Cutset Conditioning per CSP: implementazione e risultati

Samuel Bruno

Sommario

La relazione descrive l'implementazione di un solver per problemi di soddisfacimento di vincoli (CSP) basato sulla tecnica di *cutset conditioning*. All'interno è mostrato il metodo adottato, la struttura dei file sorgente, il flusso d'esecuzione del programma e i risultati sperimentali su istanze di *map coloring* e *cryptoarithmetic*. Vengono inoltre fornite indicazioni per riprodurre gli esperimenti.

1 Introduzione

Cutset conditioning è una tecnica che sfrutta la struttura del grafo dei vincoli: si sceglie un sottoinsieme di variabili, detto cutset, e si esplorano tutte le possibili assegnazioni per esse. Una volta fissato il cutset, le variabili rimanenti (dette residui) formano un grafo senza cicli, che può quindi essere risolto con algoritmi più semplici ed efficienti. Nel progetto, in particolare, il grafo residuo viene risolto tramite l'algoritmo *TREE-CSP-SOLVER* basato su *directional arc consistency (DAC)*, che garantisce una risoluzione polinomiale.

Per ogni assegnazione del cutset si costruisce e si risolve il problema residuo; se il residuo ammette una soluzione, questa viene combinata con i valori assegnati al cutset per ottenere una soluzione completa del CSP.

Ad esempio, nella mappa dell'Australia, la variabile South Australia (SA) appartiene a tutti i cicli. Se la si fissa, le altre regioni rimaste formano una struttura aciclica che può essere risolta tramite TREE-CSP-SOLVER.

Questa strategia è discussa nella letteratura di riferimento (Russell & Norvig, Artificial Intelligence: A Modern Approach, §6.5.1; Dechter, Tractable Structures for Constraint Satisfaction Problems, §3.1).

2 Metodo utilizzato

In questa sezione vengono descritti i punti essenziali del metodo implementato:

1. **Rappresentazione CSP:** ogni istanza è rappresentata da:
   - `variables`: lista di nomi delle variabili;
   - `domains`: lista del dominio dei valori possibili;
   - `constraints`: lista di coppie $(\text{constraint\_variables}, \text{constraint\_function})$ dove la prima è una tupla di coppie di variabili (per le mappe ad esempio, sono le coppie di regioni adiacenti) mentre constraint_function è una funzione booleana che, dati i valori assegnati a quelle variabili, nello stesso ordine, restituisce `True` se il vincolo è soddisfatto.
2. **Selezione del cutset:** si costruisce il grafo di adiacenza considerando solo i vincoli binari; si applica un'euristica greedy *min-fill*, che rimuove iterativamente la variabile che introduce il minor numero di archi aggiuntivi (fill-in) finché non rimangono cicli (cycle-cutset).

3. **Combinazioni e verifica parziale:** si generano tutte le possibili combinazioni di valori per le variabili del cutset. Ogni assegnazione parziale viene controllata per verificare che rispetti i vincoli interni al cutset (ad esempio, che due regioni adiacenti non abbiano lo stesso colore). Se i vincoli non sono rispettati, quell'assegnazione viene scartata.

4. **Costruzione del problema residuo:** una volta fissato il cutset, il problema rimanente (che coinvolge tutte le altre variabili) diventa privo di cicli. A questo punto:

   - si mantengono i vincoli che riguardano solo le variabili residue;
   - i vincoli binari che collegano una variabile del cutset a una variabile residua vengono trasformati in vincoli unari sulla variabile residua, sfruttando i valori fissati del cutset;
   - in presenza di vincoli n-ari, invece di inserire tutte le variabili nel cutset, se ne sceglie una sola (tipicamente quella con grado massimo nel grafo) per evitare esplosioni combinatorie;
   - eventuali vincoli più complessi che coinvolgono sia cutset che residuo verranno verificati solo quando l'assegnazione completa sarà disponibile.

5. **Risoluzione del residuo:** il problema residuo, ora ad albero, viene risolto con l'algoritmo *TREE-CSP-SOLVER*, che prevede una passata bottom-up (arc-consistency tra parent e child) e una successiva passata top-down (assegnamento senza backtracking). Se si trova una soluzione per il residuo, questa viene combinata con l'assegnazione del cutset per ottenere la soluzione completa del CSP. In caso contrario, si passa alla successiva combinazione di valori per il cutset.

