Confronto strategie di ricerca per programmazione a vincoli

Alice Casali

Abstract

In questo esercizio, scritto in linguaggio Python, si confronta diverse strategie di ricerca per programmazione a vincoli (CSP). In particolare si utilizza la ricerca con Backtracking (BT) (descritto in [1]), inizialmente senza inferenza col BT Puro, poi BT con Forward Checking, infine BT con Maintaining Arc Consistency (MAC). Si confronta le applicazioni su varie istanze di Sudoku.

1 Introduzione

Il programma utilizza una struttura generale per la risoluzione dei CPS basata sul codice fornito da [2]. Nello specifico si converte il gioco del Sudoku in un problema a vincoli risolvibile attraverso la ricerca con Backtrack nelle sue varie forme. La struttura del codice è rimasta generale per poterla utilizzare anche su altri tipi di CSP. La risoluzione viene eseguita sul gioco del Sudoku, si va a confrontare i tempi di esecuzione e il numero di passi all'indietro necessari per risolvere il puzzle.

2 Ricerca con Backtracking

Prima di iniziare la ricerca con Backtracking viene eseguita la funzione AC-3 (destritta in [1]) per inferire riduzioni del dominio delle variabili del Sudoku, in modo da eliminare gli assegnamenti banali che altrimenti rallenterebbero inutilmente il Backtrack in ogni sua forma.
L'algoritmo di Backtrack è modellato sulla ricerca in profondità ricorsiva. Si usano le seguenti funzioni:

- **Select Unassigned Variable:** sceglie una variabile non assegnata del CSP, in questo caso si utilizza sempre l'euristica MRV (Minimum Remaining Values). Essa sceglie la variabile per cui è maggiore la probabilità di arrivare presto al fallimento, potando così l'albero di ricerca. Si adopera questa scelta per ridurre i tempi di esecuzione del software.
- **Order Domain Values:** non si utilizza nessuna euristica per la scelta del valore perchè nel caso del gioco del Sudoku, la soluzione è unica.

- **Inference:** si può usare la strategia di inferenza che si preferisce. In questo caso si esegue la ricerca: senza inferenza (BT Puro), inferendo con verifica in avanti (Forward Checking) e con consistenza d'arco (Maintaining Arc Consistency).

2.1 Backtracking Puro

Il Backtracking puro viene eseguito senza inferenza. Si sceglie la variabile di partenza con l'euristica MRV, successivamente si sceglie un valore all'interno del dominio della variabile e si controlla che con questa scelta non si verifichino inconsistenze con la funzione **nconflicts**. Il vincolo che si potrebbe violare nel gioco del sudoku è la presenza del solito valore su due celle diverse nella stessa riga, nella stessa colonna o nello stesso riquadro 3x3. Se non si presentano violazioni si assegna il valore corrente senza ridurre i domini delle altre variabili e si procede ricorsivamente. Se viene rilevata un'inconsistenza l'algoritmo di backtrack restituisce il fallimento e la chiamata precedente prova ad assegnare un altro valore.

2.2 Backtracking con Forward Checking

Il Backtracking viene eseguito con l'algoritmo di inferenza Forward Checking per diminuire il numero di backtrack utilizzati nel risolvere il gioco del sudoku. Si decide di scegliere la variabile utilizzando nuovamente l'euristica MRV in modo da poter confrontare solo l'impiego dell'inferenza (assente nel BT puro). Dunque una volta assegnata la variabile V, il processo di verifica in avanti stabilisce la consistenza d'arco per essa: per ogni variabile non assegnata W collegata a V da un vincolo, l'inferenza cancella dal dominio di W ogni valore non consistente con quello scelto per X. Se durante l'assegnamento parziale il dominio di una variabile rimane vuoto vuol dire che l'assegnamento non è consistente e l'algoritmo tornerà indietro con il backtracking.

2.3 Backtracking con Maintaining Arc Consistency

Il Backtracking questa volta viene eseguito con l'algoritmo di inferenza MAC che, a differenza del forward checking, rende la variabile corrente arco-consistente e inoltre guarda anche avanti per rendere consistenti tutte le altre variabili. Per farlo, una volta che alla variabile Vi (scelta con euristica MRV) viene assegnato un valore, la procedura **inference** richiama AC-3, utilizzando una coda con gli archi (Vi, Vj) per tutte le Vj che sono variabili non assegnate adiacenti a Vi. Dunque sarà la procedura AC-3 questa volta a ridurre i domini e a notificare l'inconsistenza di un assegnamento quando un dominio sarà svuotato. In questo modo si riduce ulteriormente il numero di BT effettuati.

