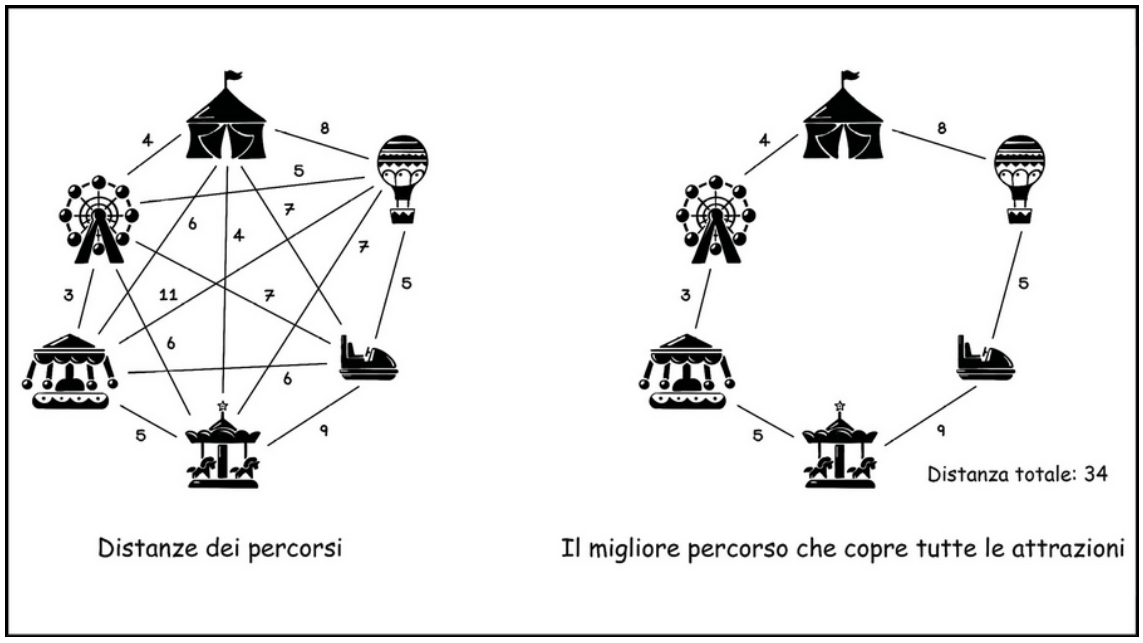


Figura 6.5 Distanze fra le attrazioni e un possibile percorso.

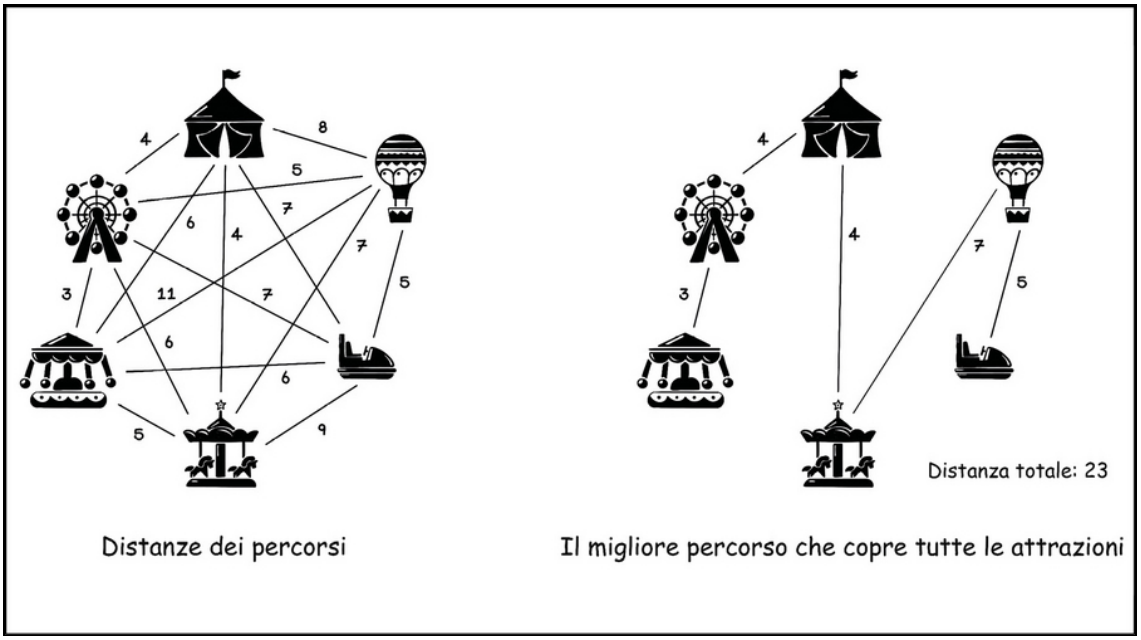


Figura 6.6 Distanze fra le attrazioni e un percorso ottimale.

Il piccolo dataset con sei attrazioni è banale da risolvere a mano, ma se aumentiamo il numero di attrazioni a quindici, il numero di possibilità esplode (Figura 6.7). Supponiamo che le “attrazioni” siano dei server e che i percorsi siano connessioni di rete. È necessario ricorrere ad algoritmi intelligenti per risolvere questi problemi.

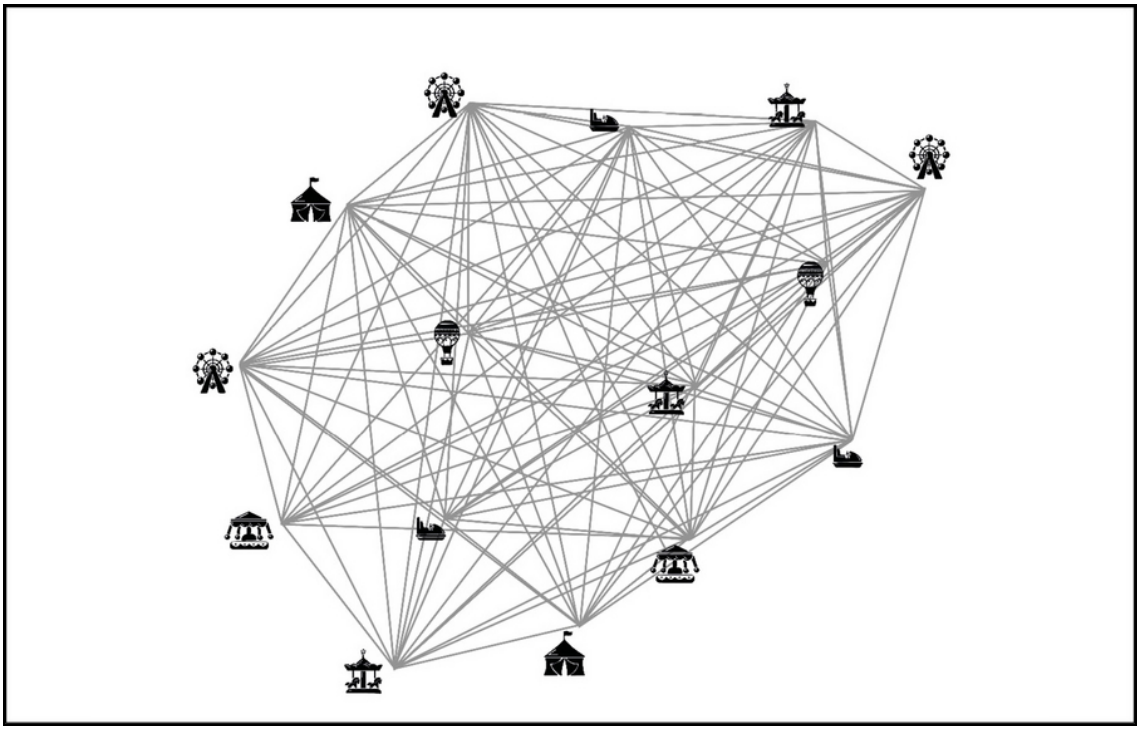
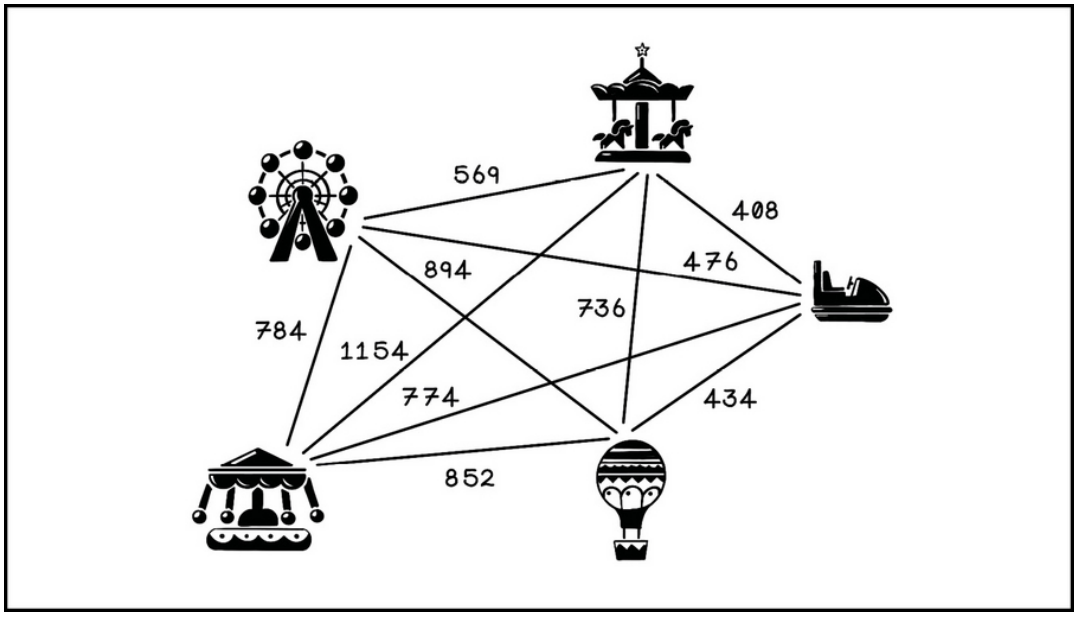


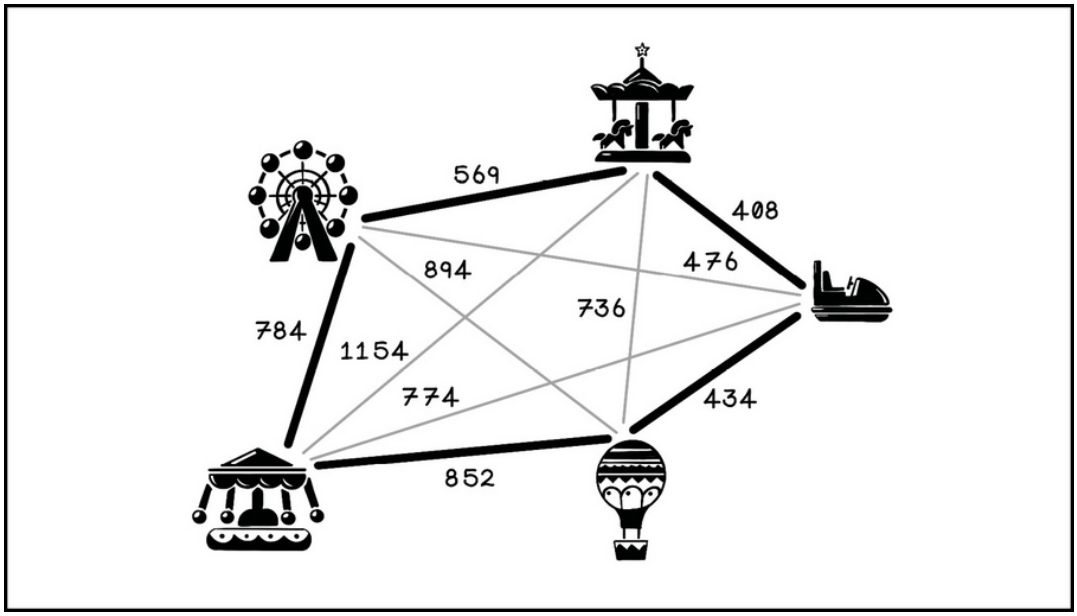
Figura 6.7 Un dataset più ampio di attrazioni e percorsi fra di esse.

Esercizio: trovate, a mano, il percorso più breve in questa configurazione

del luna park



Soluzione



Un modo per risolvere questo problema in modo computazionale consiste nel tentare un approccio a forza bruta: generare e valutare ogni combinazione di “tour” delle attrazioni (dove un tour è una sequenza di visite in cui ogni attrazione viene visitata una volta) fino a trovare la distanza totale più breve. Ancora una volta, questa soluzione può sembrare ragionevole, ma in un dataset di grandi dimensioni, questo calcolo è costoso e richiede troppo tempo. Un

approccio a forza bruta con 48 attrazioni richiede decine di ore prima di trovare una soluzione ottimale.

Rappresentazione degli stati: che aspetto hanno i percorsi e le formiche?

Dato il problema del luna park, dobbiamo rappresentare i suoi dati in un modo che sia adatto a essere elaborato dall'algoritmo di ottimizzazione a colonia di formiche. Abbiamo le attrazioni e le relative distanze, quindi possiamo usare una matrice delle distanze per rappresentare lo spazio del problema in un modo accurato e semplice.

Una *matrice delle distanze* è un array bidimensionale in cui ogni indice rappresenta un'entità; l'insieme correlato indica la distanza fra due entità. Questa matrice è simile alla matrice di adiacenza che abbiamo esaminato nel Capitolo 2 (Figura 6.8 e Tabella 6.1).

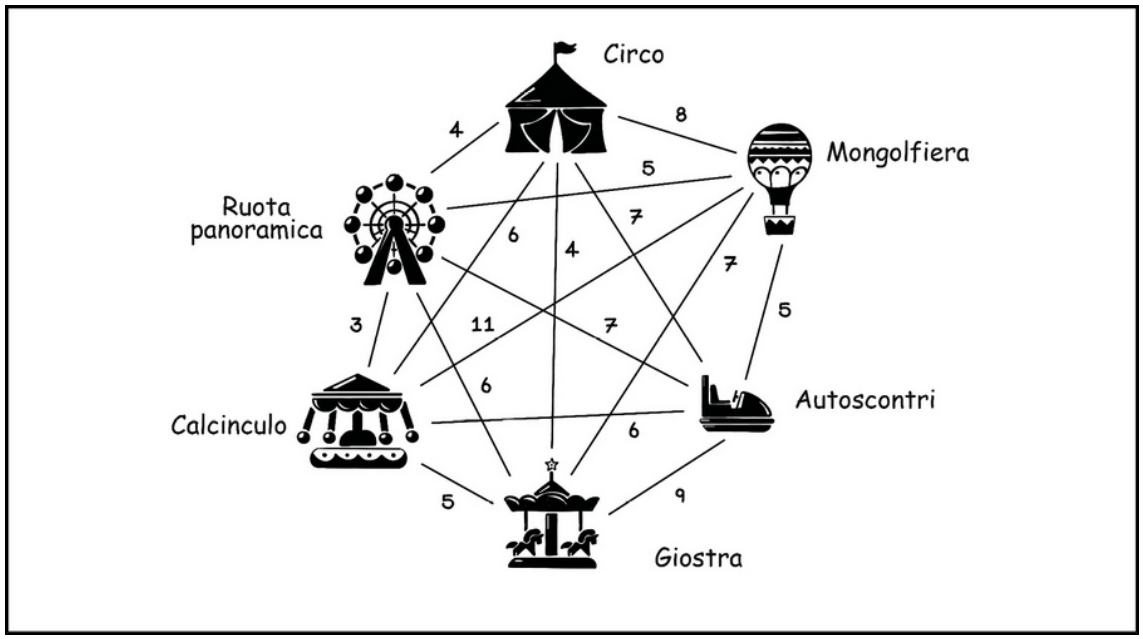


Figura 6.8 Un esempio del problema del luna park.

Tabella 6.1 Distanze fra le attrazioni.

	Circo	Mongolfiera	Autoscontri	Giostra	Calcinculo	Ruota panoramica
Circo	0	8	7	4	6	4
Mongolfiera	8	0	5	7	11	5
Autoscontri	7	5	0	9	6	7
Giostra	4	7	9	0	5	6
Calcinculo	6	11	6	5	0	3
Ruota panoramica	4	5	7	6	3	0

Pseudocodice

Le distanze fra le attrazioni possono essere rappresentate come una matrice delle distanze, un array di array in cui un riferimento a x, y nell'array fa riferimento alla distanza fra le attrazioni x e y. La distanza fra un'attrazione e se stessa è ovviamente 0. Questo array può anche essere creato a livello di codice, iterando i dati da un file e creando ogni elemento:

```
let attraction
–
distances equal
[
[0,8,7,4,6,4],
[8,0,5,7,11,5],
[7,5,0,9,6,71,
[4,7,9,0,5,6],
[6,11,6,5,0,3],
[4,5,7,6,3,01,
]
```

Il prossimo elemento da rappresentare sono le formiche. Le formiche si spostano verso le varie attrazioni e lasciano dietro di sé dei feromoni. Poi le formiche valutano quale attrazione visitare dopo. Infine, le formiche conoscono la distanza totale percorsa. Ecco le proprietà di base di una formica (Figura 6.9).

- *Memoria*: nell'algoritmo ACO, questo è l'elenco delle attrazioni già visitate.
- *Migliore qualità*: questa è la distanza totale più breve percorsa per visitare tutte le attrazioni.
- *Azione*: scelta della prossima destinazione da visitare, rilasciando feromoni lungo il percorso.

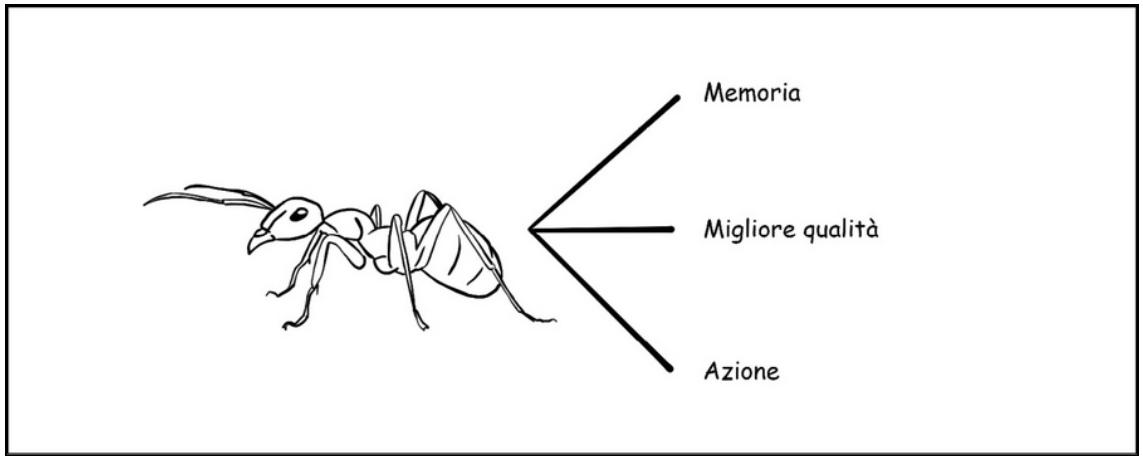


Figura 6.9 Proprietà di una formica.

Pseudocodice

Sebbene il concetto astratto di formica preveda le doti di memoria, qualità e azione, per risolvere il problema del luna park sono necessari dati e funzioni specifici. Per incapsulare la logica di una formica, possiamo usare una classe. Quando viene inizializzata un'istanza della classe `Ant`, viene predisposto un array vuoto per rappresentare l'elenco di attrazioni visitate dalla formica. Inoltre, verrà selezionata un'attrazione casuale, che sarà il punto di partenza di quella specifica formica:

```
Ant(attraction_count):
  let ant.visited_attractions equal an empty array
    append a random number between 0 and (attraction_count - 1) to
ant.visited_attractions
```

La classe `Ant` contiene anche diverse funzioni, utilizzate per il movimento delle formiche. Le funzioni `visit_*` vengono utilizzate per determinare verso quale attrazione la formica si sposterà successivamente. La funzione `visit_attraction`

genera una possibilità casuale di visitare un'attrazione casuale. In questo caso, viene richiamata `visit_random_attraction`; altrimenti viene richiamata `roulette_wheel_selection` con un elenco calcolato di probabilità. Troverete maggiori dettagli nel prossimo paragrafo:

```
Funzioni di Ant:
visit_attraction(pheromone_trails)
visit_random_attraction()
visit_probabilistic_attraction(pheromone_trails)
roulette_wheel_selection(probabilities)
get_distance_traveled()
```

Infine, la funzione `get_distance_traveled` viene utilizzata per calcolare la distanza totale percorsa da una determinata formica, utilizzando la sua lista di attrazioni visitate. Questa distanza deve minimizzata, per trovare il percorso più breve, e per farlo usiamo una funzione di qualità del percorso:

```
get_distance_traveled(ant):
  let total_distance equal 0
  for a in range(1, length of ant.visited_attractions):
        total_distance += distance between ant.visited_attractions[a - 1]
and ant.visited_attractions[a]
  return total_distance
```

L'ultima struttura di dati da creare riguarda il concetto di scia di feromoni. Analogamente alle distanze fra le attrazioni, l'intensità dei feromoni su ciascun percorso può essere rappresentata come una matrice delle distanze, ma invece di contenere le distanze, contiene le intensità dei feromoni. Nella Figura 6.10, le linee più spesse indicano scie di feromoni più intense. La Tabella 6.2 descrive le scie di feromoni fra le attrazioni.

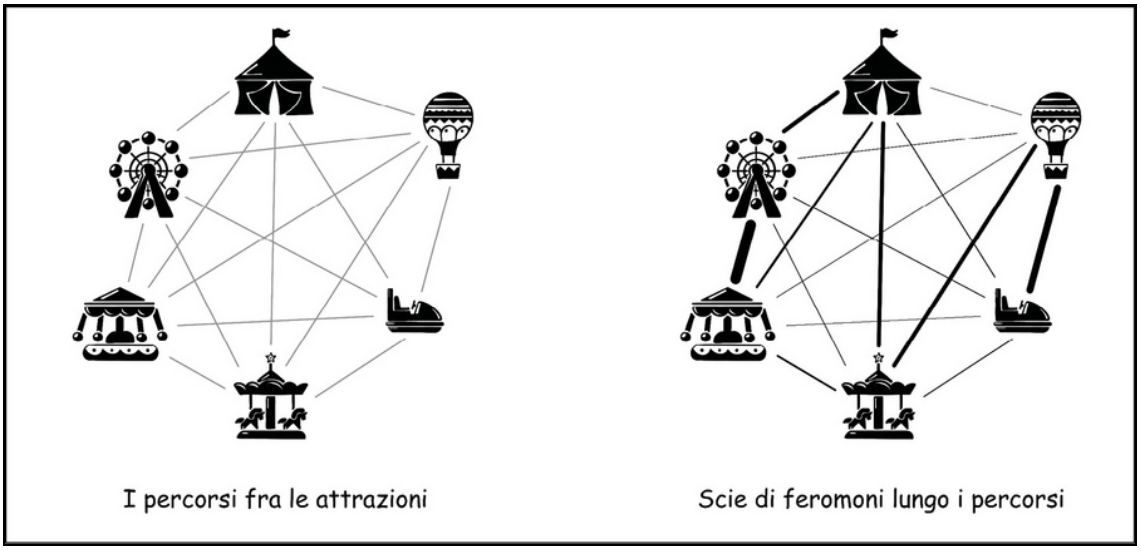


Figura 6.10 Esempio di intensità dei feromoni sui percorsi.

Tabella 6.2 Intensità dei feromoni fra le attrazioni.

	Circo	**Mongolfiera**	**Autoscontri**	**Giostra**	**Calcinculo**	**Ruota panoramica**
Circo	0	2	0	8	6	8
Mongolfiera	2	0	10	8	2	2
Autoscontri	2	10	0	0	2	2
Giostra	8	8	2	0	2	2
Altalene	6	2	2	2	0	10
Ruota panoramica	8	2	2	2	10	0

Il ciclo di vita dell'algoritmo di ottimizzazione a colonia di formiche

Ora che sappiamo le strutture di dati richieste, possiamo immergerci nel funzionamento dell'algoritmo di ottimizzazione a colonia di formiche. L'approccio nella progettazione di tale algoritmo si basa sullo spazio del problema affrontato. Ogni problema ha un contesto unico e un dominio differente in cui i dati sono rappresentati, ma i principi rimangono gli stessi.

Detto questo, esaminiamo come configurare un algoritmo di ottimizzazione a colonia di formiche per risolvere il problema del luna park. Il ciclo di vita generale di tale algoritmo è il seguente.

- *Inizializza le scie di feromoni*. Crea il concetto di scie di feromoni fra le attrazioni e inizializza i loro valori di intensità.
- *Imposta la popolazione di formiche*. Crea una popolazione di formiche in cui ogni formica parte da un'attrazione differente.
- *Scegli la prossima visita per ogni formica*. Sceglie la successiva attrazione da visitare per ogni formica, finché ogni formica non ha visitato una volta tutte le attrazioni.
- *Aggiorna le scie di feromoni*. Aggiorna l'intensità delle scie di feromoni in base ai movimenti delle formiche, tenendo conto anche dell'evaporazione dei feromoni.
- *Aggiorna la soluzione migliore*. Aggiorna la soluzione migliore, data la distanza totale percorsa da ciascuna formica.
- *Determina i criteri di arresto*. Il processo delle formiche che visitano le attrazioni si ripete per diverse iterazioni. Un'iterazione prevede che ogni formica visiti tutte le attrazioni una volta. Il criterio di arresto determina il numero totale di iterazioni da

eseguire. Un maggior numero di iterazioni consente alle formiche di prendere decisioni migliori in base alle tracce di feromoni.

La Figura 6.11 descrive il ciclo di vita generale dell'algoritmo di ottimizzazione a colonia di formiche.

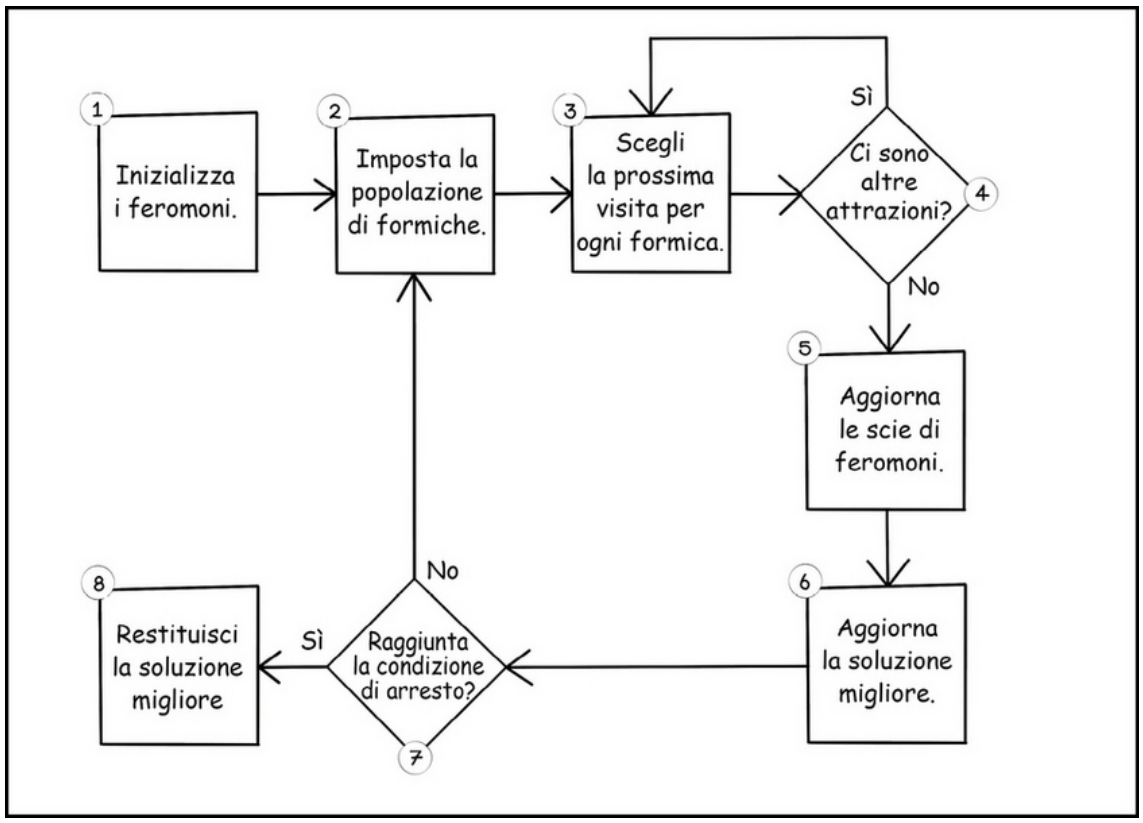


Figura 6.11 Il ciclo di vita dell'algoritmo di ottimizzazione a colonia di formiche.

Inizializza le scie di feromoni

Il primo passo nell'algoritmo di ottimizzazione a colonia di formiche consiste nell'inizializzare le scie di feromoni. Poiché nessuna formica ha ancora camminato sui percorsi fra le attrazioni, i percorsi dei feromoni verranno tutti inizializzati a 1. In tal modo, nessun percorso avrà alcun vantaggio rispetto agli altri. L'aspetto importante è

la definizione di una struttura di dati affidabile per rappresentare le scie di feromoni, che esamineremo in seguito (Figura 6.12).

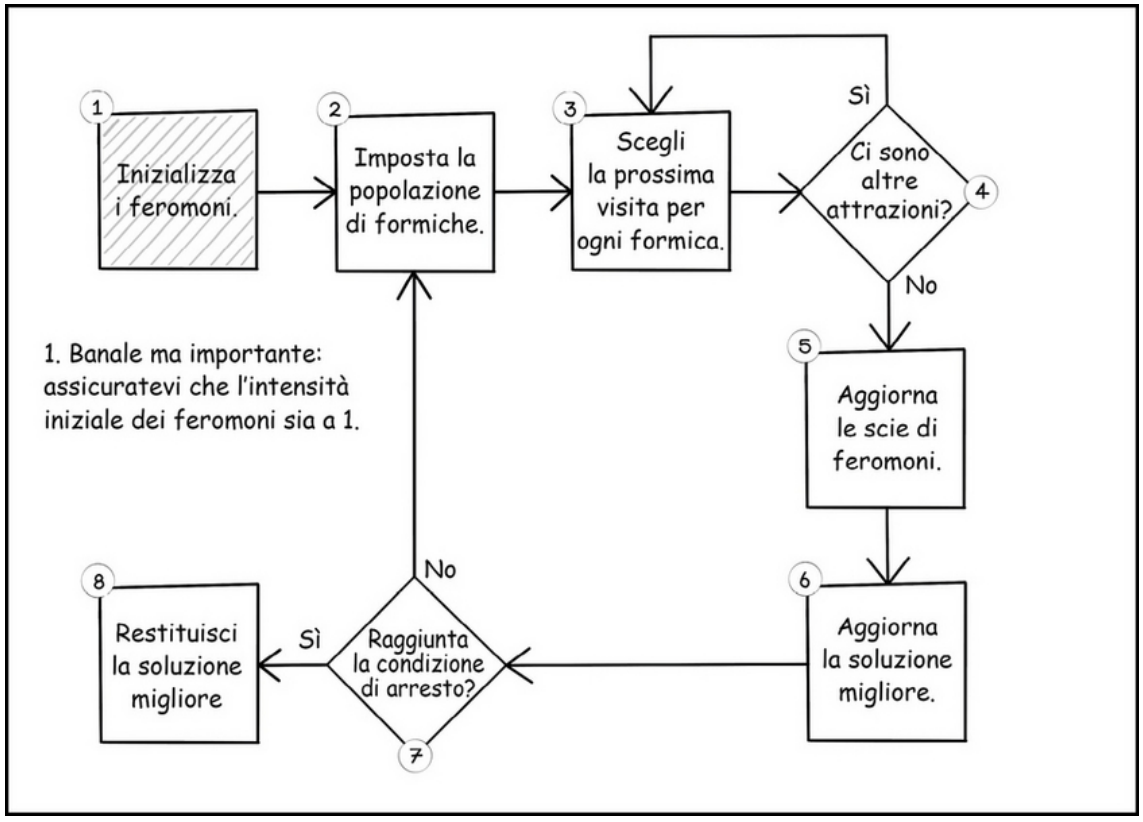


Figura 6.12 Inizializza i feromoni.

Questo concetto può essere applicato ad altri problemi nei quali invece dell'intensità dei feromoni, si applica un'altra euristica.

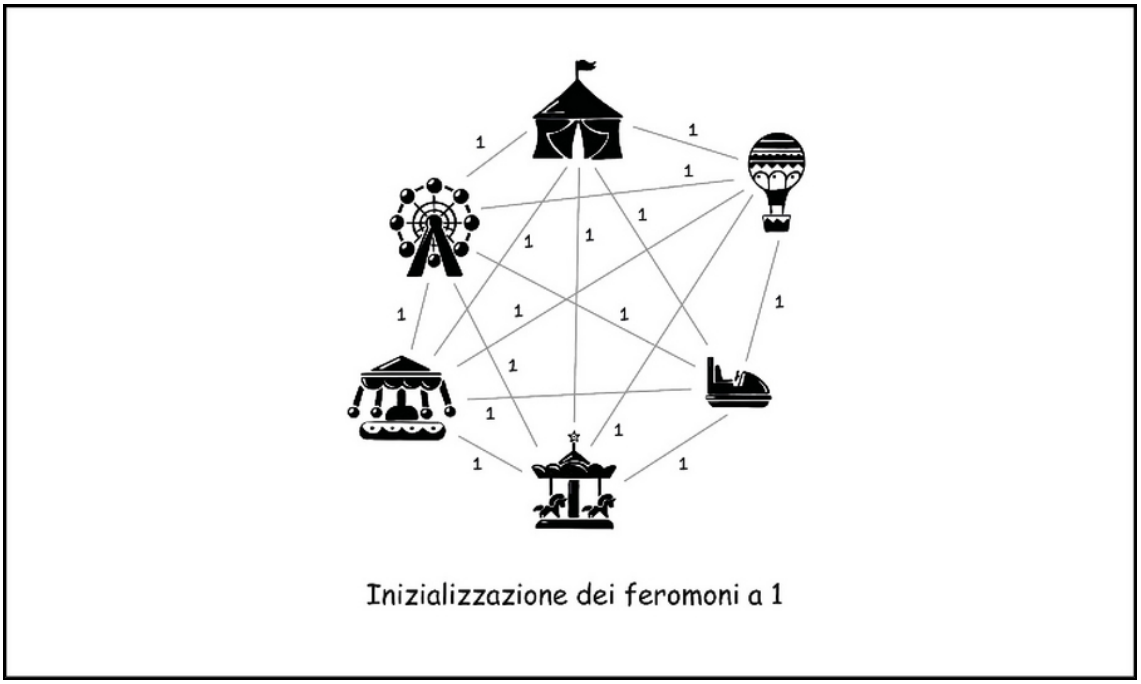


Figura 6.13 Inizializzazione dei feromoni.

Pseudocodice

Analogamente alle distanze fra le attrazioni, anche le scie di feromoni possono essere rappresentate da una matrice delle distanze, dove il riferimento a x, y fornisce l'intensità dei feromoni sul percorso fra le attrazioni x e y. L'intensità iniziale del feromone su ogni percorso è inizializzata a 1. I valori per tutti i percorsi devono essere inizializzati tutti con lo stesso valore, per evitare di influenzare i percorsi fin dall'inizio:

```
let pheromone_trails equal
  [
      [1,1,1,1,1,1],
      [1,1,1,1,1,1],
      [1,1,1,1,1,1],
      [1,1,1,1,1,1],
      [1,1,1,1,1,1],
      [1,1,1,1,1,1]
  ]
```

Imposta la popolazione di formiche

Il passo successivo dell'algoritmo è la creazione di una popolazione di formiche che si sposteranno fra le attrazioni lasciando scie di feromoni (Figura 6.14).

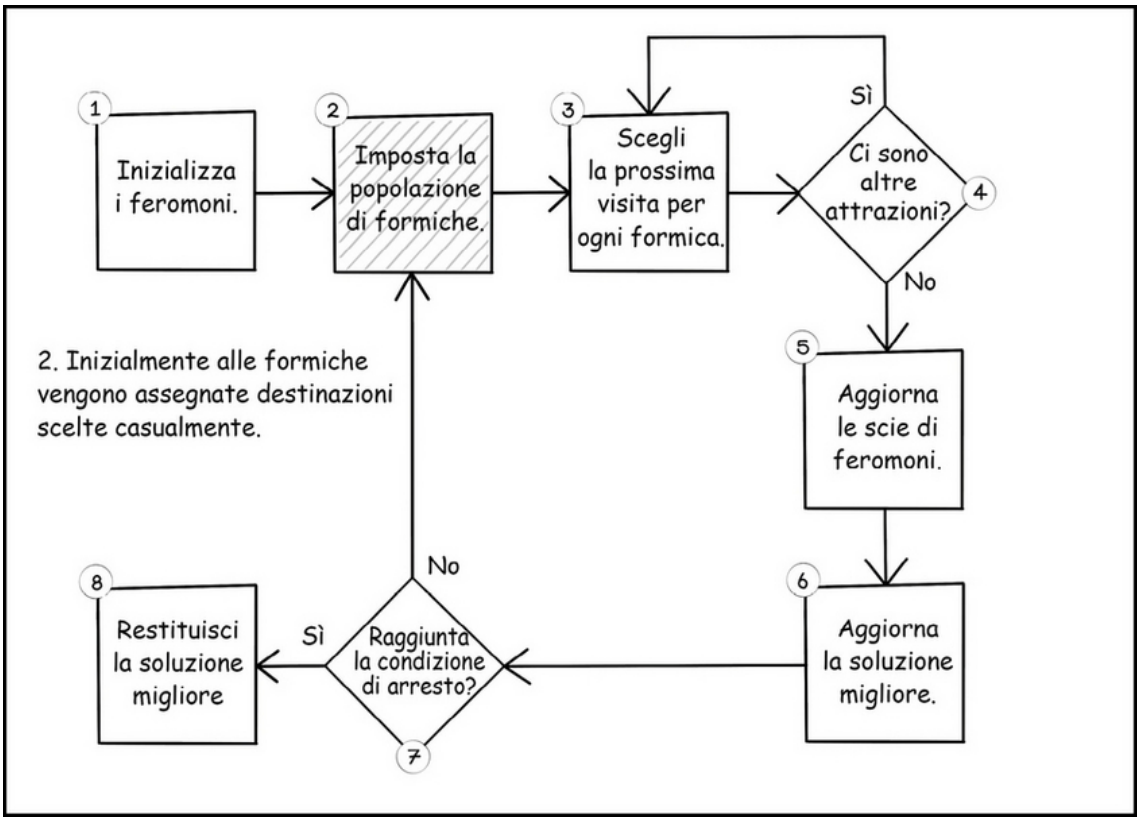


Figura 6.14 Imposta la popolazione di formiche.

Le formiche inizieranno da attrazioni assegnate casualmente (Figura 6.15): un punto casuale in una potenziale sequenza, perché l'algoritmo di ottimizzazione a colonia di formiche può essere applicato a problemi che non riguardano una distanza. Dopo aver visitato tutte le destinazioni, le formiche vengono riportate ai rispettivi punti di partenza.

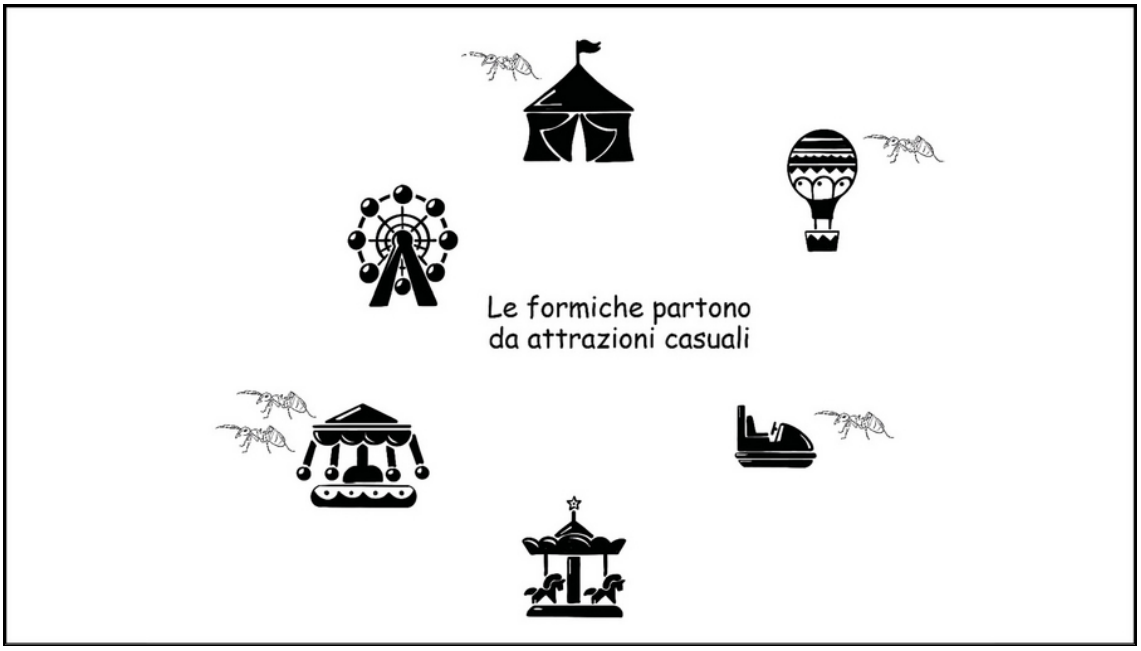


Figura 6.15 Le formiche partono da attrazioni casuali.

Possiamo adattare questo principio a un problema differente. In un problema di pianificazione delle attività, ogni formica inizia da un'attività differente.

Pseudocodice

L'impostazione della colonia di formiche include l'inizializzazione di diverse formiche e la loro aggiunta a una lista, cui è possibile fare riferimento in seguito. Ricordate che la funzione di inizializzazione della classe `Ant` sceglie un'attrazione casuale da cui partire:

```
setup_ants(attraction_count, number_of_ants_factor):
  let number_of_ants equal round(attraction_count * number_of_ants factor)
  let ant_colony equal to an empty array
  for i in range(0, number_of_ants)
      append new Ant to ant_colony
  return ant_colony
```

Scegli la prossima visita per ogni formica

Le formiche devono selezionare la prossima attrazione da visitare. Visitano nuove attrazioni fino a quando non hanno visitato una volta

tutte le attrazioni, una cosa che chiamiamo “tour”. Le formiche scelgono la destinazione successiva in base a due fattori (Figura 6.16).

- *Intensità dei feromoni*: l’intensità dei feromoni di tutti i percorsi disponibili.
- *Valore euristico*: un risultato di un’euristica definita per tutti i percorsi disponibili, ovvero la distanza del percorso fra le attrazioni, nell’esempio del luna park.

Le formiche non si dirigeranno verso destinazioni che hanno già visitato. Se una formica ha già visitato gli autoscontri, non si recherà più a quell’attrazione nel suo tour.

La natura stocastica delle formiche

L’algoritmo di ottimizzazione a colonia di formiche ha un elemento di casualità. L’intenzione è quella di offrire alle formiche la possibilità di esplorare percorsi meno ottimali, ma che potrebbero tradursi in una più breve distanza complessiva del tour.

Innanzitutto, una formica ha una probabilità casuale di decidere di scegliere una destinazione casuale. Potremmo generare un numero casuale compreso fra `0` e `1`, e se il risultato è `0,1` o meno, la formica deciderà di scegliere una destinazione casuale; dunque una probabilità del 10% di scegliere una destinazione casuale. Se una formica decide che sceglierà una destinazione casuale, deve selezionare casualmente una destinazione da visitare, fra tutte le destinazioni disponibili.

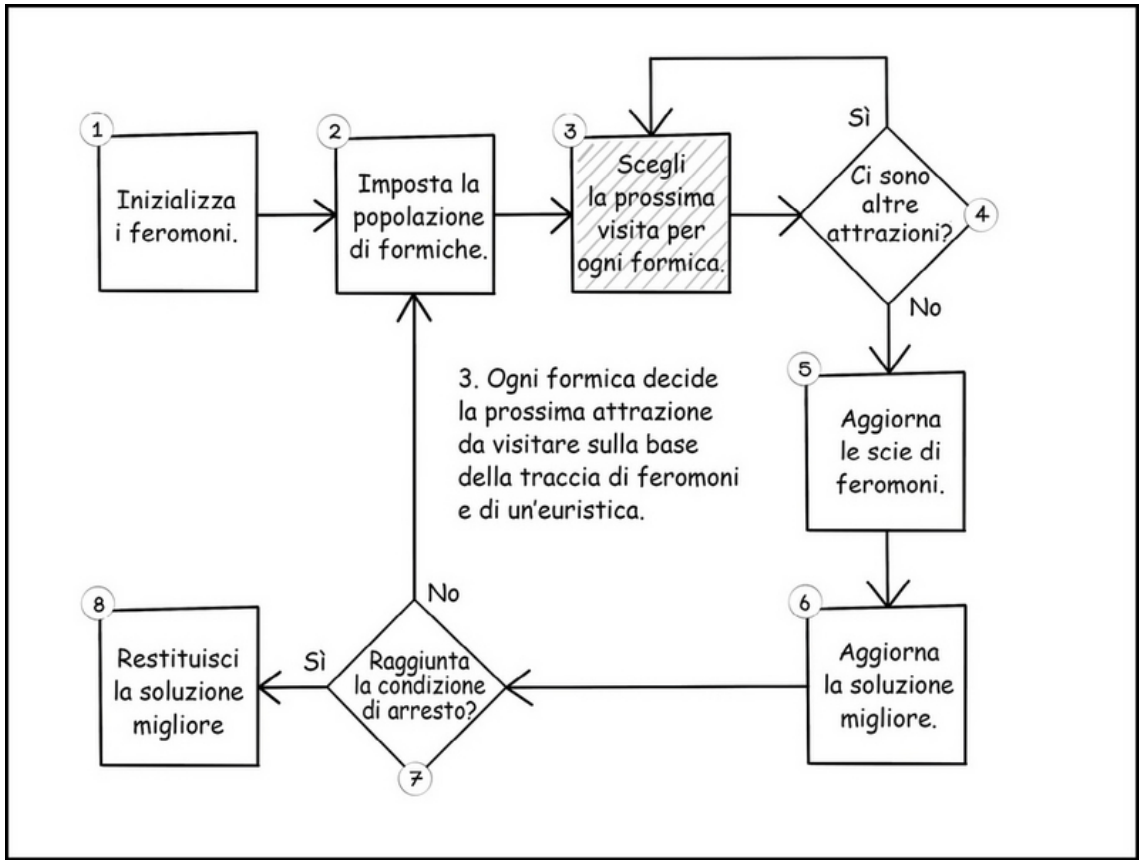


Figura 6.16 Scegli la prossima visita per ogni formica.

Selezione della destinazione in base a un'euristica

Quando una formica deve affrontare la decisione di scegliere la destinazione successiva in modo non casuale, valuta l'intensità dei feromoni su quel percorso e il valore euristico utilizzando la seguente formula:

$$\frac{(\text{feromoni sul percorso x})^a * (1 \,/\, \text{euristica per il percorso x})^b}{\text{somma di tutte le destinazioni disponibili } ((\text{feromoni sul percorso n})^a * (1 \,/\, \text{euristica per il percorso n})^b)}$$

Dopo aver applicato questa funzione a ogni possibile percorso verso la rispettiva destinazione, la formica seleziona la destinazione con il

miglior valore complessivo. La Figura 6.17 illustra i possibili percorsi dal circo con le rispettive distanze e intensità di feromoni.

Esaminiamo la formula per chiarire i calcoli in corso e il modo in cui i risultati influenzano il processo decisionale (Figura 6.18).

Le variabili *alfa* (`a`) e *beta* (`b`) vengono utilizzate per dare maggior peso all'influenza dei feromoni o dell'euristica. Queste variabili possono essere regolate per bilanciare il giudizio della formica fra fare una mossa basata sulle tracce di feromoni (che rappresentano ciò che la colonia sa su quel percorso) e ciò che lei sa del problema.

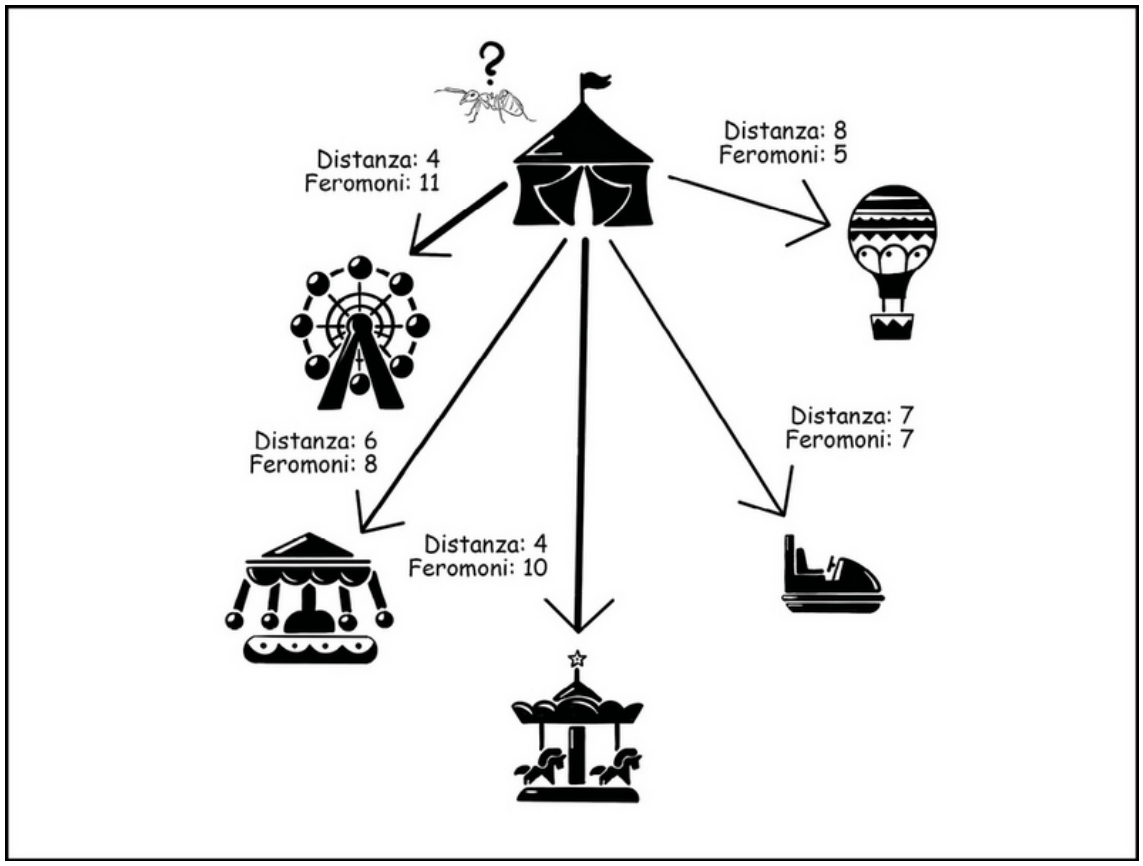


Figura 6.17 Esempio di possibili percorsi dal circo.

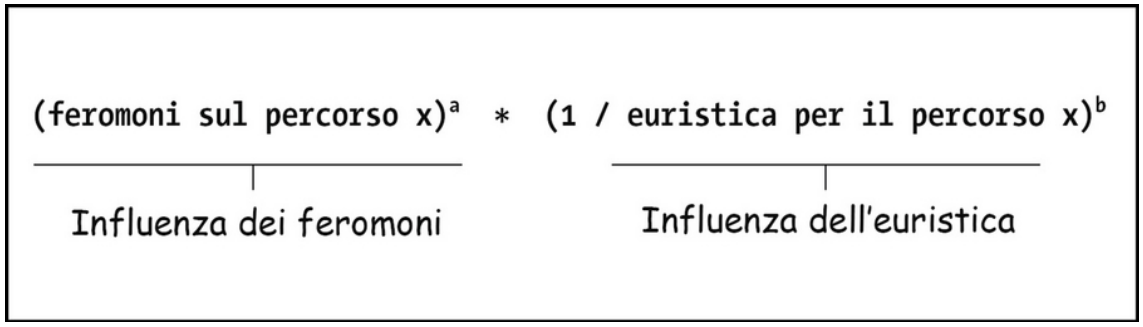


Figura 6.18 L'influenza dei feromoni e l'influenza dell'euristica nella formula.

Questi parametri sono definiti in anticipo e di solito non vengono modificati durante l'esecuzione dell'algoritmo.

L'esempio seguente analizza ogni percorso a partire dal circo e calcola le probabilità di spostarsi verso un'altra attrazione.

- `a` (*alfa*) è impostato a `1`.
- `b` (*beta*) è impostato a `2`.

Poiché `b` è maggiore di `a`, in questo esempio viene favorita l'influenza dell'euristica.

Esaminiamo un esempio dei calcoli utilizzati per determinare la probabilità di scegliere un determinato percorso (Figura 6.19).

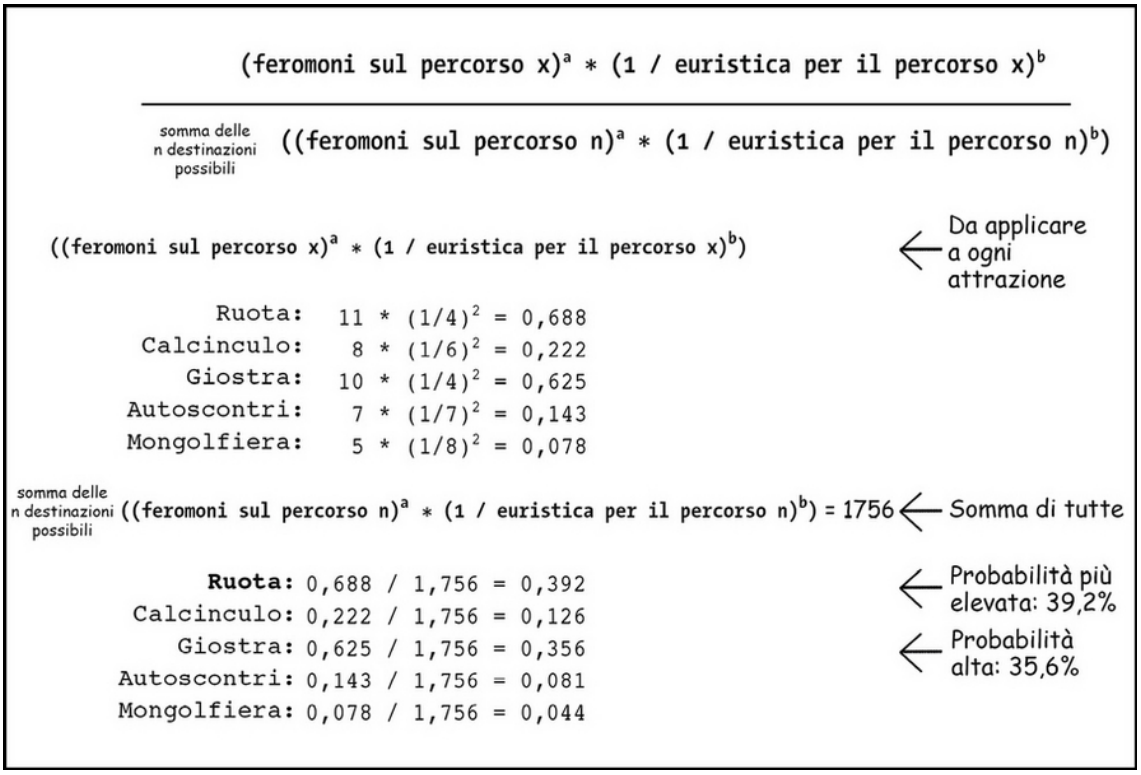


Figura 6.19 Calcoli delle probabilità per i vari percorsi.

Dopo aver applicato questo calcolo, date tutte le destinazioni disponibili, alla formica rimangono le opzioni mostrate nella Figura 6.20.

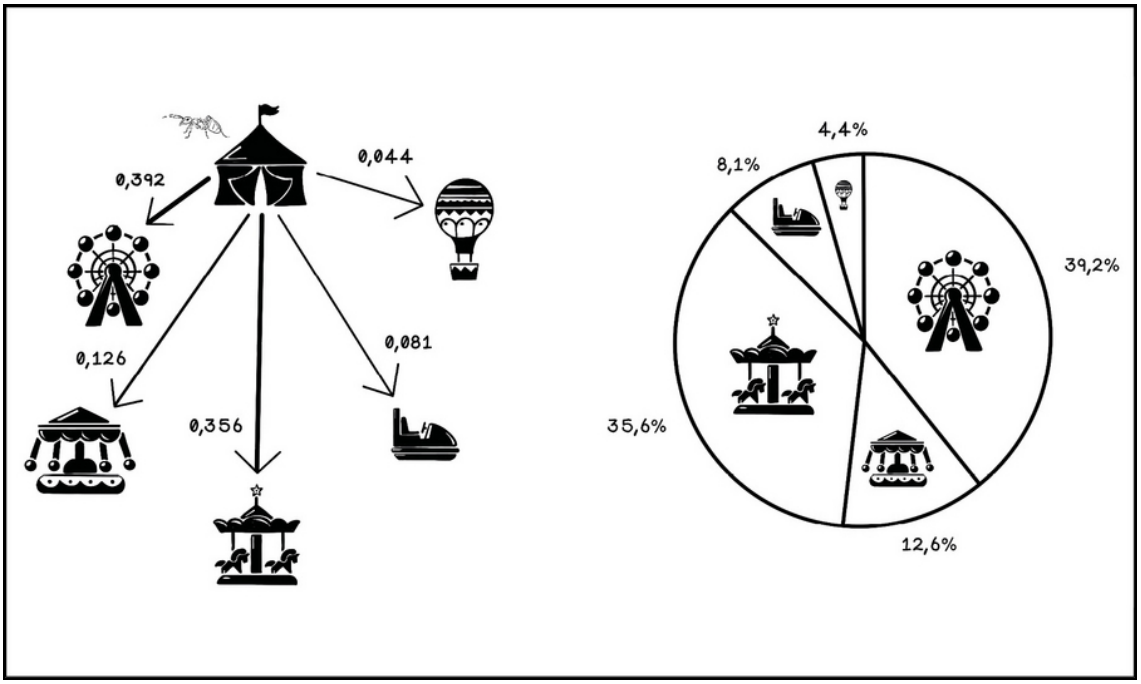


Figura 6.20 Le probabilità finali che ogni attrazione venga selezionata.

Ricordate che vengono considerati solo i percorsi disponibili, ovvero i percorsi non ancora esplorati. La Figura 6.21 illustra i possibili percorsi dal circo, esclusa la ruota panoramica, già visitata. La Figura 6.22 mostra i calcoli delle probabilità per i percorsi.

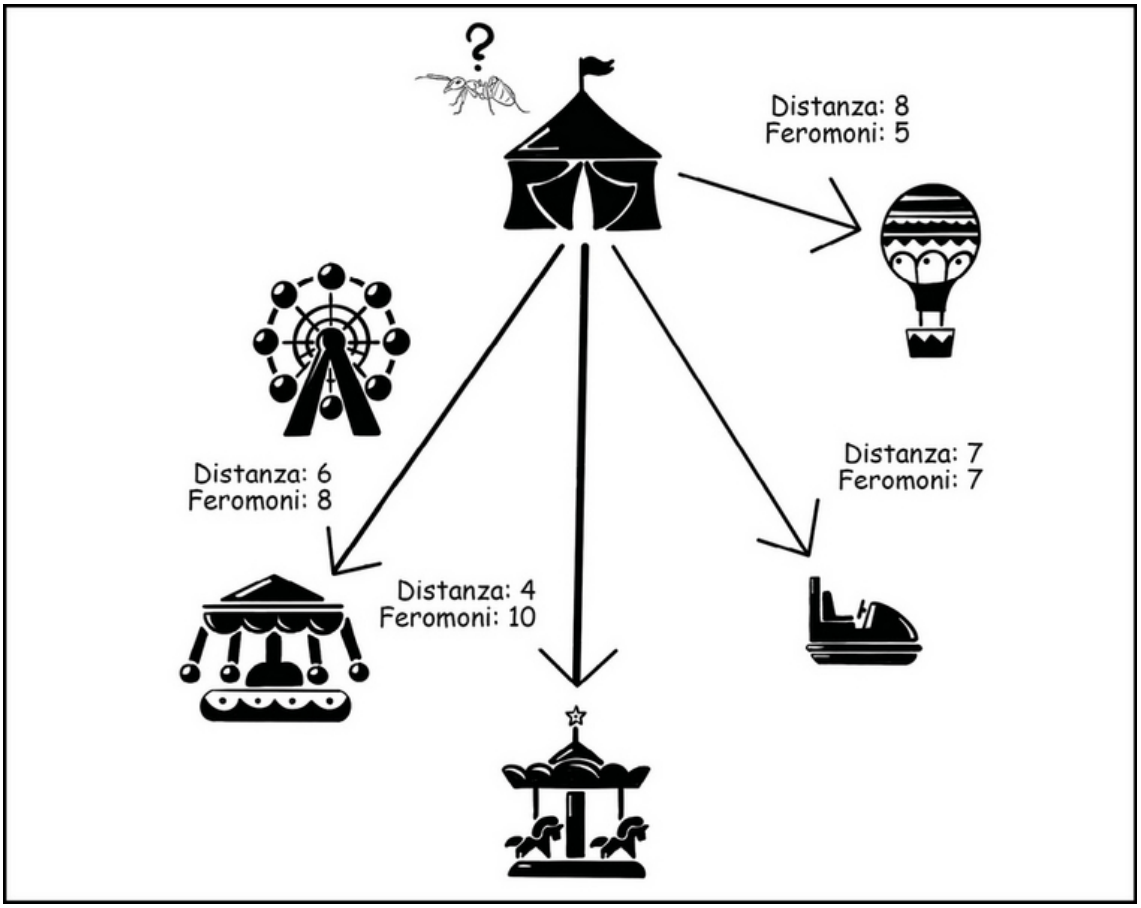


Figura 6.21 Esempio di possibili percorsi dal circo, escluse le attrazioni visitate.

La decisione della formica ora è quella rappresentata nella Figura 6.23.

Pseudocodice

Lo pseudocodice per calcolare le probabilità di visitare le possibili attrazioni rispecchia direttamente le funzioni matematiche che abbiamo elaborato. Ecco alcuni aspetti interessanti di questa implementazione.

- *Determinare le attrazioni ancora da visitare*: poiché la formica può già aver visitato delle attrazioni, non deve tornarvi. L'array `possible_attractions` memorizza questo valore rimuovendo le `visiting_attractions` dall'elenco completo delle attrazioni: `all_attractions`.

- *Tre variabili per memorizzare il risultato dei calcoli delle probabilità*: `possible_indexes` memorizza gli indici delle attrazioni; `possible_probabilities` memorizza le probabilità per il rispettivo indice; `total_probabilities` memorizza la somma di tutte le probabilità, che dovrebbe essere uguale a `1` quando la funzione è completa. Queste tre strutture di dati potrebbero essere convenzionalmente rappresentate da una classe, per produrre codice più pulito.

```
visit_probabilistic_attraction(pheromone_trails, attraction_count, ant,
alpha, beta):
  let current_attraction equal ant.visited_attractions[-1]
  let all_attractions equal range(0, attraction_count)
  let possible_attractions equal all_attractions - ant.visited_attractions
  let possible_indexes equal empty array
  let possible_probabilities equal empty array
  let total_probabilities equal 0
  for attraction in possible_attractions:
      append attraction to possible_indexes
                              let pheromones on path equal
math.pow(pheromone_trails[current_attraction] [attraction],
alpha)
                    let heuristic for path equal math.pow(1 /
attraction_distances[current_attraction] [attraction], beta)
      let probability equal pheromones_on_path * heuristic_for_path
      append probability to possible_probabilities
      add probability to total_probabilities
   let possible_probabilities equal(probability / total_probabilities for
probability in possible_probabilities)
  return(possible_indexes, possible_probabilities)
```

Torniamo di nuovo alla selezione a ruota della roulette. Tale funzione prende come input le possibili probabilità e gli indici delle attrazioni. Genera un elenco di “fette”, ciascuna delle quali include l’indice dell’attrazione nell’elemento 0, l’inizio della sezione nell’elemento 1 e la fine della sezione nell’elemento 2. Tutte le “fette” contengono un inizio e una fine compresi fra 0 e 1 Viene generato un numero casuale compreso fra 0 e 1 e le sezioni in cui cade vengono elette vincitrici:

```
roulette_wheel_selection(possible_indexes, possible_probabilities,
possible_attraction_count):
  let slices equal empty array
  let total equal 0
  for i in range (0, possible_attractions_count):
                      append [possible_indexes[i], total, total +
possible_probabilities[i]] to slides
      total += possible_probabilities[i]
  let spin equal random(0, 1)
   let result equal [slice for slice in slices if slice[1] < spin <= slice
[2]]
  return result
```

Ora che abbiamo le probabilità di selezionare le diverse attrazioni da visitare, useremo la selezione a ruota della roulette.

Per ricapitolare, la selezione a ruota della roulette (dai Capitoli 3 e 4) offre diverse probabilità alle fette di una ruota in base alla loro qualità. La ruota viene "girata" e viene selezionato un individuo. Una qualità più elevata offre a un individuo una fetta più ampia della ruota, come mostra la Figura 6.23. Il processo di scelta e di visita delle attrazioni prosegue per ogni formica, finché ognuna di esse non ha visitato una sola volta tutte le attrazioni.

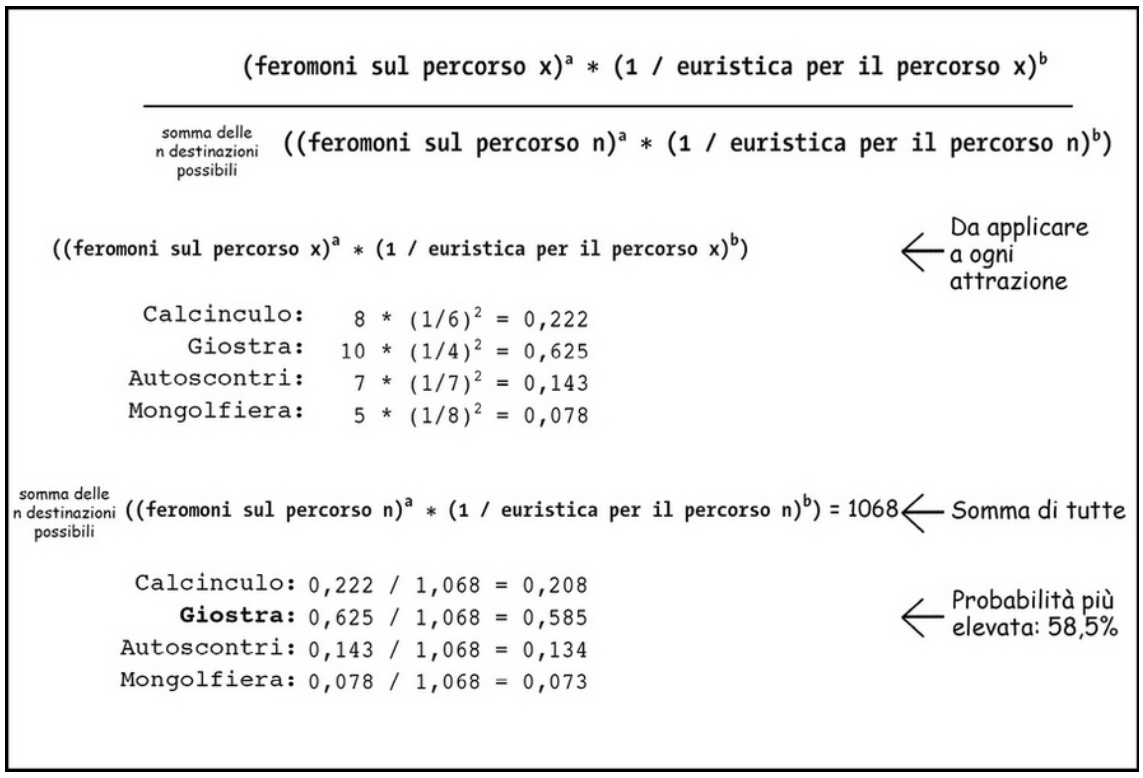


Figura 6.22 Calcoli delle probabilità per i percorsi.

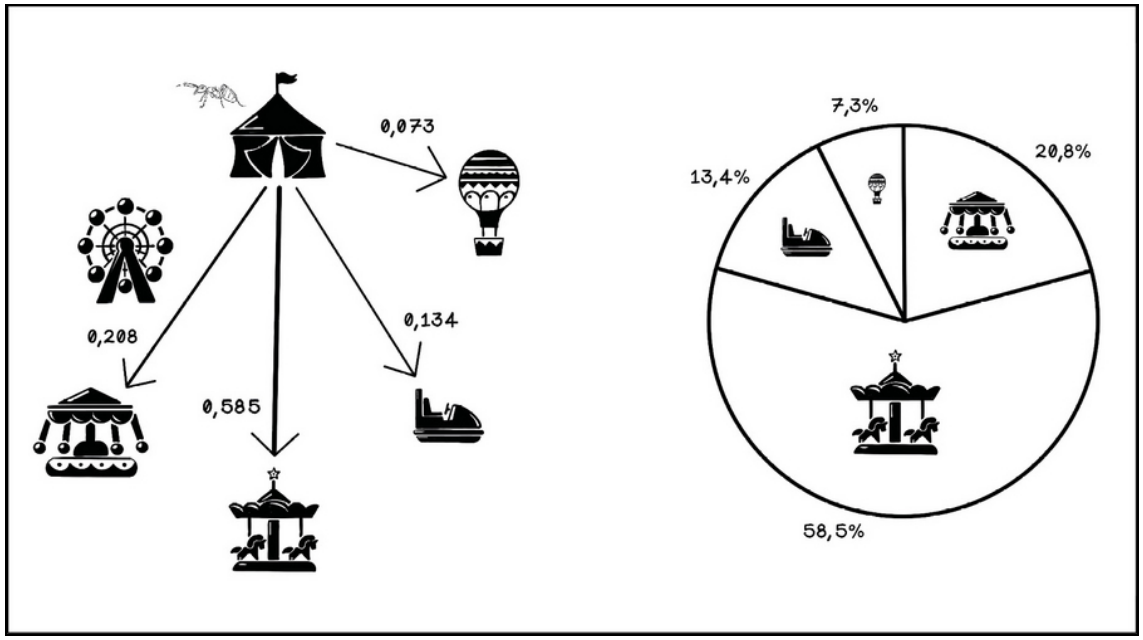
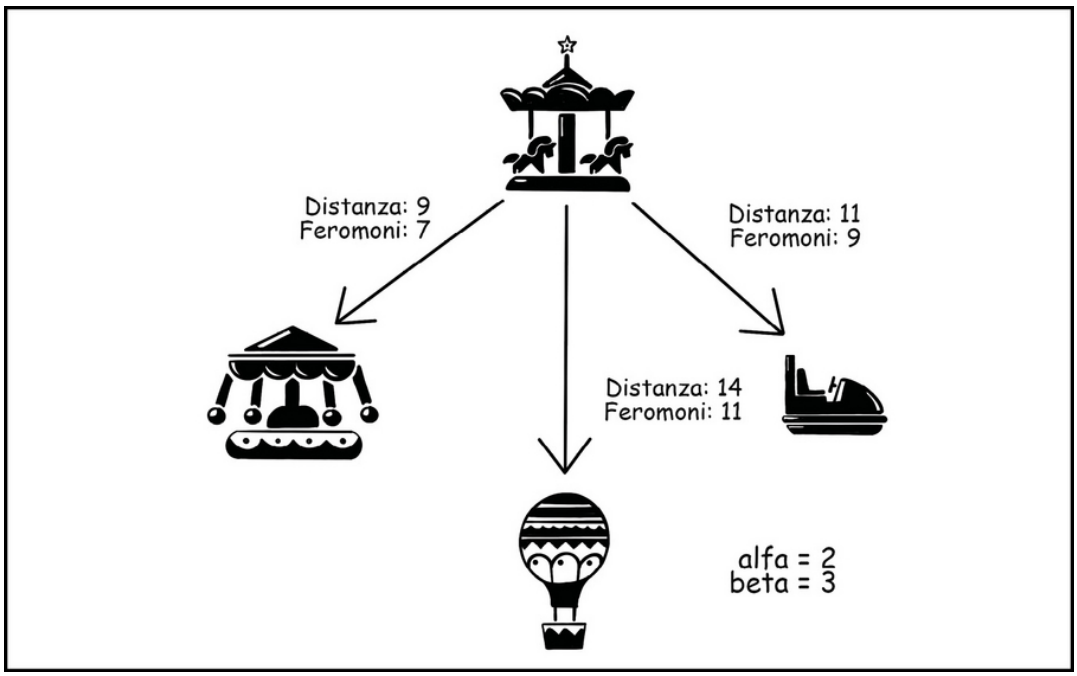


Figura 6.23 La probabilità finale che ogni attrazione venga selezionata.

Esercizio: determinare le probabilità di visitare le attrazioni con le seguenti

informazioni



Soluzione

$$\frac{(\text{feromoni sul percorso x})^a * (1 / \text{euristica per il percorso x})^b}{\text{somma delle n destinazioni possibili } ((\text{feromoni sul percorso n})^a * (1 / \text{euristica per il percorso n})^b)}$$

$((\text{feromoni sul percorso x})^a * (1 / \text{euristica per il percorso x})^b)$

Calcinculo: $7^2 * (1/9)^2 = 0,067$
Autoscontri: $9^2 * (1/11)^2 = 0,061$
Mongolfiera: $11^2 * (1/14)^2 = 0,044$

somma delle n destinazioni possibili $((\text{feromoni sul percorso n})^a * (1 / \text{euristica per il percorso n})^b) = 0,172$

Calcinculo: 0,067 / 1,172 = 0,39
Autoscontri: 0,061 / 1,172 = 0,355
Mongolfiera: 0,044 / 1,172 = 0,256

Aggiorna le scie di feromoni

Ora che le formiche hanno completato il tour di tutte le attrazioni, si sono lasciate dietro i feromoni, cosa che modifica le scie di feromoni fra le attrazioni (Figura 6.24).

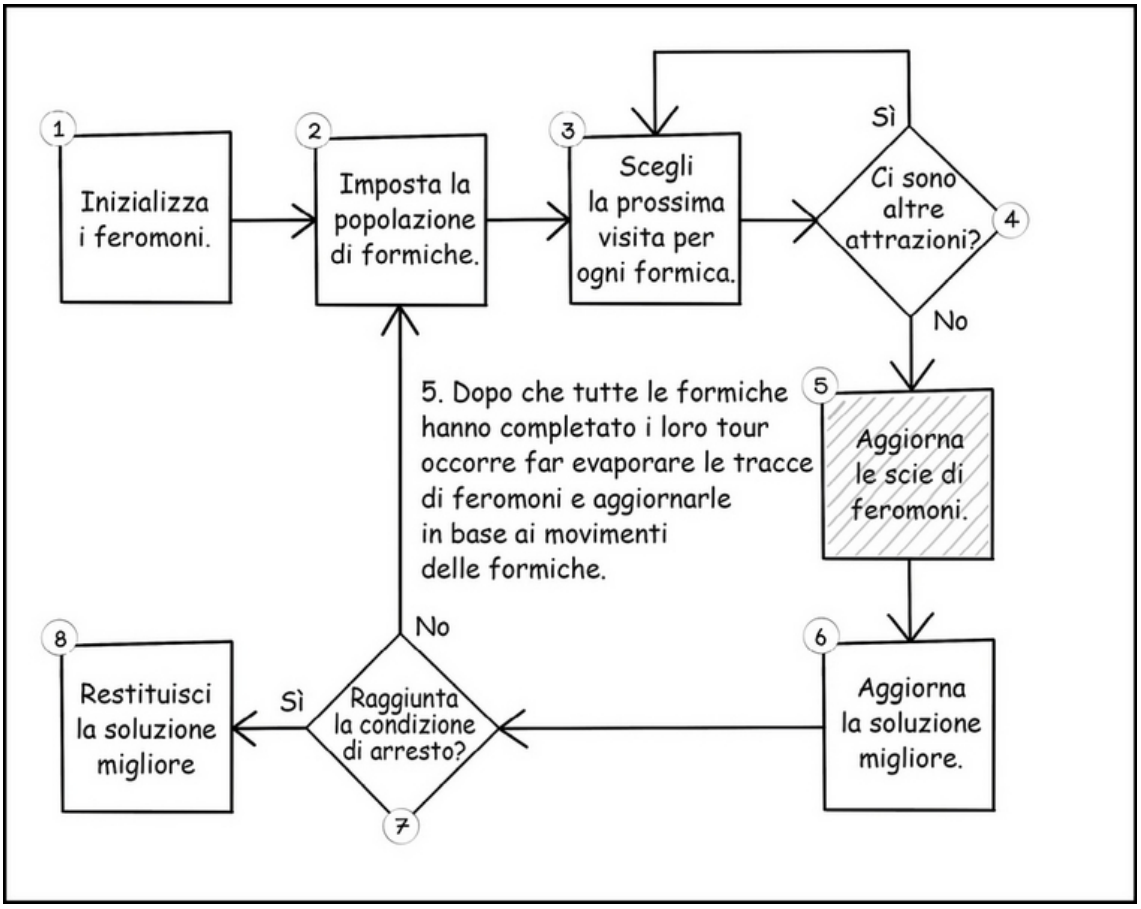


Figura 6.24 Aggiorna le scie di feromoni.

Aggiornamento dei feromoni dovuto all'evaporazione

Anche il concetto di evaporazione si ispira alla natura. Nel tempo, le scie di feromoni perdono la loro intensità. I feromoni vengono aggiornati moltiplicando i loro valori correnti per un fattore di evaporazione, un parametro che può essere regolato per modificare le prestazioni dell'algoritmo in termini di esplorazione e

approfondimento. La Figura 6.25 illustra le scie dei feromoni aggiornate a causa dell'evaporazione.

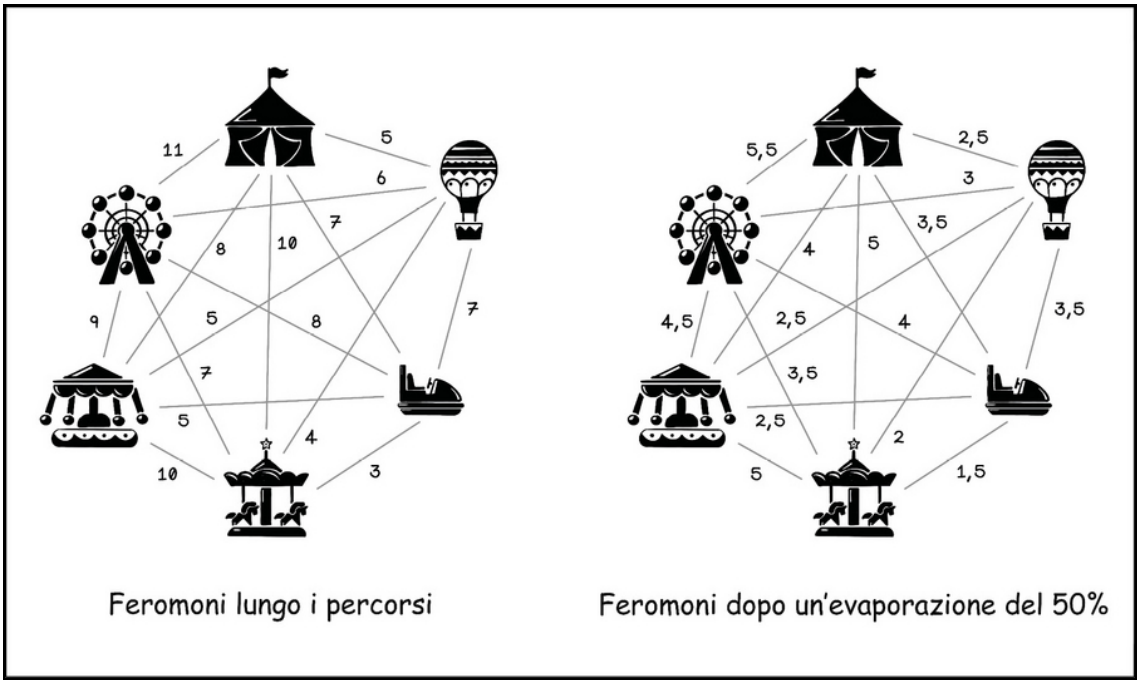


Figura 6.25 Esempio di aggiornamento delle tracce di feromoni per l'evaporazione.

Aggiornamento dei feromoni sulla base dei tour delle formiche

I feromoni vengono aggiornati in base alle formiche che si sono spostate lungo i percorsi. Se più formiche si muovono su un determinato percorso, vi rimarranno depositati più feromoni.

Ogni formica contribuisce con il proprio valore alla quota di feromoni lungo ogni percorso che ha seguito. L'effetto è che le formiche che hanno trovato soluzioni migliori hanno una maggiore influenza sulla scelta dei percorsi. La Figura 6.26 illustra le scie aggiornate dei feromoni sulla base dei movimenti delle formiche.