3 Descrizione dei file sorgente e del loro scopo

Il progetto è organizzato in più file Python che realizzano l'implementazione della tecnica di cutset conditioning:

csp.py

Definisce la classe `CSP` che contiene variabili, domini e vincoli. Fornisce il metodo `is_consistent`, che serve a controllare passo dopo passo se una certa assegnazione parziale rispetta i vincoli già definiti. In questo modo, durante la ricerca di una soluzione, si scartano subito le strade che non possono portare a un risultato valido.

tree_solver.py

Implementa l'algoritmo `TREE-CSP-SOLVER` (Russell & Norvig, §5.5), che risolve CSP ad albero in tempo lineare rispetto al numero di variabili. L'algoritmo applica una fase di propagazione di vincoli (directional arc consistency) e successivamente assegna i valori ai nodi top-down senza necessità di backtracking.

cutset.py

Qui si trovano le parti principali che implementano la tecnica del cutset conditioning. Nel dettaglio:

- una funzione costruisce il grafo di adiacenza delle variabili a partire dai vincoli binari, e utilizza la procedura greedy *min-fill* integrata con leaf pruning per individuare un insieme di variabili da rimuovere (il cutset) in modo da eliminare i cicli;

- una funzione si occupa di trasformare vincoli binari che collegano variabili del cutset e variabili residue in vincoli unari sulle variabili residue, fissando i valori già assegnati nel cutset;
- nel caso di vincoli n-ari, il cutset include solo una variabile scelta in maniera euristica (grado massimo) per controllare la complessità;
- infine, la funzione `solve_with_cutset` si occupa di: provare tutte le possibili assegnazioni per il cutset, di verificare quali siano ammissibili, di costruire di volta in volta il problema residuo e di risolverlo con `tree_solve`. Se il residuo ha una soluzione, questa viene combinata con i valori del cutset per ottenere una soluzione completa del CSP.

mapcolor.py e cryptarithmetic.py

Questi due file hanno lo scopo di generare istanze concrete del problema da dare in input al solver.

- `mapcolor.py` produce problemi di colorazione di mappe: la mappa dell'Australia (7 regioni e 3 colori), una mappa semplificata dell'Europa (11 regioni e 4 colori) e una piccola versione degli Stati Uniti (5 regioni e 3 colori). I vincoli stabiliscono che due regioni confinanti non possano avere lo stesso colore.
- `cryptarithmetic.py` produce puzzle di criptoaritmetica, ovvero `T+T=EE`, `SEND+MORE=MONEY` e `TWO+TWO+TWO=SIX`. In questo caso le lettere rappresentano cifre e i vincoli impongono sia che tutte le lettere abbiano valori diversi, sia che l'equazione sia rispettata.

main.py

E' lo script principale che gestisce l'esecuzione del programma. Per ogni istanza costruita (sia di mappe che di criptoaritmetica), lo script crea il corrispondente oggetto CSP, lo risolve tramite `solve_with_cutset` e salva l'intero risultato in un file di log all'interno della cartella `logs_of_istances`. I log sono utili per seguire i passaggi compiuti dal solver e per verificare i risultati ottenuti, anzichè farli stampare ad ogni iterazione in console.

4 Istanze testate e risultati

Elenco delle istanze eseguite e risultato osservato con la configurazione adottata:

Categoria	Istanza	Risultato
Map coloring	Australia (7 regioni, 3 colori)	Soluzione trovata
Map coloring	Europa semplificata (11 regioni, 4 colori)	Soluzione trovata
Map coloring	USA semplificata (5 regioni, 3 colori)	Soluzione trovata
Crypto	T + T = EE	Nessuna soluzione (problema mal posto)
Crypto	SEND + MORE = MONEY	Soluzione trovata
Crypto	TWO + TWO + TWO = SIX	Soluzione trovata

5 Flusso di esecuzione del programma sulla mappa dell'Australia

In questa sezione viene descritto in modo dettagliato il flusso del solver applicato all'istanza `australia`. Le variabili sono `['WA','NT','SA','Q','NSW','V','T']`, ciascuna con dominio $\{R, G, B\}$, e i vincoli binari impongono che regioni confinanti abbiano colori diversi.

1. Costruzione del grafo dei vincoli Il modulo `cutset.py` costruisce il grafo di adiacenza a partire dai vincoli binari. Ogni nodo rappresenta una regione, e ogni arco un vincolo di diversità di colore tra due regioni adiacenti. La lista di adiacenza iniziale è:

- `WA: {NT, SA}` (grado 2)
- `NT: {WA, SA, Q}` (grado 3)
- `SA: {WA, NT, Q, NSW, V}` (grado 5)
- `Q: {NT, SA, NSW}` (grado 3)
- `NSW:{SA, Q, V}` (grado 3)
- `V: {SA, NSW}` (grado 2)
- `T: {}` (isolata, grado 0)

2. Individuazione del cutset con l'algoritmo Min-Fill Per rendere il grafo aciclico e poter applicare un risolutore ad albero, il programma utilizza la funzione `find_cycle_cutset_min_fill`. L'algoritmo seleziona iterativamente i nodi da rimuovere in modo da ridurre al minimo il numero di nuovi archi (fill-in) necessari tra i loro vicini. Questo processo elimina progressivamente i cicli presenti nella rete dei vincoli.

