PROGETTO INFORMATICO IN SISTEMI DISTRIBUITI

Algoritmi evolutivi paralleli sviluppati con Jade

Sviluppato da: Enrico Cagnazzo

Sommario

Abstract

Nel corso del seguente lavoro verrà utilizzato il middleware JADE per implementare l'algoritmo evolutivo parallelo poiché, al momento, non è disponibile una tecnologia simile.

Si cercherà di rendere il prodotto finale flessibile in modo tale da poter essere adattato a più problemi di ottimizzazione.

Bisognerà quindi trovare la soluzione più opportuna per implementare il problema stesso. In particolare saranno soggetto di studio l'implementazione delle isole e della migrazione degli abitanti da un'isola all'altra.

Introduzione agli algoritmi evolutivi paralleli

Questa breve sezione serve ad introdurre gli algoritmi evolutivi paralleli (PEA) e la nomenclatura delle parti che lo compongono.

Aspetti generici degli algoritmi evolutivi

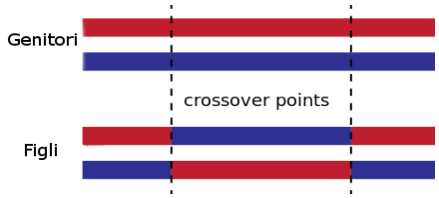
Gli aspetti chiave degli algoritmi evolutivi sono comuni sia alla versione sequenziale che a quella parallela.

Le strutture dati che contengono le possibili soluzioni sono dette *"abitanti"* o *"cromosomi"* e vengono sottoposte ad una serie di operazioni che simulano l'evoluzione genetica. Le unità che compongono i cromosomi sono detti *"geni"*.

L'insieme di tutti gli abitanti è detto *"popolazione"*, in quest'insieme sarà scelto l'abitante migliore che rappresenterà la soluzione al problema.

Come descritto in [3] le fasi dell'algoritmo evolutivo sono le seguenti:

1) **Inizializzazione.** Viene creata (spesso stocasticamente) una popolazione di partenza.
2) **Riproduzione.** Questa fase rappresenta l'incrocio tra due abitanti che produrrà un *"figlio"* che potrebbe far parte della nuova popolazione. Il modo in cui far incrociare i due *"genitori"* è libero, ad esempio in [2], in [3] e in [4] è utilizzato il *"two point crossover"* ovvero vengono creati due figli uguali ai genitori ad eccezione della stringa centrale che viene invertita tra i genitori. L'insieme di tutti i figli è definito come la *"nuova generazione"*.



Rappresentazione grafica del "two points crossover"

3) **Valutazione.** La nuova generazione viene valutata in base ad una funzione di *fitness* e ad ogni abitante viene associato un valore numerico che quantifica la sua qualità.
4) **Selezione.** Vengono selezionati tra tutti gli abitanti attuali quelli che faranno parte della prossima generazione. È importante non scegliere solo i migliori cromosomi per evitare che la popolazione entri in una soluzione di stallo (in un punto di ottimo locale).

Le fasi 2-4 vengono ripetute ciclicamente fin quando non viene raggiunta una condizione di stop (solitamente un numero di iterazioni).

Inoltre spesso è aggiunta tra la riproduzione e la valutazione un'ulteriore fase di mutazione che serve ad accelerare lo sviluppo della popolazione, per questo motivo con probabilità *p* viene modificato un gene all'interno di un cromosoma.

Aspetti relativi agli algoritmi evolutivi paralleli

Alcuni concetti sono invece esclusivi dell'implementazione parallela degli algoritmi evolutivi.

In questo caso gli abitanti vengono suddivisi (non obbligatoriamente in maniera uniforme [3]) in diversi gruppi che comunicano tra di loro. La fase della comunicazione si può inserire (come fatto in [2]) dopo quella della selezione.

Stato dell'arte

Sebbene sia già stato implementato l'algoritmo genetico parallelo con JADE in [1] alcuni aspetti pratici sono stati omessi dalla trattazione.

Infatti in [1] non si fa riferimento a come gli agenti comunicano tra di loro e quale protocollo è stato utilizzato per avere una comunicazione priva di errori.

