Algoritmo Tree CSP Solver: applicazione al problema di colorazione di una mappa

Denny Sbanchi

5 aprile 2020

1 Introduzione

Il problema proposto consiste nella colorazione di una mappa costituita da n nodi (rappresentanti le nazioni di una mappa geografica) i quali non devono avere lo stesso colore di uno dei loro vicini.
Veniva richiesta l'implementazione dell'algoritmo Tree-CSP-Solver per l'assegnamento dei valori ai singoli nodi, apportando le dovute modifiche per codificare al suo interno i vincoli imposti dalla natura del problema.

La creazione di un'istanza del problema e la sua risoluzione possono essere schematizzate nelle seguenti fasi:

- Generazione casuale degli n nodi a partire da un quadrato di dimensione unitaria (*unit square*)
- Creazione dei collegamenti fra i nodi osservando opportuni criteri imposti dal testo del problema
- Rilassamento della rete di vincoli creata nello step precedente con un suo albero di copertura (*spanning tree*)
- Implementazione dell'algoritmo *Tree-CSP-Solver* per la risoluzione del problema a cui viene dato in input lo *spanning tree* precedentemente calcolato oltre alla lista dei nodi della rete

2 Ambiente di sviluppo e specifiche hardware

Il progetto assegnato è stato interamente sviluppato in Java in ambiente di sviluppo Eclipse (Versione Neon.2 e release (4.6.2)) installato su sistema operativo *Windows 10 Home* 64 bit.
I test sono stati effettuati con un laptop Lenovo Y50 con le seguenti specifiche hardware:

- CPU: Intel Core i7 2.60GHz (quad core)
- RAM: 8,00 GB di tipo DDR3 da 789 MHz

3 Classi del programma

Le principali classi in cui è suddiviso il programma sono le seguenti:

- Node: rappresenta un singolo nodo della mappa
- Colour: classe dei colori appartenenti al dominio
- Connection: rappresenta la connessione presente tra due nodi
- LinesChecker: classe che verifica la possibilità di tracciare un segmento di collegamento tra due nodi, tramite il metodo *calculateIntesection* a cui vengono passate due coppie di nodi su cui effettuare i controlli

- Map: genera gli n nodi della rete distinti tra loro, dopodiché per ogni nodo:
  - Tenta di collegarlo con il nodo a lui più vicino, a patto che non siano già collegati o non ci abbia già provato
  - Controlla che il nuovo collegamento non si sovrapponga a nessuno di quelli già effettuati
- SpanningTreeGenerator: a partire dalla lista di connessioni stabilite crea un albero di copertura della rete generata precedentemente
- `Tree_CSP_Solver`: prende in input i nodi della mappa ed un suo albero ricoprente. Ordina topologicamente i nodi in modo che ognuno di essi possa essere seguito dai figli ma non da uno dei loro genitori. Sfruttando l'ordinamento dei nodi precedentemente costruito, inizia ad assegnare i colori prima ai nodi con il numero di ordinamento maggiore (le foglie) per poi risalire fino alla radice. La strategia utilizzata per scegliere il colore di un nodo prevede di controllare quale, tra quelli disponibili, permetterebbe al padre di preservare più valori possibili nel proprio dominio (euristica ***least-costraining-value***).

4 Test effettuati e tabulazione dei risultati

Il programma realizzato è stato testato con un numero variabile n di nodi e con una quantità di colori fissa (4 colori) che permettesse di trovare sempre un valido assegnamento per tutti i nodi in modo che ognuno avesse un colore diverso da tutti gli altri a cui è direttamente connesso. Questo risultato è assicurato dal teorema matematico conosciuto come ***Teorema dei 4 colori***. Di seguito sono riportati alcuni degli output ottenuti al variare del numero dei nodi inserito dall'utente.

```
Inserisci un numero di nodi positivo:
1
E' stato possibile trovare un valido assegnamento di colori per ogni nodo.
Tempo impiegato esecuzione dell'algoritmo Tree-CSP-solver: 0 millisecondi.
Tempo impiegato esecuzione totale: 0 millisecondi.
```

Figura 1: Test con un solo nodo

```
Inserisci un numero di nodi positivo:
100
E' stato possibile trovare un valido assegnamento di colori per ogni nodo.
Tempo impiegato esecuzione dell'algoritmo Tree-CSP-solver: 0 millisecondi.
Tempo impiegato esecuzione totale: 84 millisecondi.
```

Figura 2: Test con 100 nodi

```
Inserisci un numero di nodi positivo:
1000
E' stato possibile trovare un valido assegnamento di colori per ogni nodo.
Tempo impiegato esecuzione dell'algoritmo Tree-CSP-solver: 4 millisecondi.
Tempo impiegato esecuzione totale: 55726 millisecondi.
```

Figura 3: Test 1000 nodi

Sotto è mostrata una tabella che riassume i risultati di tutti i test effettuati, dove la colonna **Tempo Tree-solver** indica il tempo richiesto dall'algoritmo *Tree-CSP-solver* per essere eseguito una volta creata e modificata la rete dei vincoli a dovere, mentre **Tempo totale** indica il tempo di esecuzione dell'intero programma.

n	Tempo Tree-solver	Tempo totale
1	0 ms	0 ms
10	0 ms	0 ms
100	0 ms	84 ms
500	0 ms	8780 ms $\approx$ 8.78 s
1000	4 ms	55726 ms $\approx$ 55.73 s
1500	4 ms	227566 ms $\approx$ 3.80 min
2000	8 ms	567924 ms $\approx$ 9.47 min
3000	20 ms	1930618 ms $\approx$ 32.18 min

Si fa notare che per rilevare il tempo di esecuzione è stata utilizzata la funzione System.currentTimeMillis() della classe Java.lang.System, i cui risultati possono variare nell'ordine dei 10 ms a seconda del sistema operativo su cui viene effettuato il test.

5 Conclusioni

Come ci si aspettava, nonostante il numero di nodi dato in input al programma aumenti, con 4 colori è sempre stato possibile trovare un assegnamento di colore valido per ogni nodo.
Inoltre osservando i dati ricavati dai test eseguiti si nota subito che la maggior parte (quasi la totalità) del tempo di esecuzione del programma viene spesa per la creazione della rete di collegamenti e le operazioni su di essa al fine di prepararla per essere passata in input all'algoritmo Tree-CSP-Solver.
Il tempo effettivo per l'ordinamento topologico della rete ricevuta dall'algoritmo e quello necessario per l'assegnamento dei valori del dominio si rivela dell'ordine di pochi ms nei test eseguiti, molto minore rispetto al tempo di esecuzione dell'intero programma che, nel caso di $n = 3000$, arriva a sforare i 30 min.