Report di progetto - SudokuWhiz

DOROTEA SERRELLI and RAFFAELLA SABATINO

In questo report si descrive come si è pensato di costruire un agente intelligente capace di risolvere il gioco del Sudoku in modo corretto ed efficiente nel tempo, utilizzando i seguenti algoritmi di ricerca: backtracking, ricerca A*, simulated annealing, algoritmi genetici generazionali.

Per poter osservare il comportamento dell'agente nella risoluzione del gioco, si è realizzato un progetto Java che, mediante interfaccia grafica, legge il file .txt contenente la griglia Sudoku inserita dall'utente e fornisce la griglia Sudoku risultante, risolvendola con l'algoritmo di ricerca scelto dall'utente.

Si riporta, dunque, il link al repository Github per la demo: https://github.com/SabatinoRaffaella/SudokuWhiz.

Gli algoritmi di ricerca precedentemente citati sono stati descritti nel loro meccanismo di funzionamento, nel loro adattamento al problema, nella loro complessità, confrontandoli tra loro in termini di prestazioni: i nodi visitati durante la ricerca, i nodi visitati utili per risolvere la griglia Sudoku, il tempo di risoluzione.

Si precisa che l'applicazione Java è stata eseguita su un terminale avente le seguenti caratteristiche:

- Processore: 3.10GHz
- RAM: 16GB

Additional Key Words and Phrases: Sudoku, applicazione Java, Backtracking, Ricerca A*, Simulated annealing, Algoritmi genetici generazionali

ACM Reference Format:

Dorotea Serrelli and Raffaella Sabatino. 2024. Report di progetto - SudokuWhiz. 1, 1 (January 2024), 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' address: Dorotea Serrelli; Raffaella Sabatino.

ACM XXXX-XXXX/2024/1-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Indice

Comprensione del problema e dell'ambiente

Comprensione del problema e dell'ambiente

1 INTRODUZIONE

1.1 Il gioco Sudoku

Il Sudoku è un gioco di logica nel quale viene proposta una griglia di 9×9 celle, ciascuna delle quali può contenere un numero da 1 a 9, oppure essere vuota; la griglia è suddivisa in 9 righe orizzontali, 9 colonne verticali e in 9 "sottogriglie" di 3×3 celle contigue. Queste sottogriglie sono delimitate da bordi in neretto e chiamate *regioni*. Le griglie proposte al giocatore, in generale, hanno da 20 a 35 celle contenenti un numero.

Il gioco fu inventato dal matematico svizzero Eulero da Basilea (1707-1783). La versione moderna del gioco fu pubblicata per la prima volta nel 1979 dall'architetto statunitense Howard Garns all'interno del *Dell Magazines* con il titolo *Number Place*. In seguito fu diffuso in Giappone dalla casa editrice *Nikoli* nel 1984, per poi diventare noto a livello internazionale soltanto a partire dal 2005, quando fu proposto in molti periodici.

Lo scopo del gioco è quello di riempire le caselle bianche con numeri da 1 a 9 in modo tale che in ogni riga, in ogni colonna e in ogni regione siano presenti tutte le cifre da 1 a 9 senza ripetizioni.

1.2 Obiettivi del progetto

In questo progetto si intende costruire un agente intelligente in grado di risolvere il gioco in modo corretto e nel minor tempo possibile.

La soluzione del gioco, ovvero una griglia Sudoku completamente riempita e che rispetta le regole del gioco, verrà determinata applicando i seguenti algoritmi di ricerca: backtracking, ricerca A*, simulated annealing, algoritmi genetici generazionali.

2 DESCRIZIONE DELL'AMBIENTE

2.1 Osservazioni preliminari per la formulazione della specifica PEAS dell'ambiente

Il gioco si svolge in matrici di aspetto 9×9 (le griglie), denotate *matrici Sudoku*, le cui caselle possono contenere un intero da 1 a 9 oppure il valore 0, per denotare la casella bianca o vuota.