Pseudocodice

La funzione `update_pheromones` applica due concetti importanti alle scie di feromoni. Innanzitutto, l'attuale intensità dei feromoni viene fatta evaporare in

base al tasso di evaporazione. Se, per esempio, il tasso di evaporazione è di 0,5 l'intensità diviene la metà. La seconda operazione aggiunge i feromoni dovuti ai movimenti delle formiche su quel percorso. La quantità di feromoni fornita da ciascuna formica è determinata dalla qualità della formica, che in questo caso è la distanza totale da essa percorsa:

```
update_pheromones(evaporation_rate, pheromone_trails, attraction_count):
  for x in range(0, attraction_count):
      for y in range(0, attraction_count):
                let pheromone_trails[x][y] equal pheromone_trails[x][y] *
evaporation_rate
          for ant in ant_colony:
              pheromone_trails[x][y] += 1 / ant.get_distance_traveled()
```

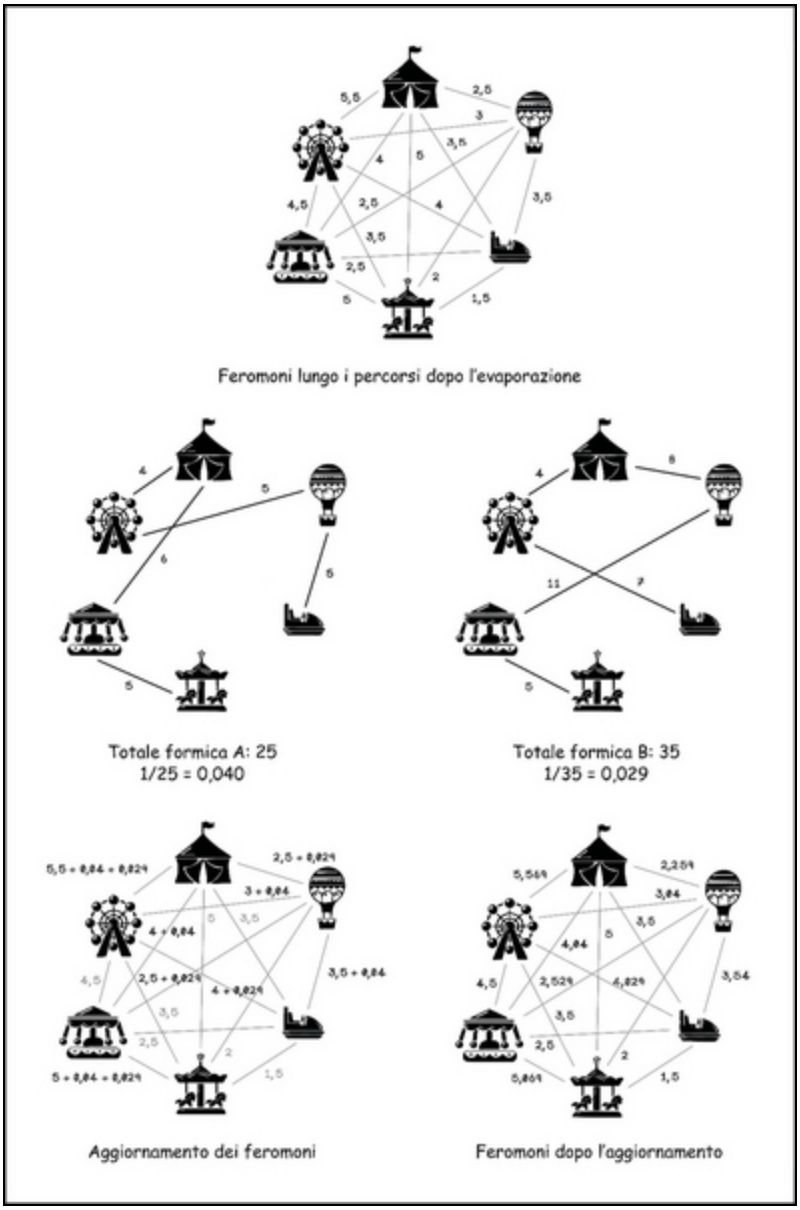
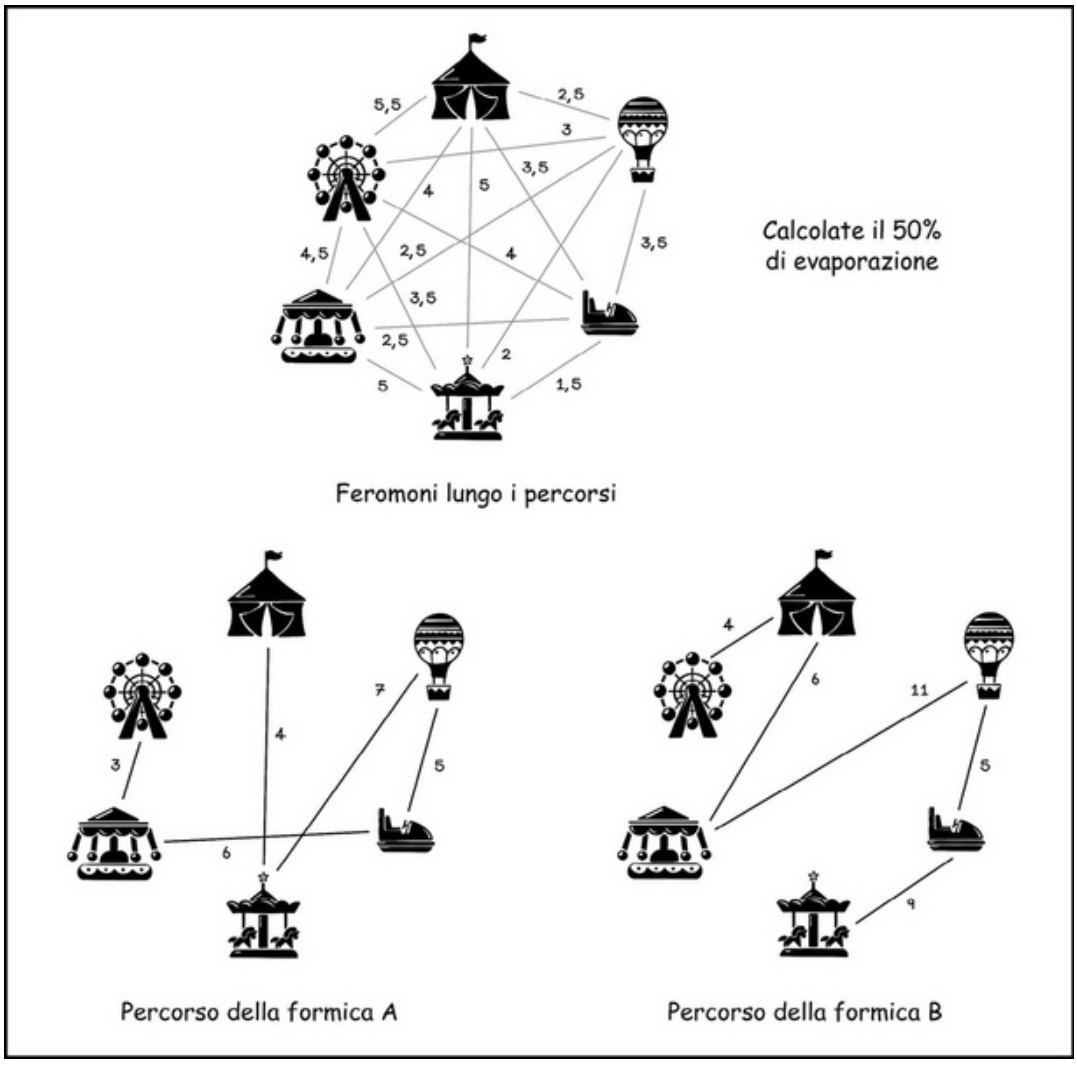
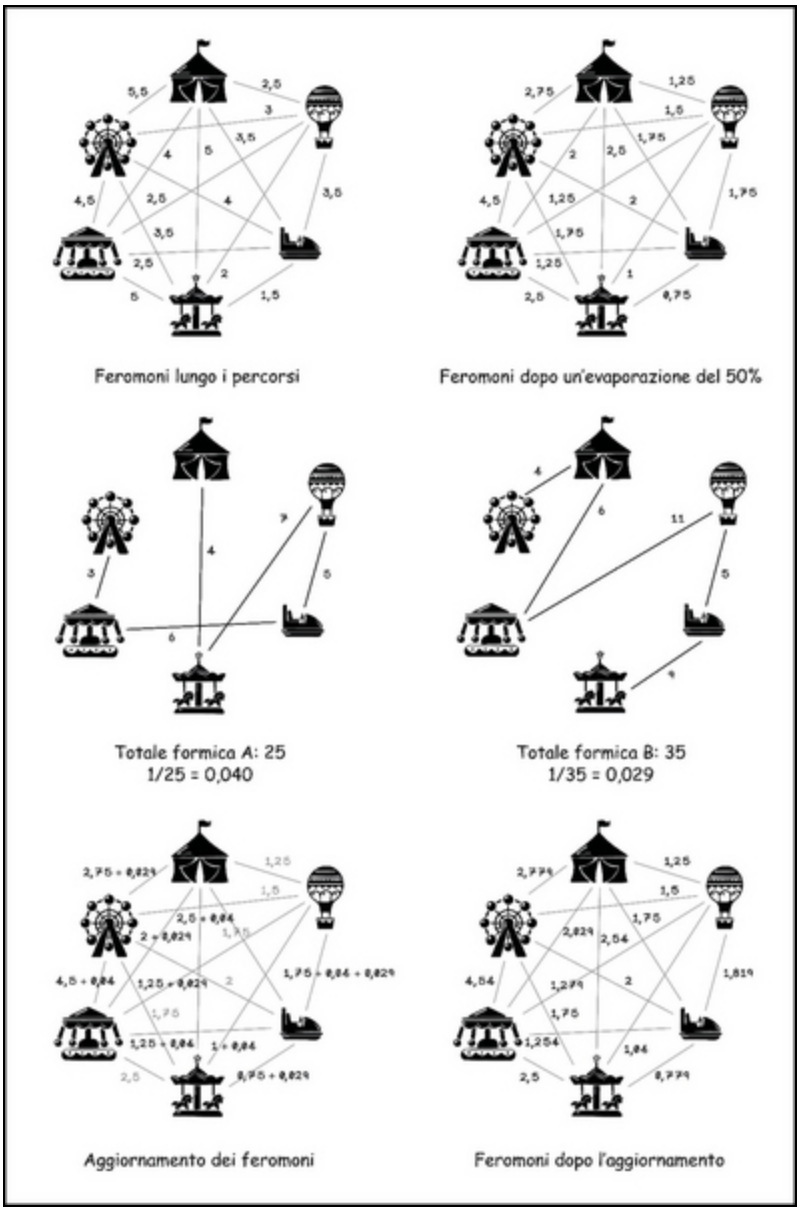


Figura 6.26 Aggiornamenti dei feromoni basati sui movimenti delle formiche.

Esercizio: calcolare l'aggiornamento dei feromoni dato il seguente scenario



Soluzione



Aggiorna la soluzione migliore

La soluzione migliore è descritta dalla sequenza di visite alle attrazioni che presenta la distanza totale più bassa (Figura 6.27).

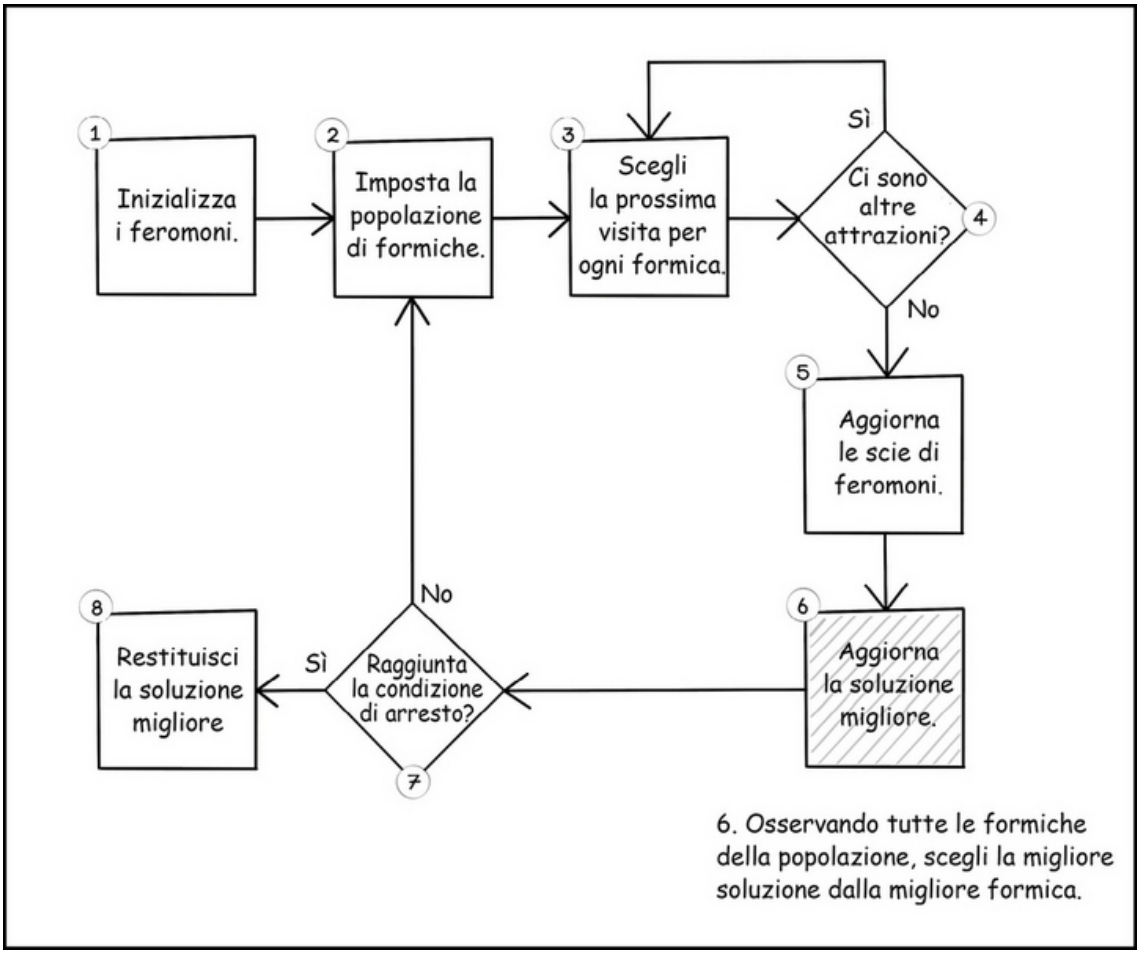


Figura 6.27 Aggiorna la soluzione migliore.

Pseudocodice

Dopo un'iterazione, dopo che ogni formica ha completato un tour (un tour è completo quando una formica ha visitato ogni attrazione), deve essere determinata la migliore formica della colonia. Per farlo, troviamo la formica che ha percorso la distanza totale più breve e la eleggiamo come la nuova migliore formica della colonia:

```
get_best(ant_population, previous_best_ant):
  let best_ant equal previous_best_ant
  for ant in ant_population:
      let distance_traveled equal ant.get_distance_traveled()
      if distance_traveled < best_ant.best_distance:
          let best_ant equal ant
  return best_ant
```

Determina i criteri di arresto

L'algoritmo si ferma dopo varie iterazioni: concettualmente, il numero di tour che il gruppo di formiche conclude. Dieci iterazioni fanno sì che ogni formica compia 10 tour; ogni formica visiterà ogni attrazione una volta e lo farà per dieci volte (Figura 6.28).

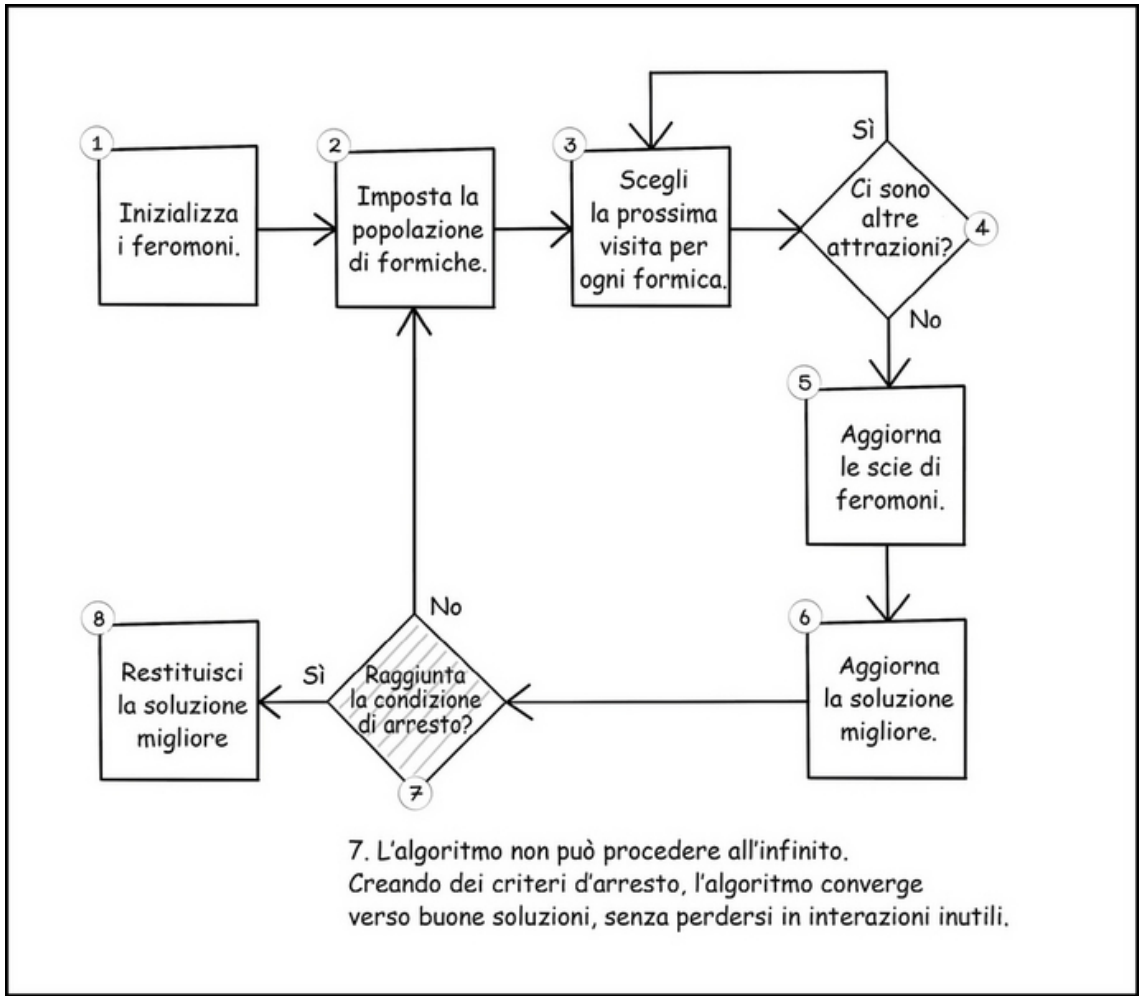


Figura 6.28 Condizione di arresto raggiunta?

I criteri di arresto per l'algoritmo di ottimizzazione a colonia di formiche possono differire in base al dominio del problema da risolvere. In alcuni casi, i limiti realistici sono noti. Quando invece non sono noti, sono disponibili le seguenti opzioni.

- *Fermati una volta raggiunto un numero predefinito di iterazioni*. In questo scenario, definiamo un numero totale di iterazioni totali dell'algoritmo. Se vengono definite 100 iterazioni, ciascuna formica completerà 100 tour prima che l'algoritmo termini.
- *Fermati quando la soluzione migliore ristagna*. In questo scenario, la soluzione migliore dopo ogni iterazione viene confrontata con la soluzione migliore precedente. Se la soluzione non migliora dopo un certo numero di iterazioni, l'algoritmo termina. Se l'iterazione 20 ha prodotto una soluzione di qualità 100 e tale qualità si ripete fino all'iterazione 30, è probabile (ma non garantito) che non esista una soluzione migliore.

Pseudocodice

La funzione `solve` lega tutto insieme e dovrebbe dare un'idea della sequenza di operazioni e del ciclo di vita complessivo dell'algoritmo. Notate che l'algoritmo viene eseguito per un certo numero totale di iterazioni. La colonia di formiche viene inoltre riportata al suo punto di partenza all'inizio di ogni iterazione, e dopo ogni iterazione viene eletta una nuova formica migliore:

```
solve(total_iterations,       evaporation_rate,       number_of_ants_factor,
attraction_count):
  let pheromone_trails equal setup_pheromones()
  let best_ant equal Nothing
  for i in range(0, total_iterations):
      let ant_colony equal setup_ants(number_of_ants_factor)
      for r in range(0, attraction_count - 1):
          move_ants(ant_colony)
                    update_pheromones(evaporation_rate,   pheromone_trails,
attraction_count)
      let best_ant equal get_best(ant_colony)
```

Possiamo modificare diversi parametri per alterare l'esplorazione e l'approfondimento dell'algoritmo di ottimizzazione a colonia di formiche. Questi parametri influenzano quanto tempo impiegherà l'algoritmo a trovare una buona soluzione. Un po' di casualità fa sempre bene all'esplorazione. Bilanciare la ponderazione fra euristiche e feromoni influenza il "carattere" delle formiche: se favoriamo

l'euristica tenteranno una ricerca avida, altrimenti si fideranno maggiormente dei feromoni. Anche il tasso di evaporazione influenza questo equilibrio. Il numero di formiche e il numero totale di iterazioni che fanno influenza la qualità di una soluzione. Aggiungendo più formiche e più iterazioni, sarà necessario svolgere più calcoli. In base al problema in questione, il tempo di calcolo può influenzare questi parametri (Figura 6.29).

```
Definisce la probabilità che le formiche scelgano di visitare un'attrazione
casuale (0,0-1,0, ovvero 0%-100%).
RANDOM_ATTRACTION_FACTOR = 0.3

Definisce il peso dei feromoni sul percorso per le preferenze delle formiche.
ALPHA = 4

Definisce il peso dell'euristica del percorso per le preferenze delle formiche.
BETA = 7

Definisce la percentuale delle formiche della colonia sulla base del numero totale delle attrazioni.
NUMBER_OF_ANTS_FACTOR = 0.5

Definisce il numero di tour che devono essere completati dalle formiche.
TOTAL_ITERATIONS = 1000

Definisce il tasso di evaporazione dei feromoni (0,0-1,0, ovvero 0%-100%).
EVAPORATION_RATE = 0.4
```

Figura 6.29 Parametri che possono essere modificati nell'algoritmo di ottimizzazione a colonia di formiche.

Ora avete un'idea di come funzionano gli algoritmi di ottimizzazione a colonia di formiche e di come possono essere utilizzati per risolvere un problema come quello del luna park. Il prossimo paragrafo descrive alcuni altri possibili casi d'uso. Forse

questi esempi possono aiutarvi a trovare specifici usi dell'algoritmo nel vostro lavoro.

Casi d'uso per gli algoritmi di ottimizzazione a colonia di formiche

Gli algoritmi di ottimizzazione a colonia di formiche sono versatili e utili in diverse applicazioni del mondo reale. Queste applicazioni di solito si concentrano su complessi problemi di ottimizzazione, come i seguenti.

- *Ottimizzazione del percorso*: i problemi relativi a percorsi di solito includono diverse destinazioni che devono essere visitate con alcuni vincoli. In un esempio di logistica, può trattarsi della distanza fra le destinazioni, delle condizioni del traffico, dei tipi di pacchi da consegnare e degli orari della giornata. Sono vincoli importanti che devono essere considerati per ottimizzare le attività. Gli algoritmi di ottimizzazione a colonia di formiche possono essere utilizzati per risolvere questo problema. Il problema è simile a quello del luna park esplorato in questo capitolo, ma è probabile che la funzione euristica sia più complessa e specifica.
- *Pianificazione del lavoro*: la pianificazione del lavoro riguarda quasi tutti i settori. I turni degli infermieri sono importanti, per garantire che possa essere fornita una buona assistenza sanitaria. I lavori computazionali sui server devono essere pianificati in modo ottimale, per massimizzare l'utilizzo dell'hardware senza sprechi. Gli algoritmi di ottimizzazione a colonia di formiche possono essere utilizzati per risolvere questi problemi. Le formiche non visiteranno luoghi, ma attività, con sequenze differenti. La funzione euristica includerà vincoli e regole specifici per il

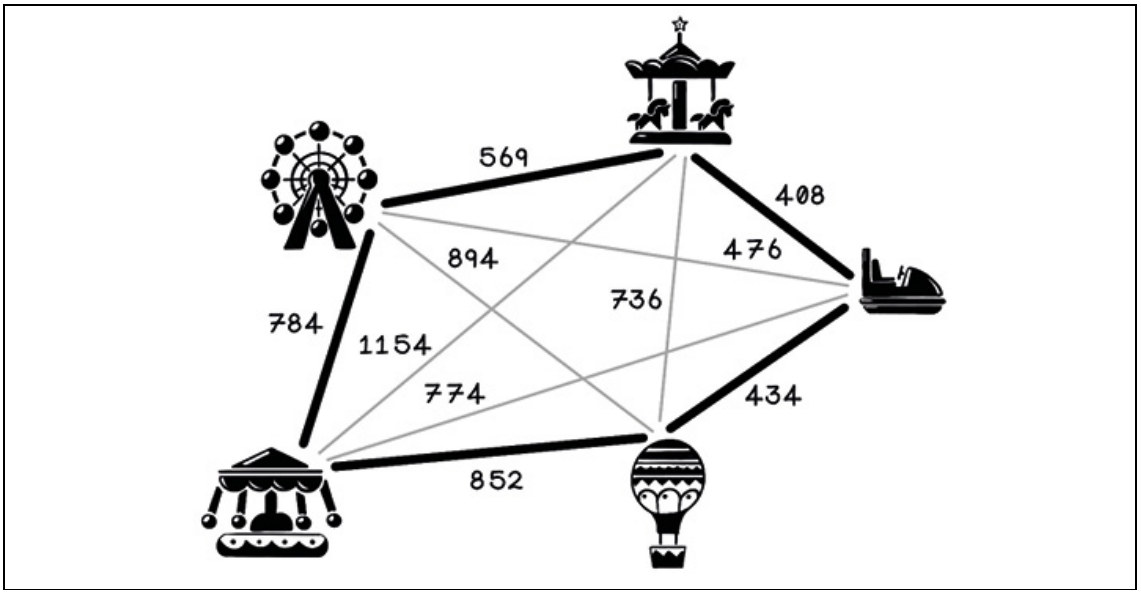
contesto della pianificazione dei lavori. Gli infermieri, per esempio, hanno bisogno di giorni liberi per prevenire l'affaticamento; su un server dovrebbero essere favoriti i lavori con priorità elevate.

- *Elaborazione delle immagini*: l'algoritmo di ottimizzazione a colonia di formiche può essere utilizzato per il rilevamento dei bordi nell'elaborazione delle immagini. Un'immagine è composta da diversi pixel adiacenti e le formiche si spostano da un pixel all'altro, lasciando dietro di sé scie di feromoni. Le formiche rilasciano feromoni più forti in base all'intensità dei colori dei pixel, producendo scie di feromoni lungo i bordi degli oggetti che contengono un contrasto più elevato. Questo algoritmo traccia essenzialmente il contorno dell'immagine, tramite il rilevamento dei bordi. Le immagini potrebbero richiedere una pre-elaborazione per decolorare l'immagine in scala di grigi, in modo da poter confrontare in modo coerente le tonalità dei pixel.

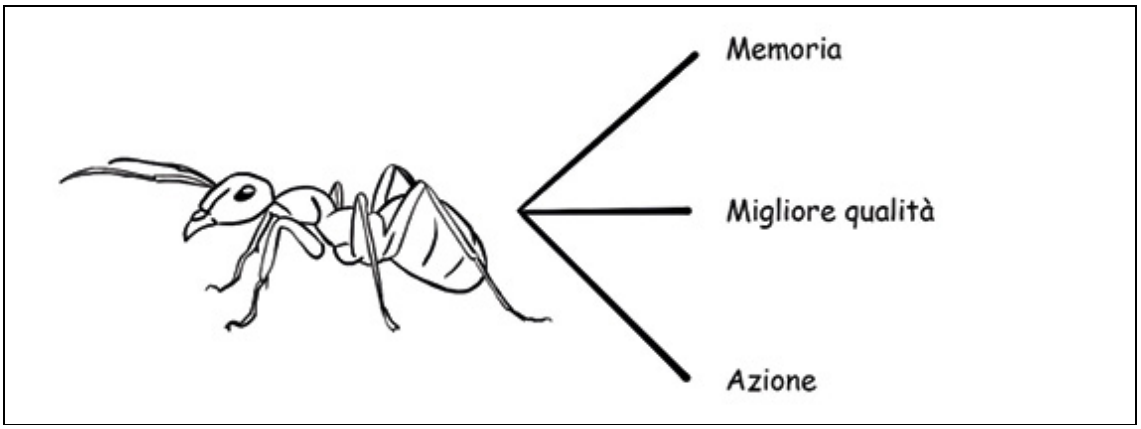
Riepilogo

Gli algoritmi di ottimizzazione a colonia di formiche usano feromoni e informazioni euristiche.

- Questi algoritmi sono utili per problemi di ottimizzazione, come la ricerca del percorso più breve o lo scheduling dei task.



- Le formiche sono dotate dei concetti di memoria e prestazioni e possono svolgere azioni.



- Per calcolare la probabilità che i percorsi vengano selezionati, vengono attribuiti appositi pesi all'euristica e ai feromoni sui percorsi.

Intelligenza di sciame: particelle

Che cos'è l'ottimizzazione a sciame di particelle?

L'*ottimizzazione a sciame di particelle* è un altro algoritmo a sciame. L'intelligenza di sciame si basa sul comportamento che emerge da molti individui, che diventano in grado di risolvere problemi difficili come collettività. Abbiamo visto nel Capitolo 6 come le formiche possano trovare i percorsi più brevi fra le destinazioni attraverso l'uso dei feromoni.

Gli stormi di uccelli sono un altro esempio ideale di algoritmo di sciame in natura. Osservando un singolo uccello volare, vediamo che impiega diverse manovre e tecniche per conservare l'energia, come virare e planare nell'aria o sfruttare le correnti per dirigersi nella direzione desiderata. Questo comportamento indica un livello primitivo di intelligenza insita nel singolo individuo. Ma gli uccelli hanno anche la necessità di migrare al cambio di stagione. In inverno c'è meno disponibilità di insetti e altro cibo. Anche i luoghi di nidificazione adatti diventano scarsi. Gli uccelli tendono ad affluire in zone più calde per approfittare di migliori condizioni meteorologiche, il che migliora la loro probabilità di sopravvivenza. La migrazione di solito non è un viaggio breve. Ci vogliono migliaia di chilometri di volo per arrivare in un'area dotata di condizioni adeguate. Quando gli uccelli percorrono queste lunghe distanze, tendono a radunarsi in stormi. Gli uccelli si

affollano perché c'è la forza del numero quando si affrontano i predatori; inoltre, il volo in gruppo consente di risparmiare energia. La formazione che osserviamo negli stormi di uccelli presenta diversi vantaggi. Un uccello grande e forte prenderà il comando e, quando batterà le ali, creerà un movimento di sollevamento per gli uccelli dietro di lui. Questi uccelli possono così volare consumando molta meno energia. Gli stormi possono cambiare leader se la direzione cambia o se il leader si affatica. Quando un uccello esce dalla formazione, incontra maggiori difficoltà nel volo, a causa della resistenza dell'aria e subito corregge il suo movimento per tornare in formazione. La Figura 7.1 illustra la formazione di uno stormo di uccelli; potreste aver già visto in cielo qualcosa di simile.

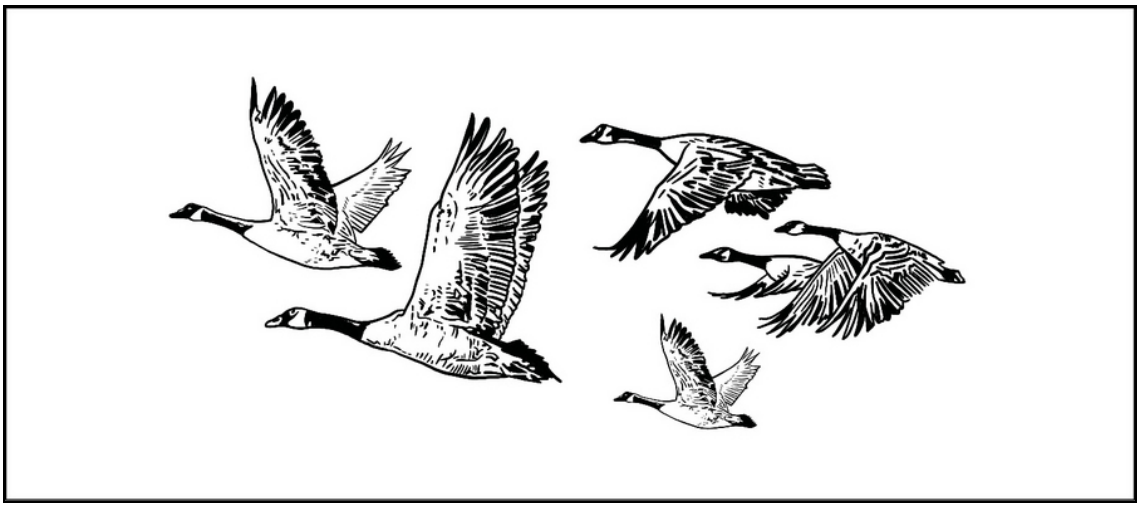


Figura 7.1 Un esempio di formazione di uno stormo di uccelli.

Craig Reynolds ha sviluppato un programma di simulazione nel 1987 per comprendere gli attributi del comportamento emergente negli stormi di uccelli e ha utilizzato le seguenti regole per guidare il gruppo, regole estratte dall'osservazione degli stormi di uccelli.

- *Allineamento*: un individuo deve seguire la direzione media dei suoi vicini, per garantire che il gruppo viaggi in una direzione

simile.

- *Coesione*: un individuo deve spostarsi verso la posizione media dei suoi vicini, per mantenere la formazione del gruppo.
- *Separazione*: un individuo deve evitare l'affollamento o la collisione con i suoi vicini, per garantire che gli individui non si scontrino, danneggiando il gruppo.

Le varianti prevedono regole aggiuntive, nel tentativo di simulare il comportamento dello stormo. La Figura 7.2 illustra il comportamento di un individuo in diversi scenari, nonché la direzione in cui è influenzato a muoversi per obbedire alla rispettiva regola. La regolazione del movimento è un equilibrio di questi tre principi mostrati nella figura.

L'ottimizzazione a sciame di particelle coinvolge un gruppo di individui che si trovano in punti differenti dello spazio della soluzione, dove tutti utilizzano i concetti dello sciame per trovare una soluzione ottimale in questo spazio. Questo capitolo approfondisce il funzionamento dell'algoritmo di ottimizzazione a sciame di particelle e mostra come può essere utilizzato per risolvere i problemi. Immaginate uno sciame di api che si dispiega alla ricerca di fiori e poi converge gradualmente su un'area che ha la maggiore densità di fiori. A mano a mano che più api trovano quei fiori, aumenta il numero di api attratte. Fondamentalmente, questo è un esempio di ottimizzazione a sciame di particelle (Figura 7.3).

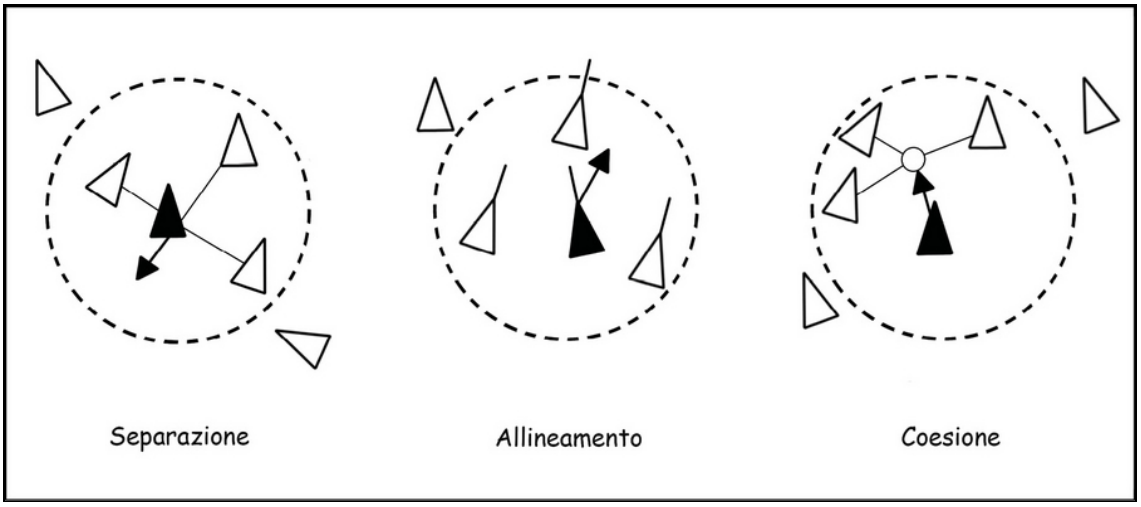


Figura 7.2 Regole che guidano lo sciame.

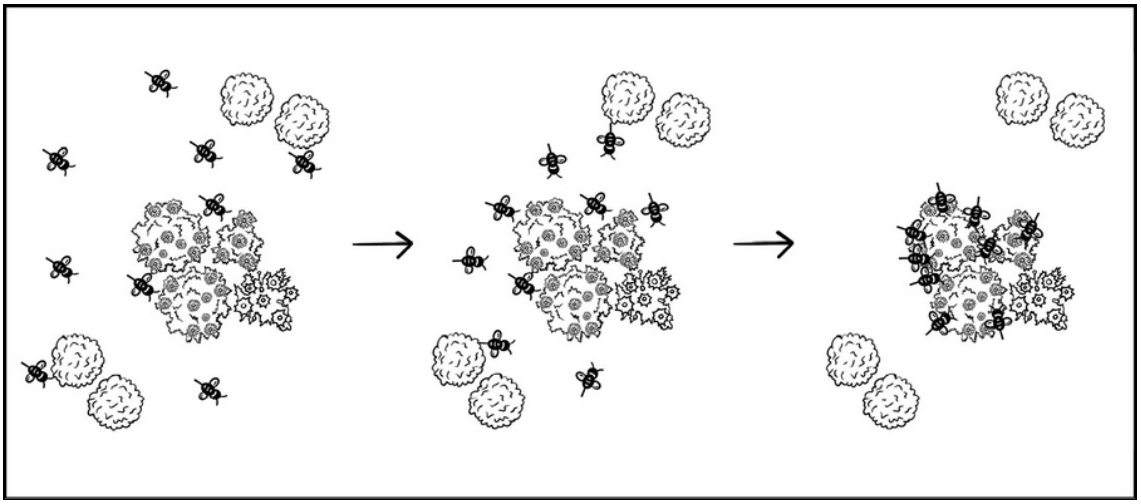


Figura 7.3 Uno sciame di api che converge verso il suo obiettivo.

I problemi di ottimizzazione sono già stati menzionati in diversi capitoli. Trovare il percorso ottimale attraverso un labirinto, determinare gli elementi ottimali per riempire uno zaino e trovare il percorso ottimale fra le attrazioni in un luna park sono tutti esempi di problemi di ottimizzazione. Ne abbiamo parlato senza però immergerci nei dettagli. Da questo capitolo in poi punteremo a una comprensione più approfondita dei problemi di ottimizzazione. Il prossimo paragrafo

espone alcune idee utili per individuare i problemi di ottimizzazione quando si presentano.

Problemi di ottimizzazione: una prospettiva leggermente più tecnica

Supponiamo di avere alcuni peperoni di diverse dimensioni. Di solito, i peperoni piccoli tendono a essere più piccanti dei grandi. Se tracciamo i peperoni su un grafico che mette in relazione dimensioni e piccantezza, potremmo ottenere qualcosa di simile a quanto rappresentato nella Figura 7.4.

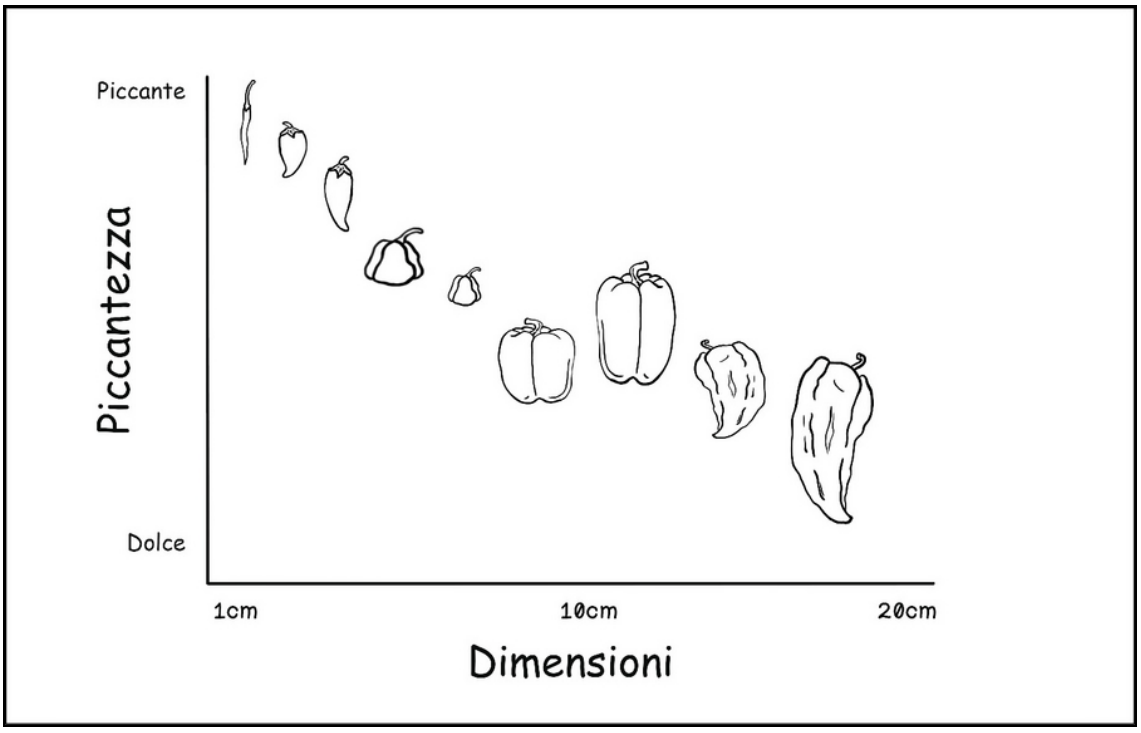


Figura 7.4 Piccantezza vs. dimensioni nei peperoni.

La figura mostra la dimensione di ciascun peperone e il suo grado di piccantezza. Ora, rimuovendo le immagini dei peperoni, individuando i punti di dati e tracciando una possibile curva fra di essi, otteniamo la

Figura 7.5. Se avessimo più peperoni, avremmo più punti di dati e la curva sarebbe più precisa.

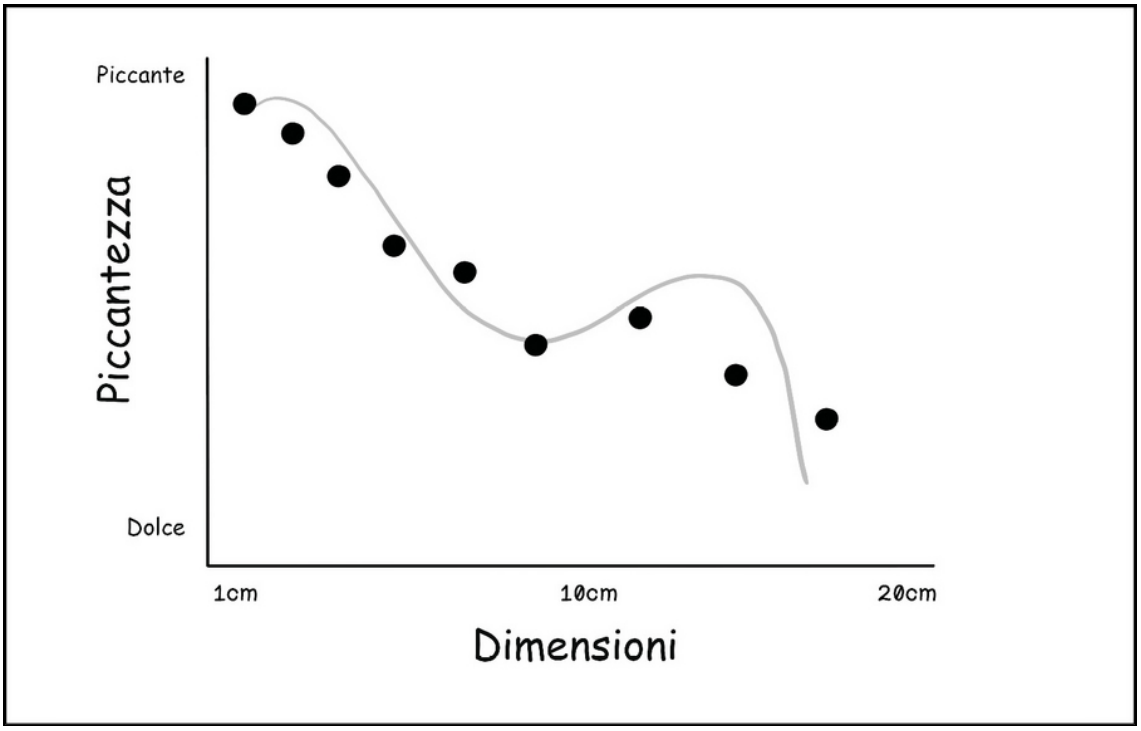


Figura 7.5 Curva della piccantezza rispetto alle dimensioni dei peperoni.

Questo esempio potrebbe far pensare a un problema di ottimizzazione. Se cercassimo il minimo procedendo da sinistra a destra, troveremmo continuamente diversi punti inferiori rispetto ai precedenti, ma poi incontriamo un punto superiore. Dovremmo fermarci? Se lo facessimo, perderemmo il vero minimo, che è l'ultimo punto di dati, il *minimo globale*.

La linea/curva di tendenza approssimata può essere rappresentata da una funzione, come quella mostrata nella Figura 7.6. Questa funzione stabilisce la piccantezza del peperone dove x è la dimensione del peperone.

$$f(x) = -(x-4)(x-0.2)(x-2)(x-3)+5$$

Figura 7.6 Una funzione di esempio per la piccantezza del peperone rispetto alle sue dimensioni.

I problemi concreti, in genere, prevedono migliaia di punti di dati, e l'output minimo della funzione non è così chiaro come questo esempio. Gli spazi di ricerca sono enormi e difficili da risolvere a mano.

Notate che abbiamo utilizzato solo due proprietà del peperone per creare i punti di dati, il che ha prodotto una semplice curva. Se consideriamo una terza proprietà del peperone, come il colore, la rappresentazione dei dati cambia sensibilmente. Ora il grafico deve essere rappresentato in 3D e il trend deve essere rappresentato da una superficie anziché una curva. Una superficie è come una "coperta elastica" (Figura 7.7). Anche questa superficie è rappresentabile come una funzione, ma più complessa.

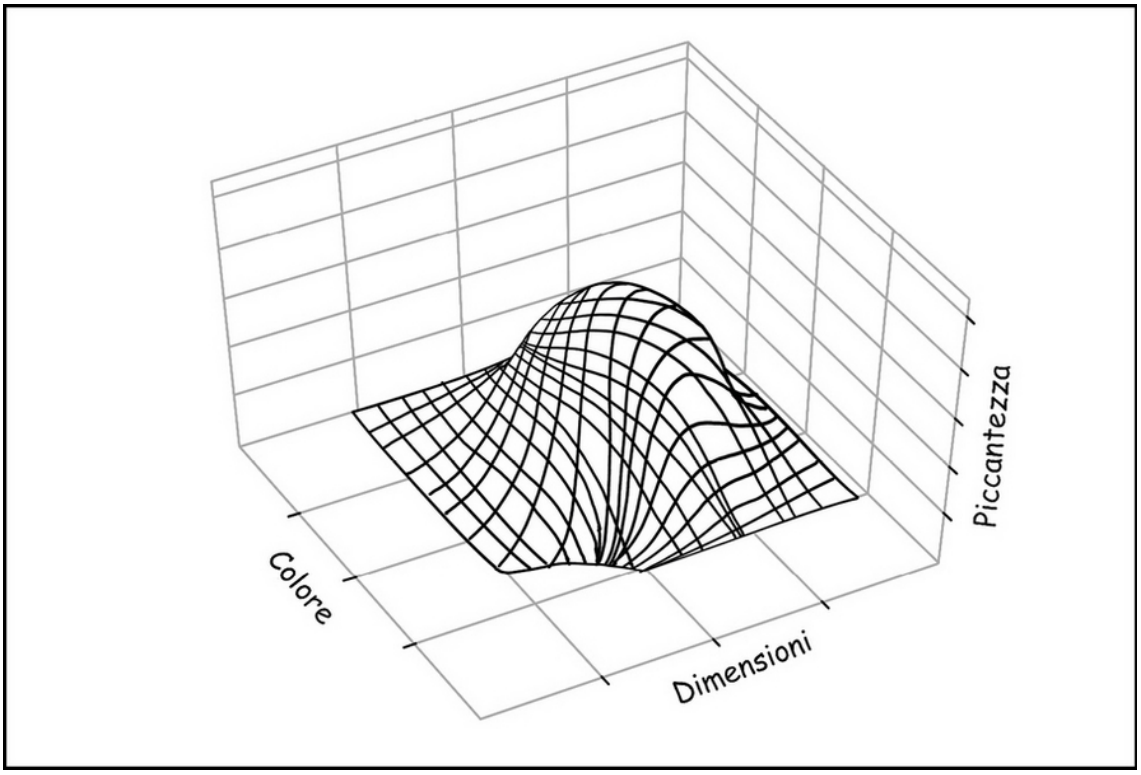


Figura 7.7 Piccantezza dei peperoni in relazione alle sue dimensioni e al suo colore.

Uno spazio di ricerca 3D potrebbe avere un aspetto abbastanza semplice, come nella Figura 7.7, o essere talmente complesso che tentare di ispezionarlo visivamente per trovare il minimo sarebbe quasi impossibile (Figura 7.8).

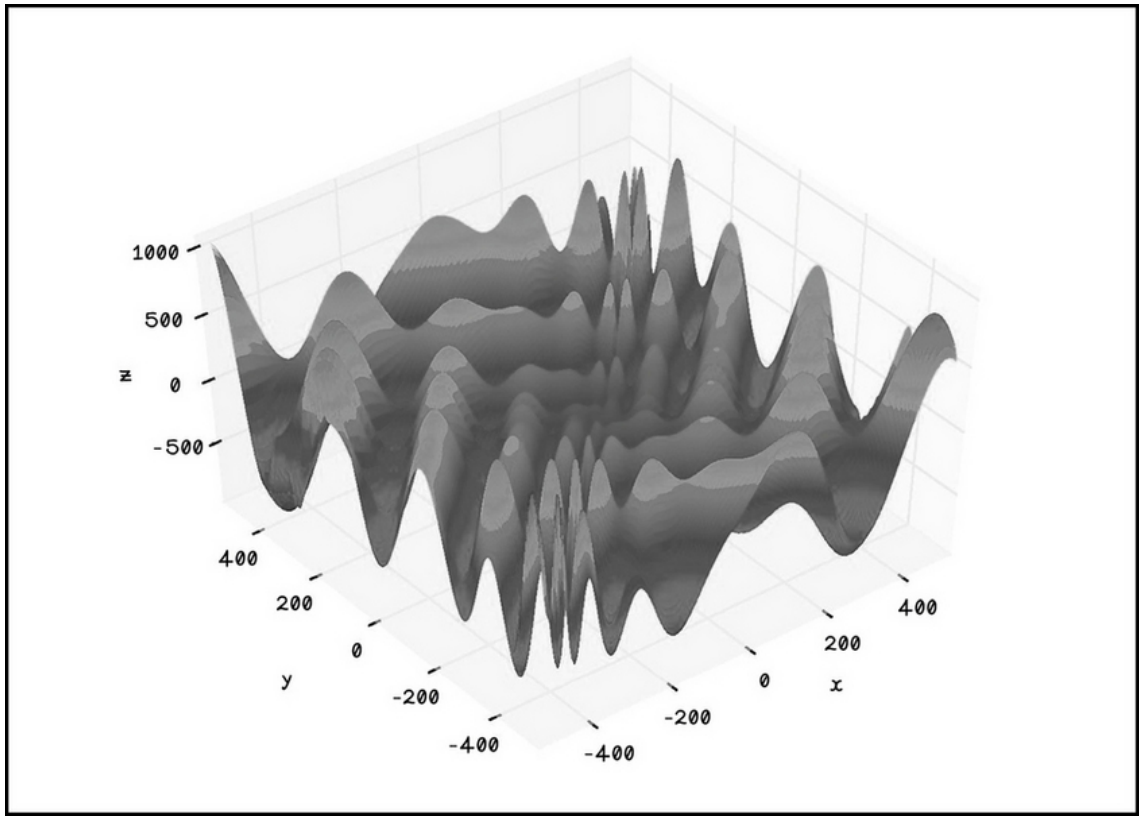


Figura 7.8 Una funzione visualizzata nello spazio 3D come un piano.

La Figura 7.9 mostra la funzione che rappresenta questo piano.

$$f(x, y) = -(y + 47) \sin \sqrt{\left| \frac{x}{2} + (y + 47) \right|} - x \sin \sqrt{|x - (y + 47)|}$$

Figura 7.9 La funzione che rappresenta la superficie rappresentata nella Figura 7.8.

Le cose si fanno più interessanti! Abbiamo esaminato tre attributi di un peperone: le sue dimensioni, il suo colore e la sua piccantezza. Di conseguenza, stiamo cercando in tre dimensioni. E se volessimo includere il luogo di crescita? Questo attributo renderebbe ancora più difficile la rappresentazione grafica e la conoscenza dei dati, perché la ricerca deve operare su quattro dimensioni. E se aggiungessimo l'età

del peperone e la quantità di fertilizzante utilizzato per la sua coltivazione, ci rimane un enorme spazio di ricerca a sei dimensioni e davvero non riusciremmo a immaginare come potrebbe essere questa ricerca. Anche questa ricerca è rappresentata da una funzione, ma ancora più complessa e difficile da risolvere per una persona.

Gli algoritmi di ottimizzazione a sciame di particelle sono particolarmente adatti a risolvere problemi di ottimizzazione difficili. Le particelle si distribuiscono nello spazio di ricerca multidimensionale, e collaborano per trovare buoni massimi o minimi.

Gli algoritmi di ottimizzazione a sciame di particelle sono particolarmente utili nei seguenti scenari.

- *Grandi spazi di ricerca*: ci sono molti punti di dati e possibilità di combinazioni.
- *Ricerca in spazi di dimensioni elevate*: c'è complessità data dalla numerosità delle dimensioni. È necessario considerare le molte dimensioni del problema per trovare una buona soluzione.

Esercizio: di quante dimensioni sarà lo spazio di ricerca del seguente scenario?

In questo scenario, dobbiamo determinare una buona città in cui vivere in base alla temperatura media minima durante l'anno, perché non amiamo il freddo. È anche importante che la popolazione sia inferiore alle 700.000 persone, perché le aree troppo affollate possono essere congestionate. Il prezzo medio degli immobili dovrebbe essere il più basso possibile, e più treni ci sono in città, meglio è.

Soluzione

Il problema in questo scenario prevede cinque dimensioni:

- temperatura media;
- dimensione della popolazione;
- prezzo medio degli immobili;
- numero di treni;
- risultato di questi attributi, che informeranno la nostra decisione.

Problemi risolvibili dall'ottimizzazione a sciame di particelle

Immaginate di sviluppare un drone e di utilizzare diversi materiali per creare il suo corpo e le sue eliche. Attraverso numerose prove, abbiamo scoperto che diverse quantità di due materiali specifici producono risultati diversi in termini di prestazioni ottimali nel sollevarsi e resistere ai forti venti. Questi due materiali sono l'alluminio, per il telaio, e la plastica, per le eliche. Troppo o troppo poco di entrambi i materiali si tradurrà in un drone con prestazioni scadenti. Diverse combinazioni producono un drone dotato di buone prestazioni ma solo una combinazione si traduce in un drone dalle prestazioni eccezionali.

La Figura 7.10 illustra i componenti in plastica e in alluminio. Le frecce illustrano le forze che influenzano le prestazioni del drone. In termini semplici, vogliamo trovare un buon rapporto fra plastica e alluminio per ottenere un drone che abbia una minima resistenza durante il sollevamento e una ridotta oscillazione nel vento. Quindi gli input sono plastica e alluminio e l'output è la stabilità del drone. Descriviamo la stabilità ideale come la riduzione della resistenza al decollo e dell'oscillazione nel vento.

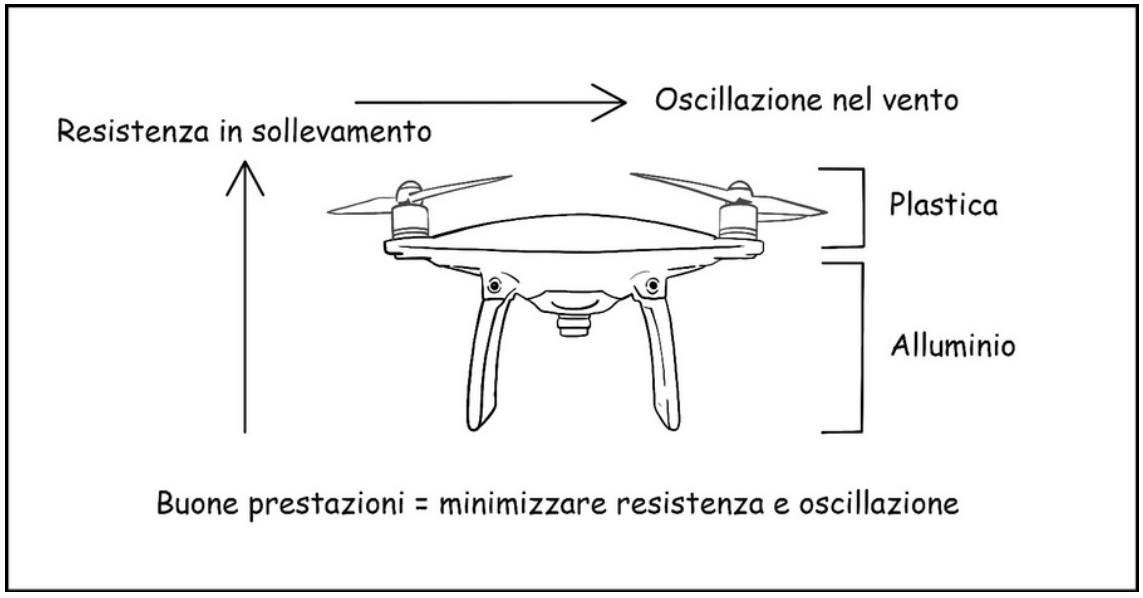


Figura 7.10 L'esempio di ottimizzazione del drone.

La precisione nel rapporto fra alluminio e plastica è importante, e la gamma di possibilità è ampia. In questo scenario, i ricercatori hanno trovato la funzione per calcolare il rapporto fra alluminio e plastica. Utilizzeremo questa funzione in un ambiente virtuale simulato, che sottopone a test la resistenza e l'oscillazione per trovare i valori migliori per ciascun materiale prima di produrre un nuovo prototipo di drone. Sappiamo anche che i rapporti massimo e minimo per i materiali sono rispettivamente 10 e -10. Questa funzione di calcolo della qualità è simile a un'euristica.

La Figura 7.11 mostra la funzione di valutazione della qualità per il rapporto fra alluminio (x) e plastica (y). Il risultato è un valore prestazionale basato su resistenza e oscillazione, dati i valori di input per x e y.

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Figura 7.11 La funzione di esempio per l'ottimizzazione dell'alluminio (x) e della plastica (y).

Come possiamo trovare le due quantità di alluminio e plastica necessarie per creare un buon drone? Una possibilità consiste nel provare ogni combinazione di valori di alluminio e plastica fino a trovare il miglior rapporto per il drone. Ma provate a immaginate la quantità di calcolo necessaria per trovare questo rapporto. Potremmo condurre un numero quasi infinito di calcoli prima di trovare una soluzione, se dovessimo provare ogni combinazione possibile. Dobbiamo calcolare il risultato per gli elementi rappresentati nella Tabella 7.1. Notate che i numeri negativi per alluminio e plastica sono bizzarri, in realtà; tuttavia, li stiamo usando per illustrare il funzionamento della funzione di valutazione della qualità utilizzata per ottimizzare questi valori.

Tabella 7.1 Possibili valori per le composizioni di alluminio e plastica.

Quante parti in alluminio? (x)	Quante parti di plastica? (y)
-0,1	1,34
-0,134	0,575
-1,1	0,24
-1,1645	1,432
-2,034	-0,65
-2,12	-0,874

0,743	-1,1645
0,3623	-1,87
1,75	-2,7756
…	…
-10 ≥ Alluminio ≥ 10	-10 ≥ Plastica ≥ 10

Questo calcolo andrà avanti per ogni combinazione possibile fra i vincoli ed è computazionalmente costoso, quindi è realisticamente impossibile risolvere questo problema con la forza bruta. È necessario un approccio migliore.

L'ottimizzazione a sciame di particelle fornisce un mezzo per eseguire ricerche in un ampio spazio di ricerca senza verificare ogni singolo valore in ogni dimensione. Nel problema dei droni, una dimensione del problema è l'alluminio, la seconda dimensione è la plastica e la terza dimensione è data dalle prestazioni risultanti del drone.

Nel prossimo paragrafo, determiniamo le strutture di dati richieste per rappresentare una particella, inclusi i dati sul problema.

Rappresentare lo stato: che aspetto hanno le particelle?

Poiché le particelle si muovono nello spazio di ricerca, è innanzitutto necessario definire il concetto di particella (Figura 7.12).

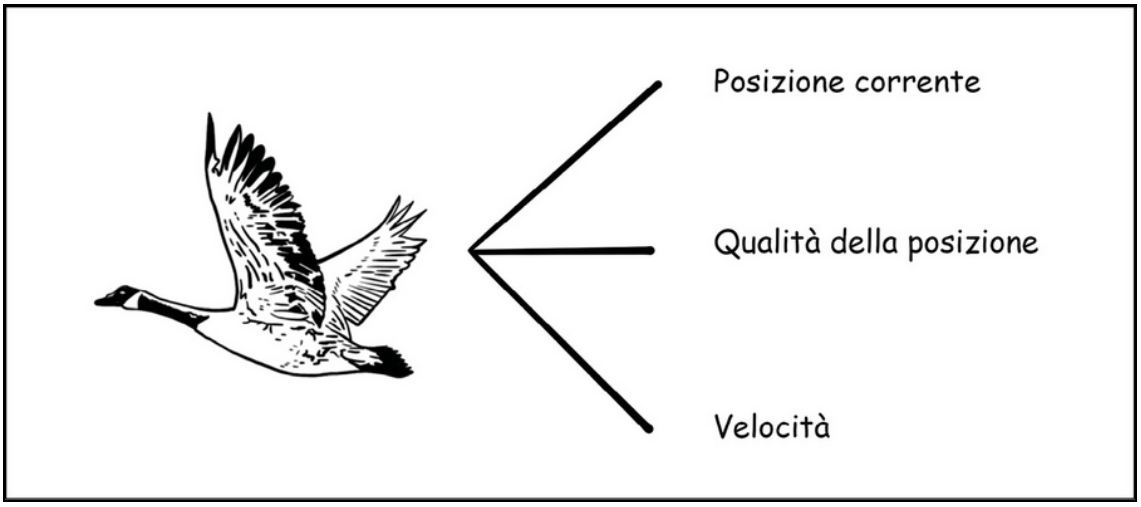


Figura 7.12 Proprietà di una particella.

Quanto segue rappresenta il concetto di particella.

- *Posizione*: la posizione della particella in tutte le dimensioni.
- *Qualità della posizione*: viene calcolata utilizzando una funzione di valutazione.
- *Velocità*: la velocità corrente del movimento della particella

Pseudocodice

Per soddisfare i tre attributi di una particella, fra cui posizione, qualità della posizione e velocità, sono richieste le seguenti proprietà nel costruttore della particella impiegata per le varie operazioni dell'algoritmo di ottimizzazione a sciame di particelle. Per il momento trascuriamo i componenti inerziale, cognitivo e sociale; ne parleremo nei prossimi paragrafi:

```
Particle(x, y, Inertia, cognitive_constant, social_constant):
  let particle.x equal to x
  let particle.y equal to y
  let particle.fitness equal to infinity
  let particle.velocity equal to 0
  let particle.best_x equal to x
  let particle.best_y equal to y
  let particle.best_fitness equal to infinity
  let particle.inertia equal to inertia
  let particle.cognitive_constant equal to cognitive_constant
  let particle.social_constant equal to social_constant
```

Ciclo di vita dell'ottimizzazione a sciame di particelle

L'approccio alla progettazione di un algoritmo di ottimizzazione a sciame di particelle si basa sullo spazio del problema che viene affrontato. Ogni problema ha un contesto unico e un dominio differente nei quali i dati vengono rappresentati. Anche le soluzioni a problemi differenti vengono misurate in modo differente. Analizziamo come progettare l'ottimizzazione di uno sciame di particelle per risolvere il problema della costruzione dei droni.

Il ciclo di vita generale di un algoritmo di ottimizzazione a sciame di particelle è il seguente (Figura 7.13).

1. *Inizializza la popolazione di particelle*. Determinare il numero di particelle da utilizzare, e inizializzare ciascuna particella in una posizione casuale dello spazio di ricerca.
2. *Calcola la qualità di ogni particella*. Data la posizione di ogni particella, determinare la qualità di quella particella in quella posizione.
3. *Aggiorna la posizione di ogni particella*. Aggiornare ripetutamente la posizione di tutte le particelle, utilizzando i principi dell'intelligenza di sciame. Le particelle esploreranno così lo spazio di ricerca e poi convergeranno verso buone soluzioni.
4. *Determina i criteri di arresto*. Determinare quando le particelle smettono di aggiornarsi e l'algoritmo si deve fermare.

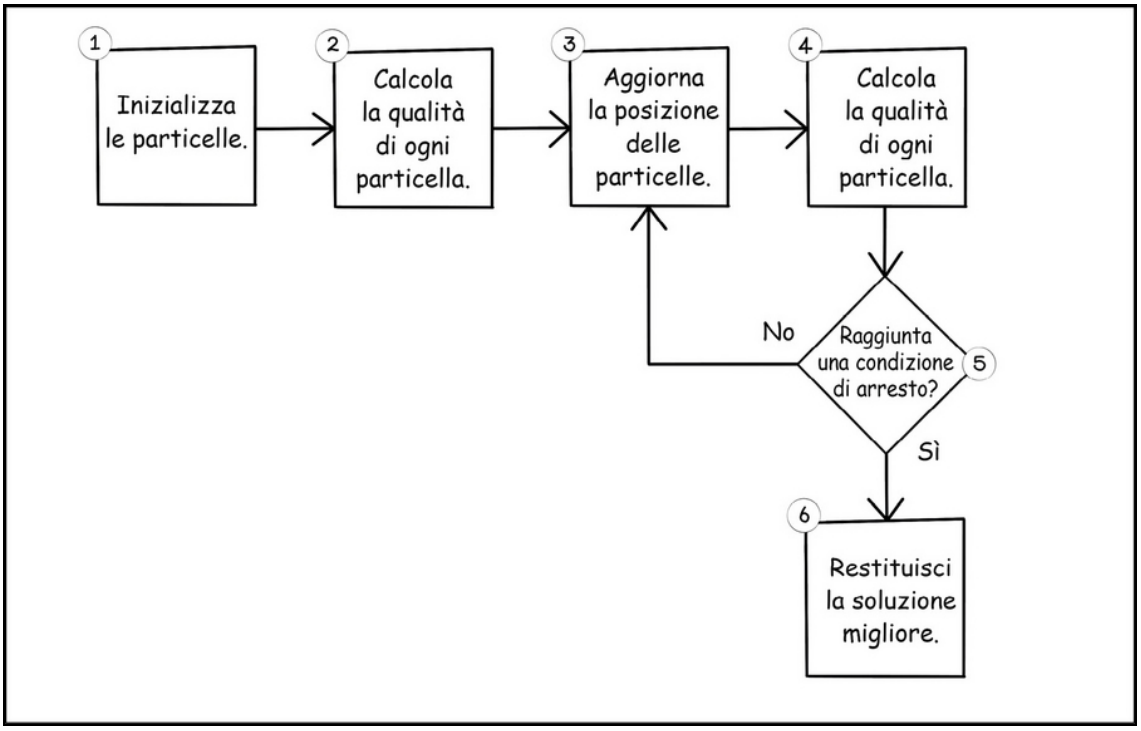


Figura 7.13 Il ciclo di vita di un algoritmo di ottimizzazione a sciame di particelle.

L'algoritmo di ottimizzazione a sciame di particelle è abbastanza semplice, ma i dettagli del Passaggio 3 sono particolarmente intricati. I prossimi paragrafi esaminano ogni passaggio isolatamente, svelando i dettagli che fanno funzionare l'algoritmo.

Inizializza la popolazione di particelle

L'algoritmo inizia creando un numero specifico di particelle, che rimarrà lo stesso per tutta la durata dell'algoritmo (Figura 7.14).

Ecco i tre fattori importanti nell'inizializzazione delle particelle (Figura 7.15).

- *Numero di particelle*: il numero di particelle influenza il calcolo. Più particelle esistono, più calcoli saranno necessari. Inoltre, più particelle probabilmente significheranno che la convergenza su una migliore soluzione globale richiederà più tempo, perché più

particelle verranno attratte dalle loro migliori soluzioni locali. Anche i vincoli del problema influenzano il numero di particelle. L'esplorazione di uno spazio di ricerca più ampio potrebbe richiedere più particelle. Potrebbero esserci anche 1.000 particelle o magari solo 4. Di solito, da 50 a 100 particelle producono buone soluzioni senza essere troppo costose dal punto di vista computazionale.

- *Posizione iniziale per ogni particella*: la posizione iniziale per ogni particella dovrebbe essere casuale in tutte le dimensioni. È importante che le particelle siano distribuite uniformemente nello spazio di ricerca. Se la maggior parte delle particelle si trova in una determinata regione dello spazio di ricerca, faticherà a trovare soluzioni al di fuori di tale area.

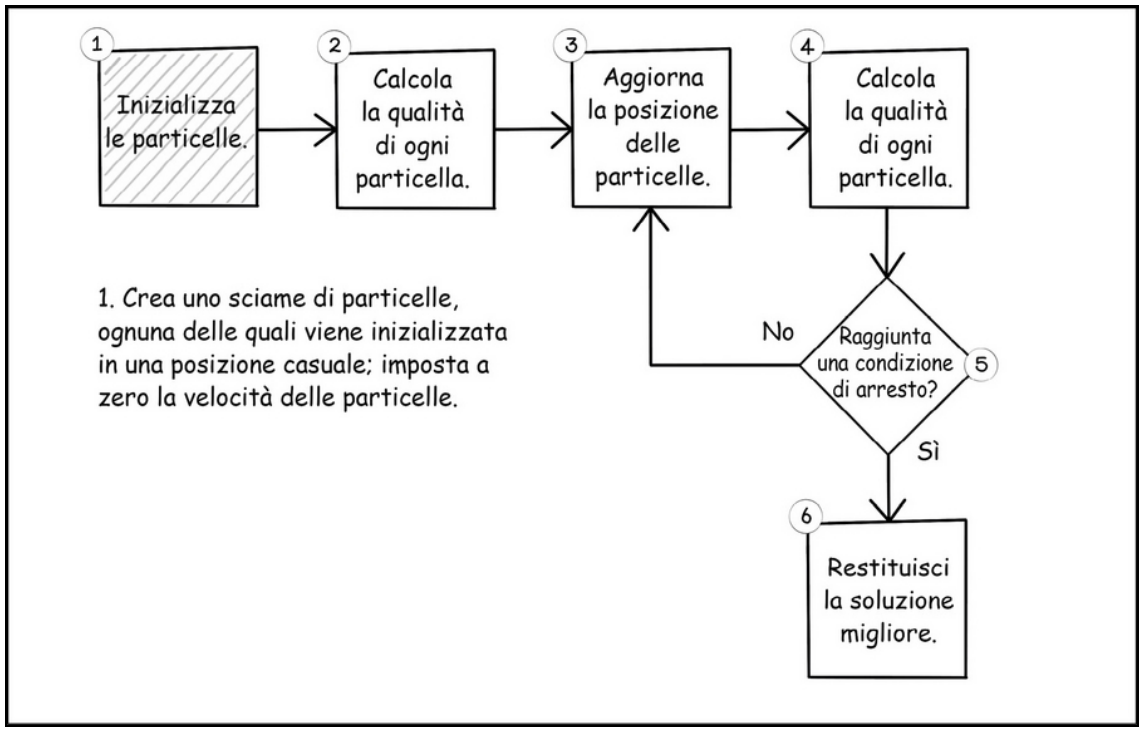


Figura 7.14 Disporre le particelle.

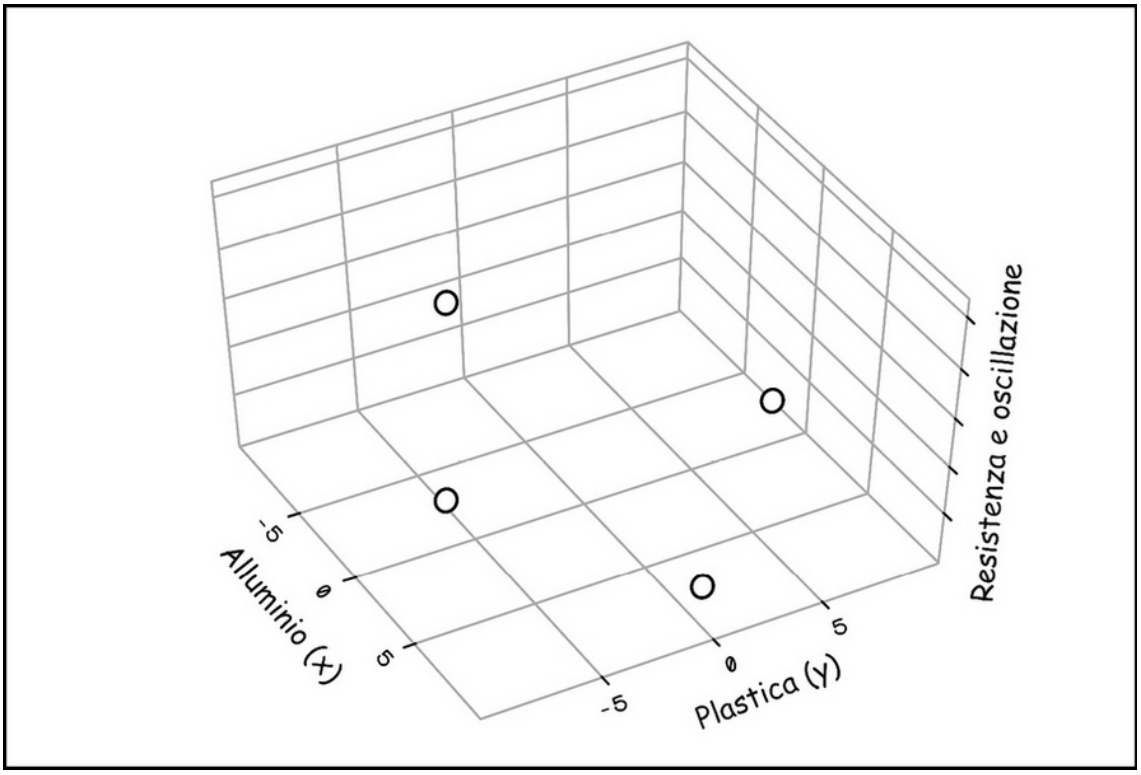


Figura 7.15 Una visualizzazione delle posizioni iniziali di quattro particelle in un piano 3D.

- *Velocità iniziale di ogni particella*: la velocità delle particelle è inizializzata a 0, perché le particelle non sono ancora state interessate. Una buona analogia con gli uccelli: prendono il volo da una posizione stazionaria.

La Tabella 7.2 descrive i dati incapsulati da ciascuna particella durante la fase di inizializzazione dell'algoritmo. Notate che la velocità è `0`; i valori di qualità corrente e migliore sono `0`, perché non sono ancora stati calcolati.

Tabella 7.2 Attributi di ogni particella nei dati.

Particella	Velocità	Alluminio corrente (x)	Plastica corrente (y)	Qualità corrente	Alluminio migliore (x)	Plastica migliore (y)	Qualità migliore

1	0	7	1	0	7	1	0
2	0	-1	9	0	-1	9	0
3	0	-10	1	0	-10	1	0
4	0	-2	-5	0	-2	-5	0

Pseudocodice

Il metodo per generare uno sciame consiste nel creare una lista vuota e aggiungervi le nuove particelle. I fattori chiave sono i seguenti.

- Garantire che il numero di particelle sia configurabile.
- Garantire che la generazione di numeri casuali avvenga in modo uniforme; i numeri sono distribuiti nello spazio di ricerca, entro vincoli. Questa implementazione dipende dalle caratteristiche del generatore di numeri casuali utilizzato.
- Garantire che i vincoli dello spazio di ricerca siano specificati: in questo caso, -10 e 10 sia per *x* sia per *y*.

```
generate_swarm(number_of_particles):
  let particles equal an empty list
  for particle in range(number_of_particles):
            append  Particle(random(-10,  10),  random(-10,  10),  INERTIA,
COGNITIVE_CONSTANT,
             SOCIAL_CONSTANT) to particles
  return particles
```

Calcola la qualità di ogni particella

Il passo successivo consiste nel calcolare la qualità di ogni particella nella sua posizione attuale. La qualità delle particelle viene calcolata ogni volta che l'intero sciame cambia posizione (Figura 7.16).

Nello scenario del drone, gli scienziati hanno fornito una funzione in cui il risultato è la quantità di resistenza e oscillazione dato un numero specifico di componenti di alluminio e plastica. Questa funzione viene utilizzata per calcolare la qualità nell'algoritmo di ottimizzazione a sciame di particelle di questo esempio (Figura 7.17).

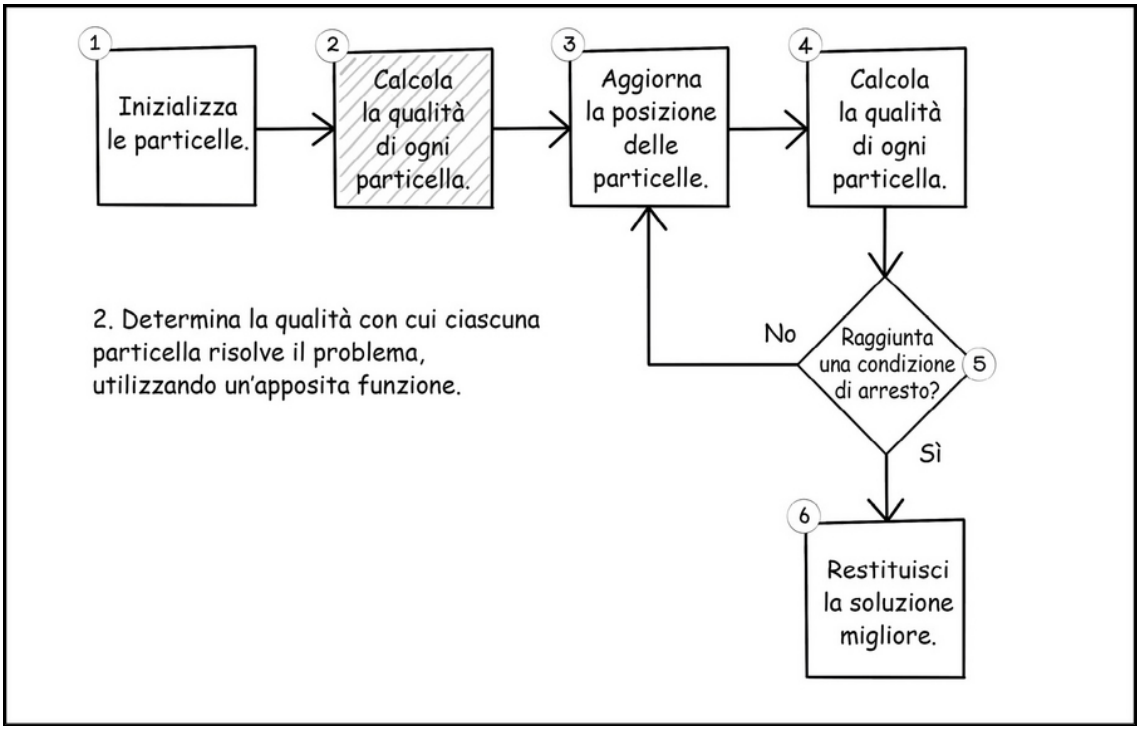


Figura 7.16 Calcolare la qualità delle particelle.

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Figura 7.17 La funzione di esempio per l'ottimizzazione dell'alluminio (x) e della plastica (y).

Se *x* è la quantità di alluminio e *y* è la quantità di plastica, è possibile eseguire i calcoli della Figura 7.18 per ciascuna particella per determinarne la qualità sostituendo a *x* e *y* le quantità di alluminio e plastica.

$$f(7,1) = (7 + 2(1) - 7)^2 + (2(7) + 1 - 5)^2 = 104$$
$$f(-1,9) = (-1 + 2(9) - 7)^2 + (2(-1) + 9 - 5)^2 = 104$$
$$f(-10,1) = (-10 + 2(1) - 7)^2 + (2(-10) + 1 - 5)^2 = 801$$
$$f(-2,-5) = (-2 + 2(-5) - 7)^2 + (2(-2) - 5 - 5)^2 = 557$$

Figura 7.18 Calcoli di qualità per ogni particella.

Ora la tabella delle particelle rappresenta la qualità calcolata per ciascuna particella (Tabella 7.3). È anche impostata come la migliore qualità per ogni particella, perché è l'unica qualità nota alla prima iterazione. Dopo la prima iterazione, la migliore qualità di ogni particella diverrà la migliore qualità calcolata in tutta la storia di ciascuna particella.

Tabella 7.3 Attributi dei dati per ogni particella.

Particella	Velocità	Alluminio corrente (x)	Plastica corrente (y)	Qualità corrente	Alluminio migliore (x)	Plastica migliore (y)	Qualità migliore
1	0	7	1	*296*	7	1	*296*
2	0	-1	9	*104*	-1	9	*104*
3	0	-10	1	*80*	-10	1	*80*
4	0	-2	-5	*365*	-2	-5	*365*

Esercizio: quale sarebbe la qualità per i seguenti input, data la seguente funzione di calcolo della qualità del drone?

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Particella	Velocità	Alluminio corrente (x)	Plastica corrente (y)	Qualità corrente	Alluminio migliore (x)	Plastica migliore (y)	Qualità migliore
1	0	5	-3	0	5	-3	0
2	0	-6	-1	0	-6	-1	0
3	0	7	3	0	7	3	0
4	0	-1	9	0	-1	9	0

Soluzione

$$f(5,-3) = (5 + 2(-3) - 7)^2 + (2(5) - 3 - 5)^2 = 68$$

$$f(-6,-1) = (-6 + 2(-1) - 7)^2 + (2(-6) - 1 - 5)^2 = 549$$

$$f(7,3) = (7 + 2(3) - 7)^2 + (2(7) + 3 - 5)^2 = 180$$

$$f(-1,9) = (-1 + 2(9) - 7)^2 + (2(-1) + 9 - 5)^2 = 104$$

Pseudocodice

La funzione di valutazione della qualità (`calculate_fitness`) rappresenta la funzione matematica impiegata nel codice. Qualsiasi libreria matematica conterrà le operazioni richieste (l'elevamento a potenza e la radice quadrata):

```
calculate_fitness(x, y):
  return power(x + 2 * y - 7, 2) + power(2 * x + y - 5, 2)
```

Anche la funzione per aggiornare la qualità di una particella (`update_fitness`) è banale, in quanto determina se la nuova qualità è migliore di una qualità calcolata in passato e se necessario salva il nuovo valore:

```
update_fitness(x, y):
  let particle.fitness equal the result of calculate_fitness(x, y)
  if particle.fitness is less than particle.best_fitness:
      let particle.best_fitness equal particle.fitness
      let particle.best_x equal x
      let particle.best_y equal y
```

La funzione per determinare la migliore particella dello sciame (`get_best`) itera tutte le particelle, aggiorna la loro qualità in base alle loro nuove posizioni e trova la particella che produce il valore più piccolo per la funzione di valutazione della qualità. In questo caso, stiamo minimizzando, quindi ci interessa il valore più piccolo:

```
get_best(swarm):
  let best_fitness equal infinity
  let best_particle equal nothing
  for particle in swarm:
      update fitness of particle
      if particle.fitness is less than best_fitness:
          let best_fitness equal particle.fitness
          let best_particle equal particle
  return best_particle
```

Aggiorna la posizione di ogni particella

La fase di aggiornamento dell'algoritmo è la più complessa, perché è qui che avviene la "magia". La fase di aggiornamento distilla le proprietà dell'intelligenza di sciame in natura in un modello matematico che consente di esplorare lo spazio di ricerca perfezionando le soluzioni valide (Figura 7.19).

Le particelle dello sciame aggiornano la loro posizione sulla base di un'abilità cognitiva e di fattori che dipendono dall'ambiente che le circonda, come l'inerzia e ciò che sta facendo lo sciame. Questi fattori influenzano la velocità e la direzione di spostamento di ciascuna particella. Il primo passo consiste nel capire come viene aggiornata la velocità, che determina anche la direzione di movimento della particella.

Le particelle dello sciame si spostano in punti differenti dello spazio di ricerca per trovare soluzioni sempre migliori. Ogni particella fa affidamento sulla propria memoria di una buona soluzione e sulle conoscenze della migliore soluzione acquisite dallo sciame. La Figura 7.20 illustra il movimento delle particelle dello sciame a mano a mano che le loro posizioni vengono aggiornate.

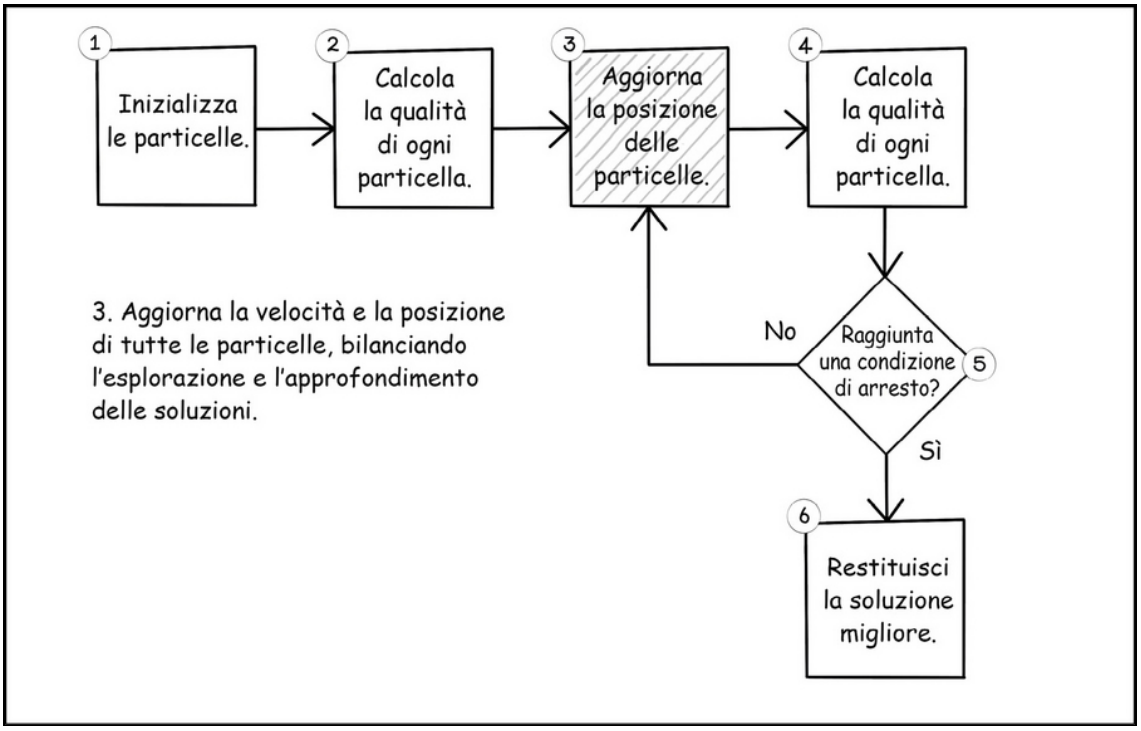


Figura 7.19 Aggiornare la posizione di ogni particella.

I componenti che contribuiscono all'aggiornamento della velocità

Sono tre i componenti utilizzati per calcolare la nuova velocità di ciascuna particella: inerziale, cognitiva e sociale. Ognuno di questi componenti influenza il movimento della particella. Li esamineremo isolatamente, prima di approfondire il modo in cui si combinano per aggiornare la velocità e, in definitiva, la posizione di ogni particella.

- *Inerziale*: rappresenta la resistenza al movimento o al cambiamento di direzione di una particella, cosa che ne influenza la velocità. Il componente inerziale è composto da due valori: l'entità dell'inerzia e la velocità attuale della particella. L'entità dell'inerzia è un valore compreso fra `0` e `1`.

  - Un valore più vicino a `0` si traduce in un'esplorazione più limitata, che richiede potenzialmente più iterazioni.

- Un valore più vicino a `1` si traduce in una maggiore esplorazione per le particelle, con meno iterazioni.

- *Cognitivo:* rappresenta la competenza cognitiva interna di una particella. L'abilità cognitiva riguarda le conoscenze di una particella, che le consentono di decidere una posizione migliore e che essa usa per influenzare il proprio movimento. La costante cognitiva è un numero maggiore di `0` e minore di `2`. Una costante cognitiva maggiore genera un maggiore approfondimento da parte delle particelle.

