Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in
Ingegneria Informatica

Algoritmi ispirati alla natura

Relatore:
Ch.mo Prof. Geppino Pucci

Laureando:
Alessandro Corrò

Anno Accademico 2022-2023

Data di laurea: 29 settembre 2023

Indice

Sommario

Una tendenza scientifica che è emersa a metà degli anni '40, ma che ha ricevuto molta attenzione negli ultimi due o tre decenni, è la fusione di idee provenienti dalla natura e dall'informatica, termini apparentemente antitetici.
Il Natural Computing è un campo interdisciplinare che trae ispirazione dai processi naturali, come quelli osservati negli organismi viventi e nei sistemi biologici, per sviluppare nuovi approcci e paradigmi di calcolo. Tra le molte tecniche utilizzate, gli algoritmi *Ant Colony Optimization* sono particolarmente rilevanti per la loro capacità di risolvere problemi complessi attraverso l'imitazione del comportamento delle colonie di formiche nella ricerca di cibo.
L'obiettivo principale di questa tesi è quello di esplorare le varianti degli algoritmi ACO e le loro applicazioni in diversi contesti. In particolare offre un'analisi dettagliata dell'applicazione degli algoritmi ACO al *Problema del Commesso Viaggiatore*, esplorando le fondamenta teoriche e i meccanismi di base di questa metodologia.

Capitolo 1

Natural Computing

Nell'era moderna, in cui l'elaborazione delle informazioni e la computazione sono diventate parte essenziale della nostra quotidianità, l'ispirazione proveniente dalla natura ha dimostrato di essere una risorsa inestimabile per lo sviluppo di nuovi paradigmi computazionali. Il *Natural Computing* [1] rappresenta un campo interdisciplinare in cui concetti, modelli e meccanismi tratti dal mondo naturale sono applicati per risolvere problemi complessi e affrontare sfide computazionali che spesso sfuggono alle soluzioni tradizionali.
Da algoritmi evolutivi ispirati al processo di selezione naturale alla simulazione di comportamenti di colonie di insetti sociali, passando per reti neurali artificiali che si basano sulla struttura del cervello umano, il Natural Computing offre un'ampia gamma di approcci che spaziano dalla biologia alla matematica, dall'informatica all'ingegneria. In modo particolare, questo campo di ricerca si propone di catturare l'essenza di fenomeni naturali complessi e di tradurli in strategie computazionali innovative.

La crescente complessità dei problemi che affrontiamo richiede nuove prospettive e strumenti per affrontarli in modo efficiente ed efficace. Le tradizionali metodologie computazionali, seppur potenti, possono raggiungere limiti quando si tratta di problemi che coinvolgono elevati livelli di incertezza, dinamiche complesse o enormi spazi di ricerca. In questi contesti, il Natural Computing, offre una serie di approcci flessibili e adattabili.

La computazione naturale può essere suddivisa in tre principali branche:

- Computazione ispirata alla natura, la cui idea principale è quella di sviluppare strumenti computazionali prendendo ispirazione dalla natura per la soluzione di problemi complessi.

- Simulazione ed emulazione della natura mediante il calcolo, basata essenzialmente su processi volti a creare modelli, forme e comportamenti per la sintesi e lo studio dei fenomeni naturali.

- Computazione con materiali naturali, la quale, mediante l'utilizzo di materiali innovativi per sostituire o integrare i computer basati sul silicio, costituisce un vero e proprio nuovo paradigma computazionale.

1.1 Computazione bio-ispirata

Tra tutti gli approcci di computazione naturale, gli algoritmi e i sistemi ispirati alla natura sono quelli sviluppati da più tempo, oltre ad essere i più popolari. Gli algoritmi bio-ispirati sono approcci computazionali che traggono ispirazione dai processi naturali, dai comportamenti e dalle strutture presenti nella biologia, al fine di risolvere problemi complessi. Questi algoritmi prendono spunto da fenomeni biologici e li adattano in soluzioni computazionali per affrontare sfide in vari campi, dalla matematica all'ottimizzazione, all'intelligenza artificiale e altro ancora. I più conosciuti sono: le reti neurali artificiali, gli algoritmi evolutivi, l'intelligenza di sciame e i sistemi immunitari artificiali.

1.1.1 Reti neurali artificiali

Le reti neurali artificiali, o *ANN* (*Artificial Neural Networks*), sono sistemi di elaborazione dell'informazione progettati sulla falsariga del sistema nervoso umano. Esse vengono utilizzate principalmente nell'apprendimento automatico per modellare relazioni complesse tra dati.

Il concetto di ANN trae ispirazione dalle reti neurali biologiche, che costituiscono la base del cervello umano. I neuroni costituiscono l'unità di base utilizzata dal cervello per la computazione e sono interconnessi tra loro mediante sinapsi. Questa interconnessione tra neuroni forma la cosiddetta "rete neurale", le cui caratteristiche principali sono la rappresentazione delle informazioni in modo distribuito e l'elaborazione parallela di queste informazioni. Il meccanismo più importante nelle reti neurali, biologiche e artificiali, è l'apprendimento mediante l'impostazione dell'efficienza sinaptica, considerato alla base della maggior parte delle capacità cognitive umane come percezione, pensiero e deduzione.

Da una prospettiva progettuale, una ANN è caratterizzata da tre aspetti: un insieme di nodi, una struttura e uno strumento per determinare i valori dei pesi associati ai collegamenti tra i nodi.

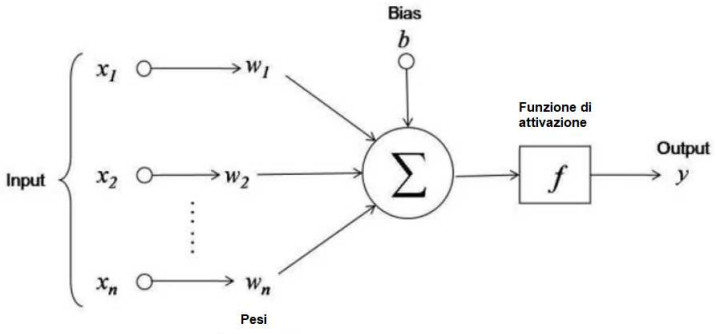

Figura 1.1: Schema di un neurone artificiale

La Fig. 1.1 illustra un neurone artificiale standard, composto da:

- **Sinapsi**, caratterizzate dai valori di peso.
- Un **giunto sommatore**, il quale somma tutti i segnali di ingresso pesati dai valori di peso sinaptico più una soglia prestabilita (*bias*) per consentire maggior flessibilità.
- Una **funzione di attivazione**, la quale determina l'uscita di un neurone in relazione al suo ingresso netto.

Matematicamente, l'uscita y_k del neurone k può essere descritta da una semplice equazione:

$$y_k = f(u_k) = f(\sum_{j=1}^{n} w_{kj}x_j + b_k)$$

Struttura
La maggior parte delle ANN utilizza architetture standardizzate appositamente progettate per risolvere problemi ingegneristici. In generale, è possibile distinguere tre tipi di architetture di rete: reti *feedforward* a singolo strato, reti *feedforward* a più strati e reti ricorrenti. Nel primo caso c'è un solo strato di nodi e i segnali vengono propagati in modo puramente positivo, ovvero dagli input verso gli output e mai in senso opposto. La seconda classe di ANN feedforward è nota come reti multistrato. Queste si distinguono dalle reti a singolo strato per la presenza di uno o più strati intermedi o nascosti, i quali aumentatano la capacità di elaborazione e memorizzazione della rete. Infine, la terza classe principale di reti è nota come reti ricorrenti, che si distinguono dalle reti feedforward per la presenza di almeno un ciclo ricorrente, o di *feedback*. In un'organizzazione di questo tipo c'è comunicazione tra i neuroni, può esserci una revisione del piano e possono essere prese decisioni intermedie.

Apprendimento
L'apprendimento corrisponde al processo mediante il quale i pesi della rete vengono regolati attraverso un meccanismo di presentazione di stimoli di input. Il tipo di apprendimento è definito dal modo in cui vengono regolati i pesi. I tre approcci principali all'apprendimento sono:

- **Apprendimento supervisionato**, in cui viene incorporato il concetto di supervisore onniscente che possiede la conoscenza sull'ambiente in cui opera la rete. I parametri liberi della rete (pesi) vengono regolati tramite la combinazione dei segnali di input e di errore, dove il segnale di errore e_j è la differenza tra l'output desiderato d_j e l'output corrente della rete y_j: $e_j = d_j - y_j$.
- **Apprendimento non supervisionato**, nel quale la rete si adatta alle regolarità statistiche dei dati di input. Solitamente, gli algoritmi di auto-organizzazione utilizzano uno schema di apprendimento competitivo, in cui i neuroni di output della rete competono tra loro per essere attivati.
- **Apprendimento per rinforzo**, il quale si differenzia dagli altri approcci poiché si basa sull'apprendimento da interazione diretta con l'ambiente. Ciò si basa su un *framework* che definisce l'interazione tra la ANN e il suo ambiente in termini dei valori correnti dei parametri liberi, della risposta della rete e delle ricompense.

L'apprendimento supervisionato e quello non supervisionato sono i paradigmi di apprendimento più popolari. Le reti neurali supervisionate più tradizionali sono il perceptron a singolo strato, l'elemento lineare adattativo (ADALINE), il perceptron a più strati e la rete a funzione di base radiale. La rete neurale non supervisionata più nota è la mappa delle caratteristiche auto-organizzante di Kohonen.

1.1.2 Computazione evolutiva

La computazione evolutiva è un campo di ricerca che si ispira alla biologia evolutiva per sviluppare tecniche di ricerca e ottimizzazione per risolvere problemi complessi. La maggior parte degli algoritmi evolutivi affonda le proprie radici nella biologia evolutiva darwiniana, la quale afferma essenzialmente che una popolazione di individui capaci di riprodursi e soggetti a variazioni genetiche seguite da selezione porta alla formazione di nuove popolazioni di individui sempre più adatti al loro ambiente.

I metodi di computazione evolutiva sono progettati per cercare soluzioni in uno spazio di ricerca complesso, dove le soluzioni possono essere rappresentate da vettori di parametri o strutture più complesse. L'idea fondamentale è di creare una popolazione iniziale P di n individui $P = \{x_1, x_2, ..., x_n\}$ e farla evolvere attraverso iterazioni successive. Con il passare delle iterazioni, le soluzioni tendono ad adattarsi meglio al problema, evolvendo verso soluzioni più efficaci o ottimali.
Un algoritmo evolutivo standard può quindi essere proposto come segue:

- **Una popolazione di individui che si riproducono con l'ereditarietà**. Ogni individuo rappresenta o codifica un punto nello spazio di ricerca delle possibili soluzioni a un problema. A questi individui è consentito riprodursi, generando discendenti che ereditano alcune caratteristiche dai loro genitori.

- **Variazione genetica**. I discendenti sono soggetti a variazioni genetiche attraverso la mutazione, che altera la loro composizione genetica. La mutazione consente l'insorgenza di nuove caratteristiche nei discendenti e, quindi, l'esplorazione di nuove regioni dello spazio di ricerca.

- **Selezione naturale**. La valutazione degli individui nel loro ambiente fornisce una misura di adattabilità, o valore di *fitness*, da assegnare loro. Un confronto dei valori di fitness degli individui porta a una competizione per la sopravvivenza e la riproduzione nell'ambiente, e vi sarà un vantaggio selettivo per gli individui con un fitness più elevato.

Nonostante le radici della computazione evolutiva risalgano agli anni '50 e '60, solo dagli anni '90 ha subito una crescita significativa. Le principali tendenze attuali nel campo includono indagini teoriche; applicazioni all'ottimizzazione dinamica, multimodale, multi-obiettivo e vincolata; indagini sulla robotica evolutiva, l'hardware evolutivo e l'elettronica evolutiva; applicazioni biologiche e bioinformatica; applicazioni nell'estrazione di dati, nei giochi, nell'arte e nella musica. I problemi aperti e le principali sfide includono la progettazione di algoritmi evolutivi ad alte prestazioni, l'incorporazione della conoscenza di dominio negli algoritmi, lo sviluppo di sistemi evolutivi online in cui l'adattamento e l'evoluzione avvengono contemporaneamente.

Esempio: Progettazione di aeromobili
Gli algoritmi evolutivi sono spesso impiegati nella progettazione di aeromobili per ottimizzare la forma e le caratteristiche di un velivolo. In questo contesto, l'algoritmo evolutivo può essere utilizzato per cercare la configurazione ottimale dell'aeromobile, considerando una serie di parametri come la forma delle ali, la disposizione dei motori, la distribuzione del carico e altri fattori critici.
Il funzionamento tipico di questo processo è il seguente:

1. **Generazione casuale di configurazioni iniziali**. Vengono generate una serie di configurazioni iniziali casuali degli aeromobili. Queste configurazioni includono vari parametri che definiscono la forma e le dimensioni delle parti dell'aereo.

2. **Valutazione delle prestazioni**. Ciascuna configurazione viene valutata in base a criteri di prestazione specifici come l'efficienza aerodinamica, la stabilità, la resistenza e altri fattori.

3. **Selezione e riproduzione**. Le configurazioni migliori, ossia quelle che si avvicinano di più agli obiettivi di prestazione desiderati, vengono selezionate per sopravvivere e riprodursi.

4. **Mutazione**. Le configurazioni selezionate subiscono piccole modifiche casuali, analoghe alle mutazioni biologiche, per creare nuove configurazioni.

5. **Iterazione**. Il processo di valutazione, selezione e riproduzione viene ripetuto per diverse generazioni di configurazioni. Man mano che le generazioni passano, ci si aspetta che le configurazioni migliori emergano e convergano verso una soluzione ottimale.