Una matrice Sudoku M è suddivisa in 9 blocchi di aspetto 3×3, denotati $B_{\mathrm{h,k}}$ con $h, k = 1, 2, 3$; il blocco $B_{\mathrm{h,k}}$ riguarda, per la matrice M, le righe relative agli indici $3h - 2$, $3h - 1$ e $3h$ e le colonne relative agli indici $3k - 2$, $3k - 1$ e $3k$. In ogni riga, colonna e regione di una matrice Sudoku i valori interi da 1 a 9 non possono essere ripetuti.

La griglia Sudoku proposta o *matrice Sudoku incompleta* è una matrice Sudoku che presenta alcune celle bianche.

Lo scopo del gioco è trasformare la griglia proposta in una matrice completa, cioè in una matrice priva di celle bianche e, quindi, tale che in ogni sua riga, colonna e regione compaiano tutti i numeri dell'insieme $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, ciascuno una sola volta.

Affinché una matrice incompleta sia considerata valida ai fini del gioco, è necessario che la soluzione sia univoca, ovvero non devono sussistere due o più soluzioni differenti. Risulta cruciale, dunque, definire il numero minimo di indizi presenti nella matrice Sudoku, ovvero il numero minimo di valori presenti all'inizio nella griglia Sudoku.

Il numero massimo di indizi di partenza non conta; tuttavia, se ce ne sono troppi il gioco diventa banale e si riduce ad una procedura meccanica di riempimento senza che occorra alcun procedimento logico.

Pertanto, si è deciso di considerare griglie Sudoku che avessero al massimo 30 indizi.

La quantità minima di numeri di partenza è, invece, importante perché, sotto una certa soglia, il rompicapo diventa impossibile da risolvere; infatti, si rischia che la matrice Sudoku ammetta più di una soluzione e ciò la renderebbe non valida (ed esteticamente poco interessante).

Per definire questo valore, ci si basa su uno studio della rivista *Nature* del 2013, condotto da un gruppo di matematici diretto da Gary McGuire dell'Università di Dublino. Tale gruppo di ricerca ha dimostrato che il numero minimo di indizi necessari per risolvere una griglia Sudoku 9×9 è 17; infatti, con meno indizi è impossibile riempire univocamente la griglia del gioco su cui giornalmente milioni di persone si cimentano.

Di seguito si riporta il link al nostro repository di Github per visionare l'abstract Solving the Sudoku Minimum: https://github.com/SabatinoRaffaella/SudokuWhiz/blob/main/Documents/Solving%20the%20Sudoku%20Minimum_abstract.pdf.

Pertanto, per garantire che il Sudoku abbia una sola soluzione, si considereranno griglie Sudoku aventi almeno 17 indizi.

2.2 Specifica PEAS dell'ambiente

Tenendo conto delle osservazioni preliminari fatte nel paragrafo precedente, di seguito si delinea la specifica PEAS dell'ambiente in cui opererà l'agente:

- **Performance**

  La misura delle prestazioni è la capacità dell'agente di risolvere una griglia Sudoku in modo corretto ed efficiente nel tempo, trovando una sola soluzione.

- **Environment**

  L'ambiente è rappresentato dalla griglia Sudoku stessa, che è costituita da una griglia 9×9 parzialmente riempita, divisa in 9 regioni 3×3. L'ambiente include anche le celle vuote — denotate con il valore 0 – da riempire con numeri da 1 a 9, rispettando le regole del gioco. Nel paragrafo successivo si descriveranno le caratteristiche dell'ambiente.

- **Actuators**

  Si intendono gli attuatori dell'agente disponibili per compiere delle azioni, al fine di alterare l'ambiente: inserimento di numeri nelle celle vuote della griglia; cancellazione di un numero inserito dall'agente; spostamento in una nuova cella; aggiornamento della rappresentazione interna della griglia, a seguito di una modifica alla configurazione della griglia; valutazione della griglia Sudoku in relazione con le regole del gioco.

- **Sensors**

  I sensori dell'agente sono utilizzati per percepire lo stato corrente della griglia Sudoku, ossia per rilevare i numeri già presenti nella griglia e quelli che l'agente può inserire in modo legale nelle celle vuote.

