Risoluzione del Sudoku mediante backtracking

Andrea De Fanti VR089606

15 aprile 2011

Indice

1 Le regole del gioco

Un Sudoku è una griglia di 9x9 quadretti in ognuno dei quali si dovrà scrivere un numero, da 1 a 9. La griglia a sua volta è divisa in 9 regioni di 3x3 quadretti. Le regole del gioco sono semplici si tratta di riempire una tabella, in cui sono già inseriti alcuni numeri in modo tale da rispettare le seguenti regole:

- ogni riga deve contenere tutti i numeri da 1 a 9
- ogni colonna deve contenere tutti i numeri da 1 a 9
- ogni blocco 3x3 deve contenere tutti i numeri da 1 a 9
- in nessuna riga, colonna e blocco 3x3 può apparire due volte lo stesso numero

La difficoltà del gioco varia in base al numero e alla disposizione iniziale delle caselle scoperte.
Bertram Felgenhauer e Frazer Jarvis si sono posti il problema di contare il numero di griglie Sudoku complete. Questo numero è 6 670 903 752 021 072 936 960 $\simeq 6.671 * 10^{21}$

2 La formalizzazione

Il Sudoku può essere formalizzato come rete di vincoli. Una rete di vincoli è formata da tre componenti:

1. X insieme di variabili
2. D insieme di domini
3. C insieme di vincoli

Utilizzando una rappresentazione matriciale della griglia, è possibile mappare le componenti nel seguente modo:

- v_{ij} è il valore nella *j*th cella della *i*th riga
- $D_{i,j}$= D= {1,2,3,4,5,6,7,8,9}
  Blocchi:
  B1 = {11, 12, 13, 21, 22, 23, 31, 32, 33} ... B9 = {77, 78, 79, 87, 88, 89, 97, 98, 99}
- $C^R : \forall_i, \bigcup_j v_{ij} = D$ (ogni valore appare in ogni riga)
  $C^C : \forall_j, \bigcup_i v_{ij} = D$ (ogni valore appare in ogni colonna)
  $C^B : \forall_k, \bigcup (v_{ij} | ij \in B_k) = D$ (ogni valore appare in ogni blocco)

Alternativamente i vincoli possono essere considerati come disuguaglianza di coppie:

- $I^R : \forall i, j \neq j^{'} : v_{ij} \neq v_{ij'}$ (nessun valore doppio in ogni riga)
  $I^C : \forall j, i \neq i^{'} : v_{ij} \neq v_{i'j}$ (nessun valore doppio in ogni colonna)
  $I^B : \forall k, ij \in B_k, i'j' \in B_k, ij \neq i'j' : v_{ij} \neq v_{i'j'}$ (nessun valore appare doppio in ogni blocco)

La seconda rappresentazione include un vantaggio: ho tutti vincoli binari.

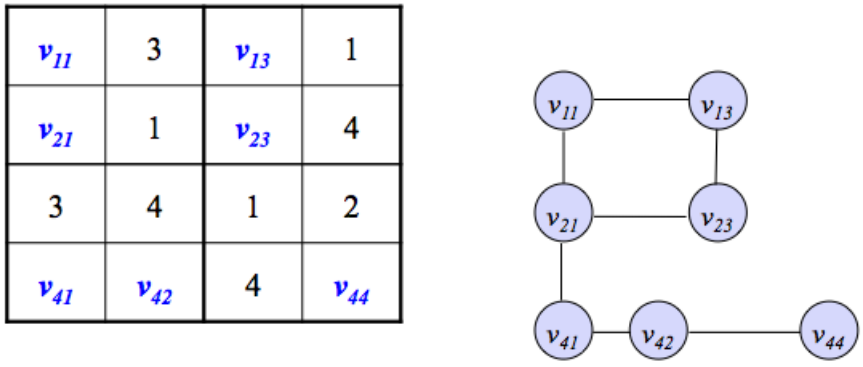


Figura 1: Trasformazione in rete binaria di vincoli

3 Lo scopo del progetto

Il progetto consiste nell' analizzare le prestazioni, in particolare i tempi di esecuzione e il numero degli assegnamenti di due o più algoritmi risolutivi con la tecnica di backtracking. A tal proposito sono stati implementati quattro algoritmi:

1. risolvi per riga (BackTracking1.java)
2. risolvi per colonna (BackTracking2.java)
3. risolvi in modalità random (BackTracking3.java)
4. risolvi dinamicamente tramite l'euristica, considera il blocco più vincolato (BackTracking4.java) (Euristica)

Ogni algoritmo implementa l'interfaccia Solver.java in modo tale da rendere l'implementazione il più generale possibile. Inoltre tutti e quattro gli algoritmi estendono la classe BaseSolver.java ereditando solamente la variabile stop. Questa variabile non influenza l'algoritmo di backtracking, ma viene utilizzata per forzare la terminazione delle chiamate risolutive nel caso in cui l'utente esca improvvisamente dalla griglia grafica.

