Confronto strategie di ricerca per programmazione a vincoli

Studente: Steven Salazar (5971614)

February 2018

Abstract

In questo esercizio si implementa, in linguaggio Python , l'algoritmo di Backtracking (descritto in [1]) con le sue variazioni per la propagazione dei vincoli, in particolare si utilizzerà un Backtracking puro (senza inferenza), BT con Forward-checking e BT con MAC (Maintaining Arc Consistency), nella parte finale dell'esercizio si confronterà la loro applicazione su diverse istanze di Sudoku.

1 Introduzione

Il software realizzato implementa principalmente l'algoritmo di backtracking con una sua rappresentazione astratta nella quale non è specificato il modo con cui:

- Scegliere la variabile a cui assegnare un valore.
- Scegliere il valore da assegnare alla variabile scelta.
- Fare inferenza sulle altre variabili.

Come base di implementazione è stato preso in considerazione il codice fornito da [2], questo è stato leggeramente modificato in modo da migliorare le prestazioni dell'applicazione degli algoritmi a diverse istanze di CSP, tuttavia il codice è rimasto sufficientemente generale in modo da permettere di risolvere un generico problema, che però dovrà specificare:

- Le variabili da tenere in considerazione.
- Il dominio delle variabili.
- La relazione che lega le variabili.
- Le variabili legate a una certa variabile da una relazione.

2 Backtracking

Come è già stato anticipato nel *Paragrafo 1* in questo esercizio è stato implementato l'algoritmo di backtracking che però presenterà diverse sue rappresentazioni, per la scelta della variabile è stata presa in considerazione l'euristica Minimum Remaining Values (MRV) che ci permette di scegliere la variabile con il dominio minore in modo da fare backtrack velocemente (chiamata anche fail-first), per la scelta del valore da assegnare invece non è stato presa in considerazione nessuna euristica, questo perché normalmente nel problema del sudoku la soluzione è unica e

quindi è indiferente cercare di trovare la prima soluzione (in caso di fallimento il programma sviluppato mostrerà un messaggio di errore dal momento che non è possibile trovare la soluzione), invece è utile cercare di accorgerci che abbiamo scelto la variabile sbagliata con il MRV, questo è anche molto intuitivo, ad esempio nel gioco del Sudoku se vediamo riga con una sola casella libera (Dominio con un unico elemento) la prima cosa che si fa è proprio assegnare un valore a questa casella, invece per quanto riguarda la propragazione dei vincoli abbiamo 3 diverse opzioni:

2.1 Backtracking Puro

Nel caso di un backtracking puro, ovvero senza inferenza, al momento della scelta di un valore da assegnare a una certa variabile $v_1 \in V$ si procede chiamando il metodo $nconflicts$ che restituisce il numero di violazioni che si avrebbe all'assegnare questo valore a v_1, se il numero di violazioni è 0 si procede con l'assegnazione di v_1 (senza informare alle altre variabili di questa scelta, per questo motivo il loro dominio continuerà ad essere invariato) per poi scegliere una nuova variabile ricorsivamente, però quando una variabile $v_2 \in V$ (con $v_2 \neq v_1$) non trova un valore per il quale il numero di violazioni (rispetto alle assegnazioni già fatte) è 0 l'algoritmo si accorge di questo e si procede con il backtrack cambiando il valore assunto per v_1, quindi è naturale aspettarci un numero elevato di backtrack nella risoluzione di un qualunque problema di CSP.

2.2 Backtracking con Forward-checking

A differenza del backtracking puro, con il backtracking con forward-checking al momento dell'assegnazione di un valore ad una variabile $v1$, si informa alle variabili (legate a questa da un vincolo) in modo che queste riducano il loro dominio per poi diminuire il numero di possibili stati da analizzare, ovvero si cerca di tagliare l'albero in modo che sia molto più veloce trovare la soluzione, questi tagli non vanno a cancellare la soluzione dal momento che sono tagli di archi non consistenti con l'assegnazione fatta.
Ad esempio nel caso del Sudoku quando una casella assume un valore, si informa alle varibili della stessa riga, colonna e square box in modo che queste non abbiano nel loro dominio questo valore, altrimenti ci sarebbe una inconsistenza tra le variabili, quindi in questo caso è naturale aspettarci un minor numero di possibili stati analizzati, ovvero un albero molto più ridotto rispetto al backtracking puro, e dunque anche un minor numero di backtrack.

2.3 Backtracking con MAC

Come è già stato descritto nel *Paragrafo 2.2* il forward-checking ci permette di fare inferenza sulle variabili legate ad una variabile v_1, e se si trova che dare un certo valore a v_1 provoca un dominio vuoto in una variabile v_2 (legata a v_1 da un vincolo) allora si torna indietro e si cambia il valore, però è utile notare che questo si poteva prevedere prima ancora di assegnare il valore a v_1, questa soluzione è chiamata Maintaining arc-consistency, dopo avere assegnato un valore a una variabile v_0, si chiama l'AC-3 (descritto in [1]) su tutti i vicini di v_0 (tra cui ci sarà anche v_1), si tolgono i valori in D_1 per i quali si avrebbe un dominio vuoto tra i suoi vicini (tra i quali ci sarà anche v_2), e se $|D_1| = 0$ allora si ha un'inconsistenza all'assegnare il valore a v_0 e dunque si procede cambiando il valore per v_0, in questo modo si riesce a guardare avanti quando si assegna un valore a una variabile.
Per quanto riguarda il backtracking con MAC ci si aspetta un minor numero di backtrack rispetto al FC ma con un tempo che potrebbe essere leggeramente elevato dal momento che i passi del backtracking con MAC sono più lunghi, ovvero richiedono un maggiore costo computazionale.

