# Implementazione backtracking e constraint propagation

Lorenzo Cioni
5076361

*Intelligenza Artificiale 2012/13*

**Sommario**

In questo esercizio si implementa l'algoritmo MAC (maintaining arc consistency) per problemi di soddisfacimento di vincoli descritto in R&N 2009, utilizzando la tecnica AC3 per la propagazione di vincoli. L'algoritmo può essere implementato in un linguaggio di programmazione a scelta. La base del codice dovrebbe essere sufficientemente generale da permettere di risolvere un problema generico ma le descrizioni del dominio e dei vincoli possono essere direttamente codificate nel programma (a differenza di quanto accadrebbe con un risolutore general purpose che sarebbe tipicamente in grado di leggere la specifica del CSP in un linguaggio formale). I programmi python in http://aima.cs.berkeley.edu/python/readme.html possono essere usati come punto di partenza o come esempio di implementazione concreta degli algoritmi. Il codice sviluppato dovrà essere testato su almeno due problemi scelti a piacere da CSPLib.[1]

## 1 Introduzione

Il software realizzato implementa, nel linguaggio di programmazione Java, un solver per problemi di soddisfacimento di vincoli. La base del codice è sufficientemente generica per la risoluzione di qualsiasi problema, ma in questo caso la definizione delle variabili, dei domini e dei vincoli è codificata direttamente all'interno del programma. Il codice implementato è un'estensione del codice sviluppato da Ruediger Lunde.[2] Per questo esercizio sono stati presi in esame ed implementati due problemi di soddisfacimento di vincoli: il *quadrato magico*[3] ed il problema numerico di Langford[4].

## 2 Backtracking e AC3

Il solver implementato in questo esercizio si basa sull'algoritmo backtracking search, utilizzando l'algoritmo MAC (*maintaining arc consistency*) tramite l'algoritmo AC3. L'idea di fondo di questo approccio è quella di sfruttare gli assegnamenti effettuati dall'algoritmo di backtracking durante la sua esecuzione per eseguire ulteriori controlli di consistenza ed eliminare un numero maggiore di valori non consistenti dal dominio delle variabili del problema. I domini delle variabili saranno così ridotti, passo dopo passo, rispetto all'originale, semplificando il problema.

Un arco $(X_i, X_j)$(con $i \neq j$) viene detto *arco-consistente* se e solo se $X_i$ e $X_j$ sono *nodo-consistenti* (soddisfano il vincolo unario) e per ogni valore $a$ di $X_i$ esiste un valore $b$ di $X_j$ tale che $X_i = a, X_j = b$ è permesso dai vincoli binari presenti tra $X_i$ ed $X_j$.

[1] CSP Lib: http://ianm.host.cs.st-andrews.ac.uk/CSPLib/subjects.shtml
[2] GitHub: https://github.com/gnufs/aima-java/tree/master/aima-core/src/main/java/aima/
[3] CSP Lib prob0019 : http://ianm.host.cs.st-andrews.ac.uk/CSPLib/prob/prob019/spec.html
[4] CSP Lib prob024: http://ianm.host.cs.st-andrews.ac.uk/CSPLib/prob/prob024/spec.html

Una volta ottenuta la nodo-consistenza per tutti i vari nodi, ottenere la arco-consistenza è abbastanza semplice: per ogni coppia di variabili si verificano tutti i valori della prima variabile; per ogni valore della prima variabile si provano tutti i valori della seconda variabile. Se nessun valore della seconda variabile permette di soddisfare tutti i vincoli tra le due variabili, il valore che è stato testato per la prima variabile può essere rimosso dal dominio.

Per la risoluzione di un problema viene quindi innanzitutto invocato AC3 che riduce i domini delle variabili eliminando eventuali valori tali da rendere non valide le *arc-consistency*, viene poi invocato l'algoritmo di backtracking search.

L'algoritmo di backtracking opera in maniera ricorsiva. Prende in ingresso un assegnamento delle variabili (inizialmente nullo) e procede, una variabile alla volta, ad assegnarne un valore preso dal suo dominio. Se l'assegnamento è consistente con tutti i vincoli procede invocando AC3 per ridurre ulteriormente il dominio della variabile (*inference*) e successivamente invoca nuovamente se stesso con il nuovo assegnamento delle variabili. Un controllo verifica se tutte le variabili hanno assegnato un valore facendo terminare così la ricorsione. Ritorna un possibile assegnamento delle variabili consistente con tutti i vincoli del problema o *null* se ciò non è possibile.