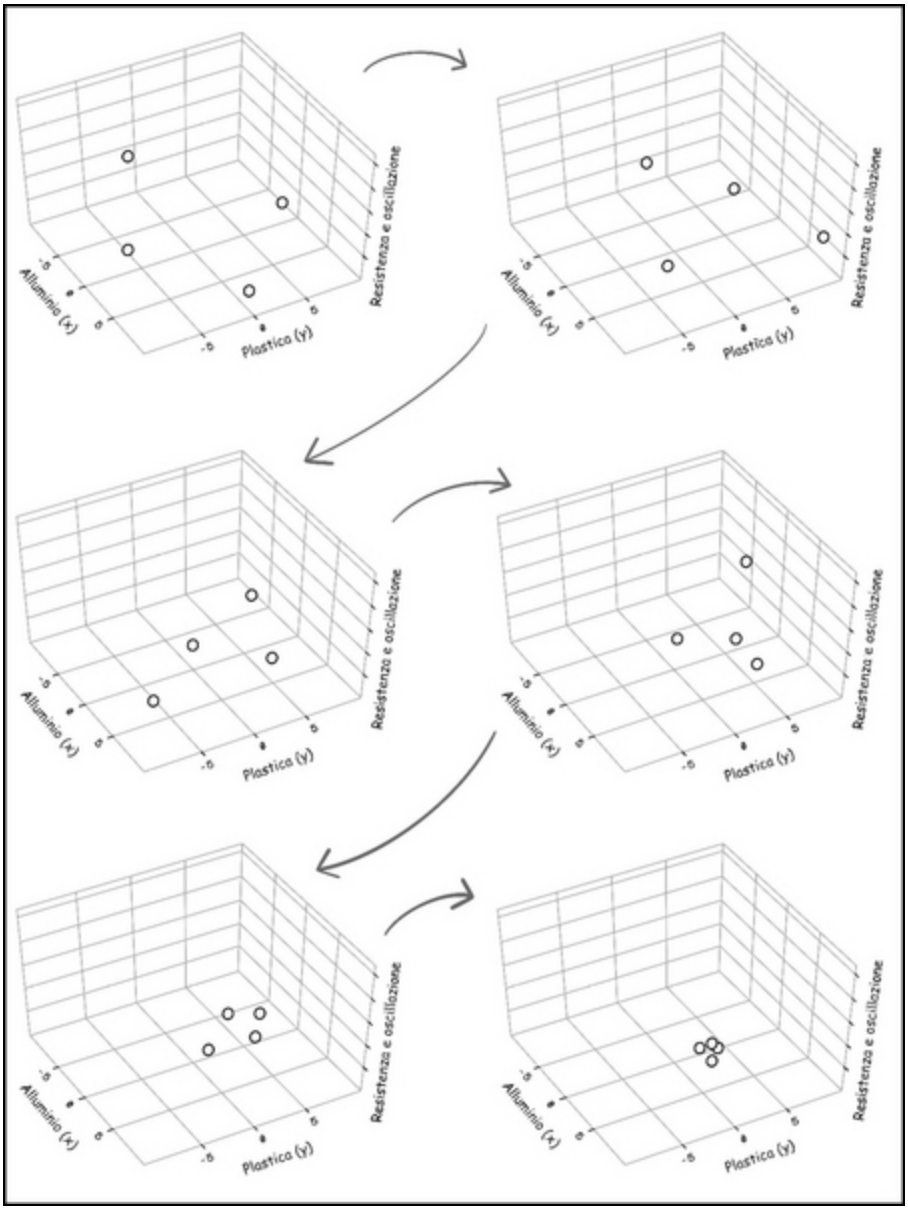
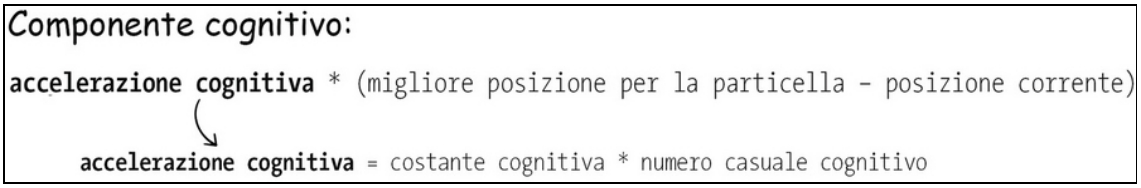
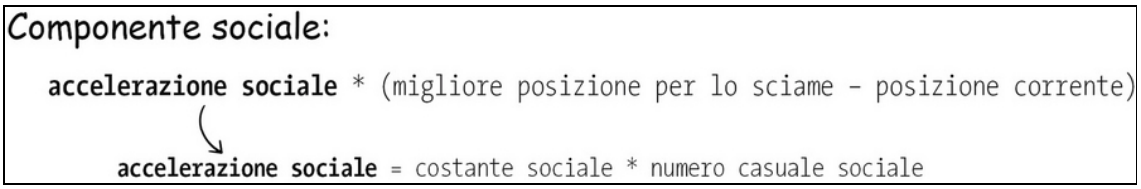


Figura 7.20 Il movimento delle particelle per cinque iterazioni.



- *Sociale:* rappresenta la capacità di una particella di interagire con lo sciame. Una particella conosce anche la posizione migliore per lo sciame e usa questa informazione per influenzarne il proprio movimento. L'accelerazione sociale è determinata utilizzando una costante e ridimensionandola con un numero casuale. La costante sociale rimane la stessa per tutta la durata dell'algoritmo, mentre il fattore casuale incoraggia la diversità. Maggiore è la costante sociale, maggiore sarà l'esplorazione, perché la particella favorisce maggiormente la sua componente sociale. La costante sociale è un numero compreso fra 0 e 2.



Aggiornamento della velocità

Ora che abbiamo introdotto i componenti inerziale, cognitivo e sociale, vediamo come possono essere combinati per determinare la nuova velocità delle particelle (Figura 7.21).

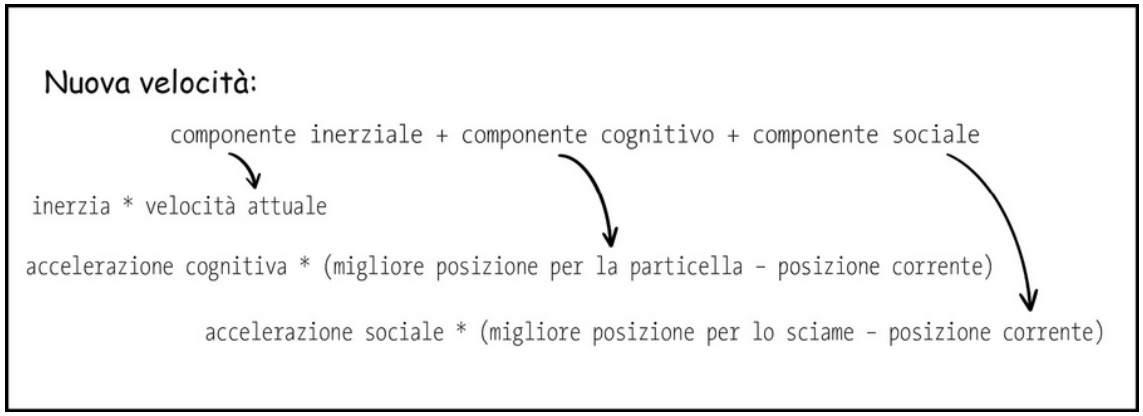


Figura 7.21 Formula per calcolare la velocità.

Osservando i calcoli, non è banale capire come i diversi componenti della funzione influenzino la velocità delle particelle. La Figura 7.22 ne chiarisce il significato pratico.

La Tabella 7.4 mostra gli attributi di ciascuna particella dopo i calcoli relativi alla qualità di ognuna di esse.

Tabella 7.4 Attributi dei dati per ogni particella.

Particella	Velocità	Alluminio corrente (x)	Plastica corrente (y)	Qualità corrente	Alluminio migliore (x)	Plastica migliore (y)	Qualità migliore
1	0	7	1	296	2	4	296
2	0	-1	9	104	-1	9	104
3	0	-10	1	80	-10	1	80
4	0	-2	-5	365	-2	-5	365

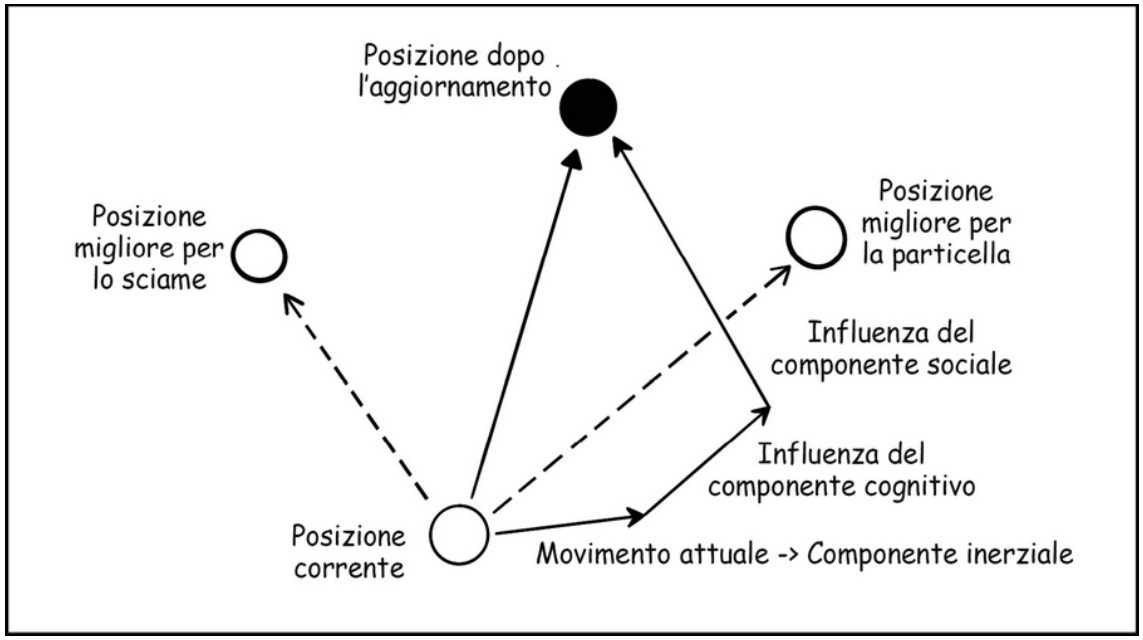


Figura 7.22 I fattori che influenzano gli aggiornamenti di velocità.

Ora ci immergeremo nei calcoli di aggiornamento della velocità per una particella, date le formule che abbiamo elaborato.

Ecco le configurazioni delle costanti che sono state impostate per questo scenario.

- La *costante inerziale* è impostata a `0,2`. Questa impostazione favorisce un'esplorazione più lenta.
- La *costante cognitiva* è impostata a `0,35`. Poiché questa costante è inferiore alla costante sociale, il componente sociale è favorito rispetto al componente cognitivo di una singola particella.
- La *costante sociale* è impostata a `0,45`. Poiché questa costante è maggiore della costante cognitiva, il componente sociale è favorito. Le particelle danno più peso ai migliori valori trovati dallo sciame.

La Figura 7.23 descrive i calcoli dei componenti inerziale, cognitivo e sociale per la formula di aggiornamento della velocità.

Dopo aver svolto questi calcoli per tutte le particelle, la velocità di ciascuna particella viene aggiornata, come rappresentato nella Tabella 7.5.

Tabella 7.5 Attributi dei dati per ogni particella.

Particella	Velocità	Alluminio corrente (x)	Plastica corrente (y)	Qualità corrente	Alluminio migliore (x)	Plastica migliore (y)	Qualità migliore
1	2,295	7	1	296	7	1	296
2	1,626	-1	9	104	-1	9	104
3	2,043	-10	1	80	-10	1	80
4	1,35	-2	-5	365	-2	-5	365

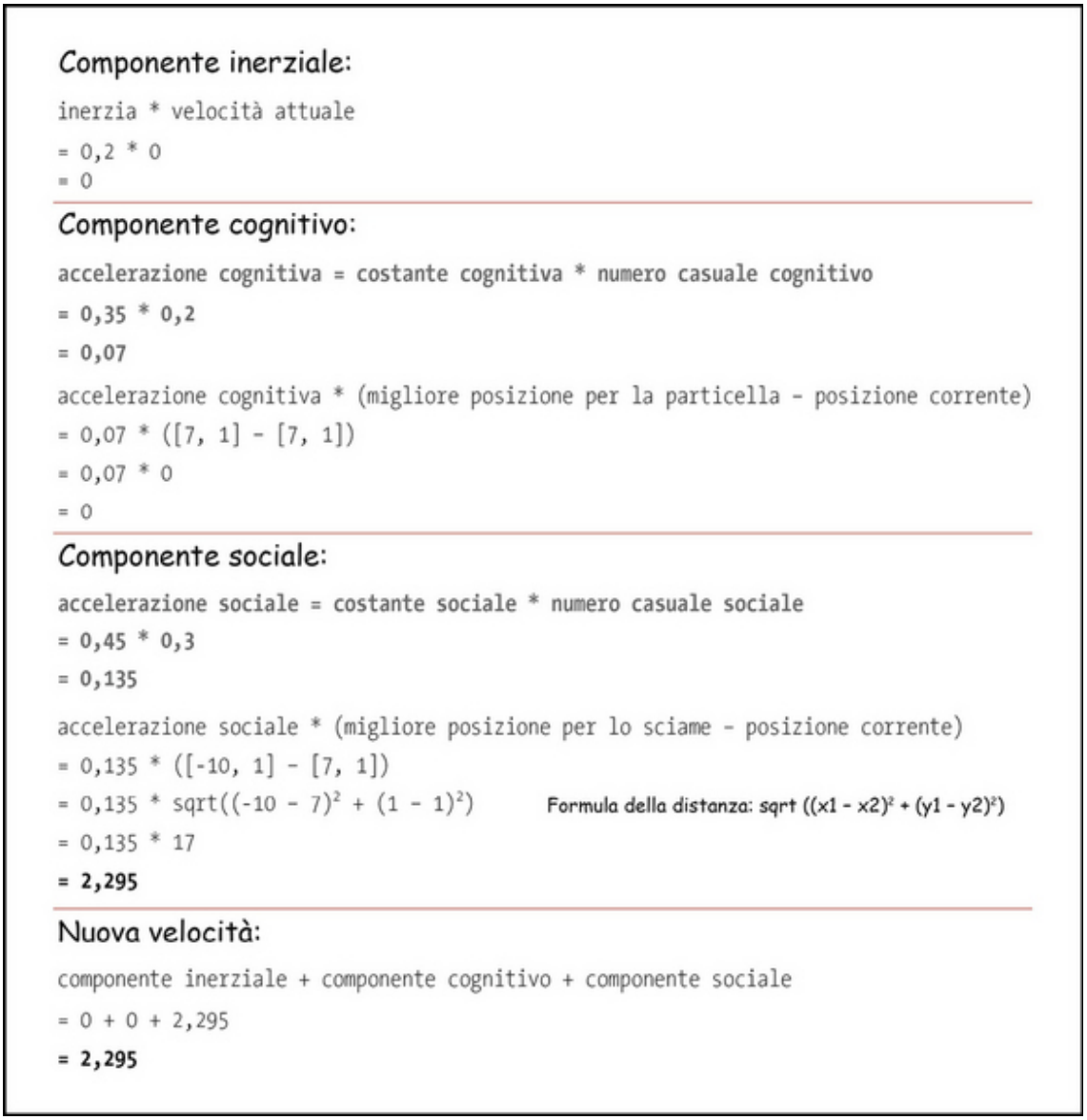


Figura 7.23 Calcolo della velocità delle particelle.

Aggiornamento della posizione

Ora che abbiamo compreso come aggiornare la velocità, possiamo aggiornare la posizione corrente di ciascuna particella, utilizzando la nuova velocità (Figura 7.24).

Aggiungendo la posizione corrente e la nuova velocità, possiamo determinare la nuova posizione di ciascuna particella e aggiornare la tabella degli attributi delle particelle con le nuove velocità. Poi si calcola nuovamente la qualità di ogni particella, data la sua nuova posizione, e si registra la sua migliore posizione (Tabella 7.6).

Tabella 7.6 Attributi dei dati per ogni particella.

Particella	Velocità	Alluminio corrente (x)	Plastica corrente (y)	Qualità corrente	Alluminio migliore (x)	Plastica migliore (y)	Qualità migliore
1	2,295	9,925	3,325	721,286	7	1	296
2	1,626	0,626	10	73,538	0,626	10	73,538
3	2,043	7,043	1,043	302,214	-10	1	80
4	1,35	-0,65	-3,65	179,105	-0,65	-3,65	179,105

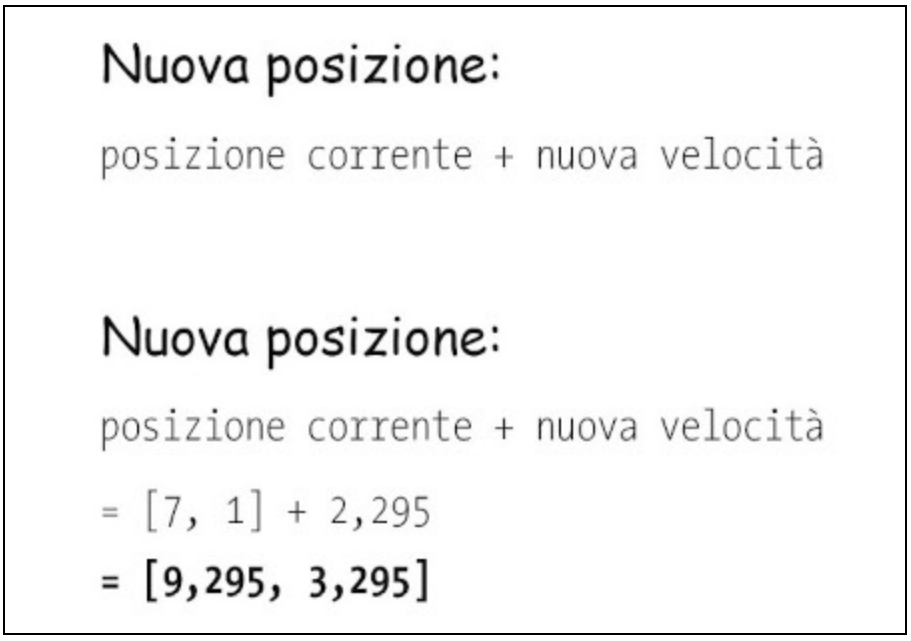


Figura 7.24 Calcolo della nuova posizione di una particella.

Calcolare la velocità iniziale per ciascuna particella nella prima iterazione è abbastanza semplice, perché non vi è una posizione migliore precedente per ciascuna particella, solo una posizione migliore per lo sciame, che interessa solo il componente sociale.

Esaminiamo come sarà il calcolo dell'aggiornamento della velocità con le nuove informazioni per la migliore posizione di ogni particella e la nuova migliore posizione per lo sciame. La Figura 7.25 descrive il calcolo per la sola particella 1.

In questo scenario, il componente cognitivo e sociale svolgono entrambi un ruolo nell'aggiornamento della velocità, mentre lo scenario descritto nella Figura 7.23 è influenzato dal solo componente sociale, essendo la prima iterazione.

Le particelle si spostano in posizioni differenti nelle diverse iterazioni. La Figura 7.26 mostra il movimento delle particelle e la loro convergenza verso una soluzione.

Nell'ultimo passaggio della Figura 7.26, tutte le particelle sono confluite in una determinata regione dello spazio di ricerca. Come soluzione finale verrà utilizzata la soluzione migliore trovata dallo sciame. Nei problemi di ottimizzazione del mondo reale, non è mai possibile visualizzare l'intero spazio di ricerca (il che, fra l'altro, renderebbe superfluo l'uso di algoritmi di ottimizzazione). Ma la funzione che abbiamo usato per l'esempio del drone è nota come funzione Booth. Mappandola sul piano cartesiano 3D, possiamo vedere che le particelle convergono effettivamente sul punto minimo nello spazio di ricerca (Figura 7.27).

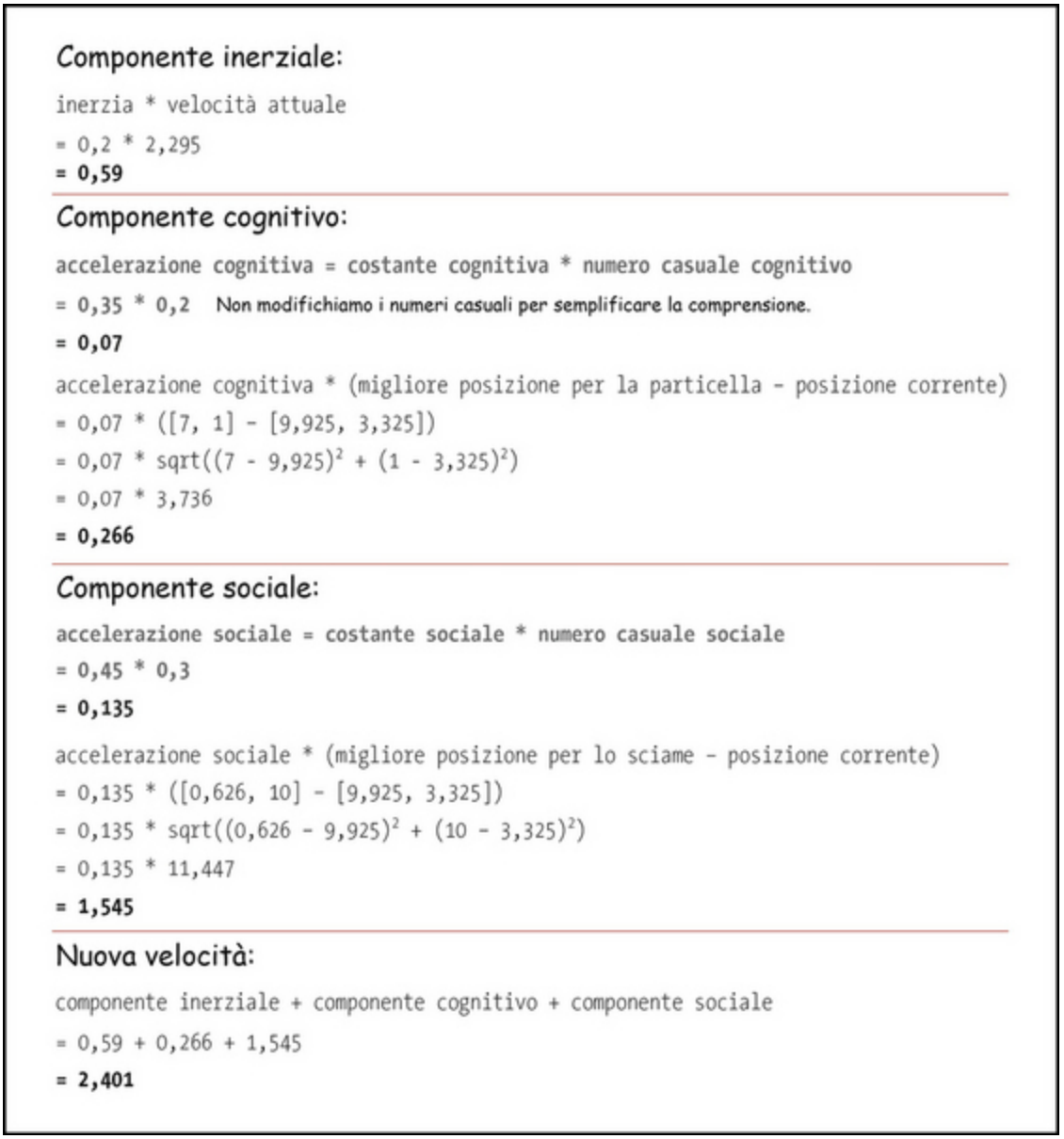


Figura 7.25 Calcolo della velocità delle particelle.

Dopo aver utilizzato l'algoritmo di ottimizzazione a sciame di particelle per l'esempio del drone, scopriamo che il rapporto ottimale fra alluminio e plastica per ridurre al minimo la resistenza e l'oscillazione è di 1:3, ovvero 1 parte di alluminio e 3 parti di plastica. Inserendo questi valori nella funzione di valutazione della qualità, il risultato è 0, che è il valore minimo per la funzione.

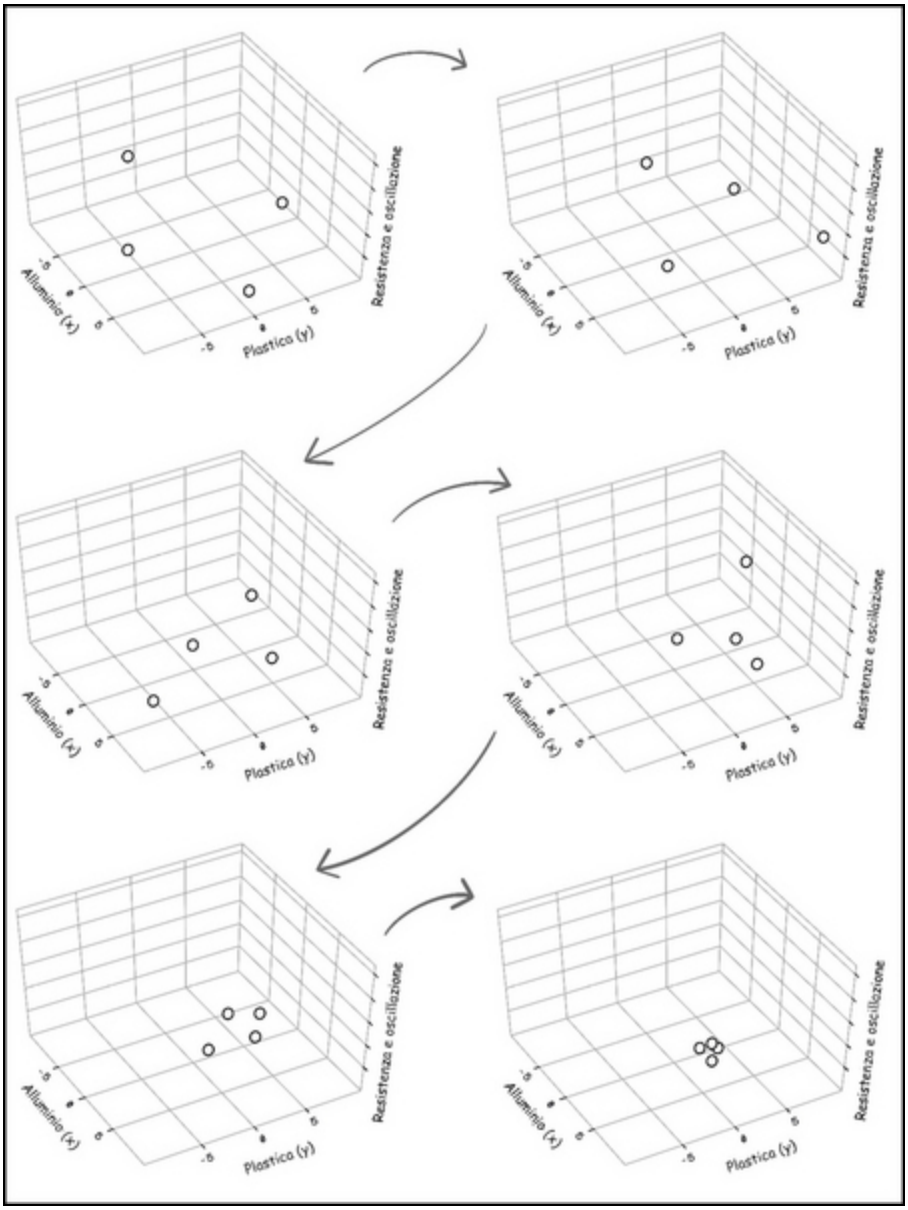


Figura 7.26 Il movimento delle particelle nello spazio di ricerca.

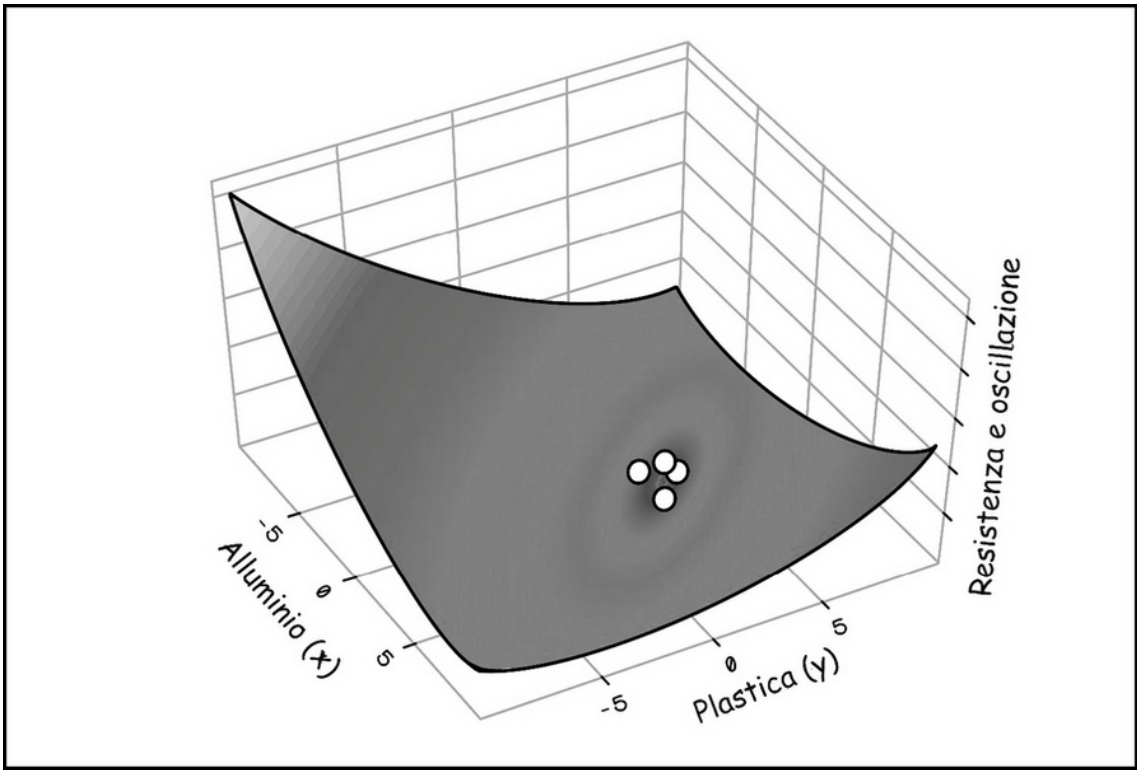


Figura 7.27 Visualizzazione della convergenza delle particelle e di una superficie nota.

Pseudocodice

La fase di aggiornamento può sembrare assai complessa, ma se i componenti vengono suddivisi in semplici funzioni mirate, il codice diventa più semplice e facile da scrivere, utilizzare e comprendere. Le prime funzioni sono la funzione di calcolo dell'inerzia, la funzione di accelerazione cognitiva e la funzione di accelerazione sociale. Abbiamo anche bisogno di una funzione per misurare la distanza fra due punti, che è la radice quadrata della somma fra il quadrato della differenza nei valori *x* e il quadrato della differenza nei valori *y*:

```
calculate_inertia(inertia_constant, velocity):
  return inertia_constant * current velocity
calculate_cognitive_acceleration(cognitive_constant):
  return cognitive_constant * random number between 0 and 1
calculate_social_acceleration(social_constant):
  return social_constant * random number between 0 and 1
calculate_distance(best_x, best_y, current_x, current_y):
    return square root(power(best_x - current_x), 2) + power(best_y -
current_y), 2))
```

Il componente cognitivo viene calcolato trovando l'accelerazione cognitiva, con la funzione che abbiamo definito in un paragrafo precedente, e la distanza fra la posizione migliore della particella e la sua posizione attuale:

```
calculate_cognitive(cognitive_constant,
      particle_best_x, particle_best_y,
      particle_current_x, particle_current_y):
  let acceleration equal cognitive_acceleration(cognitive_constant)
   let distance equal calculate distance(particle_best_x, particle_best_y,
particle_current_x, particle_current_y)
  return acceleration * distance
```

Il componente sociale viene calcolato trovando l'accelerazione sociale, con la funzione che abbiamo definito in precedenza, e la distanza fra la migliore posizione per lo sciame e la posizione attuale della particella:

```
calculate_social(social_constant,
      swarm_best_x, swarm_best_y,
      particle_current_x, particle_current_y):
  let acceleration equal social_acceleration(social_constant)
  let distance equal calculate_distance(swarm_best_x, swarm_best_y,
         particle_current_x, particle_current_y)
  return acceleration * distance
```

La funzione di aggiornamento incorpora tutto ciò che abbiamo definito, per eseguire l'effettivo aggiornamento della velocità e della posizione di una particella. La velocità viene calcolata utilizzando i componenti inerziale, cognitivo e sociale. La posizione viene calcolata sommando la nuova velocità alla posizione corrente della particella:

```
update_particle(cognitive_constant, social_constant, particle_velocity,
      particle_best_x, particle_best_y,
      swarm_best_x, swarm_best_y,
      particle_current_x, particle_current_y)
  let inertia equal calculate_inertia(inertia_constant, particle_constant)
  let cognitive equal calculate_cognitive(cognitive_constant,
          particle_best_x, particle_best_y,
          particle_current_x, particle_current_y)
  let social equal calculate social(social_constant,
          swarm_best_x, swarm_best_y,
          particle_current_x,particle_current_y)
  let particle.velocity equal inertia + cognitive + social
  let particle.x equal particle.x + velocity
  let particle.y equal particle.y + velocity
```

Esercizio: calcolare la nuova velocità e posizione per la particella 1 date le seguenti informazioni sulle particelle

- L'inerzia è `0,1`.
- La costante cognitiva è `0,5` e il numero casuale cognitivo è `0,2`.
- La costante sociale è `0,5` e il numero sociale cognitivo è `0,5`.

Particella	Velocità	Alluminio corrente (x)	Plastica corrente (y)	Qualità corrente	Alluminio migliore (x)	Plastica migliore (y)	Qualità migliore
1	3	4	8	721,286	7	1	296
2	4	3	3	73,538	0,626	10	73,538
3	1	6	2	302,214	-10	1	80
4	2	2	5	179,105	-0,65	-3,65	179,105

Soluzione

Componente inerziale:

inerzia * velocità attuale

= 0,1 * 3

= 0,3

Componente cognitivo:

accelerazione cognitiva = costante cognitiva * numero casuale cognitivo

= 0,5 * 0,2

= 0,1

accelerazione cognitiva * (migliore posizione per la particella - posizione corrente)

= 0,1 * ([7, 1] - [4, 8])

= 0,1 * sqrt((7 - 4)² + (1 - 8)²)

= 0,1 * 7,616

= 0,7616

Componente sociale:

accelerazione sociale = costante sociale * numero casuale sociale

= 0,5 * 0,5

= 0,25

accelerazione sociale * (migliore posizione per lo sciame - posizione corrente)

= 0,25 * ([0,626, 10] - [4, 8])

= 0,25 * sqrt((0,626 - 4)² + (10 - 8)²)

= 0,25 * 3,922

= 0,981

Nuova velocità:

componente inerziale + componente cognitivo + componente sociale

= 0,3 + 0,7616 + 0,981

= 2,0426

Determina i criteri di arresto

Le particelle dello sciame non possono continuare ad aggiornarsi e cercare all'infinito. È necessario determinare un criterio di arresto per consentire all'algoritmo di funzionare per un numero ragionevole di iterazioni fino a trovare una soluzione adeguata (Figura 7.28).

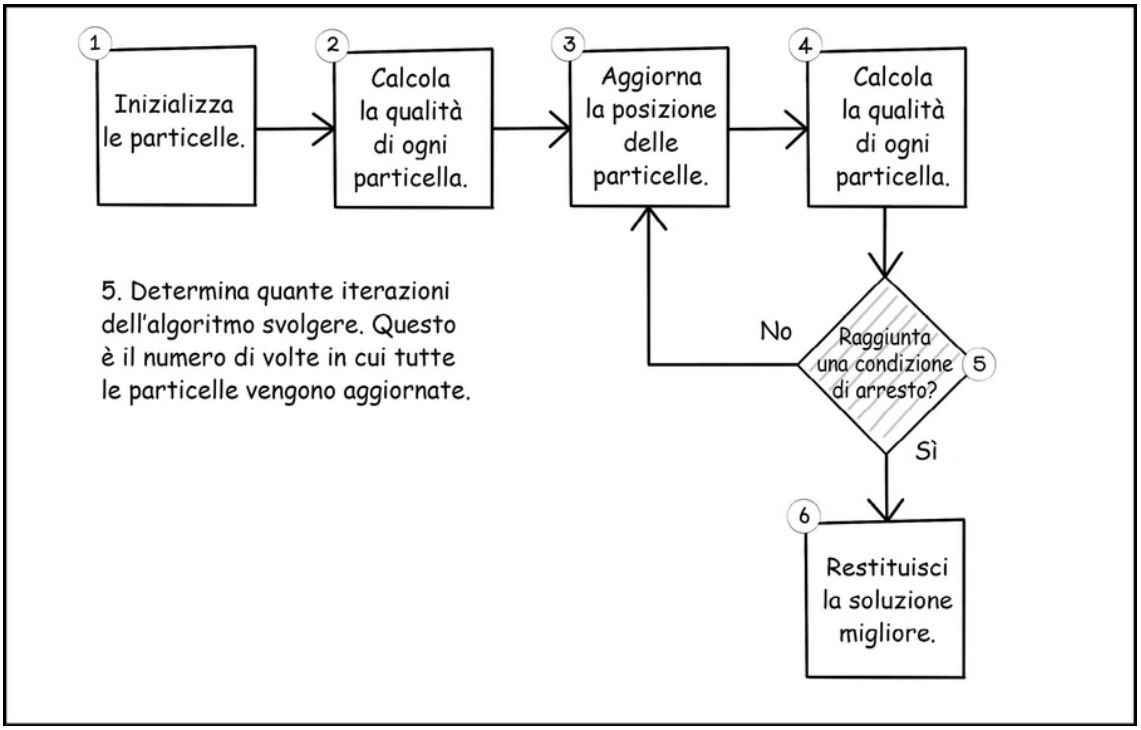


Figura 7.28 L'algoritmo ha raggiunto una condizione di arresto?

Il numero di iterazioni influenza diversi aspetti della ricerca di soluzioni, fra cui i seguenti.

- *Esplorazione*: le particelle richiedono tempo per esplorare lo spazio di ricerca e trovare aree con soluzioni migliori. L'esplorazione è influenzata anche dalle costanti definite nella funzione di aggiornamento della velocità.
- *Approfondimento*: le particelle dovrebbero convergere su una buona soluzione dopo un'esplorazione ragionevole.

Una strategia per fermare l'algoritmo consiste nell'esaminare la soluzione migliore nello sciame e determinare se sta stagnando. La stagnazione si verifica quando il valore della soluzione migliore non cambia o cambia solo di una quantità minima. L'esecuzione di ulteriori iterazioni non aiuterebbe a trovare soluzioni migliori. Quando la soluzione migliore ristagna, i parametri nella funzione di aggiornamento possono essere regolati per favorire una maggiore esplorazione. Se desiderate una maggiore esplorazione, questa regolazione di solito produrrà più iterazioni. La stagnazione potrebbe però significare due cose: che è stata trovata una buona soluzione o che lo sciame è rimasto bloccato in una migliore soluzione locale. Se all'inizio l'esplorazione è stata sufficiente e lo sciame ristagna gradualmente, lo sciame sta convergendo su una buona soluzione (Figura 7.29).

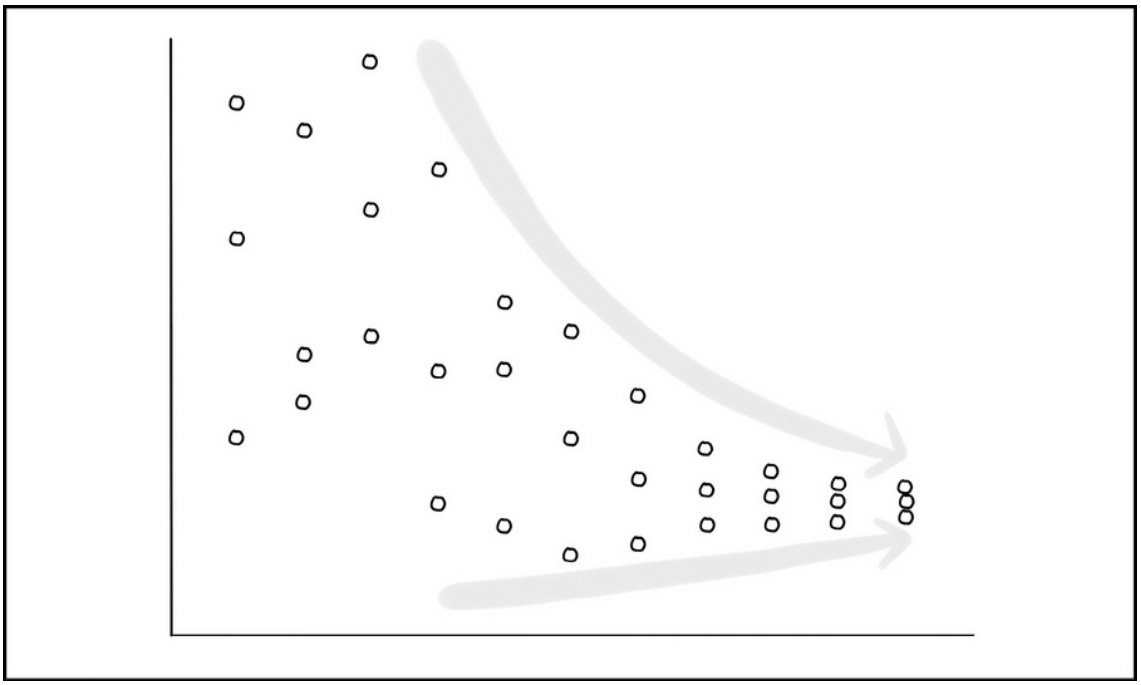


Figura 7.29 Esplorazione e approfondimento convergenti.

Casi d’uso per algoritmi di ottimizzazione a sciame di particelle

Gli algoritmi di ottimizzazione a sciame di particelle sono interessanti perché simulano un fenomeno naturale, il che li rende più facili da comprendere, ma possono essere applicati a vari problemi, a diversi livelli di astrazione. Questo capitolo ha esaminato un problema di ottimizzazione per la produzione di droni, ma gli algoritmi di ottimizzazione a sciame di particelle possono essere utilizzati anche insieme ad altri algoritmi, come le reti neurali artificiali, svolgendo un ruolo piccolo ma fondamentale nella ricerca di buone soluzioni.

Un’interessante applicazione di un algoritmo di ottimizzazione a sciame di particelle è la stimolazione cerebrale profonda. Il concetto prevede l’installazione di sonde con elettrodi nel cervello, per stimolarlo in modo da contrastare condizioni come il morbo di Parkinson. Ogni sonda contiene elettrodi che possono essere configurati in diverse direzioni, per trattare correttamente la condizione per paziente. I ricercatori dell’Università del Minnesota hanno sviluppato un algoritmo di ottimizzazione a sciame di particelle per ottimizzare la direzione di ciascun elettrodo in modo da massimizzare la regione di interesse (ROI), ridurre al minimo la regione di evitamento (ROA) e ridurre al minimo il consumo di energia. Poiché le particelle sono efficaci nelle ricerche in questi spazi dei problemi multidimensionali, l’algoritmo di ottimizzazione a sciame di particelle è risultato efficace nel trovare le configurazioni ottimali degli elettrodi posti sulle sonde (Figura 7.30).

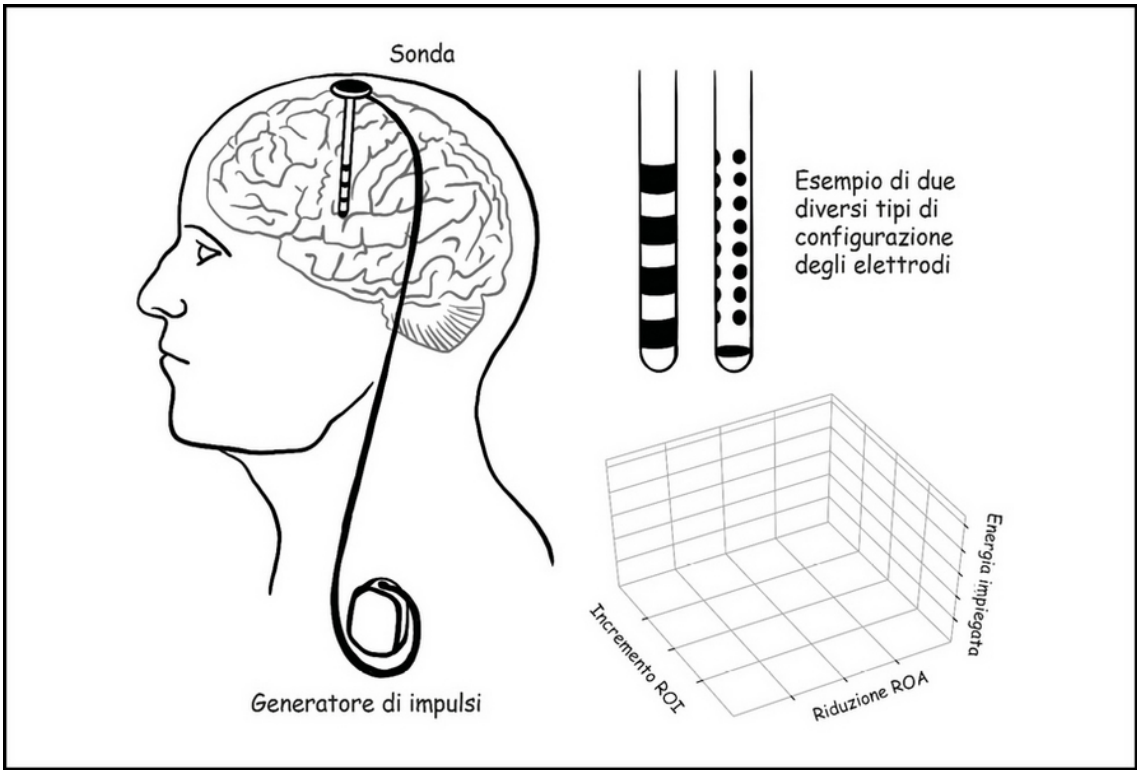


Figura 7.30 Esempio di fattori coinvolti nelle sonde per la stimolazione cerebrale profonda.

Ecco alcune altre applicazioni del mondo reale degli algoritmi di ottimizzazione a sciame di particelle.

- *Ottimizzazione dei pesi in una rete neurale artificiale*: le reti neurali artificiali sono modellate su un'idea del funzionamento del cervello umano. I neuroni trasmettono segnali ad altri neuroni e ciascun neurone regola il segnale prima di trasmetterlo. Una rete neurale artificiale utilizza dei pesi per regolare ogni segnale. La potenza della rete consiste nel trovare il giusto equilibrio di pesi per trovare modelli nelle relazioni presenti nei dati. La regolazione dei pesi è computazionalmente costosa, poiché lo spazio di ricerca è enorme. Immaginate di dover risolvere a forza bruta ogni possibile combinazione di numeri decimali per dieci pesi. Un processo che richiederebbe anni.

Non preoccupatevi se questo concetto vi sembra confuso. Parleremo delle reti neurali artificiali nel Capitolo 9. L'ottimizzazione a sciame di particelle può essere utilizzata per regolare più velocemente i pesi delle reti neurali, perché cerca valori ottimali nello spazio di ricerca senza provare in modo esaustivo ognuno di essi.

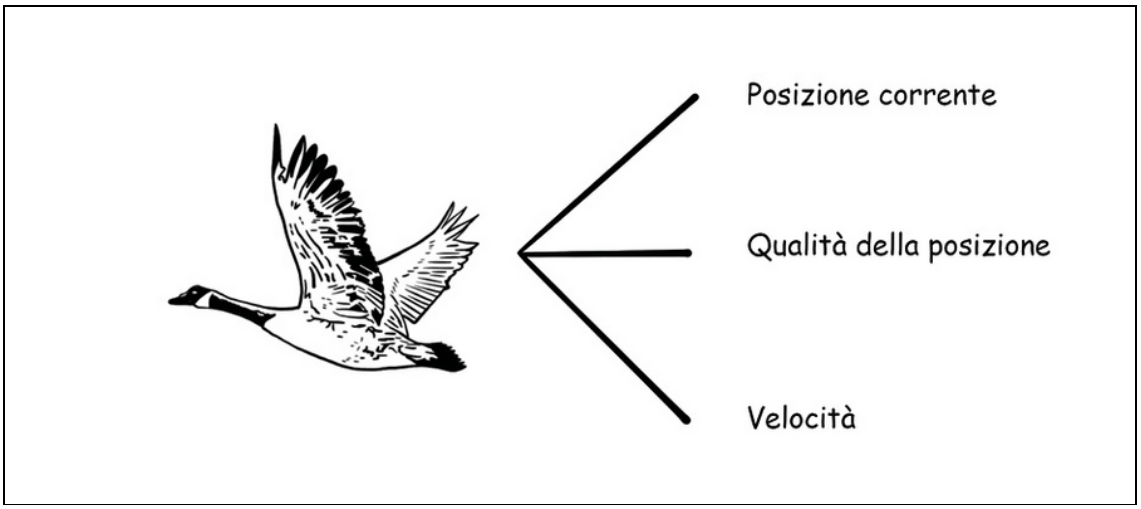
- *Tracciamento del movimento nei video*: il tracciamento del movimento delle persone è un compito impegnativo nella visione artificiale. L'obiettivo è quello di identificare la posizione delle persone e implicare un movimento utilizzando solo le informazioni delle immagini contenute nel video. Le persone si muovono in modo differente, anche se le articolazioni si muovono tutte in modo piuttosto simile. Poiché le immagini contengono molti dettagli, lo spazio di ricerca diventa ampio, con molte dimensioni per riuscire a prevedere il movimento di una persona. L'ottimizzazione a sciame di particelle è particolarmente efficace negli spazi di ricerca ad alta dimensionalità e può essere utilizzata per migliorare le prestazioni del tracciamento e della previsione del movimento.
- *Miglioramento del parlato nell'audio*: le registrazioni audio sono spesso di scarsa qualità. C'è sempre del rumore di fondo che può interferire con ciò che qualcuno sta dicendo. Una soluzione consiste nel rimuovere il rumore dalle clip audio. Una tecnica utilizzata a questo scopo consiste nel filtrare la clip audio con del rumore e confrontare i suoni simili per rimuovere il rumore dalla clip. Questa soluzione è complessa, poiché la riduzione di alcune frequenze può essere efficace per alcune parti della clip audio, ma può deteriorare altre parti. Per una buona rimozione del rumore è necessario eseguire una ricerca e trovare una buona corrispondenza. I metodi tradizionali sono lenti, poiché lo spazio

di ricerca è esteso. L'ottimizzazione a sciame di particelle si comporta molto bene in ampi spazi di ricerca e può essere utilizzata per velocizzare il processo di rimozione del rumore.

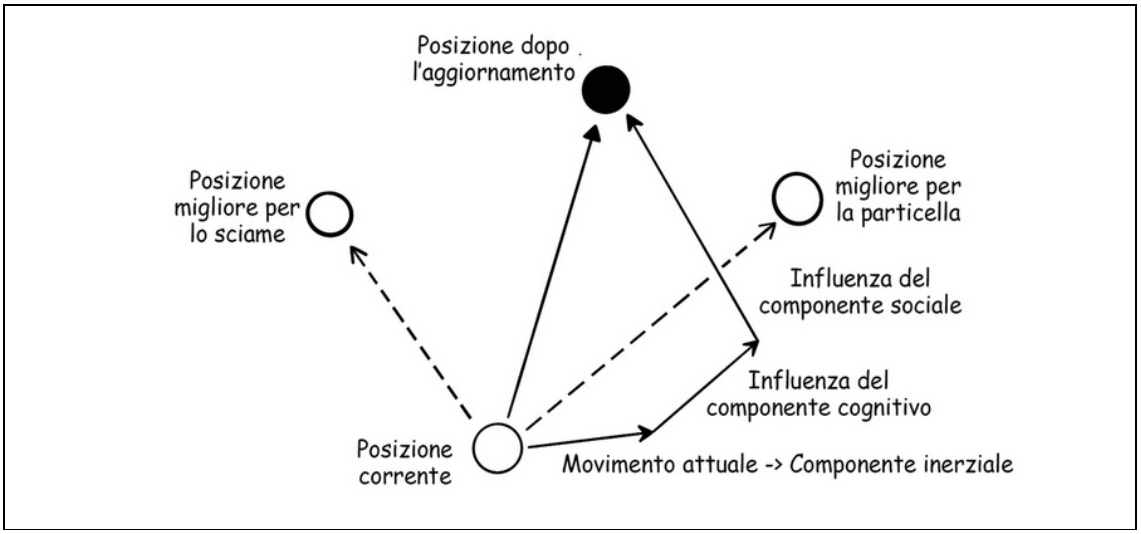
Riepilogo

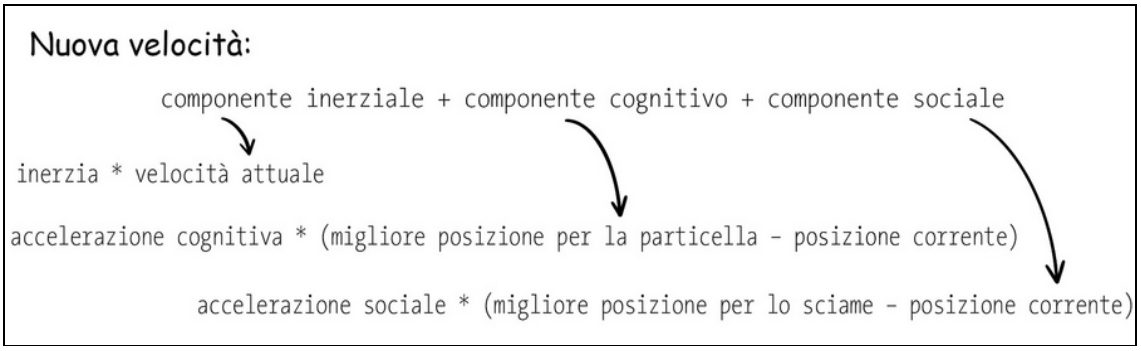
L'ottimizzazione a sciame di particelle trova buone soluzioni quando gli spazi di ricerca sono estesi.

- Le particelle utilizzano la migliore posizione che hanno trovato e la migliore posizione per lo sciame per procedere all'interno dello spazio di ricerca.

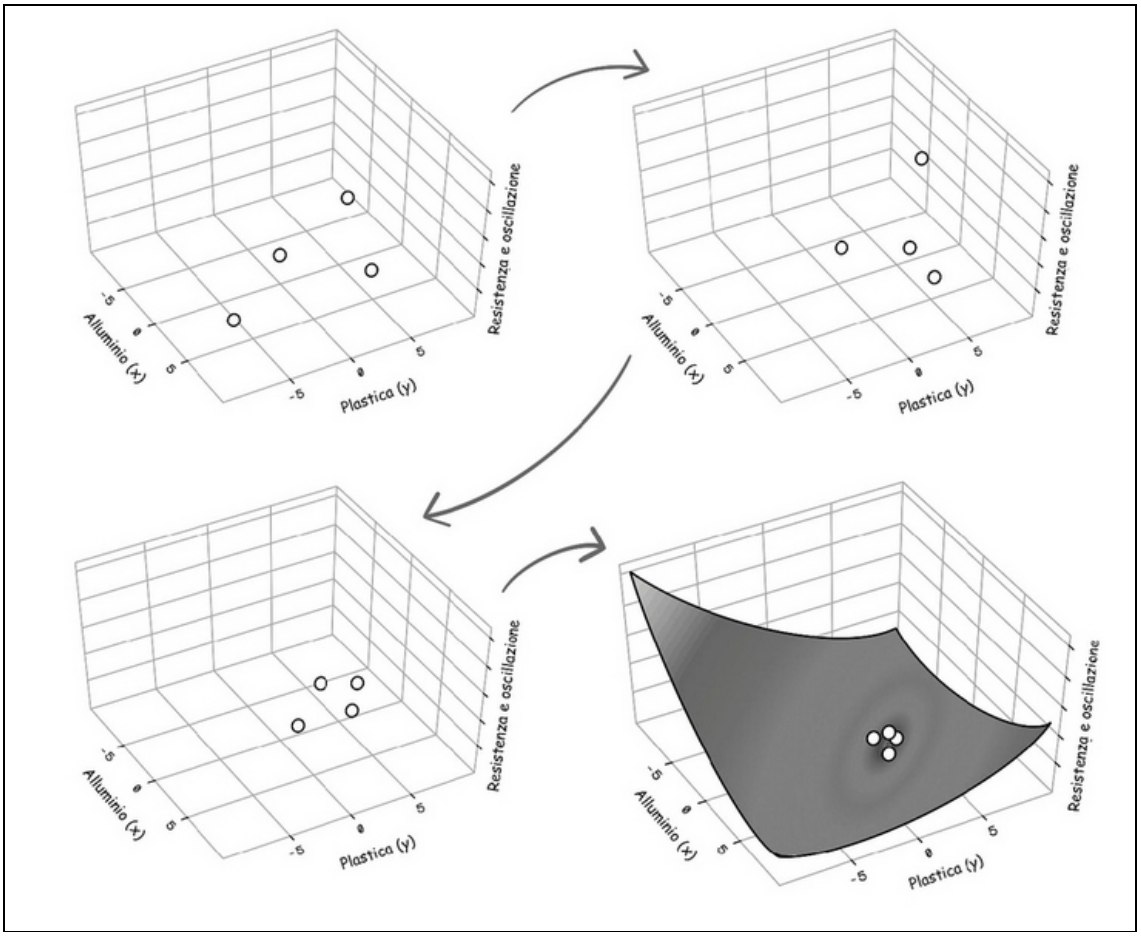


- La regolazione della velocità delle particelle è il passo critico dell'algoritmo di ottimizzazione a sciame di particelle, utilizzando tre componenti inerziale, cognitivo e sociale.





- Le particelle si muovono entro lo spazio di ricerca alla ricerca di buone soluzioni, e idealmente convergono verso la migliore soluzione globale.


Alluminio (x)
Plastica (y)
Resistenza e oscillazione
-5
0
5

Machine learning

Che cos'è il machine learning?

Il machine learning può sembrare un concetto complesso sia da apprendere sia da impiegare, ma con la giusta definizione e comprensione del processo e degli algoritmi, può essere un argomento molto interessante e perfino divertente.

State cercando un nuovo appartamento in affitto. Parlate con amici e familiari e fate alcune ricerche online sugli appartamenti in città. Notate che gli appartamenti nelle diverse zone hanno prezzi diversi. Ecco alcune delle osservazioni che traete dalle vostre ricerche.

- Un monolocale in centro (vicino al lavoro) costa $ 5.000 al mese.
- Un bilocale in centro costa $ 7.000 al mese.
- Un monolocale in centro con garage costa $ 6.000 al mese.
- Un monolocale in periferia e lontano dal lavoro, costa $ 3.000 al mese.
- Un bilocale in periferia costa $ 4.500 al mese.
- Un monolocale in periferia con garage costa $ 3.800 al mese.

Notate alcuni schemi. Gli appartamenti in centro sono più costosi e di solito costano fra i 5.000 e i 7.000 dollari al mese. Gli appartamenti fuori città sono più economici. Una stanza in più aggiunge fra i 1.500 e i 2.000 dollari al mese e la disponibilità di un garage aggiunge dagli 800 ai 1.000 dollari al mese (Figura 8.1).

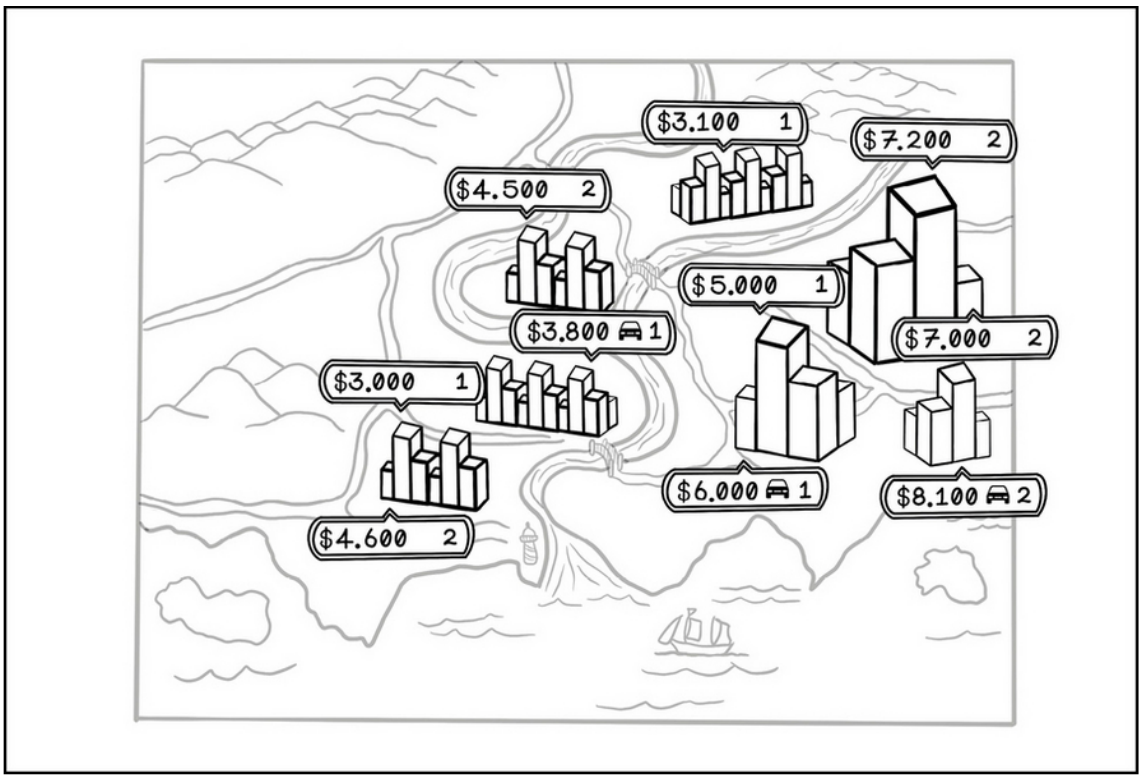


Figura 8.1 Un'illustrazione dei prezzi e delle caratteristiche degli immobili nelle diverse aree.

Questo esempio mostra come utilizziamo i dati per individuare dei modelli e prendere decisioni. Se trovate un bilocale in centro con un garage, è ragionevole supporre che il suo prezzo oscilli sugli 8.000 dollari al mese.

Il *machine learning* ha lo scopo di individuare dei modelli nei dati, per impiegarli in applicazioni utili. Potremmo individuare facilmente lo schema di questo piccolo dataset, ma il machine learning potrà farlo per noi in dataset molto grandi e complessi. La Figura 8.2 illustra le relazioni fra i diversi attributi dei dati. Ogni punto rappresenta una singola proprietà.

Notate che ci sono più punti vicini al centro città e che esiste un chiaro schema relativo al prezzo mensile: il prezzo diminuisce gradualmente a mano a mano che aumenta la distanza dal centro. C'è anche un modello

nel prezzo mensile relativo al numero di stanze; il divario fra il gruppo di punti in basso e il gruppo in alto mostra che il prezzo varia in modo significativo. Potremmo ingenuamente presumere che questo effetto possa essere correlato alla distanza dal centro città. Gli algoritmi di machine learning possono aiutarci a convalidare o confutare questa ipotesi. In questo capitolo approfondiremo esattamente il funzionamento di questo processo.

In genere, i dati sono rappresentati in tabelle. Le colonne sono denominate *caratteristiche* dei dati e le righe sono denominate *esempi*. Quando confrontiamo due caratteristiche, la caratteristica che viene misurata è talvolta rappresentata come y e le caratteristiche che vengono modificate sono raggruppate come x. Otterremo una migliore comprensione di questa terminologia a mano a mano che risolveremo alcuni problemi.

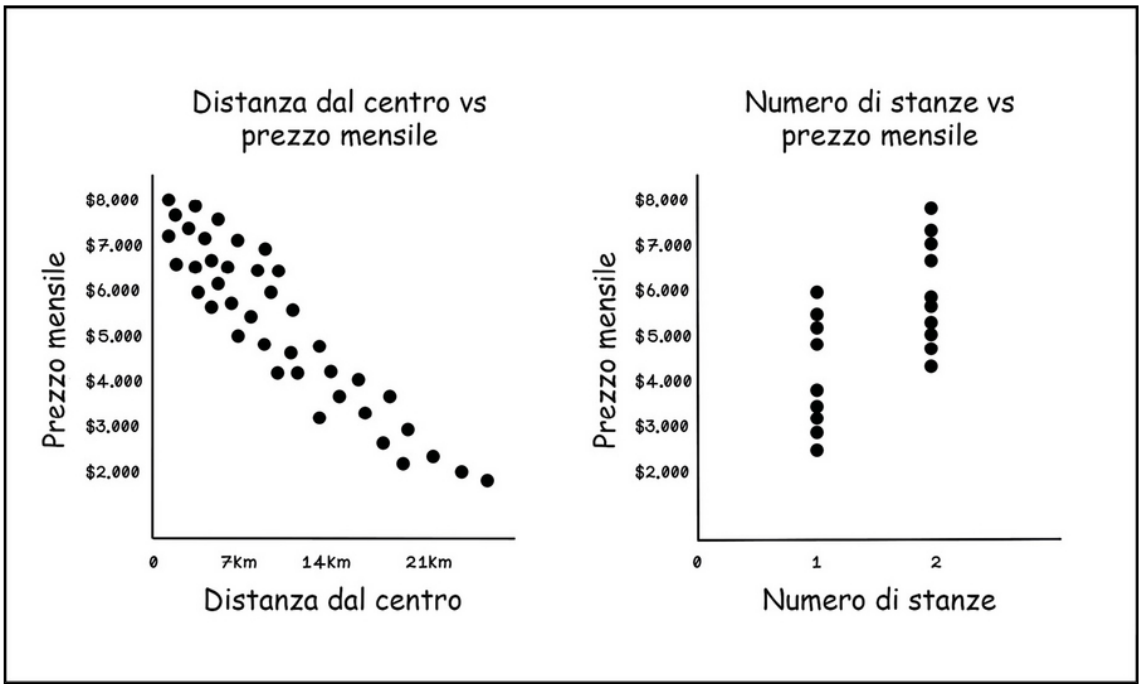


Figura 8.2 Esempio di visualizzazione delle relazioni fra i dati.

Problemi risolvibili dal machine learning

Il machine learning è utile quando disponete di dati e intendete porre domande alle quali i dati potrebbero dare una risposta. Gli algoritmi di machine learning trovano gli schemi presenti nei dati, ma non possono "fare magie". Le varie categorie di algoritmi di machine learning utilizzano approcci differenti per scenari differenti per rispondere a domande differenti. Tali approcci si chiamano apprendimento con supervisione, apprendimento senza supervisione e apprendimento per rinforzo (Figura 8.3).

Apprendimento con supervisione

Una delle tecniche più comuni nel machine learning tradizionale è l'*apprendimento con supervisione*. Esaminiamo i dati, ne comprendiamo i modelli e le relazioni e prevediamo i risultati quando ci vengono forniti nuovi esempi di dati diversi ma nello stesso formato. Il problema della ricerca dell'appartamento è un esempio di apprendimento con supervisione, alla ricerca di uno schema. Vediamo questo esempio in azione anche quando digitiamo una ricerca testuale che si completa automaticamente o quando le applicazioni musicali ci suggeriscono nuovi brani in base alla nostra attività e alle nostre preferenze. L'apprendimento con supervisione rientra in due sottocategorie: regressione e classificazione.

La *regressione* comporta il tracciamento di una linea attraverso un insieme di punti di dati, alla ricerca del miglior adattamento alla forma complessiva dei dati. La regressione può essere utilizzata per applicazioni come l'individuazione di tendenze fra iniziative di marketing e vendite. La domanda potrebbe essere: "Esiste una relazione diretta fra il marketing tramite annunci online e le vendite effettive di un prodotto?".

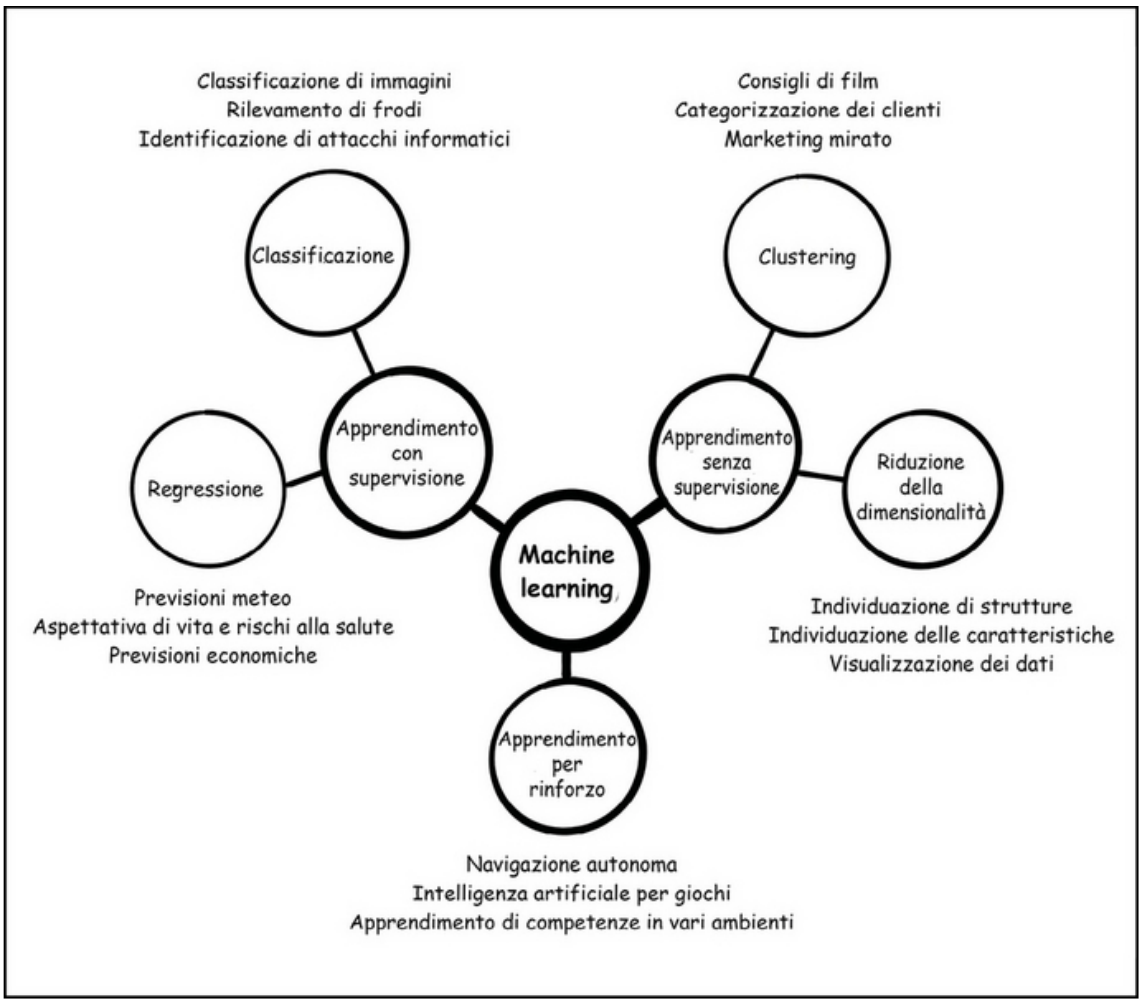


Figura 8.3 Categorizzazione del machine learning e dei suoi utilizzi.

Può anche essere utilizzata per determinare i fattori che influenzano qualcosa: “Esiste una relazione diretta fra il tempo e il valore di una criptovaluta? La criptovaluta aumenterà esponenzialmente di valore con il passare del tempo?”.

La *classificazione* ha lo scopo di creare categorie fra gli esempi in base alle loro caratteristiche: “Possiamo determinare se qualcosa è un’auto o un camion in base al numero di ruote, al peso e alla velocità massima?”.

Apprendimento senza supervisione

L'*apprendimento senza supervisione* comporta la ricerca di modelli nei dati che potrebbero essere difficili da estrarre ispezionando manualmente i dati. L'apprendimento senza supervisione è utile per la categorizzazione (*clustering*) di dati con caratteristiche simili e per scoprire caratteristiche importanti nei dati. Su un sito di e-commerce, per esempio, i prodotti potrebbero essere raggruppati in base ai comportamenti di acquisto dei clienti. Se molti clienti acquistano insieme sapone, spugne e asciugamani, è probabile che anche altri clienti desiderino quella combinazione di prodotti, quindi sapone, spugne e asciugamani dovrebbero essere raggruppati e consigliati insieme ai clienti.

Apprendimento per rinforzo

L'*apprendimento per rinforzo* si ispira alla psicologia comportamentale e opera premiando o punendo un algoritmo in base alle sue attività in un ambiente. Ha somiglianze con l'apprendimento con e senza supervisione, ma con molte differenze. L'apprendimento per rinforzo ha lo scopo di istruire un agente in un ambiente basato su premi e sanzioni. Immaginate di premiare con dei dolcetti un animale domestico per i buoni comportamenti; più viene ricompensato per un determinato comportamento, più esibirà quel comportamento. Parleremo dell'apprendimento per rinforzo nel Capitolo 10.

Un flusso di lavoro di machine learning

Il machine learning non riguarda solo gli algoritmi. Spesso influenza anche il contesto di raccolta dei dati, la preparazione dei dati e le domande da porre. Vi sono due tipi di domande.

- Un problema può essere risolto con il machine learning, e per risolverlo è necessario raccogliere i dati giusti. Supponiamo che una banca disponga di una grande quantità di dati relativi a transazioni legittime e fraudolente e che voglia addestrare un modello con questa domanda: “Possiamo rilevare le transazioni fraudolente in tempo reale?”.
- Disponiamo di dati in un determinato contesto e vogliamo determinare come possono essere utilizzati per risolvere diversi problemi. Un’azienda agricola, per esempio, potrebbe disporre di dati sul clima in varie aree, sui concimi richiesti per varie coltivazioni e sulla composizione del suolo in quelle aree. La domanda potrebbe essere: “Quali correlazioni e relazioni possiamo trovare fra i diversi tipi di dati?”. Queste relazioni possono informare una domanda più concreta, come: “Possiamo determinare la posizione migliore per coltivare una determinata coltura in base al clima e alla composizione del suolo in quella posizione?”.

La Figura 8.4 mostra uno schema semplificato dei passaggi coinvolti in una tipica attività di machine learning.

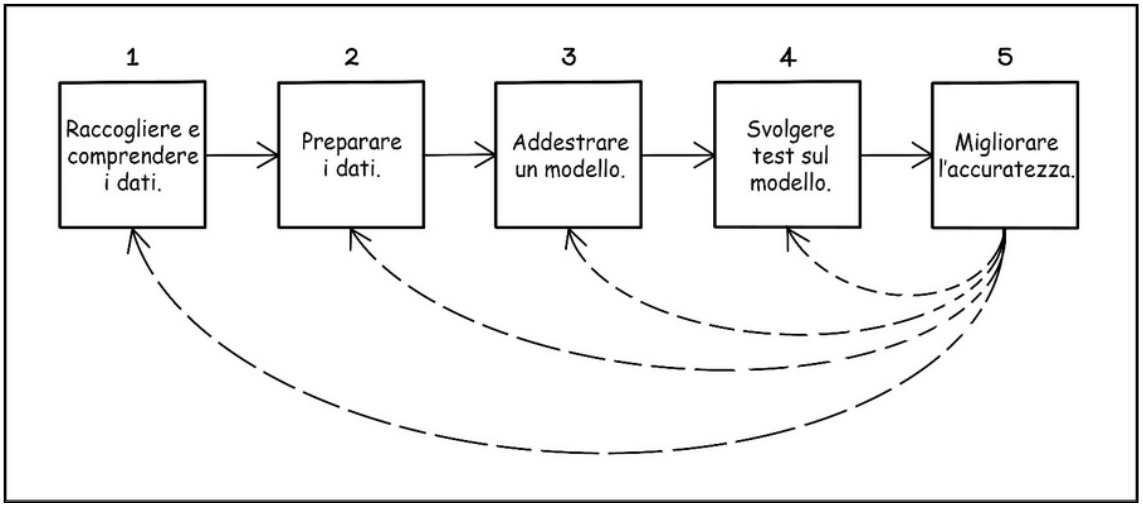


Figura 8.4 Un flusso di lavoro per esperimenti e progetti di machine learning.

Raccolta e conoscenza dei dati: conoscere il contesto

Raccogliere e comprendere i dati con cui state lavorando è fondamentale per avere successo in un'attività di machine learning. Se lavorate nel settore finanziario, la conoscenza della terminologia e del funzionamento dei processi e dei dati in quell'area è importante per reperire i dati migliori con lo scopo di rispondere a domande relative all'obiettivo che state cercando di raggiungere. Se dovete creare un sistema di rilevamento delle frodi, è importante capire quali dati vengono archiviati dai sistemi sulle transazioni e che cosa significano. Potrebbe anche essere necessario estrarre i dati da più sistemi e combinarli per renderli più efficaci. A volte, i dati che utilizziamo vengono integrati con dati provenienti dall'esterno dell'azienda, per migliorarne l'efficacia. In questo paragrafo, utilizzeremo un dataset di esempio sulle misurazioni dei diamanti per comprendere il flusso di lavoro di un'attività di machine learning ed esplorare i vari algoritmi impiegabili (Figura 8.5).

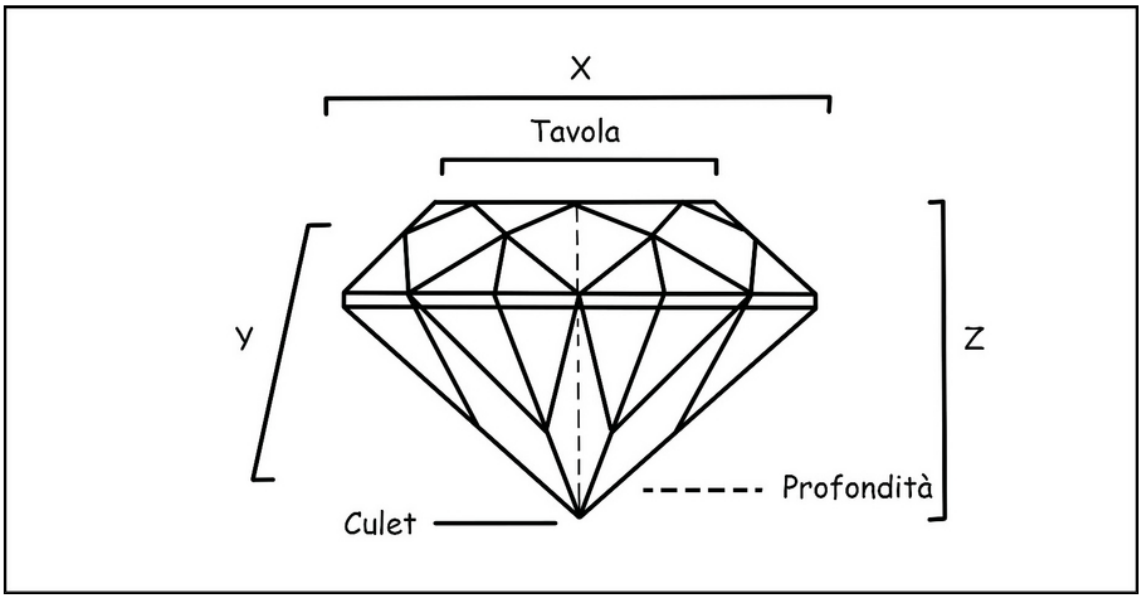


Figura 8.5 Terminologia delle misurazioni nel mondo dei diamanti.

La Tabella 8.1 descrive diversi diamanti e le loro proprietà. X, Y e Z descrivono la grandezza di un diamante nelle tre dimensioni spaziali.

Negli esempi viene utilizzato solo un sottoinsieme di tali dati.

Tabella 8.1 Il dataset per i diamanti.

	Carati	Taglio	Colore	Purezza	Profondità	Tavola	Prezzo	X	Y	Z
1	0,30	Good	J	SI1	64,0	55	339	4,25	4,28	2,73
2	0,41	Ideal	I	SI1	61,7	55	561	4,77	4,80	2,95
3	0,75	Very Good	D	SI1	63,2	56	2760	5,80	5,75	3,65
4	0,91	Fair	H	SI2	65,7	60	2763	6,03	5,99	3,95
5	1,20	Fair	F	I1	64,6	56	2809	6,73	6,66	4,33
6	1,31	Premium	J	SI2	59,7	59	3697	7,06	7,01	4,20
7	1,50	Premium	H	I1	62,9	60	4022	7,31	7,22	4,57
8	1,74	Very Good	H	I1	63,2	55	4677	7,62	7,59	4,80
9	1,96	Fair	I	I1	66,8	55	6147	7,62	7,60	5,08
10	2,21	Premium	H	I1	62,2	58	6535	8,31	8,27	5,16

Il dataset per i diamanti è costituito da dieci colonne di dati, denominate *caratteristiche*. Il dataset completo ha più di 50.000 righe. Ecco il significato di ogni caratteristica.

- *Carati*: il peso del diamante. 1 carato equivale a 200 mg.
- *Taglio*: la qualità del diamante. La scala della qualità è: Fair, Good, Very Good, Premium e Ideal.
- *Colore*: il colore del diamante, che va da D, il colore migliore, a J, il colore peggiore. D indica un diamante limpido e J indica un diamante opaco.
- *Purezza*: le imperfezioni del diamante, per qualità decrescente: FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2 e I3. Non è importante

capire il significato di questi nomi in codice; rappresentano semplicemente i diversi livelli di perfezione.
- *Profondità*: la percentuale di profondità, misurata dal culet alla tavola del diamante. In genere, il rapporto tavola-profondità è importante per l'estetica "brillante" di un diamante.
- *Tavola*: la percentuale dell'estremità piatta del diamante rispetto alla dimensione *x*.
- *Prezzo*: il prezzo del diamante quando è stato venduto.
- *X*: la dimensione *x* del diamante, in millimetri.
- *Y*: la dimensione *y* del diamante, in millimetri.
- *Z*: la dimensione *z* del diamante, in millimetri.

Tenete a mente questo dataset; lo useremo per vedere in quale modo i dati vengono preparati ed elaborati dagli algoritmi di machine learning.

Preparazione dei dati: pulizia e valutazione

I dati effettivamente esistenti non sono mai l'ideale per lavorare. Potrebbero provenire da sistemi differenti e aziende differenti, e potrebbero riferirsi a standard e regole differenti, anche in termini di integrità dei dati stessi. Ci sono sempre dati mancanti, dati incoerenti e dati in un formato con cui gli algoritmi faticherebbero a lavorare.

Nel dataset dei diamanti della Tabella 8.2, ancora una volta, è importante ricordare che le colonne sono le *caratteristiche* dei dati e che ogni riga è un *esempio*.

Tabella 8.2 Dataset dei diamanti con dati mancanti.

	Carati	Taglio	Colore	Purezza	Profondità	Tavola	Prezzo	X	Y	Z
1	0,30	Good	J	SI1	64,0	55	339	4,25	4,28	2,73
2	0,41	Ideal	I	**si1**	61,7	55	561	4,77	4,80	2,95
3	0,75	Very Good	D	SI1	63,2	56	2760	5,80	5,75	3,65

	Carati	Taglio	Colore	Purezza	Profondità	Tavola	Prezzo	X	Y	Z
4	0,91		H	SI2	**-**	60	2763	6,03	5,99	3,95
5	1,20	Fair	F	I1	64,6	56	2809	6,73	6,66	4,33
6	1,21	Good	E	I1	57,2	62	3144	7,01	6,96	3,99
7	1,31	Premium	J	SI2	59,7	59	3697	7,06	7,01	4,20
8	1,50	Premium	H	I1	62,9	60	4022	7,31	7,22	4,57
9	1,74	Very Good	H	**i1**	63,2	55	4677	7,62	7,59	4,80
10	1,83	**fair**	J	I1	70,0	58	5083	7,34	7,28	5,12
11	1,96	Fair	I	I1	66,8	55	6147	7,62	7,60	5,08
12	**-**	Premium	H	**i1**	62,2	**-**	6535	8,31	**-**	5,16

Dati mancanti

Nella Tabella 8.2, all'Esemplare 4 mancano i valori per le funzioni Taglio e Profondità; all'Esemplare 12 mancano i valori per Carati, Tavola e Y. Per confrontare gli esempi, abbiamo bisogno di una conoscenza completa dei dati, e i valori mancanti complicano le cose. Un obiettivo per un progetto di machine learning potrebbe essere quello di stimare questi valori; occupiamoci quindi delle stime nel materiale in arrivo. Supponiamo che i dati mancanti si riveleranno problematici per il nostro obiettivo di utilizzarli per qualcosa di utile. Ecco alcuni modi per gestire i dati mancanti.

- *Rimozione*: rimuovere gli esempi che hanno valori mancanti per le caratteristiche, in questo caso gli esempi 4 e 12 (Tabella 8.3). Il vantaggio di questo approccio è che i dati diventano più affidabili,

perché non si presume nulla; tuttavia, gli esempi rimossi potrebbero essere importanti per l'obiettivo che stiamo cercando di raggiungere.

Tabella 8.3 Il dataset dei diamanti con dati mancanti: rimozione degli esempi.

	Carati	Taglio	Colore	Purezza	Profondità	Tavola	Prezzo	X	Y	Z
1	0,30	Good	J	SI1	64,0	55	339	4,25	4,28	2,73
2	0,41	Ideal	I	**si1**	61,7	55	561	4,77	4,80	2,95
3	0,75	Very Good	D	SI1	63,2	56	2760	5,80	5,75	3,65
4	**0,91**		**H**	**SI2**	**-**	**60**	**2763**	**6,03**	**5,99**	**3,95**
5	1,20	Fair	F	I1	64,6	56	2809	6,73	6,66	4,33
6	1,21	Good	E	I1	57,2	62	3144	7,01	6,96	3,99
7	1,31	Premium	J	SI2	59,7	59	3697	7,06	7,01	4,20
8	1,50	Premium	H	I1	62,9	60	4022	7,31	7,22	4,57
9	1,74	Very Good	H	**i1**	63,2	55	4677	7,62	7,59	4,80
10	1,83	**fair**	J	I1	70,0	58	5083	7,34	7,28	5,12
11	1,96	Fair	I	I1	66,8	55	6147	7,62	7,60	5,08
12	**-**	**Premium**	**H**	**I1**	**62,2**	**-**	**6535**	**8,31**	**-**	**5,16**

- *Media o mediana*: un'altra opzione consiste nel sostituire i valori mancanti con la media o la mediana della rispettiva caratteristica. La *media* viene calcolata sommando tutti i valori e dividendo per il numero di esempi. La *mediana* viene calcolata ordinando gli esempi per valore crescente e scegliendo il valore centrale.
  Usare la media è facile ed efficiente, ma non tiene conto delle possibili correlazioni fra le caratteristiche. Questo approccio non

può essere utilizzato con caratteristiche categoriche come Taglio, Purezza e Profondità nel dataset dei diamanti (Tabella 8.4).

