Attività Progettuale in Fondamenti di Intelligenza Artificiale M

Luca Marongiu

Settembre 2025

Indice

1 Introduzione

Questo progetto consiste nell'implementazione e confronto di diversi algoritmi per la risoluzione automatica di Sudoku. Il Sudoku è stato scelto come dominio di applicazione in quanto rappresenta un tipico esempio di *Problema di Soddisfacimento di Vincoli* (CSP), si tratta di un problema ben definito, con regole semplici ma con uno spazio di ricerca estremamente ampio, che rende necessaria l'adozione di strategie intelligenti per ottenere soluzioni efficienti.

Per studiare i diversi approcci è stato realizzato anche un generatore di Sudoku con tre diversi livelli di difficoltà (*facile*, *medio*, *difficile*), utile sia per effettuare test sperimentali che per validare la correttezza delle implementazioni. Sono stati poi condotti esperimenti sistematici con lo scopo di misurare tempi di esecuzione e numero di ricorsioni, così da mettere a confronto le diverse tecniche di risoluzione. Infine, è stata sviluppata un'interfaccia grafica interattiva che consente di generare e risolvere schemi di Sudoku in modo intuitivo, permettendo di osservare direttamente le prestazioni degli algoritmi.

L'implementazione è stata realizzata in `Python`, in particolare, per l'interfaccia grafica è stata utilizzata la libreria `pygame`.

L'obiettivo principale del progetto è stato duplice:

1. Implementare e confrontare tre approcci differenti alla risoluzione del Sudoku: *backtracking semplice*, *backtracking con euristica MRV (Minimum Remaining Values)* e *Dancing Links*.
2. Realizzare una semplice applicazione grafica con `pygame` che permetta di generare e risolvere automaticamente schemi di Sudoku, rendendo più immediata la dimostrazione pratica degli algoritmi implementati.

2 Sudoku

Sudoku è un puzzle logico-matematico che consiste nel riempire una griglia 9×9, suddivisa in 9 sottogriglie 3×3, inserendo i numeri da 1 a 9 in modo che ogni numero compaia esattamente una volta in ogni riga, colonna e sottogriglia.

Il gioco, nato in Giappone negli anni '80 e poi diffusosi a livello mondiale, rappresenta un problema semplice da comprendere per un giocatore umano, ma dal punto di vista computazionale nasconde una notevole complessità. È infatti stato dimostrato che il Sudoku appartiene alla classe dei problemi **NP-completi**, il che implica che non si conoscono algoritmi in grado di risolverlo in tempo polinomiale per tutte le istanze possibili.

Dal punto di vista dell'intelligenza artificiale, il Sudoku può essere formalizzato come un *Problema di Soddisfacimento di Vincoli* (CSP, Constraint Satisfaction Problem). In questa formalizzazione:

- le **variabili** sono le 81 celle della griglia,

- il **dominio** di ciascuna variabile è l'insieme $\{1, 2, \ldots, 9\}$,
- i **vincoli** impongono che ogni numero compaia una sola volta in ciascuna riga, colonna e sottogriglia 3×3.

Questa formulazione rende il Sudoku un banco di prova ideale per lo studio e l'applicazione di tecniche di ricerca, euristiche e strutture dati avanzate. La presenza di forti vincoli locali (riga, colonna e sottogriglia) permette infatti di valutare concretamente l'efficacia di strategie di *backtracking*, euristiche come MRV (Minimum Remaining Values), o approcci più sofisticati come *Dancing Links*, confrontando prestazioni e scalabilità.

3 Backtracking standard

Il primo approccio implementato è stato il **backtracking semplice**. L'idea di base consiste nell'assegnare progressivamente valori alle celle vuote, verificando a ogni passo la consistenza con i vincoli del Sudoku (unicità dei numeri in riga, colonna e sottogriglia). Se un'assegnazione porta a una violazione, si effettua un *backtrack*, cioè si torna indietro e si prova un valore diverso.

Dal punto di vista algoritmico, il backtracking può essere visto come una *ricerca in profondità* nello spazio delle possibili configurazioni della griglia. Ogni scelta corrisponde all'espansione di un nodo dell'albero di ricerca, e in caso di conflitto si risale al livello precedente. Questo garantisce la correttezza della soluzione, ma risulta inefficiente in assenza di euristiche.