Questo processo iterativo di evoluzione delle configurazioni viene eseguito automaticamente dall'algoritmo e, nel corso delle iterazioni, l'aeromobile può essere ottimizzato per soddisfare criteri specifici come il massimo carico utile, il consumo di carburante ridotto o la stabilità in diverse condizioni atmosferiche.

1.1.3 Intelligenza swarm

L'intelligenza di sciame, o *swarm intelligence*, è un approccio computazionale che si ispira al comportamento collettivo e alla cooperazione osservati negli esseri viventi che formano gruppi o sciami. Questo concetto prende spunto da fenomeni naturali come il comportamento degli uccelli migratori, delle formiche o dei pesci in un banco.
Gli algoritmi basati sull'intelligenza di sciame cercano di emulare questi comportamenti collettivi attraverso la simulazione computazionale, al fine di risolvere problemi complessi di ottimizzazione, ricerca e decisione. Esistono svariate tipologie di algoritmi di questo tipo, ma l'obiettivo comune è utilizzare la cooperazione decentralizzata tra individui semplici per raggiungere obiettivi complessi. Questo approccio è stato applicato con successo in vari settori, tra cui la robotica, l'ottimizzazione, la gestione del traffico e l'analisi dei dati, dove la complessità dei problemi può essere affrontata in modo più efficiente e innovativo attraverso l'interazione collettiva.
Un'analisi più approfondita verrà trattata nel capitolo successivo.

1.1.4 Sistemi immunitari artificiali

Con *Sistemi Immunitari Artificiali*, o *SIA*, si intendono tutti quei sistemi adattivi ispirati al funzionamento dei sistemi immunitari biologici.

Il sistema immunitario è un complesso sistema biologico presente nel corpo di tutti gli esseri viventi, compresi gli esseri umani. La sua funzione principale è quella di difendere l'organismo da agenti patogeni come batteri, virus, funghi e altri microorganismi. Il sistema immunitario può essere diviso in sistema immunitario innato e sistema immunitario adattativo, composti da diverse categorie di cellule, molecole e organi che lavorano insieme per proteggere l'organismo. Il sistema immunitario innato costituisce il primo livello di difesa e agisce in modo rapido e generico. Il sistema immunitario adattativo, invece, è specifico per ogni agente patogeno e presenta la capacità di "imparare", ovvero estrarre informazioni, dalle esperienze passate.

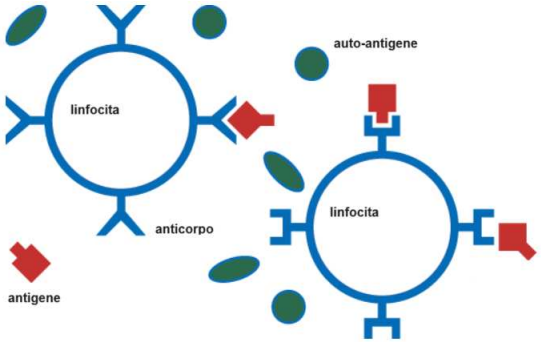

Figura 1.2: Comportamento del sistema immunitario umano

La maggior parte degli algoritmi SIA si concentra sul sistema immunitario adattivo. L'obiettivo di tali sistemi è quello di creare modelli computazionali che possano emulare alcune delle caratteristiche dei sistemi immunitari biologici come la capacità di riconoscere pattern, adattarsi a nuove minacce e rispondere in modo efficace a situazioni anomale. Questi modelli possono essere utilizzati in molte applicazioni differenti, tra cui la rilevazione di intrusioni in sistemi di sicurezza informatica e la diagnosi medica. In particolare, un algoritmo di questo tipo è composto dai seguenti elementi base:

- Una rappresentazione per i componenti del sistema, solitamente una stringa di attributi a valori interi, reali o binari.
- Un insieme di meccanismi per valutare l'interazione degli individui con l'ambiente e tra loro. Per le stringhe binarie, ad esempio, la misura di affinità più comune è la distanza di Hamming.
- Un insieme di procedure di adattamento che governano la dinamica del sistema.

Esempio: Sicurezza informatica
I passi utilizzati da un algoritmo SIA sono i seguenti:

1. **Modello di rilevamento delle intrusioni**. Si crea un modello di rilevamento delle intrusioni basato su SIA che rappresenta il comportamento normale del sistema. Questo modello può includere informazioni sulle attività di rete, i pattern di utilizzo delle risorse e altri indicatori di comportamento del sistema.

2. **Rilevamento delle anomalie**. Il sistema monitora costantemente le attività del sistema o della rete e confronta i dati osservati con il modello di comportamento normale. Qualsiasi deviazione significativa dal comportamento normale viene considerata un'anomalia e viene lanciato un allarme.

3. **Adattamento dinamico**. Il sistema di rilevamento delle intrusioni è in grado di adattarsi dinamicamente ai cambiamenti del sistema. Questo adattamento può comportare l'aggiornamento del modello di comportamento normale o la modifica delle regole di rilevamento delle anomalie.

4. **Risposta agli incidenti**. Quando viene rilevata un'anomalia o un potenziale attacco, il sistema può intraprendere misure di risposta, come l'allarme all'amministratore di sistema o l'isolamento della parte del sistema compromessa per impedire ulteriori danni.

5. **Apprendimento automatico**. I SIA possono utilizzare tecniche di apprendimento automatico per migliorare la loro capacità di rilevare nuove minacce e adattarsi a situazioni in evoluzione. Ad esempio, possono utilizzare algoritmi di clustering per identificare pattern di comportamento anomali.

Poiché i SIA possono adattarsi a minacce sconosciute e comportamenti inusuali, sono particolarmente utili nella difesa contro le minacce emergenti e nel rilevamento di attività malevole che potrebbe sfuggire a rilevamenti basati su regole statiche.

1.2 Sintesi dei fenomeni naturali al computer

L'informatica ispirata dalla natura ha principalmente l'obiettivo di affrontare problemi complessi. In parallelo, un secondo ambito dell'informatica naturale offre nuovi mezzi per creare e indagare fenomeni naturali. Questi strumenti possono essere impiegati per testare teorie biologiche che altrimenti non sarebbero verificabili tramite le tecniche sperimentali e analitiche tradizionali. Inoltre, c'è una relazione complementare tra la teoria biologica e i processi sintetici di simulazione ed emulazione della natura sui computer. Le analisi teoriche suggeriscono come attuare la sintesi, mentre l'applicazione pratica della teoria nella sintesi può fungere da prova per la stessa teoria.

Sostanzialmente, ci sono due approcci principali alla simulazione e all'emulazione della natura nei computer. Uno coinvolge l'utilizzo di strumenti per esaminare la geometria frattale che caratterizza molti aspetti della natura; l'altro impiega tecniche di vita artificiale, che mirano a creare sistemi computazionali in grado di emulare i processi biologici.

1.2.1 Geometria frattale

Una delle principali innovazioni nella sintesi di modelli e strutture naturali è il riconoscimento che la natura è frattale. Un frattale è una figura geometrica che si ripete all'infinito uguale a sé stessa, su scala sempre più piccola. Ciò significa che una parte qualsiasi del frattale riproduce, in piccolo, la figura nella sua totalità e in tutti i suoi dettagli. La natura fornisce molti esempi di frattali, ad esempio felci, linee costiere, broccoli e polmoni.

Benoit Mandelbrot coniò il termine "frattale" per identificare una famiglia di forme che descrivono modelli irregolari e frammentati nella natura, differenziandoli così dalle forme geometriche pure della geometria euclidea. La parola "frattale" deriva dall'aggettivo latino *fractus*, il cui verbo corrispondente "frangere" significa "rompere".
Nella geometria frattale la dimensione frattale è un concetto essenziale poiché fornisce un'indicazione di quanto lo spazio venga occupato dal frattale all'interno di un piano. A differenza delle dimensioni euclidee (come la dimensione 1 di una linea o la dimensione 2 di un piano), le dimensioni frattali possono essere frazionarie. Questo riflette la complessità intrinseca dei frattali, i quali non possono essere accuratamente descritti da un unico numero intero. [2]

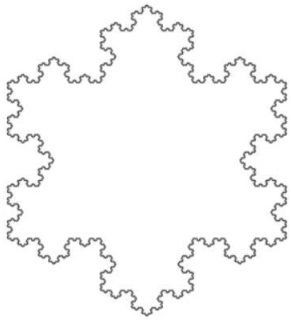

Figura 1.3: Esempio di geometria frattale: la curva di von Koch o fiocco di neve

Tra gli algoritmi che possono generare strutture frattali troviamo:

- Automi cellulari, modelli computazionali introdotti da John Von Neumann alla fine degli anni '40 come quadro per lo studio dell'autoriproduzione. Essi sono costituiti da una griglia di celle, ciascuna delle quali può trovarsi in uno stato particolare. Queste cellule evolvono nel tempo seguendo regole predefinite che dipendono dai loro stati e dagli stati delle celle circostanti. Gli automi cellulari sono stati ampiamente studiati per simulare fenomeni complessi come la propagazione delle onde, la diffusione di sostanze chimiche e il comportamento dei fluidi.
  Formalmente [3], un automa cellulare C può essere definito come una 5-tupla: $C = <\text{S}, s_0, G, d, f>$ in cui S è un insieme finito di stati, $s_0 \in S$ sono gli stati iniziali dell'automa cellulare, G è il vicinato cellulare, $d \in Z^+$ è la dimensione di C e f è la regola di interazione cellulare locale, anche chiamata *funzione di transizione*.
  L'automa cellulare più famoso è *Game of Life*, sviluppato dal matematico britannico John Conway nel 1970, spesso utilizzato come esempio introduttivo nell'ambito degli automi cellulari e della teoria del caos. Nel Game of Life, l'universo è rappresentato come una griglia bidimensionale di celle quadrate. Ogni cella può essere in uno di due stati: viva o morta. L'evoluzione del sistema avviene in base a semplici regole:
  - Una cella viva sopravvive alla generazione successiva se ha esattamente 2 o 3 vicini vivi (cellule adiacenti).
  - Una cella morta diventa viva nella generazione successiva se ha esattamente 3 vicini vivi.
  Ciò è interessante perché dimostra come da regole molto semplici possano emergere comportamenti sorprendentemente complessi.

- Sistemi di Lindenmayer, formalismo matematico utilizzato per descrivere la crescita di strutture complesse, come ad esempio le strutture delle piante. Gli *L-systems* [4] sono costituiti da un insieme di regole di produzione che specificano come una stringa di simboli deve essere riscritta iterativamente per creare modelli strutturali. Questo formalismo è spesso utilizzato per generare rappresentazioni visive di modelli di crescita e strutture naturali.
  In particolare, una stringa, o parola, OL-system è definita come la tripla ordinata $\text{G} = <\text{V}, \omega, \text{P}>$ dove V è l'alfabeto del sistema, $\omega \in V^+$ è una parola non vuota chiamata assioma e $\text{P} \subset V \times V^*$ è un insieme finito di produzioni.

  Esempio: albero frattale in codice binario
  - Variabili: 0, 1
  - Costanti: [,]
  - Assioma: 0
  - Regole di produzione: $(1 \rightarrow 11)$, $(0 \rightarrow 1\ [0]\ 0)$
  La struttura viene creata seguendo un processo ricorsivo che inizia con un elemento di base chiamato assioma. Questo processo avanza tramite regole di produzione che coinvolgono variabili. Ogni elemento nella sequenza di input viene confrontato con un insieme di regole predefinite. Questo confronto determina quale carattere o sequenza deve sostituire l'elemento nella sequenza di output.

Applicando ciò all'assioma '0', otteniamo:

- Assioma: 0
- Prima ricorsione: 1[0]0
- Seconda ricorsione: 11[1[0]0]1[0]0
- Terza ricorsione: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

Possiamo notare che questa stringa cresce rapidamente in dimensioni e complessità.
L'interpretazione geometrica di queste parole può essere utilizzata per generare immagini schematiche attraverso il linguaggio di *grafica a tartaruga*. Nella grafica a tartaruga, il tracciamento viene eseguito da un agente intelligente che segue determinati comandi. L'idea di base di interpretazione è la seguente: lo stato della tartaruga è definito come una tripletta <x, y, α>, dove le coordinate cartesiane (x,y) rappresentano la posizione della tartaruga e l'angolo α, chiamato direzione, è interpretato come la direzione verso cui la tartaruga è rivolta.