Tabella 8.4 Il dataset dei diamanti con dati mancanti: utilizzo dei valori medi.

	Carati	Taglio	Colore	Purezza	Profondità	Tavola	Prezzo	X	Y	Z
1	0,30	Good	J	SI1	64,0	55	339	4,25	4,28	2,73
2	0,41	Ideal	I	**si1**	61,7	55	561	4,77	4,80	2,95
3	0,75	Very Good	D	SI1	63,2	56	2760	5,80	5,75	3,65
4	0,91		H	SI2	-	60	2763	6,03	5,99	3,95
5	1,20	Fair	F	I1	64,6	56	2809	6,73	6,66	4,33
6	1,21	Good	E	I1	57,2	62	3144	7,01	6,96	3,99
7	1,31	Premium	J	SI2	59,7	59	3697	7,06	7,01	4,20
8	1,50	Premium	H	I1	62,9	60	4022	7,31	7,22	4,57
9	1,74	Very Good	H	**i1**	63,2	55	4677	7,62	7,59	4,80
10	1,83	**fair**	J	I1	70,0	58	5083	7,34	7,28	5,12
11	1,96	Fair	I	I1	66,8	55	6147	7,62	7,60	5,08
12	**1,19**	Premium	H	I1	62,2	**57**	6535	8,31	-	5,16

- Per calcolare la media della funzione Tavola, sommiamo tutti i valori disponibili e dividiamo il totale per il numero di valori utilizzati:

```
Media della Tavola = (55 + 55 + 56 + 60 + 56 + 62 + 59 + 60 + 55 +
58 + 55) / 11
Media della Tavola = 631 / 11
Media della Tavola = 57,364
```

L'uso della media della Tavola per i valori mancanti sembra avere senso, perché la dimensione della tabella non sembra differire enormemente fra i diversi esemplari. Ma potrebbero esserci correlazioni che non vediamo, come la relazione fra la dimensione della tavola e la larghezza del diamante (la dimensione X).
D'altra parte, usare la media per i carati non ha senso, perché possiamo vedere una correlazione fra la caratteristica Carati e la caratteristica Prezzo, se tracciamo i dati su un grafico. Il prezzo aumenta all'aumentare del valore in carati.

- *Più frequente*: sostituire i valori mancanti con il valore presente più spesso per quella caratteristica, la *moda* dei dati. Questo approccio si comporta molto bene con le caratteristiche categoriche, ma non tiene conto delle possibili correlazioni fra le caratteristiche e può introdurre distorsioni.
- (*avanzato*) *Approcci statistici*: utilizzare il sistema k-nearest neighbor o reti neurali. Il sistema k-nearest neighbor utilizza varie caratteristiche dei dati per trovare un valore stimato. Come k-nearest neighbor, una rete neurale può prevedere accuratamente i valori mancanti, avendo dati sufficienti. Entrambi gli algoritmi sono computazionalmente costosi, per il solo scopo di gestire i dati mancanti.
- (*avanzato*) *Non fare nulla*: alcuni algoritmi gestiscono i dati mancanti senza alcun problema, come XGBoost, ma gli algoritmi non avranno successo nel farlo.

Valori ambigui

Un altro problema sono i valori che significano la stessa cosa ma sono rappresentati in modo differente. Esempi nel dataset dei diamanti sono le righe 2, 9, 10 e 12. I valori per le funzioni Taglio e Purezza sono minuscoli anziché maiuscoli. Notate che lo sappiamo solo perché leggiamo queste caratteristiche e i valori possibili. Senza poter contare su

questa conoscenza, potremmo considerare Fair e fair come categorie diverse. Per risolvere questo problema, possiamo standardizzare questi valori in maiuscolo o minuscolo, in modo da garantirne la coerenza (Tabella 8.5).

Tabella 8.5 Il dataset dei diamanti con dati ambigui: standardizzazione dei valori.

	Carati	Taglio	Colore	Purezza	Profondità	Tavola	Prezzo	X	Y	Z
1	0,30	Good	J	SI1	64,0	55	339	4,25	4,28	2,73
2	0,41	Ideal	I	**si1**	61,7	55	561	4,77	4,80	2,95
3	0,75	Very Good	D	SI1	63,2	56	2760	5,80	5,75	3,65
4	0,91		H	SI2	-	60	2763	6,03	5,99	3,95
5	1,20	Fair	F	I1	64,6	56	2809	6,73	6,66	4,33
6	1,21	Good	E	I1	57,2	62	3144	7,01	6,96	3,99
7	1,31	Premium	J	SI2	59,7	59	3697	7,06	7,01	4,20
8	1,50	Premium	H	I1	62,9	60	4022	7,31	7,22	4,57
9	1,74	Very Good	H	**i1**	63,2	55	4677	7,62	7,59	4,80
10	1,83	**fair**	J	I1	70,0	58	5083	7,34	7,28	5,12
11	1,96	Fair	I	I1	66,8	55	6147	7,62	7,60	5,08
12	1,19	Premium	H	**I1**	62,2	57	6535	8,31	-	5,16

Codifica dei dati categorici

Poiché i computer e i modelli statistici funzionano con valori numerici, ci sarà un problema con la modellazione di valori a stringa e a categoria come Fair, Good, SI1 e I1. Dobbiamo rappresentare questi valori come valori numerici. Ecco i modi per eseguire questa attività.

- *Codifica one-hot*: immaginate degli interruttori, tutti spenti tranne uno. Quello acceso rappresenta la presenza della caratteristica in quella posizione. Se dovessimo rappresentare Taglio con la codifica one-hot, la caratteristica Taglio si trasformerà in cinque caratteristiche differenti con ogni valore a 0 tranne quello che rappresenta il valore Taglio per il rispettivo esempio. Notate che le altre caratteristiche sono state rimosse per motivi di spazio nella Tabella 8.6.

Tabella 8.6 Il dataset dei diamanti con i valori codificati.

	Carati	**Taglio: Fair**	**Taglio: Good**	**Taglio: Very Good**	**Taglio: Premium**	**Taglio: Ideal**
1	0,30	0	**1**	0	0	0
2	0,41	0	0	0	**0**	**1**
3	0,75	0	0	1	0	0
4	0,91	0	0	0	0	0
5	1,20	**1**	0	0	0	0
6	1,21	0	1	0	0	0
7	1,31	0	0	0	**1**	0
8	1,50	0	0	0	**1**	0
9	1,74	0	0	**1**	0	0
10	1,83	**1**	0	0	0	0
11	1,96	**1**	0	0	0	0
12	1,19	0	0	0	**1**	0

- *Codifica a etichetta*: rappresenta ogni categoria come un numero compreso fra 0 e il numero di categorie. Questo approccio dovrebbe

essere utilizzato solo per le classificazioni o le etichette relative a classificazioni; in caso contrario, il modello che addestreremo presupporrà che il numero rappresenti un valore per l'esempio e possa introdurre distorsioni indesiderate.

Esercizio: identificare e correggere i dati del problema di questo esempio
Decidete quali tecniche di preparazione dei dati possono essere utilizzate per correggere il seguente dataset. Decidete quali righe eliminare, per quali valori utilizzare la media e come codificare i valori a categorie. Nota: il dataset è leggermente diverso da quello con cui abbiamo lavorato finora.

	Carati	Origine	Profondità	Tavola	Prezzo	X	Y	Z
1	0,35	Sudafrica	64,0	55	450	4,25		2,73
2	0,42	Canada	61,7	55	680		4,80	2,95
3	0,87	Canada	63,2	56	2689	5,80	5,75	3,65
4	0,99	Botswana	65,7		2734	6,03	5,99	3,95
5	1,34	Botswana	64,6	56	2901	6,73	6,66	
6	1,45	Sudafrica	59,7	59	3723	7,06	7,01	4,20
7	1,65	Botswana	62,9	60	4245	7,31	7,22	4,57
8	1,79		63,2	55	4734	7,62	7,59	4,80
9	1,81	Botswana	66,8	55	6093	7,62	7,60	5,08
10	2,01	Sudafrica	62,2	58	7452	8,31	8,27	5,16

Soluzione
Un approccio per correggere questo dataset prevede le seguenti tre attività.

- *Rimuovere la riga 8 a causa dell'origine mancante*. Non sappiamo per cosa verrà utilizzato il dataset. Se la caratteristica Origine è importante, questa riga mancherà e potrebbe causare problemi. In alternativa, il valore di questa caratteristica potrebbe essere stimato, se ha una relazione con altre caratteristiche.
- *Utilizzare la codifica one-hot per codificare il valore della colonna Origine*. Nell'esempio esplorato finora nel capitolo, abbiamo utilizzato la codifica a etichette per convertire i valori stringa in valori numerici. Questo approccio ha funzionato perché i valori indicavano taglio, chiarezza o colori. Nel caso

dell'Origine, il valore identifica solo la provenienza del diamante. Utilizzando la codifica a etichetta, introduciamo un pregiudizio (bias) nel dataset, perché nessuna Origine è di per sé migliore o peggiore di un'altra in questo dataset.

- *Calcolare la media dei valori mancanti*. Nelle righe 1, 2, 4 e 5 mancano rispettivamente i valori per Y, X, Tavola e Z. Usare un valore medio dovrebbe essere una buona idea, perché, come sappiamo nel caso dei diamanti, le dimensioni e le caratteristiche della tavola sono correlate.

Dati di test e addestramento

Prima di passare all'addestramento di un modello a regressione lineare, dobbiamo assicurarci di disporre dei dati per addestrarlo, nonché di alcuni dati per sottoporre a test la sua efficacia nella previsione di nuovi esempi. Pensate all'esempio del prezzo della proprietà. Dopo esservi fatti un'idea degli attributi che influenzano il prezzo, potreste fare una previsione del prezzo osservando la distanza e il numero di camere. Per questo esempio, utilizzeremo la Tabella 8.7 come dati di addestramento, perché disponiamo di altri dati reali da utilizzare per l'addestramento successivo.

Addestramento di un modello: previsione con la regressione lineare

La scelta dell'algoritmo da utilizzare si basa principalmente su due fattori: la domanda che intendete porre e la natura dei dati disponibili. Se la domanda riguarda la previsione del prezzo di un diamante con una determinata caratura, potete ricorrere a un algoritmo di regressione. La scelta dell'algoritmo dipende anche dal numero di caratteristiche presenti nel dataset e dalle relazioni esistenti fra tali funzionalità. Se i dati hanno molte dimensioni (occorre considerare molte caratteristiche per fare una previsione), possiamo considerare diversi algoritmi e approcci.

La regressione consiste nel prevedere un valore continuo, come il prezzo o il carato del diamante. Con "continuo" si intende il fatto che i valori possono essere un numero qualsiasi entro un intervallo. Il prezzo di 2271 dollari, per esempio, è un valore continuo compreso fra 0 e il prezzo massimo di qualsiasi diamante che la regressione possa aiutare a prevedere.

La regressione lineare è uno degli algoritmi di machine learning più semplici; trova relazioni fra due variabili e ci permette di prevedere una variabile data l'altra. Un esempio consiste nel prevedere il prezzo di un diamante in base al suo valore in carati. Osservando molti esempi di diamanti conosciuti, comprendenti il prezzo e il valore in carati, possiamo insegnare a un modello la relazione, e chiedergli di stimare una previsione.

Individuare una linea nei dati

Cominciamo cercando di individuare una tendenza nei dati, per tentare alcune previsioni. Per esplorare la regressione lineare, la domanda che ci poniamo è: "Esiste una correlazione fra i carati di un diamante e il suo prezzo? Se esiste, possiamo usarla per fare previsioni accurate?"

Iniziamo isolando le caratteristiche Carati e Prezzo e tracciando i dati su un grafico. Poiché vogliamo trovare il prezzo in base al valore in carati, tratteremo i carati come *x* e il prezzo come *y*. Perché abbiamo scelto questo approccio?

- *Carati come variabile indipendente (x)*: la variabile *indipendente* è quella che modifichiamo in un esperimento per valutare l'effetto su una variabile dipendente. In questo esempio, il valore dei carati verrà modificato per determinare il prezzo di un diamante di quella caratura.
- *Prezzo come variabile dipendente (y)*: la variabile *dipendente* è quella che viene sottoposta a test. È influenzata dalla variabile indipendente e cambia in base alle modifiche apportate al valore

della variabile indipendente. Nel nostro esempio, siamo interessati al prezzo, dato uno specifico peso in carati.

La Figura 8.6 mostra i dati relativi ai carati e al prezzo tracciati su un grafico; la Tabella 8.7 descrive i dati effettivi.

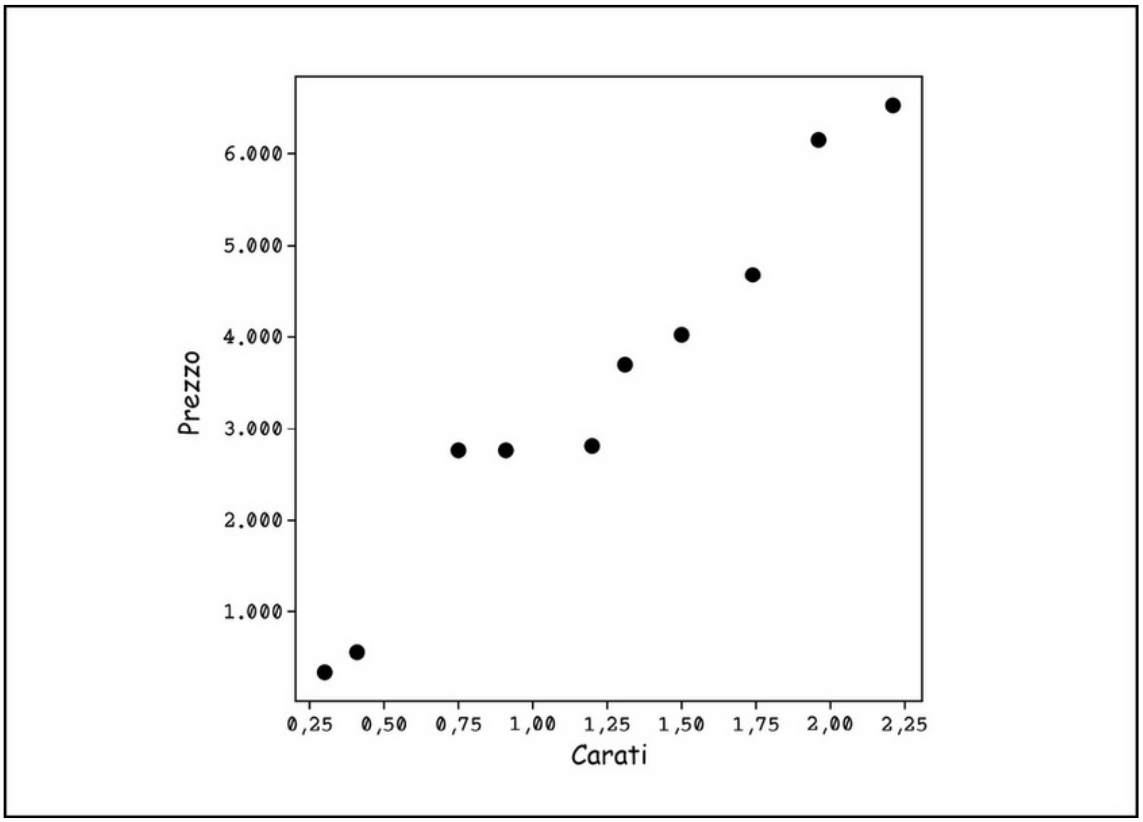


Figura 8.6 Un grafico a dispersione su carati e prezzi.

Tabella 8.7 Dati dei carati e dei prezzi.

	Carati (x)	**Prezzo (y)**
1	0,30	339
2	0,41	561
3	0,75	2760
4	0,91	2763
5	1,20	2809

6	1,31	3697
7	1,50	4022
8	1,74	4677
9	1,96	6147
10	2,21	6535

Notate che i carati presentano valori molto piccoli rispetto al prezzo. Il prezzo è in migliaia (di dollari) e i carati sono nell'intervallo dei decimali. Per rendere più uniformi i valori ai fini dell'apprendimento in questo capitolo, possiamo estendere di scala i valori dei Carati, in modo che siano confrontabili con i valori Prezzo. Moltiplicando ogni valore in carati per 1000, otteniamo numeri più facili da calcolare a mano nelle prossime procedure. Notate che, ridimensionando *tutte* le righe, in realtà non influenziamo le relazioni insite nei dati, perché a ogni esempio applichiamo la stessa operazione. I dati risultanti (Figura 8.7) sono rappresentati nella Tabella 8.8.

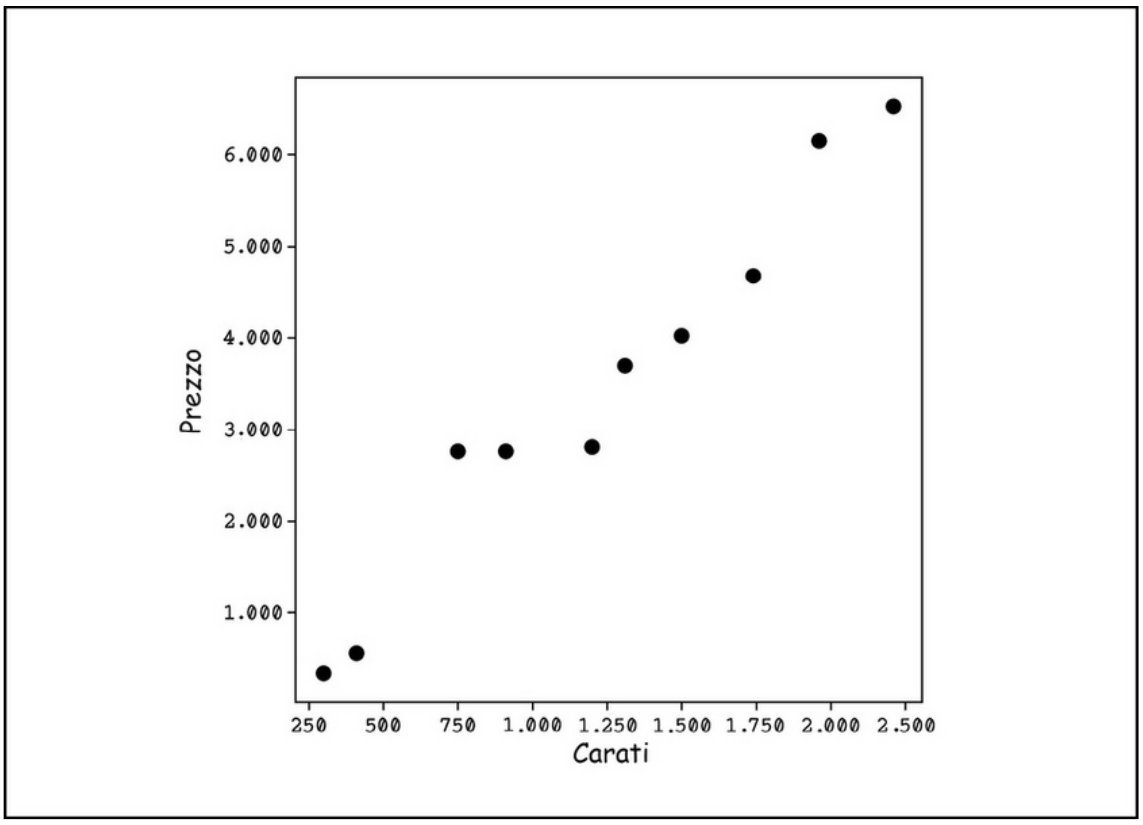


Figura 8.7 Un grafico a dispersione su carati e prezzi.

Tabella 8.8 Dati dei carati e dei prezzi.

	Carati (x)	**Prezzo (y)**
1	300	339
2	410	561
3	750	2760
4	910	2763
5	1200	2809
6	1310	3697
7	1500	4022
8	1740	4677

9	1960	6147
10	2210	6535

Calcolare la media delle caratteristiche

La prima cosa che dobbiamo fare per trovare una retta di regressione è calcolare la media per ogni caratteristica. La media è data dalla somma di tutti i valori divisa per il numero di valori. La media per i Carati è 1229, rappresentata dalla linea verticale sull'asse *x*. La media per il Prezzo è di 3431, rappresentata dalla linea orizzontale sull'asse *y* (Figura 8.8).

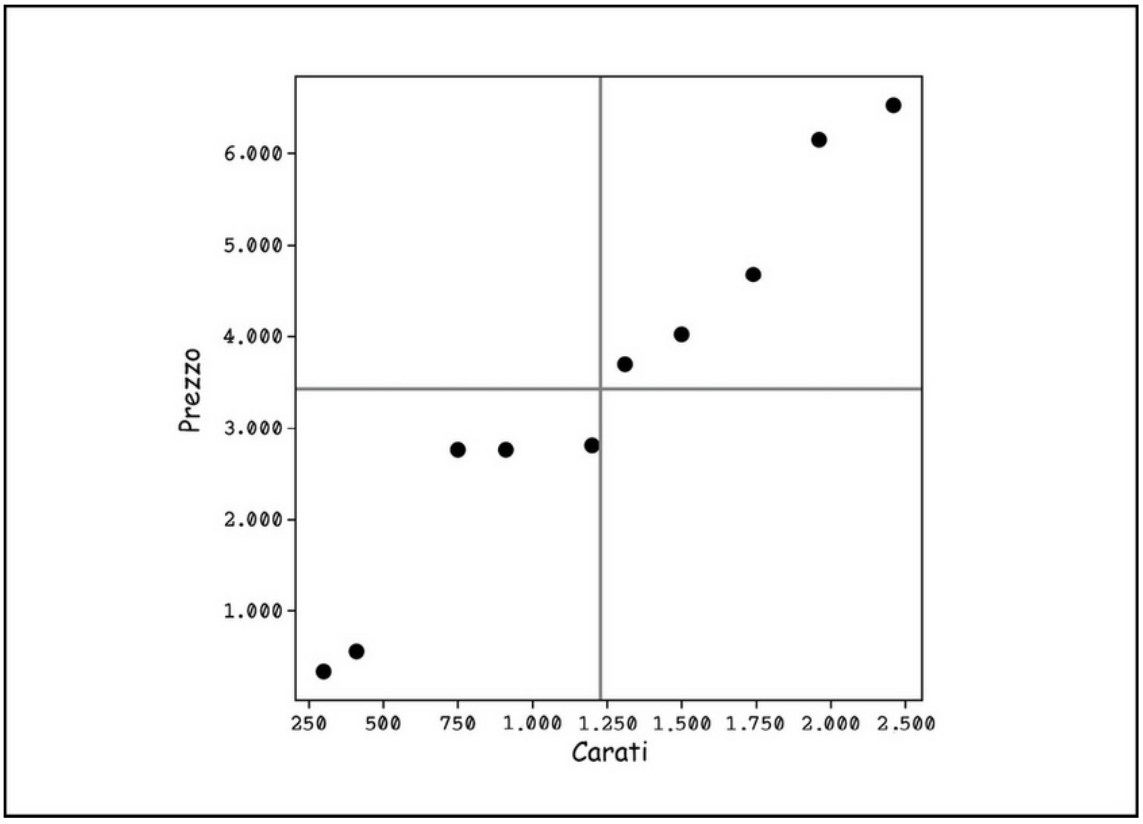


Figura 8.8 La media di x (linea verticale) e di y (linea orizzontale).

La media è importante, perché matematicamente, qualsiasi linea di regressione che troviamo passerà attraverso l'intersezione fra la media di

x e la media di *y*. Ma molte linee possono passare attraverso questo punto. Alcune linee di regressione potrebbero essere migliori di altre nell'individuare i dati. Il *metodo dei minimi quadrati* ha lo scopo di creare una linea che minimizzi le distanze fra la linea e tutti i punti che formano il dataset. Il metodo dei minimi quadrati è molto utilizzato per trovare le linee di regressione. La Figura 8.9 illustra alcuni esempi di linee di regressione.

Trovare le rette di regressione con il metodo dei minimi quadrati

Ma qual è lo scopo di una retta di regressione? Supponiamo di costruire una metropolitana che cerchi di passare il più vicino possibile a tutti i principali edifici per uffici. Non è possibile avere una linea metropolitana "spezzata" che visiti ogni edificio; ci sarebbero troppe stazioni e costerebbe molto. Quindi, proveremo a creare un percorso lineare che riduca al minimo la distanza da ciascun edificio. Alcuni pendolari potrebbero dover camminare un po' di più di altri, ma la linea retta è ottimizzata come servizio per tutti. Questo è esattamente l'obiettivo di una linea di regressione; gli edifici sono i punti di dati e la linea è il percorso rettilineo della metropolitana (Figura 8.10).

La regressione lineare troverà sempre una linea retta che "attraversa" i dati e riduce al minimo la distanza fra i punti in generale. Comprendere l'equazione di una linea è importante, perché ci consentirà di trovare i valori per le variabili che descrivono la linea.

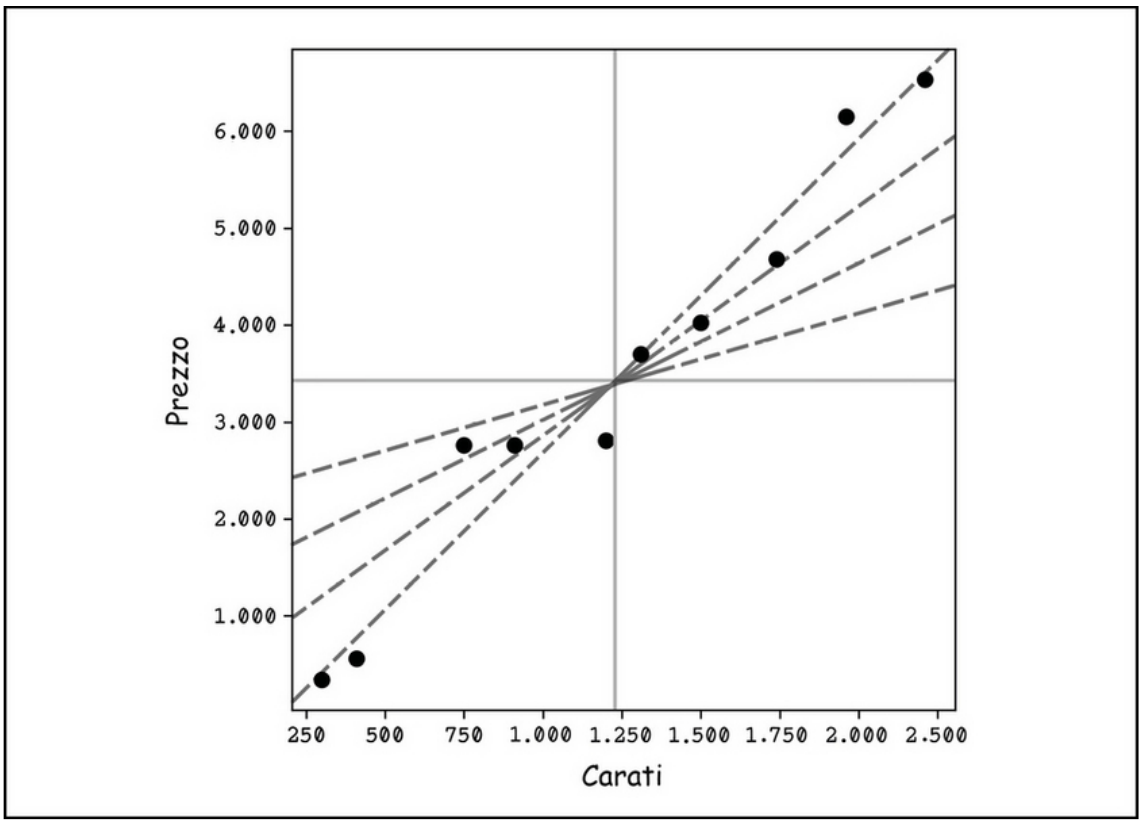


Figura 8.9 Possibili rette di regressione.

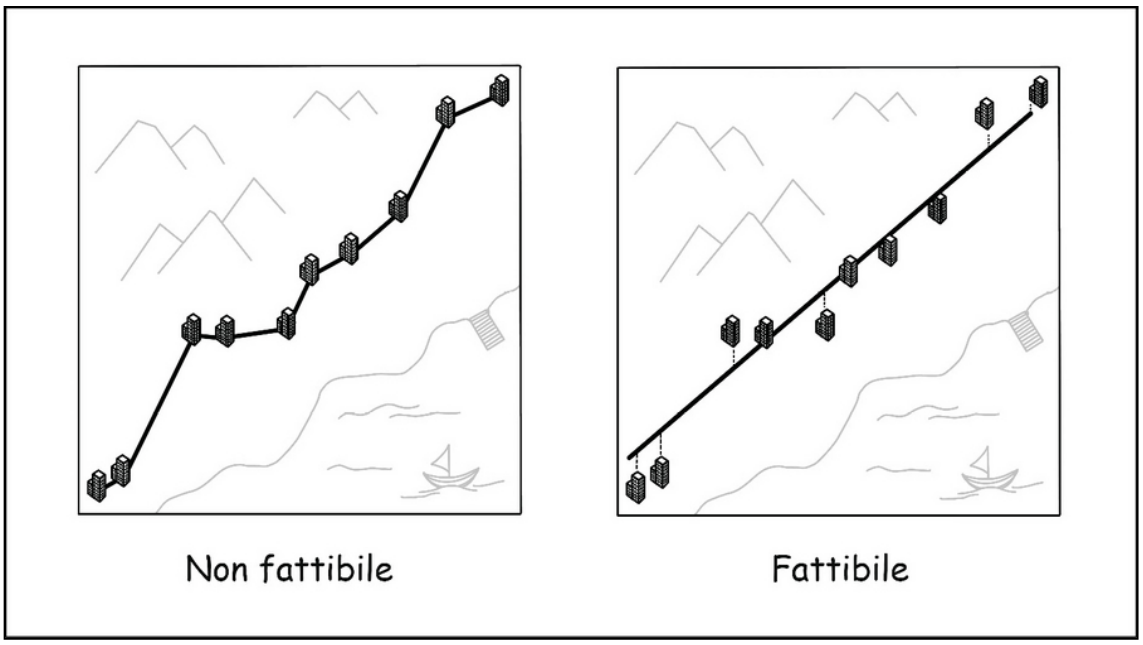


Figura 8.10 Concetti legati alle rette di regressione.

Una retta è rappresentata dall'equazione $y = c + mx$ (Figura 8.11).

- y: la variabile dipendente.
- x: la variabile indipendente.

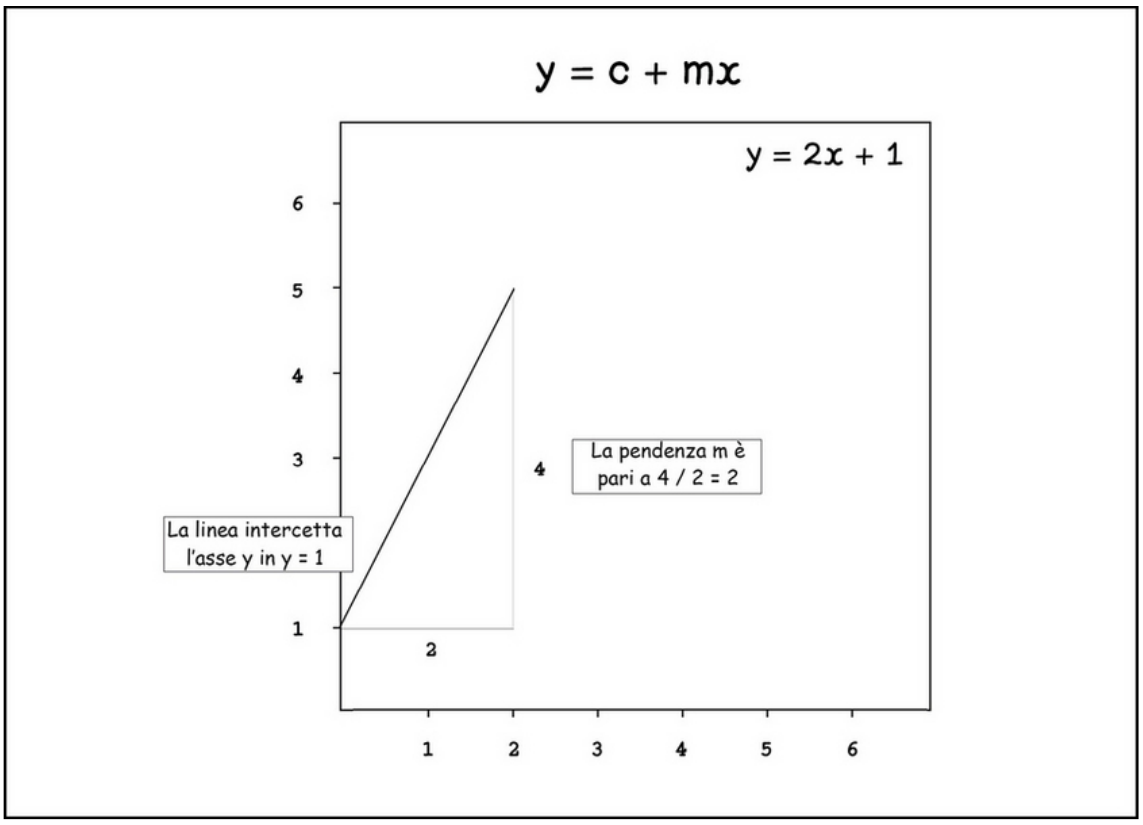


Figura 8.11 Elementi dell'equazione che rappresenta una retta.

- *m*: la pendenza della linea.
- *c*: il valore *y* in cui la linea intercetta l'asse *y*.

Il metodo dei minimi quadrati consente di trovare la retta di regressione. In generale, il processo prevede i passaggi illustrati nella Figura 8.12. Per trovare la linea più vicina ai dati, calcoliamo la differenza fra i valori *effettivi* dei dati e i valori *previsti* dei dati. Le differenze saranno diverse per i vari punti di dati: alcune saranno grandi e altre piccole. Alcune differenze saranno valori negativi e altre positivi. Elevando al quadrato le differenze e sommandole, equipariamo tutte le differenze (positive e negative) per tutti i punti di dati. A questo punto, ridurre al minimo la differenza totale (i dei minimi quadrati) consente di ottenere una buona linea di regressione. Non preoccupatevi se la Figura 8.12 sembra un po' complessa; ne esamineremo ogni passaggio.

Finora, la nostra linea ha alcune variabili note. Sappiamo che un valore *x* è 1229 e un valore *y* è 3431, come mostrato nel Passaggio 2.

Successivamente, calcoliamo la differenza fra ogni valore in carati e la media dei carati, nonché la differenza fra ogni valore di prezzo e la media dei prezzi, per trovare (*x* – media di x) e (*y* – media di *y*), che viene utilizzata al punto 3 (Tabella 8.9).

Per il Passaggio 3, dobbiamo anche calcolare il quadrato della differenza fra ogni carato e la media in carati da trovare $(x - \text{media di } x)^2$. Dobbiamo anche sommare tutti questi valori, per ottenere il valore da minimizzare, che equivale a 3.703.690 (Tabella 8.10).

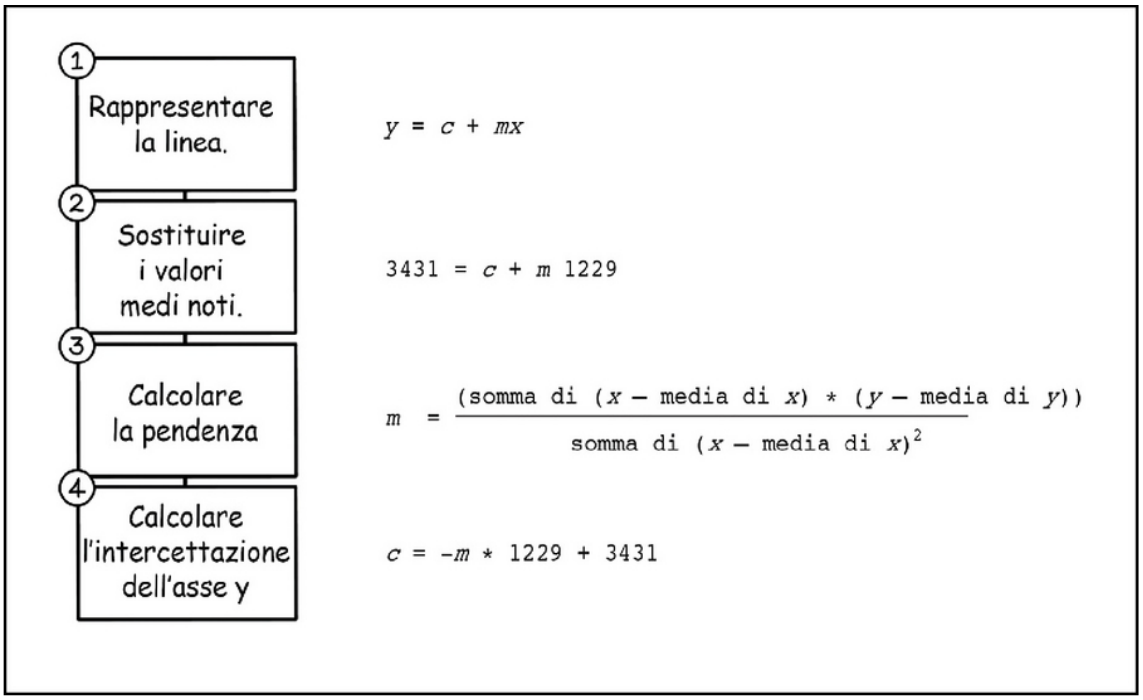


Figura 8.12 Il flusso di lavoro per il calcolo di una retta di regressione.

Tabella 8.9 Il dataset dei diamanti e i relativi calcoli.

	Carati (x)	**Prezzo (y)**	**x – media di x**		**y – media di y**	
1	300	339	300 – 1229	-929	339 – 3431	-3092
2	410	561	410 – 1229	-819	561 – 3431	-2870

3	750	2760	750 – 1229	-479	2760 – 3431	-671
4	910	2763	910 – 1229	-319	2763 – 3431	-668
5	1200	2809	2100 – 1229	-29	2809 – 3431	-622
6	1310	3697	1310 – 1229	81	3697 – 3431	266
7	1500	4022	1500 – 1229	271	4022 – 3431	591
8	1740	4677	1740 – 1229	511	4677 – 3431	1246
9	1960	6147	1960 – 1229	731	6147 – 3431	2716
10	2210	6535	2210 – 1229	981	6535 – 3431	3104
	1229	3431				
	Medie					

Tabella 8.10 Il dataset dei diamanti e i relativi calcoli, Parte 2.

	Carati (x)	**Prezzo (y)**	**x – media di x**		**y – media di y**		**(x – media di x)^2**
1	300	339	300 – 1229	-929	339 – 3431	-3092	863.041
2	410	561	410 – 1229	-819	561 – 3431	-2870	670.761
3	750	2760	750 – 1229	-479	2760 – 3431	-671	229.441
4	910	2763	910 – 1229	-319	2763 – 3431	-668	101.761
5	1200	2809	2100 – 1229	-29	2809 – 3431	-622	841
6	1310	3697	1310 – 1229	81	3697 – 3431	266	6561
7	1500	4022	1500 – 1229	271	4022 – 3431	591	73.441
8	1740	4677	1740 – 1229	511	4677 – 3431	1246	261.121
9	1960	6147	1960 – 1229	731	6147 – 3431	2716	534.361
10	2210	6535	2210 – 1229	981	6535 – 3431	3104	962.361

	Carati (x)	Prezzo (y)	x – media di x		y – media di y		(x – media di x)^2
	1229	3431					3.703.690
	Medie						Somme

L'ultimo valore mancante per l'equazione nel Passaggio 3 è il valore per (*x* – media di *x*) * (*y* – media di *y*). Di nuovo, è richiesta la somma dei valori. La somma è pari a 11.624.370 (Tabella 8.11).

Tabella 8.11 Il dataset dei diamanti e i relativi calcoli, Parte 3.

	Carati (x)	Prezzo (y)	x – media di x		y – media di y		(x – media di x)^2	(x – media di x) * (y – media di y)
1	300	339	300 – 1229	-929	339 – 3431	-3092	863.041	2.872.468
2	410	561	410 – 1229	-819	561 – 3431	-2870	670.761	2.350.530
3	750	2760	750 – 1229	-479	2760 – 3431	-671	229.441	321.409
4	910	2763	910 – 1229	-319	2763 – 3431	-668	101.761	213.092
5	1200	2809	2100 – 1229	-29	2809 – 3431	-622	841	18.038
6	1310	3697	1310 – 1229	81	3697 – 3431	266	6561	21.546
7	1500	4022	1500 – 1229	271	4022 – 3431	591	73.441	160.161

8	1740	4677	1740 – 1229	511	4677 – 3431	1246	261.121	636.706
9	1960	6147	1960 – 1229	731	6147 – 3431	2716	534.361	1.985.396
10	2210	6535	2210 – 1229	981	6535 – 3431	3104	962.361	3.045.024
	1229	3431					3.703.690	**11.624.370**
	Medie						Somme	

Ora possiamo mettere insieme i valori calcolati nell'equazione dei minimi quadrati per calcolare *m*:

```
m = 11624370 / 3703690
m = 3,139
```

Ora che abbiamo un valore per *m*, possiamo calcolare *c* sostituendo i valori medi per *x* e *y*. Ricordate che tutte le linee di regressione passeranno questo punto, che quindi è un punto noto della linea di regressione:

```
y = c + mx
3431 = c + 3,139 x
3431 = c + 391,5594
c = 3431 - 391,5594
c = 3039,4406
```

Linea di regressione completa:

```
y = 3039,4406 + 0,3186 x
```

Infine, possiamo tracciare la linea: generiamo alcuni valori per i carati fra il valore minimo e il valore massimo, li inseriamo nell'equazione che rappresenta la linea di regressione e quindi la tracciamo (Figura 8.13):

```
Minimo di x (Carati) = 300
Massimo di x (Carati) = 2210
```

Campione fra il minimo e il massimo, a intervalli di 500

```
x = [300, 2210]
```

Inseriamo i valori di *x* nella linea di regressione:

```
y = [-426 + 3,139(300) = 515,7,
   -426 + 3,139(2210) = 6511,19]
```

Campioni completi

```
x = [300, 2210]
y = [3981, 9975]
```

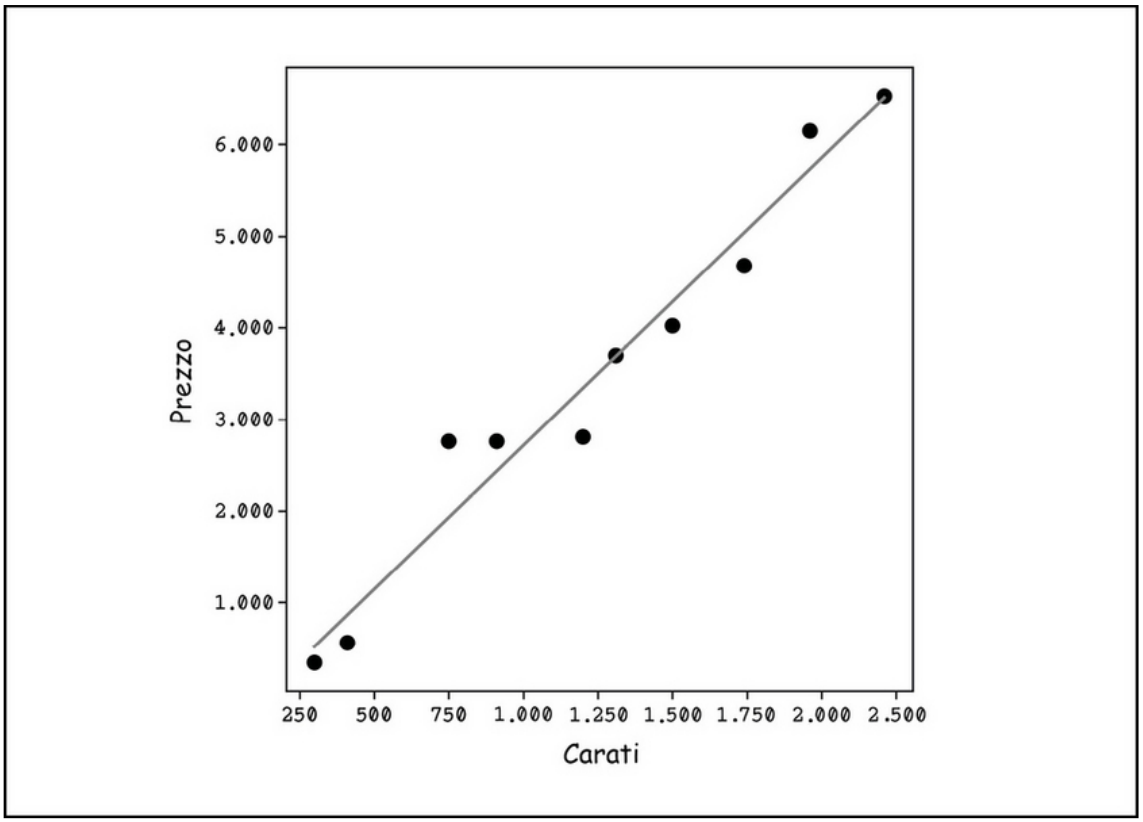


Figura 8.13 Una linea di regressione tracciata con i punti di dati.

Abbiamo addestrato una linea di regressione lineare basata sul nostro dataset che individua accuratamente i dati, quindi abbiamo eseguito manualmente un po' di "machine learning".

Esercizio: calcolare una retta di regressione utilizzando il metodo dei minimi quadrati

Seguendo i passaggi descritti e utilizzando il seguente dataset, calcolate la retta di regressione con il metodo dei minimi quadrati.

	Carati (x)	Prezzo (y)
1	320	350

2	460	560
3	800	2760
4	910	2800
5	1350	2900
6	1390	3600
7	1650	4000
8	1700	4650
9	1950	6100
10	2000	6500

Soluzione

È necessario calcolare le medie per entrambe le dimensioni. Le medie sono 1253 per x e 3422 per y. Il passo successivo consiste nel calcolare la differenza fra ciascun valore e la sua media. Successivamente, viene calcolato e sommato il quadrato della differenza fra x e la media di x, ovvero 3.251.610. Infine, la differenza fra x e la media di x viene moltiplicata per la differenza fra y e la media di y e sommata, ottenendo 10.566.940.

	Carati (x)	**Prezzo (y)**	**x – media di x**	**y – media di y**	**(x – media di x)^2**	**(x – media di x) * (y – media di y)**
1	320	350	-933	-3072	870.489	2.866.176
2	460	560	-793	-2862	628.849	2.269.566
3	800	2760	-453	-662	205.209	299.886
4	910	2800	-343	-622	117.649	213.346
5	1350	2900	97	-522	9409	-50.634
6	1390	3600	137	178	18.769	24.386
7	1650	4000	397	578	157.609	229.466
8	1700	4650	447	1228	199.809	548.916
9	1950	6100	697	2678	485.809	1.866.566
10	2000	6500	747	3078	558.009	2.299.266
	1.253	3.422			3.251.610	10.566.940

I valori possono essere utilizzati per calcolare la pendenza, *m*:

```
m = 10566940 / 3251610
m = 3,25
```

Ricordate l'equazione della linea:

```
y = c + mx
```

Sostituite i valori medi per *x*, *y* e *m*, appena calcolato:

```
3422 = c + 3,35 * 1253
c = -775,55
```

Sostituite i valori minimo e massimo per *x* per calcolare i punti per tracciare una linea:

Punto 1, usiamo il valore minimo per i Carati: `x = 320`

```
y = 775,55 + 3,25 * 320
y = 1 815,55
```

Punto 2, usiamo il valore massimo per i Carati: `x = 2000`

```
y = 775,55 + 3,25 * 2000
y = 7 275,55
```

Ora che sappiamo come utilizzare la regressione lineare e come vengono calcolate le linee di regressione, esaminiamo lo pseudocodice.

Pseudocodice

Il codice è simile ai passaggi che abbiamo seguito. Gli unici aspetti interessanti sono i due cicli `for` utilizzati per calcolare i valori sommati iterando su ogni elemento contenuto nel dataset:

```
fit_regression_line(carats, prices):
  let mean_X equal mean(carats)
  let mean_Y equal mean(price)
  let sum_x_squared equal 0
  for i in range(n):
      let ans equal(carats[i] - mean_X) ** 2
      sum_x_squared equal sum_x_squared + ans
  let sum_multiple equal 0
  for i in range(n):
      let ans equal(carats[i] - mean_X) *(price([i] - mean_Y)
      sum_multiple equal sum_multiple + ans
  let b1 equal sum_multiple / sum_x_squared
  let b0 equal mean_Y - (b1 * mean_X)
  let min_x equal min(carats)
  let max_x equal max(carats)
  let y1 equal b0 + b1 * min_x (1)
  let y2 equal b0 + b1 * max_x (2)
```

(1) Esprime il primo punto della linea di regressione tramite $y = c + mx$

(2) Esprime il secondo punto della linea di regressione tramite $y = c + mx$

Test del modello per determinarne l'accuratezza

Ora che abbiamo determinato una linea di regressione, possiamo usarla per fare previsioni dei prezzi per altri valori di Carati. Possiamo valutare le prestazioni della linea di regressione con nuovi esempi di cui conosciamo il prezzo, e in tal modo determinare l'accuratezza del modello a regressione lineare.

Non possiamo sottoporre a test il modello con gli stessi dati che abbiamo usato per addestrarlo. Questo approccio darebbe sempre un'elevata precisione e sarebbe privo di significato. Il modello appena addestrato deve essere sottoposto a test con dati reali che non ha mai visto.

Separare i dati di addestramento e di test

I dati di addestramento e di test vengono generalmente suddivisi in questo modo: l'80% dei dati disponibili viene utilizzato come dati di addestramento; il 20% viene utilizzato per sottoporre a test il modello. Vengono utilizzate queste percentuali perché è difficile conoscere il numero di esempi necessari per addestrare un modello in modo accurato; contesti e domande differenti possono richiedere più o meno dati.

La Figura 8.14 e la Tabella 8.12 rappresentano un insieme di dati di test per l'esempio dei diamanti. Ricordate che abbiamo ridimensionato i valori dei carati in modo che i numeri avessero dimensioni simili ai valori dei prezzi (tutti i valori di Carati sono stati moltiplicati per 1.000) e questo per renderli più facili da leggere e manipolare. I punti rappresentano i dati di test e la linea rappresenta la linea di regressione addestrata.

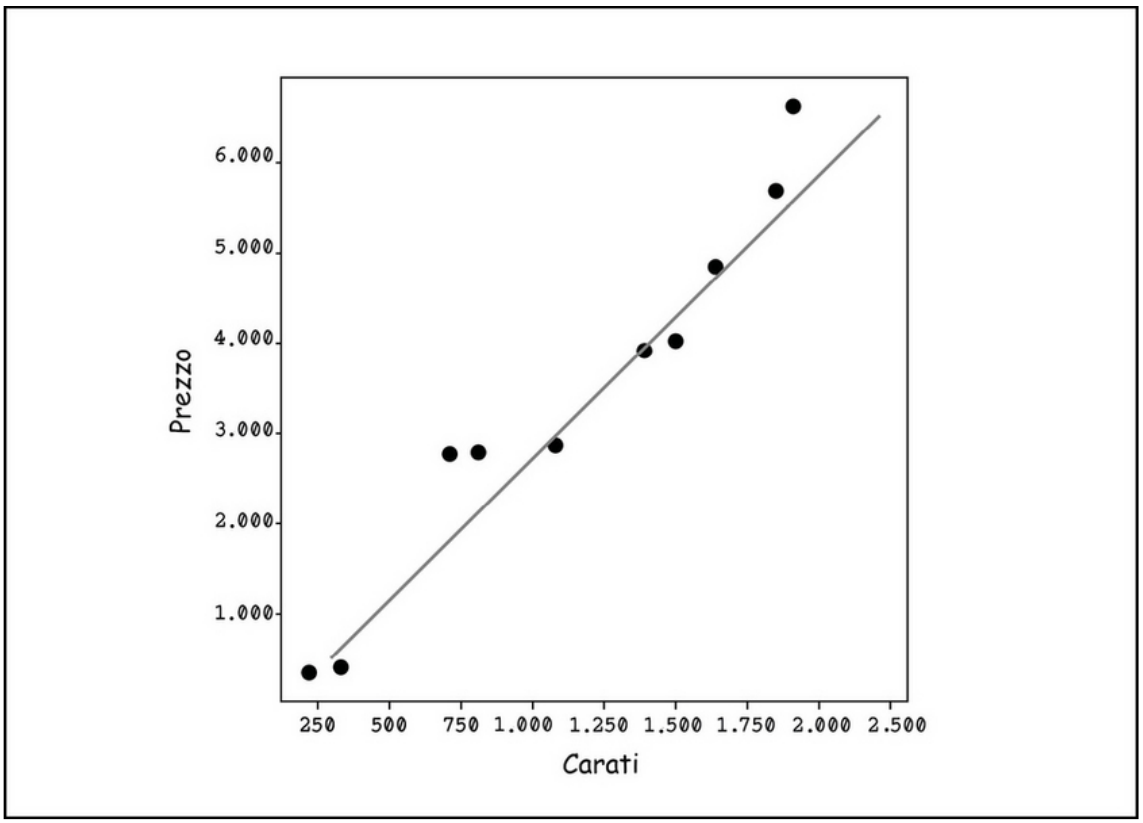


Figura 8.14 Una linea di regressione tracciata con i punti di dati.

Tabella 8.12 I dati dei carati e dei prezzi.

	Carati (x)	**Prezzo (y)**
1	220	342
2	330	403
3	710	2772
4	810	2789
5	1080	2869
6	1390	3914
7	1500	4022
8	1640	4849

9	1850	5688
10	1910	6632

Il test di un modello comporta l'esecuzione di previsioni con dati mai visti dall'algoritmo e la valutazione dell'accuratezza dei valori previsti dal modello rispetto ai valori effettivi. Nell'esempio dei diamanti, abbiamo i valori effettivi dei prezzi, quindi vogliamo vedere che cosa prevede il modello e confronteremo le differenze.

Misurare le prestazioni della linea

Nella regressione lineare, un metodo comune per misurare l'accuratezza del modello è il calcolo di R2 (R quado). R2 viene utilizzato per determinare la varianza fra il valore effettivo e il valore previsto. Per calcolare il punteggio R2 viene utilizzata la seguente equazione:

$$R^2 = \frac{\text{somma di (y previsto - media degli y effettivi)}^2}{\text{somma di (y effettivo - media degli y effettivi)}^2}$$

Le prime cose che dobbiamo fare, come abbiamo visto anche nella fase di addestramento, sono: calcolare la media dei valori effettivi dei prezzi; calcolare le distanze fra i valori effettivi dei prezzi e la media dei prezzi; quindi calcolare il quadrato di tali valori. Nella Figura 8.14 stiamo usando i valori tracciati come punti della Tabella 8.13.

Tabella 8.13 Il dataset dei diamanti e i relativi calcoli.

	Carati (x)	**Prezzo (y)**	**y – media di y**	**(y – media di y)^2**
1	220	342	-3086	9.523.396
2	330	403	-3025	9.150.625

3	710	2772	-656	430.336
4	810	2789	-639	408.321
5	1080	2869	-559	312.481
6	1390	3914	486	236.196
7	1500	4022	594	352.836
8	1640	4849	1421	2.019.241
9	1850	5688	2260	5.107.600
10	1910	6632	3204	10.265.616
		3428		37.806.648
		Media	Somma	

Il passo successivo consiste nel calcolare il valore del prezzo previsto per ogni valore in carati, elevando al quadrato i valori e calcolando la somma di tutti questi valori (Tabella 8.14).

Tabella 8.14 Il dataset dei diamanti e i relativi calcoli, Parte 2.

	Carati (x)	**Prezzo (y)**	**y – media di y**	**(y – media di y)^2**	**y previsto**	**y previsto – media di y**	**(y previsto – media di y)^2**
1	220	342	-3.086	9.523.396	264	-3164	10.009.876
2	330	403	-3.025	9.150.625	609	-2819	7.944.471
3	710	2772	-656	430.336	1802	-1626	2.643.645
4	810	2789	-639	408.321	2116	-1312	1.721.527
5	1080	2869	-559	312.481	2963	-465	215.900
6	1390	3914	486	236.196	3936	508	258.382
7	1500	4022	594	352.836	4282	854	728.562

8	1640	4849	1421	2.019.241	4721	1293	1.671.748
9	1850	5688	2260	5.107.600	5380	1952	3.810.559
10	1910	6632	3204	10.265.616	5568	2.140	4.581.230
		3428		3.7806.648		33.585.901	
		Media		Somma		Somma	

Utilizzando da un lato la somma dei quadrati della differenza fra il prezzo previsto e la media, e dall'altro la somma dei quadrati della differenza fra il prezzo effettivo e la media, possiamo calcolare il punteggio R2:

$$R^2 = \frac{\text{somma di (y previsto - media degli y effettivi)}^2}{\text{somma di (y effettivo - media degli y effettivi)}^2}$$

$$R^2 = 33585901 / 37806648$$

$$R^2 = 0,88$$

Il risultato, 0,88, ci dice che il modello è accurato all'88% rispetto ai nuovi dati mai visti in precedenza. Questo risultato è abbastanza buono, e sta a dimostrare che il modello a regressione lineare è abbastanza accurato. Per l'esempio dei diamanti, questo risultato è soddisfacente. La valutazione dell'accuratezza per il problema che stiamo cercando di risolvere dipende dal dominio del problema stesso. Esploreremo le prestazioni dei modelli di machine learning nel prossimo paragrafo.

La regressione lineare può essere applicata anche a più dimensioni. Possiamo determinare la relazione fra valori in carati, prezzi e taglio dei diamanti, per esempio, attraverso un processo chiamato *regressione multipla*. Questo processo aggiunge una certa complessità ai calcoli, ma i principi fondamentali rimangono gli stessi.

Miglioramento della precisione

Dopo aver addestrato un modello sui dati e aver misurato le sue prestazioni su nuovi dati di test, ci siamo fatti un'idea delle prestazioni del modello. Spesso, i modelli non funzionano come desiderato e, se possibile, è necessario eseguire ulteriore lavoro di raffinamento del modello. Questo miglioramento comporta l'iterazione delle varie fasi del ciclo di vita del machine learning (Figura 8.15).

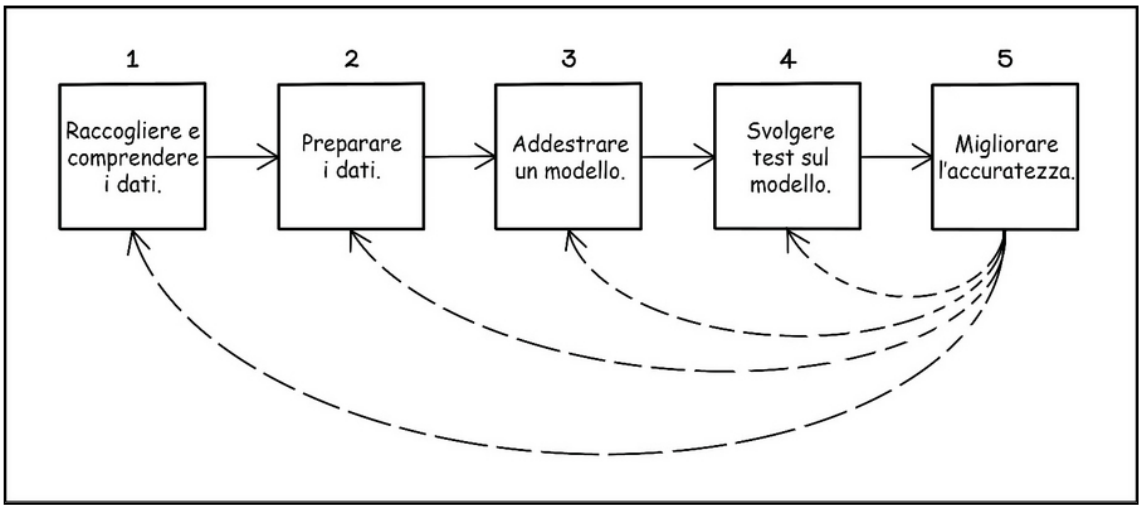


Figura 8.15 Il ciclo di vita del machine learning.

I risultati potrebbero richiedere di prestare attenzione a una o più delle seguenti aree. Il machine learning è un lavoro di sperimentazione, in cui vengono sottoposte a test diverse tattiche in diverse fasi, prima di poter stabilire l'approccio più efficace. Nell'esempio dei diamanti, se il modello che ha utilizzato i valori in carati per prevedere il prezzo ha funzionato male, potremmo utilizzare le caratteristiche dimensionali del diamante, insieme al valore in carati, per provare a prevedere il prezzo in modo più accurato. Ecco alcuni modi per migliorare la precisione del modello.

- *Raccogliere più dati*. Una soluzione potrebbe essere quella di raccogliere più dati relativi al dataset che si sta esplorando, magari

integrandoli con dati esterni pertinenti o includendo dati che in precedenza non erano stati considerati.

- *Preparare i dati in modo differente*. I dati utilizzati per l'addestramento potrebbero dover essere preparati in modo differente. Facendo riferimento alle tecniche utilizzate in precedenza in questo capitolo per correggere i dati, potrebbero esserci errori nel nostro approccio. Potrebbe essere necessario utilizzare tecniche differenti per trovare i valori per i dati mancanti, per sostituire i dati ambigui e per codificare i dati categorici.
- *Scegliere caratteristiche differenti nei dati*. Altre caratteristiche presenti nel dataset potrebbero essere più adatte alla previsione della variabile dipendente. Il valore della dimensione X potrebbe essere una buona scelta per prevedere il valore della tabella, per esempio, perché ha una relazione fisica con esso, come mostrato nella figura della terminologia del diamante (Figura 8.5), mentre la previsione della purezza in base alla dimensione X non ha alcun senso.
- *Utilizzare un algoritmo differente per l'addestramento del modello*. A volte, l'algoritmo selezionato non è adatto al problema da risolvere o alla natura dei dati. Possiamo utilizzare un algoritmo differente per raggiungere obiettivi differenti, come discusso nel prossimo paragrafo.
- *Gestione dei falsi positivi*. I test possono essere ingannevoli. Un buon punteggio del test può mostrare che il modello si comporta molto bene, ma quando al modello vengono sottoposti dati che non ha mai visto, potrebbe funzionare male. Questo problema può essere dovuto a un "overfitting" dei dati. L'*overfitting* si verifica quando il modello è *troppo allineato* con i dati di addestramento e non individua correttamente i nuovi dati, dotati di maggiore varianza. Questo approccio è generalmente applicabile ai problemi di classificazione, che approfondiremo nel prossimo paragrafo.

Se la regressione lineare non ha fornito risultati soddisfacenti o se dobbiamo porre una domanda differente, possiamo provare vari altri algoritmi. I prossimi due paragrafi esploreranno gli algoritmi da utilizzare quando cambia la natura della domanda.

Classificazione con alberi decisionali

In breve, i problemi di classificazione comportano l'assegnazione di un'etichetta a un esempio, in base ai suoi attributi. Questi problemi sono diversi da una regressione, perché non viene stimato un valore. Analizziamo i problemi di classificazione e vediamo come si risolvono.

Problemi di classificazione: o questo o quello

Come abbiamo visto, la regressione comporta la previsione di un valore basato su una o più altre variabili, per esempio la previsione del prezzo di un diamante dato il suo valore in carati. La classificazione è simile, in quanto ha lo scopo di prevedere un valore, ma entro classi discrete e non valori continui. I valori discreti sono caratteristiche categoriche di un dataset, come Taglio, Colore o Purezza nel dataset dei diamanti, al contrario dei valori continui, come Prezzo o Profondità.

Ecco un altro esempio. Supponiamo di avere diversi veicoli, che sono automobili e camion. Misureremo il peso di ciascun veicolo e il numero di ruote. Per ora facciamo finta di dimenticarci che auto e camion hanno un aspetto diverso. Quasi tutte le auto hanno quattro ruote; molti camion hanno più di quattro ruote. I camion sono generalmente più pesanti delle automobili, ma un grande SUV può pesare quanto un piccolo camion. Potremmo ricercare relazioni fra il peso e il numero di ruote dei veicoli per prevedere se un veicolo è un'auto o un camion (Figura 8.16).

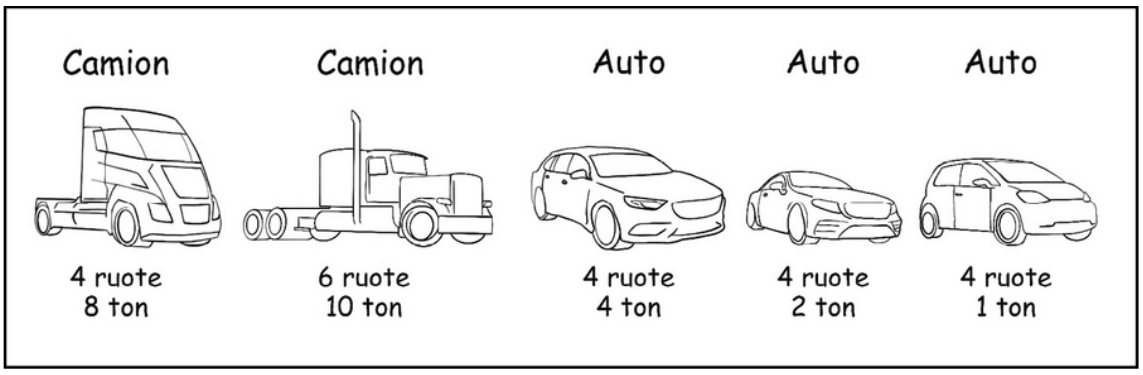


Figura 8.16 Esempi di veicoli per una potenziale classificazione in base al numero di ruote e al peso.

Esercizio: regressione vs classificazione

Considerate i seguenti scenari e determinate se si tratta di un problema di regressione o di classificazione.

1. Sulla base dei dati sulle cavie da laboratorio, abbiamo una caratteristica Aspettativa di vita e una caratteristica Obesità. Stiamo cercando di trovare una correlazione fra le due caratteristiche.
2. Sulla base dei dati sugli animali, abbiamo il Peso e la presenza di Ali. Stiamo cercando di determinare quali animali sono uccelli.
3. Sulla base dei dati sui dispositivi informatici, abbiamo le Dimensioni dello schermo, il Peso e il Sistema operativo. Vogliamo determinare quali dispositivi sono tablet, laptop o telefoni.
4. Sulla base dei dati meteorologici, abbiamo valori di Precipitazioni e di Umidità. Vogliamo determinare l'umidità nelle diverse stagioni delle piogge.

Soluzione

1. *Regressione*: viene esplorata la relazione fra due variabili. L'aspettativa di vita è la variabile dipendente e l'obesità è la variabile indipendente.
2. *Classificazione*: stiamo classificando un esemplare come uccello o non uccello, utilizzando il peso e la presenza di ali.
3. *Classificazione*: un apparecchio viene classificato come tablet, laptop o telefono utilizzando le sue altre caratteristiche.
4. *Regressione*: viene esplorata la relazione fra precipitazioni e umidità. L'umidità è la variabile dipendente e la pioggia è la variabile indipendente.

Basi degli alberi decisionali

Per risolvere problemi di regressione e classificazione vengono utilizzati algoritmi differenti. Fra gli algoritmi più utilizzati vi sono le macchine a vettori di supporto, gli alberi decisionali e le foreste casuali. In questo paragrafo, esamineremo un algoritmo ad albero decisionale per risolvere problemi di classificazione.

Gli *alberi decisionali* sono strutture che descrivono una serie di decisioni prese per raggiungere una soluzione di un problema (Figura 8.17). Se stiamo decidendo se indossare o meno dei pantaloncini corti per la giornata, potremmo prendere tutta una serie di decisioni per determinare che cosa fare. Usciamo di giorno, farà freddo? Altrimenti, se usciamo la sera, farà freddo? Potremmo decidere di indossare pantaloncini per una giornata calda, lunghi se fa freddo.

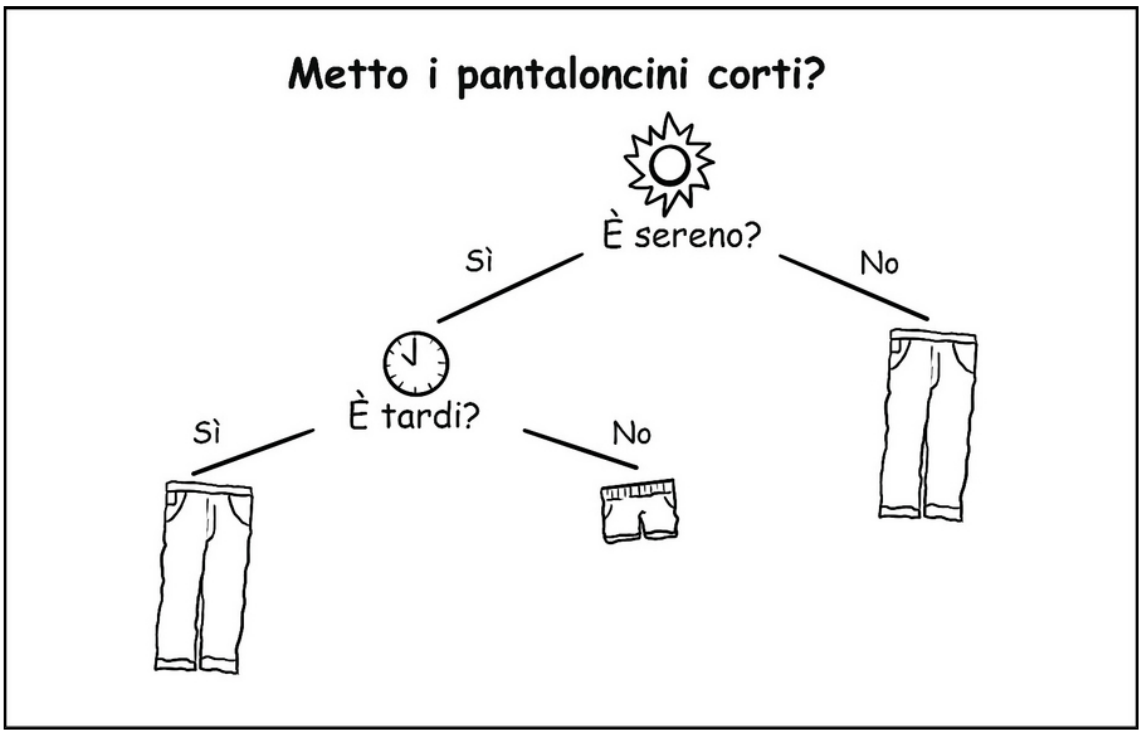


Figura 8.17 Esempio di un semplice albero decisionale.

Per l'esempio dei diamanti, proveremo a prevedere il taglio di un diamante in base ai valori Carati e Prezzo utilizzando un albero

decisionale. Per semplificare questo esempio, siamo commercianti di diamanti e non ci interessa il taglio specifico. Raggrupperemo i diversi tagli in due categorie più ampie. I tagli Fair e Good saranno raggruppati in una categoria chiamata Okay, mentre i tagli Very Good, Premium e Ideal saranno raggruppati in una categoria chiamata Perfect.

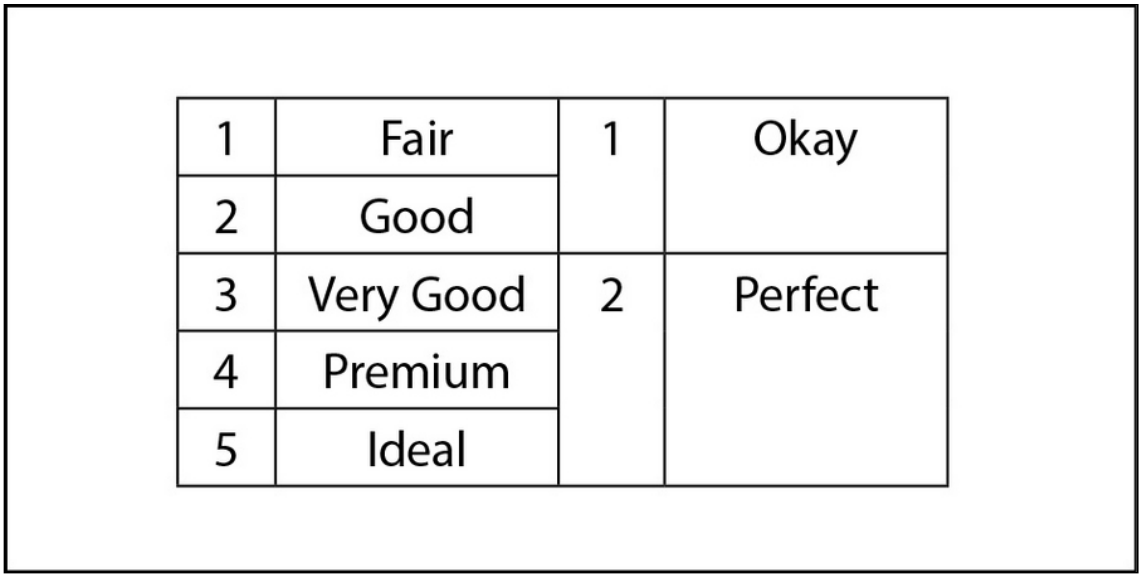


Figura 8.18

Il nostro dataset di esempio ora assomiglia alla Tabella 8.15.

Tabella 8.15 Il dataset utilizzato per l'esempio di classificazione.

	Carati	Prezzo	Taglio
1	0,21	327	Okay
2	0,39	897	Perfect
3	0,50	1122	Perfect
4	0,76	907	Okay
5	0,87	2757	Okay
6	0,98	2865	Okay
7	1,13	3045	Perfect

	Carati	Prezzo	Taglio
8	1,34	3914	Perfect
9	1,67	4849	Perfect
10	1,81	5688	Perfect

Osservando i valori in questo piccolo esempio e cercando intuitivamente i modelli, potremmo notare qualcosa. Il prezzo sembra aumentare in modo significativo dopo 0,98 carati, e l'aumento del prezzo sembra essere correlato ai diamanti perfetti, mentre i diamanti con valori di caratura inferiori tendono a essere medi. Ma l'esempio 3, che è Perfect, ha un piccolo valore in carati. La Figura 8.19 mostra che cosa accadrebbe se dovessimo creare domande per filtrare i dati e categorizzarli manualmente. Notate che i nodi decisionali contengono le nostre domande e i nodi foglia contengono gli esempi che sono stati classificati.

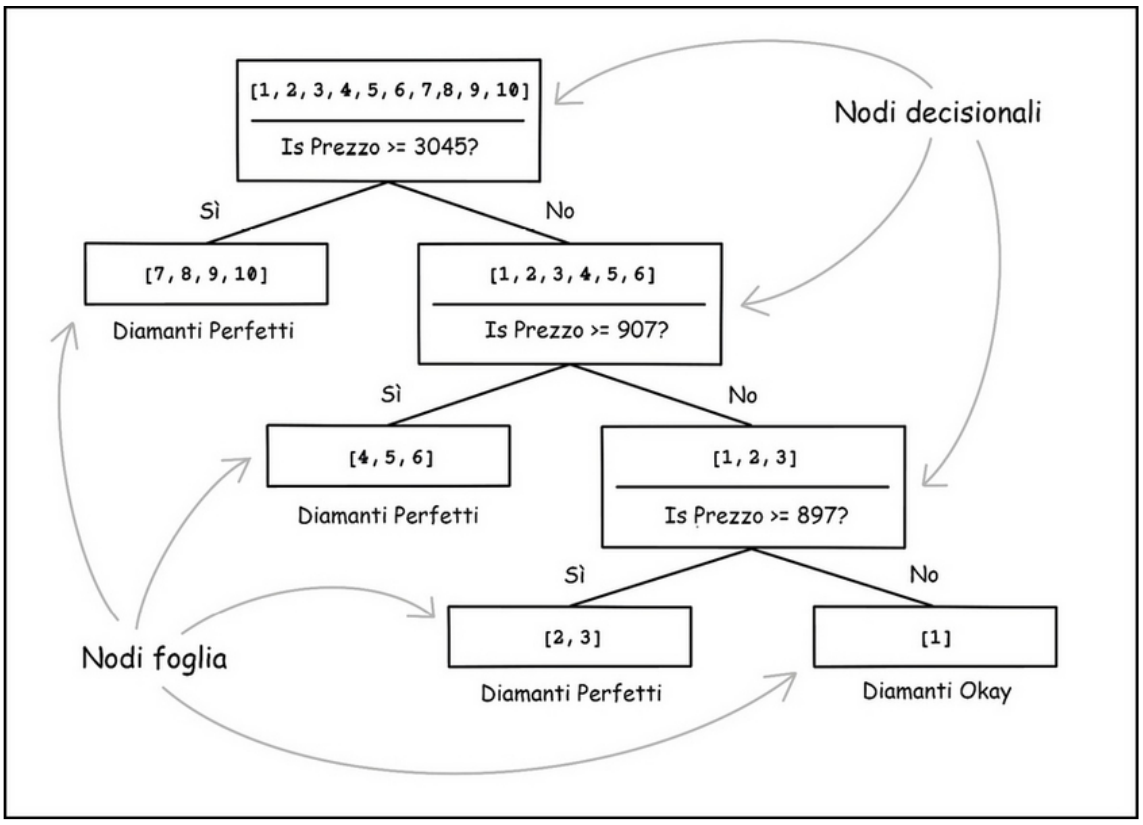


Figura 8.19 Esempio di un albero decisionale progettato attraverso una procedura manuale.

Con questo piccolo dataset, potremmo facilmente classificare i diamanti a mano. Nei dataset reali, al contrario, ci sono migliaia di esempi su cui lavorare, con forse migliaia di caratteristiche, e così per una persona diventa materialmente impossibile creare manualmente un albero decisionale. È qui che entrano in gioco gli algoritmi ad albero decisionale. Gli alberi decisionali possono creare domande che filtrano gli esempi. Un albero decisionale trova pertanto anche dei modelli che noi potremmo perderci, ed è più accurato nel suo filtraggio.

Addestramento di alberi decisionali

Per creare un albero intelligente in grado di prendere decisioni giuste per classificare i diamanti, abbiamo bisogno di un algoritmo di

addestramento che apprenda dai dati. Esiste una famiglia di algoritmi di apprendimento ad albero decisionale e ne useremo uno specifico denominato CART (*Classification and Regression Tree*).
Sostanzialmente, CART e gli altri algoritmi di apprendimento ad albero funzionano così: decidono quali domande porre e quando porre tali domande per filtrare al meglio gli esempi nelle rispettive categorie.
Nell'esempio dei diamanti, l'algoritmo deve apprendere le migliori domande da porre relativamente ai valori Carati e Prezzo e quando chiederle, per segmentare al meglio i diamanti Okay e Perfetti.

Strutture di dati per alberi decisionali

Per capire come saranno strutturate le decisioni dell'albero, possiamo esaminare le seguenti strutture di dati, che organizzano la logica e i dati in un modo adatto all'algoritmo di apprendimento ad albero decisionale.

- *Mappa di classi/raggruppamenti di etichette*: una *mappa* è una coppia di elementi chiave-valore che non può avere due chiavi uguali. Questa struttura è utile per memorizzare il numero di esempi che corrispondono a un'etichetta e sarà utile per memorizzare i valori richiesti per il calcolo dell'entropia, nota anche come *incertezza*. Impareremo presto a conoscerla.
- *Albero dei nodi*: come illustrato nella precedente figura ad albero (Figura 8.18), l'albero è formato da diversi nodi. Questo esempio può ormai esservi familiare dai capitoli precedenti. I nodi dell'albero sono importanti per filtrare/partizionare gli esempi in categorie.
  - *Nodo decisionale*: un nodo nel quale il dataset viene suddiviso o filtrato.
    -- Domanda: quale domanda porre? (Vedi sotto.)
    -- Esempi veri: gli esempi che soddisfano la domanda.
    -- Esempi falsi: gli esempi che non soddisfano la domanda.
  - *Nodo di esempio/nodo foglia*: un nodo contenente solo una lista di

esempi. Tutti gli esempi di questa lista sono classificati correttamente.

- *Domanda*: una domanda può essere rappresentata in modo differente a seconda di quanto può essere flessibile. Potremmo chiedere: “Il valore in carati è > 0,5 e < 1,13?”. Per semplificare la comprensione di questo esempio, la domanda è una caratteristica variabile, un valore variabile e l’operatore >=: “`Is Carati >= 0.5?`” o “`Is Prezzo >=3.045?`”.

  - *Caratteristica*: la caratteristica che viene interrogata.
  - *Valore*: il valore costante di riferimento.

Ciclo di vita dell’apprendimento ad albero decisionale

Ora vedremo come un algoritmo ad albero decisionale filtra i dati tramite decisioni per classificare correttamente un dataset. La Figura 8.20 mostra i passaggi coinvolti nell’addestramento di un albero decisionale. Di seguito descriveremo questo flusso.

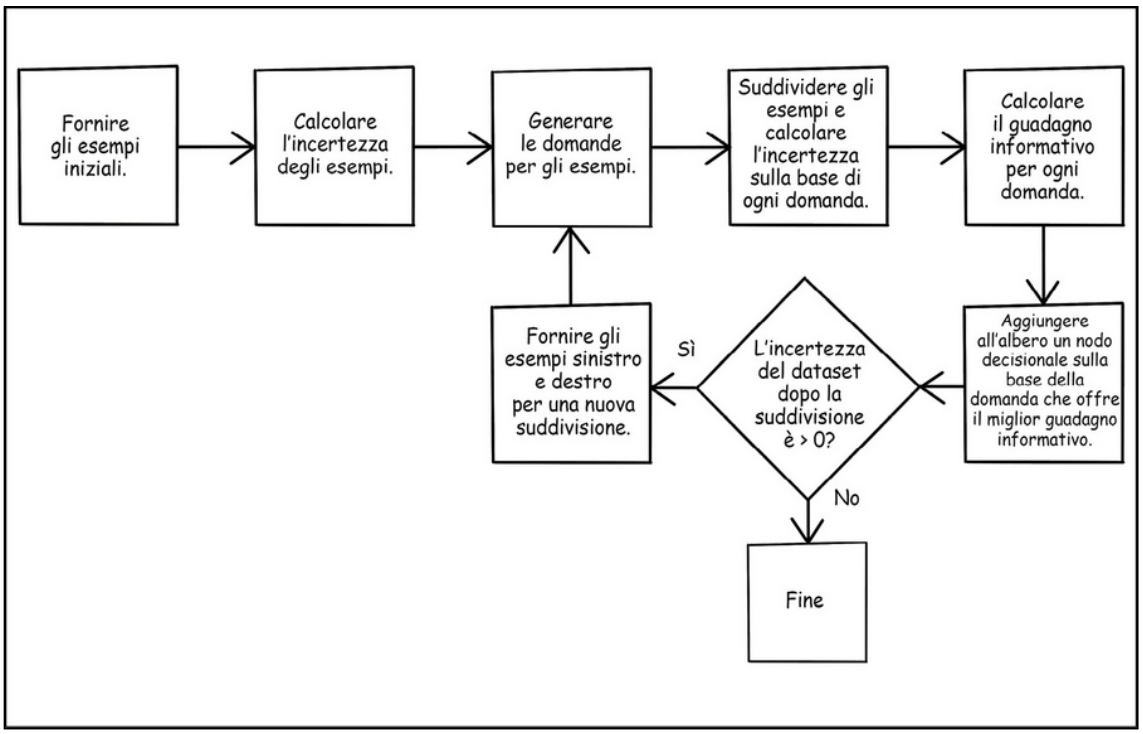


Figura 8.20 Il flusso per costruire un albero decisionale.

Nella costruzione di un albero decisionale, sottoponiamo a test tutte le possibili domande, per determinare quale sia la migliore domanda da porre in un determinato punto dell'albero decisionale. Per sottoporre a test una domanda, usiamo il concetto di *entropia*, che misura l'incertezza di un dataset. Se avessimo 5 diamanti Perfect e 5 diamanti Okay e provassimo a scegliere un diamante Perfect selezionando a caso un diamante fra i 10, quali sono le probabilità che il diamante sia Perfect (Figura 8.21)?

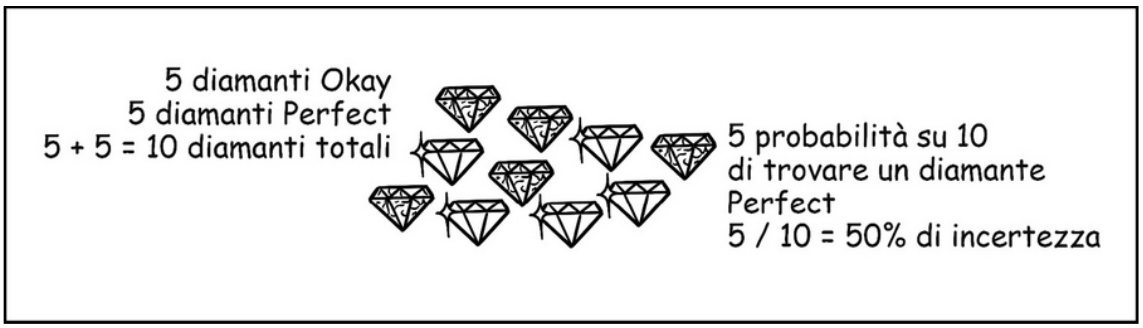


Figura 8.21 Esempio di incertezza.

Dato un dataset iniziale di diamanti con le caratteristiche Carati, Prezzo e Taglio, possiamo determinare l'incertezza del dataset utilizzando l'indice di Gini. Se l'indice di Gini è pari a 0, significa che il dataset non ha incertezza, è puro; potrebbe avere 10 diamanti Perfect, per esempio. La Figura 8.22 descrive il calcolo dell'indice di Gini.

