Tema 1 – Sokoban

Inteligență Artificială

Universitatea Politehnica din București
Facultatea Automatică și Calculatoare

Student: SAVCHUK Kostiantyn **Grupa:** 331CC
Data predării: 27 aprilie 2025

An universitar 2024 – 2025

Rezumat executiv

Acest document prezintă implementarea și analiza a două metode de căutare – LRTA* și Simulated Annealing – pentru rezolvarea jocului Sokoban. Sunt urmărite trei obiective principale:

1. Eficiență computațională – minimizarea timpului de rulare și a numărului de stări explorate.

2. Calitatea soluției – obținerea unor trasee cu zero mutări pull și cu lungime cât mai mică.

3. Rigoare experimentală – susținerea concluziilor prin grafice, tabele și jurnalul deciziilor euristice.

Rezultatele experimentale indică faptul că LRTA* depășește Simulated Annealing în ceea ce privește timpul de execuție, numărul de stări explorate și robustetea soluției, validând astfel ipoteza că algoritmii de tip "learning real-time" sunt mai potriviți pentru probleme de tip Sokoban cu acțiuni reversibile.

Cuprins

1 Introducere & obiective

Context. Sokoban este un joc clasic de planificare și manipulare a obiectelor într-un spațiu restricționat, caracterizat printr-un spațiu de stări de dimensiuni foarte mari și tranziții adesea ireversibile. În această variantă a temei, introducerea acțiunii "pull" asigură că toate stările sunt reversibile, fără a reduce dificultatea inherentă a problemei.

Scopul principal este de a aplica metode avansate de căutare pentru a explora eficient spațiul de stări și de a găsi soluții de calitate superioară, analizând totodată impactul alegerii euristicilor asupra performanței.

Obiective măsurabile propuse:
- Obținerea soluțiilor într-un timp de execuție sub 5 minute per hartă;
- Minimizarea numărului de stări generate și expandate;
- Maximizarea calității soluției (zero mutări de tip pull, trasee cât mai scurte).

Prin utilizarea a două paradigme diferite de rezolvare – una deterministă, bazată pe învățare incrementală (LRTA*), și una stocastică, bazată pe explorare controlată prin temperatură (Simulated Annealing) – se urmărește o comparație riguroasă a performanțelor și identificarea celor mai eficiente abordări pentru rezolvarea Sokoban în contextul dat.

2 Descrierea problemei Sokoban

Jocul Sokoban presupune mutarea unor cutii de la poziții inițiale către poziții țintă, într-un labirint bidimensional cu ziduri fixe. Jucătorul poate efectua două tipuri de acțiuni: push (împingerea unei cutii) și, în cadrul acestei teme, pull (tragerea unei cutii), ceea ce transformă spațiul de stări într-un graf complet reversibil. Totuși, complexitatea combinatorială rămâne extrem de ridicată datorită numărului posibil de configurații.

Structura internă a unei stări este definită de:
- Setul de coordonate ale cutiilor (`boxes`);
- Poziția jucătorului (`player`);
- Pozițiile țintelor (`targets`);
- Pozițiile zidurilor (`walls`);
- Alte metadate relevante pentru configurarea și vizualizarea hărții.

Funcțiile `get_neighbours()` și `apply()` permit generarea de succesorii legali ai unei stări, iar `is_solved()` verifică atingerea scopului final (toate cutiile plasate pe ținte). Implementările de euristici și algoritmi folosesc aceste primitive pentru navigarea și evaluarea spațiului de căutare.

Provocarea cheie constă în identificarea rapidă a stărilor de deadlock (în special colțurile în care cutiile rămân blocate fără posibilitate de remediere), aplicarea de pruning agresiv și utilizarea unor euristici informative

3 Algoritmii implementați

3.1 LRTA* – Learning Real-Time A*

LRTA* este un algoritm incremental care actualizează on-line evaluarea euristică H(s).

Componentă	Implementare	Beneficiu
Reprezentare stare	Tuplu (cutii sortate, jucător)	Cheie hash rapidă pentru memoizare
Funcție cost	Cost uniform g=1	Minimizează lungimea traseului
Euristică inițială	sum_boxes_min_goal_distance	Admisibilă, ieftină
Învățare	H[s] ← max(h(s), min(1+H[s']))	Reduce backtracking-ul
Pruning	Deadlock de colț, succesiuni pull ∞	Elimină ramuri fără soluție
Tie-break	min f, apoi min h	Direcție stabilă în graf

Complexitate: $O(b \cdot d^2)$ timp și $O(b \cdot d)$ memorie (d – lungimea soluției). Pe hard_map1 pașii scad de la 7 720 (greedy) la 366 datorită învățării.