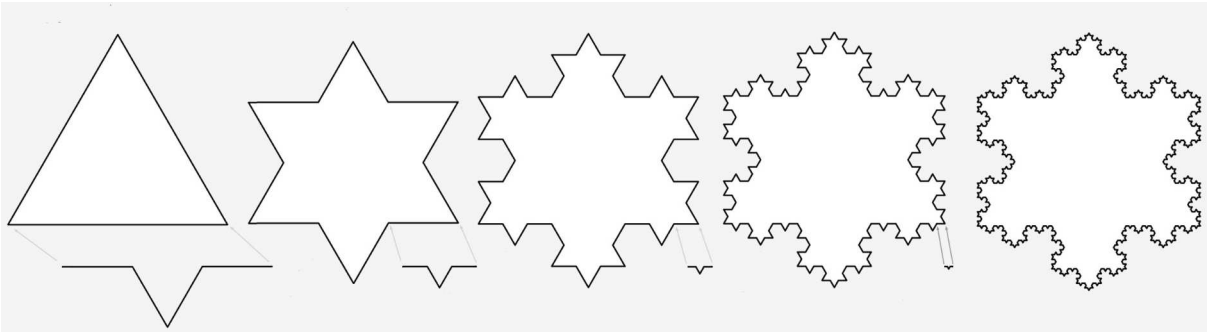

Figura 1.4: Immagine creata da un *L-system*

1.2.2 Vita Artificiale

Nell'ambito del Natural Computing, il concetto di Vita Artificiale, o *ALife*, si riferisce alla creazione e allo studio di sistemi artificiali che emulano, replicano o simulano i processi biologici e le caratteristiche della vita. Questi sistemi sono spesso basati su algoritmi ispirati alla biologia e mirano a creare forme di vita sintetiche o simulazioni di processi vitali.
Lo studio dell'ALife si basa su un importante concetto chiave: la concezione della vita come processo dinamico con determinate caratteristiche universali che non dipendono dalla materia. Pertanto, la vita è considerata una proprietà emergente dell'organizzazione della materia, anzichè una proprietà della materia stessa.
Tra le principali applicazioni di ALife vi sono:

- il linguaggio di programmazione **StarLogo** [5], sviluppato da Mitchel Resnick nel 2001, utilizzato per costruire e riflettere sui sistemi auto-organizzati. StarLogo è stato progettato come un linguaggio fortemente parallelo con un ambiente in grado di ospitare contemporaneamente un gran numero di agenti chiamati tartarughe, ognuno dei quali è capace di percepire l'ambiente circostante e interagire con esso. L'ambiente in cui vengono posizionate le tartarughe, chiamato mondo, è attivo e diviso in sezioni quadrate chiamate *patch*. Ciò significa che le patch possono memorizzare informazioni sulle attività delle tartarughe ed eseguire istruzioni proprio come fanno le tartarughe stesse. In altre parole, l'ambiente è considerato una componente "viva" in egual misura rispetto agli agenti che lo popolano.

- i ***boids***, un modello di simulazione sviluppato da Craig Reynolds nel 1986 per studiare il comportamento di animali che si muovono collettivamente. Il termine "boid", utilizzato per identificare il singolo agente virtuale, è un *portmanteau* delle parole *bird*, uccello, e *oid*, che indica un oggetto simile.
  In particolare, Reynolds ha dimostrato che il comportamento apparentemente intenzionale e centralizzato di un branco di uccelli può essere descritto da un piccolo insieme di regole che governano solo il comportamento degli agenti individuali, agendo unicamente sulla base della loro percezione locale dell'ambiente. Il modello di stormo consiste in tre semplici comportamenti: evitamento delle collisioni e separazione; allineamento della velocità e dell'orientamento; coesione dello stormo, ovvero cercare di rimanere in prossimità della posizione media dei compagni di stormo vicini. Il comportamento globale che emerge è il risultato di interazioni tra singoli elementi che seguono regole locali semplici.

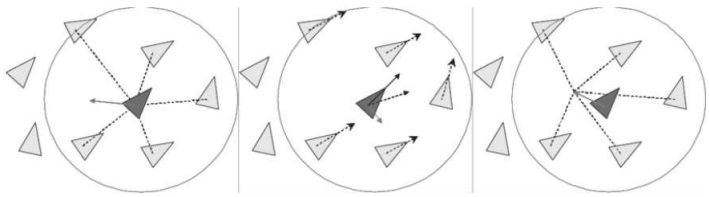

Figura 1.5: Comportamento di un *boid*

1.3 Computazione con nuovi materiali naturali

La computazione con nuovi materiali naturali rappresenta un campo di ricerca innovativo che si concentra sull'ideazione di nuovi metodi computazionali basati su materiali diversi dal tradizionale silicio. L'attuale corsa verso la miniaturizzazione dei componenti fondamentali dell'elaborazione dati sta gradualmente conducendo a una fase in cui le porte logiche diventeranno così minuscole da essere costituite da singoli atomi. In questo contesto, l'approccio della computazione con materiali naturali emerge come un potenziale catalizzatore di un significativo cambiamento nel panorama della tecnologia informatica contemporanea.

L'impulso principale alla base di questo ambito di ricerca risiede nella crescente necessità di individuare alternative per sostenere il processo computazionale. I ricercatori stanno attualmente dirigendo la loro attenzione verso la progettazione di nuovi tipi di computer, basati su elementi molecolari come il DNA e l'RNA, nonché sulla teoria quantistica. Tuttavia, è importante notare che la computazione con nuovi materiali naturali è ancora *in fieri* e presenta sfide significative legate alla stabilità, alla riproducibilità e alla scalabilità di tali sistemi.

1.3.1 Computazione a DNA

Il calcolo basato su DNA, noto anche come *DNA computing*, è un paradigma computazionale introdotto da Leonard Adleman nel 1994 che sfrutta il DNA (acido deossiribonucleico) come substrato per eseguire operazioni di calcolo. [6]
Il DNA computing si differenzia in modo sostanziale dalla computazione tradizionale basata su silicio poiché si avvale di molecole di DNA come elementi di calcolo. Queste molecole possono essere manipolate in modo da rappresentare input, output e processi intermedi di calcolo. Tale approccio si è dimostrato particolarmente adatto per risolvere alcuni tipi di problemi complessi, come il Problema del Commesso Viaggiatore o alcune istanze del problema del soddisfacimento delle clausole booleane.

Tutte le tecniche di computazione del DNA applicano un insieme specifico di operazioni biologiche ad un agglomerato di filamenti. Queste operazioni sono comunemente utilizzate dai biologi molecolari e le più importanti, nel contesto della computazione del DNA, sono:
- **Denaturazione**: separa i filamenti di DNA.
- ***Annealing***: fonde i filamenti di DNA.
- **Amplificazione**: moltiplica le molecole di DNA. Di solito, viene utilizzata una tecnica chiamata reazione a catena della polimerasi (PCR) per ottenere copie multiple di un sottoinsieme dei filamenti presenti.
- **Elettroforesi su gel**: misura le molecole di DNA e le separa in base alla lunghezza.
- **Filtraggio**: separa o estrae molecole specifiche.

Esempio: Risoluzione del Problema del Cammino Hamiltoniano attraverso il DNA computing.

Era il 1994 quando Adleman ha presentato il primo esperimento riuscito attraverso l'utilizzo di molecole di DNA e le relative tecniche di manipolazione per il calcolo. In particolare, Adleman ha affrontato con successo una modesta istanza del Problema del Cammino Hamiltoniano (*HPP*) all'interno di un grafo orientato. Tutto ciò è stato realizzato impiegando esclusivamente processi biochimici.
Un grafo orientato G con vertici designati v_1 e v_n è detto avere un cammino hamiltoniano se e solo se esiste una sequenza di archi diretti compatibili che inizia in v_1, termina in v_n e passa attraverso ogni vertice esattamente una volta.

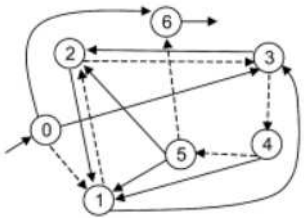

Figura 1.6: Problema del Cammino Hamiltoniano risolto da Adleman

La Fig. 1.5 illustra l'istanza del Problema del Cammino Hamiltoniano utilizzata da Adleman nella sua pionieristica implementazione del calcolo mediante DNA.
Per risolvere questo problema, Adleman ha utilizzato l'algoritmo deterministico seguente:
- Passo 1: generare percorsi casuali attraverso il grafo.
- Passo 2: mantenere solo quei percorsi che iniziano con v_1 e terminano con v_n.
- Passo 3: se il grafo ha n vertici, mantenere solo quei percorsi che attraversano esattamente n vertici.
- Passo 4: mantenere solo quei percorsi che attraversano tutti i vertici del grafo almeno una volta.
- Passo 5: se rimane almeno un percorso, il cammino è risolto, altrimenti no.

Per collegare gli archi e formare percorsi, è stato necessario sintetizzare oligonucleotidi $\bar{O}_i$ complementari a quelli che rappresentano gli archi. Per implementare il Passo 2, il prodotto del Passo 1 è stato amplificato mediante la reazione a catena della polimerasi (PCR) utilizzando come primer $\bar{O}_0$ e $\bar{O}_6$. Pertanto, solo le molecole che codificano percorsi che iniziano con il vertice 0 e terminano con il vertice 6 sono state amplificate. Successivamente, è stata eseguita un'operazione di filtraggio per separare i filamenti che iniziano con il vertice 0 e terminano con il vertice 6. In seguito, le molecole di DNA sono state separate in base alla loro lunghezza mediante elettroforesi su gel. Sono stati utilizzati cicli ripetuti di PCR ed elettroforesi per purificare ulteriormente il prodotto. Alla fine di questo passaggio, si ottiene un insieme di molecole che iniziano con 0, terminano con 6 e passano attraverso tutti vertici. Per implementare il Passo 4 dell'algoritmo, il DNA a singolo filamento è stato sottoposto a una prova con oligonucleotidi complementari attaccati a perline magnetiche. Infine, per ottenere la risposta al Passo 5, è ancora necessario amplificare il prodotto del Passo 4 mediante PCR e analizzarlo mediante elettroforesi su gel.

1.3.2 Computazione quantistica

Nel 1982 Richard Feynman, celebre fisico teorico e vincitore del premio Nobel per la fisica, tenne una conferenza presso il MIT in cui discusse l'idea di utilizzare i principi della meccanica quantistica per simulare sistemi fisici complessi. In seguito, nel 1985, pubblicò un articolo intitolato *Simulating Physics With Computers* [7], nel quale espose l'idea della creazione di un computer quantistico. Feynman intuì che la simulazione accurata di fenomeni quantistici su computer classici sarebbe stata estremamente difficile in termini di risorse a causa delle proprietà complesse della meccanica quantistica. Pertanto, propose l'idea di un computer basato su principi quantistici, il quale potrebbe essere in grado di risolvere problemi legati a tali fenomeni in modo più efficiente. Questo contributo di Feynman ha svolto un ruolo importante nell'ispirare il concetto di computazione quantistica e ha dato inizio alla ricerca e al successivo sviluppo di computer quantistici.

Il calcolo quantistico, o *quantum computing*, è un campo di ricerca e sviluppo che si concentra sull'utilizzo dei principi della meccanica quantistica per eseguire calcoli e risolvere problemi complessi in modo molto più efficiente rispetto ai computer classici. A differenza dei computer classici, che utilizzano bit per rappresentare dati, i computer quantistici utilizzano qubit, *quantum bits*, che possono rappresentare sia 0 che 1 contemporaneamente grazie a un fenomeno chiamato sovrapposizione quantistica. Questo permette a tali computer di svolgere più calcoli in parallelo, aumentando enormemente la loro potenziale capacità di risolvere problemi complessi.

Un altro concetto chiave della meccanica quantistica è l'*entanglement*, o intreccio quantistico, che permette ai qubit di essere strettamente correlati in modo che lo stato di uno influenzi istantaneamente lo stato dell'altro, indipendentemente dalla distanza che li separa. Questo settore è destinato a giocare un ruolo sempre più significativo nell'evoluzione delle tecnologie informatiche, aprendo nuove prospettive nella risoluzione di problemi che erano una volta considerati insormontabili per i computer tradizionali. Il quantum computing ha il potenziale per rivoluzionare numerosi ambiti come la crittografia, la modellazione molecolare, l'ottimizzazione complessa e la simulazione di sistemi fisici. Tuttavia, è importante notare che tale approccio è ancora in una fase di sviluppo e dunque molte sfide tecniche devono essere superate prima che possa diventare una tecnologia ampiamente accessibile e pratica.

Ad oggi, *Osprey* [8] rappresenta un importante passo avanti nel campo del quantum computing ed è il risultato dell'impegno di IBM nel promuovere l'innovazione in questo settore. Con i suoi 433 qubit, Osprey è attualmente il processore quantistico con il maggior numero di qubit mai realizzato. Sono stati superati, a tal proposito, il suo predecessore *Eagle*, anch'esso sviluppato da IBM, e persino *Sycamore*, il computer quantistico di Google che nel 2019 aveva segnato il traguardo della cosiddetta supremazia quantistica.

Capitolo 2

Intelligenza swarm e algoritmi delle colonie di formiche

2.1 Intelligenza di sciame

Il termine "intelligenza di sciame", o *swarm intelligence* [9], è stato coniato verso la fine degli anni '80 per descrivere sistemi robotici cellulari in cui un gruppo di agenti semplici agisce all'interno di un ambiente seguendo regole locali. Questa "intelligenza di branco" emerge quando una raccolta di agenti non particolarmente intelligenti e con capacità individuali limitate manifesta comportamenti collettivamente intelligenti. Questo concetto di intelligenza collettiva può essere suddiviso in due principali linee di ricerca: una basata sugli insetti sociali e l'altra sulla capacità delle società umane di elaborare conoscenza.

Sebbene i due approcci siano piuttosto diversi per sequenza di passi e fonti di ispirazione, essi presentano alcune caratteristiche comuni. In generale, entrambi si basano su una popolazione (colonia o branco) di individui (insetti sociali o particelle) capaci di interagire (direttamente o indirettamente) con l'ambiente e tra loro.
La loro flessibilità nel gestire problemi complessi e dinamici li rende una scelta interessante in molteplici settori, dalla scienza dei dati all'ingegneria, dalla biologia computazionale all'ottimizzazione industriale. Alcune delle applicazioni principali degli algoritmi di swarm intelligence includono:
- **Ottimizzazione globale**, gli algoritmi di swarm intelligence sono utilizzati per risolvere problemi in cui è necessario trovare il miglior insieme di soluzioni possibili in uno spazio di ricerca molto vasto. Questo può includere problemi di ottimizzazione, come la ricerca di minimi o massimi di funzioni complesse, e problemi NP-completi, come *Knapsack problem* e *University Class Scheduling Problem* (UCSP) [10].
- **Pianificazione di percorsi e logistica**, approcci di swarm intelligence trovano applicazione, ad esempio, nella logistica di trasporto, nella robotica mobile o nel Problema del Commesso Viaggiatore.
- **Clustering e classificazione**, la swarm intelligence può essere applicata all'aggregazione di dati in gruppi omogenei (*clustering*) o alla classificazione di dati in diverse categorie. In particolare, sono utili in campi come il riconoscimento di pattern, l'analisi dei dati e l'apprendimento automatico.

