GUIDED DOMAIN SOLVER: STRUCTURED EXPLORATION OF DOMAIN-SPECIFIC TASKS WITH LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

This work presents a method to solve domain-specific problems by leveraging Monte Carlo Tree Search (MCTS), Knowledge Graphs and Large Language Model (LLM) agents. At the core of this approach lies a MCTS algorithm, which explores the complex solution space of a given domain in a goal-directed and sample-efficient manner. In the expansion phase of the MCTS, a domain-specific knowledge graph is incorporated to encode concepts, relationships and constraints. This structured representation enables an LLM agent to make informed decisions for the node expansion. By combining a structured search of the solution space through MCTS, a representation of domain knowledge through the knowledge graph and the generalization abilities of an LLM agent, this method can solve complex tasks in domains where both creativity and adherence to expert rules are essential. In a first step, this approach is used to solve Sokoban, a puzzle game that requires planning and creativity to place several boxes at specific targets with as few moves as possible.

1 INTRODUCTION

LLMs based on the Transformer architecture (Vaswani et al., 2017) achieve good results in generalizing many tasks and excel particularly in tasks such as text generation or translation. Despite these impressive advances, they tend to hallucinate or have errors in reasoning, especially when performing complex tasks (Chang et al., 2023; Hadi et al., 2023; Ji et al., 2023; Momennejad et al., 2023). In order to improve these shortcomings, several attempts were made to improve the planning of LLMs (Minaee et al., 2025).

Fine-tuning LLMs can be tricky and requires a lot of high-quality data, as well as the construction of a training pipeline, both of which can be costly (Cao et al., 2024). In order to provide the LLM with the latest data without having to retrain it, various connections to databases (Lewis et al., 2021) and search engines (Xiong et al., 2024) were explored.

Deep Reinforcement Learning (DRL) algorithms can be applied for complex planning tasks in an environment. Since rewards are usually sparse in these domains, training is often inefficient. To overcome this, the task is usually divided into sub-goals (Pastukhov, 2025). When changes are made in the environment, DRL agents can lose performance, resulting in retraining or the need for specialized methods (Liu et al., 2023; Chen et al., 2025).

In this work, a method was developed that explores a domain step by step using an MCTS (Coulom, 2007). This allows even complex planning tasks to be addressed. These include tasks such as robotics planning, engineering design automation, puzzle games and scientific discovery. The LLM does not need to be retrained, as recent and enriched data can be embedded in the Knowledge Graph.

Fine-tuning the LLM is expected to result in a more efficient exploration of the solution space. During inference, LLM test-time compute is performed through reasoning and expanding the Knowledge Graph, which can be more effective than scaling model parameters (Snell et al., 2024).

2 Related Work

Sokoban is a popular testing ground for approaches that aim to improve the planning capabilities of LLMs. There have been several impressive attempts to solve the game, using LLMs that are retrained on search strategies, as in Searchformer (Lehnert et al., 2024). In this approach, an LLM is fine-tuned on the search dynamic of A* (Hart et al., 1968) to solve Sokoban. Other attempts involve building a world model to solve the game, as in WorldCoder (Tang et al., 2024). Here, knowledge about the game is translated into Python code, which represents the world model. This knowledge is highly interconnected, as in the Knowledge Graph, but is more difficult to expand.

Knowledge Graph combined with LLMs to enhance their capabilities is a promising field of research (Pan et al., 2024; Kau et al., 2024). The field is usually divided into these three categories: Knowledge Graph enhanced LLMs (Lewis et al., 2021; Baek et al., 2023), LLM augmented Knowledge Graph (Hu et al., 2023; Hao et al., 2023), or hybrid approaches (Zhang et al., 2019; Wang et al., 2023).

MCTS has often been used to plan the next action in a large problem space. Examples of this include AlphaZero from Silver et al. (2017) or the External and Internal Planning with Language Models from Schultz et al. (2025).

3 Guided Domain Solver

The Guided Domain Solver uses an MCTS to search a large problem space step by step. Expert knowledge is embedded in a Knowledge Graph and can be used to make the search for promising states more effective. The Knowledge Graph, which is stored in a graph-based database such as Neo4J, contains information about all visited states and the currently selected state. This information is summarized using several Cypher queries to prompt an LLM which evaluates the next action. An overview of the method can be seen in Figure 1.