3 Test Eseguiti

In base a quanto detto, si eseguono dei test così formulati: si utilizzano 6 schemi di gioco del sudoku con difficoltà crescente (presi da [3]). Essi verranno risolti con BT Puro, BT con forward checking e BT con MAC in modo da poter confrontare, per ciascuno, il numero di BT eseguiti e il tempo di esecuzione. Per una maggior accuratezza si esegue lo stesso schema 5 volte per ogni algoritmo impiegato (è possibile aumentare il numero di esecuzioni dello schema), poi viene restituito il tempo medio di esecuzione e il numero medio di BT eseguiti.

3.1 Aspettative

Secondo gli algoritmi impiegati e la teoria a loro relativa ci aspetteremo notevoli differenze nei risultati, all'aumentare della difficoltà del sudoku. In particolare:

- **BT Puro:** risolverà ogni schema con un alto numero di BT e impiegherà molto più tempo rispetto ai BT con inferenza.
- **BT con Forward-Checking:** otterrà un numero molto più basso di BT e un tempo decisamente ridotto rispetto al BT Puro.
- **BT con MAC:** avrà il numero più basso di BT ma il tempo di esecuzione sarà leggermente più alto rispetto al BT forward-checking perché i passi del BT con MAC sono più lunghi, ovvero richiedono un maggior costo computazionale.

3.2 Tabella del numero di BT

Strategia	**Easy 0**	**Easy 1**	**Medium 2**	**Hard 3**	**Hard 4**	**Evil 5**
BT Puro	125	3075	53343	310633	949320	∼128M
BT FC	0	42	158	251	466	5063
BT MAC	0	11	14	85	151	2707

Table 1: I valori inseriti nella tabella sono il numero di backtrack, misurati in base ai differenti tipi di inferenza eseguiti sul puzzle.

3.3 Tabella dei tempi

Strategia	**Easy 0**	**Easy 1**	**Medium 2**	**Hard 3**	**Hard 4**	**Evil 5**
BT Puro	0.007	0.137	2.198	79.632	46.131	∼6440
BT FC	0.004	0.006	0.012	0.014	0.027	0.256
BT MAC	0.008	0.037	0.021	0.074	0.092	2.103

Table 2: I valori inseriti nella tabella sono i tempi misurati in secondi, misurati in base ai differenti tipi di inferenza eseguiti sul puzzle.

4 Descrizione degli Esperimenti Condotti

- **Dati Utilizzati:** Giochi del sudoku presi da [3]
- **Specifiche Piattaforma:** Computer con processore i7, 8 GB di RAM. 1 TB di hard disk diviso in due partizioni: Windows 10 e Linux Mint 18.1. Ho eseguito il programma su Mint utilizzando Pycharm con Python 3.5.2.
- **Numero dei test:** I test vengono eseguiti su 6 schemi diversi di sudoku.
- **Run: Ho implementato le funzioni in modo da eseguire Test.py esplicitando nella funzione main() quanti test si vuole eseguire per fare la media (`n_test`), quale tipo di inferenza si vuole utilizzare (inf) a scelta tra `no_inference`, `forward_checking` o `mac`, infine quale gioco del sudoku si vuole eseguire: bisogna inserire un numero da 0 a 5, la difficoltà è crescente.**

5 Conclusioni

Possiamo affermare che i risultati ottenuti sono congruenti alle aspettative basate sugli aspetti teorici di questi metodi implementativi. Detto questo è preferibile utilizzare l'algoritmo di ricerca con backtracking che sfrutta la strategia di inferenza Maintaining Arc Consistency. Il numero di backtrack è ottimale rispetto agli altri algoritmi anche se il tempo di esecuzione dei test è leggermente superiore all'algoritmo con inferenza Forward Checking.

References

[1] Stuart Russell and Peter Norvig
Artificial Intelligence: A Modern Approach, 3rd edition. Pearson, 2010

[2] Code for the book "Artificial Intelligence: A Modern Approach"
`https://github.com/aimacode/aima-python/blob/master/csp.py`

[3] MiniSudoku. Sudoku stampabili
`http://www.minisudoku.it/printable-sudokus`