2.2 Algoritmi di ottimizzazione delle colonie di formiche

Gli algoritmi di ottimizzazione delle colonie di formiche, o *Ant Colony Optimization* (ACO), sono una classe di algoritmi di ricerca e ottimizzazione ispirati dal comportamento delle formiche e dalla loro capacità di trovare percorsi ottimali tra una fonte di cibo e il formicaio.

Tali algoritmi sono stati progettati appositamente per risolvere problemi di ottimizzazione discreta. I principali problemi affrontati in letteratura con algoritmi ACO sono: Problema del Commesso Viaggiatore [11], routing di rete [12], colorazione di grafi, assegnazione di frequenza e ordinamento sequenziale.
Negli ultimi anni, la maggior parte dei lavori su algoritmi ACO si è concentrata sul miglioramento degli algoritmi di base: tra le varianti più di successo ci sono *Elitist Ant System*, *Rank-Based Ant System*, *Max-Min Ant System* e *Hypercube Framework*.

2.2.1 ACO: motivazione biologica

Stephen Goss e Jean-Louis Deneubourg [1], attraverso una serie di esperimenti svolti nel 1989, hanno studiato il comportamento di tracciamento di alcune specie di formiche, in particolare l'argentina *Iridomyrmex humilis*. Queste formiche, all'interno di colonie di laboratorio, dimostrano una capacità sorprendente nel determinare il percorso più breve tra una fonte di cibo e il nido, anche quando si confrontano con una scelta tra rami di diverse lunghezze.
Inizialmente, durante una fase di transizione, le formiche esploratrici possono compiere scelte casuali. Tuttavia, in poco tempo, la maggioranza delle formiche converge verso il percorso più breve. Questa tendenza è ancora più marcata quando la differenza di lunghezza tra i rami è maggiore.

La chiave per comprendere questa abilità risiede nell'uso della stigmergia [13], una particolare forma di comunicazione indiretta impiegata dagli insetti sociali per coordinare le loro attività mediante la modifica dell'ambiente in cui operano. In particolare, risiede nell'uso dei feromoni, sostanze chimiche comunicative che vengono rilasciate dalle formiche durante il loro movimento lungo il percorso. La quantità di feromone rilasciato è inversamente proporzionale alla lunghezza totale del percorso, dunque i percorsi più brevi presentano un maggiore accumulo di feromone. Dopo che tutte le formiche hanno esplorato le possibili soluzioni, i percorsi che rappresentano la scelta più efficiente avranno una concentrazione più elevata di feromone. Inoltre, l'evaporazione naturale del feromone nel tempo contribuisce a differenziare ulteriormente i percorsi.

Questo meccanismo di comportamento collettivo e auto-organizzato delle formiche offre un affascinante esempio di come un sistema biologico possa risolvere problemi complessi attraverso interazioni semplici. Esso infatti si basa su semplici retroazioni positive (la deposizione di feromone attira altre formiche che rafforzeranno il percorso) e negative (la dissipazione del feromone a causa dell'evaporazione impedisce al sistema di bloccarsi). [14]

2.2.2 ACO: l'esperimento del doppio ponte

Il primo esperimento è stato progettato e condotto da Deneubourg e colleghi, i quali hanno utilizzato un doppio ponte collegante un nido di formiche della specie argentina *I. humilis* e una fonte di cibo. Hanno condotto esperimenti variando il rapporto $r = \frac{l_l}{l_s}$ tra la lunghezza dei due rami del doppio ponte, dove l_l era la lunghezza del ramo più lungo e l_s la lunghezza di quello più corto.

Nel primo esperimento, il ponte aveva due rami di lunghezza uguale ($r = 1$; vedi Figura 2.1a). All'inizio, le formiche sono state lasciate libere di spostarsi tra il nido e la fonte di cibo ed è stata osservata la percentuale di formiche che sceglieva uno dei due rami nel corso del tempo. Il risultato è stato che (vedi Figura 2.2a), sebbene nella fase iniziale si verificassero scelte casuali, alla fine tutte le formiche hanno utilizzato lo stesso ramo. Questo risultato può essere spiegato nel seguente modo: quando inizia una prova, non c'è feromone nei due rami, pertanto le formiche non hanno preferenze e selezionano con la stessa probabilità uno qualsiasi dei rami. Tuttavia, a causa delle fluttuazioni casuali, alcune formiche sceglieranno un ramo rispetto all'altro. Poiché le formiche depositano feromone mentre camminano, un maggior numero di formiche in un ramo comporta una maggiore quantità di feromone in quel ramo; questa maggiore quantità di feromone a sua volta stimola più formiche a scegliere nuovamente quel ramo, e così via fino a quando infine le formiche convergono verso un unico percorso. Questo processo di feedback autocatalitico è, infatti, un esempio di comportamento auto-organizzante delle formiche: un modello macroscopico (corrispondente alla convergenza verso un ramo) emerge da processi e interazioni che avvengono a livello microscopico (corrispondente alle interazioni locali tra gli individui della colonia).

Nel secondo esperimento, il rapporto di lunghezza tra i due rami è stato fissato a $r = 2$, in modo che il ramo lungo fosse il doppio del ramo corto (la Figura 2.1b mostra l'allestimento sperimentale). In questo caso, nella maggior parte delle prove, dopo un certo periodo tutte le formiche hanno scelto di utilizzare solo il ramo corto (vedi Figura 2.2b). Come nel primo esperimento, le formiche lasciano il nido per esplorare l'ambiente e arrivano a un punto decisionale in cui devono scegliere uno dei due rami. Poiché i due rami appaiono inizialmente identici alle formiche, scelgono casualmente. Pertanto, è possibile aspettarsi che in media la metà delle formiche scelga il ramo corto e l'altra metà il ramo lungo, anche se oscillazioni stocastiche possono occasionalmente favorire un ramo rispetto all'altro. Tuttavia, questo allestimento sperimentale presenta una differenza notevole rispetto al precedente: poiché un ramo è più corto dell'altro, le formiche che scelgono il ramo corto sono le prime a raggiungere il cibo e a iniziare il ritorno al nido. Ma poi, quando devono prendere una decisione tra il ramo corto e il ramo lungo, il livello più alto di feromone nel ramo corto influenzerà la loro decisione a suo favore. Pertanto, il feromone inizia ad accumularsi più velocemente nel ramo corto, che alla fine verrà utilizzato da tutte le formiche a causa del processo autocatalitico descritto in precedenza. Rispetto all'esperimento con i due rami di lunghezza uguale, l'influenza delle fluttuazioni casuali iniziali è molto ridotta e la stigmergia, l'autocatalisi e la lunghezza del percorso differenziale sono i principali meccanismi in gioco. È interessante notare che, anche quando il ramo lungo è il doppio del ramo corto, non tutte le formiche utilizzano il ramo corto, ma una piccola percentuale potrebbe prendere quello più lungo. Questo può essere interpretato come una forma di "esplorazione del percorso".

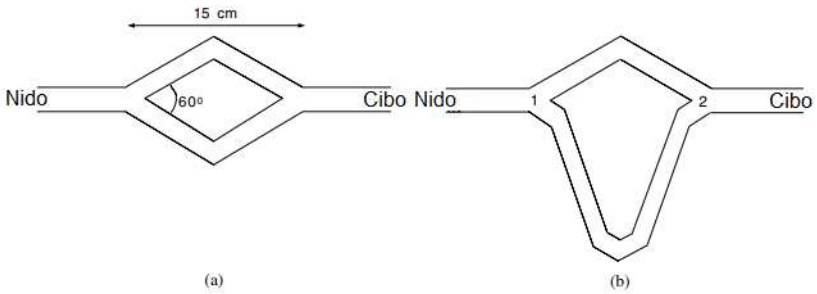

Figura 2.1: Esperimento del doppio ponte. (a) I rami hanno la stessa lunghezza. (b) I rami hanno lunghezza diversa.

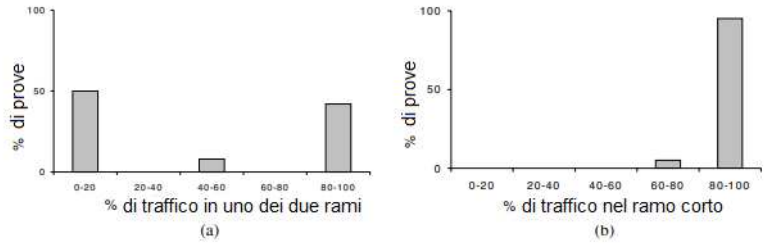

Figura 2.2: Risultati ottenuti dall'esperimento.

È interessante osservare cosa succede quando alla colonia di formiche viene offerta, dopo la convergenza, una nuova connessione più corta tra il nido e il cibo. Questa situazione è stata studiata in un esperimento aggiuntivo in cui inizialmente è stato offerto alla colonia solo il ramo lungo e dopo 30 minuti è stato aggiunto il ramo corto. In questo caso, il ramo corto veniva selezionato solo sporadicamente e la colonia rimaneva intrappolata sul ramo lungo. Ciò può essere spiegato dalla elevata concentrazione di feromone nel ramo lungo e dalla sua lenta evaporazione. Infatti, a causa della sua alta concentrazione di feromone, la grande maggioranza delle formiche sceglieva il ramo lungo, continuando a rafforzarlo, anche se ne compariva uno più corto. Ciò significa che il feromone si dissolve troppo lentamente per consentire alla colonia di formiche di "dimenticare" il percorso subottimale su cui sta convergendo, in modo che il nuovo e più breve possa essere scoperto.

2.2.3 ACO: algoritmo di base

Gli algoritmi ACO [15] forniscono soluzioni a problemi per l'ottimizzazione discreta ispirati all'osservazione del comportamento di ricerca del cibo delle colonie di formiche. Sebbene alcuni autori abbiano già sviluppato versioni di algoritmi ACO per l'ottimizzazione continua, non sono stati condotti molti studi in questa direzione e, quindi, il focus della discussione che sarà presentata qui si concentrerà sull'ottimizzazione discreta. Assumendo un grafo connesso $G = (V, E)$, l'algoritmo ACO semplice (S-ACO) può essere utilizzato per trovare una soluzione al problema del percorso più breve definito sul grafo G [1]. Una soluzione è un percorso sul grafo che collega un nodo di origine s a un nodo di destinazione d e la lunghezza del percorso è data dal numero di archi attraversati. Associata ad ogni arco (i, j) del grafo c'è una variabile τ_{ij} chiamata traccia di feromone artificiale, o semplicemente feromone. Ogni formica artificiale è in grado di "marcare" un arco con feromone e "sentire" (leggere) il feromone sulla traccia. Ogni formica attraversa un nodo per ogni iterazione t e, in ogni nodo, le informazioni locali sul livello di feromone, τ_{ij}, vengono utilizzate dalla formica in modo tale che possa decidere in modo probabilistico il prossimo nodo in cui spostarsi, secondo la seguente regola:

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)}{\sum_{j \in N_i} \tau_{ij}(t)} & se\ j \in N_i, \\ 0 & altrimenti, \end{cases}$$

dove $p_{ij}^k(t)$ è la probabilità che la formica k situata nel nodo i si sposti al nodo j, $\tau_{ij}(t)$ è il livello di feromone dell'arco (i, j), tutti presi all'iterazione t, e N_i è l'insieme dei vicini del nodo i. Durante il percorso di un arco (i, j), la formica deposita del feromone su di esso e il livello di feromone dell'arco (i, j) viene aggiornato secondo la seguente regola:

$$\tau_{ij}(t) \ \leftarrow \ (1 - \rho)\tau_{ij}(t) + \Delta\tau$$

dove t è il contatore di iterazione, τ è la quantità costante di feromone depositata dalla formica, $\Delta\tau$ è la variazione di feromone nell'arco (i, j) e $\rho \in (0, 1]$ è il tasso di decadimento del feromone.

Un algoritmo ACO alterna, per un numero massimo di iterazioni, l'applicazione di due procedure di base:

1. Una procedura di costruzione/modifica parallela delle soluzioni in cui un insieme di N formiche costruisce/modifica in parallelo N soluzioni per il problema in esame.

2. Una procedura di aggiornamento del feromone sulle tracce, in cui viene modificata la quantità di feromone sulle tracce degli archi del grafo del problema.

Il processo di costruzione o modifica di una soluzione avviene in modo probabilistico e la probabilità che un nuovo arco venga aggiunto alla soluzione in costruzione è una funzione della desiderabilità euristica dell'arco η e del feromone τ depositato dalle formiche precedenti. La desiderabilità euristica η esprime la probabilità che una formica si sposti su un dato arco. Ad esempio, nel caso si stia cercando il percorso minimo tra un numero di archi, la desiderabilità η di solito viene scelta come l'inverso della distanza tra una coppia di nodi. [16]

2.3 Ulteriori algoritmi di intelligenza di sciame: confronto tra applicazioni differenti

Questa sezione si concentrerà su un confronto tra tre differenti algoritmi di ottimizzazione ispirati al comportamento naturale, fornendo una prospettiva più ampia sui vari approcci basati sulla natura utilizzati per risolvere problemi complessi.