Il programma viene fornito con un'interfaccia grafica ed è in grado di leggere le griglie in formato standard (machine readable) (si veda l'esempio nella sezione successiva).

4 La generazione della griglia

La griglia utilizzata per l'inizializzazione del gioco può essere generata in due diverse modalità:

- internamente al programma, cliccando sul pulsante Automatic grid generation. In questa modalità ogni griglia è random ed ha esattamente 17 numeri fissati, che garantiscono un'unica soluzione[1]. La copia della griglia viene salvata nella sottodirectory ./grid/

  Il funzionamento è il seguente: si seleziona in modo random una riga, una colonna e un valore dal dominio. Se il valore scelto è consistente con i valori già assegnati su quella riga, colonna, blocco si può procedere con l'inizializzazione della cella, altrimenti si ripete il procedimento. Il procedimento termina dopo aver fissato i 17 numeri nella griglia.

- esternamente, lanciando il seguente comando di shell su piattaforma ubuntu: *sudoku -g 1 > nomefile.txt*
  Richiede l'installazione del package sudoku (apt-get install sudoku). Queste griglie hanno più di 17 numeri fissati, quindi la procedura di backtracking potrebbe terminare su una della possibili soluzioni. Per caricare la griglia da file, si utilizza il pulsante Choose input file grid. Come si può notare dall'immagine, la prima riga è un commento che identifica il grado di difficoltà di risoluzione del gioco.

Esempio formato griglia generata esternamente:

```
% randomly generated - medium
. . . | . . . | . . .
. 1 . | 9 . 8 | . 6 .
. 5 2 | 6 . 4 | 3 1 .
------+-------+------
8 . . | 5 . 6 | . . 7
. . 4 | 2 . 9 | 8 . .
6 . . | 8 . 7 | . . 2
------+-------+------
. 8 1 | 7 . 3 | 2 9 .
. 4 . | 1 . 5 | . 7 .
. . . | . . . | . . .
```

[1] The Algorithm Design Manual Second Edition Steven S. Skiena

5 L'interfaccia e gli algoritmi

L'interfaccia Solver.java indica i metodi pubblici comuni a tutti gli algoritmi risolutivi. Nel dettaglio i metodi pubblici sono i seguenti:

- setGrid: è utilizzato per settare la griglia una volta che è stata generata o caricata da file
- setView: metodo utilizzato per settare una griglia di button, utilizzati per la parte grafica
- solve: è il metodo principale, nel quale si dovrà implementare l'algoritmo di backtracking
- update: aggiorna la griglia con la nuova soluzione parziale
- checkConsistency: è utilizzato per verificare la consistenza (riga, colonna, blocco) dell'assegnamento alla variabile corrente
- getStop: metodo utilizzato per conoscere lo stato di esecuzione del programma. Se l'esecuzione viene interrotta, la variabile stop comune a tutti gli algoritmi risolutivi viene settata a true.

Per non appesantire e per velocizzare l'algoritmo di backtracking, si è deciso di utilizzare una rappresentazione matriciale della griglia. I vincoli non vengono memorizzati, ma ogni volta che viene fissato un valore per la cella se ne verifica la consistenza con il metodo checkConsistency. L'unica scelta effettuata a livello di implementazione è stata quella di tracciare con una lista linkata (freeList) la cella (riga, colonna) delle variabili ancora da assegnare. L'inizializzazione della freeList è molto importante poiché indica l' ordinamento delle variabili su cui risolvere.

Il metodo solve è simile per tutti quattro gli algoritmi. L'idea dell'algoritmo è la seguente:

- controllo se ho completato l'assegnamento per tutte le variabili (freeList vuota)
  - se si termina. Il gioco è stato completato
  - se no scegli un valore dal dominio
    - se consistente per la cella corrente assegno il valore alla cella e risolvo sulla prossima variabile non assegnata
    - altrimenti scegli un altro valore dal dominio e ripeti il controllo di consistenza

A questo punto si può verificare la seguente situazione, il dominio contiene solo valori inconsistenti per la cella corrente oppure è vuoto; quindi si azzera la cella ed inizia la fase di backtracking .

Per l'euristica considera il blocco più vincolato (MostConstrainedSquare) il metodo solve è leggermente modificato. La differenza principale consiste nell'inizializzare la freeList in modo dinamico, quindi le variabili su cui risolvere vengono selezionate solo se fanno parte del blocco più vincolato.