Strategia	Easy 0	Easy 1	Easy 2	Hard 1	Hard 2	Evil 1
BT puro	44	165	254	254 924	28 697	95 818
BT FC	1	0	0	2 875	170	12 899
BT MAC	0	0	0	587	19	3 775

Tabella 1: Numero di backtrack ottenuti applicando ciascun algoritmo a diverse istanze di sudoku

3 Sudoku

Riprendendo la parte finale del *Paragrafo 1* possiamo modellare il Sudoku in modo da poter applicare i diversi algoritmi di backtracking descritti in precedenza, per cui dovremmo specificare:

- Variabili: L'insieme composto dalle 81 caselle, $V = \{v_0, ..., v_{80}\}$.
- Dominio delle variabili: $D_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $\forall$ v_i vuota e $D_j = \{\text{valore in } v_j\}$ $\forall$ v_j non vuota.
- Vincolo: due caselle nella stessa riga, colonna e matrice 3x3 non devono avere lo stesso valore.
- Variabili legate a una certa variabile v: le caselle nella stessa riga, colonna e matrice 3x3.

In aggiunta all'implementazione degli algoritmi è stato creato un file che permette di visualizzare graficamente il sudoku, la sua soluzione e scegliere quale metodo di inferenza si vuole utilizzare, per maggiori informazioni sullo sviluppo grafico in python consultare: [5], [6] e [7].

4 Esecuzione dei test

Nel file *gui.py* sono riportate 6 istanze di sudoku con 2 diversi livelli (Gli esempi sono stati presi da [4] e [3], in seguito verranno mostrati i risultati ottenuti dall'esecuzione degli algoritmi su ciascuna istanza di sudoku (5 test per ciascun algoritmo).

4.1 Risultati sperimentali

Come si osserva nella *Tabella 1* il numero di backtrack che si realizza con il backtracking puro è sempre superiore rispetto al forward-checking e MAC, questo è del tutto normale dal momento che con il backtracking puro non facciamo inferenza, inoltre si osserva anche come le nostre aspettative teoriche siano state confermate, ovvero che, utilizzare il forward-checking implica avere un maggior numero di stati da analizzare (rispetto al MAC) e quindi un maggior numero di backtrack.
Dalla *Tabella 2* invece osserviamo i tempi che impiega ciascun algoritmo, si osserva quindi come il backtracking puro ha bisogno di più tempo per trovare la soluzione del problema, questo è del tutto normale dato l'elevato numero di stati da analizzare e quindi confermato anche dal numero di backtrack nella *Tabella 1*, però osserviamo anche una cosa molto interessante, si osserva che il forward-checking impiega un minor tempo rispetto al MAC, come era già stato anticipato nel *Paragrafo 2.3*, questo conferma i nostri sospetti teorici dal momento che il MAC realizza passi molto più impegnativi propagando l'arc-consistency tra le variabili e quindi richiede anche un maggiore tempo computazionale anche se analizza un minore numero di stati.
Per riprodurre i risultati sperimentali è sufficiente eseguire il file *test.py* (con python versione 3.4 o superiore), modificando la variabile *inf* nel metodo *main()*: inf= no_inference si riferisce al backtracking puro, inf= forward_checking al backtraking con FC e inf=mac al backtracking con MAC.

Strategia	Easy 0	Easy 1	Easy 2	Hard 1	Hard 2	Evil 1
BT puro	0.00874	0.014484	0.01868	14.46901	1.69020	5.546142
BT FC	0.00534	0.005116	0.00659	0.105089	0.011204	0.4636
BT MAC	0.0315	0.02603	0.02526	0.39495	0.05872	1.9644

Tabella 2: Tempo (in secondi) impiegato da ciascun algoritmo per trovare la soluzione delle diverse istanze di sudoku

Per selezionare l'istanza di sudoku è sufficiente cambiare i valori delle variabili *level* e *which* con i valori indicati dai commenti nel codice, inoltre (come è stato detto nel *Paragrafo 3*) è stato aggiunta la possibilità di visualizzare lo stato iniziale e finale del sudoku con i rispettivi tempi impiegati dagli algoritmi, per far partire l'interfaccia grafica è sufficiente eseguire il file *main.py* (sempre con python 3.4 o superiore).
Nella interfaccia grafica di default è selezionato il livello più facile ma per cambiare livello è sufficiente cliccare una volta sul menu "Level/Hard" (per cambiare istanza continuando con il livello Hard si deve cliccare una seconda volta su "Level/Hard", vale lo stesso per il livello Easy).

4.2 Conclusione

Dai risultati sperimentali visti in precedenza possiamo concludere che: in generale, se si vuole avere il minor numero di stati analizzati la scelta migliore è il backtracking con MAC; però se l'obiettivo è quello di trovare la soluzione impiegando il minor tempo possibile, la migliore opzione potrebbe essere il forward-checking dal momento che questo impiega un minor tempo anche se il numero di stati analizzati è leggeramente superiore rispetto al MAC; inoltre se il problema non presenta un insieme di stati così elevato, come ad esempio l'istanza di sudoku "Easy2" (`http://prntscr.com/iclrd0`), visti i risultati sperimentali nelle *Tabelle 1 e 2*, potrebbe tornare utile anche il backtracking puro se si vuole applicare un algoritmo semplice e pulito(senza troppe chiamate ad altri metodi).

Riferimenti

[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`

[3] The telegraph: World's hardest sudoku ever:
`http://bit.ly/1EN89Ke`

[4] Evil Sudoku generator:
`https://www.websudoku.com/`

[5] The new Boston: Python GUI with Tkinter
`https://youtu.be/RJB1Ek2Ko_Y`

[6] Effbot: An Introduction to Tkinter
`http://effbot.org/tkinterbook/`

[7] New Coder: Graphical User Interfaces
`http://newcoder.io/gui/`