2.3.1 Algoritmo dello sciame di particelle

L'algoritmo dello sciame di particelle, o *Particle Swarm Optimization* (PSO), si propone di costruire una simulazione del comportamento sociale umano, cioè l'abilità delle comunità umane di elaborare conoscenze. Alla base del PSO vi è una popolazione di individui capaci di interagire con l'ambiente e tra di loro, in modo particolare con i propri vicini, creando così un terreno fertile per l'emergere di comportamenti collettivi a livello di intera popolazione.
Nel contesto dell'algoritmo PSO, gli individui coinvolti nella risoluzione di un problema specifico traggono insegnamento sia dalle loro esperienze individuali che dalle esperienze condivise con gli altri membri della popolazione. Questi individui si sottopongono a una valutazione personale, la quale è poi messa a confronto con il rendimento dei loro vicini, permettendo loro di emulare soltanto quei vicini che dimostrano performance superiori. Questa dinamica consente agli individui di effettuare valutazioni, comparazioni ed emulazioni in risposta a diverse situazioni fornite dall'ambiente.

Il PSO [17] è progettato principalmente per la ricerca di massimi all'interno dello spazio reale L-dimensionale $\mathbb{R}^L$. Le variabili coinvolte nella funzione da ottimizzare possono essere visualizzate come un vettore che corrisponde a un punto dello spazio di ricerca multidimensionale. Questa visione consente di seguire più individui all'interno di uno stesso sistema di coordinate, dove diversi individui si traducono in un insieme di punti.

In termini matematici, l'algoritmo dello sciame di particelle può essere implementato come segue. La posizione di una particella i è data da x_i, che è un vettore L-dimensionale in $\mathbb{R}^L$. La variazione nella posizione di una particella è data dalla sua velocità, anch'essa rappresentata tramite il vettore L-dimensionale v_i:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

L'algoritmo dello sciame di particelle campiona lo spazio di ricerca modificando la velocità di ciascuna particella. Bisogna capire ora come definire la velocità della particella in modo che si muova nelle direzioni appropriate e con la "dimensione del passo" adeguata nello spazio di ricerca. Inoltre, è necessario definire il vicinato di ciascuna particella.

L'ispirazione presa dalle scienze sociali e psicologiche suggerisce che gli individui dovrebbero essere influenzati dalla propria esperienza precedente e dall'esperienza dei loro vicini. Qui, il termine "vicinato" si riferisce alla somiglianza topologica in una determinata struttura della popolazione. La maggior parte delle implementazioni dello sciame di particelle utilizza due semplici principi sociometrici. Il primo, chiamato g_{best}, collega concettualmente tutti i membri della popolazione tra loro. L'effetto di ciò è che

ciascuna particella è influenzata dalle migliori prestazioni di qualsiasi membro dell'intera popolazione. Il secondo, chiamato l_{best}, crea un vicinato per ciascun individuo composto da se stesso e dai suoi k vicini più prossimi nella popolazione. Una particella si muoverà in una certa direzione in base alla sua posizione attuale $x_i(t)$, alla sua velocità $v_i(t)$, alla posizione migliore raggiunta finora dalla particella p_i stessa e alla migliore posizione trovata da qualsiasi membro del suo vicinato p_g:

$$x_i(t+1) = f(x_i(t), v_i(t+1), p_i, p_g).$$

L'influenza dei termini $v_i(t)$, p_i e p_g può essere riassunta da un cambiamento $v_i(t+1)$ da applicare all'iterazione $t+1$:

$$v_i(t+1) = v_i(t) + \phi_1 \otimes (p_i - x_i(t)) + \phi_2 \otimes (p_g - x_i(t)),$$

dove ϕ_1 e ϕ_2 rappresentano vettori casuali positivi composti da numeri estratti da distribuzioni uniformi con un limite superiore predefinito: $\phi_1 = U(0, AC_1)$ e $\phi_2 = U(0, AC_2)$. $U(0, AC)$ è un vettore composto da numeri casuali uniformemente distribuiti, mentre AC è chiamato costante di accelerazione e solitamente vale 2.05. Il simbolo $\otimes$ rappresenta la moltiplicazione vettoriale elemento per elemento.
Il secondo termine del lato destro dell'equazione è proporzionale alla differenza tra il miglior risultato precedente della particella e la sua posizione corrente, mentre l'ultimo termine del lato destro dell'equazione è proporzionale alla differenza tra il miglior risultato del vicinato e la posizione corrente della particella.

Al fine di limitare il cambiamento nella posizione di una particella, vengono definiti due valori v_{min} e v_{max} per il cambiamento v, garantendo così che le particelle oscillino sempre entro alcuni limiti predefiniti.

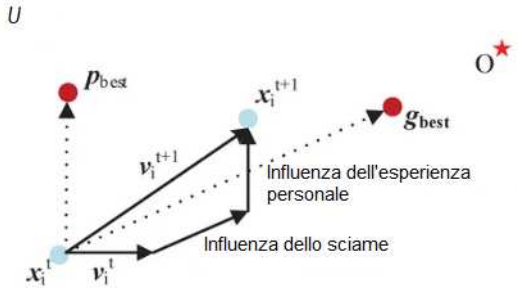

Figura 2.3: Movimento di una particella in un algoritmo PSO

In Figura 2.1, l'area U è lo spazio delle soluzioni di una funzione, mentre O rappresenta la soluzione ottima. $x_i(t)$ è la posizione iniziale della particella, $v_i(t)$ è la velocità della particella stessa. All'iterazione successiva, velocità ($v_i(t+1)$) e posizione ($x_i(t+1)$) saranno determinate da diversi aspetti, tra cui g_{best}, il miglior risultato globale ottenuto finora dalle particelle.

L'algoritmo PSO è uno strumento flessibile e potente che può essere applicato a una vasta gamma di problemi di ottimizzazione, compresi giochi, routing [18], robotica e *machine learning*.

2.3.2 Algoritmo della colonia di api

L'algoritmo della colonia di api artificiali, o *Artificial Bee Colony* (ABC), è un algoritmo di ottimizzazione ispirato al comportamento delle api nella ricerca di cibo. Questo algoritmo è stato proposto da Karaboga nel 2005 [19] e si basa sulla cooperazione e l'auto-organizzazione delle api reali. In particolare, la comunicazione delle informazioni avviene tramite la *waggle dance*: attraverso questa danza, l'ape indica sia la distanza (tramite la durata della danza) che la direzione di una risorsa (calcolando l'angolo che questa risorsa forma con la direzione del sole). Maggiori sono le informazioni che vengono condivise nell'alveare riguardo a una specifica risorsa, maggiori sono le probabilità che altre api vengano reclutate per sfruttarla.

Nell'ABC, la colonia di api artificiali contiene tre tipi di api: *api impiegate* associate a fonti di cibo specifiche, *api osservatrici* che guardano la danza delle api impiegate dall'interno dell'alveare in modo da scegliere una fonte di cibo e *api esploratrici* che cercano fonti di cibo casualmente. Sia le api osservatrici che le api esploratrici vengono chiamate anche "api disoccupate". Inizialmente, tutte le posizioni delle fonti di cibo vengono scoperte dalle api esploratrici. Successivamente, il nettare delle fonti di cibo viene sfruttato dalle api impiegate e dalle api osservatrici, fino a che la fonte di cibo non sarà esaurita. Quindi, l'ape impiegata che stava sfruttando la fonte di cibo esaurita diventa nuovamente un'ape esploratrice alla ricerca di ulteriori fonti di cibo. In altre parole, l'ape impiegata la cui fonte di cibo è stata esaurita diventa un'ape esploratrice.

Dal punto di vista computazionale, nell'ABC la posizione di una fonte di cibo rappresenta una possibile soluzione al problema e la quantità di nettare di tale fonte corrisponde alla qualità della soluzione associata. Il numero di api impiegate è uguale al numero di fonti di cibo (soluzioni) poiché ogni ape impiegata è associata a una e una sola fonte di cibo.

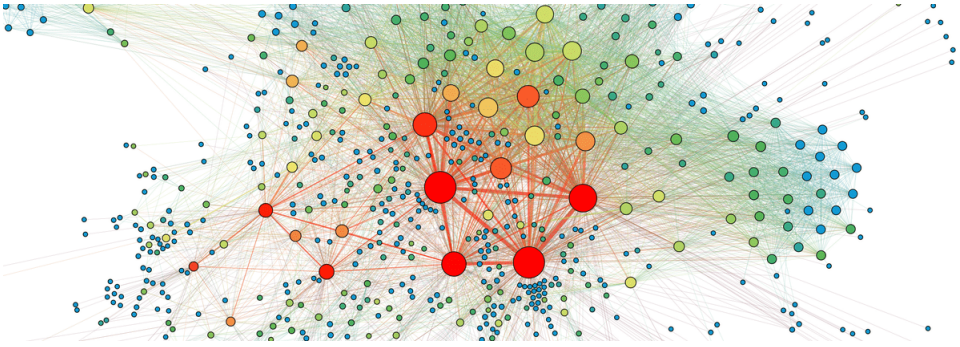

Figura 2.4: Algoritmo ABC per risolvere un problema di clustering

2.3.3 Confronto tra ACO, PS e ABC

Gli algoritmi basati sull'intelligenza di sciame presentati in questo capitolo sono approcci computazionali ispirati al comportamento collettivo degli insetti sociali. Questi algoritmi cercano di emulare i modelli di cooperazione e coordinazione presenti in natura per risolvere problemi complessi di ottimizzazione.
Tuttavia, ciascun algoritmo ha delle caratteristiche distintive che lo differenzia dagli altri in termini di meccanismi di ricerca, comunicazione e adattamento [20]. Di seguito, verranno confrontati gli algoritmi ACO, PS e ABC in termini di vari aspetti chiave dell'intelligenza di sciame.

1. **Meccanismo di coordinazione**
   - Ant Colony Optimization (ACO): la comunicazione avviene attraverso il rilascio di feromoni. Le formiche seguono i percorsi con feromoni più densi, aggiornando dinamicamente le loro scelte in base alle informazioni raccolte.

   - Particle Swarm (PS): le particelle si muovono nello spazio delle soluzioni cercando di migliorare le proprie prestazioni seguendo le migliori esperienze dei membri del loro vicinato. La comunicazione avviene attraverso l'aggiornamento delle velocità e delle posizioni delle particelle in base ai successi delle altre.

   - Artificial Bee Colony (ABC): le api comunicano principalmente attraverso la "waggle dance", condividendo informazioni sulle posizioni delle fonti di cibo.

2. **Adattamento e esplorazione**
   - ACO: l'aggiornamento dei feromoni guida le formiche verso soluzioni promettenti. La persistenza dei feromoni può essere bilanciata per influenzare la ricerca tra esplorazione e sfruttamento.

   - PS: l'equilibrio tra esplorazione e sfruttamento è ottenuto regolando le velocità delle particelle. Maggiore è l'approccio verso le migliori soluzioni globali, maggiore è l'esplorazione; viceversa, maggiori sono i contributi locali, maggiore è lo sfruttamento.

   - ABC: la "waggle dance" comunica l'informazione sulla redditività delle fonti di cibo. Le api osservatrici e le api esploratrici contribuiscono all'esplorazione continuativa e all'adattamento delle fonti di cibo.

3. **Adattabilità all'ambiente**
   - ACO: la regolazione delle strategie di deposito e evaporazione dei feromoni può influire sull'adattamento dell'algoritmo ai diversi scenari.

   - PS: l'adattamento dei parametri di aggiornamento influenza la capacità di adattamento a diversi spazi di ricerca.

   - ABC: l'abilità di scoprire nuove fonti di cibo e abbandonare quelle esaurite conferisce una certa flessibilità nell'affrontare cambiamenti ambientali.

In sintesi, mentre PS, ACO e ABC condividono l'obiettivo di utilizzare l'intelligenza collettiva per risolvere problemi complessi, ognuno ha un proprio approccio distintivo nel modo in cui le informazioni vengono condivise, la ricerca viene guidata e l'adattamento all'ambiente viene gestito. La scelta tra questi algoritmi dipenderà dal problema specifico da affrontare e dalle caratteristiche dell'ambiente di ottimizzazione.

Capitolo 3

Applicazione di algoritmi ACO al Problema del Commesso Viaggiatore

Il Problema del Commesso Viaggiatore è un problema ampiamente studiato in letteratura e da molto tempo ha attratto un considerevole sforzo di ricerca. Il *Traveling Salesman Problem*, o TSP, svolge un ruolo importante nella ricerca sulle colonie di formiche: il primo algoritmo ACO, chiamato *Ant System*, così come molti degli algoritmi ACO proposti successivamente, è stato inizialmente testato sul TSP [11].

Ci sono diverse ragioni per la scelta del TSP come problema per spiegare il funzionamento degli algoritmi ACO:

- E' un importante problema di ottimizzazione NP-*hard* che emerge in diverse applicazioni.
- E' un problema a cui gli algoritmi ACO possono essere facilmente applicati.
- E' facilmente comprensibile, in modo che il comportamento dell'algoritmo non sia oscurato da troppi tecnicismi.

Questo capitolo è dedicato a una spiegazione dettagliata dei principali membri della famiglia ACO attraverso esempi di applicazioni al TSP: gli algoritmi sono descritti in dettaglio e viene fornita una guida per la loro implementazione.

3.1 Il Problema del Commesso Viaggiatore

In modo intuitivo, il Problema del Commesso Viaggiatore rappresenta il dilemma di un venditore che, partendo dalla sua città natale, desidera trovare un percorso più breve che lo porti attraverso un dato insieme di città e poi torni a casa, visitando ogni città esattamente una volta.
In modo più formale, il TSP può essere rappresentato da un grafo completo $G = (N, A)$, dove N è l'insieme dei nodi che rappresentano le città e A è l'insieme degli archi. Anche se il grafo non è completo, è sempre possibile aggiungere archi per ottenere un nuovo grafo completo con esattamente le stesse soluzioni ottimali di G: ciò può essere ottenuto assegnando ai nuovi archi pesi sufficientemente grandi da garantire che non

verranno utilizzati in nessuna soluzione ottimale. Ad ogni arco (i, j) appartenente ad A è assegnato un valore d_{ij}, che rappresenta la distanza tra le città i e j, con $i, j \in N$. Nel caso generale del TSP asimmetrico, la distanza tra una coppia di nodi (i, j) dipende dalla direzione di attraversamento dell'arco, ovvero esiste almeno un arco (i, j) per il quale $d_{ij} \neq d_{ji}$. Nel TSP simmetrico, vale $d_{ij} = d_{ji}$ per tutti gli archi in A.