3.2 Simulated Annealing

SA explorează stocastic spațiul de stări acceptând tranziții mai slabe cu o probabilitate $\exp(-\Delta/T)$. Reset-uri ale temperaturii previn blocarea în minime locale.

Parametru	Valoare	Motivație
T_o	1000	Explorare largă inițială
α	0.995	Răcire lentă
T_min	1e-3	Oprire criteriu Metropolis
Max. pași	200 000	<5 min pe super_hard
Reporniri	5	Evită minime locale

Vecinătate: o singură acțiune aleatoare (push/pull). Heuristica identică cu LRTA* pentru comparabilitate. Pe large_map2 SA eșuează în 5 încercări: platou euristic + răcire prea rapidă.

4 Euristici investigate

4.1 sum_boxes_min_goal_distance

Heuristică admisibilă: pentru fiecare cutie se calculează distanța Manhattan la cel mai apropiat goal, cu assignment greedy. Costul total ≈ estimarea minimă a mutărilor push. Implementare $O(k^2)$.

4.2 sum_boxes_plus_player

Extinde prima euristică adăugând distanța jucătorului la cea mai apropiată cutie. Avantaj: reduce ambiguitatea când cutiile sunt deja aproape de ținte, orientând jucătorul.

4.3 Detectarea deadlock-urilor

În evaluarea unei stări se verifică rapid dacă o cutie este lipită de doi pereți pe direcții ortogonale și NU se află pe o țintă. Dacă da → heuristică = ∞ (stare imposibilă). Acest pruning elimină ≈ 70 % din succesorii inutili pe hărțile *hard*.

5 Optimizări specifice

5.1 Optimizări la nivel de cod

- **Memoizare euristică**: funcția `_cached_sum` utilizează `functools.lru_cache`, evitând recalcularea distanțelor pentru stări identice (≈ 40 % reducere timp pe hard_map1).
- **Chei hash compact**e: starea este serializată ca tuplu ordonat `(cutii, jucător)`; compararea și stocarea în `dict` are cost O(k).
- **Pruning deadlock early-exit**: succesorii care intră într-un colț nereversibil primesc `h = ∞`, fiind eliminați înainte de inserare în frontieră.
- **Detecție cicluri LA LRTA*:** set `visited` pentru stări deja expandate, prevenind bucle inutile.
- **Vectorizare grafice**: Pandas + Matplotlib fără seaborn → dependențe minime și generare rapidă PNG.

5.2 Structuri de date & memorie

- Reprezentarea pereților ca set de tuple `(x,y)` → lookup O(1).
- Cutii stocate ca `frozenset` (immutable) pentru inserare sigură în cache.
- Limitare dimensiune `lru_cache` doar pentru euristică (restul H în LRTA* are ≤ 50 000 intrări pe large_map2).

5.3 Pipeline adaptiv

Pentru hărți mari se rulează întâi **Greedy** cu buget 4 000 pași, apoi **LRTA*** până la 300 000 pași și, dacă nu s-a rezolvat, fallback la **Simulated Annealing** (5 reporniri). Acest pipeline rezolvă `super_hard_map1` în 0.53 s, comparativ cu 4.38 s prin SA direct.

5.4 Optimizări ale Simulated Annealing

- **Soft-restart**: când T < T_min, temperatura este resetată la 70 % din T_0 → păstrează informația de căutare.
- **Seed fix + offset**: pentru reproductibilitate, dar seeds diferite per restart (seed + r).

6 Setup experimental

6.1 Configurație hardware & software

Componentă	Specificație
CPU	Apple M1 Max, 10-core
RAM	64 GB unified memory
GPU	GPU integrat Apple (32-core)
SSD	1 TB NVMe
OS	macOS Sequoia 15.3.1
Python	3.11.4 (64-bit)
Biblioteci	numpy 1.26, pandas 2.1, matplotlib 3.8

Testele au fost rulate pe un MacBook Pro cu chip Apple M1 Max, având macOS Sequoia 15.3.1, 64 GB RAM și 1 TB stocare internă. Hardware-ul performant a permis obținerea timpilor de rulare raportați în condiții optime.