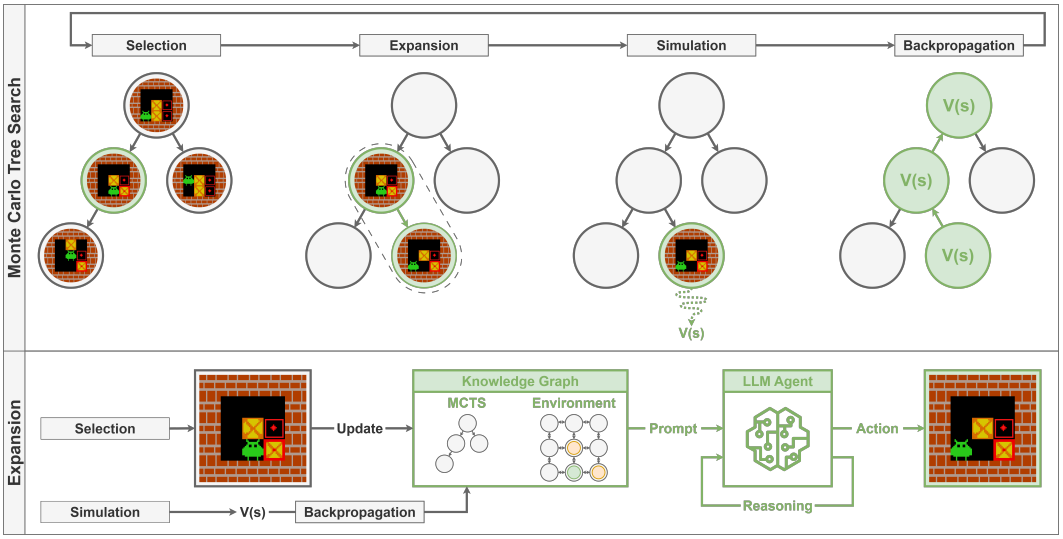


Figure 1: Overview that describes the Guided Domain Solver. The upper half shows the MCTS procedure. The lower half provides a more detailed view of the expansion step, showing how the LLM agent is prompted with the structured information from the Knowledge Graph.

3.1 DOMAIN: SOKOBAN

Sokoban is a Japanese puzzle game in which the player must move boxes in a warehouse so that they reach specified target fields. The implementation as OpenAI Gym was created by Schrader (2018). An example of the starting point of the game can be seen on the left side of Figure 2. The player can only move one box at a time. If two boxes are lined up behind each other, they cannot be moved. Dragging boxes is not allowed. A strategic approach is required to avoid maneuvering the boxes into dead ends or blocking the access to the remaining boxes (Racanière et al., 2017).

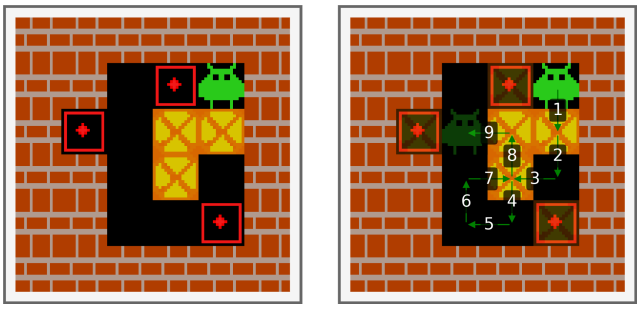


Figure 2: On the left is the starting position of a Sokoban game. To the right is the optimal sequence of actions to solve the game.

The goal is to solve the game in as few steps as possible. An optimal solution looks like the example on the right side of Figure 2. In order to achieve an optimal solution in larger scenes, it is necessary to develop creative long-term strategies.

3.2 MONTE CARLO TREE SEARCH

The algorithm is based on MCTS to find a solution for the Sokoban game. It involves the steps of selection, expansion, simulation, and backpropagation, which are repeated several times. The resulting search tree is stored in the Knowledge Graph so that already explored game states can be factored into the expansion of new ones.

3.2.1 SELECTION

During the selection phase of MCTS, the algorithm traverses the current search tree to identify the most promising node for further investigation. Normally, an upper confidence bound (UCB) for trees (Auer et al., 2002) is used to find a balance between exploitation and exploration:

$$\text{UCB}_i = V_i + C \cdot \sqrt{\frac{\ln N_i}{n_i}} \tag{1}$$

The exploitation is defined by the value of the state V_i. Exploration is controlled by the exploration constant C. N_i is the number of visits to the parent node, whereas n_i is the number of visits to the child node. In the case of the Sokoban game, only exploitation is used. This is because it is a fully observable domain and therefore an error-free simulation can be used. Thus, the most promising node that still contains unexplored actions is selected. Once the node has been selected, the Knowledge Graph is updated with the current game status.

3.2.2 EXPANSION

In the expansion step, the search tree is extended with a new node. Multiple Cypher queries are made on the Knowledge Graph, which summarize the current state of the game and game states that have already been reached. In addition, the shortest paths to place each of the remaining boxes are calculated. For this purpose, all other boxes are considered walls, which makes some box placements impossible. Missing box placements should be interpolated by the LLM agent using the additional

information. With all this information, an LLM agent is prompted to execute the next action in the game state. An example prompt can be seen in Figure 3.