L'obiettivo del TSP è trovare un cammino hamiltoniano di lunghezza minima nel grafo, dove un circuito hamiltoniano è un percorso chiuso che visita ognuno degli n nodi di G esattamente una volta. Quindi, una soluzione ottimale per il TSP è una permutazione π degli indici dei nodi tale che la lunghezza $f(\pi)$ sia minima, dove $f(\pi)$ è data da:

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$$

Una caratteristica chiave del TSP è la sua complessità computazionale. È stato dimostrato che il TSP è un problema NP-*hard*, il che significa che non esiste un algoritmo efficiente (cioè, di complessità polinomiale della dimensione dell'istanza) per risolverlo in modo ottimale quando il numero di città diventa grande. Data la difficoltà di risolvere il TSP in modo esatto per grandi istanze, esistono diversi approcci per affrontare il problema:

- **Algoritmi esatti**: cercano di trovare la soluzione ottimale esaminando tutte le possibili permutazioni delle città. Tuttavia, la complessità di questo approccio cresce in modo esponenziale con il numero di città, rendendolo praticamente inapplicabile per grandi istanze.

- **Algoritmi euristici**: forniscono soluzioni approssimate al TSP in un tempo ragionevole. Alcuni esempi di algoritmi euristici includono l'algoritmo *Nearest Neighbor* e l'algoritmo del *Minimum Spanning Tree*.

- **Metaeuristica**: un metodo per la soluzione di una classe molto ampia di problemi computazionali che combina diverse procedure a loro volta euristiche, con l'obiettivo di ottenere una procedura più robusta ed efficiente. Alcuni esempi includono l'ottimizzazione per sciami di particelle (PSO) e l'ottimizzazione per colonie di formiche (ACO).

La Figura 3.1 mostra un esempio di TSP in cui il numero di città è uguale a 12.

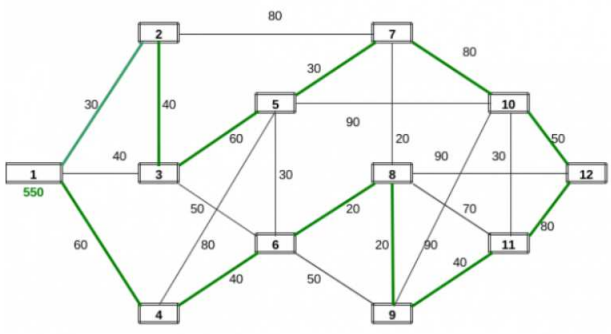

Figura 3.1: Istanza di TSP e relativa soluzione

3.2 Applicazione di algoritmi ACO al Problema del Commesso Viaggiatore

Il problema di trovare il percorso più breve dal nido alla fonte di cibo discusso nel Capitolo 2 è simile al Problema del Commesso Viaggiatore appena analizzato. Il commesso deve trovare il percorso più breve per visitare un determinato numero di città, ognuna esattamente una volta, minimizzando il costo del viaggio. Ispirandosi agli esperimenti di Goss, Dorigo e collaboratori hanno esteso il modello di approvviggionamento di cibo delle formiche per risolvere tale problema. [11]

Il grafo di costruzione $G = (C, L)$, in cui l'insieme L connette completamente i componenti C, è identico al grafo del problema, ovvero $C = N$ e $L = A$. L'insieme di stati del problema corrisponde all'insieme di tutti i possibili percorsi parziali e i vincoli Ω garantiscono che le formiche costruiscano solo percorsi opportuni che corrispondono a permutazioni degli indici delle città. Si noti che, essendo il grafo G completo, una qualsiasi permutazione costituisce una soluzione ammissibile, poiché il grafo di costruzione è un grafo completo.

In tutti gli algoritmi ACO disponibili per il TSP, le tracce di feromone sono associate agli archi e quindi τ_{ij} si riferisce alla desiderabilità di visitare la città j direttamente dopo la città i. L'informazione euristica è scelta come $h_{ij} = \frac{1}{d_{ij}}$, ovvero la desiderabilità euristica di passare dalla città i alla città j è inversamente proporzionale alla distanza tra le due città. Nel caso in cui $d_{ij} = 0$ per qualche arco (i, j), l'h_{ij} corrispondente viene impostato su un valore molto piccolo. Come verrà discusso in seguito, ai fini dell'implementazione, le tracce di feromone vengono raccolte in una matrice i cui elementi sono i vari τ_{ij}. Questo può essere fatto in modo analogo per l'informazione euristica.

I percorsi vengono costruiti applicando la seguente procedura a ciascuna formica:

1. Scegliere, secondo un certo criterio, una città di partenza in cui la formica è posizionata.

2. Utilizzare i valori di feromone e di informazione euristica per costruire in modo probabilistico un percorso aggiungendo iterativamente città che la formica non ha ancora visitato.

3. Ritornare alla città iniziale.

Dopo che tutte le formiche hanno completato il loro percorso, possono depositare feromone sui percorsi che hanno seguito.

Vedremo che, in alcuni casi, prima di aggiungere il feromone, i percorsi costruiti dalle formiche possono essere migliorati mediante l'applicazione di una procedura di ricerca locale. Questa descrizione ad alto livello si applica alla maggior parte degli algoritmi ACO pubblicati per il TSP, con una nota eccezione che è *Ant Colony System* (descritto nel Capitolo 3, sezione 3.3), in cui l'evaporazione del feromone è intercalata con la costruzione del percorso.

Lo pseudocodice di questo algoritmo è il seguente:

```
procedure AlgoritmoACO
    Inizializzazione parametri e feromoni
    while (condizione di risoluzione non soddisfatta) do
      CostruzioneSoluzioniFormiche
      RicercaLocale (opzionale)
      AggiornamentoFeromoni
    end
end
```

Dopo l'inizializzazione dei parametri e delle tracce di feromone, questi algoritmi ACO iterano attraverso un ciclo principale in cui prima vengono costruiti tutti i percorsi delle formiche, poi può avere luogo una fase opzionale in cui i percorsi delle formiche vengono migliorati mediante l'applicazione di un algoritmo di ricerca locale, e infine le tracce di feromone vengono aggiornate. Quest'ultimo passaggio coinvolge l'evaporazione del feromone e l'aggiornamento delle tracce da parte delle formiche per riflettere la loro esperienza di ricerca.

Come già menzionato, il primo algoritmo ACO, *Ant System* (AS), è stato introdotto utilizzando il TSP come esempio di applicazione. Ant System ha ottenuto risultati inizialmente incoraggianti, ma è comunque risultato inferiore nelle prestazioni rispetto agli algoritmi all'avanguardia per il TSP. L'importanza di tale algoritmo risiede quindi principalmente nell'ispirazione che ha fornito per una serie di estensioni che hanno migliorato significativamente le prestazioni e sono attualmente tra gli algoritmi ACO più efficaci. Infatti, la maggior parte di queste estensioni sono estensioni dirette di AS nel senso che mantengono la stessa procedura di costruzione delle soluzioni e la stessa procedura di evaporazione del feromone. Queste estensioni includono *Elitist Ant System*, *Rank-Based Ant System*, *Max-Min Ant System*.

Le principali differenze tra AS e queste estensioni riguardano il modo in cui viene eseguito l'aggiornamento del feromone, oltre ad alcuni dettagli aggiuntivi nella gestione delle tracce. Come ultima osservazione introduttiva, notiamo che non vengono presentati gli algoritmi ACO disponibili in ordine cronologico della loro prima pubblicazione, ma piuttosto nell'ordine della crescente complessità delle modifiche che introducono rispetto all'algoritmo originale.

3.3 Ant System e i suoi successori

In questa sezione vengono presentati gli Ant System e quegli algoritmi di Ant Colony Optimization che sono affini ad AS. Non viene considerato l'utilizzo della fase opzionale di ricerca locale.

3.3.1 Ant System

Inizialmente furono proposte tre diverse versioni di Ant System [16]. Queste erano chiamate *ant-density*, *ant-quantity* e *ant-cycle*. Mentre nelle versioni ant-density e ant-quantity le formiche aggiornavano direttamente il feromone dopo uno spostamento da una città a una città adiacente, nella versione ant-cycle l'aggiornamento del feromone

veniva effettuato solo dopo che tutte le formiche avevano costruito i percorsi e la quantità di feromone depositata da ciascuna formica veniva impostata come una funzione della qualità del percorso svolto. Oggi, quando si fa riferimento ad AS, si fa effettivamente riferimento ad ant-cycle, poiché le altre due varianti sono state abbandonate a causa delle loro prestazioni inferiori.

Le due fasi principali di tale algoritmo comprendono la costruzione della soluzione da parte delle formiche e l'aggiornamento del feromone. In AS, una buona euristica per inizializzare i sentieri di feromone è impostarli su un valore leggermente più alto rispetto alla quantità di feromone attesa depositata dalle formiche in una singola iterazione; una stima approssimativa di questo valore può essere ottenuta impostando $\forall(i,j)$, $\tau_{ij} = \tau_0 = \frac{m}{C^{nn}}$, dove m è il numero di formiche e C^{nn} è la lunghezza di un percorso generato dall'euristica del *Nearest Neighbor*. Il motivo di questa scelta è che se i valori iniziali del feromone τ_0 sono troppo bassi, la ricerca viene rapidamente influenzata dai primi itinerari generati dalle formiche, il che in generale porta all'esplorazione di un minor numero di zone dello spazio di ricerca. D'altra parte, se i valori iniziali del feromone sono troppo alti, molte iterazioni vengono perse nell'attesa che l'evaporazione del feromone ne riduca i valori, in modo che il feromone aggiunto dalle formiche possa iniziare a influenzare la ricerca.

Costruzione del percorso

In AS, m formiche artificiali costruiscono contemporaneamente un itinerario per il Problema del Commesso Viaggiatore. Inizialmente, le formiche vengono posizionate casualmente su città diverse. Ad ogni passo della costruzione, la formica k applica una regola di scelta d'azione probabilistica, chiamata *regola di proporzionale casuale*, per decidere quale città visitare successivamente. In particolare, la probabilità con cui la formica k, attualmente nella città i, sceglie di andare nella città j è determinata da una formula:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \cdot \eta_{il}^\beta}, \text{ se } j \in N_i^k,$$

in cui $\eta_{ij} = \frac{1}{d_{ij}}$ è un valore euristico disponibile a priori, mentre α e β sono due parametri che determinano l'influenza relativa della traccia di feromone e delle informazioni euristiche. N_i^k rappresenta il vicinato ammissibile della formica k quando si trova nella città i, cioè l'insieme di città che la formica k non ha ancora visitato (la probabilità di scegliere una città al di fuori di N_i^k è 0). Con questa regola probabilistica, la probabilità di scegliere un particolare arco (i, j) aumenta con il valore della traccia di feromone associata τ_{ij} e del valore delle informazioni euristiche h_{ij}. Il ruolo dei parametri α e β è il seguente. Se $\alpha = 0$ le città più vicine sono le più probabili da selezionare: ciò corrisponde a un classico algoritmo greedy stocastico. Se $\beta = 0$ è in atto solo l'amplificazione del feromone, ovvero viene utilizzato solo il feromone senza alcun bias euristico. Questo porta generalmente a risultati piuttosto scarsi e, in particolare, per valori di $\alpha > 1$, si verifica rapidamente l'emergere di una situazione di stallo, cioè una situazione in cui tutte le formiche costruiscono la stessa strada, che in generale è fortemente subottimale. Valori adatti a tali parametri sono stati verificati sperimentalmente e sono presentati in Tabella 3.1.

Tabella 3.1: Impostazione dei parametri per algoritmi ACO

Algoritmo ACO	α	β	ρ	m	τ_0
AS	1	2-5	0.5	n	m/C^{nn}
EAS	1	2-5	0.5	n	$(\epsilon + m)/\rho C^{nn}$
AS_{rank}	1	2-5	0.1	n	$0.5r(r-1)/\rho C^{nn}$
MMAS	1	2-5	0.02	n	$1/\rho C^{nn}$

Ogni formica k mantiene una memoria M^k che contiene un elenco ordinato delle città già visitate. Questa memoria viene utilizzata per definire il vicinato ammissibile N_i^k nella regola di costruzione data dall'equazione proposta in precedenza. Inoltre, la memoria M^k consente alla formica k di calcolare la lunghezza del percorso T^k che ha generato, oltre a ritracciare il percorso per depositare il feromone. Per quanto riguarda la costruzione della soluzione, ci sono due modi diversi di implementarla: costruzione parallela e costruzione sequenziale della soluzione. Nell'implementazione parallela, ad ogni passo di costruzione, tutte le formiche si spostano dalla loro città attuale alla successiva, mentre nell'implementazione sequenziale una formica costruisce un percorso completo prima che la successiva inizi a costruirne un altro. Entrambe le scelte per l'implementazione della costruzione del percorso sono equivalenti nel senso che non influenzano significativamente il comportamento dell'algoritmo.