2.2.1 Caratteristiche dell'ambiente. L'agente si interfaccia in un ambiente con le seguenti caratteristiche:

- **completamente osservabile**: l'agente è a conoscenza in ogni istante della configurazione della griglia del Sudoku;
- **singolo agente**: l'unico agente che opera in questo ambiente è quello in oggetto;
- **deterministico**: la configurazione della griglia Sudoku M_j è il risultato solo e soltanto dell'azione dell'agente eseguita sulla configurazione M_i della griglia Sudoku corrente;
- **sequenziale**: la scelta dell'azione dell'agente sulla configurazione della griglia Sudoku corrente dipende dalle azioni fatte nelle configurazioni precedenti (inserimento/cancellazione di un numero, ...);
- **statico**: durante l'esecuzione dell'agente, la griglia Sudoku non muta mentre l'agente pensa alla prossima azione;
- **discreto**: in ogni configurazione della griglia Sudoku c'è un insieme finito di percezioni ed azioni; infatti, la griglia è formata da un numero di celle finito, ciascuna con un numero discreto (da 0 a 9).

3 FORMULAZIONE DEL PROBLEMA

Di seguito si riporta l'analisi del problema da risolvere:

- **Stato iniziale**

  Lo stato iniziale corrisponde alla configurazione iniziale della griglia 9×9, parzialmente riempita con almeno 17 indizi e massimo 30 indizi, è definita in un file di estensione .txt.
  Ogni riga del file è una riga della griglia 9×9 ed i numeri presenti in una riga sono separati l'uno dall'altro mediante uno spazio. Le caselle vuote sono indicate con il valore 0.

- **Azioni**

  Si elencano di seguito le azioni possibili per l'agente.
  - Inserimento di numeri da 1 a 9 nelle celle vuote, rispettando le regole del gioco.
  - Lettura di un numero in una cella;
  - Cancellazione di un numero inserito in una cella (ad eccezione delle celle definite nel file, contenenti gli indizi);
  - Spostamento verso una nuova cella.

- **Modello di transizione**

Il modello di transizione descrive come lo stato della matrice Sudoku cambia in risposta alle azioni eseguite dall'agente.
Se l'agente inserisce un numero legale x in una cella vuota y (con $1 \leq x \leq 9$), la matrice Sudoku corrente transita ad un nuovo stato, ossia la matrice Sudoku risultante avrà la cella y contenente il valore x. Il valore x scelto non sarà un numero già presente nella riga, nella colonna e nella regione in cui si trova la cella y.
Se l'agente rimuove un numero x che ha precedentemente inserito in una cella y, la matrice Sudoku corrente transita ad un nuovo stato: la matrice Sudoku risultante avrà la cella y priva del valore x, ed è denotata con il valore 0.
Se l'agente decide di leggere il valore di una cella, la configurazione della matrice Sudoku non transita in nuovo stato.
Se l'agente decide di spostarsi in una nuova cella, la configurazione della matrice Sudoku non transita in nuovo stato, bensì cambia la cella che l'agente deve considerare.

- **Test-obiettivo**

  Il test-obiettivo consiste nel verificare se il Sudoku è stato risolto correttamente e in modo efficiente nel tempo, ovvero se tutte le regole del gioco sono state rispettate.

- **Costo di cammino**

  Il costo di cammino è definito come il costo dell'esecuzione di una o più azioni necessarie per risolvere il Sudoku. Si suppone che la lettura di un valore della cella e lo spostamento da una cella all'altra hanno un costo pari a 1, mentre le altre operazioni hanno un costo pari a 2.

Algoritmi di ricerca

Algoritmi di ricerca

4 ALGORITMI DI RICERCA ANALIZZATI

Il problema della risoluzione di una griglia Sudoku parzialmente riempita è stato affrontato guardandolo da diverse prospettive.

Innanzitutto, si è guardato al primo possibile approccio per risolvere il gioco, applicato da un giocatore naïve: il backtracking. Successivamente, si è pensato ad una possibile strategia che può utilizzare un giocatore che pratica regolarmente il gioco, però senza aver raggiunto il livello di esperto: la ricerca informata A*.

