UNIVERSITÀ DI PARMA
DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

K-MEANS PSO PER LA RICERCA DI SOTTOINSIEMI RILEVANTI DI VARIABILI NEI SISTEMI COMPLESSI

K-MEANS PSO FOR SEARCHING RELEVANT VARIABLE SUB-SETS IN COMPLEX SYSTEMS

Relatore: Chiar.mo Prof. Stefano Cagnoni
Correlatore: Dott.Ing. Laura Sani

Candidato: GIANLUIGI SILVESTRI

ANNO ACCADEMICO 2016/2017

Ringraziamenti

Ringrazio

il relatore di questa tesi
Chiar.mo Prof. Stefano Cagnoni

e il correlatore
Dott. Ing. Laura Sani

Ringrazio inoltre:
Prof. Michele Amoretti
Prof.ssa Monica Mordonini

Indice

Elenco delle figure

Elenco degli Algoritmi

Capitolo 1

Introduzione

1.1 Analisi di sistemi complessi

Un sistema complesso è un sistema in cui le singole parti sono interessate da interazioni locali, di breve raggio d'azione, che provocano cambiamenti nella struttura macroscopica. Numerosi sistemi del mondo reale possono essere visti come sistemi complessi. Alcuni esempi possono essere organizzazioni politiche, il cervello umano, i social network, e anche le reti di regolatori genetici all'interno di una cellula. I sistemi complessi sono sistemi caratterizzati da proprietà ben definite che, oltre ad essere utili ad indentificarli, chiariscono la loro natura ed importanza. In particolare possiamo di individuare come sistemi complessi quei sistemi che presentano le seguenti proprietà:

- **Auto-organizzazione**: I sistemi complessi operano senza un controllo centrale, e si organizzano da soli "a basso livello";
- **Comportamento emergente**: I sistemi complessi mostrano un comportamento emergente. Dalle interazioni tra gli elementi individuali del sistema emerge il suo comportamento globale. Questo viene chiamato comportamento di ordine superiore e non può essere derivato semplicemente aggregando i comportamenti al livello dei singoli elementi.

- **Non linearità**: I sistemi complessi mostrano un comportamento non lineare. Questo significa che potrebbero cambiare comportamento improvvisamente, o passare da un alto grado di stabilità ad un comportamento instabile.
- **Prevedibilità limitata**: Il comportamento di un sistema complesso non può essere previsto completamente. Piccoli cambiamenti nelle condizioni iniziali o nella storia possono portare a dinamiche completamente differenti nel tempo. La non linearità del loro comportamento aggiunge imprevedibilità.
- **Grandi eventi**: Cambiamenti relativamente piccoli possono portare a grandi conseguenze.
- **Dinamiche evolutive**: Sistemi complessi adattivi sono spesso modellati da dinamiche evolutive.

L'analisi dei sistemi complessi [13] riguarda la misura della loro complessità, oltre alla ricerca di strutture rilevanti a elevata interazione, sulla base dell'evoluzione del loro stato nel tempo. La ricerca di queste strutture si può basare sull'analisi di caratteristiche correlate alla topologia delle reti o sull'osservazione delle dinamiche del sistema. La metodologia da noi utilizzata [22] consiste nel considerare diversi sottoinsiemi del sistema, cercando gli elementi che sembrano ben coordinati tra di loro e che abbiano deboli interazioni con il resto del sistema (Mesolevel Dynamical Structures). Per ogni sottoinsieme misureremo un "cluster index" *CI*, derivato dalla teoria dell'informazione, che ci consentirà di analizzare e individuare delle strutture rilevanti. Questo approccio ha il vantaggio di non richiedere conoscenza sulla struttura del sistema e sulle regole che guidano le sue componenti, ma il calcolo dell'indice *CI* deve essere effettuato in maniera esaustiva su tutti i possibili sottoinsiemi delle variabili del sistema, in modo da ottenerne una descrizione completa. All'aumentare del numero di variabili, i tempi impiegati da questa procedura crescono esponenzialmente e, anche ottimizzando la ricerca parallelizzando l'elaborazione (per esempio su GPU), con un numero relativamente grande di variabili i tempi di calcolo non risulterebbero comunque ragionevoli. Per ovviare a questo problema, si cercano quindi procedure più veloci che

forniscano risultati comparabili con quelli che fornirebbe la ricerca esaustiva in tempi significativamente ridotti.

1.2 Metaeuristiche e calcolo evolutivo

Da quello che abbiamo visto nella sezione precedente, possiamo considerare la ricerca di strutture rilevanti in un sistema complesso come un problema di ottimizzazione il cui obiettivo è individuare quei sottoinsiemi di variabili che massimizzano il *CI*. A seconda delle dimensioni e degli obiettivi del problema, possiamo utilizzare diverse metodologie di risoluzione. Se le dimensioni del problema sono relativamente piccole, il metodo esaustivo garantisce una soluzione esatta del problema in tempi ragionevoli. Al contrario, in caso di problemi di dimensioni elevate, l'unica soluzione sta nell'utilizzare metodi euristici, cioè che risolvano un problema di ottimizzazione utilizzando in genere regole di buon senso, fornendo una soluzione ammissibile ma non necessariamente ottima in un tempo relativamente breve. Per essere efficace, un metodo euristico deve sfruttare le caratteristiche strutturali del problema che deve risolvere: non esiste quindi un'euristica generale. Esistono però degli approcci generici a cui si può fare riferimento per sviluppare euristiche specifiche. Tali approcci vengono detti metaeuristiche.
A partire dagli anni '80 sono stati proposti numerosi paradigmi metaeuristici, tra cui i più popolari sono:

- Simulated Annealing [11];
- Tabu Search [8];
- Algoritmi evolutivi [7].
- ANT colony optimization [6];
- Particle Swarm Optimization (PSO) [10];

Queste metaeuristiche non risultano però adatte a mantenere la diversità delle soluzioni, poiché tendono a convergere verso una singola soluzione ottima. Per evitare questo

effetto, detto anche di “genetic drift”, si utilizzano delle tecniche di Niching, che mirano all’esplorazione in parallelo di diverse soluzioni ottime.
L’algoritmo progettato e realizzato ha l’obiettivo di massimizzare l’indice *CI*, individuando le prime k soluzioni migliori. La metaeuristica sviluppata si basa sul paradigma PSO con l’introduzione di tecniche di Niching.

1.3 Descrizione capitoli

La tesi è strutturata nei seguenti capitoli:

- *capitolo 2*: stato dell’arte (individuazione di strutture rilevanti nei sistemi complessi, metaeuristiche, Particle Swarm Optimization, tecniche di Niching);
- *capitolo 3*: descrizione dell’algoritmo realizzato e della parallelizzazione realizzata sfruttando l’architettura CUDA;
- *capitolo 4*: descrizione dei risultati sperimentali;
- *capitolo 5*: conclusioni e sviluppi futuri.

Capitolo 2

Stato dell'arte

2.1 Ricerca di strutture rilevanti nei sistemi complessi

Lo studio dei sitemi complessi [13] è correlato all'analisi delle proprietà emergenti e dei comportamenti collettivi dei sistemi stessi. I principali obiettivi riguardano l'identificazione di strutture rilevanti e la misura della complessità all'interno di sistemi le cui componenti sono solitamente note, ricavando informazioni dall'evoluzione del loro stato nel tempo.
Molti sistemi complessi possono essere descritti attraverso reti di nodi che interagiscono tra loro, ma le corrispondenze a livello topologico delle dinamiche espresse da questi sistemi sono spesso complicate da individuare. Per evitare quindi una descrizione meramente topologica [22] consideriamo diversi sottinsiemi del sistema, cercando quelli i cui elementi sembrino ben coordinati tra loro ed abbiano una interazione debole con i restanti (Mesolevel Dynamical Structures, MSD). I sottinsiemi possibili possono essere valutati sulla base di diverse misure derivate dalla teoria dell'informazione. Nelle seguenti sottosezioni ne vengono descritte alcune.

