Confronto strategie di ricerca per programmazione a vincoli

Roton Meta

8 Febbraio 2019

Abstract

Implementazione di un risolutore di problemi di soddisfacimento di vincoli (CSP) basato sull'algoritmo di Backtracking (BT) e confronto tra le strategie che abbiamo studiato: BT puro (senza inferenza), BT con forward-checking e BT con Maintaining Arc Consistency (MAC). Il confronto delle strategie viene fatto su diverse istanze di sudoku.

1 Introduzione

Il software è stato realizzato nel linguaggio Python ed il codice del risolutore di CSP e i vari algoritmi di bactracking sono basati sul codice fornito da (2). Il codice è generico e può essere utilizzato su diversi tipi di problemi ma in questo esercizio viene usato sul Sudoku per confrontare le performance delle strategie in termini di tempo di esecuzione e numero di backtrack (passi indietro) necessari a risolvere il puzzle.

2 Backtracking

L'algoritmo utilizzato è Backtracking che esegue una ricerca in profondità (DFS) e prova ad assegnare i valori per una variabile alla volta, ritorna indietro (back track) quando non ci sono più valori ammissibili da assegnare. Quando rileva un inconsistenza (non rispetto dei vincoli) ritorna indietro e prova un altro valore. Dato che la rappresentazione CSP è standard non c'è bisogno di fornirgli: stato iniziale, modello transizionale, funzione azione, goal test, ecc.
Una spiegazione più dettagliata di questi algoritmi può essere consultata su (1) Per rendere più "intelligente" l'algoritmo si possono definire le seguenti funzioni (euristiche):

- **Select-unassigned-variable**: sceglie la prossima variabile non assegnata su cui eseguire le assegnazioni. Noi usiamo MRV (minimum remaining values), sceglie la variabile più vincolata che è più probabile che causi

fallimento (fail-first) in modo da evitare ricerche inutili. Si può anche usare first-unassigned-variable che sceglie le variabili in base all'ordine in cui si trovano nel dominio, ma si dimostra che è altamente inefficiente.

- **Order-domain-values**: sceglie l'ordine di assegnazione dei valori di un dominio, può portare a miglioramenti di performance se usata l'euristica least-constraining-value, ma nel caso del Sudoku la soluzione è unica quindi non importa l'ordine dei valori.
- **Inference**: Sceglie la strategia di inferenza, cioè tra le 3 che dobbiamo comparare: no-inference, forward-checking e mac.

2.1 Backtracking puro

É la versione senza inferenza, dunque senza propagazione dei vincoli. Questo vuol dire che quando assegna una variabile che non crea conflitti (es. sudoku: 2 valori uguali nella stessa riga) non riduce il dominio dei suoi vicini. Quando non riesce ad assegnare più una variabile senza che si crei un conflitto fa back track e riassegna la variabile precedente.
Da questo algoritmo ci aspettiamo un numero molto grande di back track che comporta anche un tempo maggiore di esecuzine.

2.2 Backtracking con Forward-checking

Il backtracking viene eseguita con l'inferenza forward-checking che rende consistenti le variabili connesse alle variabili a cui ha assegnato un valore.
Modifica il dominio dei vicini (neighbor) della variabile assegnata, quindi pota l'albero di ricerca.
Ad esempio nel sudoku dopo l'assegnazione di una casella, ridurrebbe il dominio delle variabili nella stessa riga, nella stessa colonna e nello stesso riquadro 3x3.
Ovviamente il numero di back track qui sarà minore avendo uno 'spazio' minore in cui muoversi.

2.3 Backtracking con MAC

In un certo senso può essere visto come un miglioramento del forward-checking perché si tratta comunque di rendere consistenti gli archi ma non solo, cerca anche di mantenere questa consistenza. Per fare questo l'algoritmo dopo l'assegnazione di una variabile esegue AC-3 su tutti gli archi che lo connettono ai vicini non assegnati.
AC-3 rende consistenti gli archi e propaga questa consistenza su tutti gli archi, e quando un'assegnazione rende vuoto il dominio di un'altra variabile, torna indietro e prova un'altra assegnazione.
Forward checking invece se ne accorge più tardi solo quando analizza quella certa variabile.