Come scritto in [3] e realizzato in [2] utilizzare *m* processori può portare ad avere un incremento *super-lineare*, ovvero il tempo di esecuzione risulterà più di *m* volte più veloce dell'algoritmo sequenziale. Questo vantaggio deriva principalmente da tre fattori diversi:

1) **Fattore implementativo.** L'algoritmo eseguito su un solo processore è, in qualche modo, inefficiente. Ad esempio, se l'algoritmo sequenziale usa liste lineari di dati quello parallelo è più veloce perché deve gestire strutture dati più piccole. Inoltre anche le operazioni come la selezione e la riproduzione sono meno complesse quando la popolazione è divisa e i risultati sono gestiti in parallelo.

2) **Fattore numerico.** Di fatto, le euristiche hanno una possibilità non nulla di trovare una soluzione dopo *T* secondi, per qualsiasi $T>0$. Eseguendo l'algoritmo parallelamente su più processi vengono esaminate contemporaneamente più zone dello spazio di ricerca (di solito molto grande) e quindi aumentano le possibilità di trovare la soluzione.

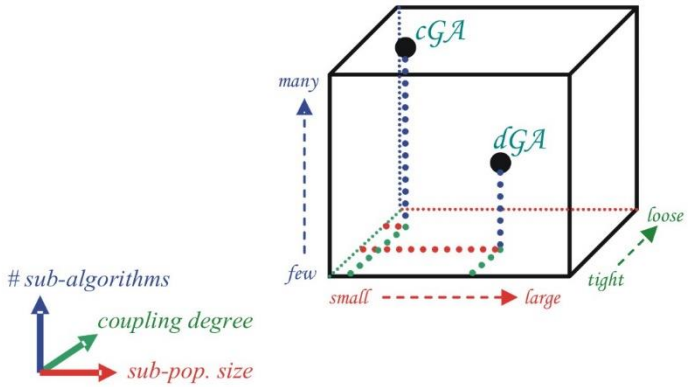
3) **Fattore fisico.** Dividendo la grande popolazione globale in diverse piccole popolazioni ci sono più possibilità che le nuove strutture dati riescano ad essere salvate in memorie più piccole e veloci (come la cache) aumentando la velocità della computazione. Inoltre macchine parallele possono avere risorse maggiori che crescono linearmente con il numero di processori, in cluster eterogenei le risorse sono diverse tra loro e i risultati non sono quantificabili a priori.

Esistono principalmente due sottoclassi dei PEA che si differenziano per il modo in cui la popolazione totale viene suddivisa tra le isole:

1) ***distribuited Evolutionary Algorithm* (dEA).** Una caratteristica principale di questo algoritmo è l'alto numero di abitanti in ciascun gruppo (>>1 [4]), che viene definito *"isola"*, rispetto al numero dei raggruppamenti. Inoltre in questo algoritmo la comunicazione non è molto frequente [2], le diverse isole comunicano tra di loro attraverso un processo di *"migrazione"* in cui degli abitanti si muovono da un'isola all'altra.
   Dei parametri aggiuntivi controllano la frequenza delle migrazioni e il numero di abitanti che vengono spostati da un'isola all'altra [3-4].

2) ***cellular Evolutionary Algorithm* (cEA).** Contrariamente al dEA il numero usuale di abitanti per gruppo, detto *"isolato"*, è molto basso (dell'ordine dell'unità o addirittura un solo abitante) ma in questo caso la comunicazione è molto più frequente. Infatti la riproduzione avviene solo all'interno di una *"vasca"* in cui sono inseriti gli isolati vicini (per questo è necessaria una topologia in N dimensioni degli isolati). Questo modo diverso di esplorare lo spazio di ricerca risulta essere più efficace del metodo sequenziale [3].

Graficamente si dEA e cEA si possono rappresentare nel modo seguente:



Modellazione del problema

La scelta di implementare solo il dEA è dovuta a vari fattori:

- Ogni isola è un'entità indipendente e computa delle informazioni autonomamente, quindi può essere facilmente astratta dagli agenti JADE.
- Questo algoritmo sfrutta in maniera minore la rete dato che sono presenti meno comunicazioni tra le isole, quindi è più efficiente se realizzato in un sistema distribuito.
- Dato il basso numero di isole (e quindi di agenti) realizzare un sistema in cui ogni core dei processori abbia da computare esclusivamente un agente è più economico
- Il dEA risulta più adatto a processori con architettura MIMD [2-3] che sono quelli presenti nella maggior parte dei computer.

