Alessandro Sestini - 5453044

RELAZIONE

INTRODUZIONE

In questo esercizio implemento la strategia di Backtracking per problemi di soddisfacimento di vincoli congiuntamente alla tecnica MAC (Mantaining Arc Consistency) per la propagazione di vincoli.
Gli algoritmi sono realizzati attraverso il linguaggio Python (per informazioni su come avviare il programma fare riferimento al file di testo readme).
Di seguito verranno discussi gli elementi fondamentali dei vari algoritmi in relazione all'implementazione concreta.

PROBLEMI CSP

Un CSP consiste in 3 elementi fondamentali:

- Variabili: una lista di variabili [X,Y,Z,…];
- Domini: un dizionario che ha come chiavi ogni variabile e come oggetti una lista di valori che la chiave può assumere {X:[1,2,3],Y:[1,2,3],Z:[1,2,3],…};
- Vincoli: una funzione (A,a,B,b) che restituisce TRUE se i vicini A e B soddisfano il vincolo quando A=a e B=b (questa definizione di vincolo è la stessa usata in http://aima.cs.berkeley.edu/python/readme.html che rende il vincolo più semplice di un insieme di valori);

Un altro elemento fondamentali per un CSP sono i:

- Vicini: un dizionario che ha come chiavi ogni variabile e come oggetti una lista di altre variabili che sono coinvolte con la prima in un qualche vincolo {X:[Y,Z],Y:[X,Z],…}

In questi problemi uno stato è un'assegnazione di valori a delle variabili.
Una variabile viene assegnata tramite una funzione assign(var, val, assignment), che riduce il dominio della variabile di riferimento ad un solo valore (quello dell'assegnamento). Questa procedura rende più semplice ed efficiente la Constraint Propagation.
Uno stato può essere:

- Consistente, se le assegnazioni soddisfano i vincoli;
- Completo, se tutte le variabili sono assegnate;

se uno stato è sia consistente che completo, allora è una soluzione.

PROPAGAZIONE DI VINCOLI

ARC CONSISTENCY: una variabile Xi è AC rispetto ad una variabile Xj se per ogni valore nel dominio[Xi] esiste un valore nel dominio[Xj] che soddisfa il vincolo tra Xi e Xj.

Con questa definizione è possibile effettuare dell'inferenza a priori sui domini delle variabili, tramite gli algoritmi AC-3 e REVISE (descritte nel file Python).
Si forma una coda di tutti gli archi del problema e per ogni arco (Xi,Xj), si controllano i valori del domino: se per ogni valore x del dominio di Xi non esiste nessun valore y del dominio di Xj che soddisfa il vincolo(Xi,x,Xj,y), allora si elimina x dal dominio[Xi]; se un dominio rimane vuoto, vuol dire che il problema è insoddisfacibile. Una volta che è stata fatta inferenza sul dominio di Xi, devo aggiungere nuovamente alla coda tutti gli archi dei vicini di Xi che potrebbero essere diventati

inconsistenti. L'algoritmo AC-3 cerca di risolvere il problema prima di fare backtracking: se trova dei domini che hanno 1 solo valore, allora si assegna quel valore alla variabile di riferimento. Se l'assegnamento risulta completo, allora è una soluzione.

BACKTRACKING SEARCH

Molti problemi (ad esempio giochi come il Sudoku o N-Queens trattati in questo esercizio) sono impossibili da risolvere solamente tramite la Constraint Propagation, ma è necessaria una ricerca.
Il Backtracking è esattamente come una ricerca in profondità ma fissando degli ordinamenti: scelgo una variabile da un particolare ordine, gli assegno un valore da un particolare ordine e a quel punto faccio Constraint Propagation. Se l'inferenza mi porta un dominio vuoto, allora ho sbagliato scelta e devo fare un passo indietro nell'albero di ricerca; se trovo un assegnamento non consistente, anche in questo caso la scelta è sbagliata e devo tornare indietro nell'albero; se ogni passo mi porta ad un dead end, allora il problema è insoddisfacibile.
L'algoritmo Backtracking è un algoritmo ricorsivo:
in ingresso viene passato un csp e un assegnamento. Se l'assegnamento è completo, allora è una soluzione, altrimenti sceglie una variabile X dall'ordine prestabilito. In seguito, per ogni valore x del dominio ordinato, copia in una variabile temporanea i vari domini del csp. Se X = x è consistente con l'assegnamento, allora si aggiunge a quest'ultimo e si fa inferenza, altrimenti scelgo un altro valore. L'inferenza restituisce o delle assegnazioni da aggiungere oppure un "Failure": se ottengo un "Failure", allora vuol dire che la scelta è sbagliata e dobbiamo fare un passo indietro nell'albero, si ristabiliscono quindi i domini salvati nella variabile temporanea, le assegnazioni (sia quelle fatte dal Backtrack sia quelle fatte dall'inferenza) e si sceglie un nuovo valore per la variabile X. Se tutti i valori del dominio di X restituiscono "Failure", allora dobbiamo fare un passo indietro e riassegnare la variabile utilizzata al passo precedente. Si ripetono questi passi fino ad avere un assegnamento completo e consistente, che rappresenta la soluzione. Se nessuno degli assegnamenti risulta essere completo e consistente, allora il problema è insoddisfacibile.