Infatti, nel caso peggiore lo spazio di ricerca teorico può arrivare fino a 9^{81} possibili configurazioni, corrispondenti a tutte le possibili griglie riempite. Tuttavia, nei Sudoku reali una parte delle celle è già riempita all'inizio e i vincoli di riga, colonna e sottogriglia riducono drasticamente lo spazio da esplorare. Nonostante ciò, la crescita rimane comunque esponenziale e il metodo tende a esplorare numerosi rami non promettenti prima di giungere a una soluzione.

Questo approccio, seppur semplice, mette in evidenza i limiti di un metodo puramente ricorsivo. Per ridurre tali inefficienze, nella sezione successiva viene introdotta un'euristica che guida la ricerca in maniera più intelligente: l'MRV.

L'implementazione si trova nel file `versione_backtracking_base.py`.

4 Backtracking con MRV

Il secondo approccio introduce l'euristica **MRV (Minimum Remaining Values)**, tipicamente utilizzata nei problemi di soddisfacimento di vincoli (CSP). L'idea è di scegliere come prossima variabile la cella con il minor numero di valori ancora ammissibili, cioè quella più vincolata.

In questo modo l'algoritmo affronta subito le scelte più difficili, riducendo la probabilità di percorrere rami destinati al fallimento. Al contrario, un backtracking senza euristiche

può riempire inizialmente celle "facili" e rimandare i conflitti a molto più tardi, sprecando numerose ricorsioni.

Dal punto di vista pratico, l'uso di MRV riduce drasticamente lo spazio di ricerca esplorato. Come mostrato nei test sperimentali, il numero medio di ricorsioni risulta sensibilmente inferiore rispetto al backtracking standard, e i tempi di risoluzione diventano molto più stabili anche nei Sudoku più complessi.

L'implementazione si trova nel file `versione_backtracking_MRV.py`.

5 Dancing Links

Il terzo approccio implementato si basa sull'algoritmo **Dancing Links** (**DLX**) proposto da Donald Knuth [1]. DLX è un'implementazione particolarmente efficiente dell'algoritmo `Algorithm X`, un metodo di backtracking non deterministico per la risoluzione di problemi di *Exact Cover*.

5.1 Rappresentazione del problema

Il Sudoku può essere trasformato in un problema di exact cover: ogni possibile assegnamento (r, c, n) (numero n nella cella (r, c)) deve soddisfare esattamente quattro vincoli:

1. la cella (r, c) deve contenere un solo numero,
2. il numero n deve comparire una volta sola nella riga r,
3. il numero n deve comparire una volta sola nella colonna c,
4. il numero n deve comparire una volta sola nel sotto-blocco di appartenenza.

Questi vincoli vengono enumerati e raccolti in una struttura X, mentre per ogni possibile assegnamento (r, c, n) si costruisce un insieme $Y[(r, c, n)]$ che indica quali vincoli quell'assegnamento soddisfa.

5.2 Struttura dati adottata

L'implementazione sviluppata simula l'Algorithm X utilizzando le strutture dati native di Python, senza implementare la rete di puntatori doppiamente collegati della versione originale di Knuth:

- X è un dizionario che associa a ciascun vincolo (colonna) l'insieme delle righe che lo soddisfano;
- Y è un dizionario che associa a ogni assegnamento (r, c, n) i vincoli che esso copre.

5.3 Algoritmo di ricerca

Il cuore dell'algoritmo è la procedura ricorsiva `solve_dlx`, che segue esattamente i passi descritti da Knuth:

1. se non rimangono vincoli in X, è stata trovata una soluzione;
2. altrimenti si seleziona il vincolo con il minor numero di possibilità (euristica di riduzione della ramificazione);
3. per ciascun assegnamento che soddisfa quel vincolo, si aggiunge l'assegnamento alla soluzione parziale e si **coprono** i vincoli coinvolti;
4. si prosegue ricorsivamente; al termine, si esegue la procedura inversa (**scopri**) per ripristinare lo stato e continuare l'esplorazione.