2.1.1 Cluster Index

Il Cluster Index (CI) è una misura introdotta nel 1994 e 1998 da Edelman, Tononi et al. per individuare gruppi funzionali di regioni del cervello [21]. Un cluster [13] è un

sottinsieme di elementi relativamente isolati dal resto del sistema, ma che tra di loro mostrano un comportamento coordinato, coerente e coeso. Proprietà dei cluster sono:

- **mutua informazione:** misura della dipendenza statistica tra un sottinsieme di elementi e il resto del sistema;
- **integrazione:** misura della dipendenza statistica totale all'interno di un sottoinsieme di elementi.

Consideriamo un insieme U composto da N elementi con valore finito e discreto [23]. Il Cluster index è una misura teorica basata sull'entropia di Shannon degli elementi e dei gruppi di elementi di U.
In accordo con la teoria dell'informazione, l'entropia di un elemento x_i è definita come:

$$H(x_i) = -\sum_{v \in V_i} p(v) log(p(v))$$

dove Vi è l'insieme di possibili valori di x_i e $p(v)$ è la probabilità di occorrenza del simbolo v.
L'entropia di una coppia di elementi x_i e x_j è definita come:

$$H(x_i, x_j) = -\sum_{v \in V_i} \sum_{w \in V_j} p(v,w) log(p(v,w))$$

Questa equazione si può estendere a un gruppo di k elementi considerando la probabilità di occorrenze di vettori di k valori.
Il cluster index $C(S)$ di un insieme S di k elementi è definito come il rapporto tra l'integrazione $I(S)$ di S e la mutua informazione tra S e il resto del sistema U/S.
Definiamo l'integrazione come:

$$I(S) = \sum_{x \in S} H(x) - H(S)$$

dove *I(S)* rappresenta la deviazione dall'indipendenza statistica dei k elementi in S.
Definiamo ora la mutua informazione *M(S; U/S)* come:

$$M(S;U/S) = H(S) + H(S|U/S) = H(S) + H(U/S) - H(S,U/S)$$

dove *H(A| B)* è l'entropia condizionata e *H(A, B)* è l'entropia congiunta.
Infine, il cluster index *C(S)* è definito come:

$$C(S) = \frac{I(S)}{M(S;U/S)}$$

Visto che C viene definito come rapporto, è indefinito in tutti i casi in cui *M(S; U/S)* si annulla. In questo caso, comunque, il sottinsieme S è statisticamente indipendente dal resto del sistema e va quindi analizzato separatamente.
Bisogna notare che *C(S)* dipende dalle dimensioni di S, quindi il valore del cluster index associato a sottosistemi di dimensioni differenti non può essere confrontato. Per superare questa limitazione, dobbiamo ricorrere alla normalizzazione del cluster index. Per farlo definiamo un sistema omogeneo di riferimento U_h. Nei sistemi omogenei ogni variabile ha lo stesso valore atteso del sistema che si sta analizzando, ma la correlazione fra coppie di variabili è imposta costante. Poi, per avere un valore di riferimento per il cluster index, per ogni dimensione d di un sottinsieme di U_h calcoliamo le medie dell'integrazione e della mutua informazione $\langle I_{\mathrm{h,d}} \rangle$ e $\langle M_{\mathrm{h,d}} \rangle$.
Quindi, il valore del cluster index di ogni sottosistema S può essere normalizzato per eliminare la dipendenza dalla dimensione dividendolo per la media delle grandezze considerate per ogni dimensione di S:

$$C'(S) = \frac{I(S)/\langle I_h \rangle}{M(S;U/S)/\langle M_h \rangle}$$

Inoltre, per valutare la rlievanza delle differenze osservate nei valori dei cluster, viene calcolato un indice statistico T_c:

$$T_c(S) = \frac{C'(S) - \langle C'_h \rangle}{\sigma(C'_h)} = \frac{vC(S) - v\langle C_h \rangle}{v\sigma(C_h)} = \frac{C(S) - \langle C_h \rangle}{\sigma(C_h)}$$

dove $\langle C'_{\mathrm{h}} \rangle$ e $\sigma(C'_h)$ sono rispettivamente la media e la deviazione standard dei cluster index della popolazione normalizzati con le stesse dimensioni di S del sistema omogeneo, usando $v = \langle M_{\mathrm{h}} \rangle / \langle I_{\mathrm{h}} \rangle$ come la costante di normalizzazione.

2.2 Metaeuristiche

Si definisce *metaeuristica* [13] l'insieme di algoritmi, tecniche e studi relativi all'applicazione di criteri euristici per risolvere problemi di ottimizzazione, in modo da ottenere una ricerca globale basata su un principio generale.
Alcune metaeuristiche il cui utilizzo è ampiamente diffuso sono:

- **Simulated Annealing:** strategia utilizzata per risolvere problemi di ottimizzazione, che mira a trovare un minimo globale quando si è in presenza di più minimi locali. L'algoritmo si ispira al processo metallurgico di ricottura dei metalli;

- **Tabu Search**: strategia utilizzata per la soluzione di numerosi problemi di ottimizzazione, tra cui problemi di scheduling e routing, problemi su grafi e programmazione intera. Realizza una generalizzazione di una ricerca locale consentendo di accettare "mosse" peggiorative, che sfrutta una memoria di breve termine (tabu list) per evitare di riconsiderare le ultime t soluzioni visitate.

- **Algoritmi evolutivi**: algoritmi euristici che si ispirano al principio di evoluzione naturale degli esseri viventi. Semplificando, si può affermare che un algoritmo evolutivo prevede di partire da una soluzione e di farla evolvere con una serie di modifiche casuali e un processo di selezione per ottenere soluzioni sempre migliori.

- **ANT colony optimization**: metaeuristica per risolvere problemi di ottimizzazione, si ispira al rilascio di feromone e alla tendenza a seguirne la traccia che caratterizza le formiche.

- **Particle Swarm Optimization**: famiglia di algoritmi relativamente recenti che possono essere utilizzati per trovare soluzioni ottime a problemi numerici e com-

binatori. Fa emergere intelligenza computazionale a partire da semplici analogie con l'interazione sociale, in particolare dal comportamento di uno stormo di uccelli alla ricerca di cibo.

In questa tesi abbiamo utilizzato un algoritmo basato sul Particle Swarm Optimization, che verrà descritto dettagliatamente nella sezione seguente.

2.3 Particle Swarm Optimization

Nella PSO [18], un numero di semplici entità (particelle) viene introdotto nello spazio di ricerca di qualche problema o funzione. Di ciascuna particella viene valutata la funzione obiettivo (fitness) nella sua posizione corrente. Successivamente, ogni particella determina il proprio movimento all'interno dello spazio di ricerca, combinando informazioni sulla propria miglior posizione visitata con quella di uno o più dei restanti membri dello sciame. Una volta che tutte le particelle si sono mosse, si ripete questo passaggio iterativamente e lo sciame, che inizialmente era disperso per tutto lo spazio, tende a convergere al valore ottimo della funzione di fitness.

2.3.1 Versione originale del PSO

Ogni particella dello sciame è descritta da tre vettori di D dimensioni, dove D è il numero di dimensioni dello spazio di ricerca. Questi vettori sono la posizione corrente $\vec{x}_i$, la miglior posizione visitata dalla particella $\vec{p}_i$, e la velocità $\vec{v}_i$. La posizione corrente $\vec{x}_i$ può essere considerata come un vettore di coordinate che descrivono un punto nello spazio. Ad ogni iterazione dell'algoritmo si calcola la fitness nella posizione corrente e, se questo valore è migliore di quelli trovati finora dalla particella, si memorizza in una variabile $pbest_i$ e si salva la posizione corrente nel vettore $\vec{p}_i$. L'obiettivo sarà ovviamente trovare posizioni sempre migliori. Le nuove posizioni delle particelle si ottengono aggiungendo le coordinate del vettore $\vec{v}_i$ ad $\vec{x}_i$:
$\vec{x_{i+1}} = \vec{x}_i + \vec{v}_i$.
Una componente importante dell'algoritmo è quella di interazione tra le particelle, che si scambiano informazioni tra loro secondo un'opportuna struttura di comunicazione.