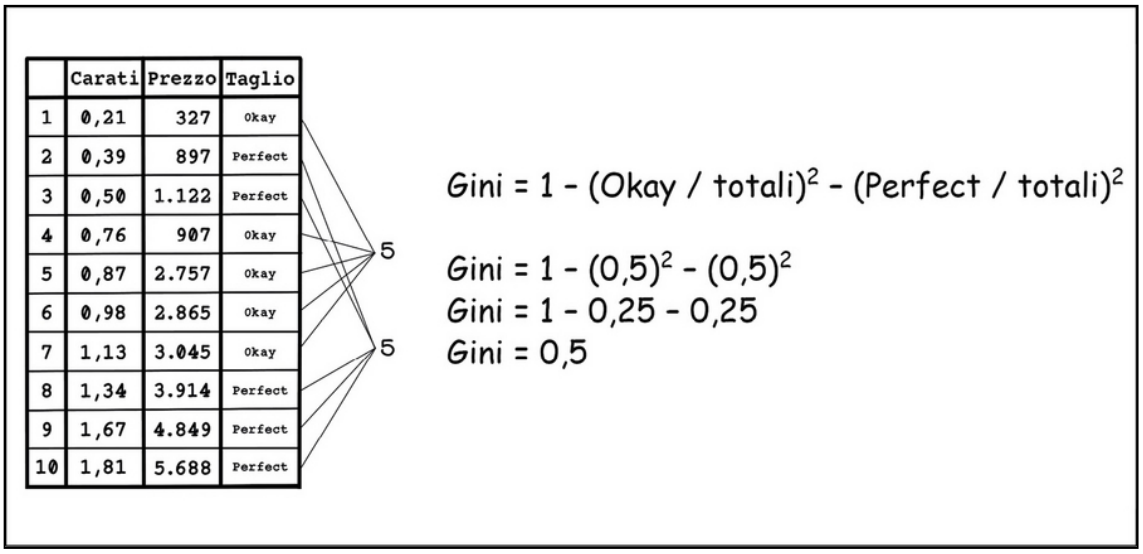


Figura 8.22 Il calcolo dell'indice di Gini.

L'indice di Gini è 0,5, quindi, scegliendo a caso, c'è una probabilità del 50% di scegliere un esempio etichettato in modo errato, come mostra la Figura 8.20 precedente.

Il passo successivo consiste nel creare un nodo decisionale per suddividere i dati. Il nodo decisionale include una domanda che può

essere utilizzata per suddividere i dati in modo tale da ridurre l'incertezza. Ricordate che 0 significa "nessuna incertezza". Puntiamo a suddividere il dataset in sottoinsiemi con incertezza zero.

Vengono generate molte domande in base a ogni caratteristica di ciascun esempio, in modo da suddividere i dati e determinare il miglior risultato di ogni suddivisione. Poiché abbiamo 2 caratteristiche e 10 esempi, il numero totale di domande generate sarebbe 20. La Figura 8.23 illustra alcune delle domande poste: semplici domande sul fatto che una caratteristica sia maggiore o uguale a un determinato valore.

L'incertezza in un dataset è determinata dall'*indice di Gini*, e le domande puntano a ridurre l'incertezza. L'*entropia* è un altro concetto che misura il disordine utilizzando l'indice di Gini per una specifica suddivisione dei dati sulla base di una domanda. Dobbiamo avere un modo per determinare in che misura una domanda ha ridotto l'incertezza e svolgiamo questo compito misurando il guadagno informativo. Il *guadagno informativo* descrive la quantità di informazioni che otteniamo ponendo una determinata domanda. Se otteniamo un grande guadagno informativo, avremo anche una minore incertezza.

Il guadagno informativo viene calcolato sottraendo l'entropia prima di porre la domanda dall'entropia dopo aver posto la domanda, seguendo questi passaggi.

1. Dividi il dataset ponendo la domanda.
2. Misura l'indice di Gini per la divisione a sinistra.
3. Misura l'entropia per la divisione a sinistra rispetto al dataset prima della divisione.
4. Misura l'indice di Gini per la divisione a destra.
5. Misura l'entropia per la divisione a destra rispetto al dataset prima della divisione.
6. Calcola l'entropia totale sommando l'entropia sinistra e l'entropia destra.

7. Calcola il guadagno informativo sottraendo l'entropia totale dopo la domanda dall'entropia totale prima della domanda.

La Figura 8.24 illustra la suddivisione dei dati e il guadagno informativo per la domanda "Is Prezzo >= 3914?".

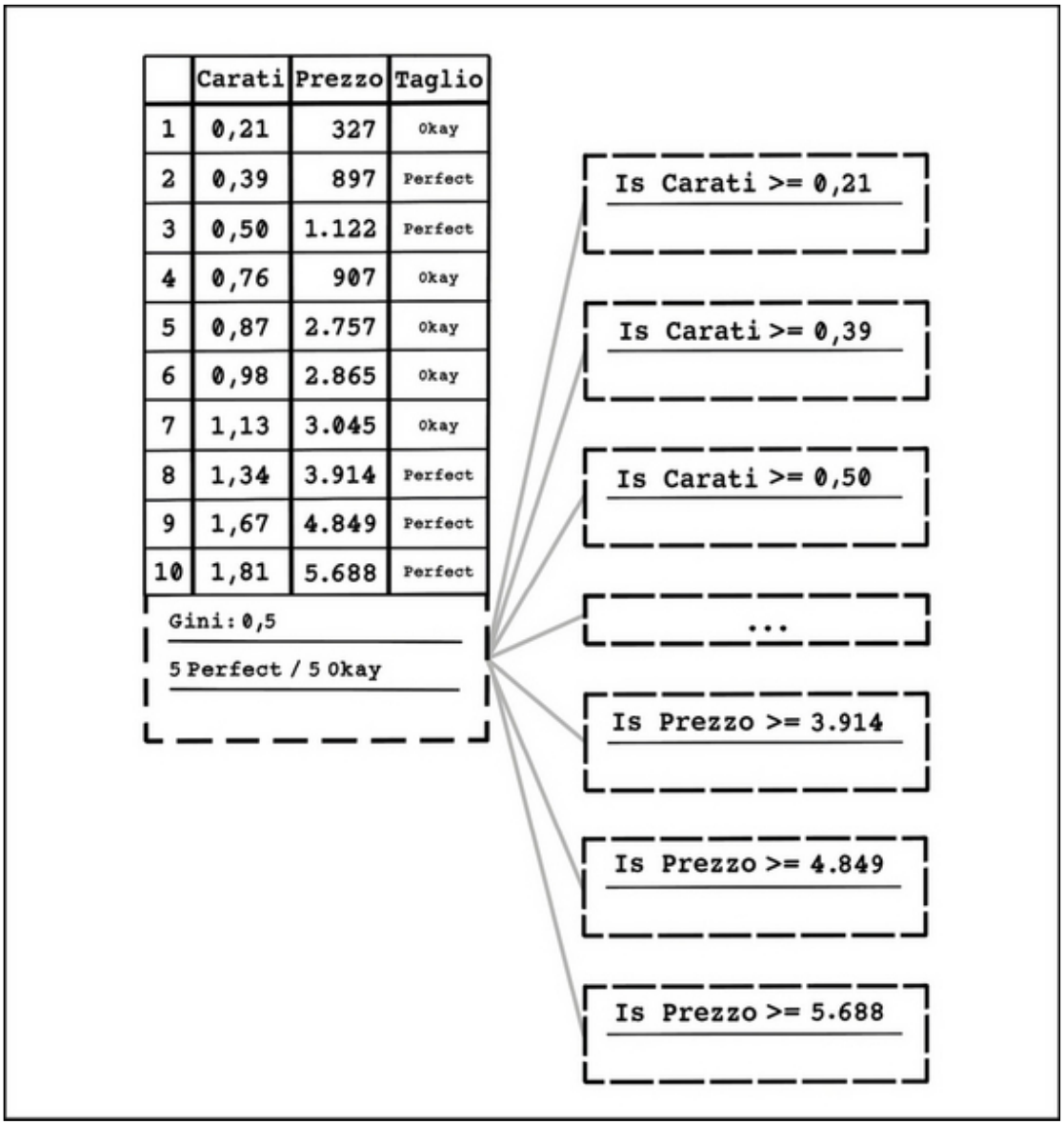


Figura 8.23 Un esempio di domande poste per suddividere i dati con un nodo decisionale.

Nell'esempio della Figura 8.24, viene calcolato il guadagno informativo per tutte le domande; la domanda che offre il maggior

guadagno informativo viene selezionata come migliore domanda da porre in quel punto dell'albero. Quindi il dataset originale viene suddiviso in base al nodo decisionale con la domanda "Is Prezzo >= 3,914?". All'albero decisionale viene aggiunto un nodo decisionale contenente questa domanda, e le divisioni sinistra e destra derivano da quel nodo.

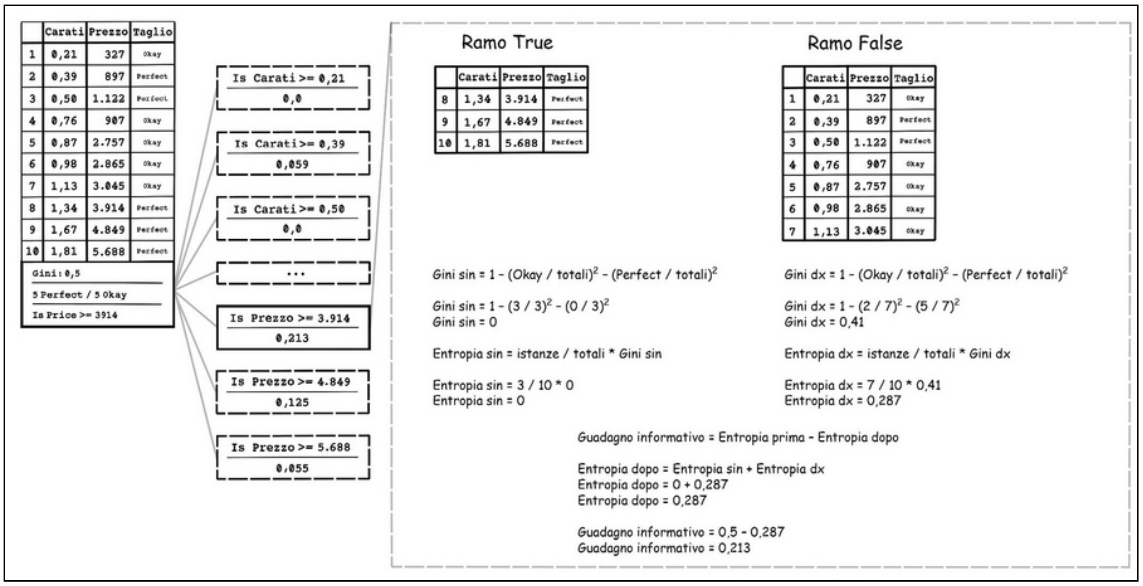


Figura 8.24 Illustrazione della suddivisione dei dati e dell'acquisizione di informazioni sulla base di una domanda.

Nella Figura 8.25, dopo aver diviso il dataset, il lato sinistro contiene un dataset puro di soli diamanti Perfect e il lato destro contiene un dataset con classificazioni miste di diamanti: due diamanti Perfect e cinque diamanti Okay. Deve essere posta un'altra domanda sul lato destro del dataset, per suddividerlo ulteriormente. Ancora una volta, vengono generate diverse domande, utilizzando le caratteristiche di ciascun esempio presente nel dataset.

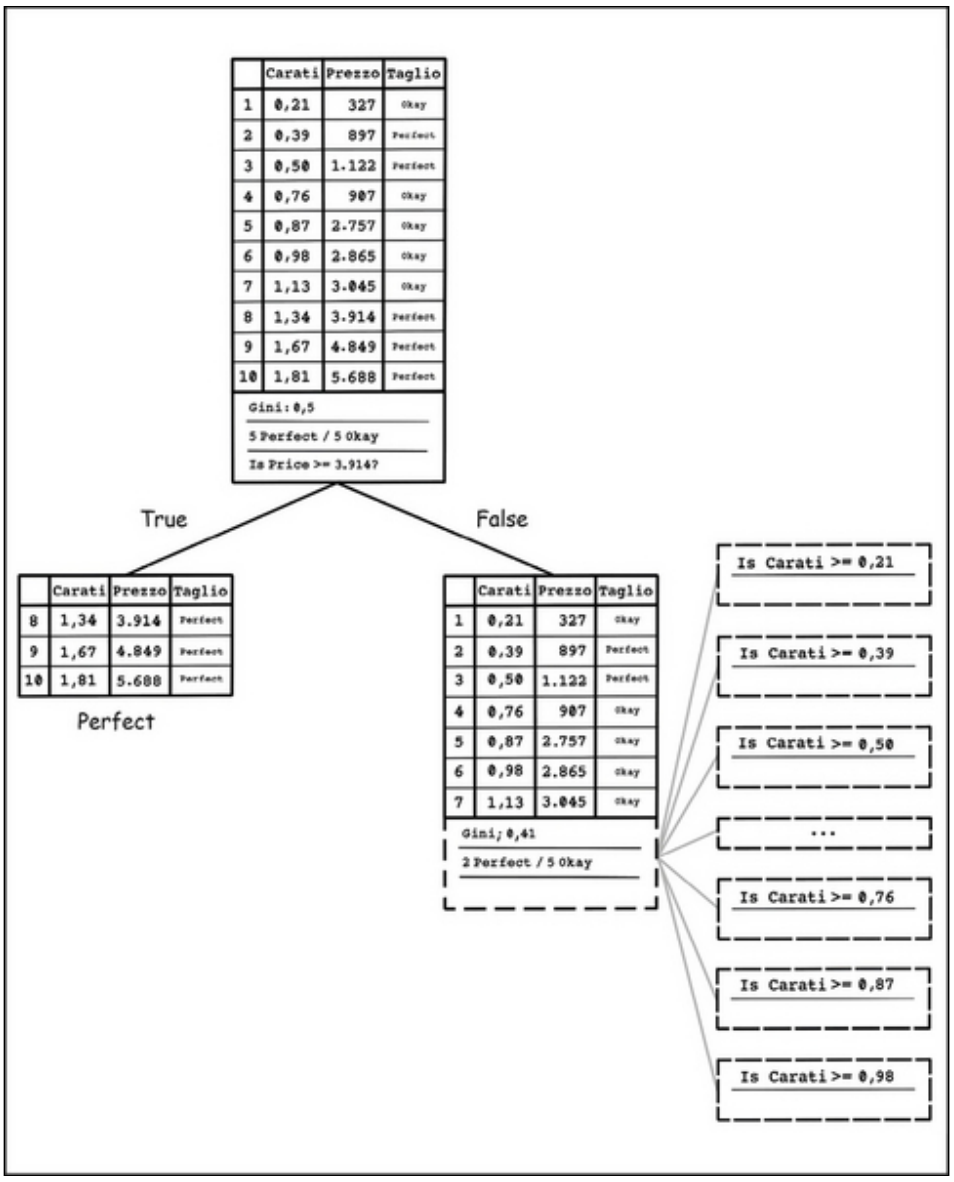
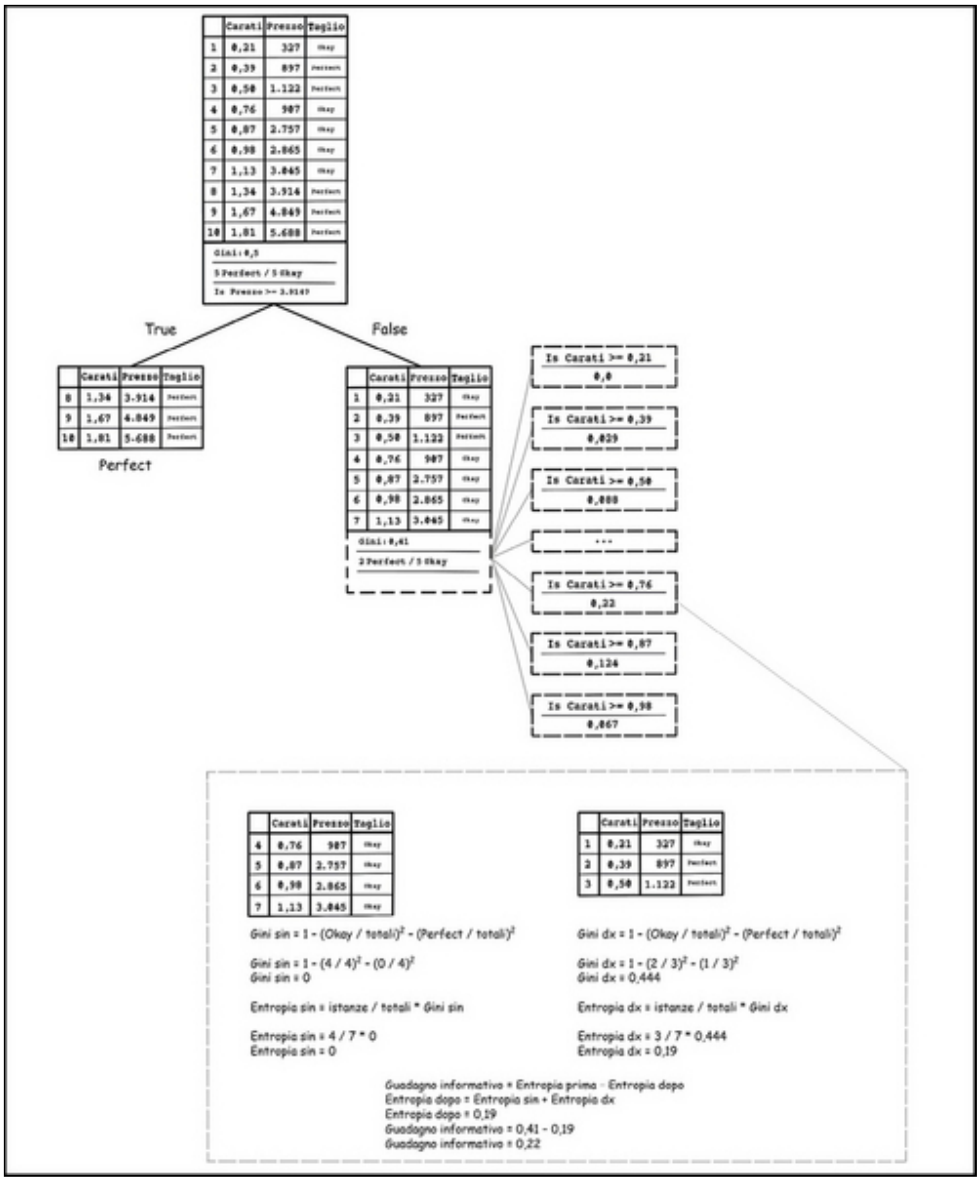


Figura 8.25 L'albero decisionale risultante dopo il primo nodo decisionale e le possibili domande.

Esercizio: calcolare l'incertezza e il guadagno informativo per una domanda
Utilizzando come guida le conoscenze acquisite e la Figura 8.23, calcolare il guadagno informativo per la domanda "Is Carati >= 0,76?".

Soluzione
La soluzione illustrata di seguito evidenzia il riutilizzo della sequenza di calcoli che determinano l'entropia e il guadagno informativo, data una domanda. Provate a

esercitarvi con altre domande e a confrontare i risultati con i valori di guadagno informativo indicati nella figura.



Il processo di suddivisione, generazione di domande e determinazione del guadagno informativo avviene in modo ricorsivo fino a quando il dataset non risulta completamente classificato in base alle domande. La Figura 8.26 mostra l'albero decisionale completo, comprese tutte le domande poste e le conseguenti suddivisioni.

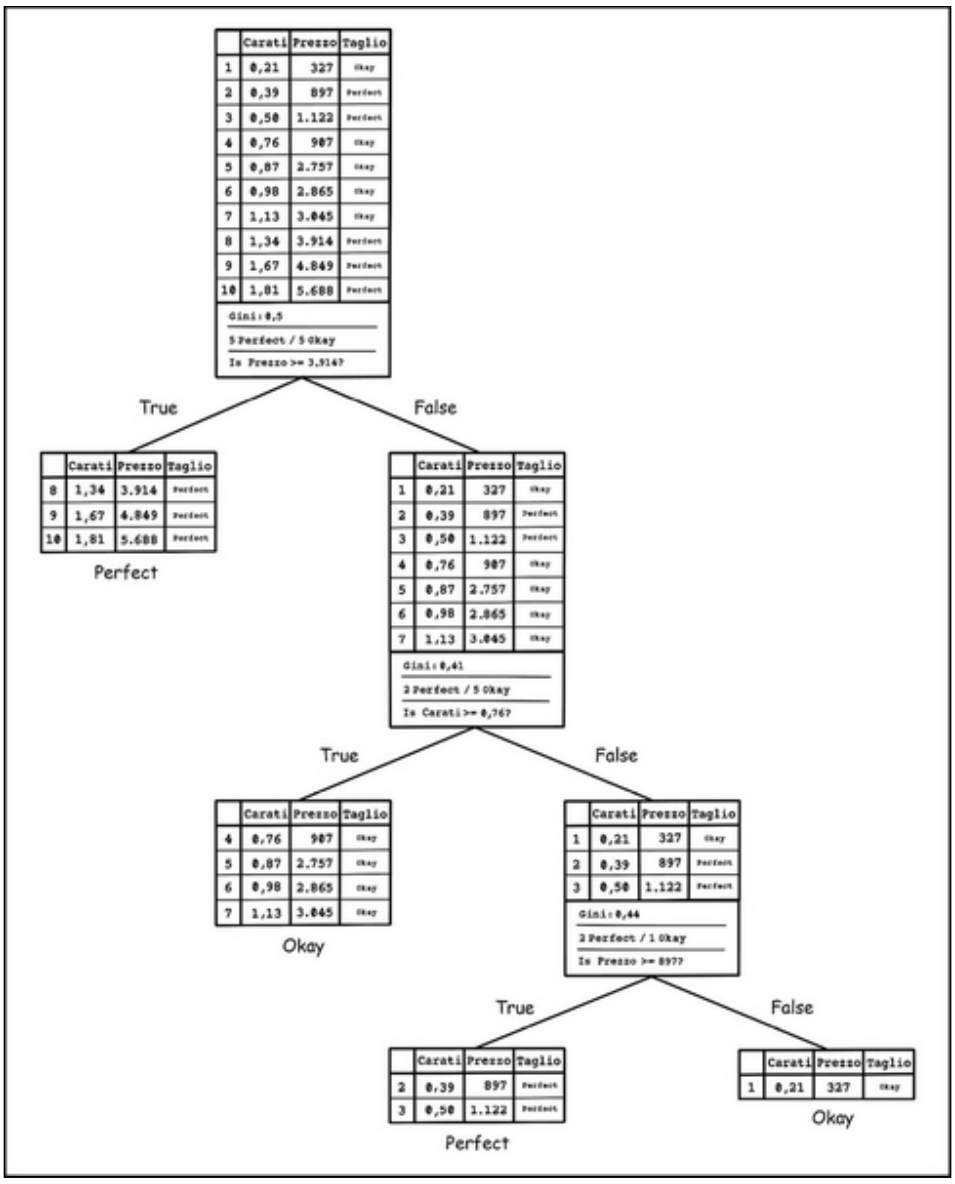


Figura 8.26 L'albero decisionale addestrato e completo.

È importante notare che gli alberi decisionali vengono generalmente addestrati con un campione di dati ben più ampio. Le domande poste devono essere più generali, per considerare una più ampia varietà di dati, quindi, c'è bisogno di una grande varietà di esempi da cui imparare.

Pseudocodice

Quando si programma un albero decisionale da zero, il primo passaggio consiste nel contare il numero di esempi di ciascuna classe, in questo caso il numero di

diamanti Okay e il numero di diamanti Perfect:

```
find_unique_label_counts(examples):
  let class_count equal empty map
  for example in examples:
      let label equal example['quality']
      if label not in class_count:
          let class_count[label] equal 0
      class_count[label] equal class_count[label] + 1
  return class_count
```

Successivamente, gli esempi vengono suddivisi in base a una domanda. Gli esempi che soddisfano la domanda vengono archiviati in `examples_true` e gli altri in `examples_false`:

```
split_examples(examples, question):
  let examples_true equal empty array
  let examples_false equal empty array
  for example in examples:
      if question.filter(example):
          append example to examples_true
      else:
          append example to examples_false
  return examples_true, examples_false
```

Abbiamo bisogno di una funzione che calcoli l'indice di Gini per una serie di esempi. Questa funzione calcola l'indice di Gini utilizzando il metodo descritto nella Figura 8.23:

```
calculate_gini(examples):
  let label_counts equal find_unique_label_counts(examples)
  let uncertainty equal 1
  for label in label_counts:
      let probability_of_label equal label_counts[label] / length(examples)
      let uncertainty equal uncertainty - probability_of_label ^ 2
  return uncertainty
```

La funzione `calculate_information_gain` utilizza le divisioni sinistra e destra e l'incertezza corrente per determinare il guadagno informativo:

```
calculate_information_gain(left, right, current_uncertainty):
  let total equal length(left) + length(right)
  let left_gini equal calculate_gini(left)
  let left_entropy equal length(left) / total * left_gini
  let right_gini equal calculate_gini(right)
  let right_entropy equal length(right) / total * right_gini
  let uncertainty_after equal left_entropy + right_entropy
  let information_gain equal current_uncertainty - uncertainty_after
  return information_gain
```

La prossima funzione può sembrare complessa, ma sta solo iterando su tutte le caratteristiche del dataset e i loro valori, per trovare il miglior guadagno informativo in modo da determinare la migliore domanda da porre:

```
find_best_split(examples, number_of_features):
  let best_gain equal 0
  let best_question equal None
  let current_uncertainty equal calculate_gini(examples)
```

```
  for feature_index in range (number_of_features):
      let values equal [example[feature_index] for example in examples]
      for value in values:
          let question equal Question(feature_index, value)
             let true_examples, false_examples equal split_examples(examples,
question)
          if length(true_examples) != 0 or length (false_examples) != 0:
                    let gain equal calculate_information_gain(true_examples,
false_examples,                    current_uncertainty)
              if gain >= best_gain:
                  best_gain, best_question equal gain, question
  return best_gain, best_question
```

La funzione successiva mette insieme il tutto, utilizzando le funzioni definite in precedenza per costruire un albero decisionale:

```
build_tree(examples, number_of_features):
  let gain, question equal find_best_split(examples, number_of_features)
  if gain == 0:
      return ExamplesNode(examples)
  let true_examples, false_examples equal split_examples(examples, question)
  let true_branch equal build_tree(true_examples)
  let false_branch equal build_tree(false_examples)
  return DecisionNode(question, true_branch, false_branch)
```

Notate che questa funzione è ricorsiva. Suddivide i dati e quindi divide in modo ricorsivo il dataset risultante fino a quando non nota più alcun guadagno informativo, il che indica che gli esempi non possono essere ulteriormente suddivisi. Come promemoria, i nodi decisionali vengono utilizzati per suddividere gli esempi e i nodi di esempio vengono utilizzati per archiviare insiemi di esempi già divisi.

Ora abbiamo imparato come costruire un classificatore ad albero decisionale. Ricordate che il modello dell'albero decisionale addestrato verrà sottoposto a test tramite dati che non ha mai visto in precedenza, in modo simile all'approccio a regressione lineare esplorato in precedenza.

Un problema degli alberi decisionali è l'*overfitting*, che si verifica quando il modello viene addestrato in modo troppo aderente agli esempi che ha esaminato ma si comporta male con i nuovi esempi. L'overfitting si verifica quando il modello apprende i modelli presenti nei dati di addestramento, ma i dati in arrivo dal mondo reale sono leggermente diversi e non soddisfano appieno i criteri di suddivisione del modello addestrato. Un modello con una precisione del 100% è solitamente afflitto da un overfitting sui dati. In un modello ideale, alcuni esempi

saranno classificati in modo errato, in conseguenza del fatto che tale modello è generale e supporta anche altri casi. L'overfitting può verificarsi con qualsiasi modello di machine learning, non solo con gli alberi decisionali.

La Figura 8.27 illustra il concetto di overfitting. L'underfitting prevede *troppe classificazioni errate* e l'overfitting prevede *troppo poche o addirittura nessuna classificazione errata*; l'ideale è una buona via di mezzo.

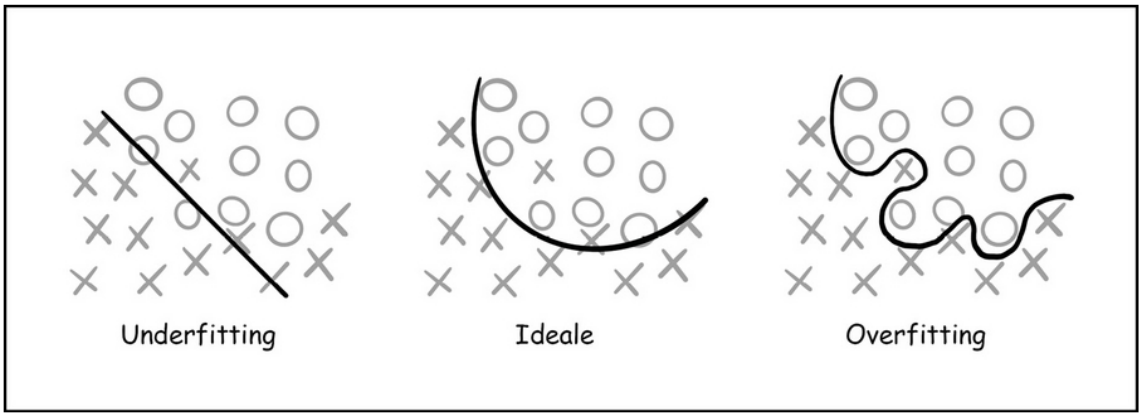


Figura 8.27 Underfitting, ideale e overfitting.

Classificazione degli esempi con alberi decisionali

Ora che abbiamo addestrato un albero decisionale e abbiamo determinato le domande giuste, possiamo sottoporlo a test fornendogli nuovi dati da classificare. Il modello a cui ci riferiamo è l'albero decisionale delle domande che è stato creato dalla fase di addestramento.

Per sottoporre a test il modello, forniamo diversi nuovi esempi di dati e misuriamo se vengono classificati correttamente. Pertanto, abbiamo bisogno di conoscere già l'etichettatura dei dati di test. Nell'esempio dei diamanti, abbiamo bisogno di più dati sul diamante, inclusa la caratteristica Taglio, per sottoporre a test l'albero decisionale (Tabella 8.16).

Tabella 8.16 Il dataset dei diamanti per la classificazione.

	Carati	Prezzo	Taglio
1	0,26	689	Perfect
2	0,41	967	Perfect
3	0,52	1012	Perfect
4	0,76	907	Okay
5	0,81	2650	Okay
6	0,90	2634	Okay
7	1,24	2999	Perfect
8	1,42	3850	Perfect
9	1,61	4345	Perfect
10	1,78	3100	Okay

La Figura 8.28 illustra il modello di albero decisionale che abbiamo addestrato, che verrà utilizzato per elaborare i nuovi esempi. Ogni esempio viene inserito nell'albero e classificato.

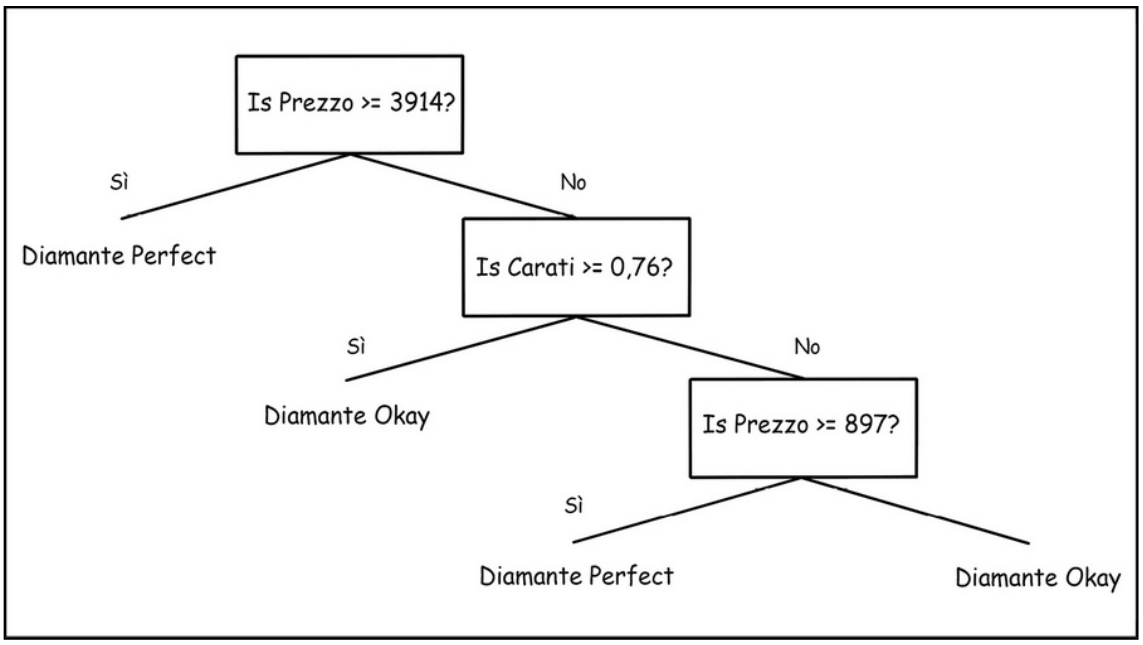


Figura 8.28 Il modello dell'albero decisionale che elaborerà nuovi esempi.

Le risultanti classificazioni previste sono riportate nella Tabella 8.17. Supponiamo che stiamo cercando di prevedere i diamanti Okay. Notate che tre esempi sono errati, 3 su 10, il che significa che il modello ha previsto correttamente 7 esempi su 10 o il 70% dei dati di test. Questa performance non è pessima, ma illustra come gli esempi possano essere classificati erroneamente.

Tabella 8.17 Il dataset dei diamanti per la classificazione e le previsioni.

	Carati	Prezzo	Taglio	Predizione	
1	0,26	689	Okay	Okay	✓
2	0,41	880	Perfect	Perfect	✓
3	0,52	1012	Perfect	Perfect	✓
4	0,76	907	Okay	Okay	✓
5	0,81	2650	Okay	Okay	✓
6	0,90	2634	Okay	Okay	✓

7	1,24	2999	Perfect	Okay	✝
8	1,42	3850	Perfect	Okay	✝
9	1,61	4345	Perfect	Perfect	✓
10	1,78	3100	Okay	Perfect	✝

Per misurare le prestazioni di un modello con i dati di test viene normalmente utilizzata una matrice di confusione. Una *matrice di confusione* descrive le prestazioni utilizzando le seguenti metriche (Figura 8.29).

- *Vero positivo* (*TP*): esempi correttamente classificati come Okay.
- *Vero negativo* (*TN*): esempi correttamente classificati come Perfect.
- *Falso positivo* (*FP*): esempi Perfect classificati come Okay.
- *Falso negativo* (*FN*): esempi Okay classificati come Perfect.

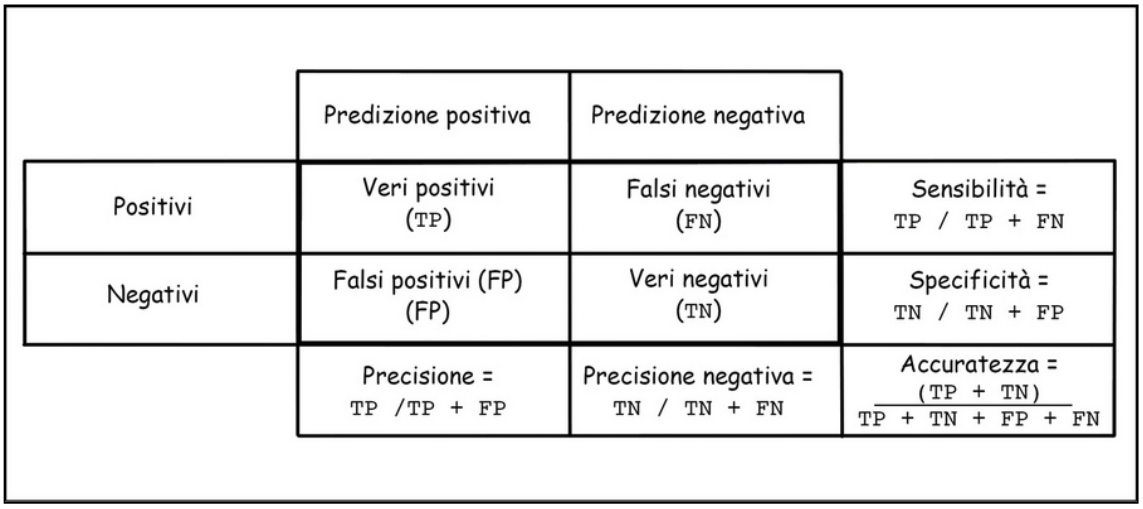


Figura 8.29 Una matrice di confusione.

I risultati del test del modello con esempi che non ha mai visto prima possono essere utilizzati per dedurre diverse misurazioni.

- *Precisione*: la frequenza con cui gli esempi Okay vengono classificati correttamente.

- *Precisione negativa*: la frequenza con cui gli esempi Perfect vengono classificati correttamente.
- *Sensibilità o richiamo*: è il *tasso di veri positivi*, il rapporto fra i diamanti Okay classificati correttamente e tutti i diamanti effettivamente Okay nel set di addestramento.
- *Specificità*: è il *tasso di veri negativi*, il rapporto fra i diamanti Perfect classificati correttamente e tutti i diamanti effettivamente Perfect nel set di addestramento.
- *Precisione*: quanto spesso il classificatore è corretto in generale considerando entrambe le classi.

La Figura 8.30 mostra la matrice di confusione risultante, con i diamanti dell'esempio. La precisione è importante, ma le altre misurazioni possono rivelare ulteriori informazioni utili sulle prestazioni del modello.

Utilizzando queste misurazioni, possiamo prendere decisioni più informate tramite il ciclo di vita di machine learning, per migliorare le prestazioni del modello. Come accennato in tutto il capitolo, il machine learning è un esercizio sperimentale, che procede per tentativi ed errori. Queste metriche ci guideranno in questo processo.

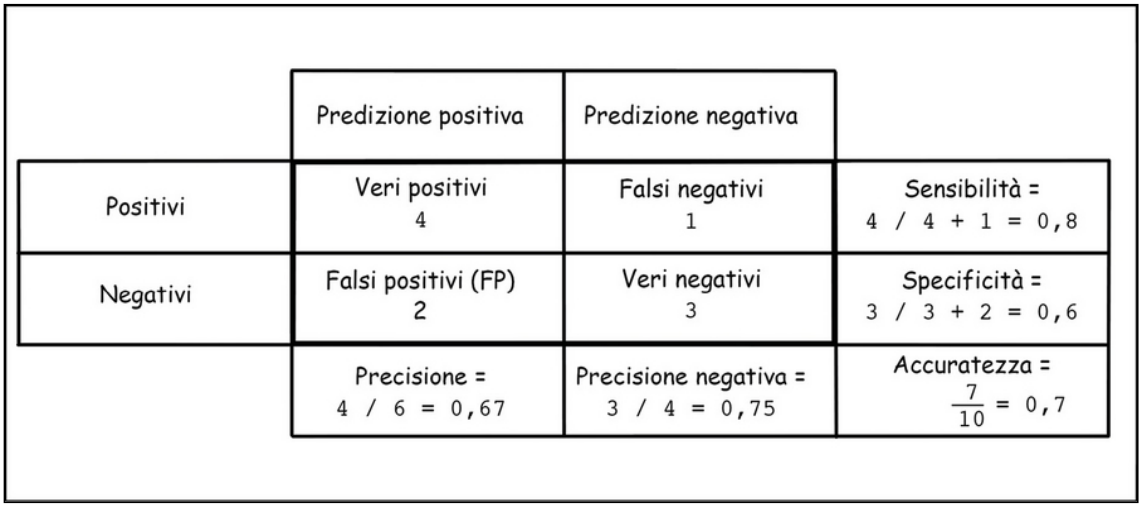


Figura 8.30 Matrice di confusione per i diamanti di questo esempio.

Altri algoritmi di machine learning noti

Questo capitolo ha esplorato due noti e importanti algoritmi di machine learning. L'algoritmo di regressione lineare viene utilizzato per problemi nei quali occorre scoprire le relazioni esistenti fra le caratteristiche. L'algoritmo ad albero decisionale viene utilizzato per problemi di classificazione, nei quali vengono scoperte le relazioni fra caratteristiche e determinate categorie. Ma molti altri algoritmi di machine learning sono adatti a contesti differenti e per risolvere problemi differenti. La Figura 8.31 illustra alcuni degli algoritmi più noti e mostra come si collocano nel panorama del machine learning.

Gli algoritmi di classificazione e regressione riguardano problemi simili a quelli esplorati in questo capitolo. L'apprendimento senza supervisione comprende algoritmi che possono essere utili in alcune fasi di preparazione dei dati o per trovare relazioni nascoste nei dati e informare delle domande che possono essere poste in un esperimento di machine learning.

Notate la presenza del *deep learning* nella Figura 8.31. Il Capitolo 9 tratterà le reti neurali artificiali, un concetto chiave del deep learning. Tale capitolo vi darà maggiori informazioni sui tipi di problemi che possono essere risolti con questi approcci e su come vengono implementati gli algoritmi.

Casi d'uso per gli algoritmi di machine learning

Le tecniche di machine learning possono essere applicate in quasi tutti i settori per risolvere una miriade di problemi in diversi domini. Avendo i giusti dati e le giuste domande, le loro possibilità sono potenzialmente infinite. Abbiamo tutti interagito con un prodotto o un servizio che utilizza tecniche di machine learning e di modellazione dei dati nella

nostra vita quotidiana. Questo paragrafo evidenzia alcuni dei modi più diffusi nei quali il machine learning può essere utilizzato per risolvere problemi concreti su larga scala.

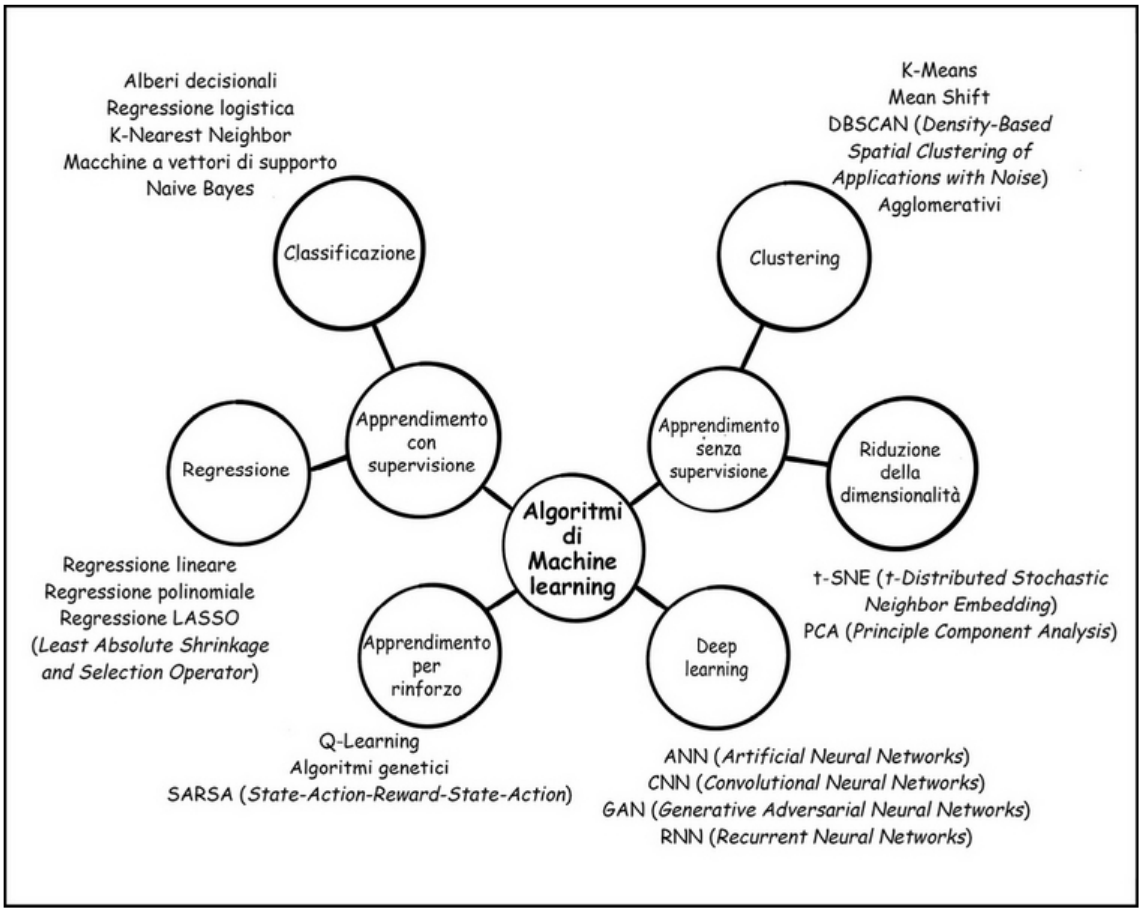


Figura 8.31 Una mappa degli algoritmi di machine learning più utilizzati.

- *Rilevamento di frodi e minacce*: il machine learning è utilizzato per rilevare e prevenire transazioni fraudolente nel settore finanziario. Nel corso degli anni, le istituzioni finanziarie hanno acquisito una grande quantità di informazioni sulle transazioni, inclusi i rapporti sulle transazioni fraudolente dei loro clienti. Queste segnalazioni sono un input prezioso per etichettare e catalogare le transazioni fraudolente. I modelli possono considerare l'ubicazione della transazione, l'importo, il commerciante e così via per classificare le

transazioni, risparmiando ai consumatori le potenziali perdite e all'istituto finanziario le spese assicurative. Lo stesso modello può essere applicato al rilevamento delle minacce di rete, per rilevare e prevenire gli attacchi sulla base di tecniche note e comportamenti insoliti.

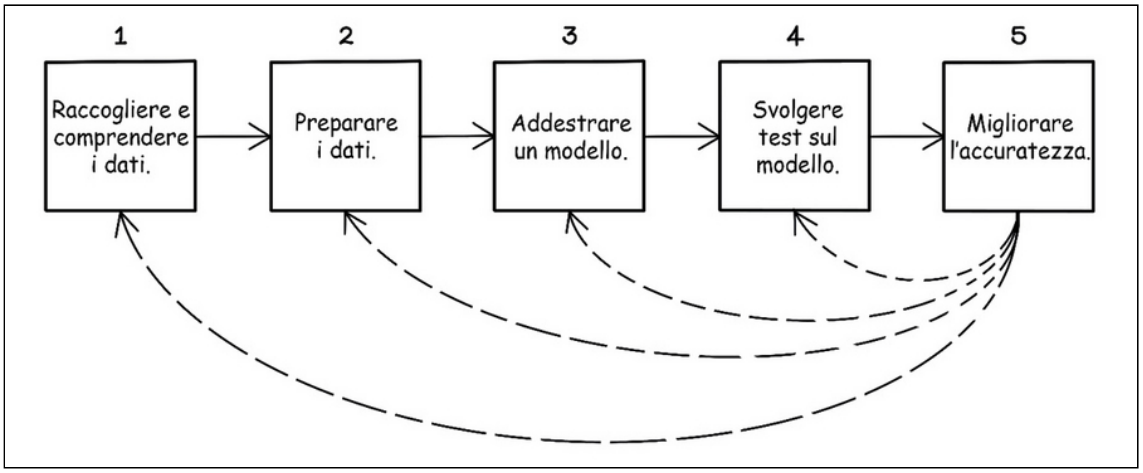
- *Suggerimenti su prodotti e contenuti*: molti di noi utilizzano siti di e-commerce per acquistare beni o servizi di streaming multimediale. I prodotti possono essere consigliati in base a ciò che stiamo acquistando o ai nostri interessi. Alla base di questa funzionalità, generalmente vi è un algoritmo di machine learning, in cui i modelli nel comportamento di acquisto o visualizzazione derivano dalle interazioni dei molti utenti. I sistemi di raccomandazione vengono utilizzati in un numero sempre crescente di settori e applicazioni, per favorire le vendite o fornire una migliore esperienza all'utente.
- *Prezzi dinamici di prodotti e servizi*: i prezzi di prodotti e servizi spesso si basano su ciò che qualcuno è disposto a pagare o in base al rischio. Per un servizio taxi, potrebbe avere senso aumentare il prezzo se ci sono meno auto disponibili rispetto alla domanda. Nel settore assicurativo, un prezzo potrebbe essere aumentato se una persona è classificata come ad alto rischio. Il machine learning viene utilizzato per trovare gli attributi e le relazioni fra quegli attributi che influenzano i prezzi in base a condizioni dinamiche e i dettagli relativi a un individuo.
- *Previsione del rischio per le condizioni di salute*: il settore medico richiede agli operatori sanitari di acquisire molte conoscenze in modo da poter diagnosticare le malattie e curare i pazienti. Nel corso degli anni, hanno acquisito una grande quantità di dati sui pazienti: gruppi sanguigni, DNA, storie di malattie familiari, posizione geografica, stile di vita e altro ancora. Questi dati possono essere utilizzati per trovare potenziali modelli che possono guidare

nella diagnosi di una malattia. Il vantaggio di utilizzare i dati per risalire a una diagnosi è che possiamo trattare le malattie prima che si aggravino. Inoltre, reimmettendo i risultati nel sistema di machine learning, possiamo rafforzare la sua affidabilità nel fare previsioni.

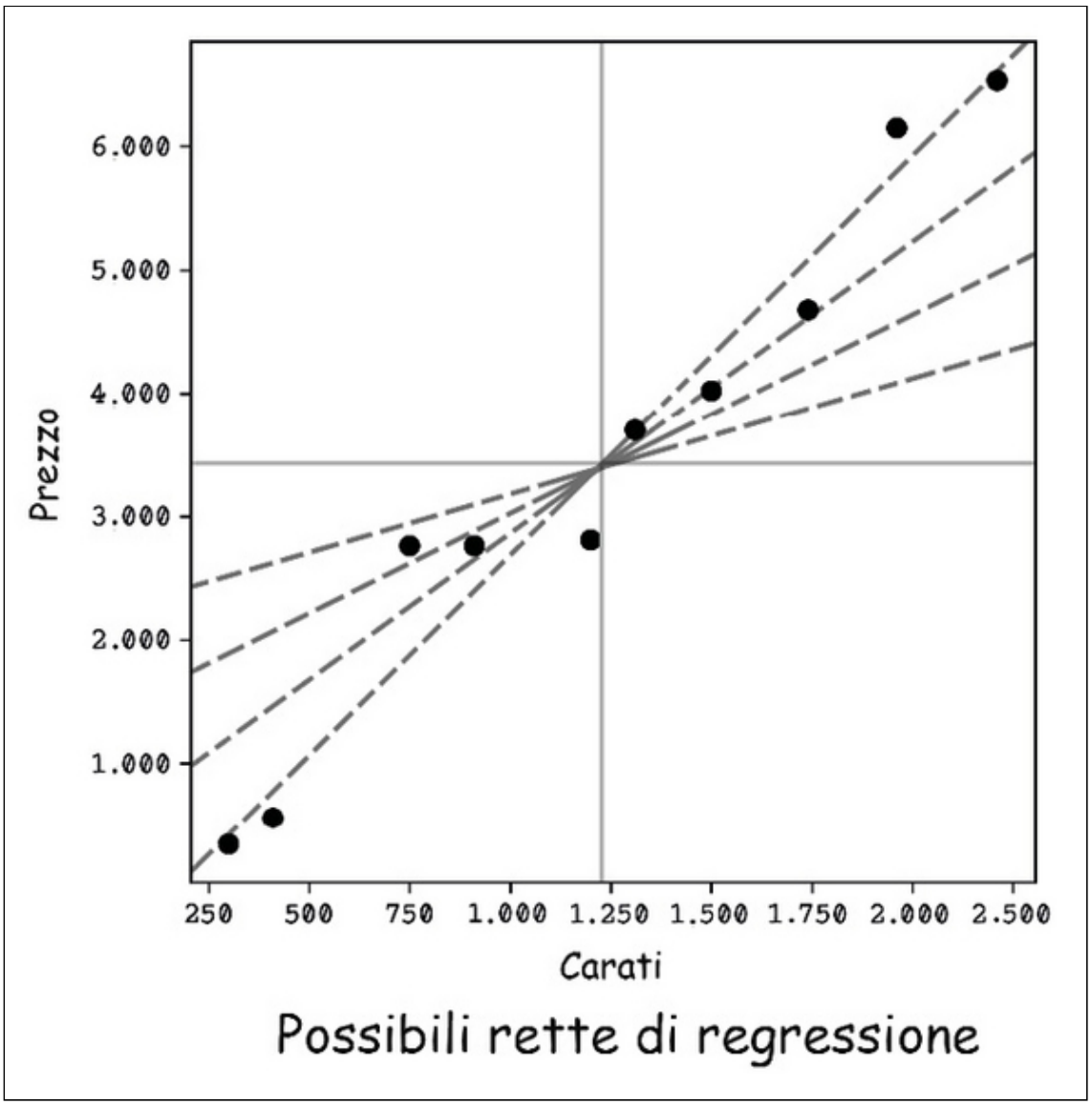
Riepilogo

Il machine learning riguarda più il contesto, la conoscenza dei dati e la scelta di domande corrette che gli algoritmi.

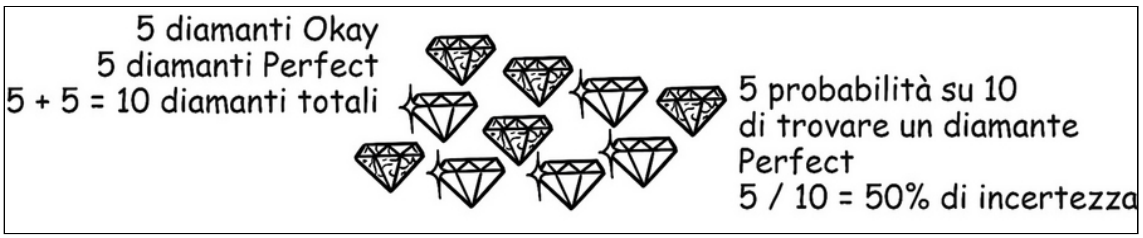
- Il ciclo di vita dei progetti di machine learning è iterativo e sperimentale.

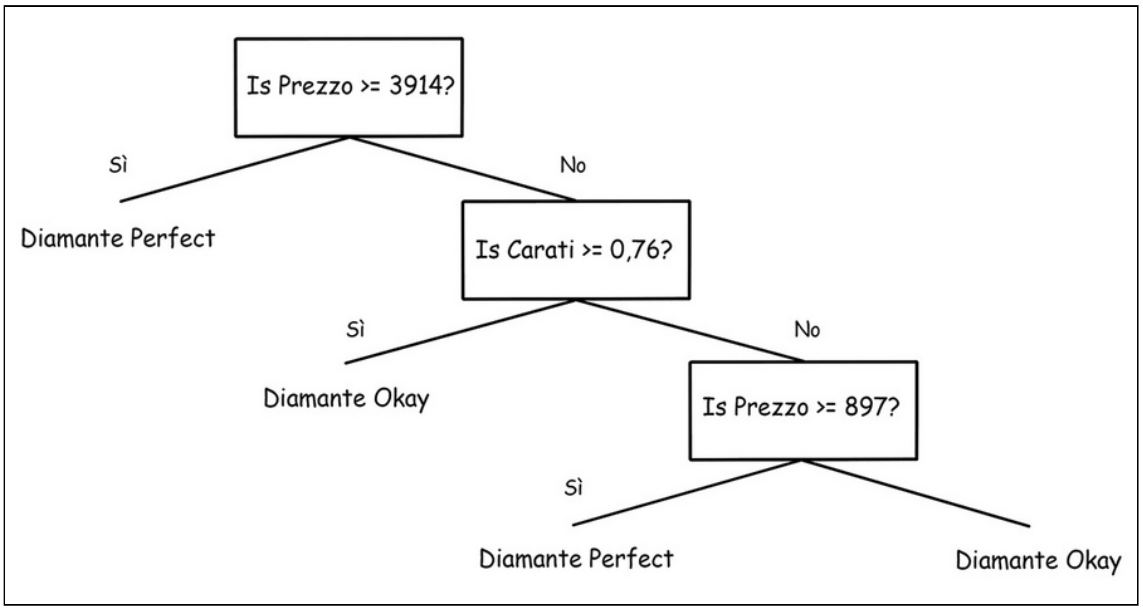


- La regressione lineare prevede la ricerca della linea più efficace nell'individuare i dati, ovvero che minimizza gli errori in ciascun punto dei dati.

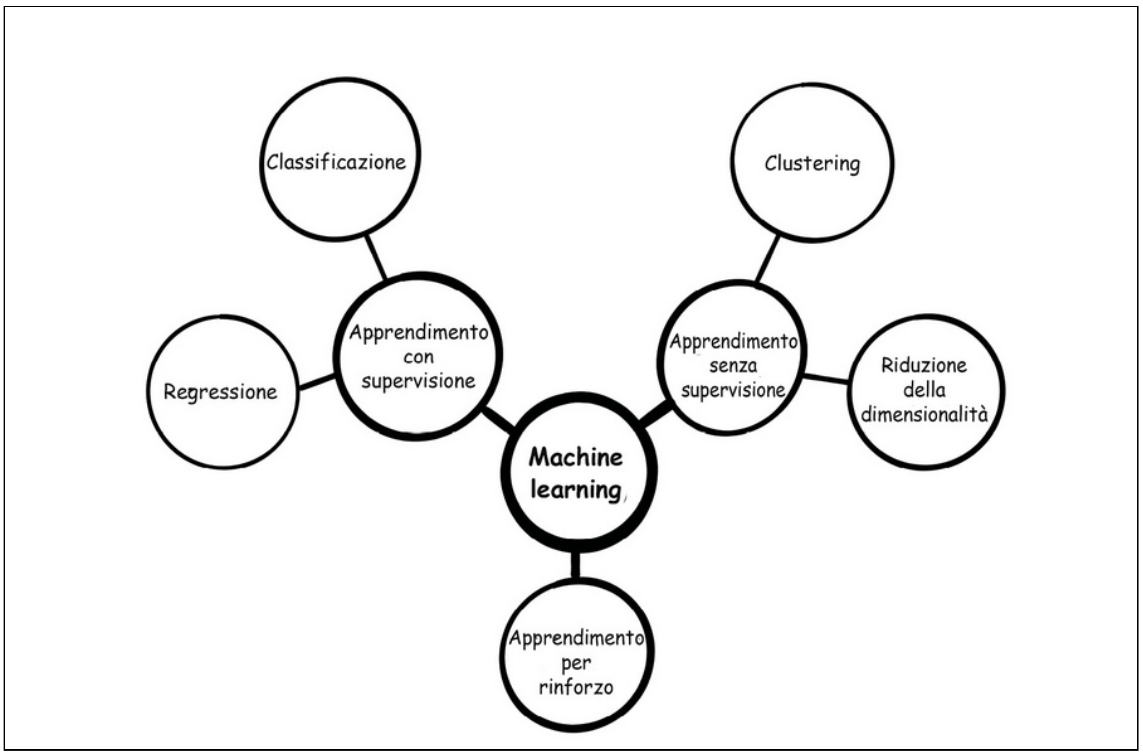


- Gli alberi decisionali suddividono i dati utilizzando domande, fino a che il dataset non viene suddiviso perfettamente in categorie. Il concetto chiave consiste nel ridurre l'incertezza nel dataset.





- Per rispondere a tipi di domande differenti e ottenere obiettivi differenti in contesti differenti vengono impiegati algoritmi di machine learning differenti.


Classificazione
Clustering
Apprendimento con supervisione
Apprendimento senza supervisione
Regressione
Riduzione della dimensionalità
Machine learning
Apprendimento per rinforzo

Reti neurali artificiali

Che cosa sono le reti neurali artificiali?

Le *reti neurali artificiali* (ANN – *Artificial Neural Network*) sono strumenti potenti nel toolkit del machine learning, e sono utilizzate in vari modi per attività come il riconoscimento delle immagini, l'elaborazione del linguaggio naturale e il gioco. Le reti neurali artificiali apprendono in modo simile ad altri algoritmi di machine learning: utilizzando dati di addestramento. Sono particolarmente adatte ai dati non strutturati, nei quali sia difficile capire le relazioni fra le caratteristiche. Questo capitolo tratta i concetti delle reti neurali artificiali; mostra anche come funziona l'algoritmo e come le reti neurali artificiali sono progettate per risolvere diversi problemi.

Per comprendere meglio come si collocano le reti neurali artificiali nel più ampio panorama del machine learning, dobbiamo rivedere la composizione e la categorizzazione degli algoritmi di machine learning. *Deep learning* è il nome comunemente dato agli algoritmi che utilizzano reti neurali artificiali, pur con architetture differenti per raggiungere un obiettivo. Il deep learning, comprese le reti neurali artificiali, può essere utilizzato per risolvere problemi di apprendimento con supervisione e senza supervisione e apprendimento per rinforzo. La Figura 9.1 mostra la correlazione fra deep learning, reti neurali artificiali e altri concetti del machine learning.

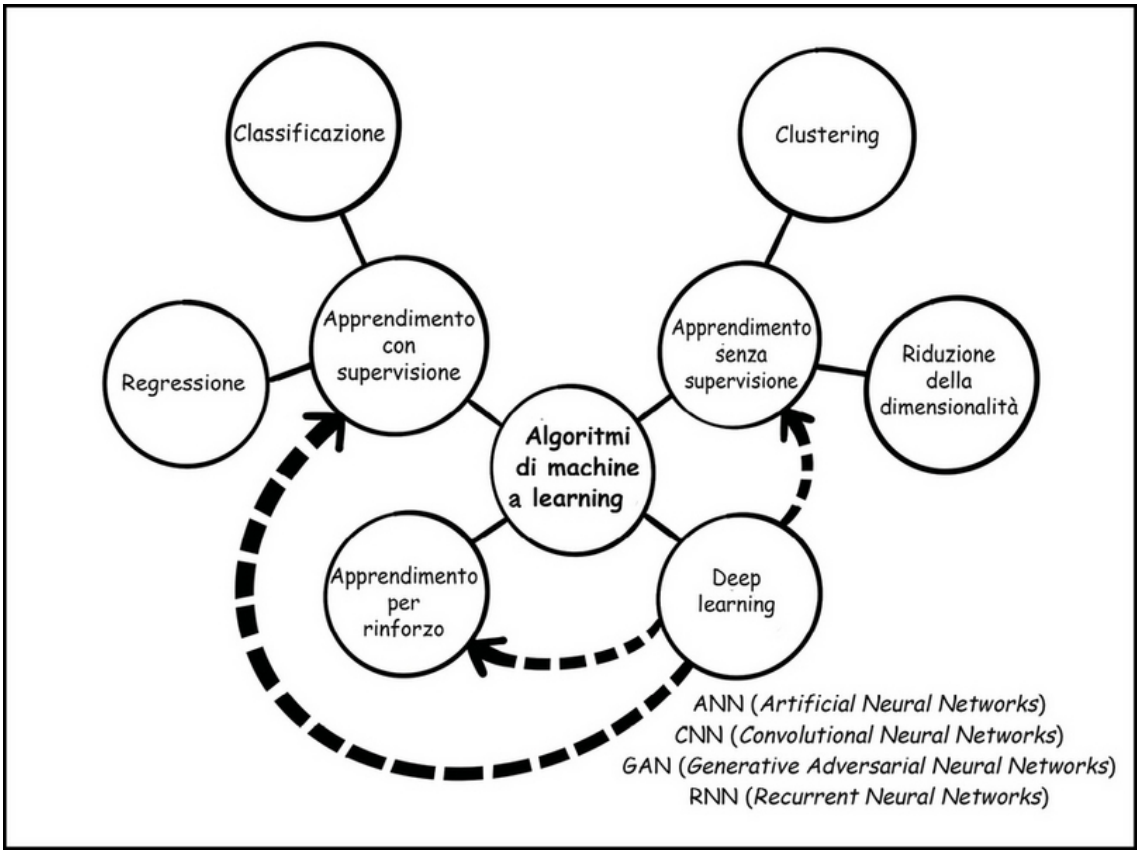


Figura 9.1 Le relazioni fra deep learning e reti neurali artificiali.

Le reti neurali artificiali possono essere considerate come un modello nel ciclo di vita del machine learning (Capitolo 8). La Figura 9.2 riassume tale ciclo di vita. È necessario identificare un problema, raccogliere, comprendere e preparare i dati e il modello di rete neurale artificiale verrà sottoposto a test e migliorato se necessario.

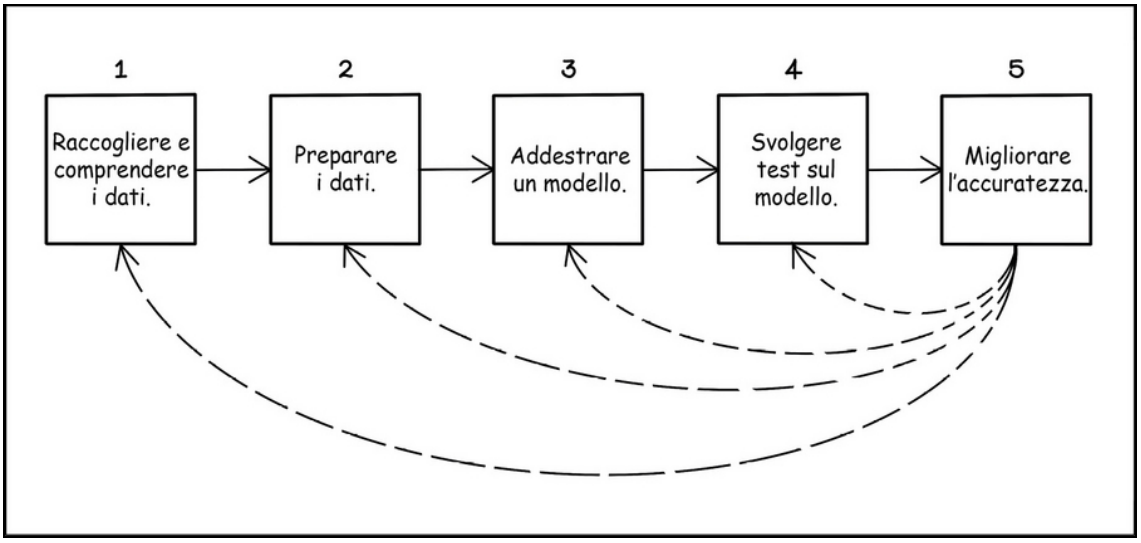


Figura 9.2 Un flusso di lavoro per esperimenti e progetti di machine learning.

Ora che abbiamo un'idea di come le reti neurali artificiali si collocano nel panorama generale del machine learning e che sappiamo che una rete neurale artificiale è un modello addestrato attraverso un ciclo di vita, esploriamone i concetti e il funzionamento. Come gli algoritmi genetici e gli algoritmi di intelligenza di sciame, le reti neurali artificiali sono ispirate da fenomeni naturali, in questo caso il cervello e il sistema nervoso. Il sistema nervoso è una struttura biologica che ci permette di provare sensazioni ed è la base del funzionamento del nostro cervello. Abbiamo nervi in tutto il nostro corpo e neuroni che si comportano tutti in modo simile nel nostro cervello.

Le reti neurali sono costituite da neuroni interconnessi che trasmettono informazioni utilizzando segnali elettrici e chimici. I neuroni trasmettono le informazioni ad altri neuroni e le adattano per svolgere una determinata funzione. Quando prendete una tazza e bevete un sorso d'acqua, milioni di neuroni elaborano l'intenzione di ciò che volete fare, l'azione fisica per realizzarlo e il feedback per determinare se avete avuto successo. Pensate ai bambini piccoli che

imparano a bere da una tazza. Di solito iniziano male, facendo cadere molta acqua. E così imparano ad afferrare la tazza con due mani. A poco a poco imparano ad afferrare la tazza con una sola mano e a berne un sorso senza problemi. Questo processo richiede mesi. Quello che sta succedendo è che il loro cervello e il loro sistema nervoso stanno imparando attraverso la pratica, l'allenamento. La Figura 9.3 illustra un modello semplificato di ricezione di input (gli stimoli), elaborazione di tali stimoli in una rete neurale e fornitura dell'output (la risposta).

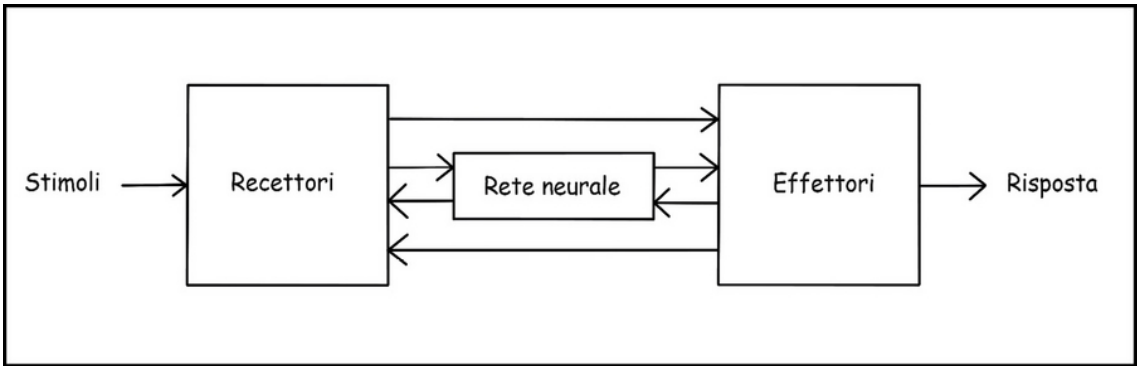


Figura 9.3 Un modello semplificato di un sistema neurale biologico.

Semplificando, un *neurone* (Figura 9.4) è costituito da: dendriti che ricevono segnali da altri neuroni; un corpo cellulare e un nucleo che attiva e regola il segnale; un assone che trasmette il segnale ad altri neuroni; sinapsi che trasportano, e modificano, il segnale prima che venga passato ai dendriti del neurone successivo. Attraverso circa 90 miliardi di neuroni che collaborano tutti fra loro, il nostro cervello può esibire l'elevato livello di intelligenza che conosciamo.

Sebbene le reti neurali artificiali siano ispirate alle reti neurali biologiche e utilizzino molti dei concetti presenti in questi sistemi, le reti neurali artificiali non sono rappresentazioni identiche dei sistemi neurali biologici. Abbiamo ancora molto da imparare sul cervello e sul sistema nervoso.

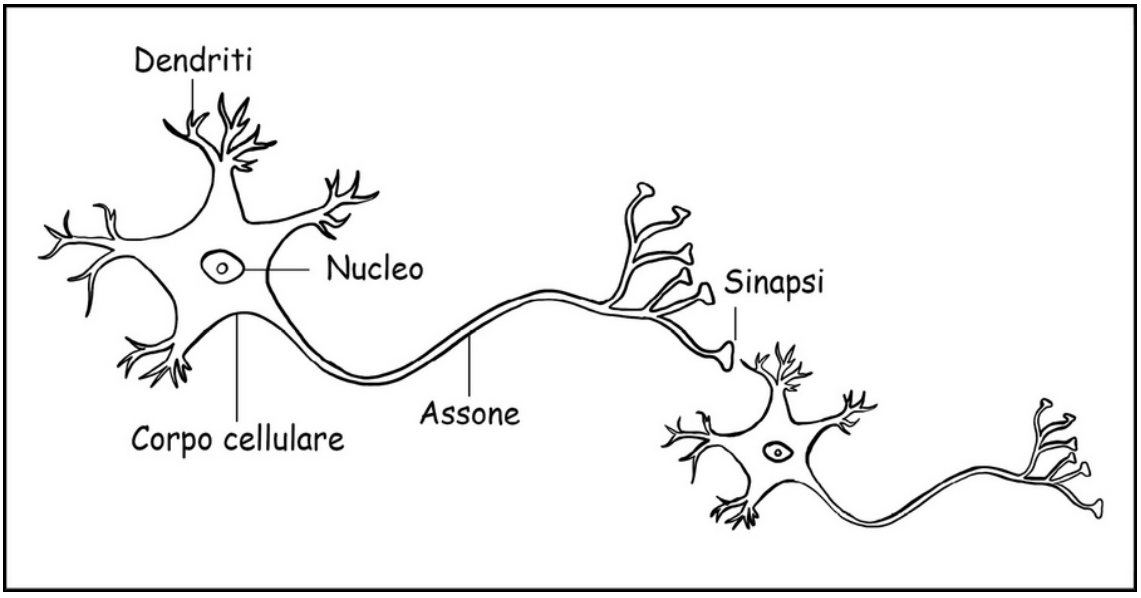


Figura 9.4 La composizione generale dei neuroni.

Il Perceptron: una rappresentazione di un neurone

Il neurone è l'elemento fondamentale che costituisce il cervello e il sistema nervoso. Come accennato in precedenza, accetta molti input da altri neuroni, li elabora e poi trasferisce il risultato ad altri neuroni a lui connessi. Le reti neurali artificiali si basano su un elemento fondamentale chiamato *Perceptron*, una rappresentazione logica di un singolo neurone biologico.

Come un neurone, un Perceptron riceve degli input (come i dendriti), modifica questi input utilizzando dei pesi (come le sinapsi), elabora in modo pesato gli input (come il corpo cellulare e il suo nucleo) e produce un risultato (come gli assoni). Il Perceptron è vagamente ispirato a un neurone. Si può notare che le sinapsi sono rappresentate dopo i dendriti, per rappresentare l'influenza delle sinapsi sugli input in ingresso. La Figura 9.5 illustra l'architettura logica del Perceptron.

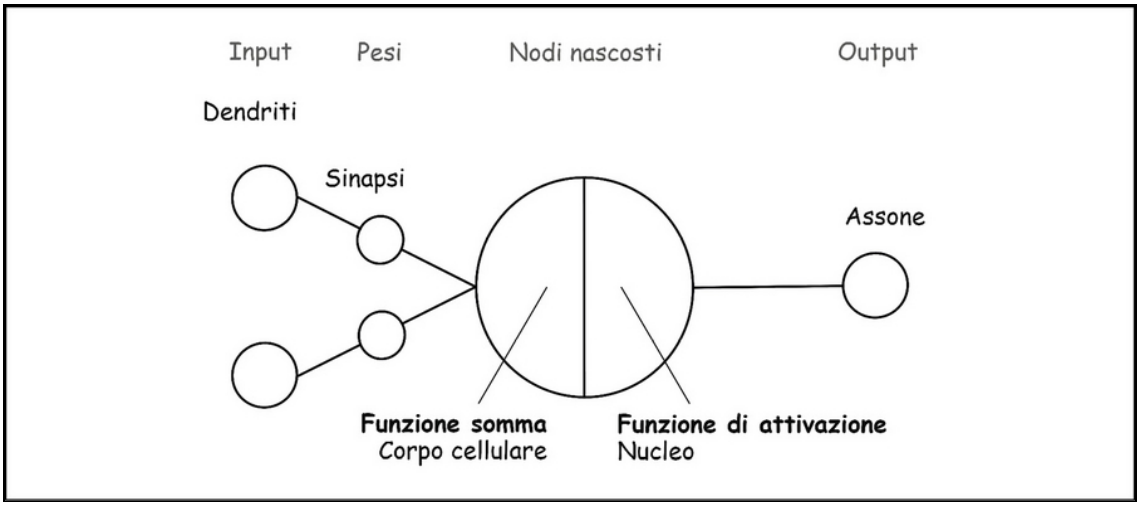


Figura 9.5 Architettura logica del Perceptron.

I componenti del Perceptron sono descritti tramite variabili, utili per il calcolo dell'output. I pesi modificano gli input; quel valore viene elaborato da un nodo nascosto; infine, come output viene fornito il risultato.

Ecco una breve descrizione dei componenti del Perceptron.

- *Input*: descrive i valori di input. In un neurone, questi valori sarebbero un segnale di input.
- *Pesi*: descrive l'importanza di ciascuna connessione di input e il nodo nascosto. I pesi influenzano l'intensità di un input e ciò determina una ponderazione dell'input. In un neurone, queste connessioni sarebbero le sinapsi.
- *Nodo nascosto* (*somma* e *attivazione*): somma i valori di input pesati e poi applica una funzione di attivazione al risultato. Una funzione di attivazione determina l'attivazione/output del nodo/neurone nascosto.
- *Output*: descrive l'output finale del Perceptron.

Per comprendere il funzionamento del Perceptron, ne esamineremo l'uso rivisitando l'esempio della ricerca di un appartamento (Capitolo

8). Supponiamo di essere un agente immobiliare che cerca di determinare se un determinato appartamento verrà affittato entro un mese, sulla base delle sue dimensioni e del suo prezzo. Supponiamo che il Perceptron sia già stato addestrato, il che significa che i suoi pesi sono già stati regolati. Più avanti in questo capitolo esploreremo il modo in cui vengono addestrati il Perceptron e le reti neurali artificiali; per ora, tenete a mente che i pesi codificano le relazioni fra gli input, regolando la loro forza.

La Figura 9.6 mostra come possiamo utilizzare un Perceptron preaddestrato per classificare un appartamento: verrà affittato o no? Gli input sono il prezzo e le dimensioni di un determinato appartamento. Utilizziamo anche il prezzo e le dimensioni massimi per ridimensionare gli input (8000 dollari per il prezzo massimo e 80 metri quadrati per le dimensioni massime). Per ulteriori informazioni sulla scalabilità dei dati, consultate il prossimo paragrafo.

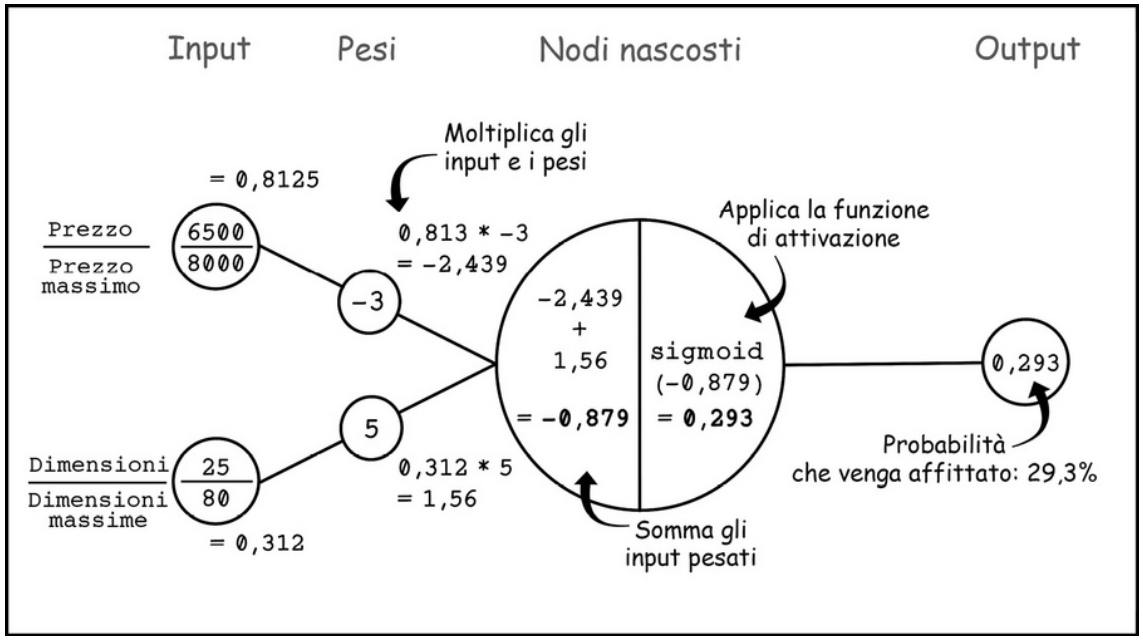


Figura 9.6 Un esempio di utilizzo di un Perceptron addestrato.

Notate che gli input sono il prezzo e le dimensioni e che l'output è la probabilità prevista che l'appartamento venga affittato. I pesi sono fondamentali per ottenere la previsione. I pesi sono le variabili che, nella rete, apprendono le relazioni fra gli input. Le funzioni di sommatoria e attivazione vengono utilizzate per elaborare gli input moltiplicati per i pesi, per emettere la previsione.

Notate che stiamo usando una funzione di attivazione chiamata *funzione sigmoide*. Le funzioni di attivazione svolgono un ruolo fondamentale nel Perceptron e nelle reti neurali artificiali. In questo caso, la funzione di attivazione ci aiuta a risolvere un problema lineare. Ma quando, nel prossimo paragrafo, esamineremo le reti neurali artificiali, vedremo come le funzioni di attivazione sono utili per ricevere input e risolvere anche problemi non lineari. La Figura 9.7 descrive le basi dei problemi lineari.

La funzione sigmoide produce una curva a "S" compresa fra 0 e 1, dati input compresi fra 0 e 1. Poiché la funzione sigmoide consente ai cambiamenti in x di provocare piccoli cambiamenti in y, rende possibile l'apprendimento graduale. Quando esamineremo i meccanismi più profondi delle reti neurali artificiali, più avanti in questo capitolo, vedremo come questa funzione aiuti anche a risolvere problemi non lineari.

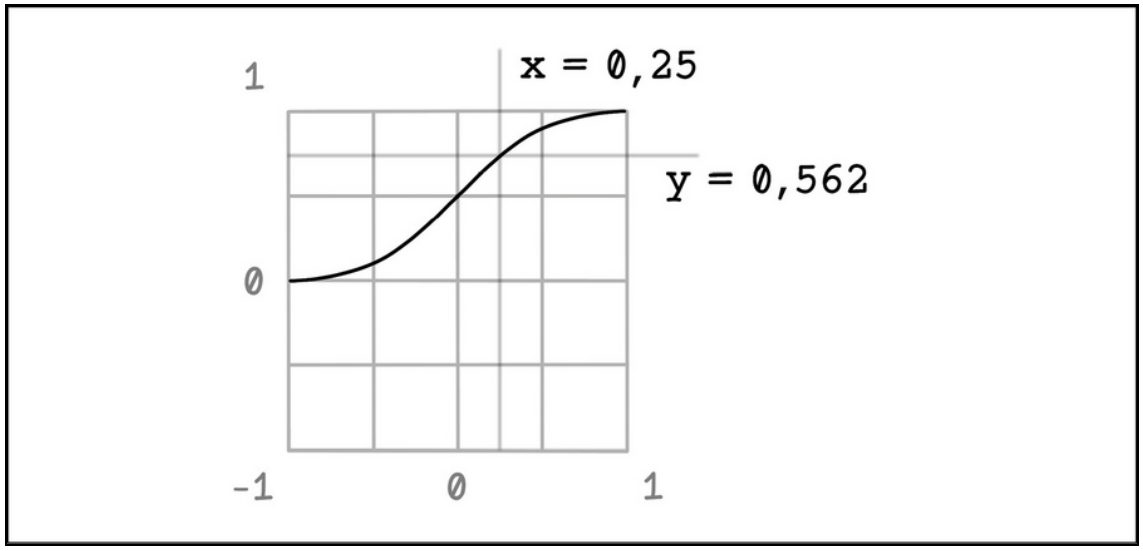


Figura 9.7 La funzione sigmoide.

Facciamo un passo indietro e guardiamo i dati che stiamo usando per il Perceptron. Comprendere i dati relativi alla vendita di un appartamento è importante per capire che cosa sta facendo il Perceptron. La Figura 9.8 presenta gli esempi del dataset, inclusi il prezzo e le dimensioni di ciascun appartamento. Ogni appartamento è etichettato in due classi: affittato o non affittato. La linea che separa le due classi è la funzione descritta dal Perceptron.

Il Perceptron è utile per risolvere problemi lineari, mentre non può risolvere problemi non lineari. Se un dataset non può essere classificato da una linea retta, il Perceptron fallirà.

Le reti neurali artificiali utilizzano il concetto di Perceptron su larga scala. Per risolvere problemi non lineari in molte dimensioni devono collaborare molti neuroni simili al Perceptron. Notate che la funzione di attivazione utilizzata influenza le capacità di apprendimento della rete neurale artificiale.

