# Cutset Conditioning e Map Coloring

Daniele Bellini

Giugno 2020

## 1 Introduzione

Lo scopo di questo progetto consiste nel realizzare un software che sia in grado di decomporre un *constraint graph* (grafo con vincoli) in un albero, mantenendone i vincoli tramite l'algoritmo di **Cutset Conditioning**. Dopodiché l'obiettivo successivo è quello di risolvere il problema di *Map Coloring*, che si occupa di colorare una mappa con regioni di spazio adiacenti aventi colori diversi. La mappa che è stata presa in esame è rappresentata inizialmente da un grafo e, successivamente, da un albero con vincoli. Per poter risolvere tale problema si è implementato l'algoritmo **Tree Csp Solver**.

## 2 Procedimento Teorico

Prima di illustrare operativamente quali siano stati gli esperimenti, si descrivono, di seguito, i principali passaggi teorici che hanno contribuito allo svolgimento dell'intero progetto.
Il materiale utilizzato per poter eseguire gli esperimenti è stato fondamentalmente l'uso di un *constraint graph* noto, con il seguente vincolo: ogni nodo ha un colore diverso rispetto ai nodi adiacenti.
Una delle premesse cardine sulla quale ci si è basati riguarda la probabilità di avere archi all'interno del grafo, fissata a priori a **0.5**.

### 2.1 Generazione grafo

La prima parte consiste nella generazione del grafo rappresentante la mappa, che sarà colorata in seguito. Per la generazione del grafo si è costruito tre classi:

- Classe **Node**, con caratteristiche relative a *colore*, espresso numericamente, *dominio* (dei possibili colori applicabili al nodo) e *puntatore* a nodi adiacenti;

- Classe **Adjacency Matrix**, A = $(a_{ij})$ = |V| x |V|, dove $(a_{ij}) = 1$ se $(i, j) \in E$, mentre è pari a 0 altrimenti;
- Classe **Graph**, costruita a partire dalla matrice di adiacenza e dai vari nodi.

Una volta generato il grafo si provvede a inserire i vincoli per ogni nodo, sfruttando la conoscenza della matrice di adiacenza.

## 2.2 Trasformazione in Albero

Dopo aver generato il grafo ed inserito i vincoli, la seconda parte del progetto consiste nel trasformare tale grafo in un albero mantenendo i vincoli per ciascun nodo. Come detto precedentemente, l'algoritmo utilizzato è stato quello di *Cutset Conditioning*, rinominato *Cutset Decomposition* all'interno del progetto, atraverso il quale viene cercato un eventuale ciclo di nodi nel grafo. In caso di esito positivo, esso viene salvato per poi assegnare un valore numerico che rappresentasse il colore ai nodi facenti parte del ciclo trovato. Per rendere piu veloci le operazioni successive si è deciso di eliminare ogni possibile riferimento dei nodi del ciclo dalla matrice di adiacenza in modo da non doverli riconsiderare successivamente.

## 2.3 Map Coloring

Come ultima parte del progetto si è sviluppato l'algoritmo di *Tree Csp Solver*, servendosi dei seguenti algoritmi:

- **Revise**, nel quale si sono eseguiti i *crossout* dei domini per ogni nodo adiacente in modo da poter eliminare da subito i possibili valori;
- **TopSort**, funzione utilizzata per ordinare l'albero garantendo un ordinamento nella scelta dei valori da assegnare;
- **DAC**, utilizzata per mantenere gli archi tra i nodi consistenti e quindi il rispetto del vincolo;
- **Assignment**, utilizzata per i vari assegnamenti, è una funzione che garantisce il corretto funzionamento dell'algoritmo. Ogni nodo ispeziona i nodi adiacenti e se il colore appartiene al dominio del nodo in esame, questo viene eliminato dai possibili valori a scelta.

# Test

Il codice per svolgere l'esperimento è stato sviluppato con Python 2.7. Inoltre codice e file README sono presenti nella seguente repository di Github.