Di seguito vengono elencati i modi in cui sono stati implementati i vari aspetti del dEA.

Abitanti

Gli abitanti sono stati implementati definendo un'apposita classe Java (Inhabitant) che definisce tutti i vari aspetti che li regolano (creazione, riproduzione e via dicendo).

All'interno di questa classe è definito implicitamente anche il problema di ottimizzazione che si vuole risolvere, per modificarlo è sufficiente modificare i campi e il metodo *setFitness()* che determina la qualità dell'abitante stesso.

Dato che in questo progetto il focus è sull'implementazione è stato scelto un problema semplice dove, avendo un vettore di 100 booleani, si cerca di massimizzare il numero di elementi *true*. Anche se il problema sembra banale lo spazio di ricerca è formato da più di 10^{30} possibili soluzioni.

La riproduzione degli abitanti avviene tramite il "*two points crossover*", la scelta è ricaduta su questo metodo per la sua semplicità e perché è il metodo maggiormente utilizzato negli articoli riportati in bibliografia.

Infine è stato necessario effettuare l'override del metodo *toString()* per poter inviare l'abitante tramite i messaggi ACL.

Isole

Le isole sono state implementate estendendo gli agenti JADE. Queste, dopo essere create dall'agente *MasterAgent*, iniziano a reiterare la sequenza Riproduzione-Mutazione-Selezione ed eventualmente l'invio e la ricezione dei migranti.

Ogni passo della sequenza è implementato in differenti *OneShotBehavior* in modo tale da separare operazioni concettualmente diverse ed indipendenti.

Nelle isole è implementata anche la selezione della successiva popolazione dell'isola che avviene tramite il metodo della roulette (utilizzato in [1,4]).

In questo metodo tutti gli abitanti e tutti i figli vengono raggruppati in un'unica lista e successivamente vengono pescati maniera casuale con probabilità pari al loro valore di fitness normalizzato rispetto al totale. Ogni volta che un abitante è stato scelto questo viene rimosso e vengono aggiornati i valori usati per il pescaggi successivi.

Migrazione

La migrazione avviene con la modalità *"unidirectional ring"* ovvero ogni migrante viene inviato all'isola topologicamente successiva.

Nel codice la procedura di migrazione è stata separata in due fasi: l'invio e la ricezione.

Nel behaviour relativo all'invio l'agente *Island* si blocca in attesa di ricevere dal *MasterAgent* le informazioni sull'isola seguente per poter inviare N migranti scelti in maniera casuale.

Anche quando si devono ricevere i migranti l'agente si blocca fin quando non riceve il messaggio con le informazioni sul migrante. Successivamente questo prenderà il posto, se è migliore, del peggiore abitante presente sull'isola, altrimenti verrà scartato.

Una possibile alternativa a questa soluzione è uno scambio di messaggi tra l'isola di partenza P e quella di arrivo A:

1) P invia il valore di fitness del migrante;
2) Se il valore è migliore rispetto a quello del peggiore abitante di A questa invia un messaggio di accettazione altrimenti risponde con un rifiuto;
3) P invia o meno il migrante in base alla risposta di A.

Nel caso di strutture dati molto pesanti questa soluzione è più efficiente perché il numero maggiore di messaggi viene bilanciato dal minore invio di dati.

In questo caso però, dato che la struttura non è particolarmente grande rispetto alle dimensioni minime di un messaggio si è preferito optare per l'invio diretto dei migranti.

MasterAgent

È stato necessario implementare anche un agente che gestisse tutto il processo, quindi il MasterAgent è stato aggiunto agli altri elementi del dEA. Questo agente serve sia creare le isole che ad elaborare il risultato finale del programma.

Può quindi essere definito come un'interfaccia di I/O in quanto, al momento della sua creazione, riceve in ingresso degli argomenti che saranno gli iperparametri del problema (numero di isole, di abitanti per isola, la frequenza delle migrazioni, il numero dei migranti per migrazione e il numero di iterazioni totali) e infine stampa a video il risultato.

