UNIVERSITÀ DI FIRENZE

SCUOLA DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Informatica

Intelligenza Artificiale

Cutset Conditioning

Del Bimbo Davide

1 Problema

In questo esercizio si implementa (in un linguaggio di programmazione a scelta) un solver per problemi di soddisfacimento di vincoli basato sulla tecnica cutset conditioning descritta in R&N 2021 §6.5.1 e in §3.1 di (Detcher 2006). Si verifica il corretto funzionamento del codice su due problemi a scelta tra (1) sudoku, (2) criptoaritmetica, (3) N-queens, (4) map coloring e usando almeno due istanze di ciascuno dei due problemi scelti.

2 Introduzione

Vogliamo implementare un sistema che permetta di risolvere un problema di soddisfacimento dei vincoli (*CSP*) attraverso l'utilizzo della tecnica di *Cutset Conditioning*.
Questa consiste nel rilevare un *cycle-cutset* (ovvero un sottoinsieme di nodi di un grafo non orientato, la cui rimozione restituisce un grafo senza cicli) all'interno del grafo dei vincoli, per individuare un albero (ovvero un grafo privo di cicli) sul quale possiamo eseguire un algoritmo di *tree solver*.
Il *cycle-cutset* sarà risolto attraverso un algoritmo di *backtracking*, il quale verificherà, ad ogni chiamata, se nel grafo è presento almeno un ciclo.

3 Implementazione

Il linguaggio di programmazione utilizzato per risolvere il problema è **Python**.
Inoltre, il corretto funzionamento del programma è stato verificato eseguendo alcuni esempi di *map coloring* e *sudoku*.
Di seguito saranno riportate e descritte le classi implementate.

3.1 Classi

3.1.1 Domain

La classe *Domain* permette di definire un domino per una determinata variabile. Questa classe contiene un set di *valori validi* per il dominio ed un set di *valori illegali*, dovuti a qualche inconsistenza con valori di altre variabili.
Perciò, il set di *valori legali* per una determinata variabile è dato dal set di *valori validi* meno il set di *valori illegali*.

3.1.2 Variable

La classe *Variable* definisce una variabile attraverso un *nome* ed un *dominio* univoco.

3.1.3 Constraint

La classe *Constraint* definisce una vincolo, ovvero una *funzione* che permette di specificare la relazione tra due variabili.
Questa classe contiene un importante metodo (*checkConstraint*) che verifica se due valori (di due diverse variabili) soddisfano la funzione.

3.1.4 Assignment

La classe *Assignment* definisce un possibile assegnamento per un valore legale di una certa variabile.
Tale classe contiene un dizionario delle variabili assegnate ed un dizionario dei valori inferenziati per ogni variabile, dovuto ad un qualche assegnamento.
Il metodo *copy*, definisce una copia difensiva che permette di fissare ad ogni passo dell'algoritmo un nuovo assegnamento. In questo modo, se un assegnamento per una certa variabile non va a buon fine, è possibile ripristinare i valori legali ottenuti a qualche passo precedente.

3.1.5 CSP

La classe *CSP* permette di definire un problema di vincoli, istanziando delle variabili e dei vincoli tra due variabili. Questo permette di definire un *grafo dei vincoli* (definito attraverso un dizionario).
Anche in questo caso è stato implementato un metodo *copy* per la copia difensiva. In questo modo, ogni volta che viene effettuato un nuovo assegnamento, viene realizzato un nuovo grafo dei vincoli a partire dal precedente ed eliminando la variabile assegnata. Così è possibile verificare ad ogni passo, se all'interno del grafo è presente un ciclo.

3.2 TreeSolver

Il *TreeSolver* contiene tutti i metodi utili per implementare l'algoritmo di *tree solver*.
In particolare, una volta individuato un albero, viene definita una visita topologica (realizzata seguendo tale implementazione) scegliendo un qualsiasi nodo dell'albero come *root*.
Una volta definita tale sequenza, viene eseguito un algoritmo di *Arc-Consistency* che permette di individuare per ogni variabile della sequenza (partendo dalle foglie fino alla radice (esclusa)) la consistenza con i propri padri.
Trovate tutte le possibili inconsistenze, si definisce una possibile assegnazione per una variabile e si verifica questa risulta essere consistente. Altrimenti si prova con un altra assegnazione.

3.3 CutsetConditioning

Il *CutsetConditioning* implementa l'algoritmo risolutivo del CSP.
Definito un nuovo assegnamento per il problema, si esegue il *backtrack*, il quale verifica ad ogni passo se è stata trovata una possibile soluzione al problema, altrimenti verifica se nel grafo è presente un ciclo. Per verificare la presenza di un ciclo, si utilizza un algoritmo di *dfs* (realizzato seguendo tale implementazione).
Se viene individuato un ciclo, allora viene scelta una variabile da inserire nel *cycle-cutset* implementando l'euristica del *Minimum Remaining Values* (ovvero si sceglie la variabile che contiene il minor numero di valori nel proprio dominio, oppure, in caso di parità, si sceglie quella che riduce il branching factor).
Individuata tale variabile, si prova un assegnamento su un qualsiasi valore legale nel dominio di essa. Quindi si esegue l'algoritmo di *Maintaining Arc Consistency* che permette di eliminare dalle variabili i valori non coerenti con l'assegnamento effettuato.
Successivamente si esegue ricorsivamente il *backtrack* sul risultato ottenuto.
Se, invece, non è stato trovato un ciclo, allora si esegue l'algoritmo di *TreeSolver*.

3.4 main

Implementa alcuni esempi sul *map coloring* e sul *sudoku*:

- Map Coloring delle regioni dell'Australia con 3 colori (soluzione trovata);
- Map Coloring delle province della Toscana con 3 colori (soluzione trovata);
- Map Coloring delle province della Toscana con 2 colori (soluzione impossibile);
- Sudoku difficile con un'unica soluzione (soluzione trovata);
- Sudoku impossibile (soluzione impossibile);