Ogni particella apparterrà quindi a un gruppo, e il suo movimento sarà influenzato dalla miglior posizione ($\vec{p}_g$) trovata dalle altre particelle che vi appartengono, oltre che dal $\vec{p}_i$.
Il procedimento originale per implementare il PSO è descritto nel seguente algoritmo:

Algorithm 1 Algoritmo originale PSO

```
1:  procedure PSO
2:      Inizializzare le particelle con posizioni e velocità casuali.
3:      Calcolare la fitness delle particelle.
4:      Impostare p⃗ e pbest di ogni particella alla posizione corrente.
5:      loop
6:          Aggiorna la velocità delle particelle.
7:          Aggiorna la posizione delle particelle.
8:          Ricalcola la fitness delle particelle.
9:          Aggiornare eventualmente p⃗ e pbest.
10:         Se un criterio di terminazione viene soddisfatto, uscire dal loop.
11:     end loop
12: end procedure
```

Come criterio di terminazione si usa generalmente il raggiungimento di un certo valore di fitness o l'esecuzione di un numero massimo di iterazioni.
L'equazione di aggiornamento di velocità e posizione è la seguente:

$$\begin{cases} \vec{v}_i \leftarrow \vec{v}_i + \vec{U}(0,\phi_1) \otimes (\vec{p}_i - \vec{x}_i) + \vec{U}(0,\phi_2) \otimes (\vec{p}_g - \vec{x}_i) \\ \vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i \end{cases}$$

Note:

- $\vec{U}(0,\phi_i)$ rappresenta un vettore di numeri naturali distribuiti uniformemente in [0,ϕ_i] che viene generato casualmente a ogni iterazione e per ogni particella.
- $\otimes$ è la moltiplicazione componente per componente.

- Nella versione originale del PSO, ogni componente $\vec{v}_i$ viene mantenuta in un range [-V_{max},+V_{max}].

2.3.2 Parametri

La procedura appena descritta ha un numero limitato di parametri da impostare. Il primo è ovviamente la dimensione della popolazione, cioè il numero di particelle che compongono lo sciame. I parametri ϕ_1 e ϕ_2, che determinano la forza d'attrazione dei vettori $\vec{p}_i$ e $\vec{p}_g$ sulla particella, sono spesso chiamati coefficienti d'accelerazione.
Per assicurare stabilità all'algoritmo ed evitare un aumento incontrollato della velocità, inizialmente si usava limitare quest'ultima, in modo tale da contentere $\vec{v}_i$ all'interno del range [-V_{max},+V_{max}]. Questo approccio però influenza il bilanciamento tra esplorazione dello spazio di ricerca e convergenza nei valori ottimi, oltre a rendere l'algoritmo dipendente dal problema specifico per la determinazione di V_{max}.

2.3.3 Peso di inerzia

Per controllare meglio l'esplorazione dello spazio di ricerca e ridurre l'importanza di V_{max}, la seguente modifica è stata apportata all'equazione di aggiornamento di $\vec{x}_i$ e $\vec{v}_i$ [20]:

$$\begin{cases} \vec{v}_i \leftarrow \omega\vec{v}_i + \vec{U}(0,\phi_1) \otimes (\vec{p}_i - \vec{x}_i) + \vec{U}(0,\phi_2) \otimes (\vec{p}_g - \vec{x}_i) \\ \vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i \end{cases}$$

dove ω è detto "peso d'inerzia", interpretabile come la fluidità del mezzo in cui le particelle si muovono.
Con questa equazione e una scelta appropriata di ω e dei coefficienti di accelerazione ϕ_1 e ϕ_2, il PSO può essere reso molto più stabile, a tal punto da poter eliminare la limitazione di velocità V_{max}.

2.3.4 Fattore di costrizione

Un altro metodo per controllare il comporamento dello sciame di particelle è l'introduzione di un "fattore di costrizione" (*constriction factor*). Questo metodo è stato

proposto da Clerc in [5] e, nella sua forma più semplice, il fattore di costrizione è un coefficiente χ applicato a entrambi i termini della formula di aggiornamento della velocità:

$$\begin{cases} \vec{v}_i \leftarrow \chi(\omega\vec{v}_i + \vec{U}(0,\phi_1) \otimes (\vec{p}_i - \vec{x}_i) + \vec{U}(0,\phi_2) \otimes (\vec{p}_g - \vec{x}_i)) \\ \vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i \end{cases}$$

Dove $\phi = \phi_1 + \phi_2 > 4$ e

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}}$$

Questo metodo di costrizione influisce sulla convergenza delle particelle nel tempo, cioè l'ampiezza delle oscillazioni delle singole particelle diminuisce lentamente concentrandosi attorno al $\vec{p}_i$.

2.3.5 Topologie degli sciami

Le prime versioni di PSO tenevano conto della vicinanza fisica tra le particelle per l'aggiornamento di velocità e posizione, ed erano quindi basate sull'effettiva prossimità nello spazio di ricerca. Questo tipo di struttura di comunicazione presentava indesiderati effetti di convergenza prematura.
Nella topologia introdotta successivamente, detta topologia *gbest* (per "global best"), ogni particella era influenzata dal miglior valore trovato dall'intero sciame.
Un'altra strategia, detta invece *lbest* (per "local best"), consiste nel formare diversi gruppo di particelle utilizzando una particolare topologia. Per esempio, nella topologia circolare, ogni particella è connessa a K$\geq$2 membri adiacenti nell'array della popolazione. Con il metodo *lbest* si ha il vantaggio di consentire una ricerca parallela, visto che i sottogruppi possono convergere su diverse regioni dello spazio di ricerca. Nonostante questa topologia converga più lentamente di quella *gbest*, risulta comunque meno vulnerabile all'attrazione di ottimi locali.

2.4 Niching

L'ottimizzazione di una funzione [9] è spesso vista come un processo per localizzare l'ottimo globale, cioè la posizione nello spazio di ricerca dove la funzione assume il miglior valore possibile. Però per diversi problemi di ottimizzazione, è necessario localizzare più di un valore ottimo. In questo caso, la parte di spazio di ricerca dove gli individui hanno una naturale tendenza a convergere su una specifica soluzione ottima è nota come *nicchia* (in inglese "niche"). Quindi gli algoritmi progettati per la localizzazione di soluzioni ottime multiple vengono detti *algoritmi di niching*.
In ecologia, una nicchia descrive la posizione relazionale di una specie nel suo ecosistema. Questa nicchia ecologica descrive come un organismo o popolazione risponda alla distribuzione di risorse e competitori, e come questi fattori possano essere alterati dall'organismo o popolazione. Quindi, popolazioni diverse possono sopravvivere e coesistere utilizzando l'ambiente in modi differenti. Specie diverse evolvono per riempire nicchie ecologiche differenti. Di conseguenza gli algoritmi di niching vengono anche chiamati *species-based*.

2.4.1 Tecniche di niching per il PSO

Lo scopo del PSO originale era quello di trovare un singolo ottimo di una funzione n-dimensionale. Una delle principali ragioni per cui la formazione di nicchie è inibita è l'influenza della componente sociale nell'equazione di aggiornamento della velocità, che fa in modo di attrarre le particelle verso la soluzione migliore in tutto lo sciame. Ogni volta che viene trovata una nuova soluzione migliore, le particelle verranno attratte verso la nuova area più promettente dello spazio di ricerca.
L'idea di ottimizzare porzioni di sciame separatamente attraverso l'uso di topologie di interconnessione è stata quindi introdotta principalmente per cercare di migliorare la diversità delle soluzioni. Questo è un buon punto di partenza per progettare algoritmi di niching, anche se per questo scopo sono più adatte topologie basate sulla vicinanza spaziale.
Nel seguito sono riportate le descrizioni delle principali tecniche di niching per il PSO.