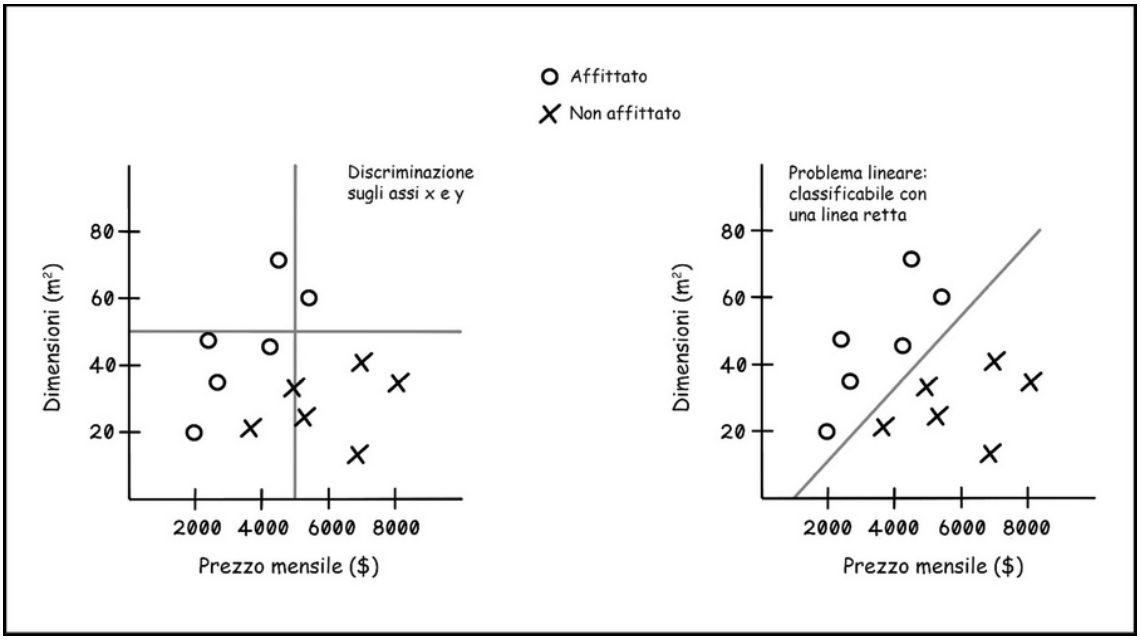
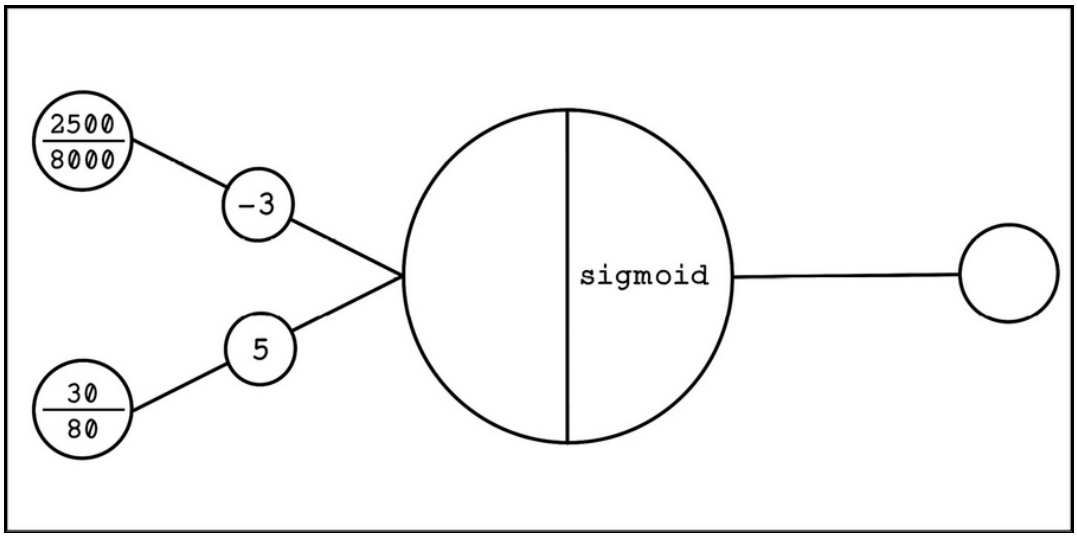


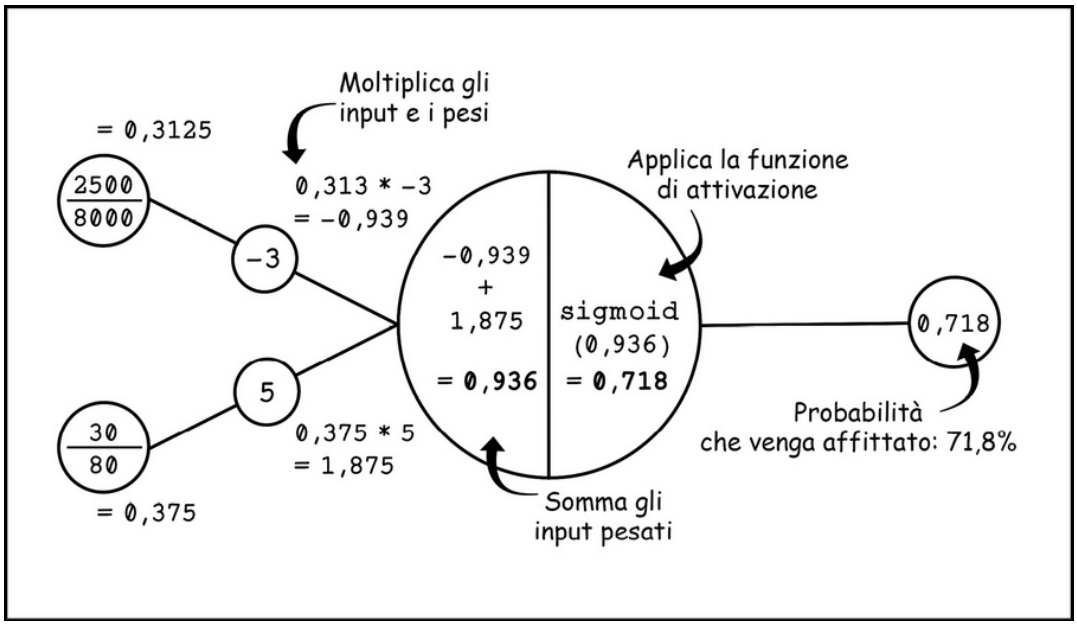
Figura 9.8 Esempio di un problema di classificazione lineare.

Esercizio: calcolare l'output del seguente input per il Perceptron

Usando le vostre attuali conoscenze del funzionamento del Perceptron, calcolare l'output per quanto segue:



Soluzione



Definizione di reti neurali artificiali

Il Perceptron è utile per risolvere problemi semplici, ma all'aumentare delle dimensioni dei dati diventa meno efficace. Le reti neurali artificiali utilizzano i principi del Perceptron e li applicano a molti nodi nascosti anziché uno solo.

Per esplorare il funzionamento delle reti neurali artificiali multinodo, considerate un dataset di esempio relativo agli incidenti automobilistici. Supponiamo di avere dati da diverse auto nel momento in cui un oggetto imprevisto entra nel percorso del loro movimento. Il dataset contiene caratteristiche relative alle condizioni rilevate e se si è verificata o meno una collisione.

- *Velocità*: la velocità alla quale viaggiava l'auto prima di trovare l'oggetto.
- *Qualità fondo*: la qualità del fondo sulla strada percorsa dall'auto prima di trovare l'oggetto.
- *Visibilità*: il grado di visibilità per il conducente prima che l'auto trovasse l'oggetto.
- *Esperienza*: l'esperienza di guida del conducente dell'auto.
- *Collisione?*: indica se si è verificata o meno una collisione.

Dati questi dati, vogliamo addestrare un modello di machine learning, ovvero una rete neurale artificiale, per apprendere la relazione fra le caratteristiche che contribuiscono a una collisione, come mostrato nella Tabella 9.1.

Tabella 9.1 Dataset degli incidenti automobilistici.

	Velocità (km/h)	**Qualità fondo**	**Visibilità**	**Esperienza (km)**	**Collisione?**
1	65	5/10	180°	80.000	No
2	120	1/10	72°	110.000	Sì
3	8	6/10	288°	50.000	No
4	50	2/10	324°	1.600	Sì
5	25	9/10	36°	160.000	No
6	80	3/10	120°	6.000	Sì

7	40	3/10	360°	400.000	No

Per classificare se si verificherà una collisione in base alle caratteristiche che abbiamo può essere utilizzata un'architettura a rete neurale artificiale. Data la rete neurale artificiale, le caratteristiche presenti nel dataset devono essere mappate come input e la classe che stiamo cercando di prevedere viene mappata come output. In questo esempio, i nodi di input sono la velocità, la qualità del fondo stradale, il grado di visibilità e l'esperienza di guida; il nodo di output è se si è verificata o meno una collisione (Figura 9.9).

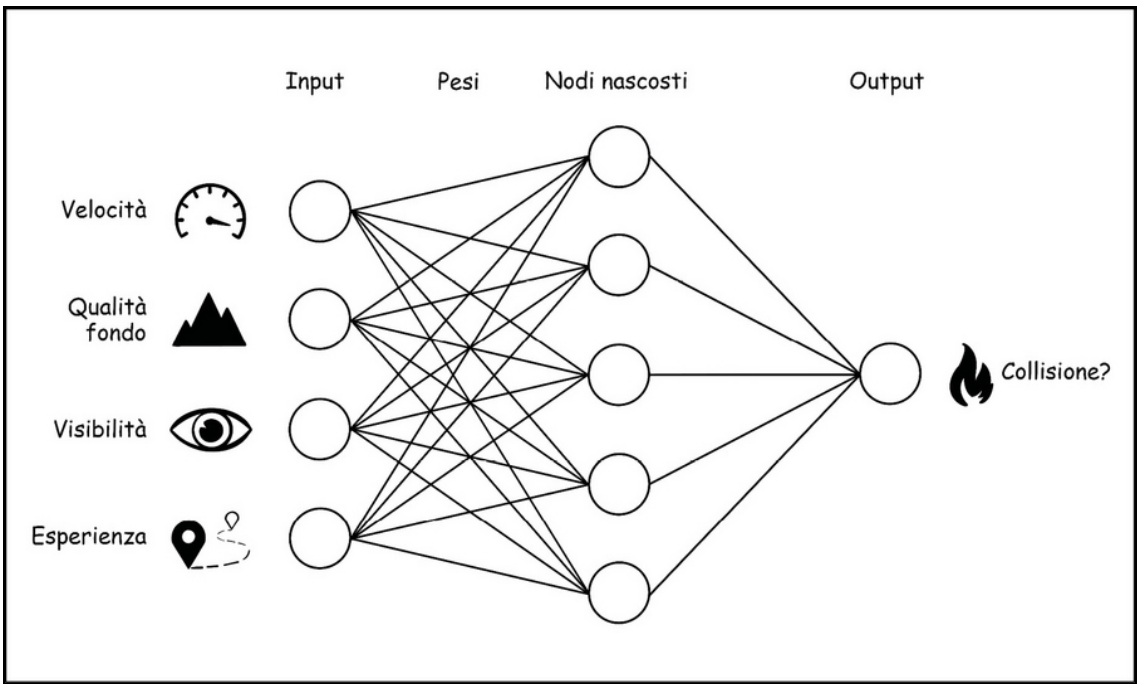


Figura 9.9 Esempio di architettura a rete neurale artificiale per l'esempio degli incidenti automobilistici.

Come per gli altri algoritmi di machine learning che abbiamo impiegato, la preparazione dei dati è importante per fare in modo che la rete neurale artificiale classifichi i dati con successo. Il problema principale è rappresentare i dati in modi paragonabili. Come esseri umani, comprendiamo il concetto di velocità e di grado di visibilità,

ma la rete neurale artificiale non può contare su questo contesto. Confrontare direttamente una velocità di 65 km/h e una visibilità di 36 gradi non ha senso per la rete neurale artificiale, ma è utile stabilire una relazione fra velocità e grado di visibilità. Per svolgere questo compito, dobbiamo ridimensionare i nostri dati.

Un modo comune per ridimensionare i dati in modo che possano essere confrontati consiste nell'utilizzare l'approccio di ridimensionamento *min-max*, che punta a ridimensionare i dati a valori compresi fra 0 e 1. Ridimensionando tutti i dati del dataset in modo che abbiano un formato coerente, rendiamo confrontabili le varie caratteristiche. Poiché le reti neurali artificiali non hanno alcun contesto sul significato delle caratteristiche, rimuoviamo anche i bias con valori di input elevati. Per esempio, 1000 sembra essere molto più grande di 65, ma 1000 km nel contesto dell'esperienza di guida è un valore piccolo e 65 km/h nel contesto della velocità di guida può essere un valore elevato. Il ridimensionamento min-max rappresenta questi dati con il contesto corretto, tenendo conto dei valori minimi e massimi possibili per ciascuna caratteristica.

Di seguito sono riportati i valori minimo e massimo selezionati per le caratteristiche nel dataset degli incidenti automobilistici.

- *Velocità*: la velocità minima è 0, il che significa che l'auto non è in movimento. Useremo come velocità massima 120, perché 120 km/h è il limite massimo di velocità in molti Paesi. Supponiamo che il conducente segua le regole.
- *Qualità fondo*: poiché i dati sono già in scala secondo un sistema di valutazione, il valore minimo è 0 e il valore massimo è 10.
- *Visibilità*: sappiamo che il campo visivo totale è di 360°. Quindi il valore minimo è 0 e il valore massimo è 360.
- *Esperienza*: il valore minimo è 0 km, se il conducente non ha alcuna esperienza di guida. Imponiamo il valore massimo 400.000

per l'esperienza di guida. Per logica, se un conducente ha 400.000 km di esperienza di guida, lo consideriamo altamente competente e qualsiasi ulteriore esperienza non ha importanza.

Il ridimensionamento min-max utilizza i valori minimo e massimo per una caratteristica e trova la percentuale del valore effettivo per tale caratteristica. La formula è semplice: sottrarre il minimo dal valore e dividere il risultato per il valore massimo - minimo. La Figura 9.10 illustra il calcolo della scala min-max per la prima riga di dati del dataset di incidenti automobilistici.

Tabella 9.2

	Velocità (km/h)	Qualità fondo	Visibilità	Esperienza (km)	Collisione?
1	65	5/10	180°	80.000	No

Notate che ora tutti i valori sono compresi fra 0 e 1 e possono essere confrontati fra loro. La stessa formula viene applicata a tutte le righe del dataset, per ridimensionare ogni valore. Notate che per il valore della caratteristica "Collisione?", Sì è 1 e No è 0. La Tabella 9.3 mostra i dati relativi agli incidenti automobilistici in scala.

Tabella 9.3 Dataset degli incidenti automobilistici in scala.

	Velocità (km/h)	Qualità fondo	Visibilità	Esperienza (km)	Collisione?
1	0,542	0,5	0,5	0,200	0
2	1,000	0,1	0,2	0,275	1
3	0,067	0,6	0,8	0,125	0
4	0,417	0,2	0,9	0,004	1
5	0,208	0,9	0,1	0,400	0
6	0,667	0,3	0,3	0,015	1
7	0,333	0,3	1,0	1,000	0

Pseudocodice

Il codice per il ridimensionamento dei dati segue la logica e i calcoli per il ridimensionamento min-max. Abbiamo bisogno dei minimi e dei massimi per ogni funzionalità, nonché del numero totale di funzionalità presenti nel nostro dataset. La funzione `scale_dataset` utilizza questi parametri per scorrere ogni esempio presente nel dataset e ridimensionare il valore utilizzando la funzione `scale_data_feature`:

```
FEATURE_MIN = [0, 0, 0, 0]
FEATURE_MAX = [120, 10, 360, 400000]
FEATURE COUNT = 4
scale_dataset(dataset, feature_count, feature_min, feature_max):
  let scaled_data equal empty array
  for data in dataset:
      let example equal empty array
      for i in range(0, feature_count):
                    append  scale_data_feature(data[i],  feature_min[i],
feature_max[i]) to example
      append example to scaled_data
  return scaled_data
scale_data_feature(data, feature_min, feature_max):
  return(data - feature_min) / (feature_max - feature_min)
```

	Velocità	Qualità fondo	Visibilità	Esperienza
	65 km/h Min: 0 Max: 120	5/10 Min: 0 Max: 10	180° Min: 0 Max: 360	80,000 Min: 0 Max: 400,000
$\frac{\text{Valore} - \text{min}}{\text{max} - \text{min}}$	$\frac{65 - 0}{120 - 0}$	$\frac{5 - 0}{10 - 0}$	$\frac{180 - 0}{360 - 0}$	$\frac{80000 - 0}{400000 - 0}$
Valore in scala	0,542	0,5	0,5	0,2

Figura 9.10 Esempio di ridimensionamento min-max con i dati del primo esempio.

Ora che abbiamo preparato i dati in un modo adatto all'elaborazione, esploriamo l'architettura di una semplice rete neurale artificiale.

Ricordate che le caratteristiche utilizzate per prevedere una classe sono i nodi di input e la classe che viene prevista è il nodo di output.

La Figura 9.11 mostra una rete neurale artificiale con un livello nascosto, che è l'unico livello verticale della figura, con cinque nodi nascosti. Questi sono detti *livelli nascosti* perché non sono direttamente visibili dall'esterno della rete. Si interagisce con essi solo attraverso gli input e gli output, il che contribuisce a considerare le reti neurali artificiali come "scatole nere". Ogni nodo nascosto è simile a un Perceptron. Un nodo nascosto prende gli input e i pesi, calcola la somma ed esegue una funzione di attivazione. Quindi i risultati di ciascun nodo nascosto vengono elaborati da un singolo nodo di output.

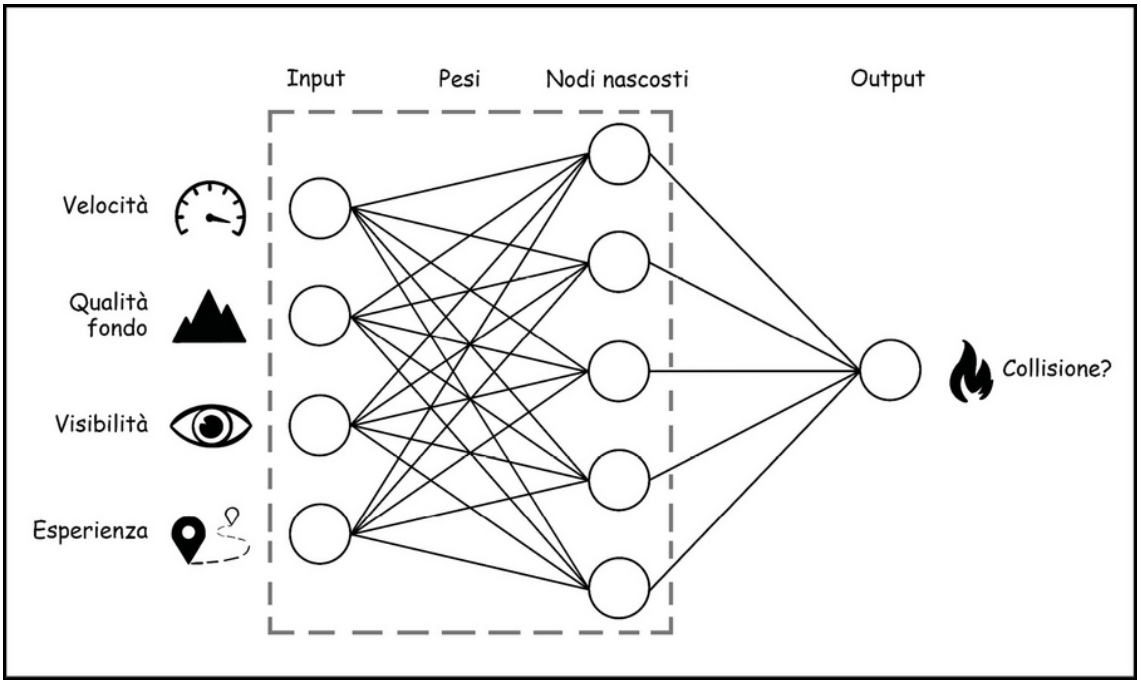


Figura 9.11 Esempio di architettura a rete neurale artificiale per il problema degli incidenti automobilistici.

Prima di considerare i calcoli di una rete neurale artificiale, proviamo a esaminare intuitivamente quello che stanno facendo, ad alto livello, i pesi della rete. Poiché un singolo nodo nascosto è connesso a ogni nodo di input ma ogni connessione ha un peso diverso, i nodi

nascosti potrebbero essere interessati a determinate relazioni fra due o più nodi di input.

La Figura 9.12 rappresenta uno scenario in cui il primo nodo nascosto ha forti pesi sul legame fra qualità del fondo stradale e grado di visibilità, ma pesi deboli sul legame fra velocità ed esperienza di guida. Questo specifico nodo nascosto riguarda la relazione fra la qualità del fondo stradale e il grado di visibilità. Potrebbe acquisire una comprensione della relazione fra queste due caratteristiche e di come essa influenzi il verificarsi di collisioni; una scarsa qualità del fondo stradale e uno scarso grado di visibilità, per esempio, potrebbero influenzare la probabilità di collisioni più di una buona qualità del fondo stradale e un grado medio di visibilità. Queste relazioni sono solitamente più complesse di questo semplice esempio.

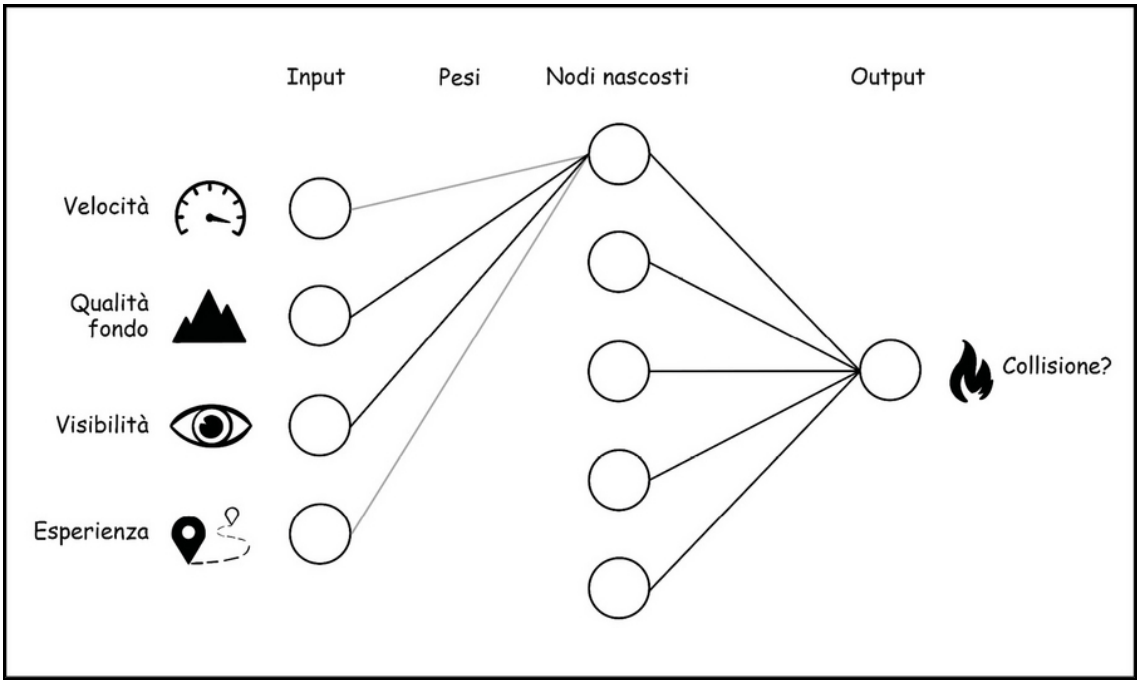


Figura 9.12 Esempio di un nodo nascosto che confronta la qualità del fondo stradale e il grado di visibilità.

Nella Figura 9.13, il secondo nodo nascosto potrebbe avere pesi elevati sul legame fra qualità del fondo stradale ed esperienza di guida.

Forse c'è una relazione fra le diverse qualità del fondo stradale e la variabilità nell'esperienza di guida totale e che contribuisce alle collisioni.

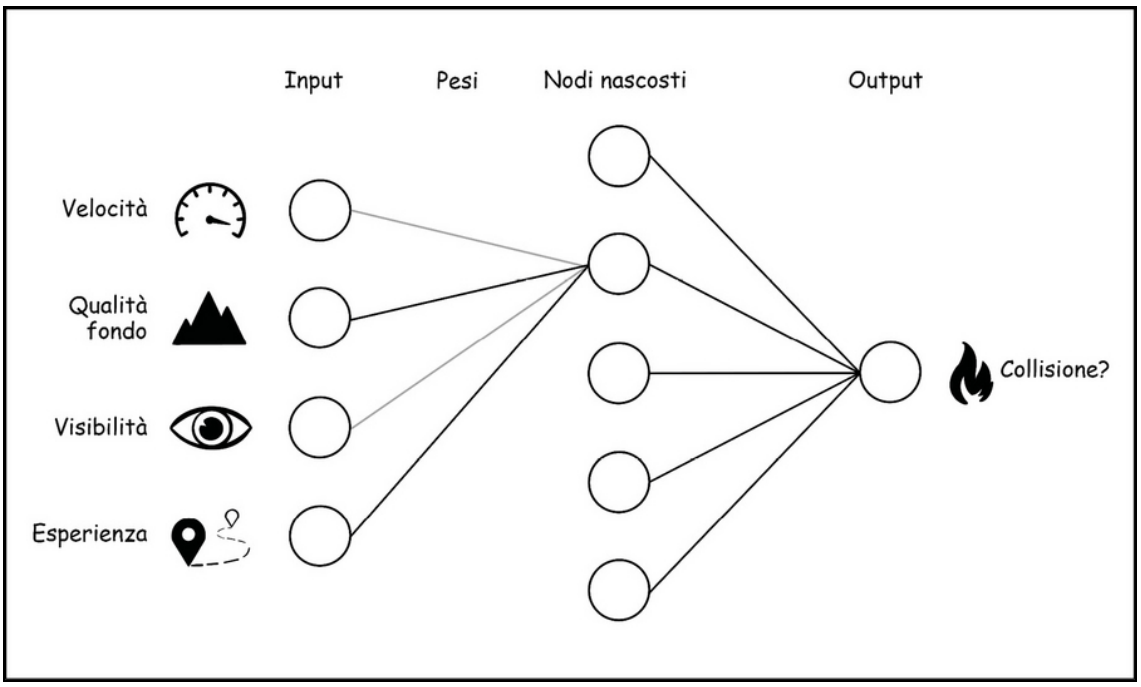


Figura 9.13 Esempio di un nodo nascosto che confronta la qualità del fondo stradale e l'esperienza di guida.

I nodi in un livello nascosto possono essere confrontati concettualmente con l'analogia delle formiche già discussa nel Capitolo 6. Le singole formiche svolgono piccoli compiti apparentemente insignificanti, ma quando agiscono come una colonia, dall'insieme emerge un comportamento intelligente. Allo stesso modo, i singoli nodi nascosti contribuiscono a un obiettivo più grande, nella rete neurale artificiale.

Analizzando la figura della rete neurale artificiale per gli incidenti automobilistici e le operazioni svolte al suo interno, possiamo descrivere le strutture di dati necessarie per l'algoritmo.

- *Nodi di input*: possono essere rappresentati da un singolo array che memorizza i valori per un determinato esempio. La dimensione dell'array è il numero di caratteristiche presenti nel dataset che vengono utilizzate per prevedere una classe. Nell'esempio degli incidenti automobilistici, abbiamo quattro input, quindi la dimensione dell'array è 4.
- *Pesi*: possono essere rappresentati da una matrice (un array bidimensionale), poiché ciascun nodo di input ha una connessione con ciascun nodo nascosto e ciascun nodo di input ha cinque connessioni. Poiché ci sono 4 nodi di input con 5 connessioni ciascuno, la rete neurale artificiale ha 20 pesi verso il livello nascosto e 5 verso il livello di output, perché ci sono 5 nodi nascosti e 1 nodo di output.
- *Nodi nascosti*: possono essere rappresentati da un singolo array, che memorizza i risultati dell'attivazione di ciascun nodo.
- *Nodo di output*: è un singolo valore che rappresenta la classe prevista per un determinato esempio o la possibilità che l'esempio si trovi in una determinata classe. L'output potrebbe essere 1 o 0, per indicare se la collisione si è verificata o meno; oppure potrebbe essere qualcosa come 0,65, che indica una probabilità del 65% che l'esempio abbia avuto come esito una collisione.

Pseudocodice

Il prossimo frammento di pseudocodice descrive una classe che rappresenta una rete neurale. Notate che i livelli sono rappresentati come proprietà della classe e che tutte le proprietà sono array, a eccezione dei pesi, che sono matrici. Una proprietà `output` rappresenta le previsioni per gli esempi forniti e durante il processo di addestramento viene utilizzata una proprietà `expected_output`:

```
NeuralNetwork(features, labels, hidden_node_count):
  let input equal features
     let weights_input equal a random matrix, size: features *
hidden_node_count
  let hidden equal zero array, size: hidden_node_count
  let weights_hidden equal a random matrix, size: hidden_node_count
```

```
  let expected_output equal labels
  let output equal zero array, size: length of labels
let nn equal NeuralNetwork(scaled_feature_data, scaled_label_data,
hidden_node_count)
```

Propagazione in avanti: utilizzo di una rete neurale artificiale addestrata

Una rete neurale artificiale addestrata è una rete che ha imparato dagli esempi e ha adattato i propri pesi per prevedere al meglio la classe dei nuovi esempi che le verranno sottoposti. Non preoccupatevi su come avviene l'addestramento e su come vengono regolati i pesi; affronteremo questo argomento nel prossimo paragrafo. Comprendere la propagazione in avanti ci aiuterà a comprendere anche la propagazione all'indietro (ovvero come vengono modificati/addestrati i pesi).

Ora che conosciamo le basi dell'architettura generale delle reti neurali artificiali e sappiamo che cosa possono fare i nodi della rete, esaminiamo l'algoritmo per l'utilizzo di una rete neurale artificiale addestrata (Figura 9.14).

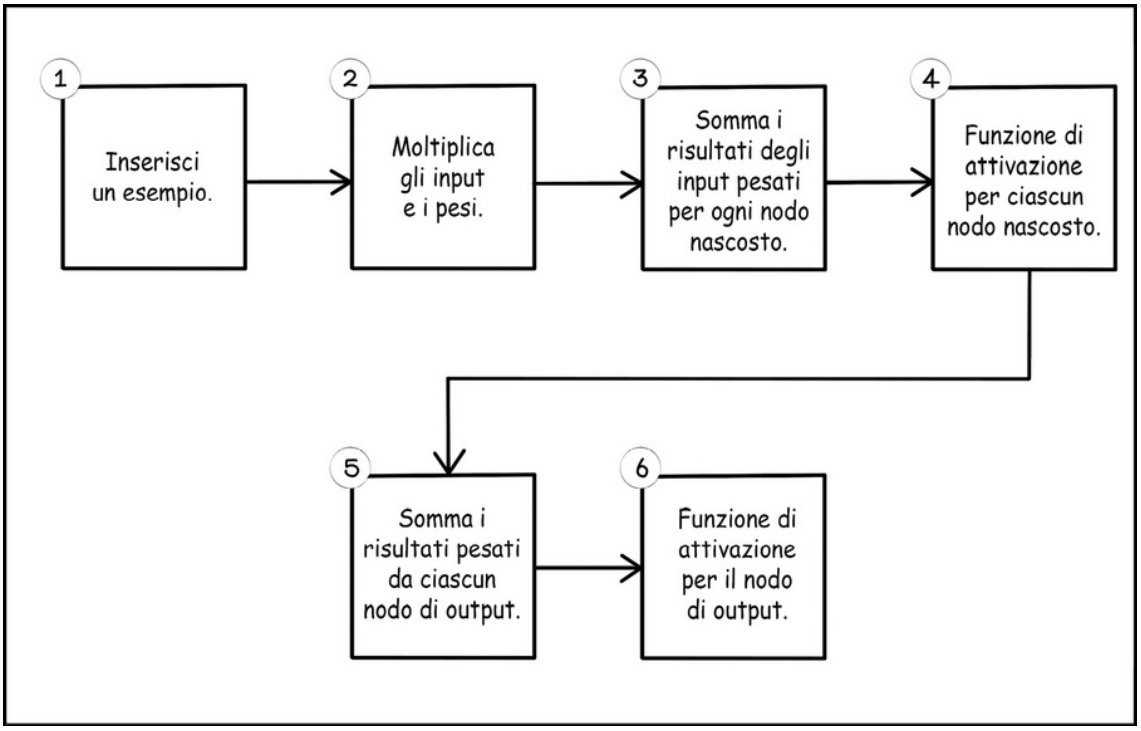


Figura 9.14 Ciclo di vita della propagazione in avanti in una rete neurale artificiale.

Come accennato in precedenza, i passaggi coinvolti nel calcolo dei risultati per i nodi di una rete neurale artificiale sono simili a quelli del Perceptron. Operazioni simili vengono eseguite in molti nodi che collaborano fra loro; questo meccanismo risolve i difetti del Perceptron e viene utilizzato per risolvere problemi che riguardano più dimensioni. Il flusso generale della propagazione in avanti include i seguenti passaggi.

1. *Inserisci un esempio*: un singolo esempio dal dataset per il quale desideriamo prevedere la classe.
2. *Moltiplica gli input e i pesi*: moltiplica ogni input per ogni peso per le connessioni con i nodi nascosti.
3. *Somma i risultati degli input pesati per ciascun nodo nascosto*: somma i risultati degli input pesati.

4. *Funzione di attivazione per ciascun nodo nascosto*: applica una funzione di attivazione agli input pesati e sommati.
5. *Somma i risultati pesati dei nodi nascosti al nodo di output*: somma i risultati pesati della funzione di attivazione da tutti i nodi nascosti.
6. *Funzione di attivazione per il nodo di output*: applica la funzione di attivazione ai nodi nascosti pesati sommati.

Allo scopo di esplorare la propagazione in avanti, presumeremo che la rete neurale artificiale sia stata addestrata e che siano stati trovati i pesi ottimali nella rete. La Figura 9.15 mostra i pesi su ciascuna connessione. La prima casella accanto al primo nodo nascosto, per esempio, ha peso 3,35, che è relativo al nodo di input Velocità; il peso -5,82 è relativo al nodo di input Qualità fondo; e così via.

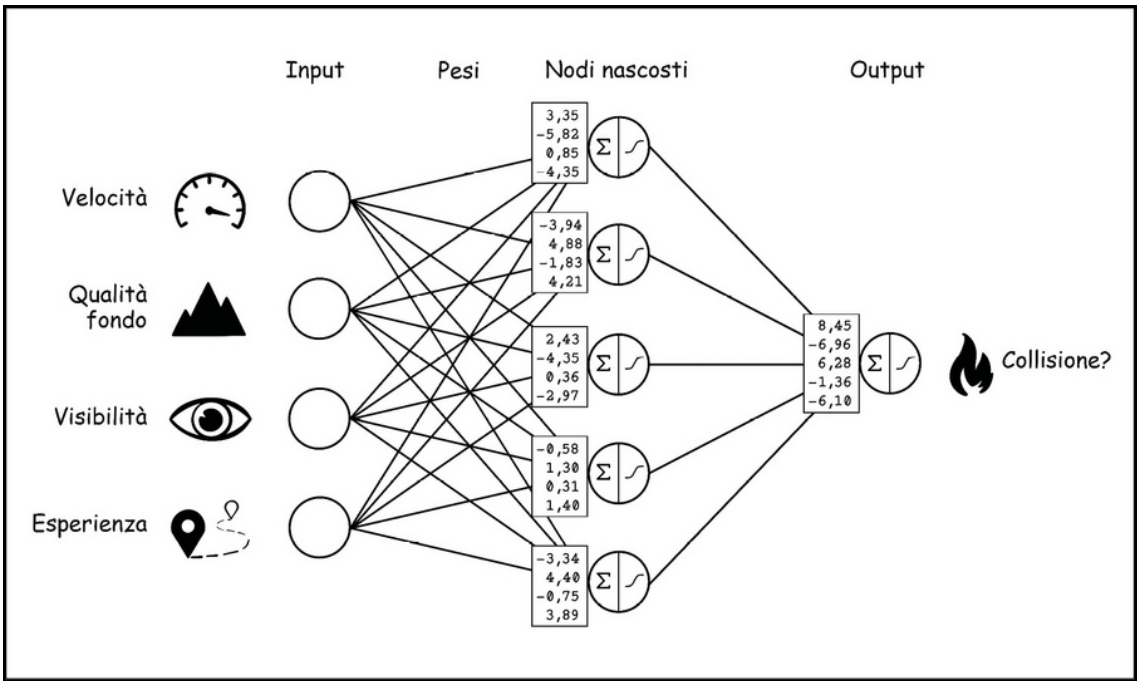


Figura 9.15 Esempio di pesi in una rete neurale artificiale pre-addestrata.

Poiché la rete neurale è stata addestrata, possiamo utilizzarla per prevedere la possibilità di collisioni fornendole un determinato

esempio. La Tabella 9.4 serve come promemoria del dataset (ridotto in scala) che stiamo utilizzando.

Tabella 9.4 Dataset degli incidenti automobilistici in scala.

	Velocità (km/h)	Qualità fondo	Visibilità	Esperienza (km)	Collisione?
1	0,542	0,5	0,5	0,200	0
2	1,000	0,1	0,2	0,275	1
3	0,067	0,6	0,8	0,125	0
4	0,417	0,2	0,9	0,004	1
5	0,208	0,9	0,1	0,400	0
6	0,667	0,3	0,3	0,015	1
	0,333	0,3	1,0	1,000	0

Se avete mai affrontato l'argomento delle reti neurali artificiali, potreste aver notato alcune notazioni matematiche oggettivamente spaventose. Analizziamo alcuni dei concetti che possono essere rappresentati matematicamente.

Gli input della rete neurale artificiale sono indicati con X. Ogni variabile di input sarà quindi X indicizzata da un numero: Xn. La velocità è X0, la qualità del fondo stradale è X1 e così via. L'output della rete è indicato con y, e i pesi della rete sono indicati con W. Poiché la rete neurale artificiale ha due livelli, un livello nascosto e un livello di output, ci sono due gruppi di pesi. Il primo gruppo è W0 e il secondo gruppo è W1. Quindi ogni peso è indicato dai nodi cui è connesso. Il peso fra il nodo Velocità e il primo nodo nascosto è W00,0 e il peso fra il nodo Qualità fondo e il primo nodo nascosto è W01,0. Queste convenzioni non sono fondamentali per comprendere questo esempio, ma la loro conoscenza vi aiuterà in futuro.

La Figura 9.16 mostra come sono rappresentati in una rete neurale artificiale i seguenti dati:

Tabella 9.5

	Velocità (km/h)	Qualità fondo	Visibilità	Esperienza (km)	Collisione?
1	0,542	0,5	0,5	0,200	0

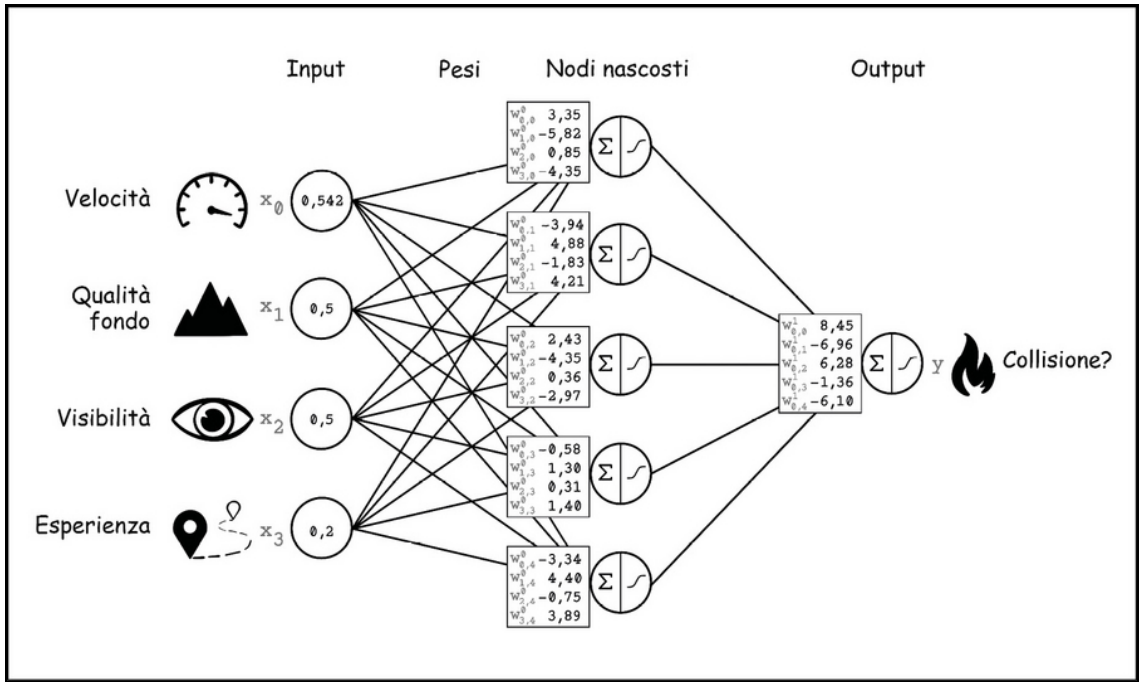


Figura 9.16 Convenzioni matematiche per una rete neurale artificiale.

Come nel caso del Perceptron, il primo passo consiste nel calcolare la somma pesata degli input e il peso di ogni nodo nascosto. Nella Figura 9.17, ogni input viene moltiplicato per ogni peso e sommato per ogni nodo nascosto.

Il passo successivo consiste nel calcolare l'attivazione di ciascun nodo nascosto. Stiamo usando la funzione sigmoide, e l'input della funzione è la somma pesata degli input calcolati per ciascun nodo nascosto (Figura 9.18).

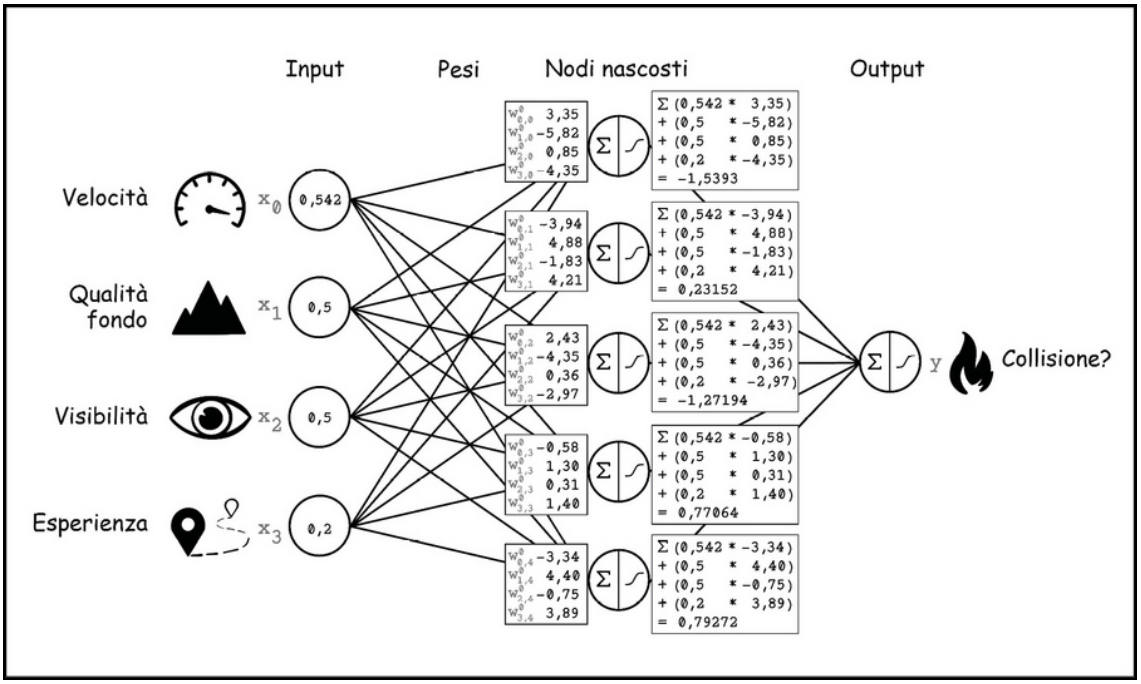


Figura 9.17 Calcolo della somma pesata per ciascun nodo nascosto.

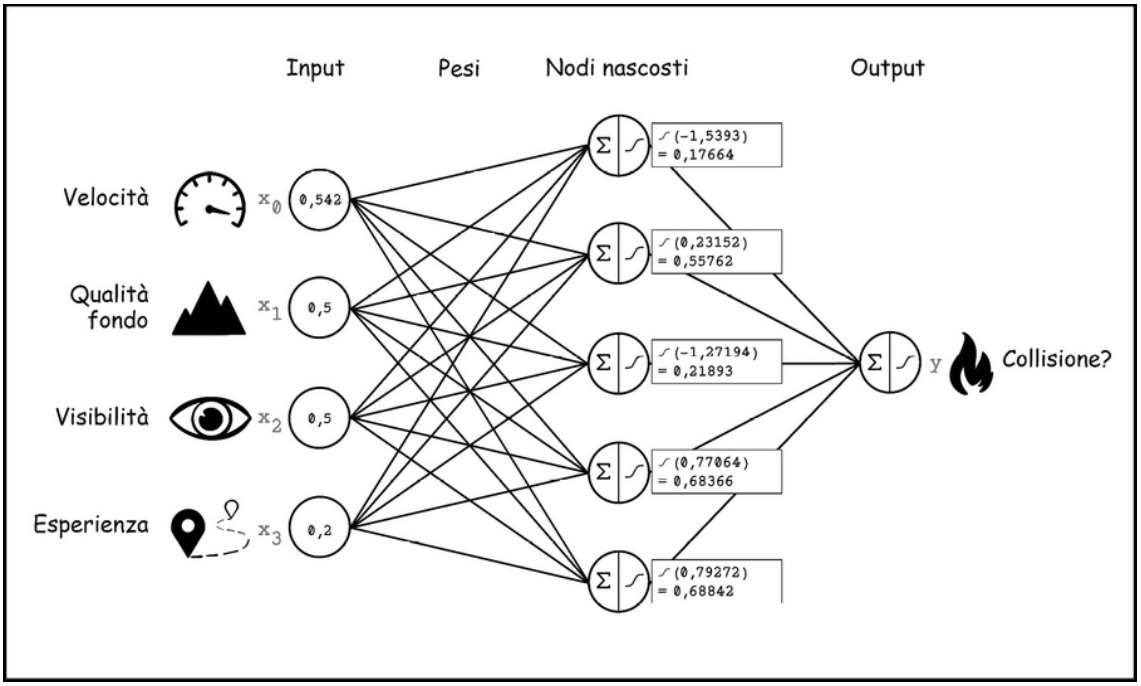


Figura 9.18 Calcolo della funzione di attivazione per ogni nodo nascosto.

Ora abbiamo i risultati dell'attivazione per ogni nodo nascosto. Se paragoniamo questo risultato ai neuroni, i risultati della funzione di attivazione rappresentano l'intensità di attivazione di ciascun neurone. Poiché nodi nascosti differenti possono essere interessati a relazioni differenti insite nei dati attraverso i pesi, le singole attivazioni possono essere utilizzate insieme per determinare un'attivazione complessiva che rappresenta la possibilità di una collisione, dati tali input.

La Figura 9.19 mostra le attivazioni per ciascun nodo nascosto e i pesi da ciascun nodo nascosto al nodo di output. Per calcolare l'output finale, ripetiamo il processo di calcolo della somma pesata dei risultati di ciascun nodo nascosto e a tale risultato applichiamo la funzione di attivazione sigmoide.

NOTA

Il simbolo sigma (Σ) nei nodi nascosti rappresenta l'operazione di sommatoria.

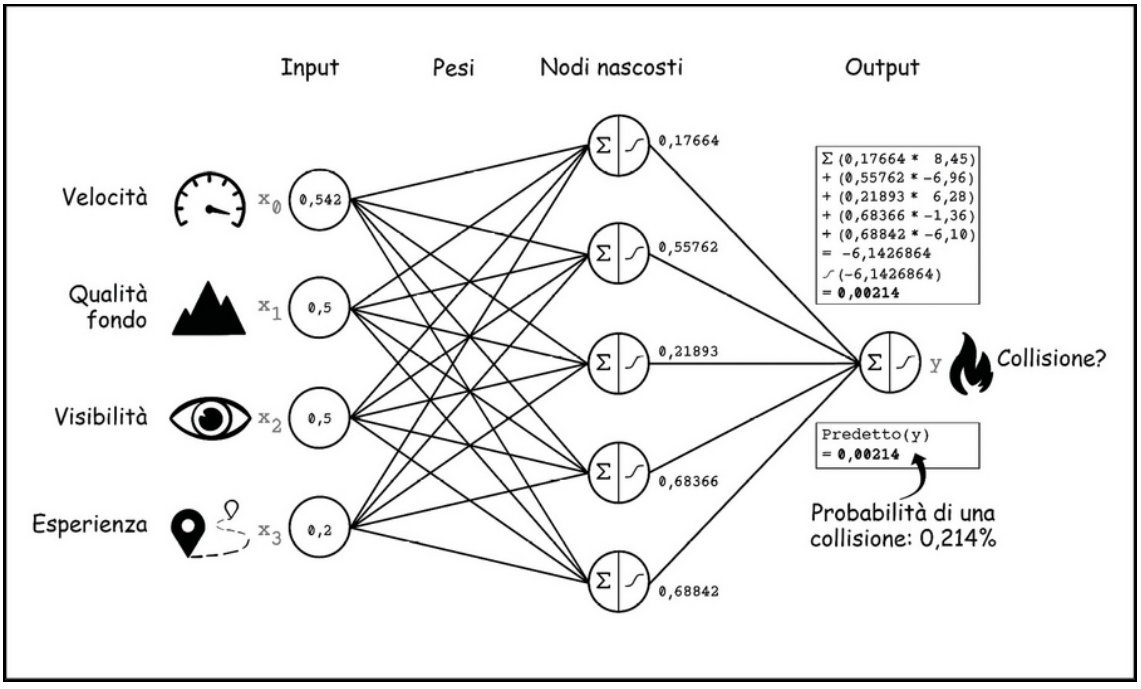


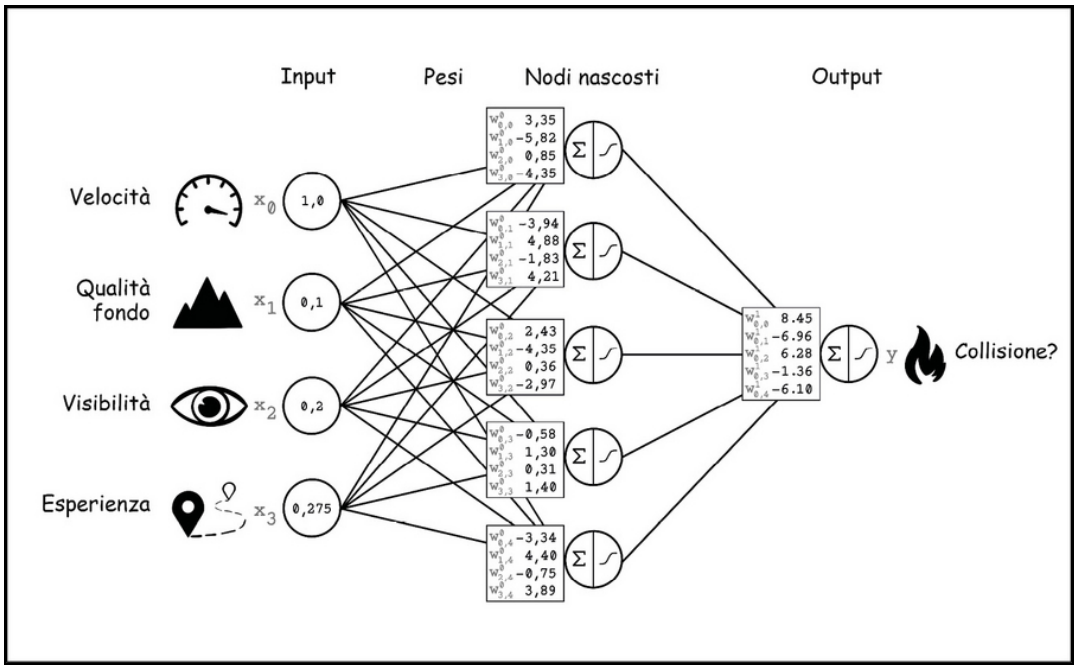
Figura 9.19 Calcolo dell'attivazione finale per il nodo di output.

Abbiamo calcolato la previsione dell'output per il nostro esempio. Il risultato è 0,00214, ma cosa significa questo numero? L'output è un valore compreso fra 0 e 1 che rappresenta la probabilità che si verifichi una collisione. In questo caso, l'output è 0,214 percento (0,00214 × 100), a indicare che la possibilità di una collisione è quasi 0.

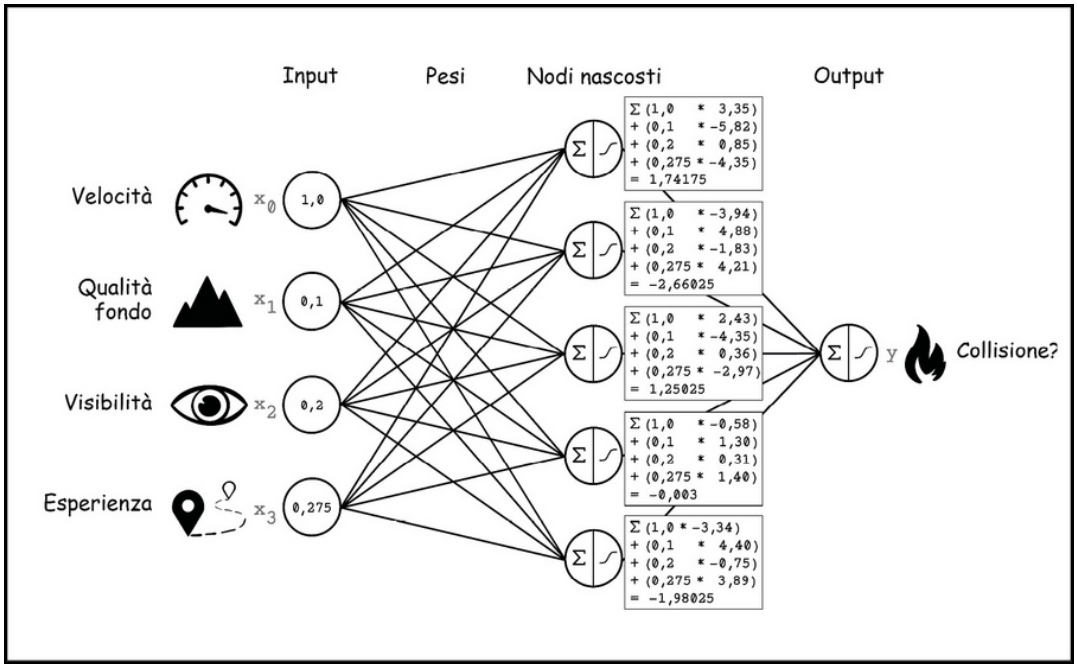
L'esercizio seguente utilizza un altro esempio sempre tratto dal dataset.

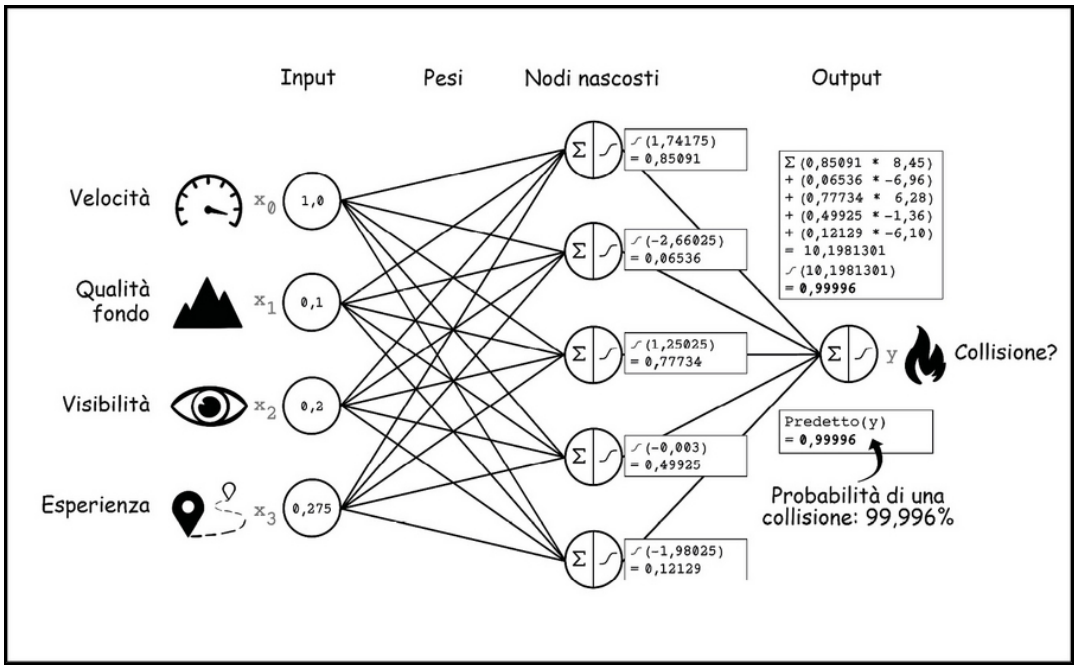
Esercizio: calcolare la previsione per l'esempio utilizzando la propagazione in avanti con la seguente rete neurale artificiale

	Velocità (km/h)	Qualità fondo	Visibilità	Esperienza (km)	Collisione?
2	1,000	0,1	0,2	0,275	1


Input
Pesi
Nodi nascosti
Output
Velocità
Qualità fondo
Visibilità
Esperienza
1,0
0,1
0,2
0,275
3,35 -5,82 0,85 -4,35
-3,94 4,88 -1,83 4,21
2,43 -4,35 0,36 -2,97
-0,58 1,30 0,31 1,40
-3,34 4,40 -0,75 3,89
8.45 -6.96 6.28 -1.36 -6.10
y
Collisione?

Soluzione


Input
Pesi
Nodi nascosti
Output
Velocità
Qualità fondo
Visibilità
Esperienza
1,0
0,1
0,2
0,275
Σ (1,0 * 3,35) + (0,1 * -5,82) + (0,2 * 0,85) + (0,275 * -4,35) = 1,74175
Σ (1,0 * -3,94) + (0,1 * 4,88) + (0,2 * -1,83) + (0,275 * 4,21) = -2,66025
Σ (1,0 * 2,43) + (0,1 * -4,35) + (0,2 * 0,36) + (0,275 * -2,97) = 1,25025
Σ (1,0 * -0,58) + (0,1 * 1,30) + (0,2 * 0,31) + (0,275 * 1,40) = -0,003
Σ (1,0 * -3,34) + (0,1 * 4,40) + (0,2 * -0,75) + (0,275 * 3,89) = -1,98025
y
Collisione?



Quando eseguiamo questo esempio attraverso la nostra rete neurale artificiale preaddestrata, l'output è 0,99996 (99,996%), quindi c'è una probabilità estremamente elevata che si verifichi effettivamente una collisione. Applicando un po' di intuizione umana a questo esempio, possiamo capire perché è probabile una collisione. Il conducente viaggiava alla massima velocità legale, su un fondo stradale in cattivo stato, con una scarsa visibilità.

Pseudocodice

Una delle funzioni più importanti per l'attivazione nel nostro esempio è la funzione sigmoide. Questo metodo descrive la funzione matematica che rappresenta la curva a "S":

```
sigmoid(x):
  return 1 / (1 + exp(-x)) (1)
```

(1) exp è una costante matematica chiamata numero di Eulero, pari a circa 2,71828.

Notate che nel codice seguente è descritta la stessa classe per rete neurale definita in precedenza nel capitolo. Questa volta, tuttavia, comprende la funzione `forward_propagation` che somma l'input e i pesi fra input e nodi nascosti, applica

la funzione sigmoide a ciascun risultato e memorizza l'output per i nodi del livello nascosto. Questo calcolo viene fatto per l'output del nodo nascosto e anche per i pesi del nodo di output:

```
NeuralNetwork(features, labels, hidden_node_count):
  let input equal features
     let weights_input equal a random matrix, size: features *
hidden_node_count
  let hidden equal zero array, size: hidden_node_count
  let weights_hidden equal a random matrix, size: hidden_node_count
  let expected_output equal labels
  let output equal zero array, size: length of labels
forward_propagation():
  let hidden_weighted_sum equal input • weights_input
  let hidden equal sigmoid(hidden_weighted_sum) (1)
  let output_weighted_sum equal hidden • weights_hidden
  let output equal sigmoid(output_weighted_sum)
```

(1) Il simbolo • rappresenta la moltiplicazione fra matrici.

Propagazione all'indietro: addestramento di una rete neurale artificiale

Capire come funziona la propagazione in avanti è utile per capire come vengono addestrate le reti neurali artificiali, perché la propagazione in avanti viene utilizzata nel processo di addestramento. Il ciclo di vita del machine learning e i principi trattati nel Capitolo 8 sono importanti per affrontare la propagazione all'indietro (retropropagazione) nelle reti neurali artificiali. Una rete neurale artificiale può essere considerata come un modello di machine learning. Dobbiamo però ancora trovare la domanda da porre. Stiamo ancora raccogliendo e comprendendo i dati nel contesto del problema e dobbiamo prepararli in un modo adatto all'elaborazione svolta dal modello.

Abbiamo bisogno di dotarci di un sottoinsieme di dati per l'addestramento e di un sottoinsieme di dati per poi sottoporre a test le prestazioni del modello. Inoltre, eseguiremo iterazioni e miglioreremo

il modello raccogliendo più dati, preparandoli in modo differente o modificando l'architettura e la configurazione della rete neurale artificiale.

L'addestramento di una rete neurale artificiale si compone di tre fasi. La Fase A prevede la configurazione dell'architettura a rete neurale artificiale, compresa la configurazione degli input, degli output e dei livelli nascosti. La Fase B è la propagazione in avanti. La Fase C è la retropropagazione, dove avviene, in effetti, l'addestramento (Figura 9.20).

Queste sono le fasi e le operazioni coinvolte nell'algoritmo di retropropagazione.

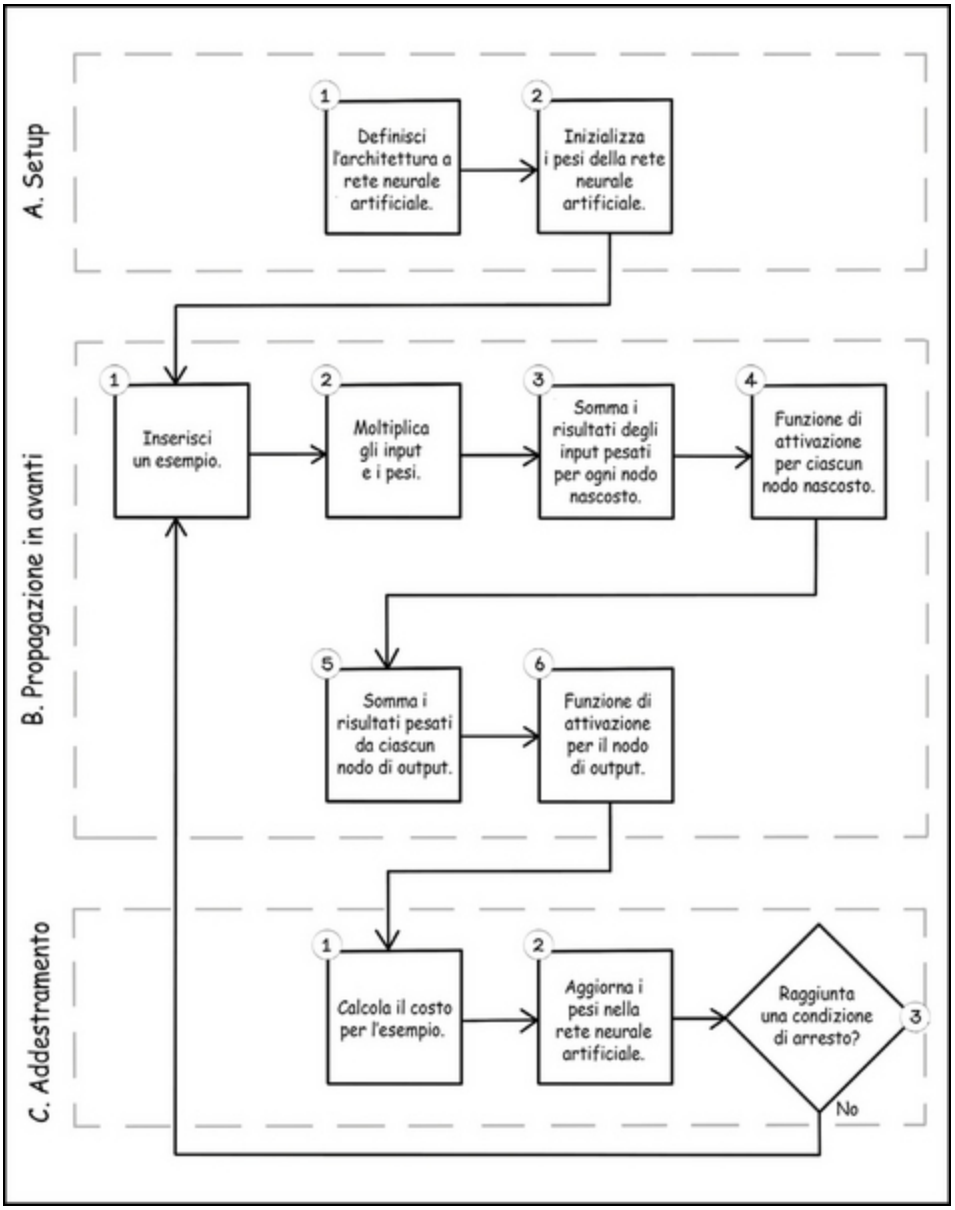


Figura 9.20 Ciclo di vita dell'addestramento di una rete neurale artificiale.

Fase A: setup

1. *Definisci l'architettura a rete neurale artificiale*. Questo passaggio comporta la definizione dei nodi di input, dei nodi di output, del

numero di livelli nascosti, del numero di neuroni in ogni livello nascosto, delle funzioni di attivazione utilizzate e altro.
2. *Inizializza i pesi della rete neurale artificiale*. I pesi nella rete neurale artificiale devono essere inizializzati con un valore. Possiamo adottare vari approcci. Il principio chiave è che i pesi verranno regolati costantemente a mano a mano che la rete neurale artificiale apprenderà dagli esempi.

Fase B: propagazione in avanti

Questo processo è lo stesso che abbiamo trattato in precedenza. Vengono eseguiti gli stessi calcoli. L'output previsto, tuttavia, verrà confrontato con la classe effettiva di ogni esempio presente nel set di addestramento della rete.

Fase C: addestramento

1. *Calcola il costo*. A seguito della propagazione in avanti, il costo è la differenza fra l'output previsto e la classe effettiva per gli esempi presenti nel set di addestramento. Il costo determina in modo efficace la bontà della rete neurale artificiale nel prevedere la classe degli esempi.
2. *Aggiorna i pesi nella rete neurale artificiale*. I pesi della rete neurale artificiale sono le uniche cose che possono essere regolate. L'architettura e le configurazioni che abbiamo definito nella Fase A non cambiano durante l'addestramento della rete. I pesi, sostanzialmente, codificano l'intelligenza della rete. I pesi vengono pertanto regolati (aumentati o ridotti) per influenzare il "peso" degli input.
3. *Definisci una condizione di arresto*. L'addestramento non può procedere all'infinito. Come per molti degli algoritmi esaminati in

questo libro, è necessario determinare una condizione di arresto ragionevole. Se disponiamo di un dataset di grandi dimensioni, potremmo decidere di addestrare la rete neurale artificiale utilizzando 500 esempi del nostro dataset di addestramento su 1000 iterazioni. In questo esempio, i 500 esempi verranno esaminati dalla rete per 1.000 volte e a ogni iterazione i pesi verranno adeguati.

Quando abbiamo lavorato alla propagazione in avanti, i pesi erano già definiti, perché la rete era già stata addestrata. Prima di iniziare ad addestrare la rete, dobbiamo inizializzare i pesi con un certo valore, e i pesi devono essere regolati in base agli esempi utilizzati per l'addestramento. Un approccio all'inizializzazione dei pesi consiste nello scegliere pesi casuali in base a una distribuzione normale.

La Figura 9.21 illustra i pesi generati casualmente per la nostra rete neurale artificiale. Mostra anche i calcoli per la propagazione in avanti per i nodi nascosti, per un singolo esempio di addestramento. Il primo input di esempio utilizzato nella sezione di propagazione in avanti viene utilizzato per evidenziare le differenze nell'output, dati i diversi pesi nella rete.

Il passo successivo consiste nella propagazione in avanti (Figura 9.22). L'elemento chiave consiste nel verificare la differenza fra la previsione ottenuta e la classe effettiva.

Confrontando il risultato previsto con la classe effettiva, possiamo calcolare un costo. La funzione di costo che useremo è semplice: sottrarre l'output previsto dall'output effettivo. In questo esempio, sottraiamo 0,84274 da 0,0 e il costo è -0,84274. Questo risultato indica quanto fosse errata la previsione e può essere utilizzato per aggiustare i pesi nella rete neurale artificiale. I pesi vengono modificati leggermente ogni volta che viene calcolato un costo. Ciò si ripete per migliaia di volte, utilizzando i dati di addestramento per determinare i pesi

ottimali per la rete neurale artificiale per fare previsioni sempre più accurate. Notate che un addestramento troppo lungo sullo stesso dataset può portare a un overfitting, problema descritto nel Capitolo 8.

È qui che entra in gioco un'operazione matematica che forse non conoscete: la regola della catena. Prima di utilizzare la regola della catena, cerchiamo di capire il significato dei pesi e come la loro regolazione migliori le prestazioni della rete neurale artificiale.

Se tracciamo su un grafico i possibili pesi rispetto al loro rispettivo costo, troviamo una funzione che rappresenta i possibili pesi. Alcuni punti sulla funzione producono un costo inferiore e altri producono un costo maggiore. Cerchiamo i punti che minimizzano i costi (Figura 9.23).

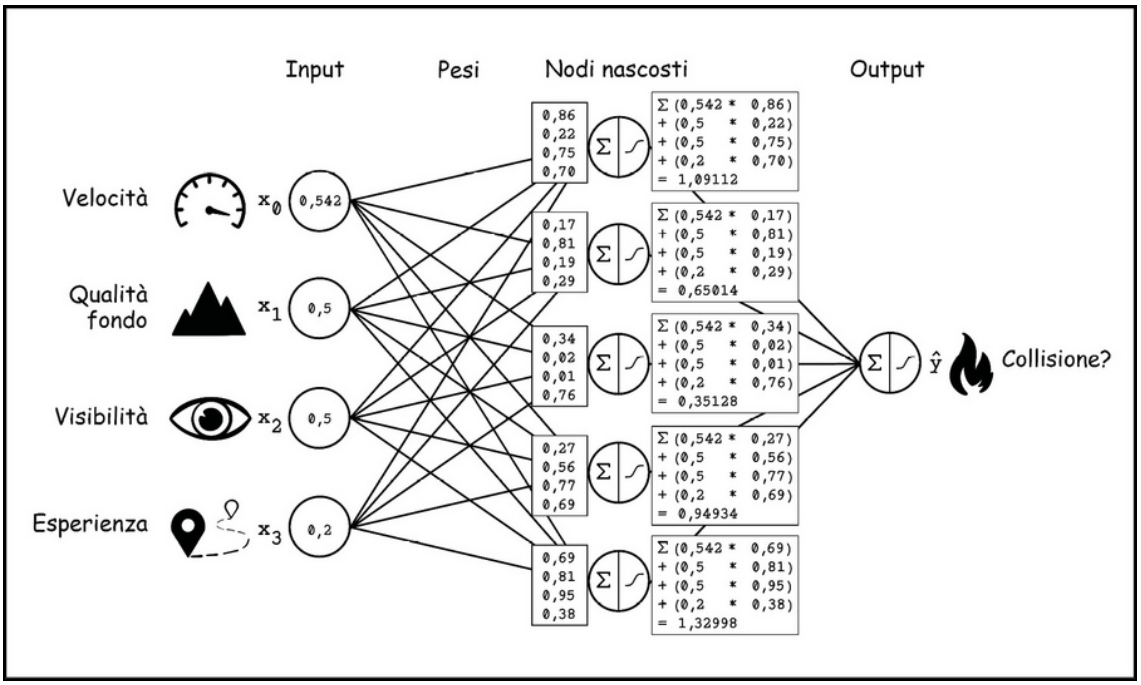


Figura 9.21 Esempio di pesi iniziali per una rete neurale artificiale.

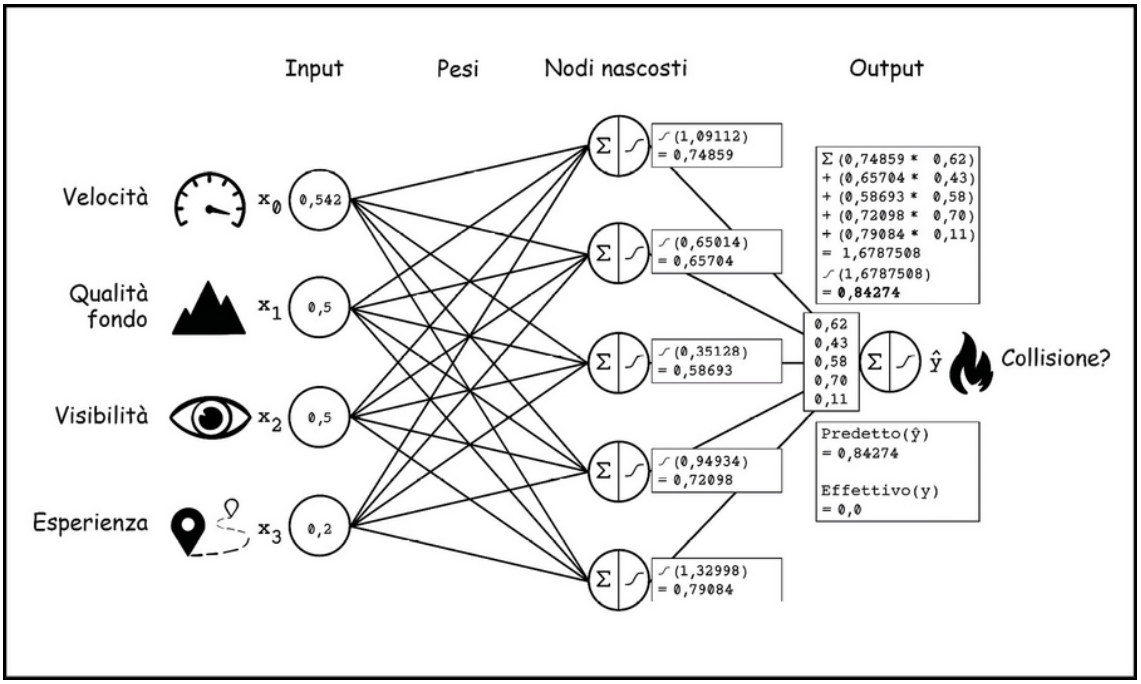


Figura 9.22 Esempio di propagazione in avanti con pesi inizializzati casualmente.

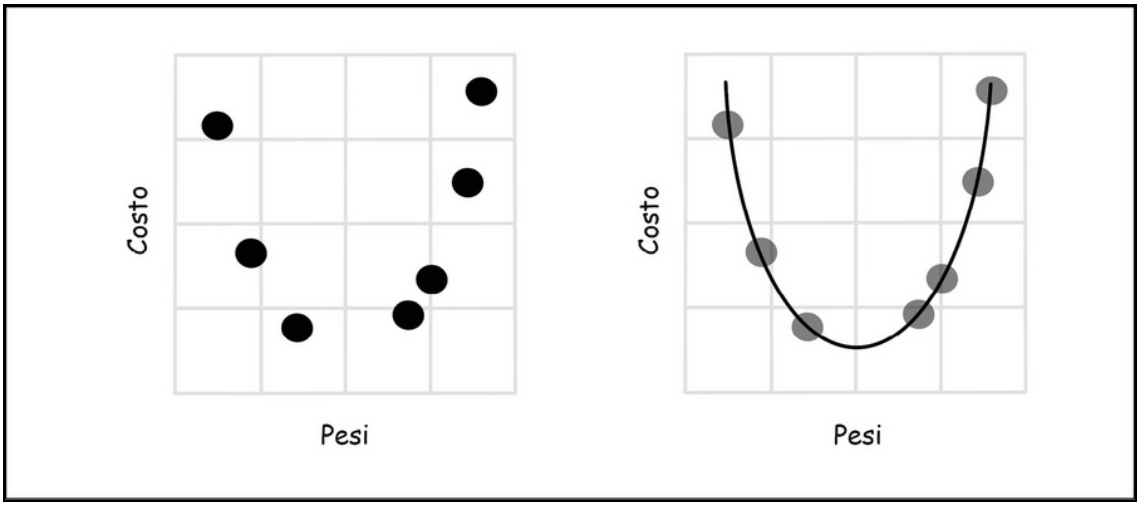


Figura 9.23 Grafico dei pesi rispetto al costo.

Uno strumento utile nel campo del calcolo, chiamato *gradiente discendente*, può aiutarci ad avvicinare i pesi al valore minimo trovando la derivata. La *derivata* è importante, perché misura la sensibilità al cambiamento per quella funzione. Per esempio, la velocità

potrebbe essere considerata la derivata della posizione di un oggetto rispetto al tempo; e l'accelerazione è la derivata della velocità dell'oggetto rispetto al tempo. Le derivate possono trovare la pendenza in un determinato punto della funzione. La discesa del gradiente utilizza la conoscenza della pendenza per determinare in che direzione muoversi e di quanto. Le Figure 9.24 e 9.25 descrivono come le derivate e la pendenza indicano la direzione dei minimi.

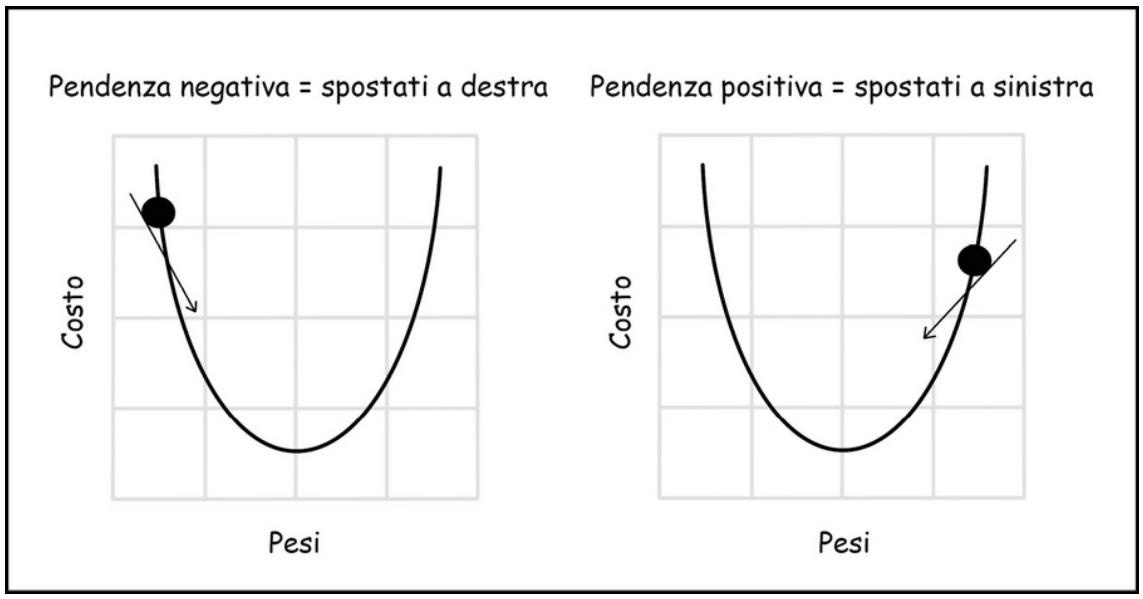


Figura 9.24 Pendenza delle derivate e direzione dei minimi.

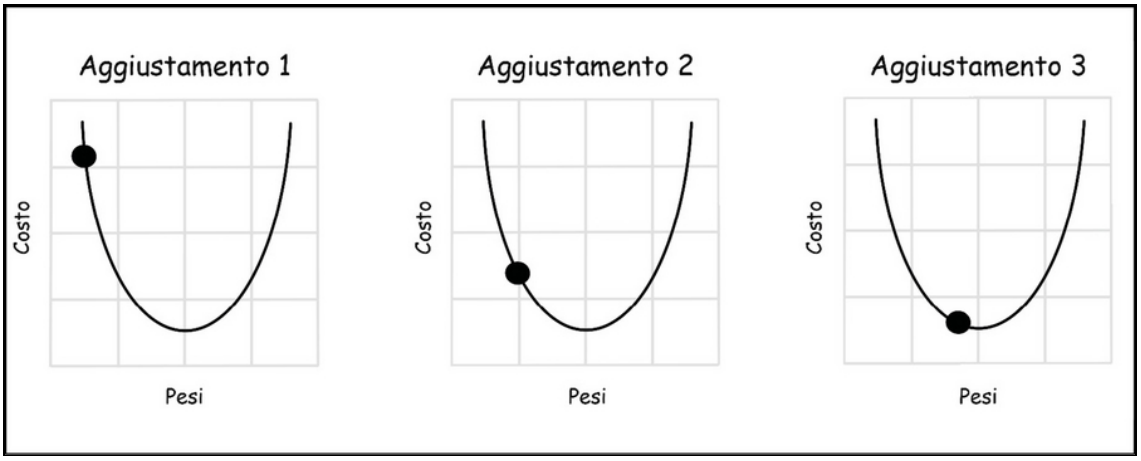


Figura 9.25 Esempio di regolazione di un peso utilizzando la discesa del gradiente.

Quando esaminiamo un peso isolatamente, può sembrare banale trovare un valore che riduca al minimo il costo, ma il bilanciamento di molti pesi influisce sul costo complessivo della rete. Alcuni pesi potrebbero essere già vicini ai loro punti ottimali nell'obiettivo di ridurre i costi e altri potrebbero non esserlo, anche se la rete neurale artificiale si comporta già molto bene.

Poiché sono molte le funzioni che controllano la rete neurale artificiale, possiamo usare la regola della catena, un teorema che calcola la derivata di una funzione composta. Una funzione composta utilizza una funzione `g` come parametro per una funzione `f` per produrre una funzione `h`: in pratica utilizza una funzione come parametro di un'altra funzione.

La Figura 9.26 illustra l'uso della regola della catena nel calcolo del valore di aggiornamento per i pesi nei diversi livelli della rete neurale artificiale.

Possiamo calcolare l'aggiornamento del peso inserendo i rispettivi valori nella formula descritta. I calcoli sembrano complessi, ma prestate attenzione alle variabili utilizzate e al loro ruolo nella rete

neurale artificiale. Sebbene la formula sembri complessa, utilizza i valori che abbiamo già calcolato (Figura 9.27).

Ecco uno sguardo più da vicino ai calcoli utilizzati nella Figura 9.27:

Calcola l'aggiustamento dei pesi fra i nodi di input e i nodi nascosti:

```
input * (2 * costo * derivata_sigmoide(output) * nascosto) *
derivata_sigmoide(nascosto)
0,542 * (2 * -0,84274 * derivata_sigmoide(0,84274) * 0,86) *
derivata_sigmoide(0,74859)
= 0,542 * (2 * -0,84274 * 0,210 * 0,86) * 0,218
= -0,0360
```

Calcola l'aggiustamento dei pesi fra i nodi nascosti e il nodo di output:

```
nascosto * (2 * costo * derivata_sigmoide(output))
0,74859 * (2 * -0,84274 * derivata_sigmoide(0,84274))
= 0,74859 * (2 * -0,84274 * 0,210)
= -0,265
```

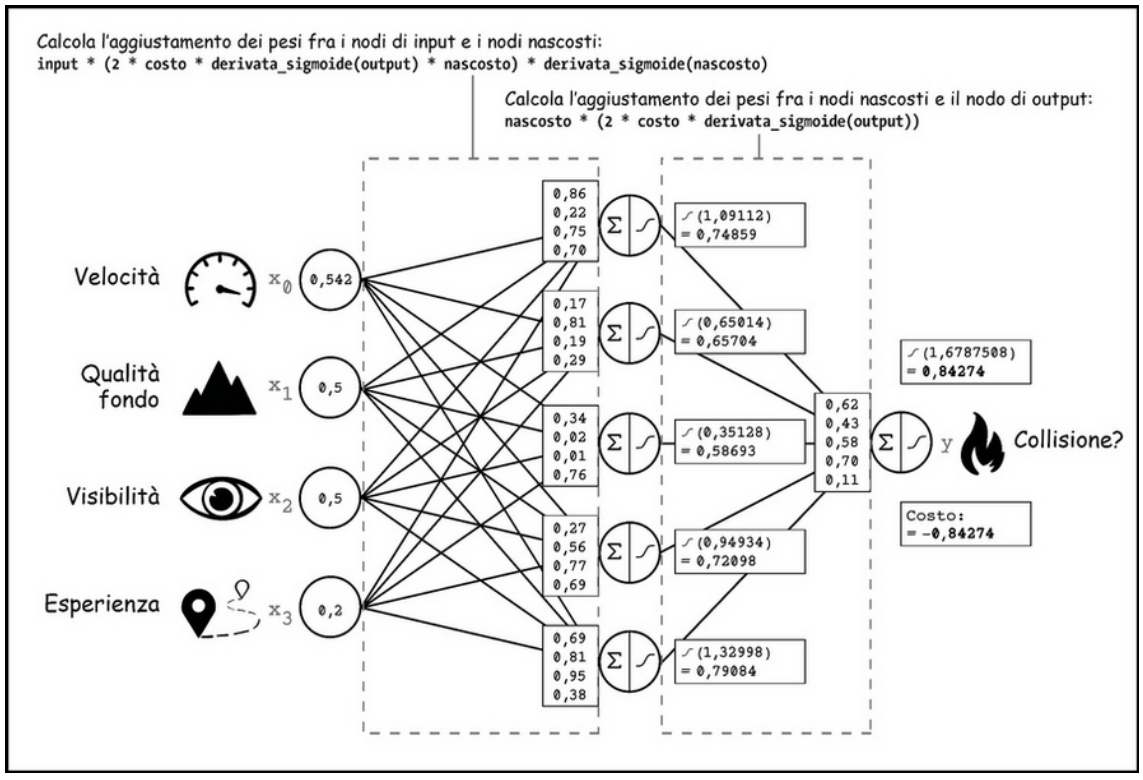


Figura 9.26 Formula per il calcolo degli aggiustamenti dei pesi con la regola della catena.