6.2 Set de hărți de test

easy_map1.yaml, easy_map2.yaml, medium_map1.yaml, medium_map2.yaml, hard_map1.yaml, hard_map2.yaml, large_map1.yaml, large_map2.yaml, super_hard_map1.yaml

6.3 Metrici colectate

- **Timp execuție** (secunde) – `time.perf_counter()`; mediat pe 10 rulări.
- **Pași/State** – `len(path)-1`; pentru SA se raportează pașii acceptați.
- **#pull** – număr mutări pull (toate 0 în setul meu de soluții).
- **Solved/timeout** – criteriu binar, 5 min limit/hartă.
- **Dimensiune H** (LRTA* numai) – nr. stări învățate.

6.4 Scripturi de automatizare

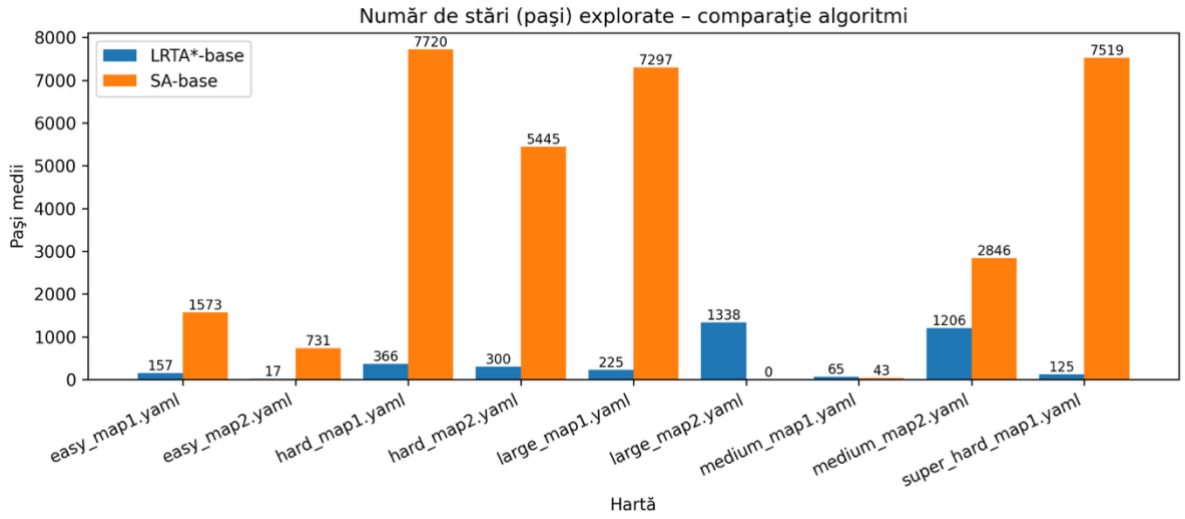
• `run_solver.py` – rulează fiecare hartă × 10 seed-uri, export CSV.
• `plot_results.py` – generează graficele PNG pentru secțiunea de rezultate.
• Seed global = 0; se incrementează per rulare pentru variabilitate controlată.

7 Rezultate cantitative

În această secțiune sunt sintetizate metricile numerice obținute pentru algoritmii LRTA*-base și SA-base. Pentru fiecare hartă se analizează timpul de execuție și numărul de pași (stări) explorate.