Objective function stretching

Questa tecnica, introdotta da Parsopoulos et al. in [15], è stata una delle prime strategie per analizzare funzioni multimodali. Lo scopo dell'autore era principalmente quello di superare le limitazioni del PSO causate da una convergenza prematura a soluzioni locali. L'approccio di "stretching" opera sulla funzione di fintess, modificandola per rimuovere ottimi locali già individuati.
Se consideriamo ad esempio un problema di minimizzazione, dove lo sciame può convergere su un minimo locale, la fitness in quel punto viene modificata, facendola diventare un massimo locale. In questo modo, iterazioni successive del PSO si concentreranno in altre aree dello spazio di ricerca, identificando altre soluzioni.

NichePSO

Nel 2002 Brits *et al.* hanno introdotto l'algoritmo *nbest* PSO, prima tecnica ad introdurre niching parallelo nel particle swarm [4]. L'*nbest* PSO era particolarmente adatto a trovare soluzioni multiple di un sistema di equazioni.
Gli stessi autori hanno poi proposto un nuovo approccio, che utilizzava *sotto-sciami* per localizzare soluzioni multiple in problemi di ottimizzazione di funzioni multimodali: NichePSO [3]. Questo algoritmo mantiene uno sciame principale che può generare sotto-sciami ogni volta che una possibile nicchia viene identificata. Lo sciame principale si evolve utilizzando il modello *cognition only*, cioè ogni particella aggiorna la propria velocità basandosi solo sul $\vec{p}_{\mathrm{i}}$.

Parallel Vector-based PSO

Schoeman e Engelbrecht hanno proposto un approccio di niching diverso in [9], implementando il Vector-Based PSO. Questo metodo considera le due componenti del vettore velocità $\vec{v}_{\mathrm{i}}$ nell'equazione di aggiornamento, che puntano rispettivamente alla posizione migliore visitata dalla particella e alla posizione migliore visitata da tutte le particelle nel suo gruppo. Le nicchie vengono quindi identificate sfruttando alcune proprietà di questi vettori. Questa metodologia ha il vantaggio di identificare nicchie

usando operazioni su vettori che sono già presenti nel PSO, e quindi di non richiedere parametri aggiuntivi.

Species-based PSO

Il concetto di diverse specie coesistenti nella stessa popolazione è stata l'ispirazione per il PSO per risolvere problemi di ottimizzazione multimodale. In biologia, una specie può essere descritta come *un gruppo di individui che sono isolati dagli altri in termini di riproduzione*.
Li [12] ha proposto lo "species-based PSO" (SPSO, PSO basato sulle specie), che incorpora l'idea di classificare la popolazione in gruppi di specie. Per *seme* di una specie si intende l'individuo con miglior fitness, o particella nel caso del PSO. La definizione di una specie include anche un parametro r_s, o raggio della specie, che rappresenta la distanza Euclidea dal seme ai confini della specie. Tutte le particelle che si trovano entro tale intorno faranno parte della specie.
L'equazione della velocità per ogni particella utilizzerà come $\vec{p}_g$ la posizione del seme della rispettiva specie.
Uno svantaggio di questo approccio sta nell'impostare r_s, che è dipendente dal problema, e diventa difficile quando la funzione presenta nicchie di dimensioni differenti.

Adaptive Niching PSO (ANPSO)

Questo metodo proposto da Bird e Li [2] rimuove la necessità di impostare il niche radius in base al problema, e i parametri vengono calcolati adattivamente durante l'esecuzione.
Inizialmente l'algoritmo calcola la distanza media r tra ogni particella e la particella più vicina. Questo valore viene utilizzato per determinare la formazione delle nicchie. ANPSO usa un grafo indiretto, g, con le particelle come nodi, per tenere traccia della distanza minima tra particelle durante l'esecuzione dell'algoritmo. Ad ogni iterazione, viene aggiunto a g un arco tra ogni coppia di particelle che sono state più vicine di r tra loro durante gli ultimi due cicli. Le nicchie vengono formate dai sottografi connessi di g, mentre tutte le particelle sconnesse rimangono fuori dalle nicchie.

Per prevenire la convergenza di troppe particelle sullo stesso ottimo, viene impostato un limite di particelle massime in una nicchia.
Il calcolo della distanza tra le particelle rende l'algoritmo più costoso dal punto di vista computazionale.

K-means PSO

In questo metodo, proposto da Passaro e Starita in [16], si utilizza una tecnica di clustering, il *k-means*, per raggruppare le particelle in sotto-sciami basandosi sul loro $pbest_i$. Poi ogni sotto-sciame esegue una ricerca locale sulla propria nicchia, utilizzando una topologia *gbest*.
Il clustering viene effettuato dopo l'inizializzazione casuale delle particelle e poi viene ripetuto ad intervalli regolari durante l'esecuzione dell'algoritmo. Tra due operazioni di clustering, le dinamiche dei sotto-sciami sono quelle del PSO standard.

Capitolo 3

Descrizione del sistema realizzato

3.1 Obiettivi

Da quello che abbiamo visto nelle sezioni precedenti, possiamo considerare la ricerca di strutture rilevanti in un sistema complesso come un problema di ottimizzazione, il cui obiettivo è quello di individuare i sottinsiemi delle variabili del sistema che corrispondono ai valori ottimi della misura scelta per descriverli.
Nel nostro caso utilizziamo come fitness il cluster index CI, che ci permette di identificare gruppi di variabili di un sistema complesso che appartengono a sottinsiemi rilevanti (*Candidate Relevant Subject*, CRS). Il nostro obiettivo sarà quindi individuare le k soluzioni ad indice più elevato.
Le motivazioni di questa scelta sono principalmente due:

- L'utilizzio del CI non richiede conoscenza specifica sulla struttura del sistema e sulle regole che lo governano, e si può quindi utilizzare in numerosi contesti applicativi diversi;
- Il CI viene normalizzato, consentendo quindi il confronto tra CRS di dimensioni differenti.

Il risultato dell'analsisi sarà un gruppo di CRS, che una volta ordinati secondo il loro CI possono costruire la base per una successiva analisi mirata ad individuare i RS e la loro gerarchia.

Per poter affrontare anche problemi con un elevato numero di variabili, non risolvibili in tempi ragionevoli con metodi esaustivi, utilizziamo un metodo euristico, che ci consente di evitare il calcolo del CI per tutti i CRS.
La metaeuristica sviluppata è basata su un algoritmo di Particle Swarm Optimization. Dovendo trovare le prime *k* soluzioni migliori, l'algoritmo PSO standard non sarebbe stato adatto ed è stato quindi scelto l'algoritmo di niching *k-means PSO*, che verrà descritto dettagliatamente nel seguito. Per migliorare ulteriormente le prestazioni dell'algoritmo sono state necessarie delle ottimizzazioni del codice per ridurre i tempi di esecuzione delle fasi computazionalmente più onerose, che sono state quindi parallelizzate su GPU (architettura CUDA).
Riassumendo, gli obiettivi principali del metodo sviluppato sono:

- correttezza delle soluzioni identificate dalla metaeuristica (confrontando i primi k CRS individuati con i risultati della ricerca esaustiva, nei problemi di piccole dimensioni);
- ricerca di più soluzioni (utilizzo di algoritmi di niching per individuare più soluzioni ottime);
- efficienza (ricerca euristica e parallelizzazione CUDA [14]).

3.2 Algoritmo

L'algoritmo realizzato si basa sul *k-means PSO* per individuare i principali massimi locali appartenenti allo spazio di ricerca, con alcune modifiche apportate per adattare la ricerca alle tipologie di sistemi analizzati. I linguaggi di programmazione utilizzati sono C++ e CUDA C.
Nel seguito descriveremo le varie fasi in maniera più dettagliata.