E' quindi importante scegliere:

1. qual è l'ordine con cui scelgo le variabili:
   in questo esercizio si implementa la tecnica MRV, Minimum Remaining Values, con la quale si sceglie la variabile con il minor numero di valori legali;
2. qual è l'ordine con cui scelgo i domini:
   in questo esercizio si implementa un algoritmo che mi renda i domini delle varie variabili in un ordine casuale, di modo da far vedere più soluzioni dove ne esistono di più;
3. qual è l'inferenza che devo attuare:
   l'inferenza è un elemento ancora più potente se viene attuata durante la ricerca (anziché solamente all'inizio). La tecnica implementata in questo esercizio è la tecnica MAC: Mantaining Arc Consistency. L'algoritmo, dopo aver assegnato una variabile Xi, chiama l'AC-3 ma, invece che una coda di tutti gli archi del problema, si inizia con quelli vicini a Xi. In seguito il MAC continua con il normale svolgimento dell'AC-3 propagando l'inferenza per tutti i domini.

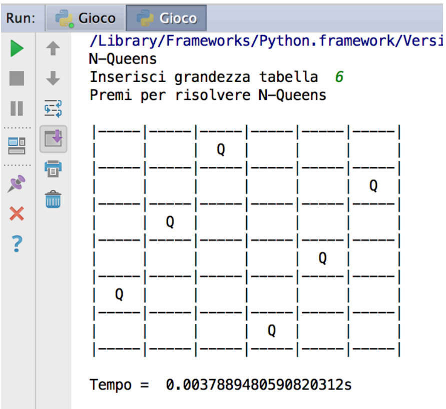
ESEMPI PRATICI, I GIOCHI N-QUEENS E SUDOKU

N-QUEENS

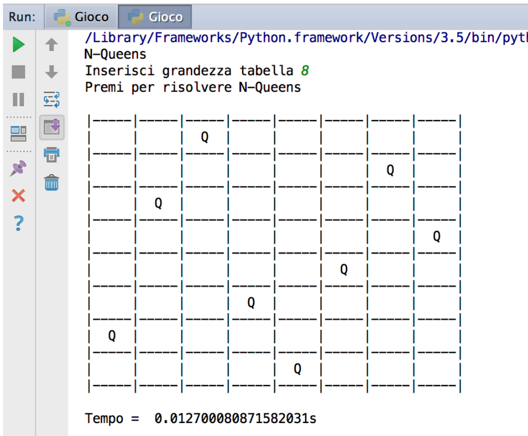
Per implementare il gioco delle N-Queens ho optato per i seguenti elementi:

- Variabili: le colonne, formate da tuple (X,i) dove i è il numero della colonna che va da 1 a N;
- Domini: numeri che rappresentano le righe, dunque per ogni variabile/colonna avrò una lista di interi che va da 1 a N;
- Vincolo: una funzione che presi due vicini (X,i) = a e (X,j) = b, controlla se hanno lo stesso valore (quindi sono sulla stessa riga) oppure controlla se sono sulla stessa diagonale superiore o inferiore, tramite questa formula: $b \neq a \pm |i - j|$