Entrambi i codici per l'implementazione degli algoritmi di backtracking search e di AC3 sono un'estensione del codice scritto da Ruediger Lunde.

# 3 Il Quadrato Magico

Il quadrato magico[5] è un quadrato di lato $n$ (composto da $n^2$ caselle) che contiene i numeri da 1 a $n^2$, tale che la somma dei numeri delle due diagonali e dei numeri di tutte le linee orizzontali o verticali sia identica.

Le variabili del problema sono quindi rappresentate dalle caselle del quadrato, ciascuna con dominio uguale a $D = \{1 \cdots n^2\}$.

La *costante magica* di questi quadrati è data dalla formula:

$$M(n) = \frac{1}{n} \sum_{k=1}^{n^2} k = \frac{1}{2} n(n^2 + 1)$$

I vincoli del problema possono quindi essere espressi tramite un *all-different constraint* su tutte le variabili ed un *exact sum constraint* su ciascuna riga e colonna del quadrato. E' necessario quindi creare questi due tipi di vincoli al fine di implementare il problema.

L'*all-different constraint*, rappresentato dalla classe *AllDifferentConstraint.java*, prende in ingresso un vettore di variabili ed è verificato se tutti i valori assegnati a queste variabili sono diversi tra loro.

L'*exact sum contraint*, rapprentato dalla classe *ExactSumConstraint.java*, prende in ingresso un vettore di variabili ed un intero che rappresenta la somma da verificare (la *costante magica* del quadrato). E' verificato se la somma dei valori assegnati alle variabili contenute nel vettore dato è equivalente alla *costante magica*.

## 3.1 Il caso $n = 3$

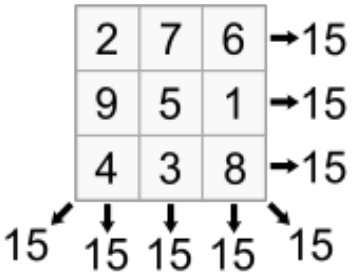
Nell'esercizio viene esaminato il caso in cui $n = 3$. Il dominio delle variabili risulta quindi essere composto da tutti gli interi da 1 fino a $n^2$: $D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. La costante magica di questo quadrato è:

$$M(3) = \frac{1}{2} 3(3^2 + 1) = 15$$

[5] CSP Lib problema numero 19.

Viene quindi impostato il vincolo di *AllDifferent* su tutte le variabili del problema e per ciascuna riga e colonna viene impostato il vincolo di *ExactSum* con somma pari alla *costante magica*, in questo caso 15.

Il risultato è il quadrato così costruito:




## 4 Il problema numerico di Langford

Consideriamo due insiemi di numeri interi contenenti ciascuno i numeri da 1 a 4. Il problema[6] consiste nell'ordinare gli 8 numeri contenuti nei due insiemi in un unico insieme nel quale i due numeri 1 compaiono ad un numero di distanza, i due numeri 2 a due numeri di distanza, i due 3 a tre numeri di distanza ed i due 4 a quattro numeri di distanza.

Le variabili in questo problema sono rappresentate dalle posizioni nell'insieme finale. Per meglio rappresentare i risultati ottenuti i domini delle variabili sono invece rappresentati da quattro colori, ai quali è associato un numero intero da 1 a 4: $D = \{black, red, blue, yellow\}$. Nell'esercizio i valori sono stati assegnati in questo modo: $black = 1, red = 2, blue = 3, yellow = 4$.

Per la specifica dei vincoli è stato necessario implementare due nuovi tipi di vincoli: il *PairConstraint* e il *PositionConstraint*.

Il *PairConstraint* riceve in ingresso il vettore delle variabili ed uno dei 4 possibili valori. E' verificato se quel valore compare al massimo 2 volte in tutto il vettore.

Il *PositionConstraint* riceve in ingresso tutto il vettore delle variabili del problema. Il vincolo è verificato se ciascun valore compare nella giusta posizione nel vettore cioè se le distanze vengono rispettate.

Viene quindi impostato il PairConstraint su tutti e 4 i valori del dominio e il PositionConstraint su tutte le variabili del problema.

Il risultato, in forma grafica, è il seguente:

[6] CSP Lib problema numero 24.