3.3 K-means PSO

L'idea di fondo di questo approccio, presentato in [17], è quella di adattare dinamicamente la struttura organizzativa dello sciame per formare diverse nicchie. Per far ciò,

si applica un algoritmo di clustering standard (*k-means*) per identificare le nicchie, e poi si determinano le particelle vicine di ogni particella considerando quelle appartenenti allo stesso cluster. In questo modo, ogni cluster di particelle tende ad eseguire una ricerca locale nel dominio della funzione, localizzando diversi ottimi.

3.3.1 K-means

Questo algoritmo di clustering funziona partizionando lo spazio di ricerca in base a k semi $\mathbf{m_1}$,...,$\mathbf{m_k}$ inizializzati casualmente. Ogni particella i nello sciame viene associata al cluster $C_{k'}$ corrispondente al seme più vicino alla sua posizione migliore precedente $\vec{p}_\mathrm{i}$:

$$k' = argmin_{1 \leq j \leq k} \|\vec{p}_\mathrm{i} - \vec{m}_\mathrm{i}\|$$

Poi, le posizioni dei semi vengono ricalcolate come media dei $\vec{p}_\mathrm{i}$ delle particelle in ogni cluster:

$$m_j = \frac{1}{N_j} \sum_{p \in C_j} p$$

dove N_j è il numero di particelle in C_j. Il partizionamento viene aggiornato finché non avviene più alcun cambiamento.
La complessità dell'algoritmo k-means è di $O(N \cdot k)$.
I cluster risultanti sono tipicamente compatti e ipersferici, dato che il k-means tende implicitamente a minimizzare l'errore quadratico J:

$$J = \sum_j \sigma_j$$

dove σ_j è la varianza del cluster C_j:

$$\sigma_j^2 = \frac{1}{N_j - 1} \sum_{p \in C_j} \|p - m_j\|^2$$

I problemi principali dell'algoritmo k-means sono i seguenti:

1. È un algoritmo euristico che non garantisce la convergenza a una soluzione ottima ma, soprattutto, dipende fortemente dalla partizione iniziale. Per diminui-

re questa dipendenza, di solito si ripete il k-means un numero r_k di volte, con inizzializzazione casuale dei semi. Alla fine si sceglie la clusterizzazione che minimizza J. Questo approccio aumenta però il costo computazionale.

2. Il numero di cluster k va deciso a priori. Di solito si assegna a k un valore maggiore del numero dei valori ottimi della funzione da analizzare, ma può risultare complicato quando non si ha conoscenza di tale funzione.

Di seguito viene riportato lo pseudocodice dell'algoritmo k-means

Algorithm 2 pseudocodice dell'algoritmo k-means

```
1:  procedure K-MEANS
2:      inizializzare i semi m1,..., mk
3:      repeat
4:          for ogni particella i do
5:              trova il seme più vicino mk'
6:              assegna i al cluster Ck'
7:          end for
8:          for ogni cluster Cj do
9:              ricalcola la nuova media
10:         end for
11:     until non ci sono cambiamenti
12: end procedure
```

3.3.2 Algoritmo k-means PSO

Il k-means PSO usa i cluster di particelle per modificare la topologia dei gruppi, consentendo ad ogni particella di comunicare solo con le particelle nello stesso cluster. Quindi lo sciame si trasforma in una collezione di sottosciami che tendono ad esplorare regioni diverse dello spazio di ricerca. Ogni sotto-sciame effettuerà la ricerca con una topologia *gbest*.

Il clustering viene effettuato dopo l'inizializzazione casuale dello sciame, e poi ripetuto ad intervalli regolari durante l'algoritmo. Tra due operazioni di clustering, i sotto-sciami seguono le loro normali dinamiche.

Dopo avere effettuato il clustering, si eliminano dalle nicchie le particelle in eccesso

nel modo seguente: si calcola il numero medio di particelle per cluster N_{avg} e si rimuovono le particelle peggiori (in base al *pbest*) da ogni cluster finché ogni cluster non contiene esattamente N_{avg} particelle. Queste verranno poi reinizzializzate randomicamente, e si evolveranno con un modello *cognition only* fino alla prossima operazione di clustering, portando all'esploraizione di nuove aree.
L'equazione di aggiornamento della velocità utilizza il *fattore di costrizione*, e la velocità delle particelle assegnate a un cluster viene limitata a $V_{max} = 2\sigma_j$.
Nel seguito vengono riportati gli pseudocodici della procedura per rimuovere le particelle dai cluster e dell'algoritmo completo k-means PSO.

Algorithm 3 pseudocodice della procedura IdentifyNiches

```
1:  procedure IDENTIFYNICHES
2:      Calcola il numero medio di particelle per cluster, N_avg.
3:      Imposta N_u = 0.
4:      for ogni cluster C_j do
5:          if N_j > N_avg then
6:              Rimuovi le N_j – N_avg peggiori particelle da C_j
7:              Somma N_j – N_avg a N_u
8:          end if
9:          adatta la struttura topologica per le particelle in C_j
10:     end for
11:     reinizializza le N_u particelle rimosse
12: end procedure
```

L'utilizzo del K-means PSO richiede l'impostazione di alcuni parametri addizionali rispetto al PSO standard. Il parametro *c* rappresenta il numero di cicli di PSO da eseguire tra due operazioni di clustering. Il parametro *k* rappresenta invece il numero di cluster, e dovrebbe essere impostato a un valore leggermente maggiore al numero di ottimi della funzione da analizzare.

3.3.3 Modifiche apportate

Binarizzazione

Ogni particella rappresenta un CRS codificato da una stringa binaria *P* di lunghezza pari alle dimensioni *D* del problema, in cui ogni elemento corrisponde ad una variabile

Algorithm 4 pseudocodice dell'algoritmo K-means PSO

```
1:  procedure KPSO
2:      inizializzare le particelle con posizione e velocità random
3:      impostare il p⃗_i di ogni particella alla posizione corrente.
4:      calcolare la fitness delle particelle
5:      for t = 1 to T do                          ▷ T = numero di cicli totali
6:          if t mod c = 0 then                                  ▷ ogni c step
7:              esegui la procedura IdentifyNiches
8:          end if
9:          aggiorna la velocità delle particelle          ▷ come nel PSO standard
10:         aggiorna la posizione delle particelle
11:         ricalcola la fitness delle particelle
12:         aggiorna la best position delle particelle e di ogni cluster.
13:     end for
14: end procedure
```

del sistema e assume il valore:

$$P(j) = \begin{cases} 1 & \text{se j fa parte del cluster P} \\ 0 & \text{altrimenti} \end{cases}$$

con $j \in D$.

Nel K-means PSO le particelle si muovono su uno spazio di ricerca reale centrato sull'origine, e quindi sono rappresentate da vettori di numeri reali p_r. Per ottenere la rappresentazione binaria precedentemente descritta si ricorre quindi ad una binarizzazione. Ai fini del calcolo della fitness ogni particella viene trasformata in una stringa binaria nel modo seguente:

$$P(j) = \begin{cases} 1 & \text{se } \ p_r(j) \geq 0 \\ 0 & \text{altrimenti} \end{cases}$$

Nelle altre fasi dell'algoritmo si continuano ad usare i valori reali p_r.

Vettore dei risultati

L'obiettivo del K-means PSO è quello di individuare i diversi punti di massimo (globale e locali) nelle funzioni analizzate. Questo comporta una convergenza delle particelle in pochi punti, mentre la nostra analisi richiede di analizzare i migliori CRS (un massimo locale può avere fitness minore di altri punti non di massimo).
Durante l'esecuzione dell'algoritmo analizziamo quindi le particelle che durante la convergenza a valori ottimi possono visitare punti di interesse, e teniamo traccia degli *n* migliori risultati, che utilizzeremo poi come output.