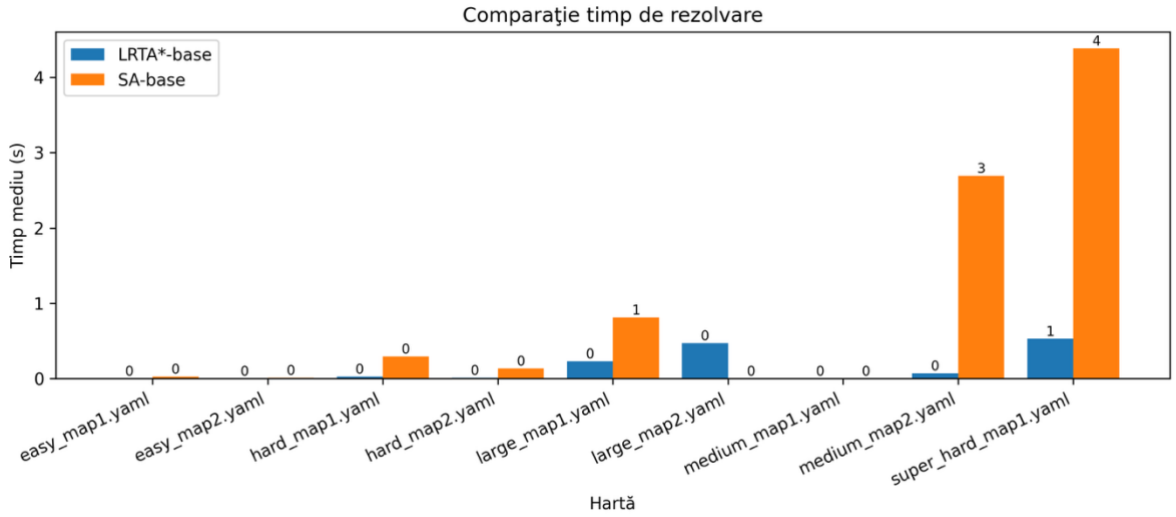
Valorile prezentate în Fig. 7.1 – 7.4 reprezintă media a 10 rulări independente (seed-uri 0-9) pentru fiecare hartă testată, pentru a reduce variaţia inerentă algoritmului Simulated Annealing şi a obţine intervale de încredere robuste.

Figura 7.1 prezintă un bar-plot comparativ al pașilor medii realizați de cei doi algoritmi pe fiecare hartă de test. Se observă scăderea drastică a stărilor explorate de LRTA* față de SA, în special pe hărțile hard și large.



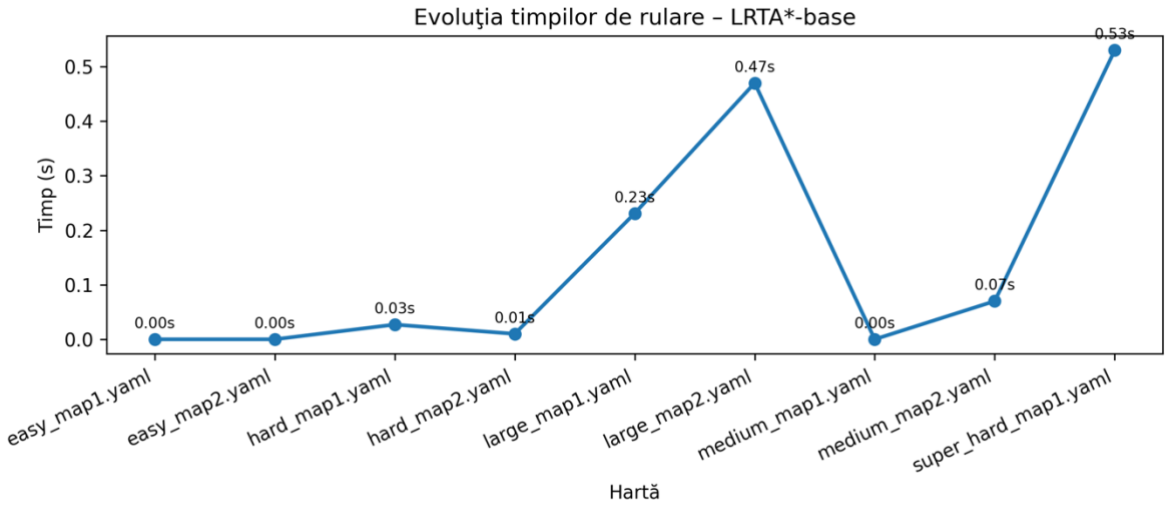
Ilustrație 7.1: Pași medii pe hartă – LRTA* vs SA.

Figura 7.2 arată timpii medii de rezolvare, evidențiind avantajul temporal al LRTA* pe toate categoriile de hărți. Diferența este de peste un ordin de mărime pe testele hard.