Sicuramente il numero di back-track con MAC è molto minore degli altri algoritmi analizzati ma essendo AC-3 una procedura lunga potrebbe richiedere più tempo.

3 Sudoku

Il gioco del sudoku è perfetto per essere posto come problema a vincoli. Possiamo modellarlo in questo modo per poterci applicare gli algoritmi del capitolo 2:

- Variabili: 81 caselle identificate tramite la coppia (riga,colonna) Vars = {'0,0','0,1',...,'9,9'}.
- Domini: Domain = {1,2,3,4,5,6,7,8,9} per ogni variabile non assegnata e Domain = *value* per le variabili assegnate dallo specifico puzzle.
- Vicini: i vicini di una certa variabile (casella) sono tutte le altre variabili nella stessa riga, colonna o quadrato 3x3.
- Vincolo: *Alldiff*, tutti i vicini diversi fra loro.
- Soluzione: Unica.

Le istanze di sudoku usate per fare i test sono 5, 4 sono state generate dal generatore di Sudoku online (3) e sono ordinate per difficoltà (la difficoltà dipende dal numero di caselle già compilate, più è basso più è difficile). L'altro invece l'ho preso dall'articolo (4), in cui si dice che sia il più difficile al mondo.

4 Test

I test consistono nella esecuzione degli algoritmi che abbiamo trattato e nella misurazione del tempo di esecuzione e del numero di backtrack. Per avere dei dati più affidabili lo stesso algoritmo sulla stessa istanza di sudoku viene eseguita 5 volte e poi viene fatta la media dei risultati.
Per eseguire i test bisogna andare nel file *main.py* e settare le variabili desiderate. Le variabili che si possono modificare sono: *n_test* che definisce il numero di test (5 di default); *level* che sceglie tra le diverse istanze di sudoku, da 1 a 5 (ordine crescente di difficoltà); *inference* e *var_selection* che si possono settare come spiegato nella lista del capitolo 2.

4.1 Analisi risultati

Come si può notare dalla Tabella 1 il numero di BT migliora parecchio con le inferenze, a tal punto che per le istanze semplici BT FC e BT MAC non fanno nessun backtrack.
Dalla tabella 2 invece vediamo che il tempo di esecuzione minore è dato da BT FC, questo viene spiegato dal fatto che la procedura di Arc-Consistency di MAC è più impegnativa dal punto di vista computazionale. Invece BT FC anche se fa

più errori (back track) impiega meno tempo essendo un algoritmo più semplice. Backtracking puro, ovviamente, è il peggiore dei tre in entrambi i parametri analizzati. BT puro è un algoritmo 'smemorato', non impara dagli errori e prova ad assegnare valori senza considerare nessuna informazione finché raggiunge la soluzione.

Algoritmo	Easy	Medium	Hard	Evil	Hardest
BT Puro	257	268	10475	24509	95818
BT FC	0	0	276	106	12899
BT MAC	0	0	45	14	3775

Table 1: Numero di back track eseguiti da ciascun algoritmo per ogni istanza di sudoku

Algoritmo	Easy	Medium	Hard	Evil	Hardest
BT Puro	0.035	0.030	0.859	2.577	7.52
BT FC	0.009	0.014	0.025	0.018	0.676
BT MAC	0.073	0.066	0.092	0.083	3.251

Table 2: Tempo di esecuzione in secondi per ciascun algoritmo su ogni istanza di sudoku

4.2 Conclusioni

I risultati ottenuti sono in accordo con la teoria.
Quindi possiamo dire che l'algoritmo migliore in termini di tempo di esecuzione è BT con Forward-checking. Se invece per un particolare problema si predilige un algoritmo che fa meno "errori" allora la scelta ricade su BT con Maintaining arc consistency.

References

[1] Russell And Norvig: "Artificial Intelligence - A Modern Approach"
3rd edition, 2010

[2] AIMA Code
`https://github.com/aimacode/aima-python`

[3] Websudoku, Sudoku generator
`https://www.websudoku.com`

[4] World's hardest sudoku: can you crack it?, The Telegraph
`https://bit.ly/2PBkQEu`