3.4 Parallelizzazione CUDA

L'obiettivo principale del sistema realizzato è di diminuire significativamente i tempi di esecuzione rispetto alla procedura esaustiva, fornendo comunque soluzioni ottime del problema analizzato.
Per diminuire ulteriormente i tempi di esecuzione abbiamo introdotto la parallelizzazione su GPU. Le parti di codice parallelizzate sono:

- il calcolo della funzione di fitness delle particelle;
- la funzione di aggiornamento di velocità e posizione di ogni particella.

La parallelizzazione è stata realizzata sfruttando l'architettura CUDA.

3.4.1 Elaborazione in parallelo

Data la grande richiesta per grafica 3-D, real time e ad alta definizione, i processori grafici (GPU, Graphic Processor Unit) si sono evoluti in processori multithread e multicore ad elevato parallelismo, con notevole potenza computazionale e banda di memoria (come mostrato in Figura 3.1, [1]). La ragione dietro questa discrepanza nella velocità di esecuzione di operazioni in virgola mobile tra CPU e GPU è dovuta al fatto che la GPU è specializzata in elaborazioni computazionalmente intensive e ad elevato parallelismo (graphic rendering) ed è quindi progettata in maniera tale che più

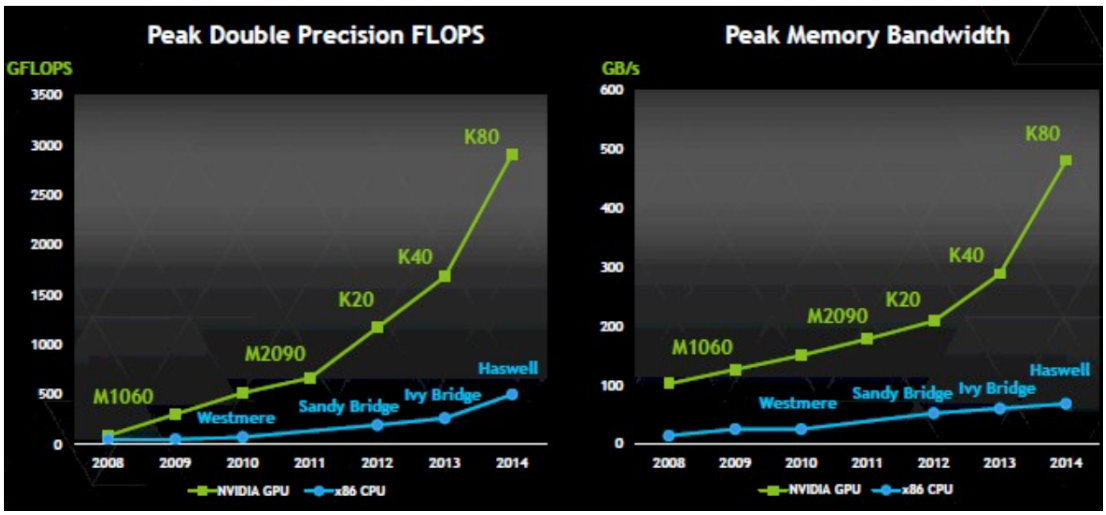

Figura 3.1: Confronto tra potenza computazionale e banda di memoria offerte da CPU e GPU

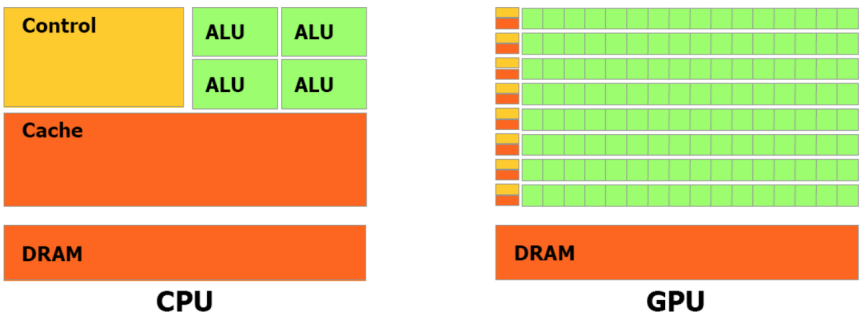

Figura 3.2: Architettura di CPU e GPU a confronto

unità vengano utilizzate per l'elaborazione di dati (Figura 3.2). La GPU è specialmente adatta a gestire problemi che:

- possono essere espressi come elaborazioni di dati massivamente parallele (le stesse operazioni vengono eseguite su più dati in parallelo);
- abbiano un'alta intensità computazionale (rapporto tra operazioni matematiche sui dati e operazioni di lettura/scrittura in memoria);
- non richiedano operazioni di controllo di flusso;
- non necessitino di cache di dimensioni elevate.

L'esecuzione di dati in parallelo mappa i dati su thread ad esecuzione parallela. Molti algoritmi al di fuori del campo dell'image rendering e dell'image processing vengono accelerati dall'esecuzione in parallelo, come ad esempio l'elaborazione dei segnali o la biologia computazionale.

3.4.2 CUDA

CUDA (*Compute Unified Device Architecture*) è un'architettura hardware per l'elaborazione parallela introdotta da NVIDIA nel 2006 insieme ad un'ambiente di sviluppo software che permette di usare il C come linguaggio ad alto livello. L'obiettivo è di sviluppare software in grado di scalare il suo parallelismo al crescere del numero di core, senza richiedere conoscenze aggiuntive a programmatori già familiari con linguaggi di programmazione standard, come ad esempio il C. Alla base ci sono tre astrazioni:

- una gerarchia di gruppi di thread;
- memorie condivise;
- sincronizzazione delle barriere.

Queste astrazioni sono fornite ai programmatori come gruppo minimale di estensioni di linguaggio, e li guidano a dividere il problema in sotto-problemi che possono essere risolti indipendentemente in parallelo. Attualmente ci sono due interfacce che supportano il linguaggio CUDA:

- CUDA C (un insieme minimale di estensioni del linguaggio C);
- CUDA driver API (un'API C di basso livello).

Capitolo 4

Valutazione sperimentale del sistema

Nel seguito viene fornita una descrizione di alcuni dei sistemi analizzati.
Vengono inoltre descritte le prestazioni della metaeuristica realizzata.

4.1 Sistemi analizzati

Come benchmark abbiamo utilizzato due sistemi dinamici: il primo ricavato da una simulazione di un sistema chimico (*Catalytic Reaction System, CatRS*) descritto da 26 variabili; il secondo è un sistema stocastico artificiale che riproduce un comportamento del tipo *Leaders & Followers (LF)* descritto da 28 variabili. Abbiamo inoltre analizzato due sistemi costituiti da dati reali (*Green Community Network, GCN*), descritti rispettivamente da 56 e 137 variabili.

4.1.1 Sistemi simulati

Catalytic Reaction System

Sistema formato da un gruppo di molecole capaci di auto-replicarsi collettivamente. Ci sono due distinte sequenze di reazioni, una lineare a catena (CHAIN) e un gruppo autocatalitico di specie molecolari (ACS) (Figura 4.1). Entrambe compaiono in un chemostato aperto e ben stimolato (CSTR) che riceve in ingresso un flusso costante di molecole e produce un flusso di tutte le specie molecolari proporzionale alla loro

concentrazione.
Le principali entità del modello sono specie molecolari (polimeri) rappresetate da stringhe lineari di lettere A e B, che insieme formano un sistema di reazioni catalitiche composto da sei reazioni di condensazione diverse in cui due specie si legano per creare una specie più lunga. Le reazioni avvengono solo in presenza di catalizzatori specifici, visto che le reazioni spontanee avvengono troppo lentamente per influenzare il comportamento del sistema.