I due succitati algoritmi di ricerca tradizionale si basano sulla formulazione del problema di tipo incrementale: essi adottano una strategia che mira a riempire, progressivamente, la griglia incompleta 9×9 fornita dall'utente.

Tale strategia prevede di visitare un albero di ricerca, rappresentazione dello Spazio degli stati del problema, al fine di ottenere il cammino che porta verso la soluzione.

D'altra parte, si è pensato di utilizzare per il problema in esame anche algoritmi di ricerca locale, al fine di fornire solamente la configurazione finale della griglia data dall'utente, piuttosto che esplorare in modo sistematico lo spazio di ricerca.

Pertanto, nel documento si sono analizzati per la risoluzione di una griglia Sudoku l'algoritmo Simulated Annealing e gli Algoritmi Genetici Generazionali.

È da precisare che, per motivi di efficienza e di efficacia, si è voluto fornire un'interpretazione del problema a stato completo per quest'ultimi algoritmi: a partire dalla griglia Sudoku 9×9 fornita dall'utente, le celle bianche vengono riempite con valori casuali, presi dall'insieme $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Dinanzi a questa configurazione della griglia, l'agente dovrà adottare una strategia per trasformare la griglia Sudoku casualmente riempita in una soluzione valida per le regole del gioco.

5 BACKTRACKING

5.1 Meccanismo di funzionamento

Il backtracking è un algoritmo utile per risolvere problemi con la ricorsione, costruendo una soluzione incrementalmente.

In generale, il backtracking intende espandere una soluzione parziale $s = (a_1, a_2, a_3, \ldots, a_k)$, dove l'elemento a_i viene scelto da un insieme ordinato S_i di possibili candidati per la posizione i.

L'espansione della soluzione s consiste nel costruire, a partire da s, l'insieme S_{k+1} dei possibili candidati per la posizione $k + 1$ e scegliere un elemento $a_{k+1} \in S_{k+1}$. Fin quando l'estensione genera una soluzione parziale, si continua ad estenderla.

Se l'insieme dei possibili candidati S_{k+1} per la posizione $k + 1$ della soluzione parziale s è vuoto, significa che non c'è la possibilità di estendere la soluzione corrente. Dunque, è necessario ritornare indietro e sostituire l'ultimo elemento a_k nella soluzione s con un altro candidato dell'insieme S_k, con $a_{k'} \neq a_k, a_{k'} \in S_k$.

5.2 Il backtracking in SudokuWhiz

Per il nostro gioco, la soluzione parziale che si intende espandere è la griglia 9×9 del Sudoku, che si suppone riempita fino alla cella posta nella riga i e colonna j ($0 \leq i < 8, 0 \leq j < 8$), denotata con $c_{i,\,j}$.

L'agente si trova nella cella $c_{i,\,j+1}$ e sta valutando l'insieme dei possibili candidati per questa cella, ossia i numeri da 1 a 9.

Si intende, quindi, inserire in modo iterativo i numeri da 1 a 9 uno per uno: se il numero inserito soddisfa tutte e 3 le condizioni, cioè quel particolare numero non è presente nella riga, colonna e regione della cella $c_{i,\,j+1}$, si conferma tale inserimento.

Poi, si passa alla cella vuota successiva e si compie la stessa iterazione dei numeri ma con la matrice Sudoku aggiornata. Man mano che si procede con la nuova cella vuota, si potrebbe raggiungere una cella, denotata con $c_{h,\,k}$ ($i \leq h \leq 8, j + 1 < k \leq 8$) in cui non si può inserire nessun numero dell'insieme $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Quindi, la soluzione costruita fino adesso viene respinta e si torna indietro, provando nuovi valori possibili in quelle celle precedenti. Poiché si assume, per configurazione iniziale della matrice Sudoku, che una matrice Sudoku ha solo una soluzione, si continuerà a respingere una o più soluzioni costruite fino a quando non si trova quella che soddisfa tutte le condizioni per ogni cella inizialmente vuota.