Nel caso della mappa australiana, i nodi rimossi sono:

$$\text{cutset (min-fill)} = [\texttt{WA},\ \texttt{NT},\ \texttt{Q},\ \texttt{SA}]$$

La rimozione di questi nodi rende il grafo residuo aciclico, composto da $\{\texttt{NSW}, \texttt{V}, \texttt{T}\}$.

3. Esplorazione e verifica delle assegnazioni del cutset Il solver genera tutte le possibili assegnazioni per le variabili del cutset. Ogni combinazione viene verificata rispetto ai vincoli interni tra le variabili del cutset stesso; quelle non consistenti vengono immediatamente scartate. Ad esempio:

- `{WA='R', NT='R', Q='R', SA='R'}` viene scartata poiché viola il vincolo `WA != NT`;
- `{WA='R', NT='G', Q='R', SA='B'}` risulta valida poiché tutti i vincoli interni sono rispettati.

4. Costruzione del problema residuo e pruning con vincoli unari Una volta trovata un'assegnazione valida per il cutset, il solver costruisce il problema residuo composto dalle variabili rimanenti:

$$R = \{\texttt{NSW}, \texttt{V}, \texttt{T}\}$$

A questo punto, i vincoli che collegavano variabili del cutset a quelle del residuo vengono trasformati in vincoli unari, sfruttando i valori già assegnati. In questo passaggio si riducono i domini delle variabili residue sulla base delle restrizioni imposte dal cutset fissato.

Ad esempio, per l'assegnazione del cutset:

$$\{\texttt{WA} =' R',\ \texttt{NT} =' G',\ \texttt{Q} =' R',\ \texttt{SA} =' B'\}$$

si ottiene:

- `NSW` è adiacente a `SA` e `Q`: quindi non può essere né B né R, e il suo dominio $\{R, G, B\}$ diventa $\{G\}$;
- `V` è adiacente a `SA`: quindi non può essere B, e il suo dominio diventa $\{R, G\}$;
- `T` è isolata, quindi mantiene il dominio $\{R, G, B\}$.

5. Risoluzione del residuo con TREE-SOLVER e arc consistency Il grafo residuo, ora aciclico, viene risolto dal modulo `tree_solver.py` attraverso la funzione `tree_solve`. Questa opera in due fasi, secondo la procedura dell'algoritmo *TREE -CSP-SOLVER* di Russell & Norvig:

1. **Passata bottom-up: MAKE-ARC-CONSISTENT** In questa fase il solver applica un altro pruning, che però riguarda i vincoli binari interni al residuo. Scorrendo i nodi in postorder (dal basso verso la radice), il solver elimina dai domini dei genitori tutti i valori che non hanno alcun valore compatibile nei figli. Questo garantisce la consistenza degli archi (*arc consistency*) all'interno del residuo.

   Ad esempio, per la coppia $\texttt{NSW} - \texttt{V}$: `NSW` ha dominio $\{G\}$ e `V` ha dominio $\{R, G\}$; poiché esiste almeno un valore compatibile ($\texttt{NSW} = G, \texttt{V} = R$), nessun valore viene eliminato e la componente risulta consistente.

2. **Passata top-down: assegnamento deterministico** Dopo la propagazione, l'albero è già consistente. Partendo dalla radice, il solver assegna il primo valore disponibile nel dominio, e per ciascun figlio seleziona il primo valore compatibile con quello del genitore. Questo produce una soluzione completa in modo deterministico:

   - $\texttt{NSW} = G$
   - $\texttt{V} = R$ (compatibile con NSW=G)
   - $\texttt{T} = R$ (isolata, primo valore disponibile)

6. Composizione finale della soluzione Combinando la soluzione del residuo con l'assegnazione del cutset, il solver ottiene la colorazione completa:

$$\{\texttt{WA} =' R',\ \texttt{NT} =' G',\ \texttt{Q} =' R',\ \texttt{SA} =' B',\ \texttt{NSW} =' G',\ \texttt{V} =' R',\ \texttt{T} =' R'\}$$

Il log finale mostra la sequenza di pruning, la propagazione dei vincoli e l'assegnamento che porta a questa soluzione coerente.

6 Riproducibilità

Per riprodurre i risultati:

1. copiare tutti i file Python nella stessa directory;
2. E' preferibile avere installata l'ultima versione di Python 3 (non sono richieste librerie esterne);
3. eseguire `python3 main.py` (i file dei risultati sono creati in `logs_of_istances`).

Riferimenti

- S. Russell, P. Norvig — *Artificial Intelligence: A Modern Approach*, 4th ed. — sezione 6.5.1 (Cutset conditioning).
- R. Dechter — *Tractable Structures for Constraint Satisfaction Problems* (2006) — sezioni su cycle-cutset e w-cutset (§3.1).