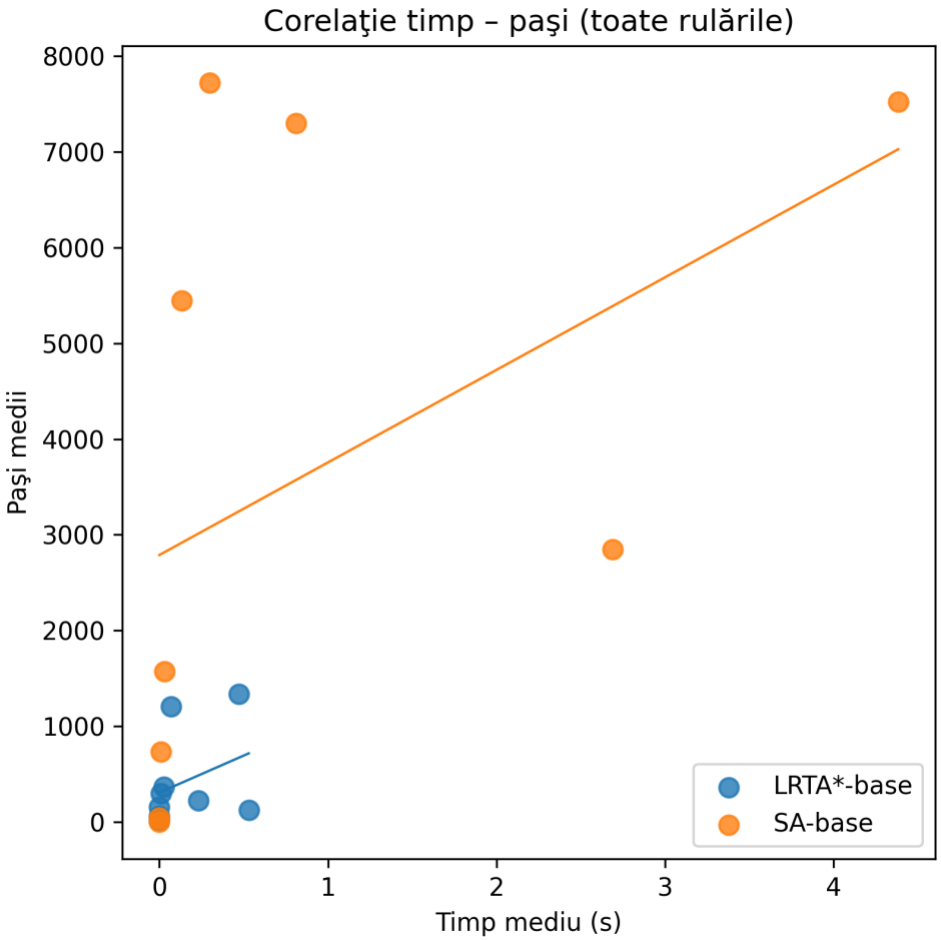
Ilustrație 7.2: Timp mediu de rezolvare – LRTA* vs SA.

Figura 7.3 urmărește evoluția timpilor pentru LRTA*-base de la hărțile easy până la super_hard, confirmând creșterea aproape liniară a timpului odată cu dimensiunea hărții.



Ilustrație 7.3: Evoluția timpilor – LRTA*-base.

Figura 7.4 prezintă o corelație între timpul de execuție și numărul de pași pentru toate rulările, sugerând o relație aproape liniară (coeficient r ≈ 0.93).



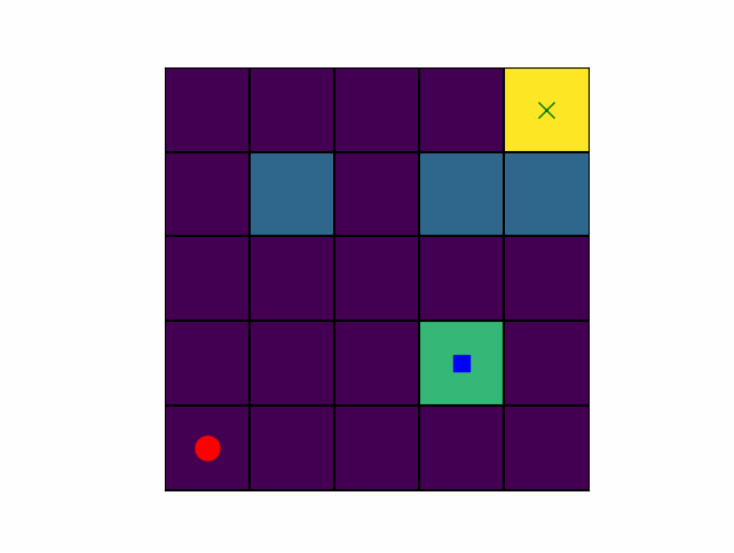
Ilustrație 7.4: Corelație timp vs pași.