Per questo problema, l'algoritmo di backtracking cercherà di posizionare ogni numero in ogni riga e colonna vuota fino a quando non è risolto.

5.3 Complessità dell'algoritmo backtracking

Nell'algoritmo di backtracking viene costruito un albero delle soluzioni per il problema, dove ciascun nodo interno x è una soluzione parziale e l'arco tra il nodo x e il nodo y nell'albero viene creato se il vertice y è frutto dell'estensione della soluzione parziale x. Le foglie dell'albero sono le soluzioni.

Mediante la verifica del soddisfacimento delle condizioni di unicità per riga, colonna e regione, si effettua una potatura delle soluzioni che non soddisfano le condizioni richieste.

Il backtracking corrisponde ad effettuare una visita in profondità dell'albero delle soluzioni poiché l'intento è quello di trasformare la griglia Sudoku all'inizio incompleta (rappresentata dal nodo radice dell'albero) in una griglia completa (rappresentata da una foglia dell'albero).

L'algoritmo risulta essere completo poiché in modo esaustivo esplora tutte le possibili combinazioni e, per via delle assunzioni fatte sulla configurazione iniziale della matrice Sudoku, troverà una soluzione.

L'algoritmo, inoltre, risulta essere ottimo perché riesce a determinare la soluzione ottima, ma non in termini di efficienza; infatti, la complessità temporale richiesta è esponenziale $O(9^{n \times n})$ poiché, per ogni cella, si hanno a disposizione 9 valori su cui scegliere – anche se si debbano fare delle scelte nel caso peggiore per $81 - 17 = 64$ caselle affinché il Sudoku abbia una sola soluzione – .

La complessità spaziale richiesta è pari a $O(n^2)$ per tenere traccia di una matrice Sudoku di dimensione $n \times n$; infatti, il backtracking mantiene soltanto una singola rappresentazione di uno stato e altera tale rappresentazione anziché crearne di nuove.

Si riporta in linguaggio Java uno screenshot relativo all'algoritmo di backtracking. ...

Nella seguente tabella si riportano i tentativi fatti per diverse tipologie di Sudoku: nel caso peggiore (17 indizi iniziali), medio (24 indizi iniziali), facile (30 indizi iniziali). Le griglie Sudoku realizzate per fare il test si trovano nel package Griglie_test del progetto Java chiamato SudokuWhiz.

Livello di difficoltà	MNE	MNS	Tempo medio per trovare una soluzione
Caso peggiore	210.969.538	197	3, 6717
Medio	1.775.908	174	0, 0961
Facile	70.154	125	0, 0201

Tabella 1. Prestazioni del backtraing su diversi livelli di difficoltà del gioco

È da sottolineare che, sottoponendo l'agente alla risoluzione delle diverse griglie Sudoku 9×9, si è notato che anche la diversa configurazione degli indizi nella griglia ha un forte impatto sul tempo di ricerca della soluzione e sul numero di nodi da esplorare.

Come esempio di quanto appena affermato, si riportano i dati della tabella relativi alle griglie Sudoku_casoPeggiore_g e Sudoku_casoPeggiore2_g.

Griglia	MNE	MNS	Tempo medio per trovare una soluzione
Sudoku_casoPeggiore	37.652	103	0, 0226
Sudoku_casoPeggiore2	622.577.597	259	7, 3376941

Tabella 2. Prestazioni del backtraing sulle griglie Sudoku_casoPeggiore e Sudoku_casoPeggiore2

6 RICERCA A*

6.1 Meccanismo di funzionamento

La ricerca A* è un algoritmo di ricerca informata che sfrutta la conoscenza specifica del dominio del problema per fornire suggerimenti su dove si potrebbe trovare l'obiettivo. I suggerimenti hanno la forma di una funzione euristica, denotata con $h(n)$, intesa come stima del costo del cammino meno costoso dallo stato corrente (associato al nodo n dell'albero di ricerca) ad uno stato obiettivo.