Le funzioni `copri` e `scopri` simulano le operazioni di cover/uncover dell'Algorithm X: rimuovono e ripristinano elementi da dizionari e insiemi, permettendo di eseguire efficacemente le operazioni di backtracking richieste dall'algoritmo.

5.4 Risultati sperimentali

Nella pratica, DLX si è dimostrato il più rapido e scalabile tra gli algoritmi implementati. Grazie alla rappresentazione con exact cover e all'uso della scelta euristica del vincolo più restrittivo, i tempi di risoluzione risultano quasi indipendenti dalla difficoltà del puzzle. Anche Sudoku molto complessi vengono risolti in tempi dell'ordine dei millisecondi e con un numero di ricorsioni stabile e ridotto, confermando l'efficienza di questo approccio.

L'implementazione completa si trova nel file `versione_dancing_links.py`.

6 Interfaccia grafica

Per rendere più interattiva la dimostrazione, è stata realizzata una semplice interfaccia grafica utilizzando la libreria `pygame`. L'applicazione presenta le seguenti funzionalità principali:

- una griglia di Sudoku, disegnata con linee sottili e spesse per distinguere le sotto-griglie 3×3,
- un pulsante **Genera**, che produce un nuovo schema casuale a partire da tre livelli di difficoltà,
- un pulsante **Risolvi**, che risolve lo schema attuale utilizzando l'algoritmo DLX e mostra a schermo il tempo di risoluzione.

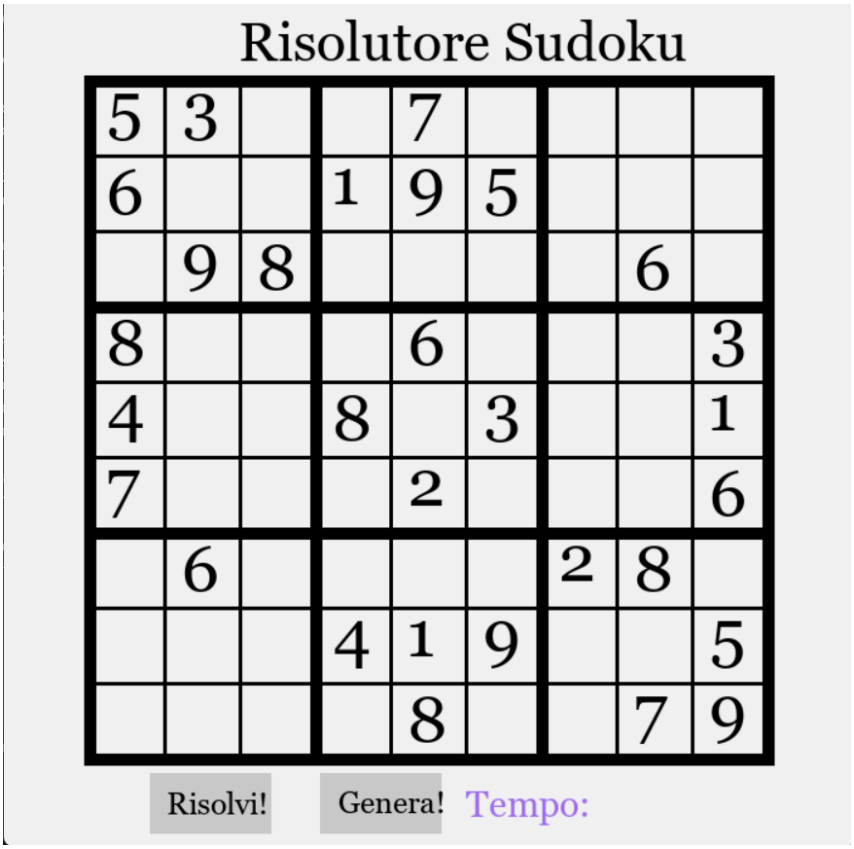

Figura 1: Interfaccia realizzata con `pygame`.