Aggiornamento delle tracce di feromone

Dopo che tutte le formiche hanno costruito i loro itinerari, le tracce di feromone vengono aggiornate. Ciò avviene innanzitutto abbassando il valore del feromone su tutti gli archi di un fattore costante e successivamente aggiungendo feromone sugli archi che le formiche hanno attraversato nei loro viaggi. L'evaporazione del feromone è implementata dalla seguente formula:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \forall (i,j) \in L,$$

dove $0 < \rho \leq 1$ è il tasso di evaporazione del feromone. Il parametro ρ serve a evitare l'accumulo illimitato delle tracce di feromone e consente all'algoritmo di "dimenticare" decisioni precedenti errate. Infatti, se un arco non viene scelto dalle formiche, il suo valore di feromone associato diminuisce esponenzialmente nel numero di iterazioni.
Dopo l'evaporazione, tutte le formiche depositano feromone sugli archi che hanno attraversato nei loro viaggi, utilizzando la seguente formula:

$$\tau_{ij} \leftarrow \tau_{ij} + \textstyle\sum_{k=1}^{m} \Delta\tau_{ij}^k,\ \forall (i,j) \in L,$$

dove $\Delta\tau_{ij}^k$ è la quantità di feromone depositata dalla formica k sugli archi che ha visitato ed è definita come segue:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{C^k} & se\ l'arco\ (i,j) \in T^k, \\ 0 & altrimenti, \end{cases}$$

dove C^k è la lunghezza del percorso T^k costruito dalla formica k, calcolata come la somma delle lunghezze degli archi che appartengono a T^k.
In generale, gli archi utilizzati da molte formiche e che fanno parte di percorsi brevi ricevono più feromone e sono quindi scelti con maggiori probabilità dalle formiche nelle future iterazioni dell'algoritmo.

Come è stato accennato precedentemente, le prestazioni di AS tendono a diminuire drasticamente man mano che aumenta la dimensione dell'istanza del problema. Pertanto, una notevole quantità di ricerca sugli algoritmi ACO si è concentrata su come migliorare AS, sviluppando varianti in grado risolvere tale problematica.

3.3.2 Elitist Ant System

Un primo miglioramento dell'AS iniziale, chiamato "strategia elitaria per Ant System" (EAS), è stato introdotto da Dorigo nel 1992. L'EAS aggiunge un elemento di elitarismo: oltre a utilizzare il comportamento delle formiche per esplorare nuove soluzioni, vengono anche mantenute e promosse le soluzioni migliori T^{bs} (*élite*) trovate finora. In altre parole, le formiche tengono traccia delle soluzioni di qualità più elevata e queste soluzioni d'élite influenzano in modo significativo il processo di ricerca.
L'aggiunta di tale comportamento può accelerare la convergenza dell'algoritmo verso soluzioni di alta qualità, ma può anche comportare un aumento della memoria richiesta, poiché è necessario conservare le soluzioni d'élite. Questa tecnica è spesso utilizzata in problemi di ottimizzazione in cui è importante ottenere risultati di alta qualità in tempi ragionevoli.

Aggiornamento delle tracce di feromone
Il rinforzo aggiuntivo del cammino T^{bs} è ottenuto aggiungendo una quantità $\frac{\epsilon}{C^{bs}}$ ai suoi archi, dove ϵ è un parametro che definisce il peso dato al percorso migliore finora T^{bs} e C^{bs} è la sua lunghezza. Pertanto, l'equazione per il deposito di feromone diventa:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + \epsilon\Delta\tau_{ij}^{bs},$$

dove $\Delta\tau_{ij}^{k}$ è la quantità di feromone depositata dalla formica k sugli archi che ha visitato e $\Delta\tau_{ij}^{bs}$ è definito come segue:

$$\Delta\tau_{ij}^{bs} = \begin{cases} \frac{1}{C^{bs}} & se\ l'arco\ (i,j) \in T^{bs}, \\ 0 & altrimenti \end{cases}$$

Si noti che in EAS l'evaporazione del feromone è implementata come in AS.
I risultati computazionali presentati in Dorigo e collaboratori suggeriscono che l'uso della strategia elitaria con un valore appropriato per il parametro ϵ consente ad AS sia di trovare tour migliori, sia di trovarli in un minor numero di iterazioni.

3.3.3 Rank-Based Ant System

Un altro miglioramento rispetto all'AS è la "versione basata sul rango" dell'AS (AS_{rank}), proposta da Bullnheimer e collaboratori nel 1999 [12]. Questa variante introduce una modifica importante rispetto all'algoritmo classico: invece di basarsi esclusivamente sulla quantità di feromoni depositati dalle formiche per guidare la ricerca, l'AS_{rank} classifica le soluzioni in base alla loro qualità e utilizza questo ranking per influenzare la scelta delle formiche.

Aggiornamento delle tracce di feromone

Prima di aggiornare le tracce di feromone, le formiche vengono ordinate in base alla lunghezza crescente del percorso trovato e la quantità di feromone depositata da una formica è ponderata in base al suo rango r. Le situazioni di parità possono essere risolte casualmente (nella nostra implementazione vengono risolte tramite ordinamento lessicografico in base al nome della formica k). Ad ogni iterazione, solo le prime w - *1* formiche classificate per rango e la formica che ha prodotto il miglior risultato fino a quel momento possono depositare feromone. Il miglior percorso provvisorio fornisce il feedback più forte, con un peso w (la sua contribuzione $\frac{1}{C^{bs}}$ è moltiplicata per w); la formica r-esima migliore dell'iterazione corrente contribuisce all'aggiornamento del feromone con il valore $\frac{1}{C^r}$ moltiplicato per un peso dato da max$\{0,\ w$ - $r\}$.
La regola di aggiornamento del feromone diventa:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1} \Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs},$$

dove $\Delta\tau_{ij}^{r} = \frac{1}{C^r}$ e $\Delta\tau_{ij}^{bs} = \frac{1}{C^{bs}}$.
Questo può essere particolarmente utile in problemi in cui la qualità delle soluzioni è di fondamentale importanza.

3.3.4 Max-Min Ant System

La caratteristica distintiva dell'algoritmo *Max-Min Ant System* (MMAS) [21] è la sua capacità di bilanciare l'intensificazione (convergenza verso soluzioni di alta qualità) con la diversificazione (esplorazione di nuove soluzioni). Questo lo rende efficace nella ricerca di soluzioni ottime o prossime all'ottimo, evitando trappole di convergenza premature. Esso introduce quattro principali modifiche rispetto all'AS. In primo luogo, sfrutta fortemente i migliori percorsi trovati: solo la formica con la miglior soluzione nell'iterazione corrente è autorizzata a depositare feromone. Purtroppo, una tale strategia può portare a una situazione di stallo in cui tutte le formiche seguono lo stesso itinerario, a causa della crescita eccessiva delle tracce di feromone sugli archi di un buon, sebbene non ottimale, percorso. Per contrastare questo effetto, una seconda modifica introdotta da MMAS è quella di limitare il possibile intervallo dei valori delle tracce di feromone all'intervallo $[\tau_{min}, \tau_{max}]$. Terzo, le tracce di feromone vengono inizializzate al limite superiore, che, insieme a un basso tasso di evaporazione delle tracce, aumenta lo spazio di ricerca. Infine, le tracce di feromone vengono reinizializzate ogni volta che il sistema si avvicina allo stallo o quando non viene generato un tour migliore per un certo numero di iterazioni consecutive.

Aggiornamento delle tracce di feromone

Dopo che tutte le formiche hanno costruito un percorso, le tracce di feromone vengono aggiornate applicando l'evaporazione come in AS, seguita dal deposito di nuovo feromone come segue:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best},$$

dove $\Delta\tau_{ij}^{best} = 1/C^{best}$.

La formica autorizzata ad aggiungere feromone può essere sia la migliore finora, nel qual caso $\Delta\tau_{ij}^{best} = \frac{1}{C^{bs}}$, sia la migliore nell'iterazione, nel qual caso $\Delta\tau_{ij}^{best} = \frac{1}{C^{ib}}$, dove C_{ib} è la lunghezza del miglior percorso dell'iterazione.

In generale, nelle implementazioni di MMAS, entrambe le regole di aggiornamento delle tracce di feromone vengono utilizzate alternativamente. Ovviamente, la scelta della frequenza relativa con cui vengono applicate le due regole influisce sulla natura della ricerca: quando gli aggiornamenti delle tracce di feromone sono sempre effettuati dalla formica migliore finora, la ricerca si concentra molto rapidamente intorno a T_{bs}, mentre quando è la formica dell'iterazione migliore a effettuare gli aggiornamenti, il numero di archi che ricevono feromone è maggiore e la ricerca è meno diretta.

I risultati sperimentali indicano che per piccole istanze del TSP può essere meglio utilizzare solo gli aggiornamenti delle tracce di feromone dell'iterazione migliore, mentre per grandi TSP con diverse centinaia di città, le migliori prestazioni si ottengono dando un'importanza sempre maggiore al miglior cammino provvisorio.

Limiti delle tracce di feromone

In MMAS vengono imposti limiti inferiori e superiori $[\tau_{min}, \tau_{max}]$ sui possibili valori delle tracce di feromone su qualsiasi arco onde evitare situazioni di stallo. In particolare, i limiti imposti hanno l'effetto di limitare la probabilità p_{ij} di selezionare una città j quando una formica è nella città i all'intervallo $[p_{min}, p_{max}]$, con $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$. Solamente quando una formica k ha una sola possibile scelta per la prossima città, cioè $|N_k^i| = 1$, abbiamo $p_{min} = p_{max} = 1$.

È facile dimostrare che, a lungo termine, il limite superiore delle tracce di feromone su qualsiasi arco è limitato da $1/(\rho C^*)$, dove C^* è la lunghezza del tour ottimale. Sulla base di questo risultato, MMAS utilizza una stima di questo valore, $1/(\rho C^{bs})$, per definire τ_{max}: ogni volta che viene trovato un percorso migliore, il valore di τ_{max} viene aggiornato. Il limite inferiore delle tracce di feromone viene impostato a $\tau_{min} = \tau_{max}/a$, dove a è un parametro preimpostato.

I risultati sperimentali suggeriscono che, al fine di evitare l'*impasse*, i limiti inferiori delle tracce di feromone svolgono un ruolo più importante rispetto a τ_{max}. D'altra parte, τ_{max} rimane utile per impostare i valori delle tracce durante le occasionali reinizializzazioni.

Inizializzazione e Reinizializzazione delle tracce

All'inizio dell'algoritmo, le tracce di feromone vengono impostate su una stima del limite superiore. Questa particolare inizializzazione, in combinazione con il parametro ρ di evaporazione delle tracce, causa un aumento lento della differenza relativa nei livelli delle tracce, in modo che la fase iniziale di ricerca di MMAS sia molto esplorativa.
Come ulteriore mezzo per aumentare l'esplorazione dei percorsi che hanno solo una piccola probabilità di essere scelti, in MMAS le tracce di feromone vengono occasionalmente reinizializzate. Tale reinizializzazione viene solitamente attuata quando l'algoritmo si avvicina al comportamento di stallo o se per un dato numero di iterazioni dell'algoritmo non viene trovato una soluzione migliore.

3.4 Estensioni di Ant System

Gli algoritmi ACO che abbiamo presentato finora ottengono prestazioni significativamente migliori rispetto all'AS introducendo piccoli cambiamenti nella struttura generale dell'algoritmo. In questa sezione discuteremo un ulteriore algoritmo ACO che, sebbene fortemente ispirato dall'AS, raggiunge miglioramenti nelle prestazioni attraverso l'introduzione di nuovi meccanismi.

3.4.1 Il Sistema di Colonie di Formiche

Il Sistema di Colonie di Formiche, o *Ant Colony System* (ACS), si differenzia dall'AS in tre punti principali:

- Sfrutta in modo più consistente l'esperienza di ricerca accumulata dalle formiche attraverso l'uso di una regola di scelta delle azioni più rigida.
- L'evaporazione e il deposito del feromone avvengono solo sugli archi che appartengono al miglior percorso provvisorio.
- Ogni volta che una formica utilizza un arco (i, j) per spostarsi dalla città i alla città j, rimuove una quantità di feromone dall'arco per aumentare l'esplorazione di percorsi alternativi.

Costruzione del percorso

Nell'ACS, quando la formica k si trova nella città i, si sposta in una città j scelta secondo la cosiddetta regola *pseudorandom proporzionale*, data da:

$$j = \begin{cases} argmax_{l \in N_i^k} \tau_{il} (\eta_{il})^\beta & se\ q \leq q_0, \\ J & altrimenti, \end{cases}$$

dove q è una variabile uniformemente distribuita in [0, 1], q_0 ($0 \leq q_0 \leq 1$) è un parametro e J è una variabile casuale selezionata secondo la distribuzione di probabilità data dall'equazione standard dell'algoritmo di base, con $a = 1$ e $\beta \in [2, 5]$.
In altre parole, con probabilità q_0 la formica effettua la mossa migliore possibile come indicato dalle tracce di feromone e dalle informazioni euristiche, mentre con probabilità (1 - q_0) effettua una esplorazione guidata degli archi. La regolazione del parametro q_0

consente di modulare il grado di esplorazione e di scegliere se concentrare la ricerca del sistema attorno al miglior percorso provvisorio.

Aggiornamento globale delle tracce

Nell'ACS solo una formica, la migliore, è autorizzata ad aggiungere feromone dopo ogni iterazione. Quindi, l'aggiornamento nell'ACS è implementato mediante l'equazione seguente:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs}, \forall(i,j) \in T^{bs},$$

dove $\Delta\tau_{ij}^{bs} = 1/C^{bs}$.

È importante notare che nell'ACS l'aggiornamento delle tracce di feromone si applica solo agli archi di T^{bs} e non a tutti gli archi come nell'AS. Questo è importante perché in questo modo la complessità computazionale dell'aggiornamento delle tracce di feromone ad ogni iterazione è ridotta da $\mathcal{O}(n^2)$ a $\mathcal{O}(n)$, dove n è la dimensione dell'istanza che viene risolta. A differenza delle equazioni dell'AS, in questa la quantità di feromone depositato è scontata di un fattore ρ; ciò comporta che la nuova traccia di feromone è una media ponderata tra il vecchio valore di feromone e la nuova quantità di feromone depositato.