L'algoritmo di ricerca A* procede espandendo il nodo avente il valore minimo della funzione di valutazione $f(n)$:

$f(n)$ = costo stimato del cammino migliore che continua da n fino ad un obiettivo = $g(n)$ + $h(n)$

dove $g(n)$ è il costo del cammino per raggiungere il nodo n a partire dalla radice, e $h(n)$ è il costo stimato dall'algoritmo per raggiungere lo stato obiettivo a partire dal nodo n.

6.2 La ricerca A* in SudokuWhiz

Nella formulazione del nostro problema, lo stato obiettivo è rappresentato dalla matrice completa Sudoku, ovvero la griglia 9×9 avente numero di caselle bianche pari a 0 e che rispetta le regole del gioco.

La funzione $g(n)$, in questo caso, determina il costo di cammino dalla configurazione iniziale della griglia Sudoku alla configurazione corrente M_j (rappresentata dal nodo n) sommando il costo delle operazioni effettuate per raggiungere il nodo n, come riportato nella formulazione del problema.

La funzione euristica $h(n)$, banalmente, è interpretabile come il numero di caselle bianche che devono essere ancora riempite nella griglia corrente M_j; infatti, la soluzione al Sudoku è una griglia 9×9 priva di celle bianche.

A differenza dell'algoritmo di backtracking, si è pensato di rendere più edotto l'agente sulla risoluzione del gioco.

In generale, se in una riga o in una colonna ci sono pochissime celle bianche, vuol dire che l'insieme dei valori assegnabili in una di queste caselle bianche è ristretto rispetto all'insieme $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Nello specifico, se l'agente inserisse correttamente un numero in una casella bianca, diminuirebbe il numero di valori da considerare per le celle bianche rimanenti della stessa riga e della stessa colonna. In questo modo, l'agente sarebbe stimolato a completare la riga/colonna in questione.

Questa intuizione può essere formulata come una funzione euristica $h_2(n)$, ausiliare ad $h(n)$ che valuta quale cella della riga o colonna corrente possiede il minor numero di alternative assegnabili, in modo da favorire l'inserimento di un valore valido. Una volta effettuato l'inserimento, il valore assegnato non verrà considerato nell'insieme delle possibili scelte per le celle che si trovano sulla stessa riga, colonna e regione.

Ricapitolando, ad ogni stadio l'algoritmo A* cerca di prendere la decisione (eseguire il prossimo inserimento di un numero) che minimizza la distanza rimanente dall'obiettivo (numero di celle vuote), garantendo al contempo che le regole del Sudoku siano soddisfatte.

Una coda prioritaria viene utilizzata per gestire i passaggi successivi possibili ed un insieme assicura che uno stato della griglia non venga visitato due volte.

Ogni volta che viene calcolata una decisione su dove effettuare il prossimo inserimento di un valore sulla griglia, viene scelta la cella, appartenente alla riga o alla colonna dell'ultima cella riempita,

che ha il minor numero di valori assegnabili, al fine di massimizzare la probabilità che la decisione porti ad un posizionamento corretto.

Le principali strutture dati utilizzate sono:

- Matrice $n \times n$: utilizzata per memorizzare la griglia Sudoku (matrice 9×9).
- Min-coda a priorità per gli stati successori: utilizzata per mantenere gli stati successori ed ordinarli in base al valore della funzione di valutazione $f(n) = g(n)+h(n)+h_2(n)$. La dimensione di questa struttura dati sarà influenzata dal numero massimo di stati successori che possono essere generati in una singola espansione.
- Insieme per gli stati visitati: tiene traccia degli stati visitati durante la ricerca.

6.3 Complessità della ricerca A*

L'algoritmo di ricerca A* risulta essere ottimo in quanto le due funzioni euristiche $h(n)$ e $h_2(n)$ sono entrambe ammissibili e consistenti.

L'ammissibilità deriva dal fatto che le euristiche non sbagliano mai per eccesso la stima del costo per arrivare all'obbiettivo e nel numero di valori ammissibili per ogni cella in un dato istante.