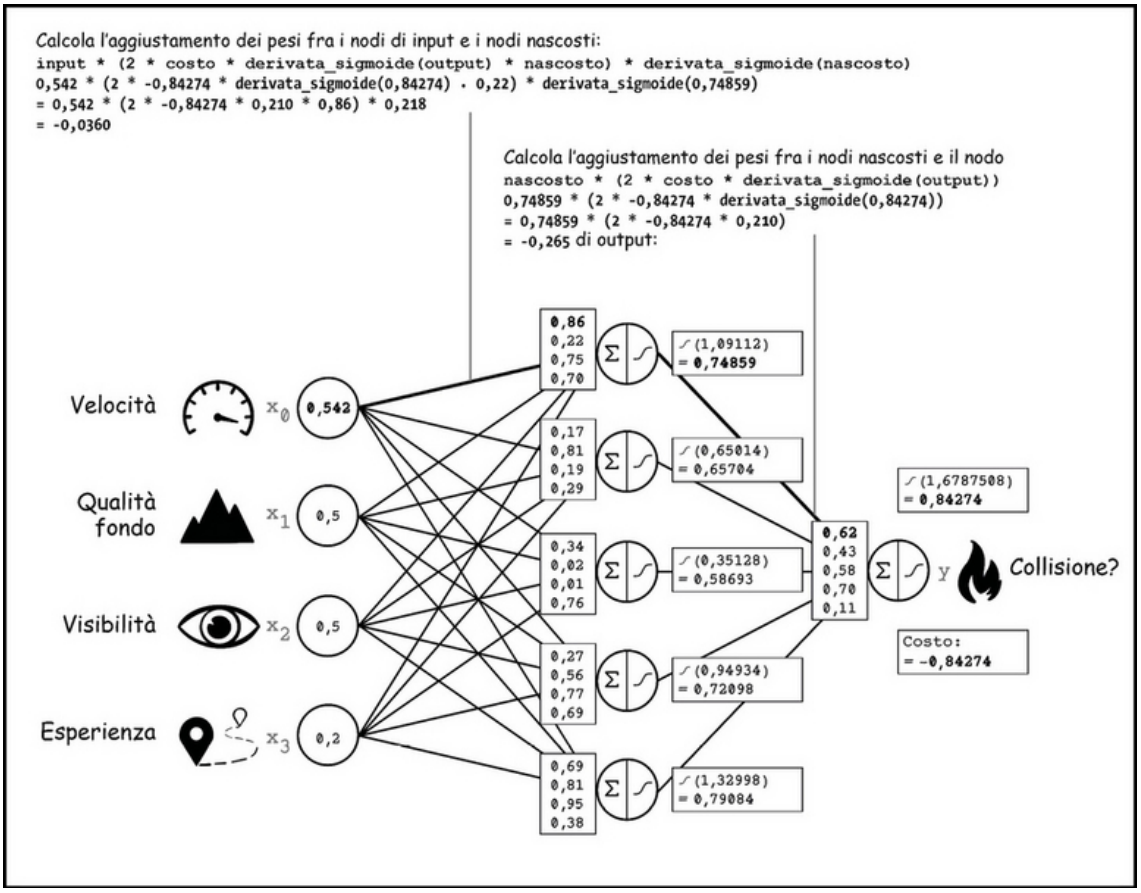


Figura 9.27 Calcolo dell'aggiornamento del peso con la regola della catena.

Ora che i valori di aggiornamento sono stati calcolati, possiamo applicare i risultati ai pesi nella nostra rete neurale artificiale, sommando il valore di aggiustamento calcolato ai rispettivi pesi.

La Figura 9.28 illustra l'applicazione dei risultati dell'aggiustamento dei pesi nei diversi livelli.

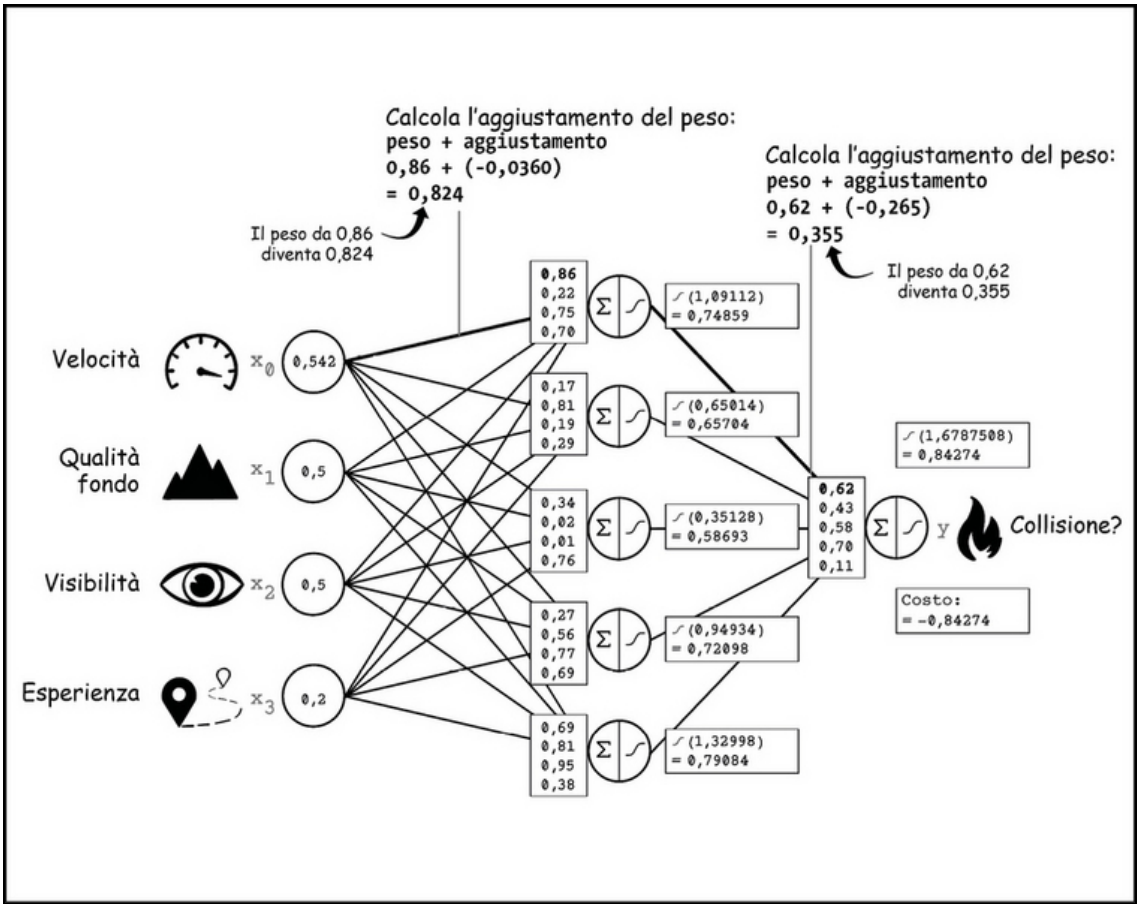


Figura 9.28 Esempio dell'aggiustamento dei pesi per la rete neurale artificiale.

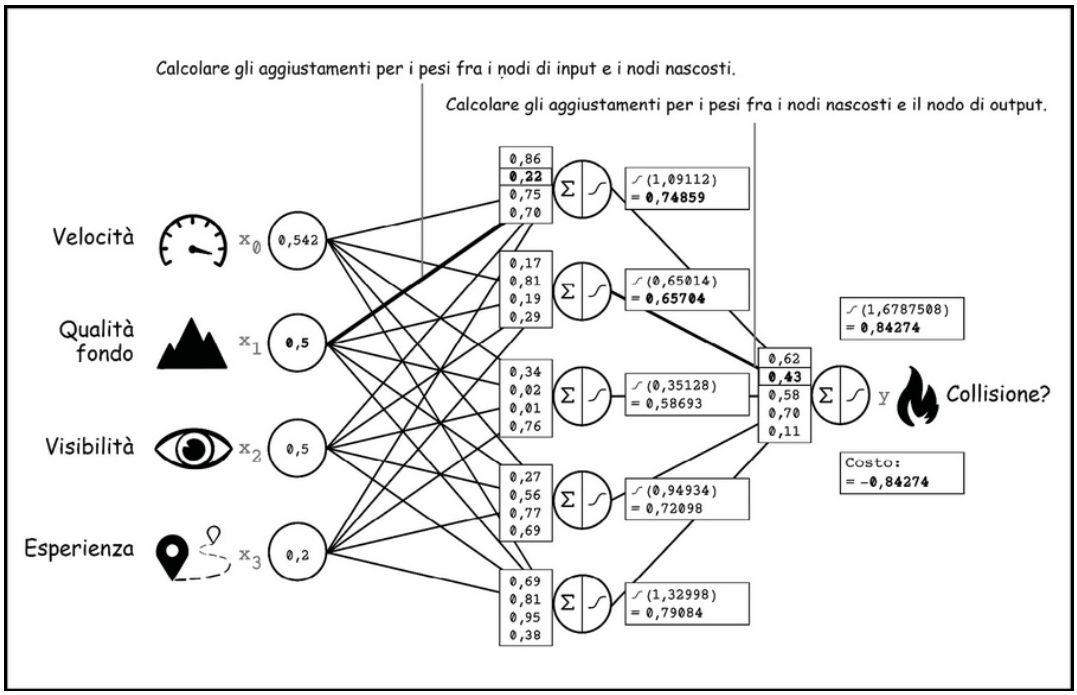
Il problema risolto dalla regola della catena potrebbe ricordarvi l'esempio del problema del drone nel Capitolo 7.

L'ottimizzazione a sciame di particelle può essere molto efficace quando si tratta di trovare valori ottimali in spazi a elevata dimensionalità, come questo, che di pesi da ottimizzare ha ben 25.

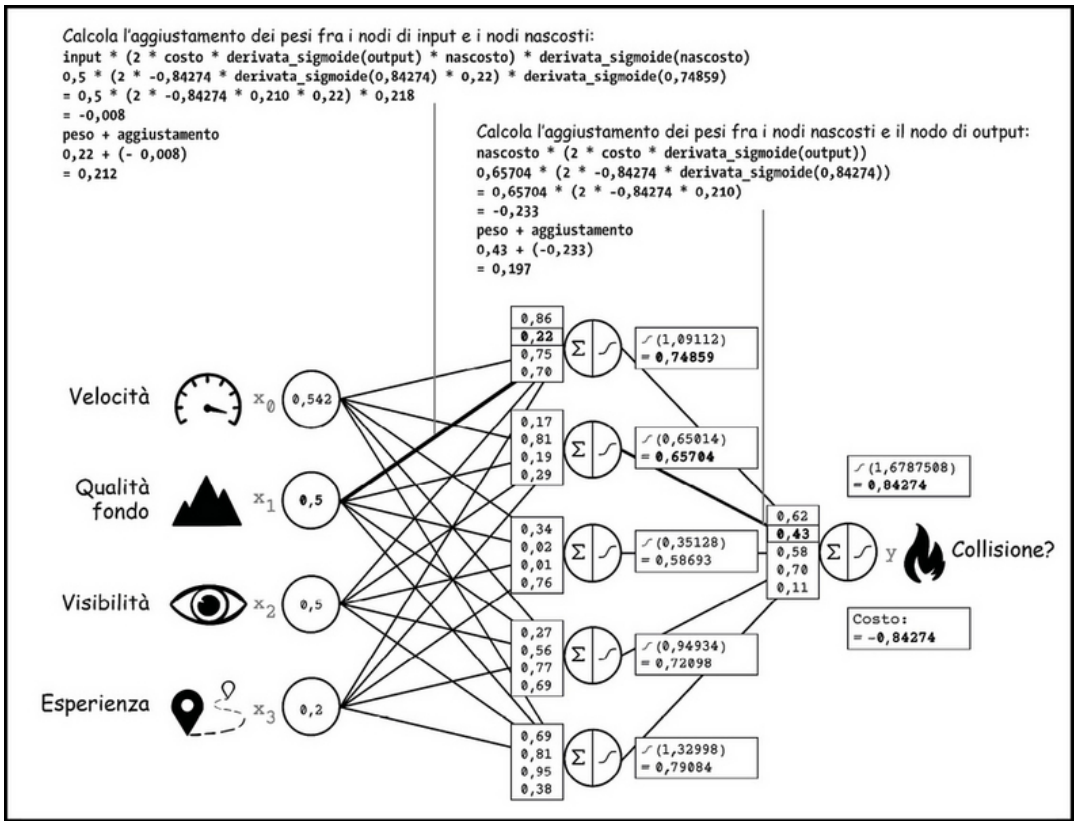
Calcolare i pesi corretti in una rete neurale artificiale è, in effetti, un problema di ottimizzazione.

La discesa del gradiente non è l'unico modo per ottimizzare i pesi; possiamo utilizzare molti approcci, a seconda del contesto e del problema da risolvere.

Esercizio: calcolare i nuovi pesi per i pesi evidenziati



Soluzione



Pseudocodice

La derivata è importante nell'algoritmo di retropropagazione. Il seguente frammento di pseudocodice rielabora la funzione sigmoide e descrive la formula per la sua derivata, di cui abbiamo bisogno per aggiustare i pesi:

```
sigmoid(x):
  return 1 / (1 + exp(-x)) (1)
sigmoid_derivative(x):
  return sigmoid(x) * (1 - sigmoid(x))
```

(1) exp è una costante matematica chiamata numero di Eulero, pari a circa 2,71828.

Rivisitiamo la classe della rete neurale, questa volta con una funzione di retropropagazione che calcola il costo, calcola l'entità di aggiustamento dei pesi utilizzando la regola della catena e aggiunge i risultati ai pesi esistenti. Questo processo calcolerà la variazione per ogni peso, in base al costo. Ricordate che il costo viene calcolato utilizzando le caratteristiche dell'esempio, l'output previsto e l'output effettivo. La differenza fra l'output previsto e l'output effettivo è il costo:

```
NeuralNetwork(features, labels, hidden_node_count):
  let input equal features
     let   weights_input   equal   a   random   matrix,   size:   features   *
hidden_node_count
  let hidden equal zero array, size: hidden_node_count
  let weights_hidden equal a random matrix, size: hidden_node_count
  let expected_output equal labels
  let output equal zero array, size: length of labels
back_propagation():
  let cost equal expected_output - output
  let weights_hidden_update equal hidden • (1)
          (2 * cost * sigmoid_derivative(output))
  let weights_input_update equal input • (1)
          (2 * cost * sigmoid_derivative(output) *
         weights_hidden) * sigmoid_derivative(hidden)
  let weights_hidden equal weights_hidden + weights_hidden_update
  let weights_input equal weights_input + weights_input_update
```

(1) Il simbolo • rappresenta la moltiplicazione fra matrici.

Poiché abbiamo una classe che rappresenta una rete neurale, le funzioni per aggiustare i dati e le funzioni per la propagazione in avanti e all'indietro, possiamo mettere insieme tutto questo codice per addestrare una rete neurale.

Pseudocodice

In questo frammento di pseudocodice, abbiamo una funzione `run_neural_network` che accetta come input `epochs`. Questa funzione aggiusta i dati e crea una nuova rete neurale con i dati ridimensionati, le etichette e il numero di nodi nascosti. Quindi la funzione esegue `forward_propagation` e `back_propagation` per il numero specificato di `epochs`:

```
run_neural_network(epochs):
  let scaled_feature_data equal scale_dataset(feature_data, feature_count,
          features_min, features_max)
     let   nn   equal   NeuralNetwork(scaled_feature_data,   scaled_label_data,
hidden_node_count)
  for epoch in range(epochs):
      nn.forward_propagation()
      nn.back_propagation()
```

Opzioni per le funzioni di attivazione

Questo paragrafo si propone di fornire alcune idee sulle funzioni di attivazione e le loro proprietà. Negli esempi del Perceptron e delle reti neurali artificiali, abbiamo usato come funzione di attivazione la sigmoide, che era soddisfacente per gli esempi con i quali stavamo

lavorando. Le funzioni di attivazione introducono proprietà non lineari nella rete neurale artificiale. Se non introduciamo una funzione di attivazione, la rete neurale si comporterà in modo simile alla regressione lineare, descritta nel Capitolo 8. La Figura 9.29 descrive alcune funzioni di attivazione comunemente usate.

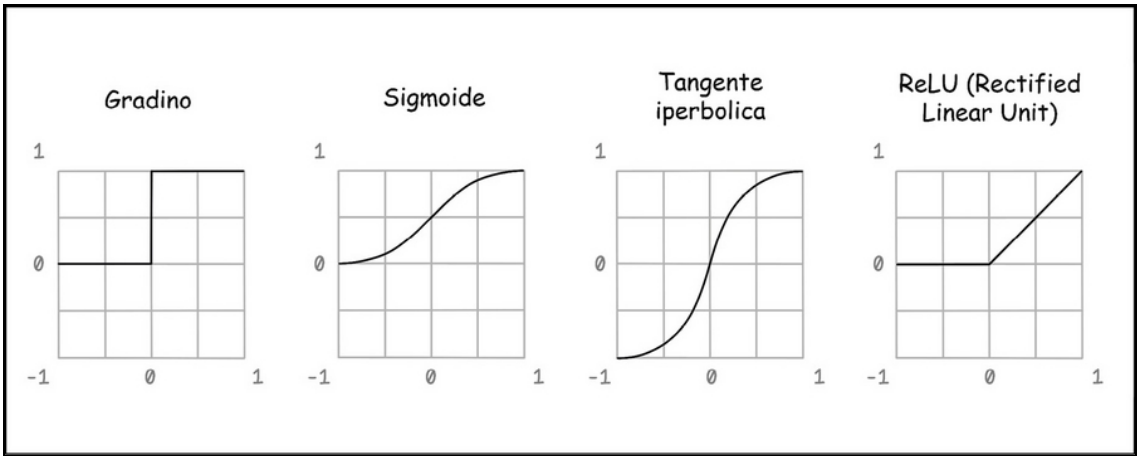


Figura 9.29 Funzioni di attivazione comunemente utilizzate.

Funzioni di attivazione differenti sono utili in scenari differenti con vantaggi differenti.

- *Gradino*: viene utilizzata per la classificazione binaria. Dato un input compreso fra -1 e 1, restituisce un risultato esattamente 0 o 1. Un classificatore binario non è utile per apprendere dai dati in un livello nascosto, ma può essere utilizzato nel livello di output per la classificazione binaria. Se vogliamo sapere se qualcosa è un gatto o un cane, per esempio, 0 potrebbe indicare “gatto”, e 1 potrebbe indicare “cane”.
- *Sigmoide*: genera una curva a “S” compresa fra 0 e 1, dato un input compreso fra -1 e 1. Poiché la funzione sigmoide consente ai cambiamenti in x di provocare piccoli cambiamenti in y, è adatta all’apprendimento e alla risoluzione di problemi non lineari. Il problema a volte riscontrato con la funzione sigmoide è

che quando i valori si avvicinano agli estremi, i cambiamenti delle derivate diventano minuscoli, con conseguente scarso apprendimento. Questo problema è noto come *problema della scomparsa del gradiente*.

- *Tangente iperbolica*: è simile alla funzione sigmoide, ma restituisce valori compresi fra -1 e 1. Il vantaggio è che la tangente iperbolica ha derivate più ripide, il che offre un apprendimento più rapido. Anche per questa funzione, come per la funzione sigmoide, si presenta il problema della scomparsa del gradiente agli estremi.
- *ReLU* (*Rectified Linear Unit*): restituisce 0 per valori di input compresi fra -1 e 0 e genera valori linearmente crescenti fra 0 e 1. In una rete neurale artificiale estesa, con molti neuroni che utilizzano la funzione sigmoide o tangente iperbolica, tutti i neuroni si attivano sempre (tranne quando danno come risultato 0), producendo molti calcoli e molti valori affinati per trovare soluzioni. La funzione ReLU fa in modo che alcuni neuroni non si attivino, il che riduce i calcoli e aiuta a trovare più velocemente le soluzioni.

Il prossimo paragrafo espone alcune considerazioni relative alla progettazione di una rete neurale artificiale.

Progettazione di reti neurali artificiali

La progettazione di reti neurali artificiali è del tutto sperimentale e dipende dal problema da risolvere. L'architettura e la configurazione di una rete neurale artificiale di solito cambiano attraverso un processo di tentativi ed errori, mentre tentiamo di migliorare le prestazioni delle previsioni. Questo paragrafo elenca brevemente i parametri dell'architettura che possiamo modificare per migliorare le prestazioni

o per risolvere vari tipi di problemi. La Figura 9.30 rappresenta una rete neurale artificiale con una configurazione diversa da quella vista in questo capitolo. La differenza più notevole è l'introduzione di un nuovo livello nascosto: ora la rete ha due output.

NOTA

Come nella maggior parte dei problemi scientifici o ingegneristici, la risposta alla domanda "Qual è il progetto ideale per una rete neurale artificiale?" spesso è: "Dipende". La configurazione delle reti neurali artificiali richiede una profonda conoscenza dei dati e del problema da risolvere. Non esiste un progetto generalizzato e universale per le architetture e le configurazioni. O almeno non esiste ancora….

Input e output

Gli input e gli output di una rete neurale artificiale sono i parametri fondamentali per l'utilizzo della rete. Dopo che un modello di rete neurale artificiale è stato addestrato, tale modello sarà potenzialmente utilizzabile in contesti e sistemi differenti e da persone differenti. Gli input e gli output definiscono l'interfaccia della rete. In questo capitolo abbiamo visto un esempio di una rete neurale artificiale con quattro input che descrivono le caratteristiche di vari scenari di guida in auto e un output che descrive la probabilità di una collisione. Tuttavia, potremmo avere un problema quando gli input e gli output significano cose differenti.

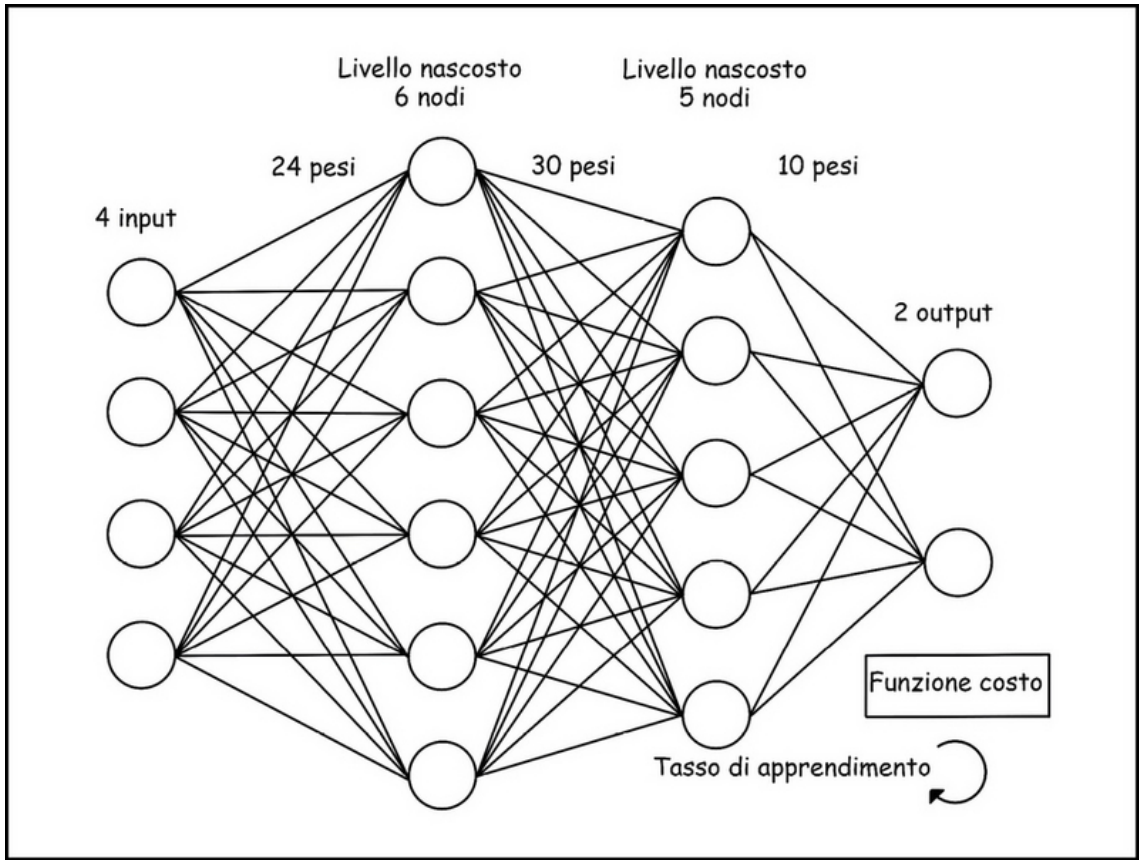


Figura 9.30 Un esempio di una rete neurale artificiale multilivello con più di un output.

Se abbiamo un'immagine di 16 × 16 pixel che rappresenta una cifra scritta a mano, per esempio, potremmo usare come input i pixel e come output la cifra che rappresentano. L'input consisterebbe in 256 nodi che rappresentano i valori dei pixel; l'output consisterebbe in 10 nodi che rappresentano le cifre da 0 a 9; ciascun risultato indica la probabilità che l'immagine sia la rispettiva cifra.

Livelli e nodi nascosti

Una rete neurale artificiale può essere costituita da più livelli nascosti con un numero variabile di nodi in ogni livello. L'aggiunta di più livelli nascosti ci consente di risolvere problemi di maggiore

dimensionalità e complessità nel trovare la linea di discriminazione della classificazione. Nell'esempio della Figura 9.8, una semplice linea retta classificava accuratamente i dati. A volte, la curva non è lineare ma è abbastanza semplice. Ma che cosa succede quando la linea è una funzione più complessa, con molte curve che attraversano più dimensioni (che non possiamo nemmeno rappresentare)? L'aggiunta di più livelli consente di trovare queste complesse funzioni di classificazione. La selezione del numero di livelli e nodi di una rete neurale artificiale di solito comporta la sperimentazione e miglioramenti iterativi. Nel corso del tempo, potremmo acquisire conoscenze specifiche sulle configurazioni adeguate, sulla base dell'esperienza con problemi simili e della loro risoluzione con configurazioni simili.

Pesi

L'inizializzazione dei pesi è importante, perché stabilisce un punto di partenza dal quale i pesi verranno alterati nelle molteplici iterazioni. Se i pesi vengono inizializzati troppo in basso, possono causare il problema della sparizione del gradiente descritto in precedenza, mentre se i pesi vengono inizializzati troppo in alto, sorge un altro problema, il problema del gradiente *esplosivo*, in cui i pesi "altalenano" attorno al miglior risultato.

Esistono vari schemi di inizializzazione dei pesi, ciascuno con i propri pro e contro. Una regola empirica consiste nel garantire che la media dei risultati di attivazione in un livello (la media di tutti i risultati dei nodi nascosti in un livello) sia 0. Inoltre, la varianza dei risultati di attivazione dovrebbe essere la stessa: la variabilità dei risultati di ciascun nodo nascosto dovrebbe essere coerente nelle diverse iterazioni.

Pregiudizi

Possiamo introdurre dei pregiudizi (bias) in una rete neurale artificiale aggiungendo un valore alla somma pesata dei nodi di input o di altri livelli nella rete. Un bias può spostare il valore della funzione di attivazione e quindi dona flessibilità a una rete neurale artificiale, spostando la funzione di attivazione a sinistra o a destra.

Un modo semplice per comprendere il bias è immaginare una linea che passa sempre per 0, 0 su un piano; possiamo influenzare questa linea, in modo che intercetti l'asse y in un altro punto aggiungendo, per esempio, +1 a una variabile. Questo valore dipende ovviamente dal problema da risolvere.

Funzioni di attivazione

In precedenza, abbiamo trattato le più comuni funzioni di attivazione utilizzate nelle reti neurali artificiali. Una regola empirica fondamentale consiste nel garantire che tutti i nodi sullo stesso livello utilizzino la stessa funzione di attivazione. Nelle reti neurali artificiali multilivello, più livelli possono utilizzare diverse funzioni di attivazione in base al problema da risolvere. Una rete che determina se i prestiti devono essere concessi, per esempio, potrebbe utilizzare la funzione sigmoide nei livelli nascosti per determinare le probabilità e una funzione a gradino nell'output per ottenere una decisione netta: 0 o 1.

Funzione di costo e tasso di apprendimento

Nell'esempio descritto in precedenza abbiamo utilizzato una semplice funzione di costo, in cui l'output previsto viene sottratto dall'output effettivo, ma esistono molte altre funzioni di costo. Le funzioni di costo influenzano notevolmente la rete neurale artificiale, e

l'utilizzo di una funzione adatta al problema e al dataset a portata di mano è importante, perché descrive l'obiettivo della rete neurale artificiale. Una delle funzioni di costo più comuni è l'*errore quadratico medio*, che è simile alla funzione utilizzata nel machine learning (Capitolo 8). Ma le funzioni di costo devono essere selezionate in base alla conoscenza dei dati di addestramento, alla dimensione dei dati di addestramento, alla precisione desiderata e alle misurazioni. A mano a mano che sperimentiamo, dovremo tornare a esaminare le opzioni della funzione di costo.

Infine, il tasso di apprendimento della rete neurale artificiale descrive in che modo i pesi vengono regolati durante la retropropagazione. Una velocità di apprendimento lenta può comportare un lungo processo di addestramento, perché i pesi vengono aggiornati ogni volta di piccole quantità; un tasso di apprendimento elevato potrebbe comportare cambiamenti eccessivi nei pesi, dando origine a un processo di addestramento caotico. Una soluzione consiste nell'iniziare con un tasso di apprendimento fisso e adeguare tale tasso se l'addestramento ristagna e non migliora il costo. Questo processo, che verrebbe ripetuto durante il ciclo di apprendimento, richiede una certa sperimentazione. La discesa del gradiente stocastico è un'utile variante che contrasta questi problemi. Funziona in modo simile alla discesa del gradiente, ma consente ai pesi di uscire dai minimi locali per esplorare soluzioni migliori.

Le reti neurali artificiali standard come quella descritta in questo capitolo sono utili per risolvere problemi di classificazione non lineare. Se stiamo cercando di classificare gli esempi in base a molte caratteristiche, questo stile di reti neurali artificiali è probabilmente una buona opzione.

Detto questo, una rete neurale artificiale non è una soluzione universale e non dovrebbe essere l'algoritmo di riferimento. Gli

algoritmi di machine learning più semplici e tradizionali descritti nel Capitolo 8 spesso offrono prestazioni migliori in molti casi d'uso comuni. Ricordate il ciclo di vita del machine learning. Potreste voler provare diversi modelli di machine learning durante le vostre iterazioni, cercando di ottenere risultati sempre migliori.

Tipi di reti neurali artificiali e casi d'uso

Le reti neurali artificiali sono versatili e possono essere configurate per affrontare vari tipi di problemi. Possono essere adottati specifici stili di reti neurali artificiali per risolvere specifici problemi. Uno stile di architettura di una rete neurale artificiale è un po' come una configurazione di base della rete. Gli esempi in questo paragrafo trattano vari tipi di configurazioni.

Reti neurali convoluzionali

Le *reti neurali convoluzionali* (CNN) sono adatte al riconoscimento delle immagini. Queste reti possono essere utilizzate per trovare relazioni fra oggetti e aree delle immagini. Nel *riconoscimento delle immagini*, la convoluzione opera su un singolo pixel e sui pixel vicini, entro un certo raggio. Questa tecnica è tradizionalmente utilizzata per il rilevamento dei bordi e per agire sulla nitidezza dell'immagine. Le reti neurali convoluzionali utilizzano la convoluzione e il raggruppamento per trovare relazioni fra i pixel di un'immagine. La *convoluzione* trova le caratteristiche presenti nelle immagini e il *pooling* esegue il downsampling dei "pattern", riassumendo le caratteristiche, cosa che consente di codificare in modo conciso le firme univoche nelle immagini attraverso l'apprendimento da più immagini (Figura 9.31).

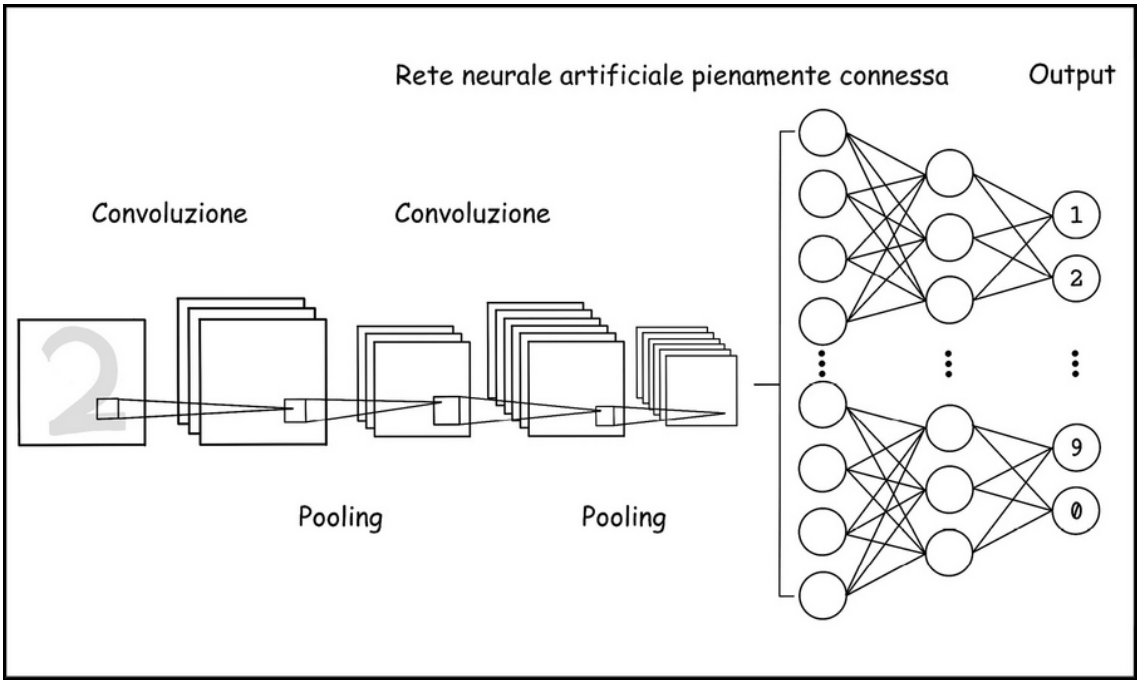


Figura 9.31 Semplice esempio di rete neurale convoluzionale.

Le reti neurali convoluzionali vengono utilizzate per la classificazione delle immagini. Se vi è capitato di cercare un'immagine online, probabilmente avete interagito indirettamente con una rete neurale convoluzionale. Queste reti sono utili anche per il riconoscimento ottico dei caratteri e per l'estrazione dei testi da un'immagine. Le reti neurali convoluzionali sono utilizzate nell'industria medica per applicazioni che rilevano anomalie e patologie tramite i raggi X e altre scansioni del corpo.

Reti neurali ricorrenti

Mentre le reti neurali artificiali standard accettano un numero fisso di input, le *reti neurali ricorrenti* (RNN) accettano una sequenza di input senza una lunghezza predeterminata. Questi input sono come "frasi". Le reti neurali ricorrenti hanno un concetto di memoria costituito dai livelli nascosti, che rappresentano il tempo; questo

concetto consente alla rete di conservare informazioni sulle relazioni fra le sequenze di input. Quando stiamo addestrando una rete neurale ricorrente, anche i pesi nei livelli nascosti sono influenzati nel tempo dalla retropropagazione; vi è il concetto di pesi multipli, che rappresentano lo stesso peso in momenti diversi nel tempo (Figura 9.32).

Le reti neurali ricorrenti sono utili nelle applicazioni di riconoscimento vocale e di riconoscimento/previsione dei testi. I casi d'uso correlati includono il completamento automatico delle frasi nelle applicazioni di messaggistica, la traduzione della lingua parlata in testo e la traduzione.

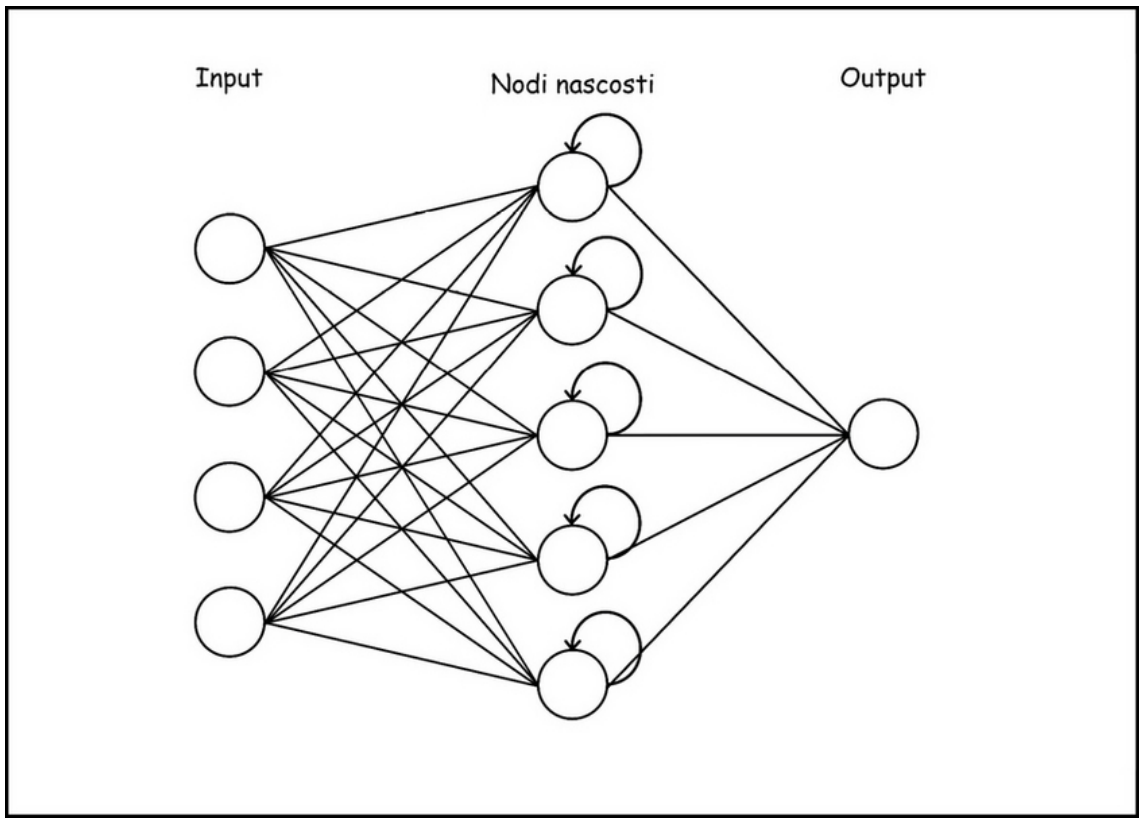


Figura 9.32 Semplice esempio di rete neurale ricorrente.

Reti generative avversarie

Una *rete generativa avversaria* (GAN) è costituita da una rete generatrice e da una rete discriminatrice. Per esempio, quella *generatrice* crea una potenziale soluzione, come un'immagine o un paesaggio; quella *discriminatrice* utilizza immagini reali di paesaggi per determinare il realismo o la correttezza del paesaggio generato. L'errore o il costo viene reimmesso nella rete per migliorare la sua capacità di generare paesaggi convincenti e determinarne la correttezza. Il termine *avversaria* è fondamentale, come abbiamo visto con gli alberi di gioco nel Capitolo 3. Queste due componenti competono per migliorare sempre più e, attraverso quella competizione, generano soluzioni sempre migliori (Figura 9.33).

Le reti generative avversarie vengono utilizzate per generare video falsi ma convincenti (i cosiddetti *deepfake*) di personaggi famosi, il che solleva preoccupazioni sull'autenticità delle informazioni pubblicate nei media. Le reti generative avversarie hanno anche applicazioni utili come la sovrapposizione di acconciature sui volti delle persone. Sono inoltre state utilizzate per generare oggetti 3D da immagini 2D, per esempio di una sedia. Questo caso d'uso può sembrare poco importante, ma il concetto è potente: la rete stima e crea informazioni accurate da una fonte incompleta. È un enorme passo avanti nel progresso dell'intelligenza artificiale e della tecnologia in generale.

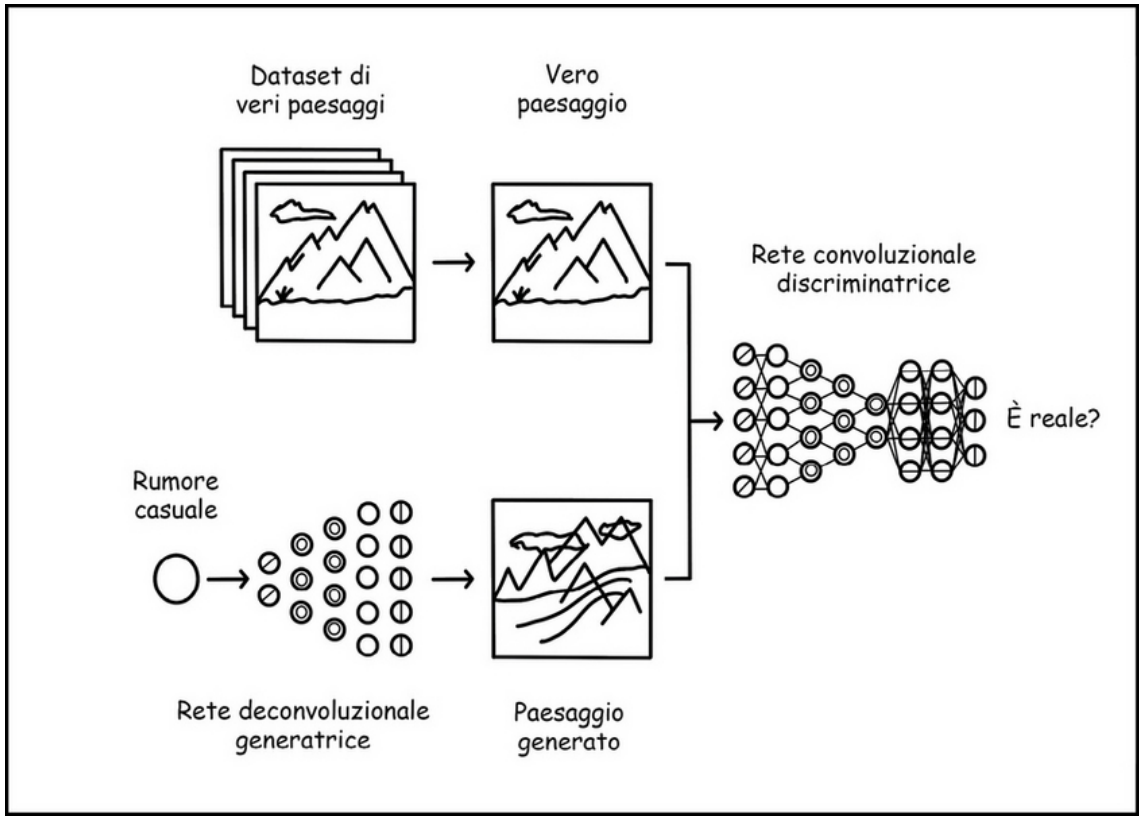


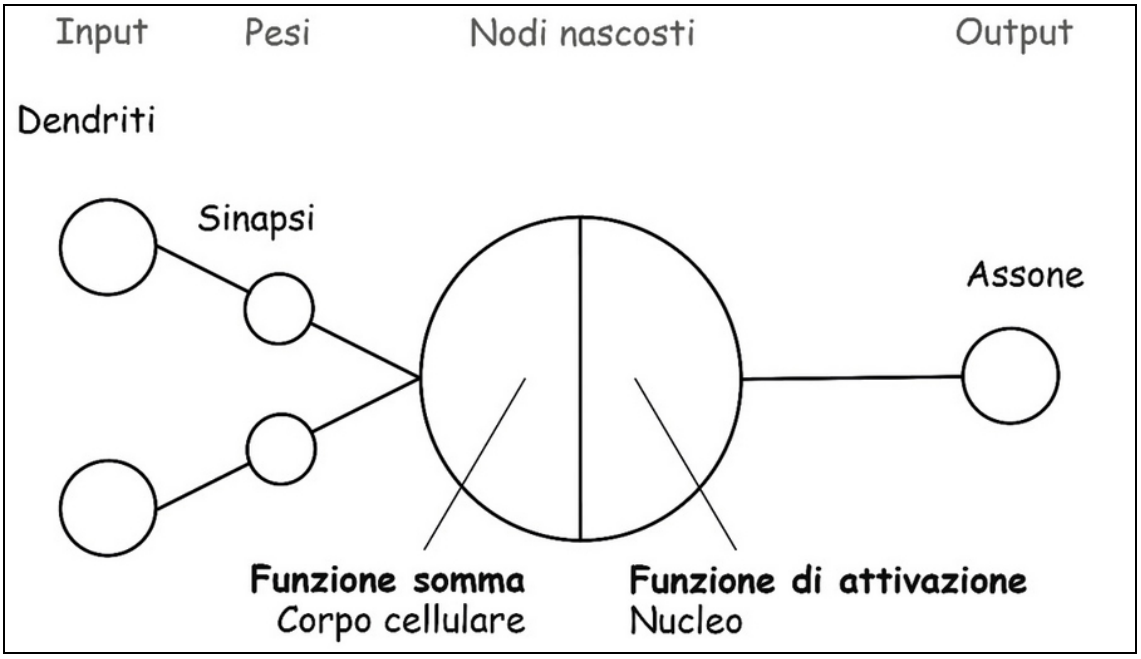
Figura 9.33 Semplice esempio di rete generativa avversaria.

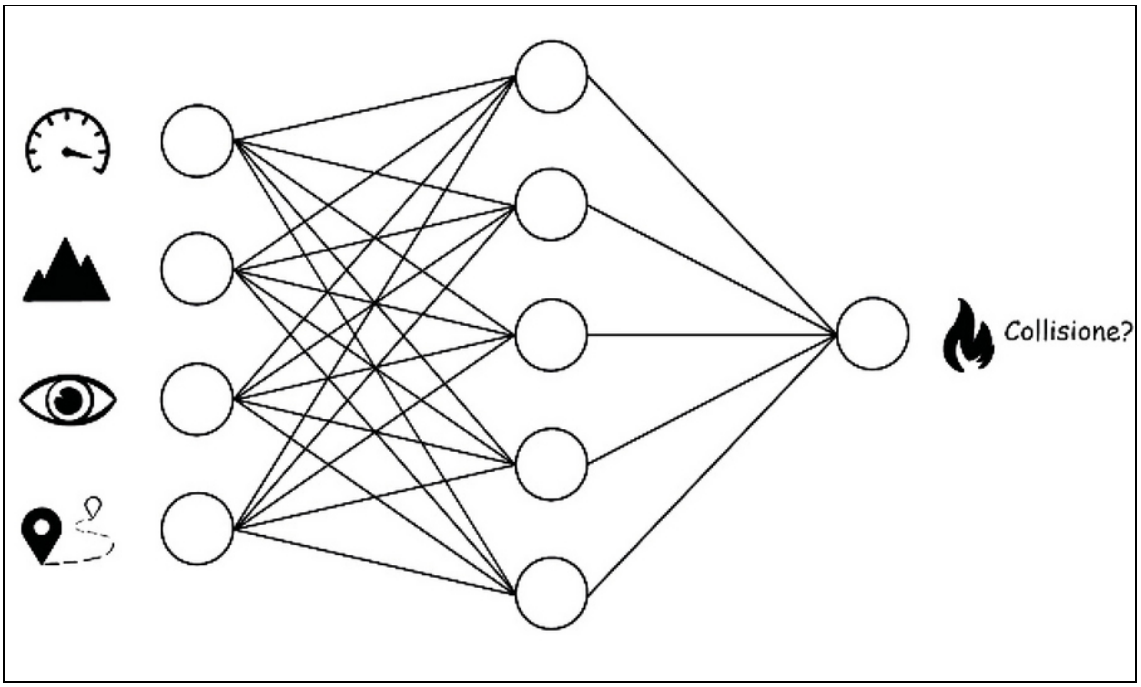
Questo capitolo puntava a unire i concetti del machine learning con il mondo un po' misterioso delle reti neurali artificiali. Per una guida pratica a un framework per la creazione di reti neurali artificiali, consultate *Deep Learning con Python* (Apogeo, 2019).

Riepilogo

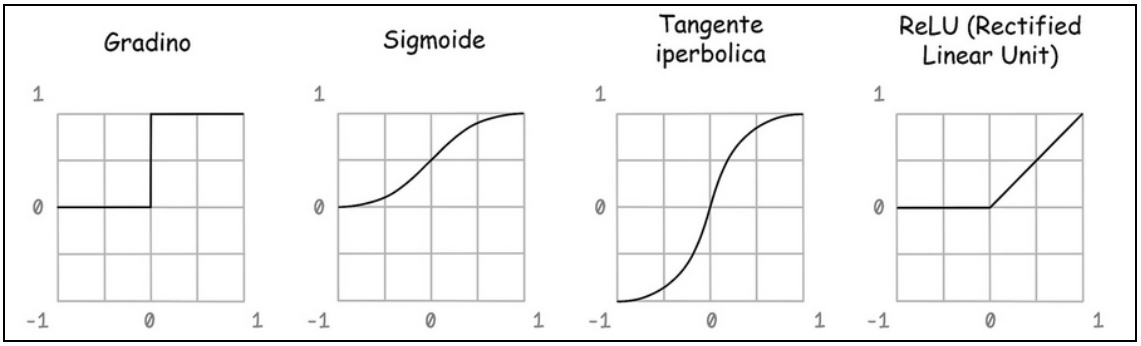
Le reti neurali artificiali sono ispirate al funzionamento del cervello e possono essere considerate come un modello di machine learning.

- Le reti neurali artificiali si basano sull'idea del Perceptron.

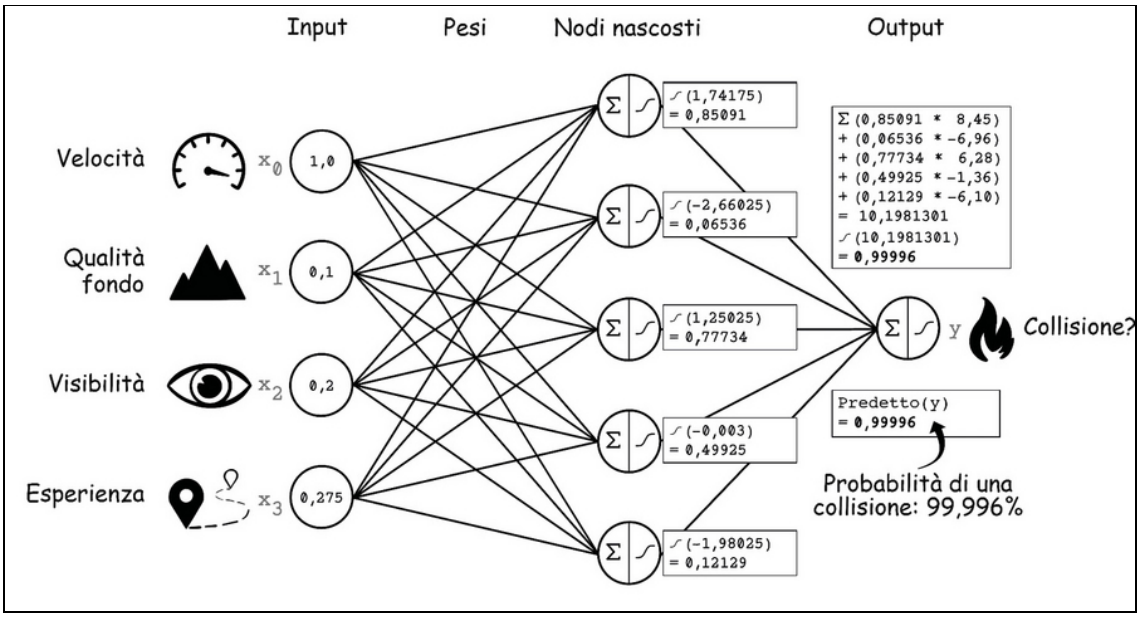




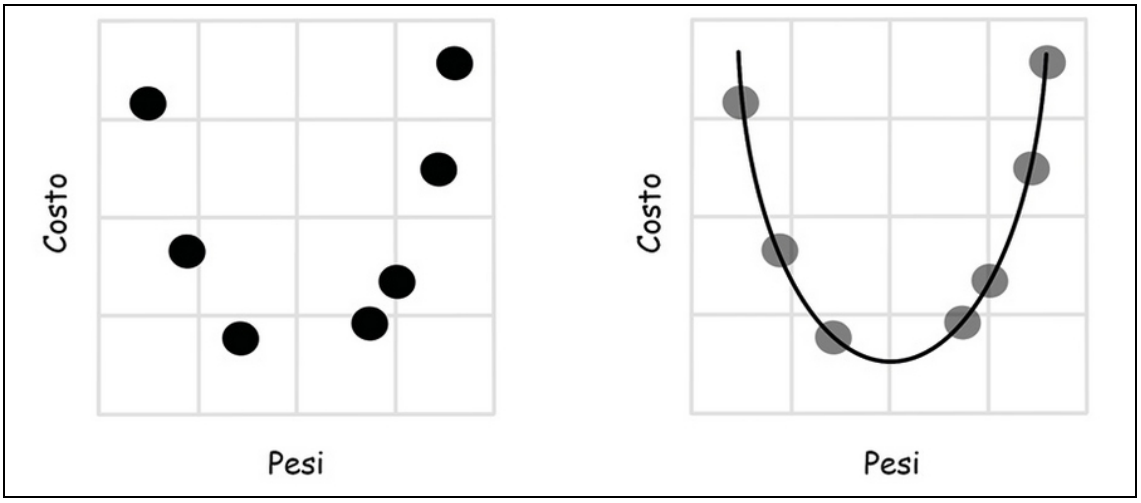
- Le funzioni di attivazione aiutano a risolvere problemi non lineari.



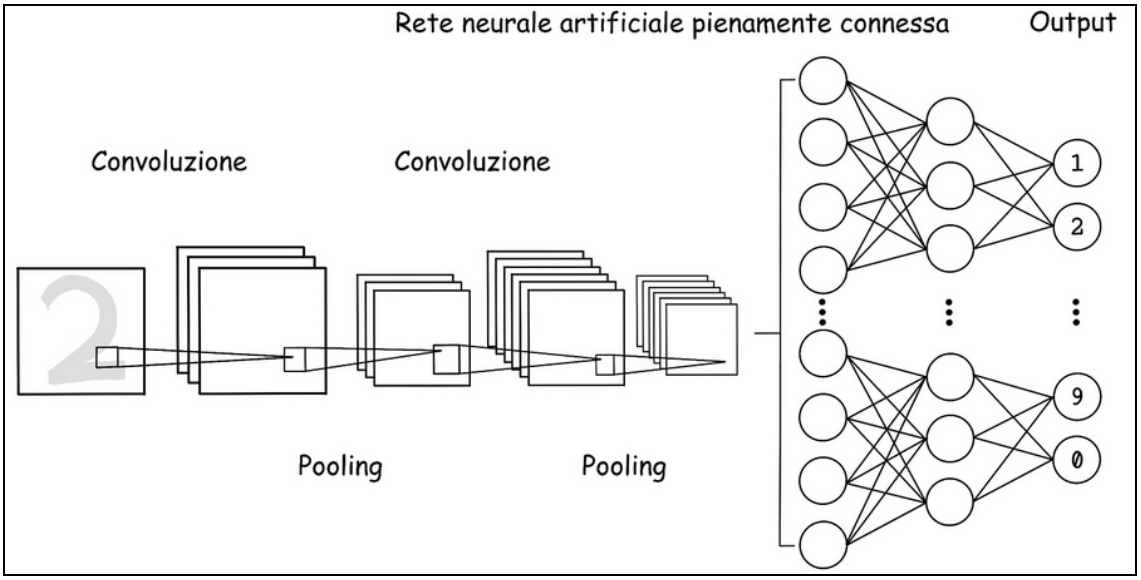
- La propagazione in avanti viene utilizzata per impiegare le reti neurali artificiali in attività di predizione; viene impiegata anche nell'addestramento.



- L'ottimizzazione a discesa del gradiente è una delle molte opzioni di ottimizzazione dei pesi.



- Le reti neurali artificiali sono flessibili e possono essere adattate alla soluzione di vari tipi di problemi.



Apprendimento per rinforzo con Q-learning

Che cos'è l'apprendimento per rinforzo?

L'*apprendimento per rinforzo* è un campo del machine learning che si ispira alla psicologia comportamentale. Il concetto di apprendimento per rinforzo si basa su premi o penalità cumulativi per le azioni intraprese da un agente in un ambiente dinamico. Pensate a un cucciolo di cane. Il cane è l'agente e l'ambiente è la nostra casa. Quando vogliamo che il cane si sieda, gli diciamo "Siediti!". Il cane non capirà, quindi potremmo spingerlo leggermente sui quarti posteriori. Dopo che si è seduto, lo accarezziamo e gli diamo un premio. Se ripetiamo questo processo più volte, dopo qualche tempo avremo rinforzato positivamente l'idea di stare seduti. Il cane assocerà sempre più la parola "Siediti" al doversi sedere.

L'apprendimento per rinforzo è un altro approccio al machine learning insieme all'*apprendimento con supervisione* e *senza supervisione*. Mentre l'apprendimento con supervisione utilizza dei dati etichettati per fare previsioni e classificazioni e l'apprendimento senza supervisione utilizza dati non etichettati per trovare raggruppamenti e tendenze, l'apprendimento per rinforzo utilizza il feedback delle azioni scelte per apprendere quali azioni (o quale

sequenza di azioni) sono più vantaggiose nei diversi scenari che conducono a un obiettivo finale. L'apprendimento per rinforzo è utile quando conosciamo l'obiettivo ma non quali azioni sono più utili per raggiungerlo. La Figura 10.1 mostra i concetti di machine learning e la collocazione in questo ambito dell'apprendimento per rinforzo.

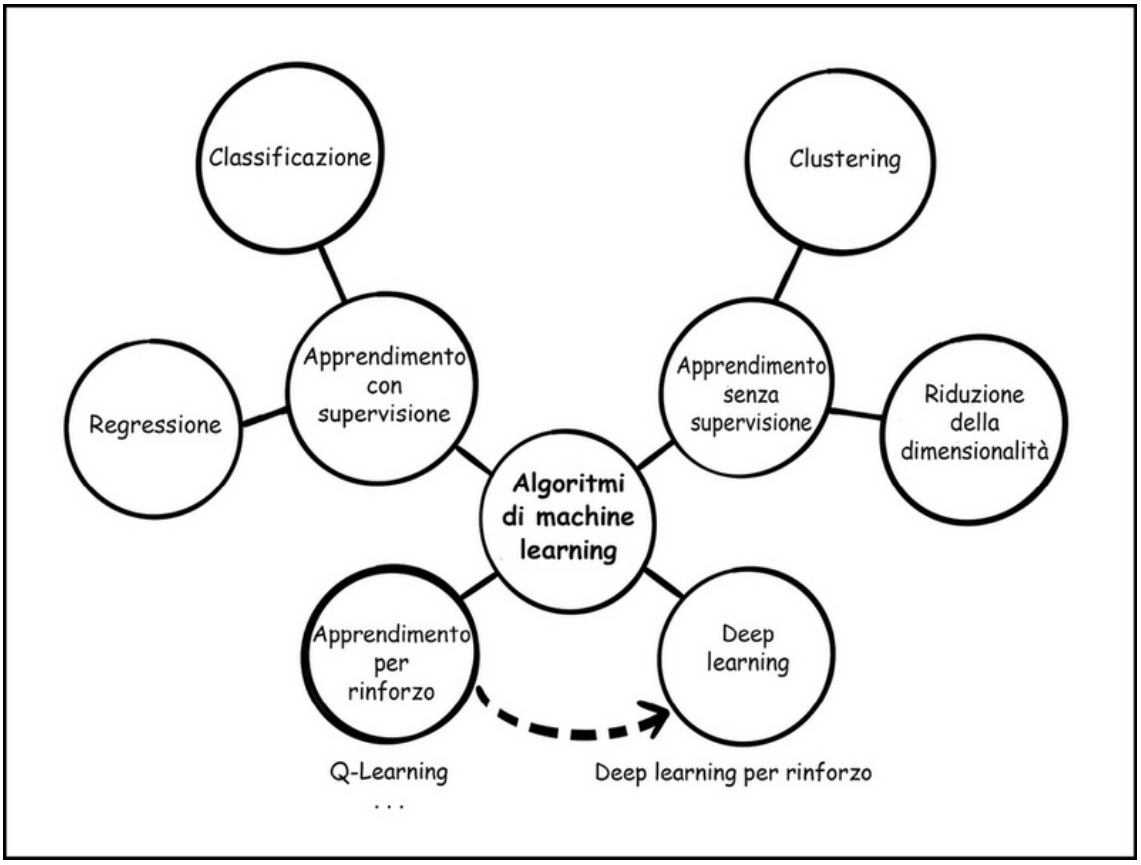


Figura 10.1 Collocazione dell'apprendimento per rinforzo nel machine learning.

L'apprendimento per rinforzo può essere ottenuto impiegando tecniche classiche o il deep learning, coinvolgendo le reti neurali artificiali. A seconda del problema da risolvere, entrambi gli approcci potrebbero essere efficaci.

La Figura 10.2 mostra quando è possibile utilizzare i diversi approcci del machine learning. In questo capitolo esploreremo l'apprendimento per rinforzo attraverso i metodi classici.

L’ispirazione dell’apprendimento per rinforzo

L’apprendimento per rinforzo deriva dalla psicologia comportamentale, un campo che si interessa al comportamento degli esseri umani e degli animali. La psicologia comportamentale di solito spiega il comportamento con un’azione riflessa o qualcosa appreso nella storia dell’individuo. Si esplora il rinforzo attraverso ricompense o punizioni, motivazioni per adottare determinati comportamenti e aspetti dell’ambiente che favoriscono certi comportamenti.

Il meccanismo di prova ed errore è uno dei modi più comuni in cui gli animali più evoluti imparano che cosa è vantaggioso (o svantaggioso). Un meccanismo che significa provare qualcosa, correre il rischio di sbagliare e provare qualcosa di diverso fino a trovare qualcosa di positivo. Questo processo può ripetersi molte volte prima di ottenere il risultato desiderato ed è in gran parte guidato da una ricompensa.

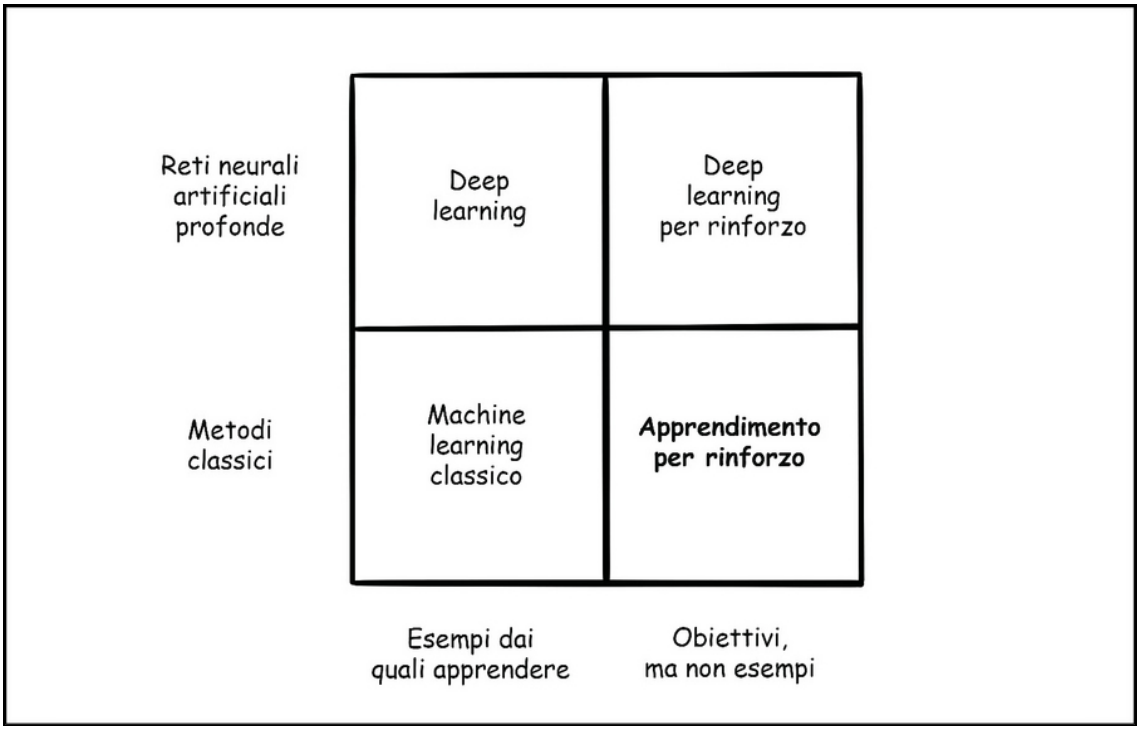


Figura 10.2 Categorizzazione del machine learning, del deep learning e dell'apprendimento per rinforzo.

Questo comportamento può essere osservato ovunque in natura. I pulcini appena nati, per esempio, cercano di beccare qualsiasi cosa vedono per terra. Attraverso prove ed errori, i pulcini imparano a beccare solo il cibo.

Un altro esempio sono gli scimpanzé, che imparano attraverso prove ed errori che usare un bastone per scavare è più vantaggioso che usare le mani. Obiettivi, ricompense e penalità sono importanti nell'apprendimento per rinforzo. Lo scimpanzé ha l'obiettivo di trovare del cibo; una ricompensa o una punizione può essere il numero di volte in cui ha scavato una buca o il tempo impiegato per scavarla. Più velocemente riesce a scavare una buca, più velocemente troverà il cibo.

La Figura 10.3 esamina la terminologia utilizzata nell'apprendimento per rinforzo, con riferimento al semplice esempio di addestramento del cane.

L'apprendimento per rinforzo prevede un rinforzo negativo e uno positivo. Il *rinforzo positivo* consiste nel ricevere una ricompensa dopo aver eseguito un'azione: un cane riceve un premio dopo essersi seduto. Il *rinforzo negativo* consiste nel ricevere una punizione dopo aver eseguito un'azione: un cane viene rimproverato dopo aver rovinato un tappeto. Il rinforzo positivo ha lo scopo di motivare il comportamento desiderato; il rinforzo negativo ha lo scopo di scoraggiare il comportamento indesiderato.

Un altro concetto nell'apprendimento per rinforzo consiste nel bilanciare la gratificazione immediata con le conseguenze a lungo termine. Mangiare una barretta di cioccolato è ottimo per ottenere una sferzata di zucchero ed energia; questa è una *gratificazione immediata*. Ma mangiare una barretta di cioccolato ogni 30 minuti probabilmente causerà problemi di salute più avanti nella vita; questa è una *conseguenza a lungo termine*.

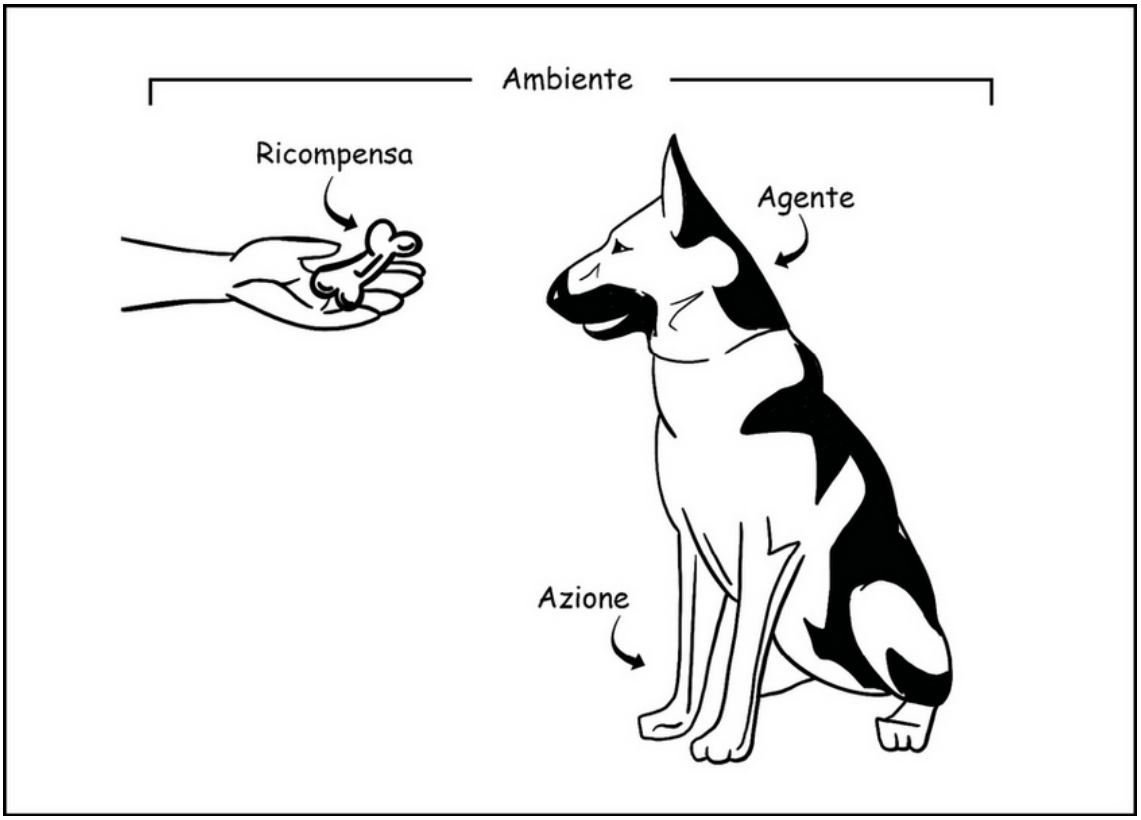


Figura 10.3 Esempio di apprendimento per rinforzo: insegnare a un cane a sedersi usando il cibo come ricompensa.

L'apprendimento per rinforzo ha lo scopo di massimizzare i benefici a lungo termine rispetto a quelli a breve termine, sebbene i benefici a breve termine possano contribuire a quelli a lungo termine.

L'apprendimento per rinforzo riguarda le conseguenze a lungo termine delle azioni in un ambiente, quindi il tempo e la sequenza delle azioni sono importanti. Supponiamo di essere bloccati nel deserto e che il nostro obiettivo sia quello di sopravvivere il più a lungo possibile viaggiando il più lontano possibile nella speranza di trovare rifugio. Siamo posizionati vicino a un fiume e abbiamo due opzioni: saltare nel fiume per viaggiare più velocemente a valle o camminare lungo il lato del fiume. Notate la barca sulla sponda del fiume nella Figura 10.4. Nuotando, viaggeremo più velocemente, ma potremmo perdere la

barca, venendo trascinati lungo la biforcazione sbagliata del fiume. Camminando, avremo la certezza di trovare la barca, che renderà il resto del viaggio molto più semplice, ma all'inizio non lo sappiamo ancora. Questo esempio mostra quanto sia importante la sequenza di azioni nell'apprendimento per rinforzo. Mostra anche come la gratificazione istantanea possa portare a un danno a lungo termine. Inoltre, in uno scenario che non preveda la barca, la conseguenza del nuoto è che viaggeremo più velocemente, ma avremo i vestiti fradici, il che può essere problematico se fa freddo. La conseguenza del camminare è che viaggeremo più lentamente, ma non ci bagneremo, il che evidenzia il fatto che una determinata azione può funzionare in uno scenario ma non in altri. Imparare da molti tentativi simulati è importante per trovare approcci più generali.

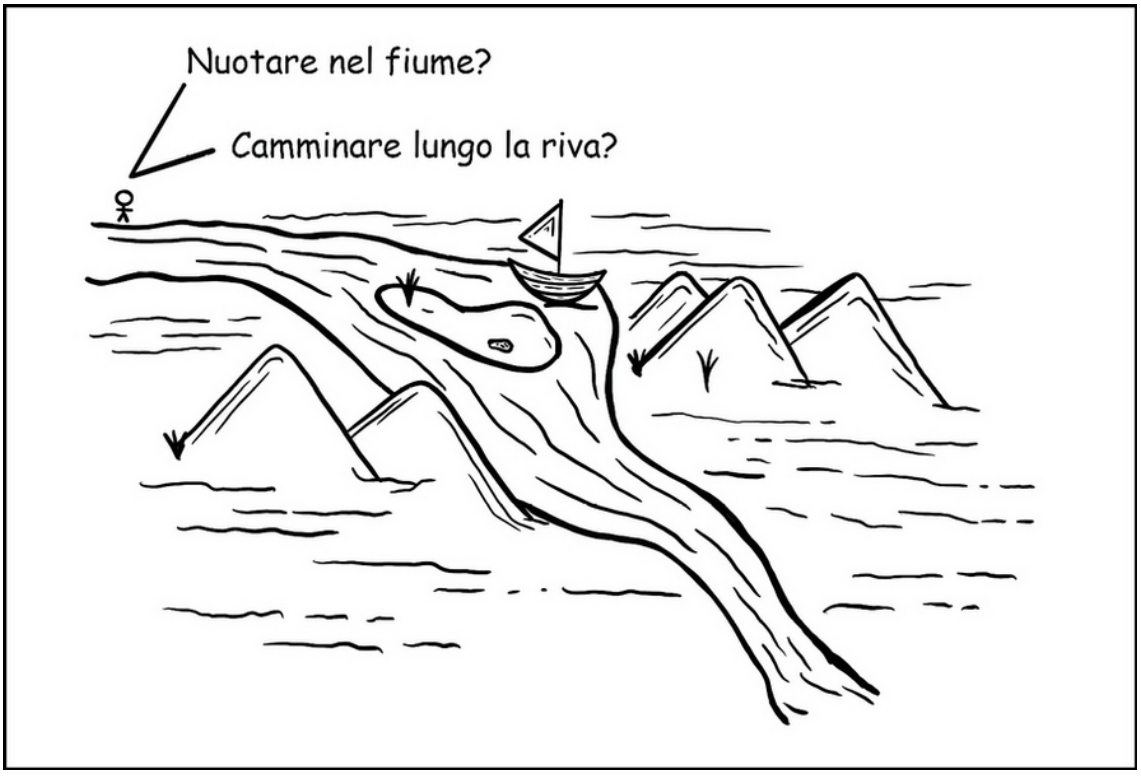


Figura 10.4 Un esempio delle azioni possibili che hanno conseguenze a lungo termine.

Problemi risolvibili dall'apprendimento per rinforzo

Per riassumere, l'apprendimento per rinforzo ha lo scopo di risolvere problemi nei quali è noto l'obiettivo, ma non le azioni necessarie per raggiungerlo. Questi problemi implicano il controllo delle azioni di un agente in un ambiente. Le singole azioni possono essere premiate, alcune più, altre meno, ma la preoccupazione principale è la ricompensa cumulativa di tutte le azioni.

L'apprendimento per rinforzo è particolarmente utile nei problemi nei quali le singole azioni si accumulano per conseguire un obiettivo più grande. Campi come la pianificazione strategica, l'automazione dei processi industriali e la robotica sono tutti esempi d'uso dell'apprendimento per rinforzo. In questi campi, le singole azioni possono non essere ottimali per ottenere un risultato favorevole. Immaginate un gioco di strategia come gli scacchi. Alcune mosse possono essere "sbagliate" in base allo stato attuale della scacchiera, ma aiutano a preparare la situazione strategica che porterà in seguito a una vittoria. L'apprendimento per rinforzo si comporta molto bene in domini nei quali le catene di eventi sono importanti per trovare una buona soluzione.

Per esaminare i passaggi di un algoritmo di apprendimento per rinforzo, useremo l'esempio del problema degli incidenti automobilistici del Capitolo 9. Questa volta, tuttavia, impiegheremo i dati visivi di un'auto a guida autonoma in un parcheggio, che cerca di raggiungere il suo proprietario. Supponiamo di avere la mappa di un parcheggio, dove si trova un'auto a guida autonoma, altre auto e pedoni. La nostra auto a guida autonoma può spostarsi a nord, sud, est e ovest. Le altre auto e i pedoni rimangono fermi in questo esempio.

L’obiettivo è che la nostra auto raggiunga il suo proprietario scontrandosi con il minor numero possibile di volte con auto e pedoni, idealmente senza scontrarsi affatto. La collisione con un’auto è punita, perché danneggia i veicoli; la collisione con un pedone è ancora più grave. In questo problema, vogliamo minimizzare le collisioni, ma se possiamo scegliere fra la collisione con un’auto e un pedone, dovremmo scegliere l’auto. La Figura 10.5 illustra questo scenario.

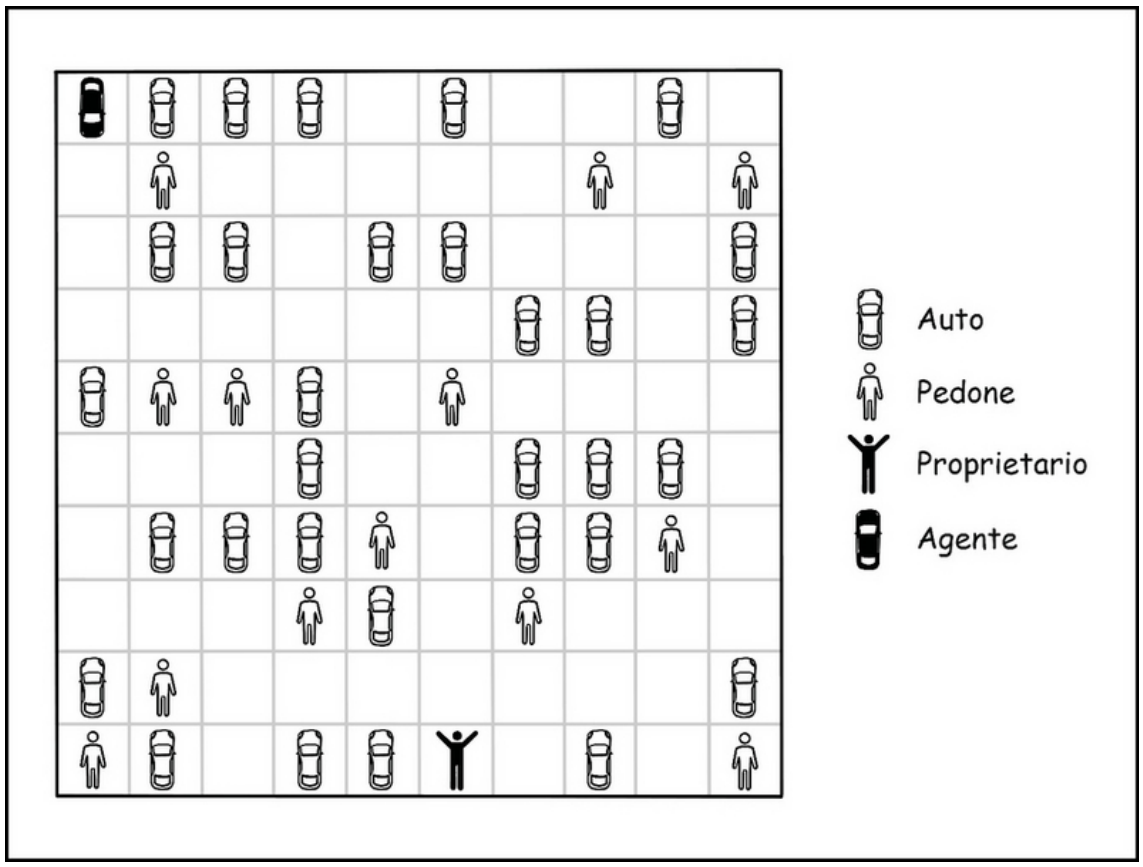


Figura 10.5 Il problema dell’auto a guida autonoma in un parcheggio.

Useremo questo problema per esplorare l’uso dell’apprendimento per rinforzo per imparare buone azioni da intraprendere in ambienti dinamici.

Il ciclo di vita dell'apprendimento per rinforzo

Come altri algoritmi di machine learning, un modello di apprendimento per rinforzo deve essere addestrato prima di poter essere utilizzato. La fase di addestramento è incentrata sull'esplorazione dell'ambiente e sulla ricezione di feedback, in base a determinate azioni eseguite in determinate circostanze o stati. Il ciclo di vita dell'addestramento di un modello di apprendimento per rinforzo si basa sul *processo decisionale di Markov*, che fornisce un quadro matematico per modellare le decisioni (Figura 10.6). Quantificando le decisioni prese e i loro risultati, possiamo addestrare un modello perché apprenda quali azioni verso un obiettivo sono le più favorevoli.

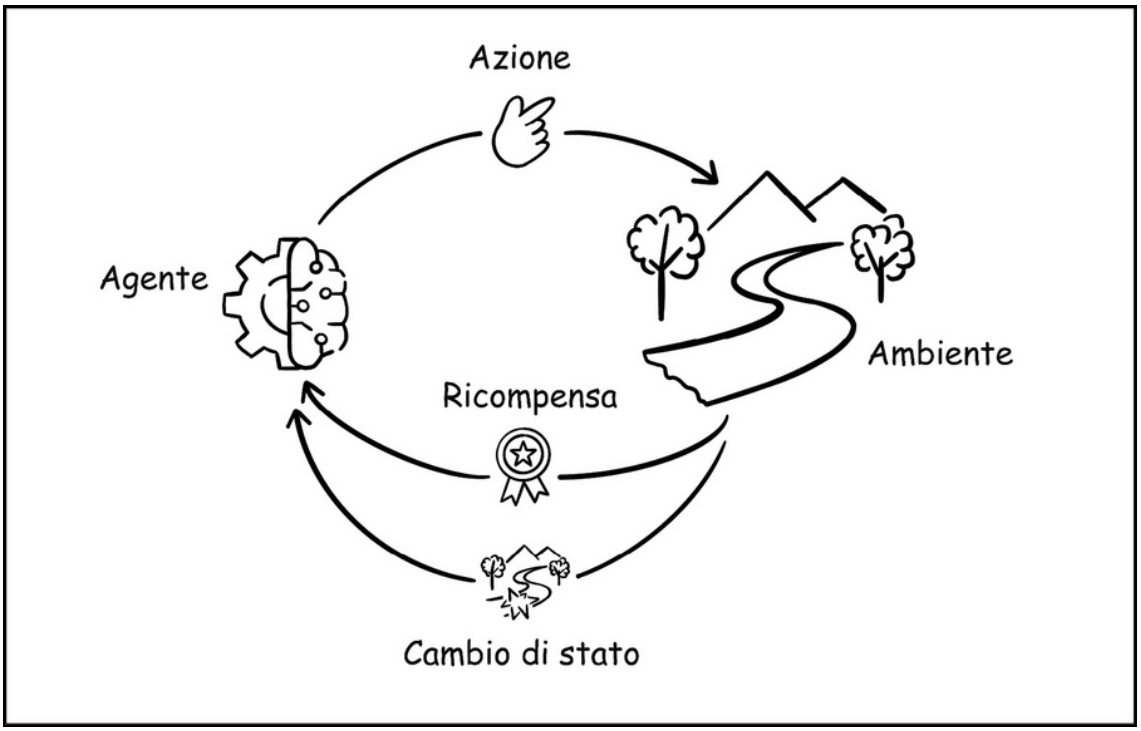


Figura 10.6 Il processo decisionale di Markov per l'apprendimento per rinforzo.

Prima di poter iniziare ad affrontare l'addestramento di un modello utilizzando l'apprendimento per rinforzo, abbiamo bisogno di un ambiente che simuli lo spazio del problema in cui stiamo lavorando. Il nostro problema di esempio comporta un'auto a guida autonoma che cerca di spostarsi in un parcheggio pieno di ostacoli, per trovare il suo proprietario evitando le collisioni. Questo problema deve essere modellato come una simulazione, in modo da poter valutare le azioni svolte nell'ambiente con l'idea di raggiungere l'obiettivo. Questo ambiente simulato è diverso dal modello che imparerà quali azioni intraprendere.

Simulazione e dati: ricreare l'ambiente

La Figura 10.7 rappresenta lo scenario di un parcheggio contenente molte altre auto e pedoni. La posizione di partenza dell'auto a guida autonoma e la posizione del suo proprietario sono rappresentate come figure nere. In questo esempio, l'auto a guida autonoma che applica azioni all'ambiente è l'*agente*.

L'auto a guida autonoma, o agente, può intraprendere diverse azioni nell'ambiente. In questo semplice esempio, le azioni la spostano verso nord, sud, est o ovest. La scelta di un'azione fa sì che l'agente si sposti di un blocco in quella direzione. L'agente non può spostarsi in diagonale.

Quando vengono intraprese determinate azioni nell'ambiente, si ottengono premi o penalità. La Figura 10.8 mostra i punti assegnati all'agente in base al risultato nell'ambiente. Una collisione con un'altra auto è "male"; una collisione con un pedone è "molto male". Spostarsi in uno spazio vuoto è "bene"; trovare il proprietario è "molto bene".

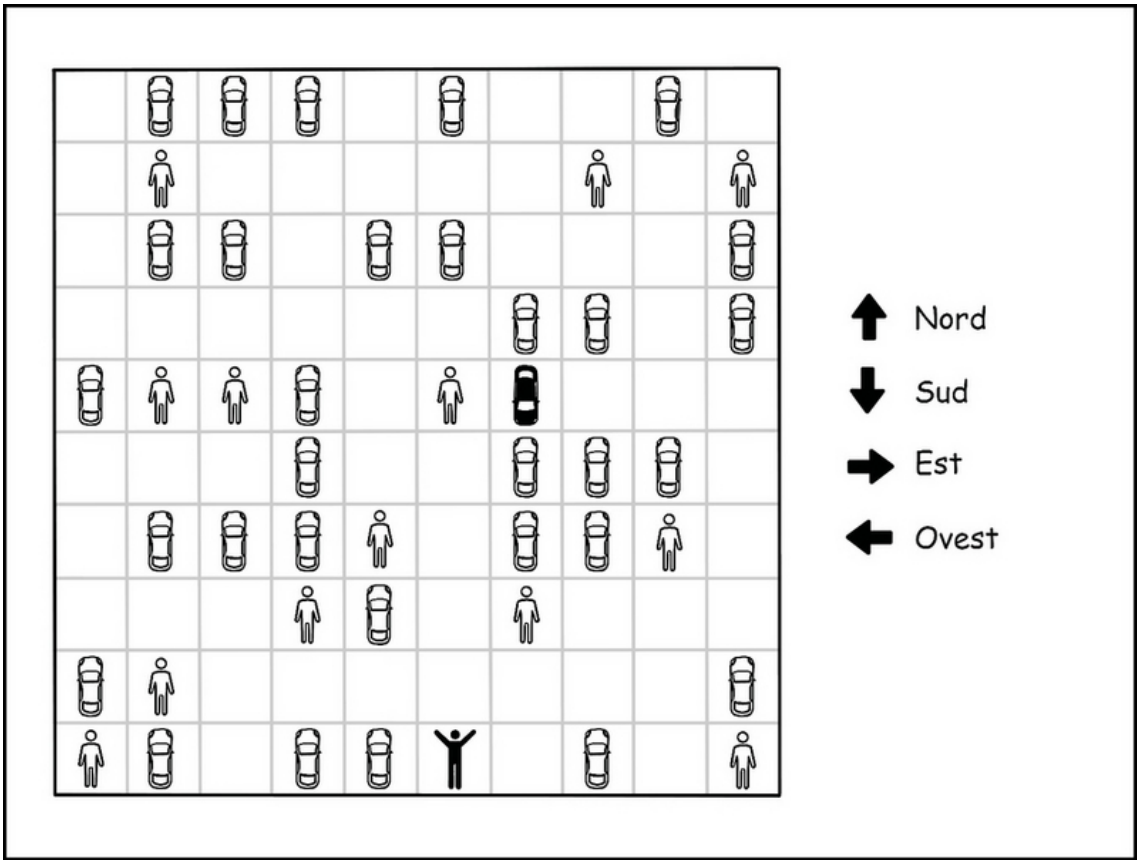


Figura 10.7 Azioni dell'agente nell'ambiente del parcheggio.