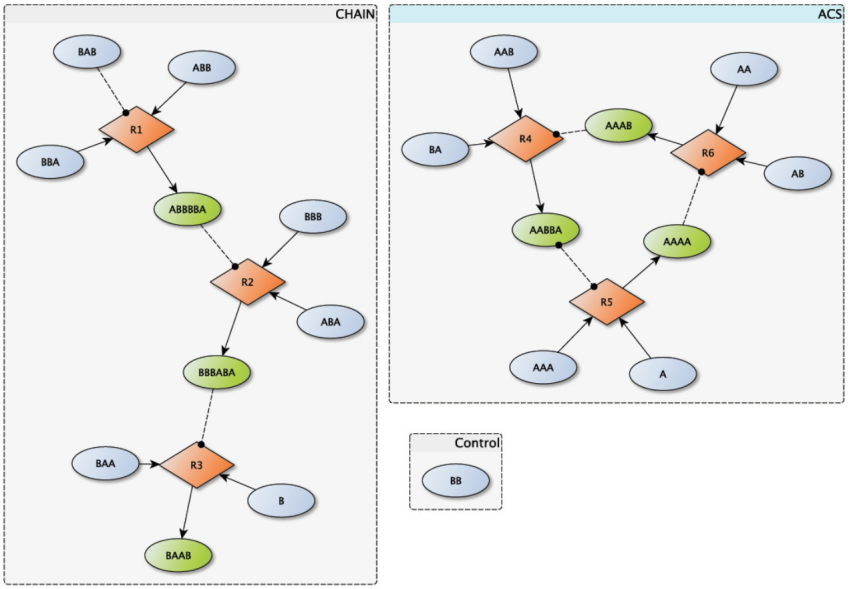

Figura 4.1: Simulazione CatRS

Gli esagoni in Figura 4.1 rappresentano le reazioni: le frecce entranti collegano i substrati alle reazioni, quelle uscenti collegano invece le reazioni ai prodotti. Le linee tratteggiate indicano invece una particolare specie molecolare che assume il ruolo di catalizzatore per una specifica reazione. È stata inoltre aggiunta al sistema un'ulteriore specie BB di controllo, che non partecipa alla reazione.

Leaders & Followers

Questo sistema è una rappresentazione astratta del modello Leaders & Followers. Rappresenta un'astrazione di specifiche relazioni funzionali tra gli elementi di un sistema composto da un vettore di n variabili binarie $X = [x_1, x_2, ..., x_n]$. Queste variabili possono assumere diversi significati, come ad esempio l'opinione favorevole o contraria di n persone ad una determinata proposta.
Il modello consente di generare un insieme di osservazioni indipendenti, basate sulle seguenti regole:

- le variabili sono suddivise in quattro gruppi:
  - G1 = {A0,A1,A2,A3}
  - G2 = {A7,A8,A9}
  - G3 = {A12,A13,A14,A15,A16,A17,A18,A19,A20}
  - G4 = {A22,A23,A24,A25,A26,A27}
- le variabili rimanenti (A4,A5,A6,A10,A11,A21) assumono valore 0 o 1 con la stessa probabilità;
- le variabili (A0,A7,A12,A22,A23) hanno il ruolo di *leader* dei loro rispettivi gruppi; ad ogni step i leader assumono valore 0 rispettivamente con probabilità 0.4, 0.3, 0.3, 0.3 e 0.6, 1 altrimenti;
- le altre variabili (*follower*) copiano oppure negano i valori dei loro leader, ad eccezione dei follower che appartengono al gruppo G4, che calcolano invece la funzione OR oppure AND dei loro due leader.

Sulla base delle regole appena descritte è possibile generare le osservazioni, corrispondenti ciascuna ad un vettore di n elementi generato indipendentemente dagli altri. Il sistema comprende perciò solo gruppi ben definiti che non interagiscono tra loro, dinamicamente separabili rispetto alle altre variabili indipendenti.

4.1.2 Sistemi reali

Green Community Network

I dati sono provenienti da un sistema reale, composto da 137 agenti (persone) coinvolti in un progetto legato alla costruzione di una "comunità verde" (*Green Community Project*) comprendente quattro comuni di montagna. Il progetto ha comportato una serie di riunioni, per un totale di 124 in un arco temporale di più di tre anni. Il valore assunto dalle variabili rappresenta la partecipazione o assenza di queste 137 persone alle riunioni. È stata inoltre valutata un'altra versione del sistema composta da 56 variabili, corrispondenti alle persone che hanno partecipato a più di una riunione.

4.2 Risultati sperimentali

Le prestazioni della metaeuristica vengono misurate secondo due aspetti fondamentali:

- *correttezza dei risultati*: confronto tra i risultati ottenuti dalla metaeuristica con quelli del metodo esaustivo (possibile solo per problemi con un numero di variabili contenuto), confronto con i risultati ottenuti con *HyReSS* (una metaeuristica già sviluppata descritta in [19]), e confronto con i risultati attesi da alcuni esperti del dominio;

- *efficienza*: confronto con le prestazioni della ricerca esaustiva in base al tempo impiegato e al numero di volte in cui la fitness viene calcolata.

Le elaborazioni sono state eseguite su un server Linux con le seguenti caratteristiche: CPU Intel I7 a 1.6 GHz, 6GB di RAM, GPU NVIDIA GeForce GTX 680 (CUDA core: 1536, quantità di memoria: 2048MB).
Di seguito sono riportati i valori dei parametri utilizzati nell'esecuzione della metaeuristica.

Sistema	S	K	x	T	c
CatRS	2000	10	3	501	20
LF	1000	10	3	501	20
GNC (56 var.)	2000	10	3	2001	20

Il significato dei parametri è il seguente:

- S: numero di particelle che compongono lo sciame;
- K: numero di cluster utilizzati;
- x: raggio di estensione di ogni dimensione (da -x a +x);
- T: numero di cicli effettuati dall'algoritmo;
- c: intervallo di esecuzione delle procedure k-means e IdentifyNiches.

Altri parametri utilizzati ma mantenuti sempre costanti sono i coefficienti di accelerazione ϕ_1 e ϕ_2 = 2.05 e il fattore di costrizione χ=0.73.
Un'altro parametro da impostare è il *seme random*, utilizzato per la generazione del sistema omogeneo sui sistemi da analizzare. Bisogna quindi assicurarsi di utilizzare lo stesso seme random sia nel metodo esaustivo che nella metaeuristica in modo da ottenere gli stessi valori per i CI. La tabella seguente riporta i risultati ottenuti:

Sistema	N. Variabili	N. Osservazioni	Tempo[s](ES)	Tempo[s](M)	speed-up
CatRS	26	751	53	6	8.5
LF	28	150	196	2	98
GCN	56	124	n.a.	18	n.a.

Per "N. Osservazioni" si intende il numero di volte in cui si è osservato lo stato del sistema nel tempo, tenendo traccia del valore assunto dalle variabili durante l'osservazione. Un maggior numero di osservazioni può aumentare i tempi di esecuzione.
I tempi sono espressi in secondi, con ES i tempi impiegati dalla ricerca esaustiva e M quelli impiegati dalla metaeuristica.
Nel seguito vengono descritti più in dettaglio i risultati ottenuti nei sistemi analizzati:

- **Catalytic Reaction System**: L'obiettivo del test consiste nel verificare se l'algoritmo realizzato è in grado di identificare correttamente le due strutture presenti (catena e anello). Essendo il sistema composto da 26 variabili, è stato possibile effettuare l'analisi anche attraverso una ricerca esaustiva.
  I test effettuati hanno individuato gruppi con significato chimico.
  Il tempo di esecuzione impiegato dalla metaeuristica è di 6s, con uno speed-up di 8.5 rispetto alla ricerca esaustiva, che impiega circa 53s. Sono state effettuate 100 prove , confrontando i primi 200 CRS trovati dalla ricerca esaustiva. Sono stati trovati risultati identici a quelli prodotti dalla ricerca esaustiva 93 volte, mentre 5 volte sono stati trovati 199/200 CRS e una volta 198/200. La fitness è stata calcolata 1052000 volte.