Un’altra sua importante funzione è l’invio ad ogni isola dei riferimenti di quella topologicamente successivamente alla quale verranno inviati i migranti.

Test e analisi dei risultati

Il computer su cui è stato eseguito il programma è un portatile Lenovo con processore Intel i7-6500U da 2.5Ghz e 8GB di RAM su cui è installato Windows 10 Home nella versione da 64 bit. L'elaborato è stato eseguito tramite la versione 4.4.0 di JADE e il Java Runtime Environment 8.

Il programma è stato provato con 4 configurazioni diverse e ripetuto 5 volte per configurazione. In questo modo i valori ottenuti sono più significativi poiché l'algoritmo evolutivo parallelo ha una natura stocastica. Di seguito sono riportate le medie dei dati ottenuti con i relativi parametri di ingresso[1]:

N. isole	N. abitanti	Tempo	Risultato	Speedup	Efficienza
6	200	5031,2	70,6	6,513	1,086
4	300	5254,2	69,6	6,237	1,559
2	600	9875,6	69,4	3,318	1,659
1	1200	32770,4	69	-	-

La formula utilizzata per calcolare lo speedup è quella descritta in [2] mentre l'efficienza è il rapporto tra speedup e numero di isole.

Quindi è l'efficienza che dimostra l'incremento superlineare delle prestazioni rispetto all'algoritmo evolutivo sequenziale (quello con una sola isola): all'aumentare di queste il tempo diminuisce più che linearmente.

Mentre il tempo diminuisce superlinearmente il risultato finale è quasi costante, quindi per avere un guadagno nelle prestazioni migliori bisogna modificare gli iperparametri per aumentando la quantità di calcolo e di conseguenza il tempo di esecuzione.

Si può notare anche che con 6 isole l'efficienza diminuisce: questo è dovuto al fatto che il processore del computer ha solo 4 core quindi l'hardware del computer non riesce a supportare un completo parallelismo tra i vari agenti.

[1] I parametri omessi sono costanti: numero iterazioni 1000, frequenza migrazione 10, numero migranti 30

Conclusioni e sviluppi futuri

Dal test effettuato si nota che l'algoritmo evolutivo parallelo porta notevoli benefici all'esecuzione, soprattutto se eseguito su un hardware proporzionato al numero di processi paralleli che vengono richiesti dalle configurazioni.

Un punto di forza di questo lavoro è la flessibilità del codice perché è sufficiente modificare la classe Inhabitant per adattare il programma a nuovi problemi di ottimizzazione.

Si può migliorare il lavoro effettuato permettendo la creazione di più isole su diversi computer: in questo modo si aumenterebbe l'efficienza dell'algoritmo per le configurazioni con un numero maggiore di isole.

Questo progetto quindi è utile per chi vuole avere una soluzione flessibile per risolvere problemi di ottimizzazione di vario tipo in tempi ridotti anche senza usare sistemi computazionali estremamente potenti.

Bibliografia

[1] L. Asadzadeh and K. Zamanifar, "An agent-based parallel approach for the job shop scheduling problem with genetic algorithms", Mathematical and Computer Modelling, 52, 2010, pp. 1957-1965.

[2] E. Alba, “Parallel evolutionary algorithms can achieve superlinear performance,” Inform. Process. Lett., vol. 82, no. 1, pp. 7–13, Apr. 2002.

[3] E. Alba and M. Tomassini, “Parallelism and evolutionary algorithms,” IEEE Transactions on Evolutionary Computation., vol. 6, no. 5, pp. 443–462, 2002.

[4] E. Alba, C. Cotta, F. Chicano, and A. J. Nebro, “Parallel evolutionary algorithms in telecommunications: Two case studies,” in Proc. 8th Argentinian Computer Science Congr., Buenos Aires, Argentina, 2002.

Istruzioni per l'uso

Per eseguire correttamente il programma bisogna creare in un container JADE un agente di tipo *MasterAgent* con i seguenti argomenti:

1) Numero delle isole
2) Numero degli abitanti per isola
3) Frequenza migrazione (ogni quante iterazioni avviene la migrazione)
4) Numero di migranti per migrazione
5) Numero di iterazioni totale

Non bisogna creare direttamente gli agenti di tipo *Island* perché questi saranno opportunamente creati dal *MasterAgent*.