Es: N = 6



N = 8



SUDOKU

Per implementare il gioco del sudoku, ho optato per i seguenti elementi:
Creo la tabella 9x9, identificano 81 variabili. Ogni variabile è una tupla con una lettera e un numero che identificano rispettivamente la riga e la colonna. Inoltre viene anche inizializzato:
row = una lista di liste, ogni sottolista identifica l'insieme delle variabili contentute in una riga;
col = una lista di liste, ogni sottolista identifica l'insieme delle variabili contentute in una colonna;
box = una lista di liste, ogni sottolista identifica l'insieme delle variabili contenute in un box;

```
          c c c    c c c    c c c
          o o o    o o o    o o o
          l l l    l l l    l l l
          1 2 3    4 5 6    7 8 9

          1 2 3 |  4 5 6 |  7 8 9
row1    A       |        |
row2    B  box1 |   box2 |   box3
row3    C       |        |
        ------------------------
row4    D       |        |
row5    E  box4 |   box5 |  box6
row6    F       |        |
        ------------------------
row7    G       |        |
row8    H  box7 |   box8 |  box9
row9    I       |        |
```

Il dominio di ogni variabile è una lista di numeri da 1 a 9.

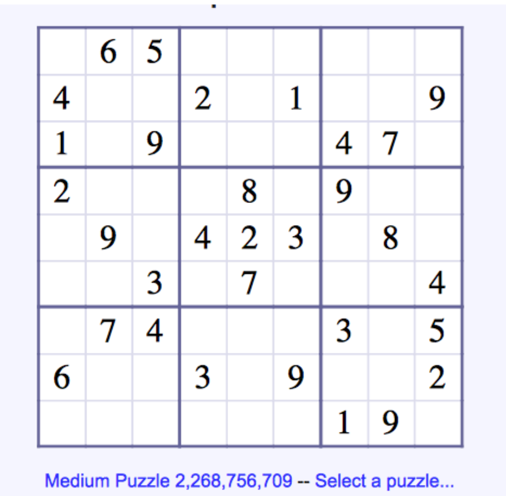
Il vincolo è una funzione(A,a,B,b) che presi due vicini A = a e B = b controlla se hanno valori uguali: $a \neq b$

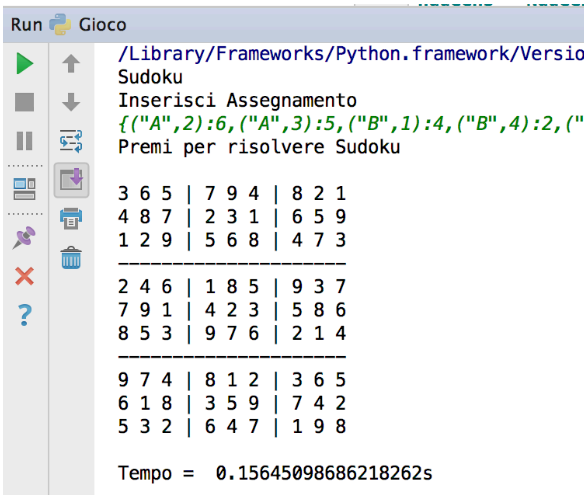
Per quanto riguarda i vicini, per ogni variabile X, si aggiunge una chiave X al dizionario dei vicini e, come oggetti relativi alla chiave, una lista di tutte le variabili che fanno parte della stessa riga, colonna e box di X.

Il Sudoku prevede anche un assegnamento iniziale. Dato questo assegnamento, prima di avviare la ricerca si applica l'inferenza tramite l'AC-3 e si cerca di eliminare i domini prima di effettuare il Backtrack. È possibile che l'AC-3 iniziale porti fin da subito la soluzione. (Esempi presi da http://www.websudoku.com/)

Es: "Medio"

{("A",2):6,("A",3):5,("B",1):4,("B",4):2,("B",6):1,("B",9):9,("C",1):1,("C",3):9,("C",7):4,("C",8):7,("D",1):2,("D",5):8,("D",7):9,("E",2):9,("E",4):4,("E",5):2,("E",6):3,("E",8):8,("F",3):3,("F",5):7,("F",9):4,("G",2):7,("G",3):4,("G",7):3,("G",9):5,("H",1):6,("H",4):3,("H",6):9,("H",9):2,("I",7):1,("I",8):9}





"Evil"

{("A",1):6,("A",2):1,("A",9):2,("B",5):7,("B",7):9,("B",8):4,("C",3):3,("C",6):9,("C",8):6,("D",1):7,("D",6):8,("D",7):5,("F",3):5,("F",4):6,("F",9):8,("G",2):4,("G",4):1,("G",7):2,("H",2):6,("H",3):8,("H",5):2,("I",1):2,("I",8):3,("I",9):4}