## 2.4 Dati utili allo svolgimento

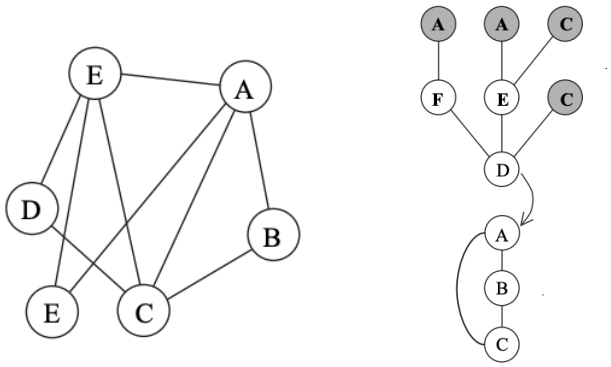
Al fine di trovare il miglior compromesso fra tempo di esecuzione ed ottimalità dei risultati sono stati usati i seguenti valori:

- Dimensione del grafo: n = 100;
- Probabilità di avere archi nel grafo: p = 0.5;
- Dimensione dominio generale dei colori: dim $\in [5, 100]$;
- Numero di colori possibili: c = 1000.

Durante i vari *run* effettuati, i valori sopra elencati sono sempre risultati sufficienti per ottenere la soluzione del problema.

# 3 Risultati dell'esperimento

Di seguito sono presentati un possibile grafo iniziale (Figure 1.a) ed il suo corrispondente albero, ricavato dall'algoritmo (Figure 1.b).




(a) *Grafo prima del Cutset conditioning*

(b) *Albero dopo aver eliminato il ciclo*

Figure 1: Esempio di applicazione dell'algoritmo di *Cutset conditioning*

Come si può notare, in Figure 1.b è rappresentato un grafo al quale sono stati eliminati i nodi facenti parte del ciclo, permettendo la generazione dell'albero corrispondente.

Per rendere comprensibile il progetto si è deciso di illustrare un esempio, con un numero di nodi pari a 10.
Data la matrice di adiacenza,

```
Matrice di adiacenza:
[[1 0 0 0 0 0 0 1 1 1]
 [0 1 0 0 0 0 0 1 1 0]
 [0 0 1 0 0 0 1 0 1 0]
 [0 0 0 1 1 1 0 0 0 0]
 [0 0 0 1 1 1 1 1 1 1]
 [0 0 0 1 1 1 0 1 0 0]
 [0 0 1 0 1 0 1 1 1 1]
 [1 1 0 0 1 1 1 1 0 0]
 [1 1 1 0 1 0 1 0 1 0]
 [1 0 0 0 1 0 1 0 0 1]]
```

è stato calcolato il ciclo che risulta essere definito come, $Cycle = [9, 7, 8, 0]$.
Quindi dopo aver assegnato un colore ai quattro nodi del ciclo, si è proceduto ad assegnare un colore ai restanti nodi del grafo.
Prima di ciò, si è, però, calcolato l'ordine di esplorazione del grafo,
$Order = [0, 9, 8, 7, 3, 1, 6, 2, 4, 5]$.
Il risultato è espresso di seguito con l'aiuto di una tabella:

| **Nodo** | **Colore** | **Nodo** | **Colore** |
|---|---|---|---|
| 0 | 57 | 1 | 51 |
| 9 | 86 | 6 | 24 |
| 8 | 56 | 2 | 99 |
| 7 | 69 | 4 | 57 |
| 3 | 41 | 5 | 93 |

Si può notare come tutti i nodi, ovvero ogni regione della mappa, abbiano un valore numerico differente, ad eccezione dei nodi *0* e *4*. Questa soluzione è possibile ed accettabile perché i due nodi non risultano adiacenti, come si puo notare nella matrice di adiacenza, $AdjacencyMatrix[0][4] = 0$.

# Conclusioni

La conclusione che è possibile trarre dall'esperimento, relativamente al grafo casuale generato, è che l'algoritmo di *Cutset Conditioning* si comporta in modo abbastanza ottimo: riesce, infatti, a calcolare un albero con caratteristiche simili a quelle del grafo iniziale, ovvero vincoli, nodi adiacenti e colore.
Allo stesso tempo l'algoritmo di *Tree Csp Solver* riesce ad analizzare la struttura dell'albero che viene mandato in input e a generare una soluzione che rispetta tutti i vincoli imposti inizialmente. Tale soluzione non è in generale fissa, poiché, dato un albero, la scelta dei colori all'interno del dominio viene eseguita in modo casuale.