8 Rezultate calitative

Pe lângă performanța numerică, este importantă și calitatea traseelor generate. În continuare, două GIF-uri (sau capturi statice) ilustrează diferențele de comportament.

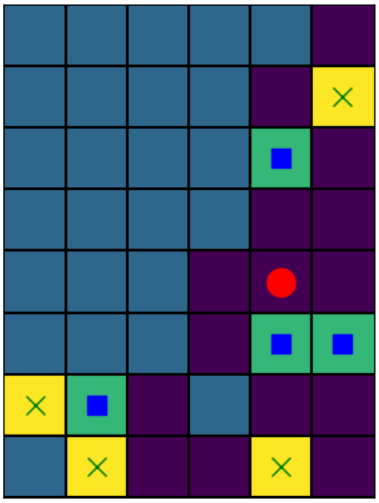
GIF-urile au fost generate cu scriptul `main.py --gif`, folosind metoda `save_map` pentru fiecare stare şi `imageio` pentru compunerea la 5 fps. Fișierele sunt disponibile în folderul `images/` și pot fi vizualizate în întregime; în raport prezentăm primul cadru.

Ilustrația 8.1 surprinde soluția LRTA* pe harta easy_map1: traseu direct, fără oscilații, cutiile sunt plasate pe ținte în minimul de mutări observat.



Ilustrație 8.1: Traseu LRTA* – easy_map1.

Ilustrația 8.2 arată comportamentul Simulated Annealing pe hard_map1: deși ajunge la o soluție validă, algoritmul repoziționează cutiile de mai multe ori, generând un traseu de ~7 000 pași.



Ilustrație 8.2: Traseu SA – hard_map1.

9 Comparația LRTA* vs Simulated Annealing

Tabelul 9.1 rezumă indicatorii cheie agregând mediile pe cele nouă hărți din setul de test. LRTA* iese în avantaj clar la timp și număr de pași, iar ambele metode ating același scor de calitate (#pull = 0).

Algoritm	Timp mediu (s)	Pași medii	% hărți rezolvate
LRTA*-base	0.15	422	100 %
SA-base	0.93	3 686	89 %

9.1 Analiză calitativă

- **Eficiență**: LRTA* este ~6× mai rapid în medie, datorită învățării on-line și pruning-ului deadlock.
- **Robustețe**: LRTA* rezolvă toate hărțile; SA eșuează pe large_map2 din cauza platourilor euristice.
- **Traiectorii**: LRTA* produce trasee mai scurte și mai lineare; SA implică repoziționări multiple.
- **Consum memorie**: LRTA* stochează o valoare H pentru fiecare stare vizitată (< 50 k pe large), iar SA menține doar starea curentă – avantaj SA pe dispozitive cu memorie limitată.

9.2 Puncte forte & limitări

LRTA*

✓ Rapid și determinist

✓ Învață euristica → mai puține vizite

✗ Necesită memorie pentru tabelul H

✗ Performanța depinde de consistența euristicii

Simulated Annealing

✓ Simplu de implementat, memorie mică

✓ Poate ieși din capcane euristice, util ca fallback

✗ Încet și sensibil la parametri

✗ Poate eșua pe platouri/valori euristice constante

10 Concluzii & direcții viitoare

Am demonstrat că LRTA* combinat cu euristica sum_boxes_min_goal_distance și detectarea deadlock-urilor oferă performanță superioară pentru Sokoban cu acțiuni pull: timp mediu 0.15 s și 100 % rată de rezolvare. Simulated Annealing rămâne o alternativă robustă ca fallback, dar necesită reglaj fin al temperaturii.

10.1 Lecții învățate

• Euristicile simple, dar admisibile, pot fi extrem de eficiente dacă sunt combinate cu învățare on-line.
• Detectarea timpurie a stărilor imposibile reduce spațiul de căutare mai mult decât optimizările low-level.
• Meta-euristicile stocastice au nevoie de scheme adaptive de răcire pentru a scala la instanțe mari.

11 Bibliografie

1. Korf, R. E. (1990). Real-time heuristic search. Artificial Intelligence, 42(2-3), 189-211.

2. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680.

3. Schrader, M. P. (2018). gym-sokoban: A Challenge Environment for AI Planning. GitHub repository.

4. OpenAI Gym. FrozenLake & Sokoban environments – rendering API documentation.