Esempio:

```
% randomly generated - fiendish
 . . 9 | . 8 1 | . . .
 . . 2 | 3 . . | 4 . .
 . . . | 2 . . | . 5 9
-------+-------+-------
 . . 8 | 4 . . | 7 . 6
 . 4 . | . . . | . 3 .
 3 . 5 | . . 2 | 8 . .
-------+-------+-------
 7 6 . | . . 5 | . . .
 . . 3 | . . 4 | 1 . .
 . . . | 9 3 . | 2 . .
```

Considerando i blocchi (3x3) per riga, abbiamo la seguente situazione:

- blocchi B2,B4,B6,B8 hanno quattro valori già assegnati
- blocchi B3, B7 hanno tre valori già assegnati
- blocchi B1, B5, B9 hanno due valori già assegnati

Le parità sui blocchi vengono rotte considerandoli per riga, in questo caso l'euristica risolve sul blocco B2. In pratica si cerca di completare i quadrati con il numero minore di celle vuote, per far sorgere prima i conflitti.

Inoltre il metodo solve utilizza due metodi di supporto getSquare e MostConstrainedSquareSelection. Al primo metodo passo riga, colonna della cella e ritorna un numero che indica il blocco mentre al secondo metodo passo l'indice della variabile corrente e aggiorna la lista freeList con le variabili del blocco su cui risolvere.

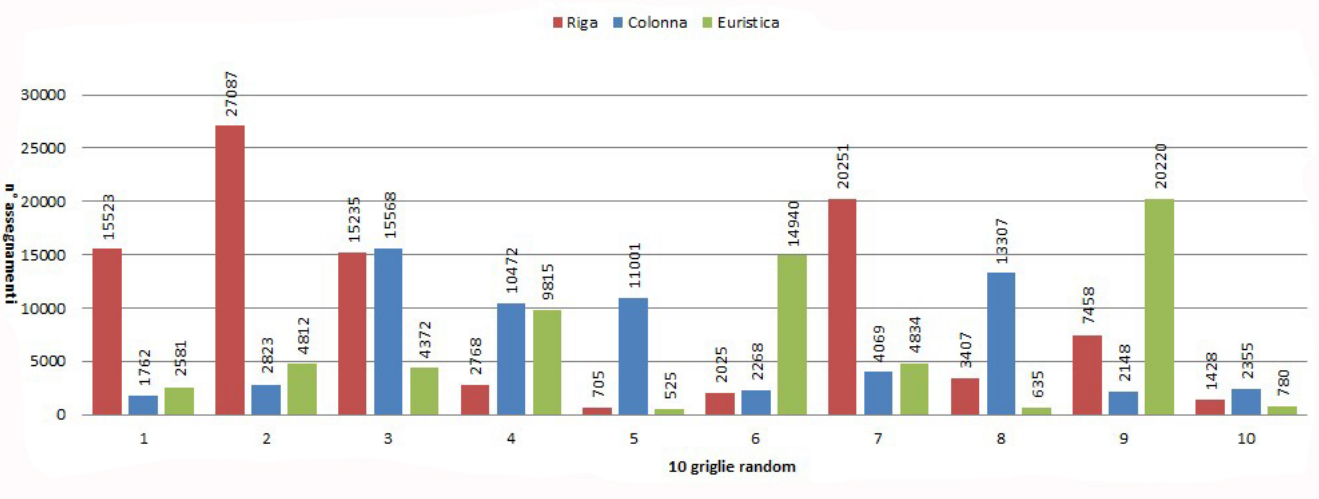
6 Compilazione e esecuzione

La compilazione del programma avviene digitando in un terminale il seguente comando: *javac *.java*
L'esecuzione avviene digitando in un terminale il seguente comando: : *java Gui* .

7 Risultati

Nella raccolta dei risultati non è stata utilizzata la risoluzione random, poiché non comparabile, sia come tempo di esecuzione e sia come numero di assegnamenti agli altri algoritmi. (esecuzione > 20 minuti) Da un'analisi più approfondita sulla risoluzione random è emerso che in media sono necessari 30 assegnamenti sulle variabili prima di effettuare backtracking. In pratica vengono posticipati i conflitti, che comportano una visita in profondità più lunga rispetto agli altri algoritmi.

Di seguito vengono riportati:[2] un grafico che illustra, fissate le 10 griglie random, il comportamento dei tre algoritmi risolutivi (riga, colonna, euristica) rispetto al numero di assegnamenti e una tabella elencante i tempi medi di esecuzione, la media del numero di assegnamenti con i rispettivi intervalli di confidenza.



	Media tempi esecuzione (ms)	Media n°assegnamenti
Riga	379,3 ± 280,4	9588,7 ± 6655,3
Colonna	244,8 ± 133,2	6577,3 ± 3848,7
Euristica	257,5 ± 169,6	6351,4 ± 4760,6

Rispetto all'implementazione, risulta più efficiente la risoluzione per colonna. L'euristica MostConstrainedSquare è migliore della risoluzione per riga, considerando sia i tempi di esecuzione e il numero di assegnamenti.

[2]Piattaforma di testing: Intel Quad 9550@2.83 GHz