- **Leaders & Followers**: Il sistema LF considerato è composto da 28 variabili. Il tempo di esecuzione della metaeuristica è di 2s, con uno speed-up pari a 98 rispetto alla ricerca esaustiva, che impiega invece 196s. Sono state effettuate 100 prove , confrontando i primi 257 CRS trovati dalla ricerca esaustiva. Sono stati trovati risultati identici a quelli prodotti dalla ricerca esaustiva 94 volte, mentre 6 volte sono stati trovati 256/257.
  Osservando i risultati si può notare che ciascuno dei CRS individuati contiene elementi appartenenti ad uno solo tra i gruppi G1, G2, G3, G4. Le prime maschere, per esempio, corrispondono a tutte le possibili combinazioni di 7 agenti sui 9 appartenenti al gruppo G3. La fitness è stata calcolata 526000 volte.

- **Green Community Network**: In entrambe le versioni precedentemente descritte il numero di variabili è troppo elevato perché il problema possa essere affrontato con una ricerca esaustiva. Il problema è stato quindi analizzato solo con la metaeuristica, ed i risultati sono stati prima validati sfruttando la conoscenza sul problema fornita da alcuni esperti di dominio, e poi confrontati con i risultati della metaeuristica *HyReSS*.
  La metaeuristica si è rivelata efficace nel caso da 56 variabili, consentendo di individuare le comunità dominanti all'interno del sistema. Il tempo di esecuzione è di 18s, e i primi 50 CSR trovati corrispondono con quelli identificati dalla me-

taeuristica *HyReSS*, che impiega 71 secondi. Su 50 prove, sono stati individuati i primi 50 CRS per 43 volte, mentre 6 volte ne sono stati individuati 49/50 e una volta 48/50. La fitness è stata calcolata 4202000 volte.
Nel caso da 137 variabili, nonostante i tempi di esecuzione rimangano comunque brevi (dai 100 ai 300 secondi in base ai parametri impostati), la metaeuristica non riesce a trovare alcuni dei CRS migliori, trovati invece dalla metaeuristica *HyReSS* che impiega 258 secondi. Le cause ipotizzate sono la presenza di un massimo locale circondato da minimi, che impediscono l'avvicinamento dello sciame a quella zona, o lo spazio di ricerca troppo grande, difficile da esplorare anche con un numero elevato di particelle.

Capitolo 5

Conclusioni e sviluppi futuri

La metaeuristica realizzata è risultata adatta per l'analisi di sistemi complessi, e il suo utilizzo trova giustificazione di impiego per problemi che presentano un numero di variabili elevate. In problemi di queste dimensioni infatti il procedimento esaustivo non è in grado di fonire soluzioni in tempi accettabili, visto che il numero di casi da analizzare cresce esponenzialmente all'aumentare delle variabili: basti pensare, per esempio, che con 50 variabili le possibili combinazioni da analizzare sono 2^{50} (più di un milione di miliardi).
Per problemi di dimensioni N < 100, la metaeuristica è risultata efficace sia riguardo i tempi di esecuzione, sia riguardo la qualità delle soluzioni. Aumentando ulteriormente il numero delle variabili, nonostante i tempi d'esecuzione rimangano accettabili, l'algoritmo non riesce tuttavia a trovare tutti i CRS migliori dei sistemi analizzati.

5.1 Sviluppi futuri

Una serie di miglioramenti possono essere apportati alla metaeuristica, per consentire l'analisi di sistemi con un numero di variabili più elevato.

Parallelizzazione

La parallelizzazione CUDA è stata applicata solo al calcolo della fitness e alla funzione di aggiornamento di velocità e posizione delle particelle. Un possibile miglioramento si potrebbe ottenere parallelizzando altre parti di codice, consentendo quindi di utilizzare un maggior numero di particelle e di eseguire più cicli in modo da aumentare la probabilità di esplorare nuove aree della funzione.

Strategie di ricerca locale

I risultati ottenuti dalla metaeuristica potrebbero essere utilizzati come base per l'esecuzione di specifiche strategie di ricerca locale che sono già comprese in *HyReSS*, con l'obiettivo di esplorare le aree che lo sciame non riesce ad analizzare.

Bibliografia

[1] M. Amoretti. *Slides del corso di Sistemi di Elaborazione: Parallelismo - CUDA*. Università degli studi di Parma, Dipartimento di Ingegneria dell'Informazione.

[2] S. Bird e X. Li. Adaptively choosing niching parameters in a PSO. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 3–9. GECCO, 2006.

[3] R. Brits, A. Engelbrecht e F. van den Bergh. A niching particle swarm optimizer. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 692–696. 2002.

[4] R. Brits, A. Engelbrecht e F. van den Bergh. Solving systems of unconstrained equations using Particle Swarm Optimization. In *Proceedings of the IEEE 2002 Conference on Systems*. 2002.

[5] M. Clerc e J. Kennedy. The particle swarm explosion, stability, and convergence in a multidimensional complex space. In *IEEE Transactions on Evolutionary Computation 6*, pp. 58–73. 2002.

[6] M. Dorigo e T. Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004. ISBN 0262042193.

[7] A. E. Eiben e J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003. ISBN 3540401849.

[8] F. Glover. *Tabu Search*. Kluwer Boston, Inc., prima ed., 1998. ISBN 0792381874.

[9] I.L.Schoemann. *Niching in Particle Swarm Optimization*. University of Pretoria, Faculty of Engineering, Built Environment and Information Technology, 2010.

[10] J. Kennedy e R. Eberhart. Particle swarm optimization, 1995.

[11] S. Kirkpatrick, C. D. Gelatt e M. P. Vecchi. Optimization by simulated annealing. *Science*, vol. 220(4598):pp. 671–680, 1983.

[12] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 105–116. GECCO, 2004.

[13] L.Sani. *Progettazione e realizzazione di una metaeuristica evolutiva per la ricerca di strutture rilevanti nei sistemi complessi*. Università degli studi di Parma, Dipartimento di Ingegneria dell'Informazione, 2015-2016. Tesi di Laurea Magistrale.

[14] NVIDIA Corporation. NVIDIA CUDA C programming guide, 2010. Version 3.2.

[15] K. Parsopoulos, V. Plaginakos, G. Magoulas e M. Vrahatis. Improving particle swarm optimizer by function "stretching". In *Nonconvex Optimization and Applications*, pp. 445–457. Kluwer Academic Publishers, 2001.

[16] A. Passaro e A. Starita. Particle swarm optimization for multimodal functions: A clustering approach. *Journal of Artificial Evolution and Applications*, vol. 2008, 2008.

[17] Passaro, A. *Niching in Particle Swarm Optimization*. Università di Pisa, Dipartimento di Informatica, 2007.

[18] R. Poli, J. Kennedy e T. Blackwell. Particle swarm optimization, an overview. In *Swarm Intell*, pp. 33–57. 2007.

[19] L. Sani, M. Amoretti, E. Vicari, M. Mordonini, R. Pecori, A. Roli, M. Villani, S. Cagnoni e R. Serra. Efficient search of relevant structures in complex systems. In *AI*IA 2016: Advances in Artificial Intelligence*, pp. 35–48. Springer, 2016.

[20] Y. Shi e R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 69–73. IEEE, 1998.

[21] G. Tononi, O. Sporns e G. M. Edelman. A measure for brin complexity: relating segregation and integration in the nervous system. In *Proceedings of the National Academy of Sciences, vol.91*, pp. 5033–5037. 1994.

[22] M. Villani, A. Filisetti, S. Benedettini, A. Roli, D. Lane e R. Serra. The detection of intermediate-level emergent structures and patterns. In Miglino, O. et al., cur., *Advances in Artificial Life, ECAL 2013*, pp. 372–378. The MIT Press, 2013.

[23] M. Villani, A. Roli, A. Filisetti, M. Fiorucci, I. Poli e R. Serra. The search for candidate relevant subsets of variables in complex systems. *Artif Life*, vol. 21(4):pp. 412–431, 2015.