La consistenza esiste perché, nel caso di $h(n)$, per ogni nodo n e ogni successore n', il costo stimato per raggiungere l'obbiettivo partendo da n non è superiore al costo di passo per arrivare a n' sommato al costo stimato per andare da lì all'obbiettivo.

In modo analogo, la funzione euristica $h_2(n)$, la quale valuta il numero di scelte rimanenti per riempire le celle vuote in una data riga o colonna, è anch'essa consistente. Poiché la distanza reale d (costo del cammino) tra due stati successivi nel Sudoku è almeno 1, poiché riempire una cella richiede almeno un passo, la funzione euristica $h_2(n)$ è consistente.

Nel dettaglio, se si considerano due stati A e B, dove B è ottenuto da A riempiendo una cella, $h_2(A)$ rappresenta il numero di opzioni rimanenti nella riga o colonna dell'ultima cella riempita in A, mentre $d(A, B)$ sarà almeno 1, poiché è necessario almeno un passo per raggiungere B da A. Pertanto, $h_2(A) \leq d(A, B)$ come richiesto per la consistenza.

La ricerca A* risulta, poi, essere completa: esiste un numero finito di nodi di costo minore o uguale a C* (il costo di cammino della soluzione ottima).

La complessità temporale della ricerca A* è $O(b^d)$ dove b è il fattore di ramificazione e d è la profondità della soluzione ottima C*. Nel caso del Sudoku, b dipenderà dal numero di scelte possibili in ogni cella vuota, mentre d sarà la profondità della soluzione ottima. Lo svantaggio di questo algoritmo sta nel fatto che il numero di nodi all'interno dello spazio di ricerca del confine dell'obbiettivo cresce esponenzialmente con la lunghezza della soluzione.

Si riporta il codice dell'algoritmo A* scritto in Java ....

Nella seguente tabella si riportano i tentativi fatti per diverse tipologie di Sudoku: nel caso peggiore (17 indizi iniziali), medio (24 indizi iniziali), facile (30 indizi iniziali). Le griglie Sudoku realizzate per fare il test si trovano nel package Griglie_test del progetto Java SudokuWhizWhiz.

Livello di difficoltà	MNG	MNS	Tempo medio per trovare una soluzione
Caso peggiore	270.812	135.402	0, 5736
Medio	3.261	1.627	0, 048
Facile	227	113	0, 0243

Tabella 3. Prestazioni della ricerca A* su diversi livelli di difficoltà del gioco

Così come è stato notato per l'algoritmo backtracking, si è visto che sottoponendo l'agente alla risoluzione delle diverse griglie Sudoku 9×9, la diversa configurazione degli indizi iniziali nella griglia ha un forte impatto sul tempo di ricerca della soluzione e sul numero di nodi da esplorare.

Come esempio di quanto appena affermato si riportano i dati della tabella relativi sulle griglie Sudoku_casoPeggiore_g e Sudoku_casoPeggiore2_g.

Griglia	MNG	MNS	Tempo medio per trovare una soluzione
Sudoku_casoPeggiore	187	93	0, 0363
Sudoku_casoPeggiore2	421.085	210.535	0, 8613

Tabella 4. Prestazioni della ricerca A* sulle due griglie Sudoku_casoPeggiore e Sudoku_casoPeggiore2 con livello di difficoltà *Caso peggiore*

GLOSSARIO

MNE acronimo di numero Medio dei Nodi Esplorati dall'algoritmo di ricerca durante la sua esecuzione . 9

MNG acronimo di numero Medio dei Nodi Generati dall'algoritmo di ricerca A* durante la sua esecuzione . 12

MNS acronimo di numero Medio dei Nodi Soluzione esplorati dall'algoritmo di ricerca durante la sua esecuzione per determinare la soluzione. 9, 12

PEAS acronimo di Performance Environment Actuators Sensors; descrive l'ambiente operativo - la misura di prestazione, l'ambiente esterno, gli attuatori e i sensori dell'agente - . 4

Spazio degli stati si intende lo stato iniziale, l'insieme di azioni possibili e il modello di transizione definiti nella formulazione di un problema da risolvere. 7