7 Test e risultati

Per confrontare l'efficacia dei tre algoritmi, sono stati generati Sudoku di difficoltà variabile (facile, medio, difficile) tramite un generatore basato su backtracking randomizzato. Il generatore funziona in due fasi principali:

1. **Creazione della soluzione completa**: partendo da una griglia vuota, vengono inseriti numeri casuali rispettando le regole del Sudoku tramite backtracking randomizzato, fino a ottenere una soluzione valida.

2. **Creazione del puzzle**: a partire dalla soluzione completa, vengono rimosse celle in modo controllato per ottenere il livello di difficoltà desiderato (facile: 40 celle vuote, medio: 50, difficile: 60). Dopo ogni rimozione, il puzzle viene verificato con il risolutore per assicurarsi che resti univocamente risolvibile.

Questa procedura garantisce che i puzzle siano coerenti con la difficoltà indicata e risolvibili dai solver utilizzati nei test.

Sono stati eseguiti diversi test confrontando i tre algoritmi su Sudoku di difficoltà variabile (facile, medio, difficile). Ogni esperimento è stato condotto su un campione di 100 Sudoku per ciascun livello di difficoltà. I risultati sono riassunti in Tabella 1.

Solver	Difficoltà	Tempo (s)	Ricorsioni
Backtracking	Facile	Medio: 0.0008	Medio: 119
	Medio	Medio: 0.0183	Medio: 2812
	Difficile	Medio: 3.7121	Medio: 406488
Backtracking MRV	Facile	Medio: 0.0059	Medio: 42
	Medio	Medio: 0.0114	Medio: 59
	Difficile	Medio: 0.0255	Medio: 80
Dancing Links	Facile	Medio: 0.0023	Medio: 41
	Medio	Medio: 0.0026	Medio: 55
	Difficile	Medio: 0.0036	Medio: 62

Tabella 1: Confronto tra i tre solver in termini di tempo medio di esecuzione e numero di ricorsioni.

Dall'analisi dei risultati emergono alcune considerazioni rilevanti:

- **Backtracking semplice**: corretto ma estremamente inefficiente. I tempi di esecuzione crescono in maniera esplosiva con la difficoltà del Sudoku e presentano una grande variabilità. Anche il numero di ricorsioni è molto elevato, superando in media le centinaia di migliaia nei casi difficili.
- **Backtracking con MRV**: l'introduzione dell'euristica riduce drasticamente lo spazio di ricerca. I tempi rimangono bassi anche per Sudoku medi e difficili e, soprattutto, il numero di ricorsioni si stabilizza intorno a poche decine o centinaia. L'algoritmo risulta molto più stabile e prevedibile rispetto al backtracking standard.
- **Dancing Links**: si conferma di gran lunga il più veloce e il più stabile. I tempi medi di risoluzione sono dell'ordine dei millisecondi per tutte le difficoltà, con intervalli molto ristretti. Anche il numero di ricorsioni rimane praticamente costante. Questo dimostra l'efficienza dell'approccio basato su exact cover e giustifica il suo utilizzo come solver di riferimento.

I file con i risultati dei test completi sono raccolti nella cartella `risultati`, che contiene i Sudoku utilizzati per ciascun algoritmo e ciascun livello di difficoltà. Il file `statistiche_riassuntive.txt` riassume invece le medie e le statistiche aggregate.

8 Conclusioni

Il progetto ha permesso di analizzare e confrontare tre approcci distinti alla risoluzione del Sudoku, mostrando in modo chiaro come l'efficienza degli algoritmi possa variare drasticamente in funzione delle tecniche adottate.

Il **backtracking semplice** si è confermato corretto ma poco scalabile, con tempi e ricorsioni che crescono in maniera esplosiva al crescere della difficoltà. L'uso dell'euristica **MRV** ha introdotto un miglioramento significativo, riducendo lo spazio di ricerca e stabilizzando le prestazioni anche su schemi complessi. Infine, l'algoritmo **Dancing Links** ha dimostrato la propria superiorità, risultando quasi indipendente dalla difficoltà e risolvendo ogni Sudoku in tempi dell'ordine dei millisecondi.

Riferimenti bibliografici

[1] Donald E. Knuth. *Dancing Links*. `https://arxiv.org/abs/cs/0011047`

[2] Pygame Community. *Pygame Documentation*. `https://www.pygame.org/docs/`