Inizialmente è stata anche presa in considerazione l'idea di utilizzare il miglior percorso dell'iterazione per gli aggiornamenti delle tracce di feromone. Anche se per piccole istanze del TSP le differenze nella qualità finale della soluzione ottenuta in questo modo sono minime, per istanze con più di 100 città l'utilizzo del miglior percorso provvisorio ha fornito risultati migliori.

Aggiornamento locale delle tracce

Oltre alla regola globale di aggiornamento delle tracce di feromone, nell'ACS le formiche utilizzano una regola locale di aggiornamento che applicano immediatamente dopo aver attraversato un arco (i, j) durante la costruzione del tour:

$$\tau_{ij} \leftarrow (1-\xi)\tau_{ij} + \xi\tau_0,$$

dove ξ, con $0 < \xi < 1$, e τ_0 sono due parametri. Il valore di τ_0 è impostato allo stesso valore iniziale delle tracce di feromone. Sperimentalmente, si è scoperto che un buon valore per ξ era 0.1, mentre un buon valore per τ_0 era $\frac{1}{n \cdot C^{nn}}$, dove n è il numero di città dell'istanza del TSP e C^{nn} è la lunghezza di un percorso del *Nearest Neighbor*.

L'effetto della regola di aggiornamento locale è il seguente: ogniqualvolta una formica utilizza un arco (i, j), la sua traccia di feromone τ_{ij} viene ridotta, in modo che l'arco diventi meno desiderabile per le formiche successive. In altre parole, ciò consente un aumento dell'esplorazione degli archi che non sono stati ancora visitati e, nella pratica, impedisce che l'algoritmo mostri un comportamento di stasi. È importante notare che, mentre per le varianti AS discusse in precedenza non fa differenza se le formiche costruiscono i percorsi in parallelo o in sequenza, questo fa differenza nell'ACS a causa della regola di aggiornamento locale delle tracce di feromone. In particolare, nella maggior parte delle implementazioni dell'ACS, la scelta è stata quella di far muovere tutte le formiche in parallelo.

3.4.2 Ulteriori considerazioni

L'ACS si basa su *Ant-Q*, un algoritmo precedente proposto da Gambardella e Dorigo nel 1995 [22]. In poche parole, l'unica differenza tra ACS e Ant-Q sta nella definizione del termine τ_0, il quale in Ant-Q viene impostato a $\tau_0 = \gamma \cdot max_{j \in N_i^k} \tau_{ij}$, dove γ è un parametro e il massimo è preso dall'insieme di tracce di feromone sugli archi che collegano la città i su cui si trova la formica k a tutte le città che la formica non ha ancora visitato, ossia quelle nel vicinato N_i^k. Questa scelta particolare per τ_0 è stata motivata da un'analogia con una formula simile utilizzata in *Q-learning*, un noto algoritmo di apprendimento per rinforzo.

Esiste anche una relazione interessante tra MMAS e ACS: entrambi utilizzano limiti di tracce di feromone, sebbene questi siano espliciti in MMAS e impliciti in ACS. Infatti, nelle implementazioni dell'ACS, le tracce di feromone non possono mai scendere al di sotto di τ_0 perché entrambe le regole di aggiornamento delle tracce aggiungono sempre una quantità di feromone maggiore o uguale a τ_0, e il valore iniziale delle tracce di feromone è impostato al valore τ_0. D'altra parte, è possibile verificare facilmente che le tracce di feromone non possono mai avere un valore superiore a $1/C^{bs}$. Pertanto, nell'ACS è implicitamente garantito che $\forall (i,j) : \tau_0 \leq \tau_{ij} \leq 1/C^{bs}$.

Infine, va menzionato che ACS è stato il primo algoritmo ACO a utilizzare liste di candidati per limitare il numero di scelte disponibili da considerare in ciascun passo di costruzione. Nel caso del TSP, una lista di candidati contiene per ogni città i le città j che si trovano a breve distanza. Ci sono diversi modi per definire quali città entrano nelle liste di candidati. ACS ordina prima i vicini di una città i in base alle distanze crescenti e inserisce quindi un numero fisso di candidati delle città più vicine nella lista di candidati di i. In questo caso, le liste di candidati possono essere create prima di risolvere un'istanza del TSP e rimangono fisse durante l'intero processo di soluzione. Quando si trova in i, una formica sceglie la prossima città tra quelle nella lista di candidati di i che non sono ancora state visitate. Solo se tutte le città nella lista di candidati sono già state segnate come visitate, verrà scelta una delle città rimanenti. Nel caso del TSP, i risultati sperimentali hanno mostrato che l'uso delle liste di candidati migliora la qualità della soluzione raggiunta.

3.5 Valutazione dei differenti approcci

Il confronto tra queste estensioni dell'AS dipende in gran parte dalla natura specifica del problema da risolvere. L'EAS e il AS_{rank} sono orientati principalmente all'esplorazione e alla diversificazione, mentre il MMAS e l'ACS tendono a favorire la convergenza verso soluzioni di alta qualità. La scelta tra queste estensioni dipenderà quindi dal peso attribuito a esplorazione ed esecuzione, oltre che dalle caratteristiche specifiche del problema. Sperimentazione e calibrazione dei parametri sono spesso necessarie per ottenere i migliori risultati in contesti reali.

Capitolo 4

Conclusioni

Esiste una vasta gamma di problemi da risolvere, fenomeni da sintetizzare e domande aperte a cui rispondere. In tutti questi casi possono essere utilizzati uno o più approcci di calcolo naturale per risolvere il problema, sintetizzare il fenomeno o rispondere alla domanda. Tuttavia, gli approcci di calcolo naturale non sono gli unici strumenti che forniscono soluzioni a questi problemi, né sono sempre gli approcci più adatti ed efficienti. In molte situazioni esistono tecniche alternative di soluzione per un determinato problema e queste possono fornire risultati migliori. Pertanto, è importante investigare attentamente il problema da risolvere prima di scegliere un metodo di soluzione specifico.

In generale, la computazione naturale dovrebbe essere utilizzata quando:

1. Il problema da risolvere è complesso, ossia coinvolge un gran numero di variabili o soluzioni potenziali.

2. Non è possibile garantire che una soluzione potenziale trovata sia ottimale, ma è possibile trovare una misura di qualità che consenta il confronto delle soluzioni tra loro.

3. Una singola soluzione non è sufficiente. La maggior parte delle tecniche standard di risoluzione dei problemi è in grado di fornire una sola soluzione a un dato problema, ma non è in grado di fornire più di una soluzione. Uno dei motivi è che la maggior parte delle tecniche standard sono deterministiche, mentre il calcolo naturale è in gran parte composto da metodi stocastici.

4. Bisogna sintetizzare con realismo sistemi e processi biologici, fisici e chimici. La geometria euclidea è molto buona ed efficiente per creare forme artificiali, ma ha difficoltà a riprodurre pattern naturali. Questo perché la natura è frattale e la geometria frattale fornisce gli strumenti più adatti per sintetizzare pattern naturali.

5. Si raggiungono i limiti della tecnologia attuale o è necessario cercare nuovi materiali per il calcolo.

In conclusione, l'avvento del Natural Computing ha aperto nuove prospettive nell'ambito della risoluzione di problemi complessi ispirandosi a principi provenienti dalla natura stessa. Abbiamo ad esempio illustrato come l'algoritmo ACO si sia dimostrato un potente strumento per affrontare il Problema del Commesso Viaggiatore. Tale approccio, ispirato al comportamento delle formiche nella ricerca di percorsi ottimali tra fonti di cibo, ha dimostrato di essere in grado di trovare soluzioni competitive in tempi ragionevoli, anche per istanze di TSP di grandi dimensioni.

In questa analisi è stato esaminato in dettaglio il funzionamento dell'algoritmo ACO, evidenziando le sue componenti chiave tra cui la costruzione delle soluzioni attraverso depositi di feromoni e l'evaporazione graduale di questi depositi per favorire l'emergere di soluzioni migliori nel corso del tempo.
La sperimentazione condotta su istanze del Problema del Commesso Viaggiatore di varie dimensioni e complessità ha dimostrato che gli algoritmi ACO sono in grado di fornire soluzioni di qualità comparabile o addirittura superiore rispetto ad altre tecniche di risoluzione. Tuttavia, è importante sottolineare che la performance degli algoritmi ACO può essere influenzata dalla scelta dei parametri e dalla struttura della rete dei depositi di feromoni. Pertanto, è fondamentale condurre un'analisi approfondita e un'ottimizzazione dei parametri per ottenere risultati ottimali.

Nonostante i successi raggiunti, esistono ancora sfide aperte nell'utilizzo degli algoritmi ACO per il Problema del Commesso Viaggiatore: la scalabilità a istanze di dimensioni molto grandi e la gestione efficiente dei vincoli sono questioni che richiedono ulteriori studi e sviluppi. Inoltre, l'ibridazione con altre tecniche di ottimizzazione e la combinazione con metodi di *machine learning* rappresentano stimoli interessanti per migliorare ulteriormente le performance degli algoritmi di questo tipo.

Bibliografia

[1] Leandro Nunes de Castro. "Fundamentals of natural computing: an overview". In: *Physics of Life Reviews* (2007). DOI: https://doi.org/10.1016/j.plrev.2006.10.002.

[2] *Mandelbrot e la geometria frattale*. URL: https://matematica.unibocconi.eu/articoli/mandelbrot-e-la-geometria-frattale.

[3] Stephen Wolfram. "Cellular automata as models of complexity". In: *Nature* 311 (1984), pp. 419–424. DOI: https://doi.org/10.1038/311419a0.

[4] James Hanan Przemyslaw Prusinkiewicz. *Lindenmayer Systems, Fractals, and Plants*. Springer Science Business Media, 2013.

[5] Mitchel Resnick. "Beyond the Centralized Mindset". In: *Journal of the Learning Sciences* 5 (2001), pp. 1–22.

[6] Junzo Watada. "DNA Computing and Its Applications". In: *IEEE Xplore* (2008). DOI: https://doi.org/10.1109/ISDA.2008.362.

[7] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21 (1982), pp. 467–488. DOI: https://doi.org/10.1007/BF02650179.

[8] *Osprey, il più grande computer quantistico mai realizzato*. URL: https://www.wired.it/article/computer-quantistico-ibm-piu-grande-mai-realizzato-dettagli.

[9] Guy Theraulaz Eric Bonabeau Marco Dorigo. *Swarm Intelligence - From Natural to Artificial Systems*. OUP USA, 1999.

[10] Al-Mahmud. "Highly Constrained University Class Scheduling using Ant Colony Optimization". In: *International Journal of Computer Science & Information Technology (IJCSIT)* 13 (2021).

[11] Thomas Stützle Marco Dorigo. "Ant Colony Optimization Algorithms for the Traveling Salesman Problem". In: *MIT Press* (2004), pp. 65–119.

[12] C. Strauss B. Bullnheimer R.F. Hartl. "An improved Ant System algorithm for the Vehicle Routing Problem". In: *Annals of Operations Research* 89 (1999), pp. 319–328. DOI: https://doi.org/10.1023/A:1018940026670.

[13] S.X. Yang Ting Lan Shirong Liu. "Collective Cooperation Inspired by Stigmergy Strategy". In: *2006 6th World Congress on Intelligent Control and Automation* (2006), pp. 441–445. DOI: https://doi.org/10.1109/WCICA.2006.1712355.

[14] S. Goss J. M. Pasteels J.L. Deneubourg S. Aron. "The self-organizing exploratory pattern of the argentine ant". In: *Journal of Insect Behavior* 3 (1990), pp. 159–168. DOI: https://doi.org/10.1007/BF01417909.

[15] Marco Dorigo. "Ant Colony Optimization". In: *IEEE Computational Intelligence Magazine* 1 (2006), pp. 28–39. DOI: https://doi.org/10.1109/MCI.2006.329691.

[16] A. Colorni Marco Dorigo V. Maniezzo. "Ant system: optimization by a colony of cooperating agents". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26 (1996), pp. 29–41. DOI: https://doi.org/10.1109/3477.484436.

[17] R. Eberhart J. Kennedy. "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks* 4 (1995), pp. 1942–1948. DOI: https://doi.org/10.1109/ICNN.1995.488968.

[18] Anxin Ye. "Study of the vehicle routing problem with time windows based on improved particle swarm optimization algorithm". In: *2011 International Conference on Computer Science and Service System* (2011), pp. 4053–4057. DOI: https://doi.org/10.1109/CSSS.2011.5974924.

[19] Celal Ozturk Dervis Karaboga Beyza Gorkemli. "A comprehensive survey: artificial bee colony (ABC) algorithm and applications". In: *Artificial Intelligence Review* 42 (2014), pp. 21–57. DOI: https://doi.org/10.1007/s10462-012-9328-0.

[20] Fırat Hardalaç Okan Erkut. "Comparison of Ant Colony Optimization and Artificial Bee Colony Algorithms for Solving Electronic Support Search Dwell Scheduling Problem". In: *15th Turkish National Software Engineering Symposium (UYMS)* (2021), pp. 1–6. DOI: https://doi.org/10.1109/UYMS54260.2021.9659666.

[21] Holger H. Hoos Thomas Stützle. "MAX-MIN Ant System". In: *Future Generation Computer Systems* 16 (2000), pp. 889–914. DOI: https://doi.org/10.1016/S0167-739X(00)00043-1.

[22] Marco Dorigo Luca M. Gambardella. "Ant-Q: A Reinforcement Learning approach to the traveling salesman problem". In: *Proceedings of the Twelfth International Conference on Machine Learning* (1995), pp. 252–260. DOI: https://doi.org/10.1016/B978-1-55860-377-6.50039-6.