Queste ricompense puntano a scoraggiare le collisioni con altre auto e soprattutto con i pedoni e a incoraggiare gli spostamenti verso spazi vuoti e in direzione del proprietario. Notate che potrebbe anche esserci una punizione per i movimenti fuori dal parcheggio, ma per semplicità non consentiremo questa possibilità.

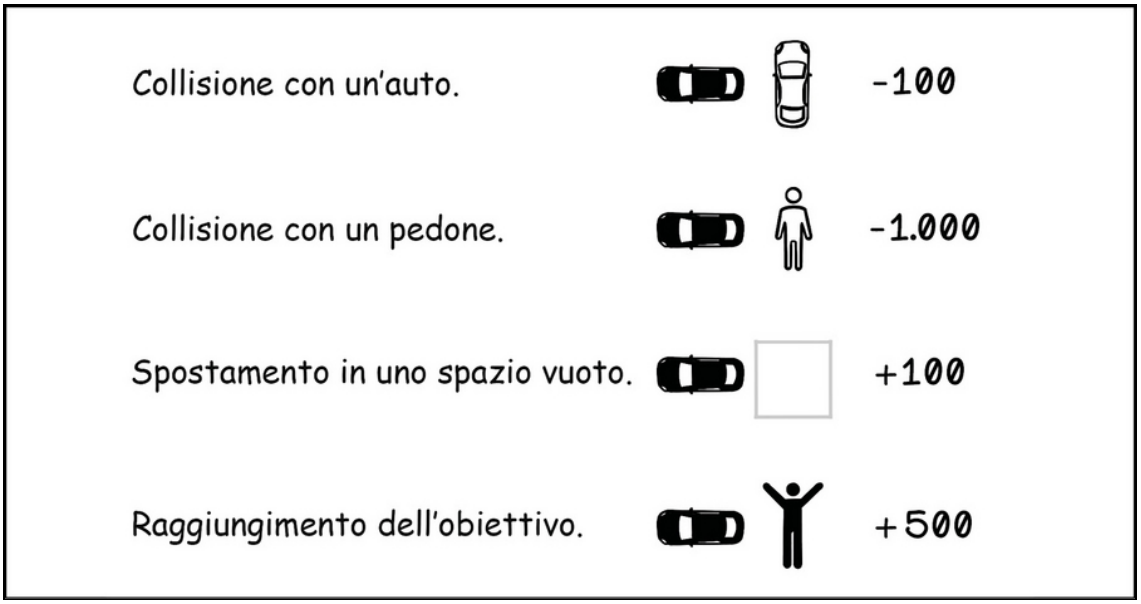


Figura 10.8 Ricompense dovute a determinate azioni nell'ambiente.

> **NOTA**
> Un risultato interessante del meccanismo a ricompense e penalità descritto è che l'auto può finire per avanzare e retrocedere indefinitamente su spazi vuoti per accumulare ricompense. La scarteremo come possibilità per questo esempio, ma sottolinea l'importanza di fare molta attenzione alle ricompense.

Il simulatore deve modellare l'ambiente, le azioni dell'agente e le ricompense ricevute dopo ogni azione. Un algoritmo di apprendimento per rinforzo utilizzerà il simulatore per apprendere attraverso la pratica intraprendendo azioni nell'ambiente simulato e misurando il risultato. Il simulatore deve fornire almeno le seguenti funzionalità e informazioni.

- *Inizializzare l'ambiente*. Questa funzione comporta il reset dell'ambiente, incluso l'agente, allo stato iniziale.
- *Ottenere lo stato attuale dell'ambiente*. Questa funzione fornisce lo stato corrente dell'ambiente, che cambierà dopo l'esecuzione di ogni azione.

- *Applicare un'azione all'ambiente*. Questa funzione prevede che l'agente applichi un'azione all'ambiente. L'ambiente è influenzato dall'azione, la quale può comportare una ricompensa.
- *Calcolare la ricompensa dell'azione*. Questa funzione è correlata all'applicazione dell'azione all'ambiente. È necessario calcolare la ricompensa per l'azione e l'effetto sull'ambiente.
- *Determinare se l'obiettivo è stato raggiunto*. Questa funzione determina se l'agente ha raggiunto l'obiettivo. L'obiettivo a volte viene normalmente rappresentato come `is complete`. In un ambiente in cui l'obiettivo non può essere raggiunto, il simulatore deve segnalare il completamento quando lo ritiene necessario.

Le Figure 10.9 e 10.10 rappresentano i possibili percorsi dell'auto a guida autonoma. Nella Figura 10.9, l'agente viaggia verso Sud fino a raggiungere il limite; poi viaggia verso Est fino a raggiungere il proprietario. Sebbene l'obiettivo sia stato raggiunto, lo scenario ha provocato cinque collisioni con altre auto e una collisione con un pedone, non un risultato ideale. La Figura 10.10 mostra l'agente che viaggia lungo un percorso più specifico verso il proprietario, senza collisioni, il che è fantastico. È importante notare che date le ricompense che abbiamo specificato, non è garantito che l'agente individui il percorso più breve; poiché lo incoraggiamo a evitare gli ostacoli, l'agente può trovare un percorso qualsiasi che sia privo di ostacoli.

In questo momento, non c'è automazione nell'invio di azioni al simulatore. È come un gioco in cui forniamo un input a un giocatore e non a un'intelligenza artificiale. Il prossimo paragrafo spiega come addestrare un agente autonomo.

Pseudocodice

Lo pseudocodice per il simulatore comprende le funzioni trattate in questo paragrafo. La classe del simulatore viene inizializzata con le informazioni

relative allo stato iniziale dell'ambiente.

La funzione `move_agent` è responsabile dello spostamento dell'agente a Nord, Sud, Est o Ovest, in base all'azione. Determina se il movimento rientra nei limiti, aggiorna le coordinate dell'agente, determina se si è verificata una collisione e restituisce un punteggio di ricompensa/punizione in base al risultato:

```
Simulator(road, road_size_x,road_size_y,
        agent_start_x, agent_start_y,
        goal_x, goal_y):
move_agent(action):
    if action equals COMMAND_NORTH:
        let next_x equal agent_x - 1
       let next_y equal agent_y
    else if action equals COMMAND_ SOUTH:
        let next_x equal agent_x + 1
        let next_y equal agent_y
    else if action equals COMMAND EAST:
        let next_x equal agent_x
        let next_y equal agent_y + 1
    else if action equals COMMAND WEST:
        let next_x equal agent_x
        let next_y equal agent_y - 1
    if is_within_bounds(next_x, next_y) equals True:
        let reward_update equal cost_movement(next_x, next_y)
        let agent_x equal next_x
        let agent_y equal next_y
    else:
        let reward_update equal ROAD_OUT_OF_BOUNDS_REWARD
    return reward_update
```

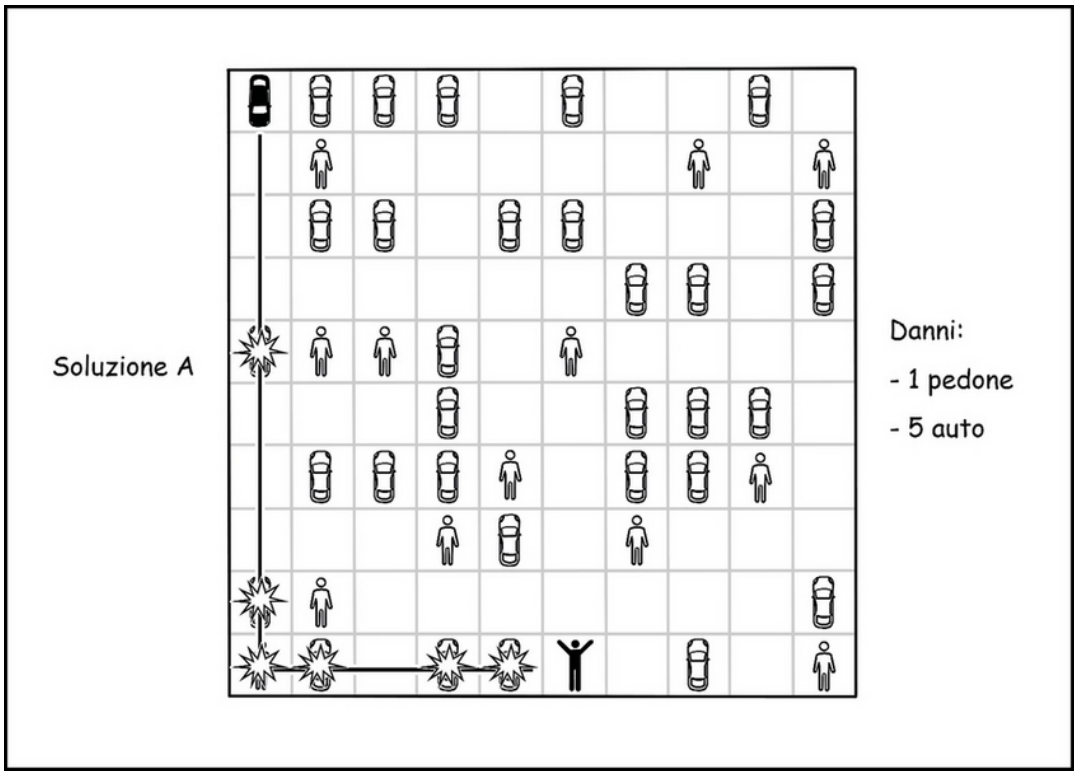


Figura 10.9 Una cattiva soluzione al problema del parcheggio.

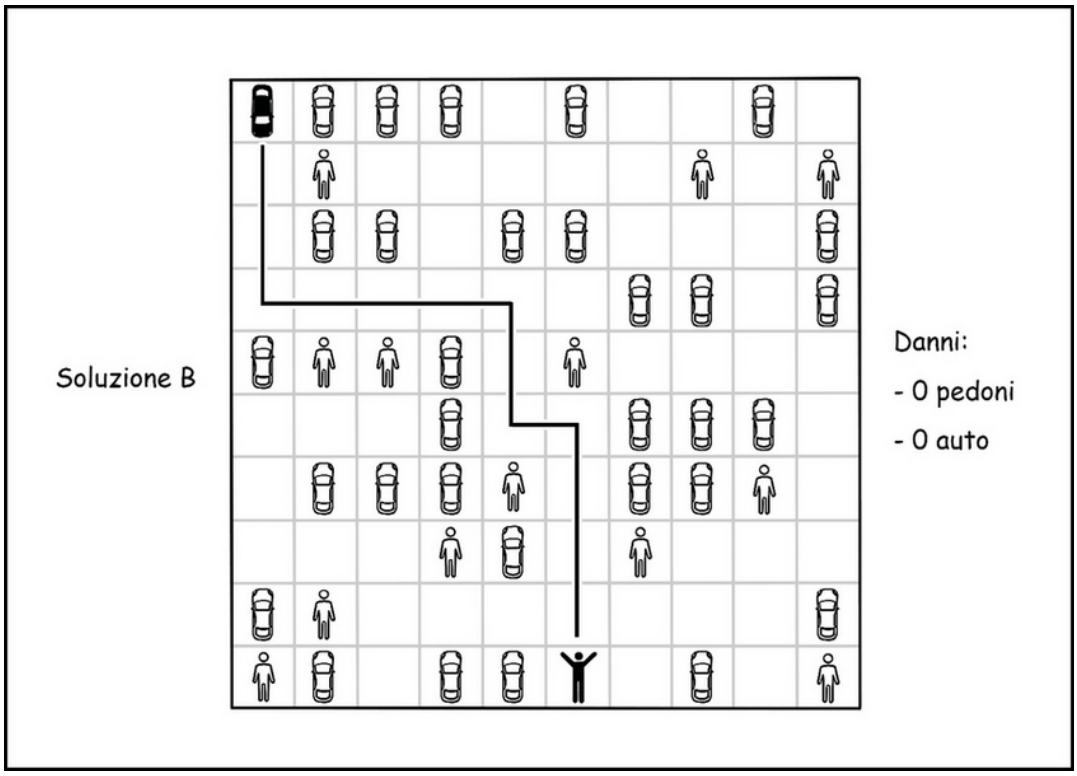


Figura 10.10 Una buona soluzione al problema del parcheggio.

Ecco le descrizioni delle funzioni presenti in questo pseudocodice.

- La funzione `cost_movement` determina l'oggetto presente nella coordinata di destinazione verso la quale si sposterà l'agente, e restituisce il relativo punteggio di ricompensa/punizione.
- La funzione `is_within_bounds` è una funzione di servizio che assicura che le coordinate di destinazione siano all'interno del confine del parcheggio.
- La funzione `is_goal_achieved` determina se l'obiettivo è stato trovato, nel qual caso la simulazione può terminare.
- La funzione `get_state` utilizza la posizione dell'agente per determinare un numero per lo stato corrente. Ogni stato deve

essere unico. In altri spazi dei problemi, lo stato può essere rappresentato dallo stato effettivo.

```
cost_movement(next_x, next_y):
    if road[next_x][next_y] equals ROAD_OBSTACLE_PERSON:
        return ROAD_OBSTACLE_PERSON_REWARD
    else if road[next_x][next_y] equals ROAD_OBSTACLE_CAR:
        return ROAD_OBSTACLE_CAR_REWARD
    else if road[next_x][next_y] equals ROAD_GOAL:
        return ROAD_GOAL_REWARD
    else:
        return ROAD_EMPTY_REWARD
is_within_bounds(next_x, next_y):
    if road_size_x > next_x >= 0 and road_size_y > next_y >= 0:
        return True
    return False
is_goal_achieved():
    if agent_x equals goal_x and agent_y equals goal_y:
        return True
    return False
get_state( ):
    return(road_size_x * agent_x) + agent_y
```

Addestramento con la simulazione usando Q-learning

Q-learning è un approccio all'apprendimento per rinforzo che utilizza gli stati e le azioni in un ambiente per modellare una tabella, la quale contiene informazioni che descrivono le azioni favorevoli sulla base di determinati stati. Il Q-learning è come un dizionario in cui la chiave è lo stato dell'ambiente e il valore è l'azione migliore da intraprendere per quello stato.

L'apprendimento per rinforzo con Q-learning utilizza una tabella di ricompensa chiamata *Q-table*. Una Q-table è composta da colonne, che rappresentano le possibili azioni, e da righe, che rappresentano i possibili stati nell'ambiente. Lo scopo di una Q-table è quello di descrivere quali azioni sono più favorevoli per l'agente mentre cerca un obiettivo. I valori che rappresentano le azioni favorevoli vengono appresi simulando le possibili azioni nell'ambiente e imparando dal

risultato e dal cambiamento di stato. Vale la pena notare che l'agente ha la possibilità di scegliere un'azione a caso o un'azione tratta dalla Q-table, come mostrato più avanti nella Figura 10.13. *Q* è la funzione che fornisce la ricompensa, o la qualità, di un'azione in un ambiente.

La Figura 10.11 mostra una Q-table addestrata e due possibili stati che possono essere rappresentati dai rispettivi valori. Questi stati sono rilevanti per il problema che stiamo risolvendo; un altro problema potrebbe consentire all'agente di muoversi anche in diagonale. Notate che il numero di stati varia in base all'ambiente e che è possibile aggiungere nuovi stati a mano a mano che vengono rilevati. Nello Stato 1, l'agente si trova nell'angolo in alto a sinistra e nello Stato 2, l'agente si trova nella posizione adiacente a "Sud". La Q-table codifica le migliori azioni da intraprendere, dato ogni rispettivo stato. L'azione con il numero più elevato è l'azione più vantaggiosa. In questa figura, i valori nella Q-table sono già stati trovati attraverso un addestramento. Presto vedremo come sono stati calcolati.

Il grosso problema con la rappresentazione dello stato utilizzando l'intera mappa è che la configurazione di altre auto e persone è specifica di questo problema. La Q-table apprende le scelte migliori solo per questa specifica mappa.

Un modo migliore per rappresentare lo stato in questo problema di esempio consiste nell'esaminare gli oggetti adiacenti all'agente. Questo approccio consente alla Q-table di adattarsi ad altre configurazioni di parcheggi, poiché lo stato non è specifico del parcheggio di esempio, che sta usando per imparare.

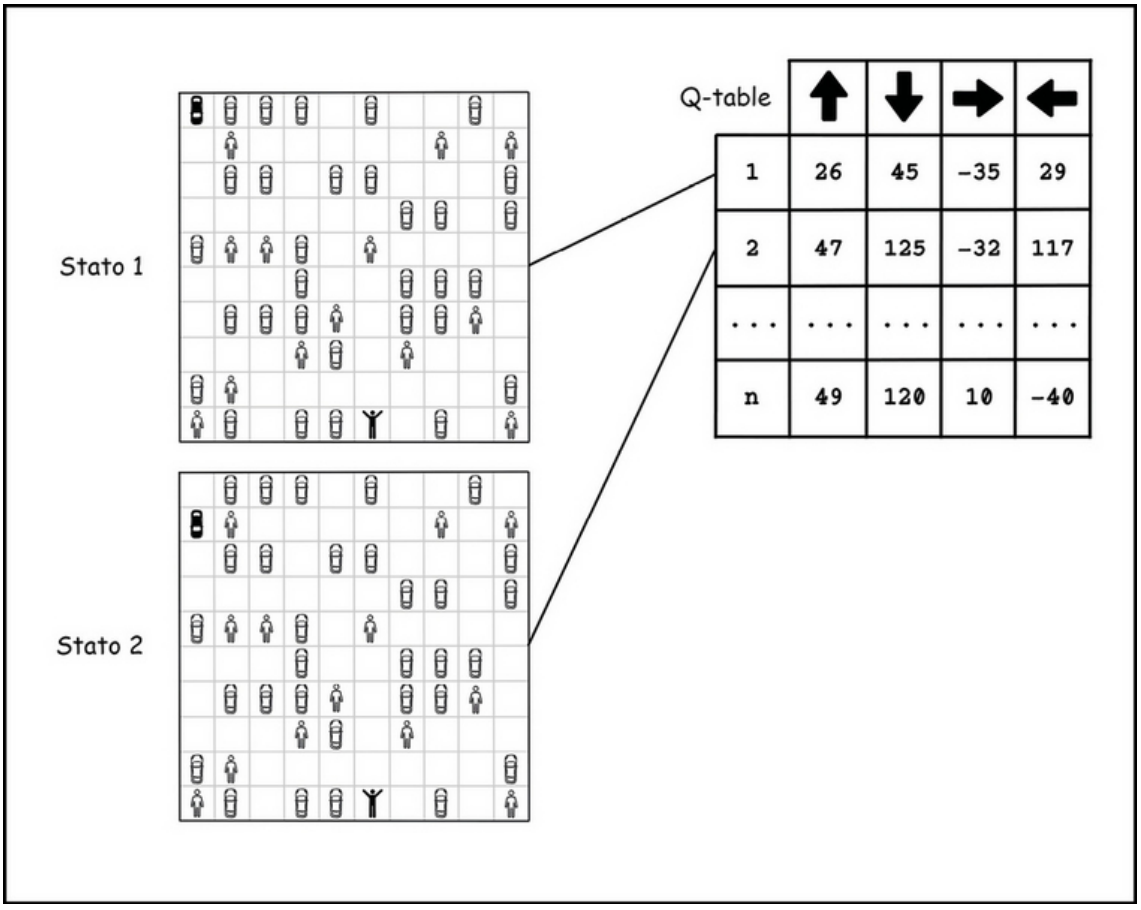


Figura 10.11 Un esempio di Q-table e gli stati che rappresenta.

Questo approccio può sembrare banale, ma un blocco adiacente potrebbe contenere un'altra auto o un pedone, oppure potrebbe essere un blocco vuoto o un blocco esterno: quattro possibilità per blocco, quindi 65.536 stati possibili. Data questa grande varietà, avremo bisogno di addestrare l'agente in molte configurazioni di parcheggio e molte volte affinché impari le buone scelte nelle azioni a breve termine (Figura 10.12).

Tenete presente l'idea di una tabella dei premi mentre esploriamo il ciclo di vita dell'addestramento di un modello utilizzando l'apprendimento per rinforzo con Q-learning. Rappresenterà il modello per le azioni che l'agente intraprenderà nell'ambiente.

Diamo un'occhiata al ciclo di vita di un algoritmo Q-learning, inclusi i passaggi coinvolti nell'addestramento. Esamineremo due fasi: l'inizializzazione e ciò che accade nelle diverse iterazioni a mano a mano che l'algoritmo apprende (Figura 10.13).

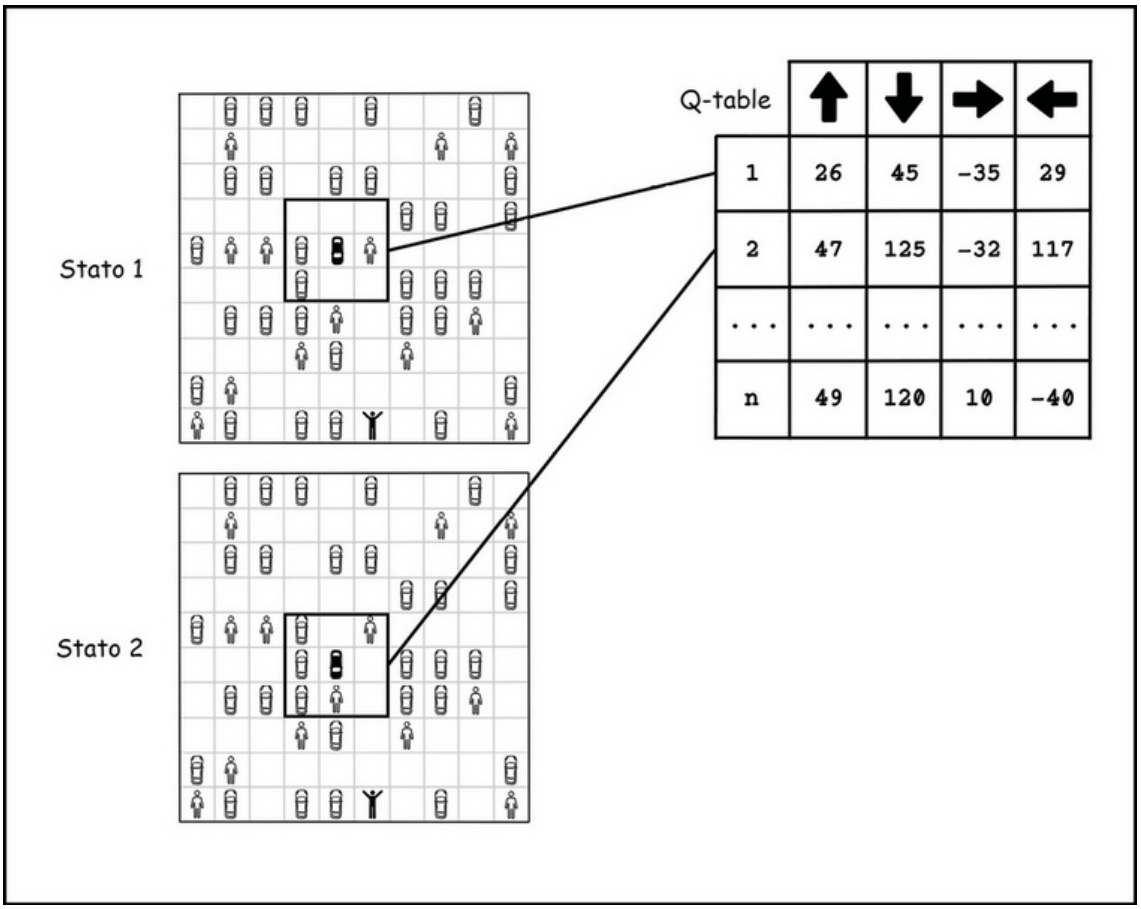


Figura 10.12 Un esempio migliore di Q-table e gli stati che rappresenta.

Inizializzazione. La fase di inizializzazione prevede l'impostazione dei parametri e dei valori iniziali per la Q-table.

1. *Inizializza la Q-table*. Inizializza una Q-table in cui ogni colonna rappresenta un'azione e ogni riga rappresenta un possibile stato. Notate che gli stati possono essere aggiunti alla tabella a mano a mano che vengono incontrati, perché all'inizio può essere difficile

conoscere il numero di stati totali dell'ambiente. I valori iniziali per ogni stato sono inizializzati a 0.

2. *Imposta i parametri*. Questo passaggio comporta l'impostazione dei parametri per gli iperparametri dell'algoritmo Q-learning, fra cui i seguenti.
   - *Possibilità di scegliere un'azione casuale*: questa è la soglia del valore per la scelta di un'azione casuale rispetto alla scelta di un'azione tratta dalla Q-table.
   - *Tasso di apprendimento*: è simile al tasso di apprendimento nel caso dell'apprendimento con supervisione. Descrive la velocità con cui l'algoritmo apprende dai premi nei diversi stati. Con un tasso di apprendimento elevato, i valori nella Q-table cambiano in modo irregolare e con un tasso di apprendimento basso, i valori cambiano gradualmente ma potrebbero essere necessarie più iterazioni per trovare valori validi.
   - *Fattore di sconto*: descrive quanto vengono valutate le potenziali ricompense future, il che si traduce nel favorire una gratificazione immediata o una ricompensa a lungo termine. Un valore piccolo favorisce le ricompense immediate; un valore elevato favorisce le ricompense a lungo termine.

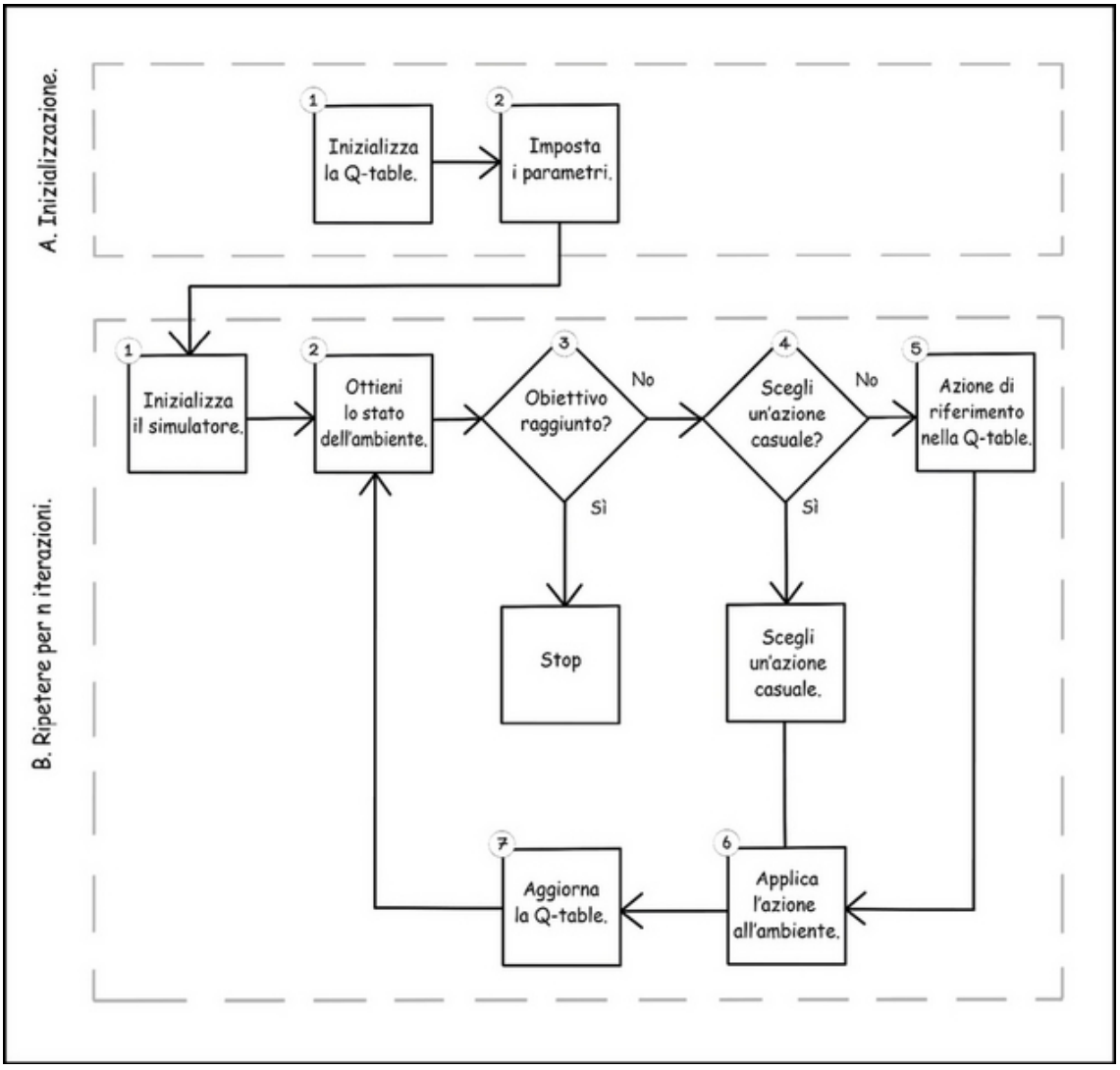


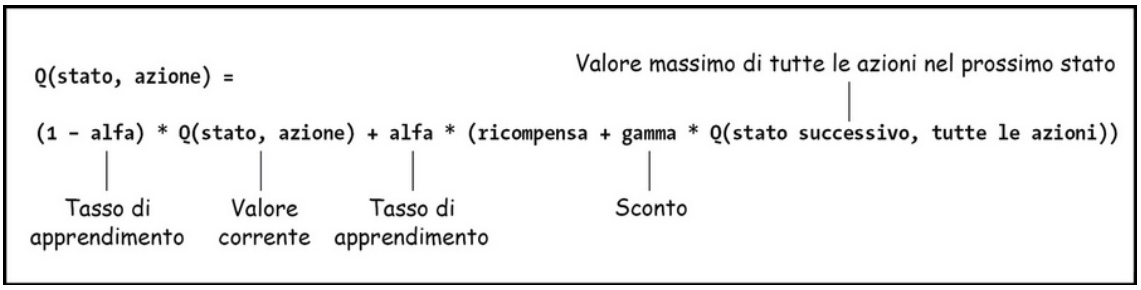
Figura 10.13 Ciclo di vita di un algoritmo di apprendimento per rinforzo Q-learning.

Ripeti per n iterazioni. I seguenti passaggi vengono ripetuti per trovare le azioni migliori negli stessi stati, valutandoli più volte. La stessa Q-table verrà aggiornata a tutte le iterazioni. Il concetto chiave è che, poiché la sequenza di azioni di un agente è importante, la ricompensa per un'azione in un determinato stato può cambiare in base alle azioni precedenti. Per questo motivo è importante prevedere più iterazioni. Considerate un'iterazione come un singolo tentativo di raggiungere un obiettivo.

1. *Inizializza il simulatore*. Questo passaggio comporta il ripristino dell'ambiente allo stato iniziale, con l'agente in uno stato neutro.
2. *Ottieni lo stato dell'ambiente*. Questa funzione dovrebbe fornire lo stato corrente dell'ambiente. Lo stato dell'ambiente cambierà dopo l'esecuzione di ogni azione.
3. *Obiettivo raggiunto?* Determina se l'obiettivo è stato raggiunto (o il simulatore ritiene che l'esplorazione sia completa). Nel nostro esempio, questo obiettivo è raggiungere il proprietario. Se l'obiettivo viene raggiunto, l'algoritmo termina.
4. *Scegli un'azione casuale*. Determina se deve essere selezionata un'azione casuale (Nord, Sud, Est o Ovest). Le azioni casuali sono utili per esplorare le possibilità nell'ambiente invece di apprendere un sottoinsieme ristretto.
5. *Azione di riferimento nella Q-table*. Se non viene presa la decisione di selezionare un'azione casuale, lo stato dell'ambiente corrente viene trasposto nella Q-table e l'azione viene selezionata in base ai valori contenuti nella tabella. Più avanti troverete maggiori informazioni sulla Q-table.
6. *Applica l'azione all'ambiente*. Questo passaggio comporta l'applicazione dell'azione selezionata, indipendentemente dal fatto che tale azione sia casuale o selezionata dalla Q-table. Un'azione avrà una conseguenza nell'ambiente e produrrà una ricompensa.
7. *Aggiorna la Q-table*. Di seguito descrivo i concetti coinvolti nell'aggiornamento della Q-table e i passaggi che vengono eseguiti.

L'aspetto chiave del Q-learning è l'equazione utilizzata per aggiornare i valori della Q-table. Questa equazione si basa sull'equazione *di Bellman*, che determina il valore di una decisione presa in un determinato momento, data la ricompensa o la penalità per

averla presa. L'equazione di Q-learning è un adattamento dell'equazione di Bellman. Nell'equazione di Q-learning, le proprietà più importanti per l'aggiornamento dei valori della Q-table sono lo stato corrente, l'azione, lo stato successivo in base all'azione e il risultato della ricompensa. Il tasso di apprendimento è simile a quello dell'apprendimento con supervisione, che determina la misura in cui viene aggiornato un valore presente nella Q-table. Lo sconto viene utilizzato per indicare l'importanza dei possibili premi futuri, e viene utilizzato per bilanciare il favore per i premi immediati rispetto ai premi a lungo termine:



Poiché la Q-table è inizializzata a 0, nello stato iniziale dell'ambiente sarà simile a quella rappresentata nella Figura 10.14.

Successivamente, vediamo come aggiornare la Q-table utilizzando l'equazione di Q-learning basata su diverse azioni con diversi valori per la ricompensa. Questi valori verranno utilizzati per il tasso di apprendimento (alfa) e lo sconto (gamma):

- Tasso di apprendimento (alfa) = 0,1
- Sconto (gamma) = 0,6

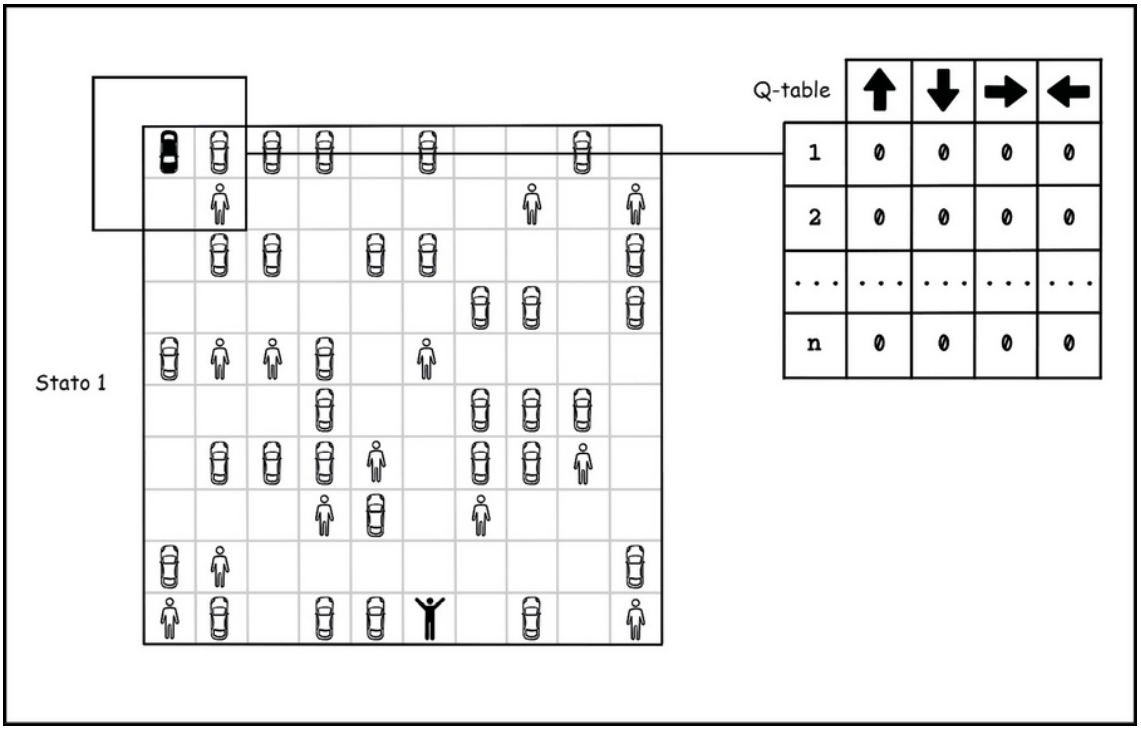


Figura 10.14 Un esempio di Q-table inizializzata.

La Figura 10.15 mostra come viene utilizzata l'equazione di Q-learning per aggiornare la Q-table se l'agente seleziona l'azione Est dallo stato iniziale nella prima iterazione. Ricordate che la Q-table iniziale è composta da tutti valori a 0. Il tasso di apprendimento (alfa), lo sconto (gamma), il valore dell'azione corrente, la ricompensa e il miglior stato successivo vengono inseriti nell'equazione per determinare il nuovo valore per l'azione intrapresa. L'azione è Est, che si traduce in una collisione con un'altra auto, che produce come "ricompensa" -100. Dopo aver calcolato il nuovo valore, il valore di Est nello Stato 1 è -10.

Il calcolo successivo riguarda lo stato successivo nell'ambiente in seguito all'azione intrapresa. L'azione è Sud e si traduce in una collisione con un pedone, che frutta -1.000 come "ricompensa". Dopo aver calcolato il nuovo valore, il valore per l'azione Sud sullo Stato 2 è -100 (Figura 10.16).

La Figura 10.17 mostra come i valori calcolati differiscono in una Q-table con i valori popolati, perché abbiamo lavorato su una Q-table inizializzata a 0. La figura è un esempio dell'equazione di Q-learning aggiornata dallo stato iniziale dopo varie iterazioni. La simulazione può essere eseguita più volte, per imparare da più tentativi. Quindi, questa iterazione ha successo molto prima, perché i valori della tabella sono stati aggiornati. L'azione per Est si traduce in una collisione con un'altra auto e frutta -100 come "ricompensa". Dopo che il nuovo valore è stato calcolato, il valore per Est nello Stato 1 passa a -34.

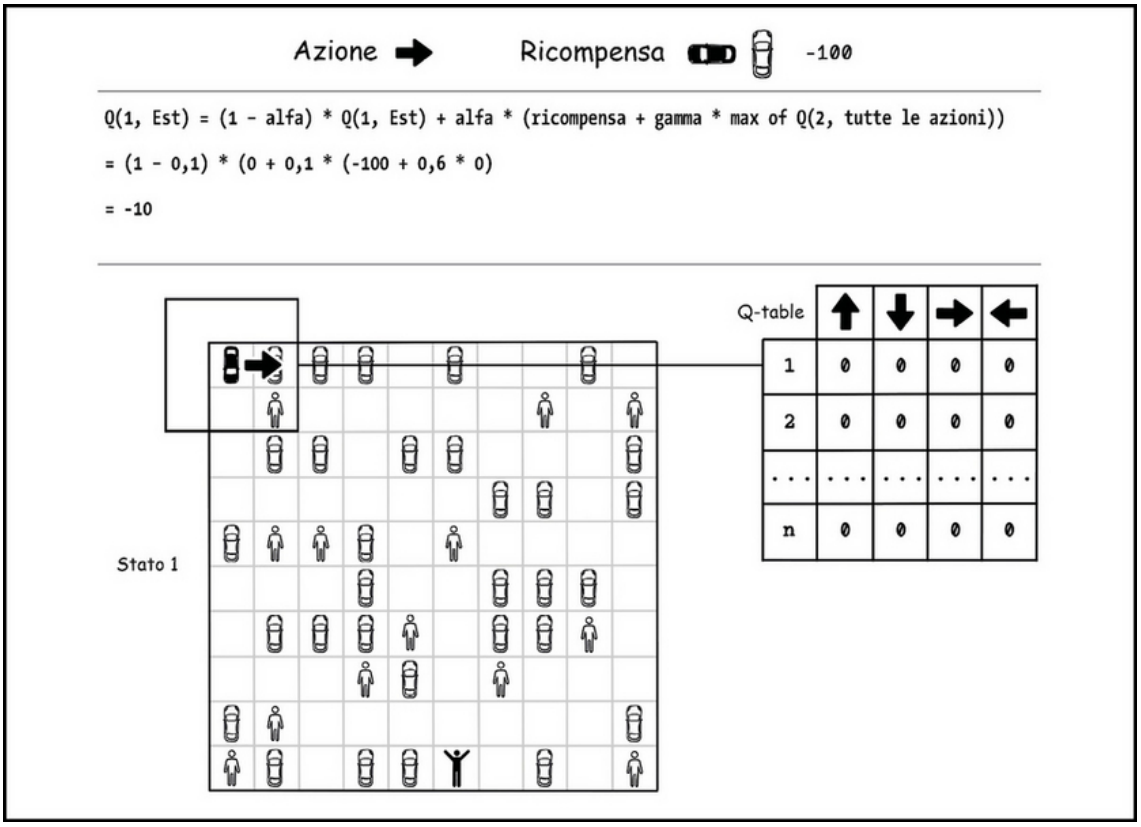


Figura 10.15 Esempio di calcolo dell'aggiornamento della Q-table per lo Stato 1.

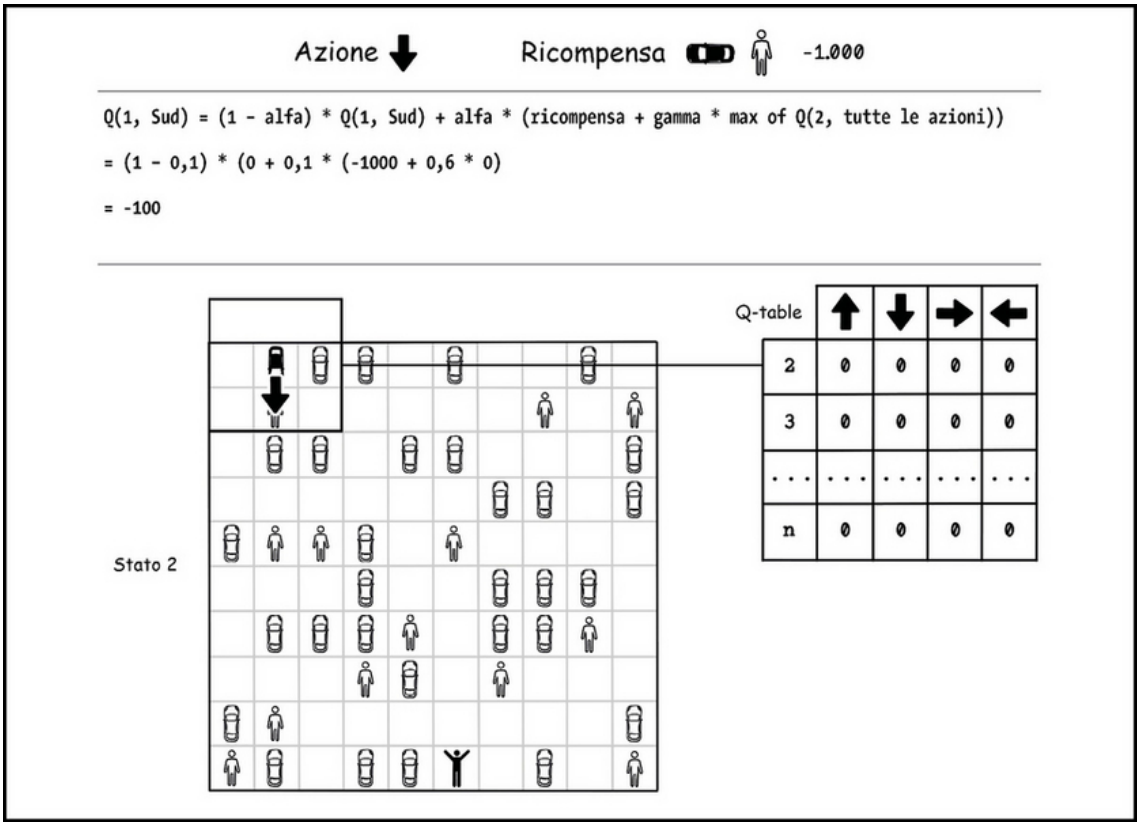


Figura 10.16 Esempio di calcolo dell'aggiornamento della Q-table per lo Stato 2.

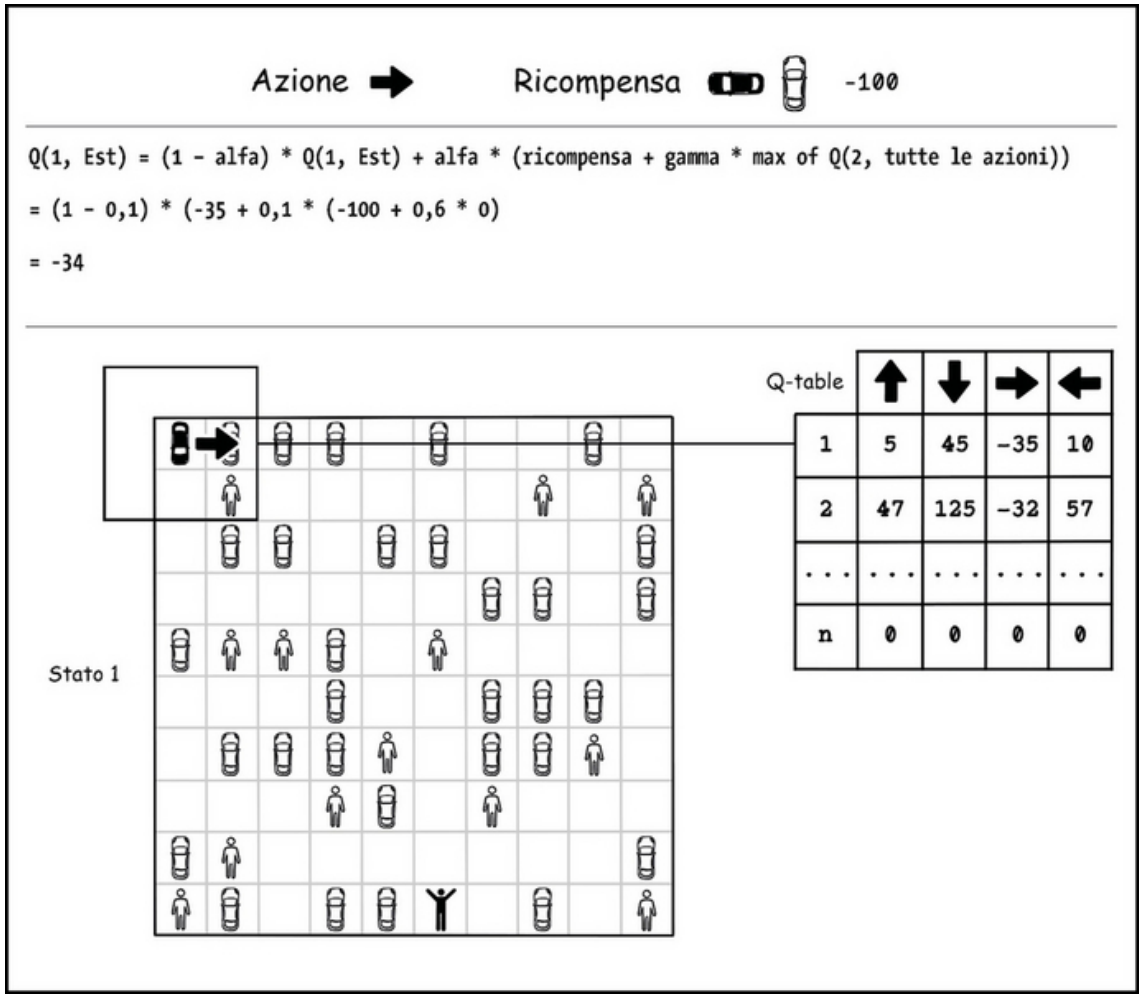
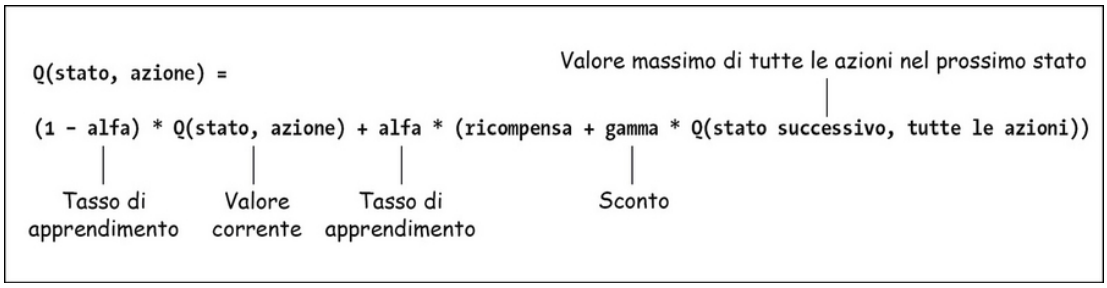
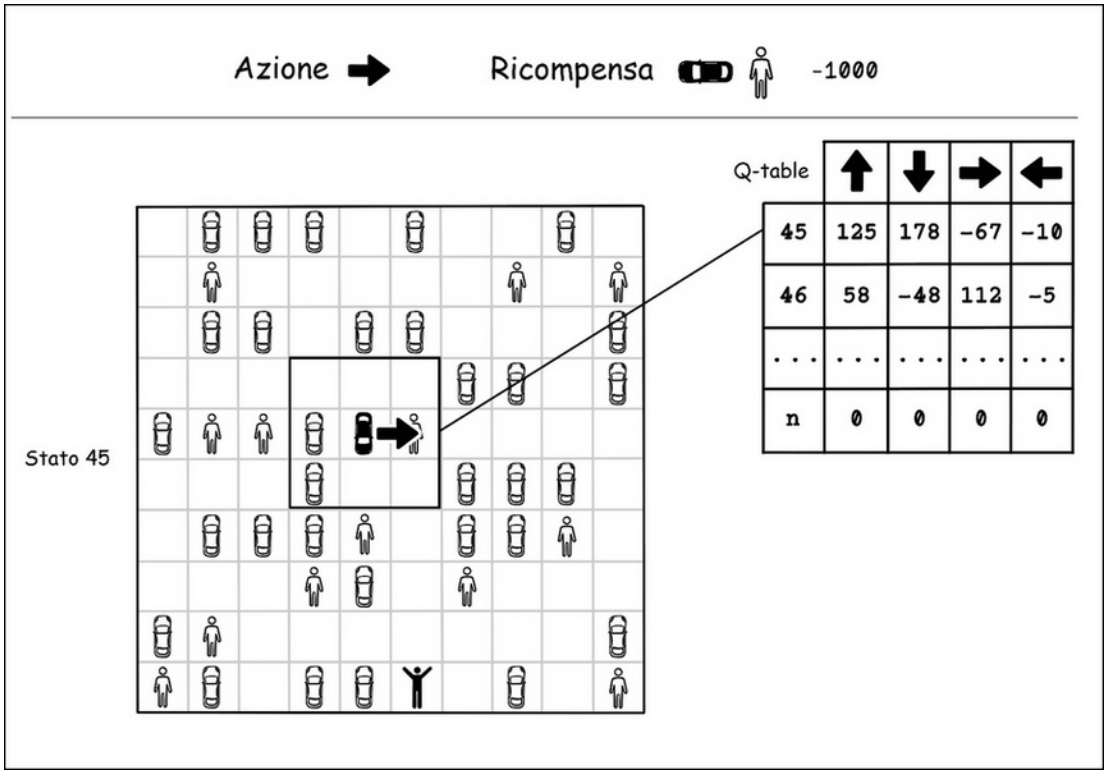


Figura 10.17 Esempio di calcolo dell'aggiornamento della Q-table per lo Stato 1 dopo diverse iterazioni.

Esercizio: calcolare la variazione dei valori per la Q-table

Utilizzando l'equazione di aggiornamento del Q-learning e lo scenario seguente, calcolare il nuovo valore per l'azione eseguita. Supponiamo che l'ultima mossa sia stata Est con un valore di -67:





Soluzione

I valori degli iperparametri e dello stato vengono inseriti nell'equazione di Q-learning, cosa che produce un nuovo valore per Q(1, Est):

- Tasso di apprendimento (alfa) = 0,1
- Sconto (gamma) = 0,6
- Q(1, Est) = -67
- Max di Q(2, tutte le azioni) = 112

```
Q(1, Est) = (1 - alfa) * Q(1, Est) + alfa * (ricompensa + gamma * max of
Q(2, tutte le azioni))
= (1 - 0,1) * (-67 + 0,1 * (-100 + 0,6 * 112)
= -64
```

Pseudocodice

Questo pseudocodice descrive una funzione che addestra una Q-table utilizzando il Q-learning. Potrebbe essere suddiviso in funzioni più semplici, ma è rappresentato in questo modo per migliorarne la leggibilità. La funzione segue i passaggi descritti in questo capitolo.

La Q-table è inizializzata a 0; poi la logica di apprendimento viene eseguita per diverse iterazioni. Ricordate che un'iterazione è un tentativo di raggiungere l'obiettivo.

Il prossimo frammento di codice viene eseguito quando l'obiettivo non è ancora stato raggiunto.

1. Decidi se è necessario intraprendere un'azione casuale per esplorare le possibilità nell'ambiente. In caso contrario, seleziona dalla Q-table l'azione di valore più elevato per lo stato corrente.
2. Procedi con l'azione selezionata e applicala al simulatore.
3. Raccogli informazioni dal simulatore, inclusa la ricompensa, lo stato successivo dell'azione e se l'obiettivo è stato raggiunto.
4. Aggiorna la Q-table in base alle informazioni raccolte e agli iperparametri. Notate che in questo codice gli iperparametri vengono passati come argomenti di questa funzione.
5. Imposta lo stato corrente in base all'esito dell'azione appena eseguita.

Questi passaggi continueranno fino a quando non verrà trovato un obiettivo. Dopo aver trovato l'obiettivo e aver raggiunto il numero desiderato di iterazioni, il risultato è una Q-table addestrata che può essere sottoposta a test in altri ambienti. Nel prossimo paragrafo vedremo come sottoporre a test la Q-table:

```
train_with_q_learning(observation_space, action_space,
       number_of_iterations, learning_rate,
        discount, chance_of_random_move):
    let q_table equal a matrix of zeros[observation_space, action_space]
    for i in range(number_of_iterations):
        let simulator equal Simulator(DEFAULT_ROAD,
                DEFAULT_ROAD_SIZE_X, DEFAULT_ROAD_SIZE_Y,
                DEFAULT_START_X, DEFAULT_START_Y,
                DEFAULT_GOAL_X, DEFAULT GOAL Y)
        let state equal simulator.get_state()
        let done equal False
        while not done:
            if random.uniform(0, 1) > chance_of_random_move:
                let action equal get_random_move()
            else:
                let action max(q_table[state])
            let reward equal simulator.move_agent(action)
            let next_state equal simulator.get_state()
```

```
            let done equal simulator.is_goal_achieved()
            let current_value equal q_table[state, action]
            let next_state_max_value equal max(q_table[next_state])
                 let new_value equal(1 - learning_rate) * current_value +
learning_rate *
 (reward + discount * next_state_max_value)
            let q_table[state, action] equal new_value
 let state equal next_state
    return a table
```

Test con la simulazione e Q-table

Sappiamo che nel caso del Q-learning, la Q-table è il modello che racchiude gli apprendimenti. Quando viene affrontato un nuovo ambiente con altri stati, l'algoritmo fa riferimento al rispettivo stato nella Q-table, e sceglie l'azione con il valore più elevato. Poiché la Q-table è già stata addestrata, questo processo consiste nell'ottenere lo stato attuale dell'ambiente e fare riferimento al rispettivo stato nella Q-table per trovare un'azione adeguata, fino al raggiungimento dell'obiettivo (Figura 10.18).

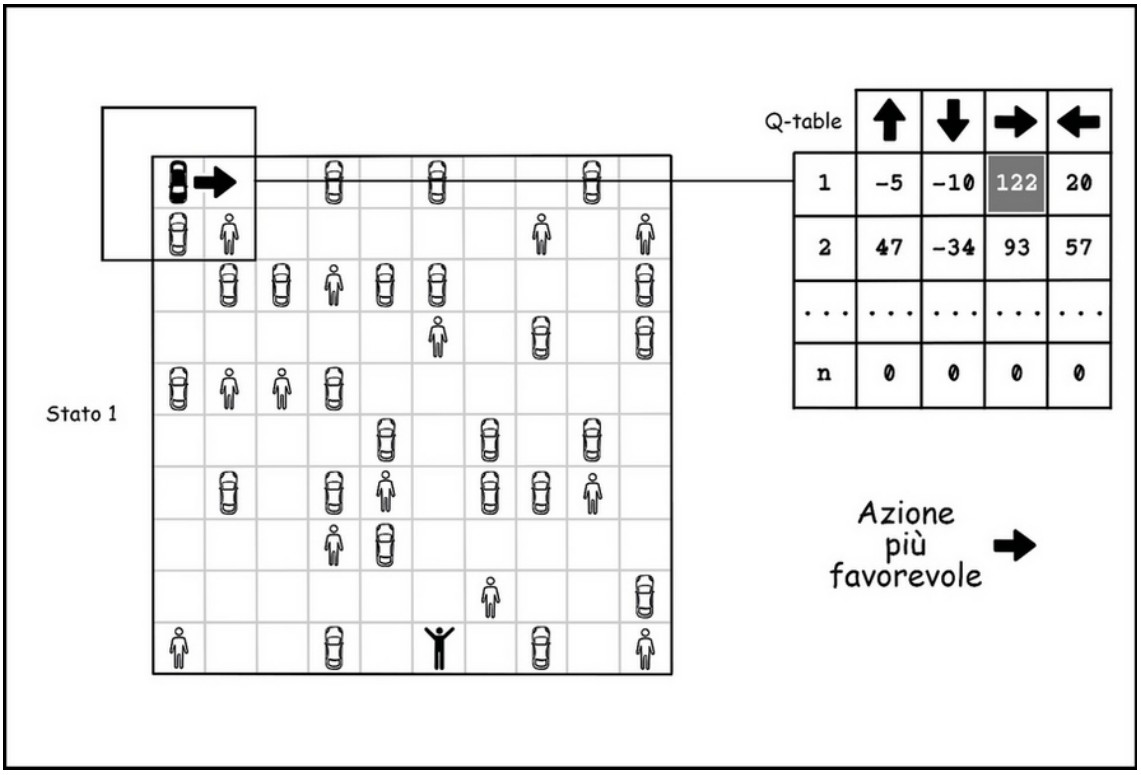


Figura 10.18 Riferimento a una Q-table per determinare quale azione intraprendere.

Poiché lo stato appreso nella Q-table considera gli oggetti adiacenti alla posizione attuale dell'agente, la Q-table ha appreso mosse buone e cattive in base a ricompense a breve termine, quindi potrebbe essere utilizzata in un parcheggio avente un'altra configurazione, come quella mostrata in Figura 10.18. Lo svantaggio è che l'agente finisce per preferire le ricompense a breve termine rispetto a quelle a lungo termine, perché non ha il contesto del resto della mappa quando intraprende ogni azione.

Un concetto che probabilmente emergerà nel processo di apprendimento per rinforzo è quello degli *episodi*. Un *episodio* include tutti gli stati fra lo stato iniziale e lo stato in cui l'obiettivo viene raggiunto. Se ci vogliono 14 azioni per raggiungere un obiettivo, abbiamo un episodio di 14 stati. Se l'obiettivo non viene mai raggiunto, l'episodio si dice *infinito*.

Misurare le prestazioni dell'addestramento

Gli algoritmi di apprendimento per rinforzo possono essere difficili da valutare. Dato un ambiente e un obiettivo, potremmo avere sanzioni e ricompense differenti, alcune delle quali hanno un effetto maggiore rispetto ad altre nel contesto del problema. Nell'esempio del parcheggio, penalizziamo pesantemente le collisioni con i pedoni. In un altro esempio, potremmo avere un agente che somiglia a un essere umano e cerca di imparare quali muscoli usare per camminare il più lontano possibile. In questo scenario, le penalità potrebbero essere il fatto di cadere, o qualcosa di più specifico, come una lunghezza eccessiva del passo. Per misurare le prestazioni in modo accurato, abbiamo bisogno di conoscere il contesto del problema.

Un modo generico per misurare le prestazioni consiste nel contare il numero di penalità in un dato numero di tentativi. Le penalità potrebbero essere eventi che vogliamo evitare nell'ambiente, come conseguenza di un'azione.

Un'altra misura delle prestazioni dell'apprendimento per rinforzo è la *ricompensa media per azione*. Massimizzando la ricompensa per azione, puntiamo a evitare le azioni sbagliate, indipendentemente dal fatto che l'obiettivo sia stato raggiunto o meno. Questa misurazione può essere calcolata dividendo la ricompensa cumulativa per il numero totale di azioni.

Apprendimento model-free e model-based

Per supportare le vostre future esperienze nell'apprendimento per rinforzo, tenete presente due approcci possibili: *basato su modello* e *senza modello*, diversi dai modelli di machine learning discussi in questo libro. Un modello è come una rappresentazione astratta dell'ambiente in cui opera un agente.

Potremmo avere in testa un modello riguardante la posizione dei punti di riferimento, conoscenze relative alla direzione o alla disposizione delle strade in un quartiere. Questo modello è stato addestrato dall'esplorazione di alcune strade, ma siamo in grado di simulare in mente degli scenari per prendere decisioni senza provare ogni opzione. Per decidere come procedere, possiamo utilizzare questo modello per prendere una decisione; questo approccio è quindi *basato su modelli*, *model-based*. L'apprendimento senza modelli, *model-free*, è simile all'approccio al Q-learning descritto in questo capitolo; procede per tentativi ed errori per esplorare molte interazioni con l'ambiente, al fine di determinare le azioni favorevoli in diversi scenari.

La Figura 10.19 illustra i due approcci alla navigazione stradale. È possibile utilizzare vari algoritmi per creare implementazioni di apprendimento per rinforzo basate su modelli.

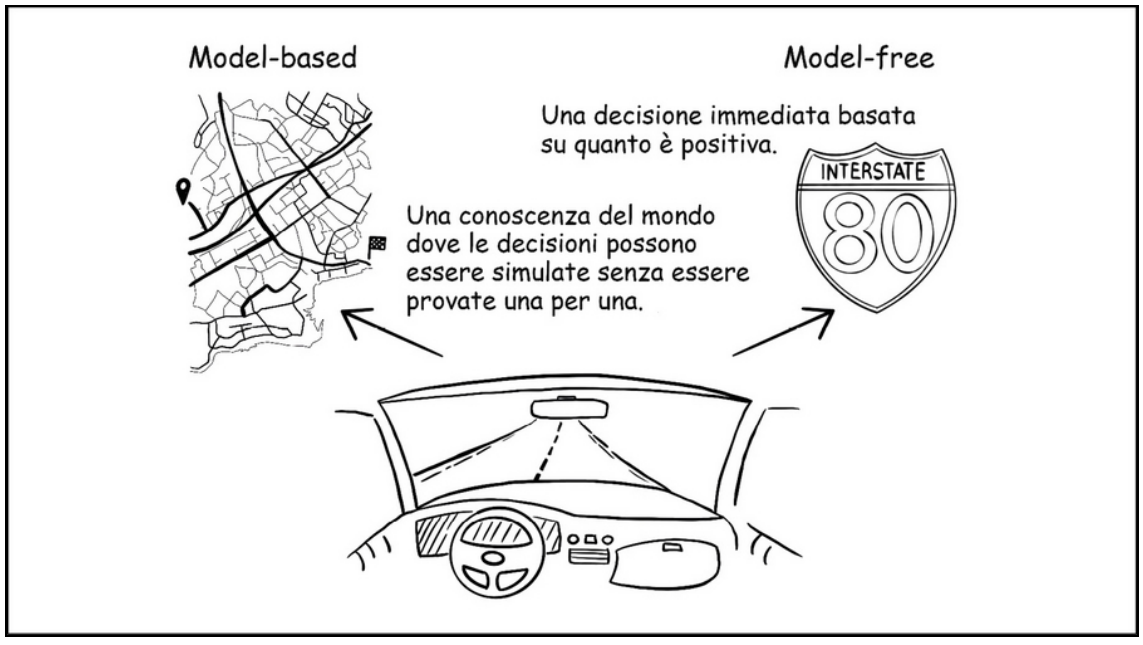


Figura 10.19 Esempi di apprendimento per rinforzo basato su modello e senza modello.

Approcci di deep learning all'apprendimento per rinforzo

Il Q-learning è solo uno degli approcci all'apprendimento per rinforzo. Avere una buona comprensione di come funziona vi consente di applicare lo stesso ragionamento e lo stesso approccio generale ad altri algoritmi di apprendimento per rinforzo. L'approccio da scegliere dipende dal problema da risolvere. Un'alternativa molto utilizzata è l'*apprendimento per rinforzo profondo*, utile per applicazioni di robotica, videogiochi e problemi che coinvolgono immagini e video.

L'apprendimento per rinforzo profondo può utilizzare reti neurali artificiali per elaborare gli stati di un ambiente e produrre un'azione. Le azioni vengono apprese regolando i pesi della rete neurale artificiale, utilizzando i feedback delle ricompense e i cambiamenti dell'ambiente. L'apprendimento per rinforzo può anche utilizzare le funzionalità delle reti neurali convoluzionali e altre architetture a rete neurale artificiale create appositamente per risolvere problemi specifici in diversi domini e casi d'uso.

La Figura 10.20 illustra, in generale, come una rete neurale artificiale può essere utilizzata per risolvere il problema del parcheggio presentato in questo capitolo. Gli input della rete neurale sono gli stati; gli output sono le probabilità della migliore selezione dell'azione per l'agente; la ricompensa e l'effetto sull'ambiente possono essere restituiti utilizzando la retropropagazione, per aggiustare i pesi nella rete.

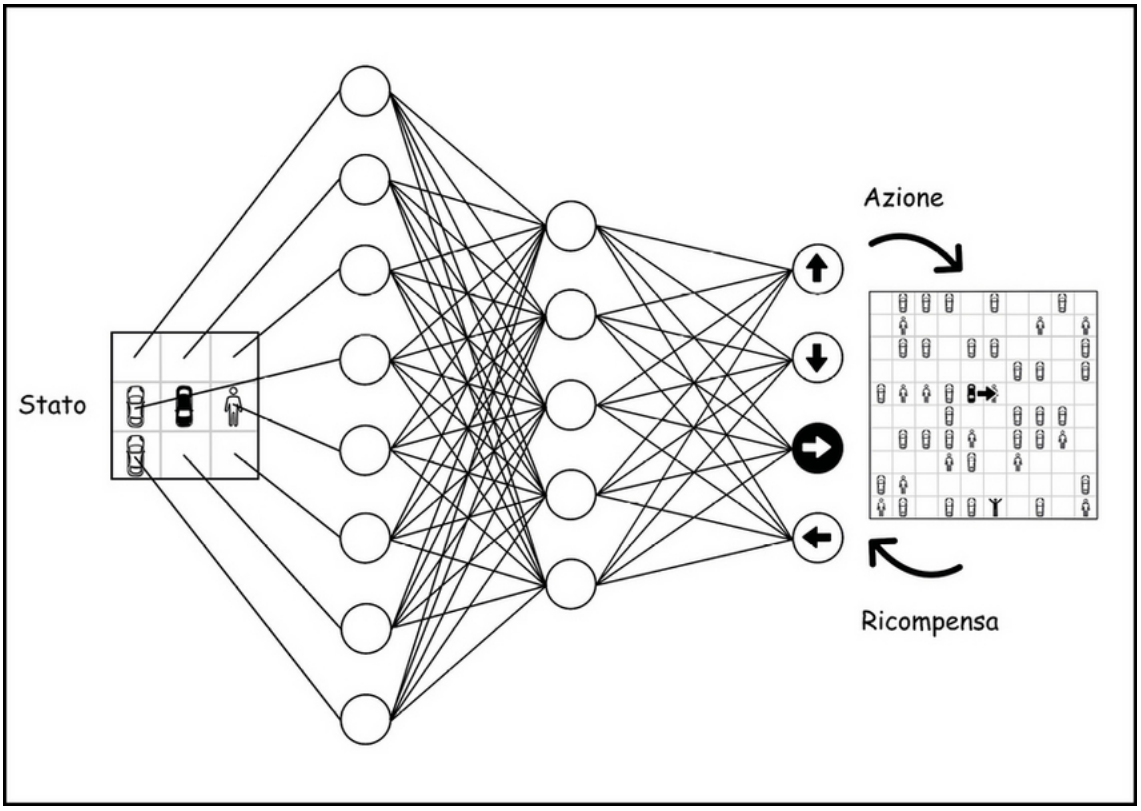


Figura 10.20 Esempio d'uso di una rete neurale artificiale per risolvere il problema del parcheggio.

Il prossimo paragrafo esamina alcuni casi d'uso classici per l'apprendimento per rinforzo nel mondo reale.

Casi d'uso per l'apprendimento per rinforzo

L'apprendimento per rinforzo ha molte applicazioni in cui ci sono pochi dati storici da cui imparare, o addirittura nessuno. L'apprendimento avviene attraverso l'interazione con un ambiente con la sola euristica di una buona prestazione. I casi d'uso per questo approccio sono potenzialmente infiniti. Questo paragrafo descrive solo alcuni dei più classici.

Robotica

La robotica implica la creazione di macchine che interagiscono con ambienti reali per raggiungere obiettivi. Alcuni robot vengono utilizzati per navigare su terreni difficili con vari tipi di superfici, ostacoli e pendenze. Altri vengono utilizzati come assistenti di laboratorio, prendendo istruzioni da uno scienziato, passando gli strumenti giusti o azionando attrezzature. Quando non è possibile modellare ogni risultato di ogni azione in un ambiente ampio e dinamico, l'apprendimento per rinforzo può essere utile. Definendo un obiettivo più generale in un ambiente e introducendo ricompense e penalità come euristica, possiamo utilizzare l'apprendimento per rinforzo per addestrare i robot in ambienti dinamici. Un robot che naviga nel terreno, per esempio, può imparare a quali ruote dare potenza e come regolare le sospensioni per attraversare con successo determinati passaggi. Questo obiettivo viene raggiunto dopo molti tentativi.

Questi scenari possono essere simulati virtualmente se gli aspetti chiave dell'ambiente possono essere modellati in un programma. I giochi per computer sono stati utilizzati in alcuni progetti come base per addestrare le auto a guida autonoma prima di addestrarle su strada nel mondo reale. Lo scopo nell'addestrare i robot con l'apprendimento per rinforzo è quello di creare modelli generali in grado di adattarsi ad ambienti nuovi e differenti mentre apprendono interazioni più generali, proprio come fanno gli esseri umani.

Motori di raccomandazione

I motori di raccomandazione sono utilizzati in molti dei prodotti digitali di uso comune. Le piattaforme di streaming video utilizzano motori di raccomandazione per apprendere i gusti di un individuo in termini di contenuti video e per provare a consigliare qualcosa di

gradito allo spettatore. Questo approccio è stato utilizzato anche nelle piattaforme di streaming musicale e nei negozi di e-commerce. I modelli di apprendimento per rinforzo vengono addestrati utilizzando il comportamento dello spettatore di fronte alla decisione di guardare i video consigliati. La premessa è che se un video consigliato è stato selezionato e guardato nella sua interezza, c'è una forte ricompensa per il modello di apprendimento per rinforzo, perché può presupporre che il video fosse un buon consiglio. Al contrario, se un video non viene mai selezionato o viene abbandonato poco dopo l'inizio, è ragionevole presumere che il video non abbia attratto lo spettatore. Questo risultato comporterebbe una ricompensa debole o una punizione.

Trading finanziario

Gli strumenti finanziari per il trading includono azioni, criptovalute e altri prodotti di investimento. Il trading è un problema difficile. Gli analisti monitorano i modelli delle variazioni di prezzo e le notizie dal mondo e usano le loro conoscenze per prendere una decisione se mantenere un investimento, venderne una parte o acquistarne di più. L'apprendimento per rinforzo può addestrare modelli che prendono queste decisioni attraverso ricompense e sanzioni basate sul guadagno o la perdita. Lo sviluppo di un modello di apprendimento per rinforzo per fare trading richiede molti tentativi ed errori, il che significa che si potrebbero perdere ingenti somme di denaro nell'addestramento dell'agente. Fortunatamente, la maggior parte dei dati finanziari pubblici storici è disponibile gratuitamente, e alcune piattaforme di investimento forniscono sandbox entro le quali sperimentare.

Sebbene un modello di apprendimento per rinforzo possa aiutare a generare un buon ritorno sull'investimento, sorge una domanda: se tutti gli investitori fossero automatizzati e completamente razionali e l'elemento umano fosse rimosso dal trading, come sarebbe il mercato?

Giochi

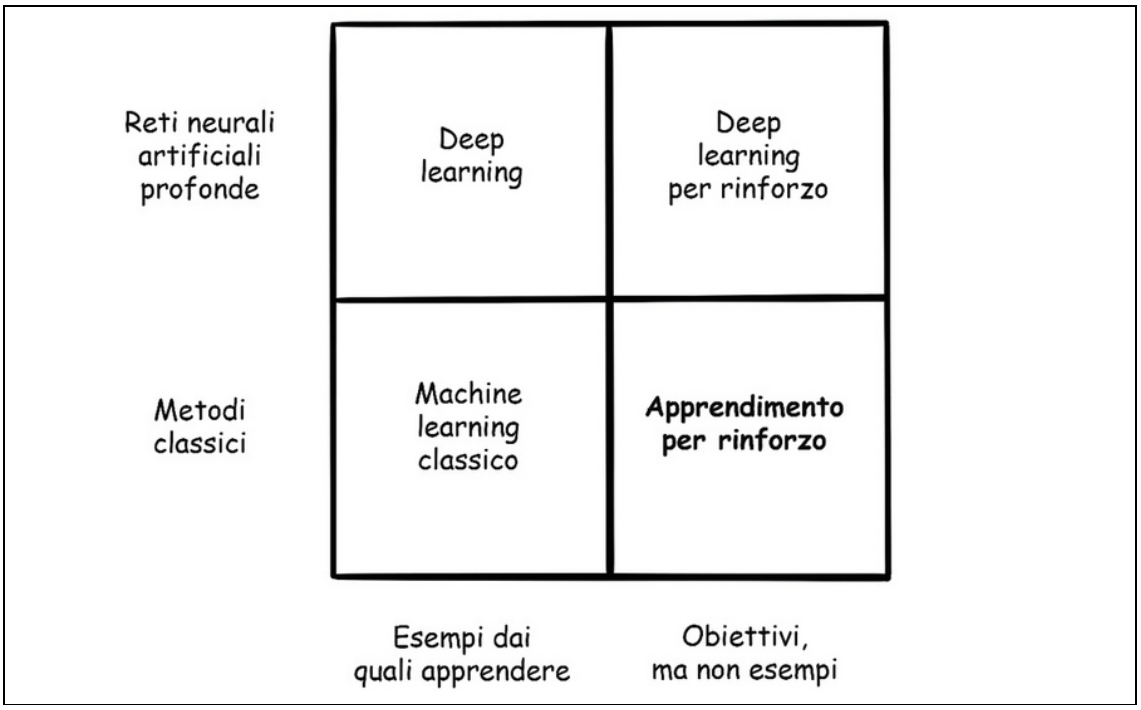
I giochi di strategia hanno spinto per anni le capacità intellettuali dei giocatori. Questi giochi, in genere, comportano la gestione di molti tipi di risorse con la pianificazione di tattiche a breve e lungo termine per battere un avversario. Questi giochi hanno riempito le arene e i più piccoli errori sono costati ai giocatori di alto livello molte partite. L'apprendimento per rinforzo è stato utilizzato per giocare a questi giochi a livello professionale. Queste implementazioni dell'apprendimento per rinforzo di solito coinvolgono un agente che guarda lo schermo come farebbe un giocatore umano, apprende gli schemi e intraprende azioni. I premi e le penalità sono direttamente associati al gioco. Dopo molte iterazioni di gioco in diversi scenari con diversi avversari, un agente di apprendimento per rinforzo impara le tattiche che funzionano meglio per l'obiettivo a lungo termine di vincere la partita. L'obiettivo della ricerca in questo spazio è legato alla ricerca di modelli più generali che possano trarre un contesto da stati e ambienti astratti, e comprendere cose che non possono essere derivate in modo logico. Da bambini, per esempio, ci è bastato scottarci una vola per apprendere che gli oggetti che scottano sono potenzialmente pericolosi. Abbiamo sviluppato una certa conoscenza e poi l'abbiamo sottoposta a test, crescendo. Questi test hanno rafforzato la nostra comprensione degli oggetti che scottano e del loro potenziale danno o beneficio.

Alla fine, la ricerca e lo sviluppo dell'intelligenza artificiale cercano di far imparare ai computer a risolvere i problemi in modi in cui gli esseri umani sono già bravi: in generale, si tratta di mettere insieme idee e concetti astratti avendo un obiettivo in mente e trovando buone soluzioni ai problemi.

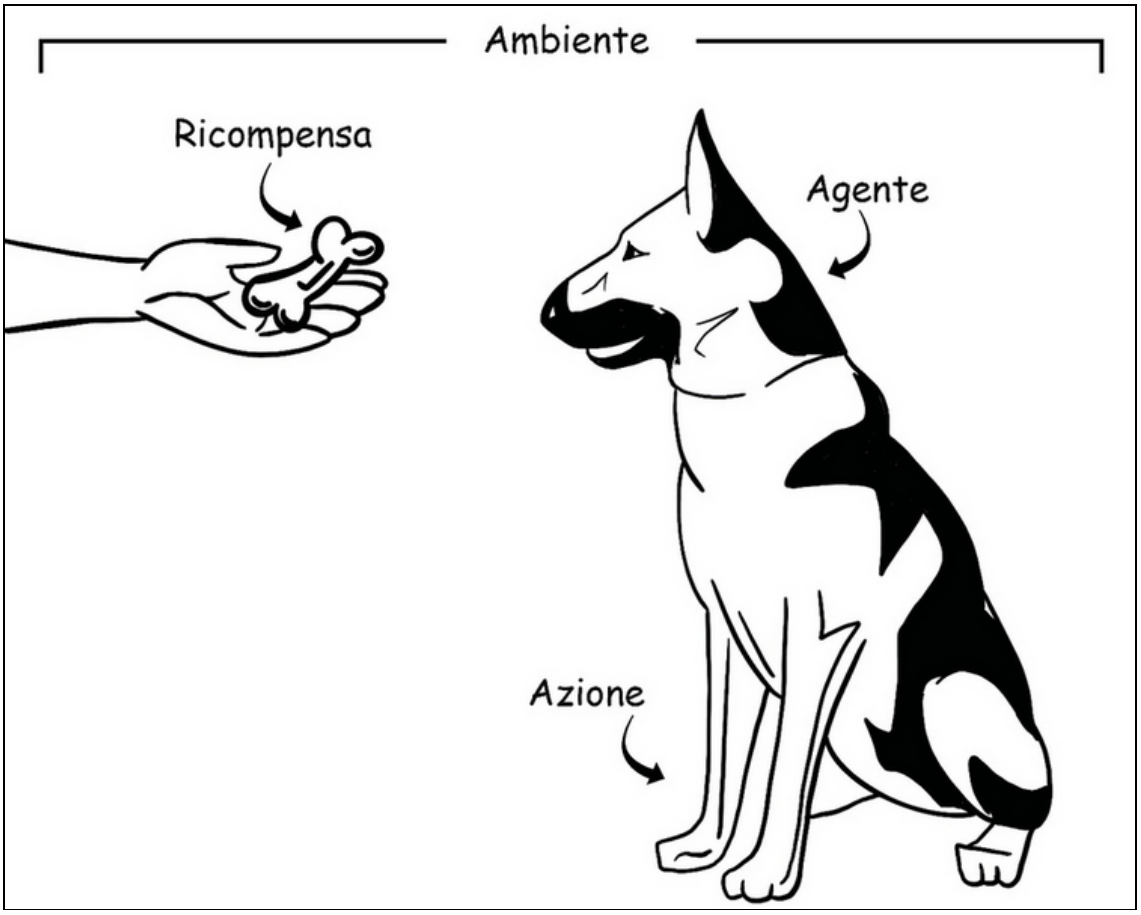
Riepilogo

L'apprendimento per rinforzo è applicabile quando è noto un obiettivo, ma non gli esempi dai quali apprendere.

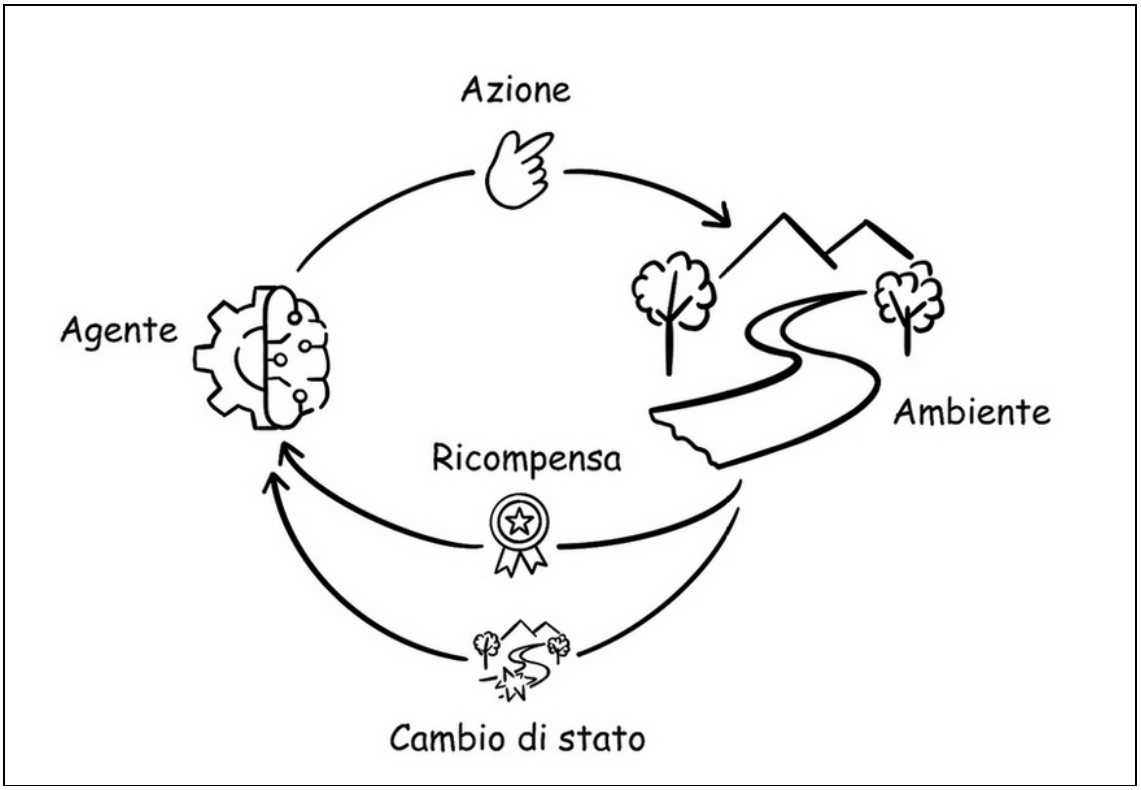
- L'apprendimento per rinforzo può utilizzare metodi classici o metodi basati su reti neurali artificiali.



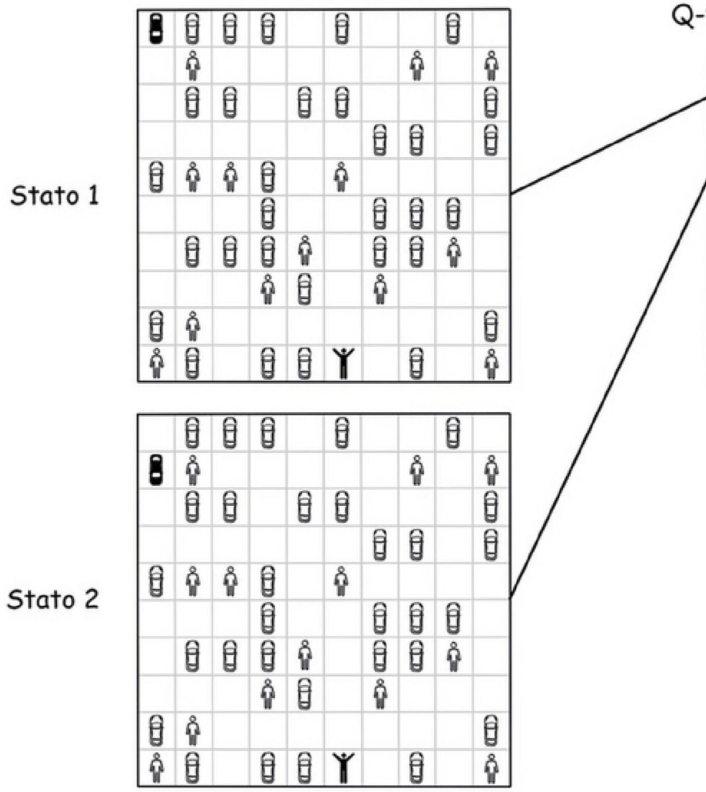
- Per apprendere in un ambiente si utilizzano tentativi ed errori.



- Una Q-Table è costituita da azioni (colonne) e stati (righe).



- Il Q-Learning sfrutta una Q-Table e una funzione di apprendimento per imparare dalle azioni scelte.



Q-table	↑	↓	→	←
1	26	45	-35	29
2	47	125	-32	117
...	...	...	...	...
n	49	120	10	-40

```
Q(stato, azione) =

(1 - alfa) * Q(stato, azione) + alfa * (ricompensa + gamma * Q(stato successivo, tutte le azioni))
```

- (1 - alfa): Tasso di apprendimento
- Q(stato, azione): Valore corrente
- alfa: Tasso di apprendimento
- gamma: Sconto
- Q(stato successivo, tutte le azioni): Valore massimo di tutte le azioni nel prossimo stato