Example Prompt

```
system:    You are a player who tries to solve a Sokoban game.
           Keep the reasoning short.
           Respond only with a single action out of ['UP', 'DOWN', 'LEFT', 'RIGHT'].
human:     Use the following results retrieved from a database to provide the next
           action for the Sokoban game.
           Environment: {environment}
           Shortest paths to place remaining boxes: {paths_to_place_remaining_boxes}
           Attempted Actions: {attempted_actions}
           Possible Actions: {possible_actions}
           Action:
response:  {action}
```

Figure 3: Example prompt for evaluating the next action in a Sokoban game.

During expansion, it is possible to introduce hard rules using domain knowledge. In the Sokoban game, the actions are mapped to the discrete action space. If the LLM agent does not provide a feasible action, a random possible action is selected as a fallback.

3.2.3 SIMULATION

In the simulation phase, the result of the newly added node is evaluated by simulating a trajectory from this point to a final state using an efficient policy. Normally, a ratio of games won to lost is used to determine the value of the game state. Since the Sokoban game has a very sparse distribution of games won, finding a solution using this method can take considerably longer. Here, it is possible to take advantage of the fact that the possible game states of Sokoban are finite. Therefore, the remaining steps to solve the game state are used as the value of the game state. This evaluation is determined using a breadth-first search, which represents an error-free evaluation of the game state that cannot be achieved in many other domains. A description of the breadth-first search algorithm can be found in the Appendix A.

3.2.4 BACKPROPAGATION

In the backpropagation step, the result of the simulation is propagated back through the search tree, updating the value estimates of each node along the path. In the case of the Sokoban game, the following update rule is applied here:

$$V(s_t) \leftarrow V(s_t) + \alpha\gamma^n V(s_{t+n}) \tag{2}$$

The update rule is an adapted variation of the n-step temporal difference learning approach from Sutton & Barto (2020) without rewards. It gets applied upwards on each node in the search tree to determine the new value of the game state $V(s_t)$. During the update, the number of steps n in the update rule depends on how deep the simulated node is in the search tree. The learning rate α and the discount factor γ can be tuned.

3.3 KNOWLEDGE GRAPH

The Knowledge Graph allows domain-specific knowledge to be encoded in a structured manner, allowing for precise queries, cross-system integration, and human-interpretable data. This is used to provide the LLM agent with domain knowledge without having to retrain the model. In the case of the Sokoban game, a representation of the game status and the MCTS is stored. The state of the game is divided into *static*, *dynamic*, and *action* layers, as shown in Figure 4. Dividing the Knowledge Graph into layers can be helpful in separating logical structures from one another,

resulting in optimized queries, better maintainability, and improved reasoning. The nodes of the *static* layer are set at the beginning and do not change during the game. All moving objects are mapped in the *dynamic* layer. The nodes in this layer remain throughout the game and only their relations change. The *action* layer determines the possible actions that can be performed in the current state. In any given state, nodes in this layer can disappear or be added.

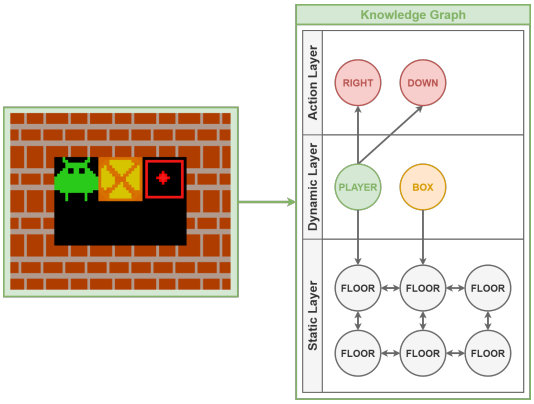


Figure 4: Conversion of the Sokoban game state to a graph representation in the Knowledge Graph.

The nodes of the floors contain the target positions of the boxes as properties. In the relations, the positions of the player and the boxes on the playing field are encoded. This allows effective queries to be performed to determine which box is closest to the player or which box can be placed the fastest. The complete schemas of the Knowledge Graph can be found in the Appendix B.

4 Experiments

The models Qwen3:32b (Yang et al., 2025), Deepseek-R1:70b (DeepSeek-AI et al., 2025), and GPT-OSS:120b (OpenAI, 2025) were used as LLM agents. These were run locally on an Nvidia RTX A6000. The solutions found for the Sokoban game are equivalent to the optimal solutions, as the same number of steps are required to solve the game. This is made possible by the optimal function within the simulation step. A subset of the solved games can be found in the Appendix C.

Since the optimal solution is always achieved in this domain, the number of nodes required to find the optimal solution is compared here. Therefore, three different methods are evaluated to determine the expansion efficiency as seen in Figure 5. One was expansion with a random possible action. In comparison, random sampling takes a random action from the shortest paths to place the unplaced boxes. The last approach was to use different LLM agents which receives all the information about the environment as well as the shortest paths to place the unplaced boxes.

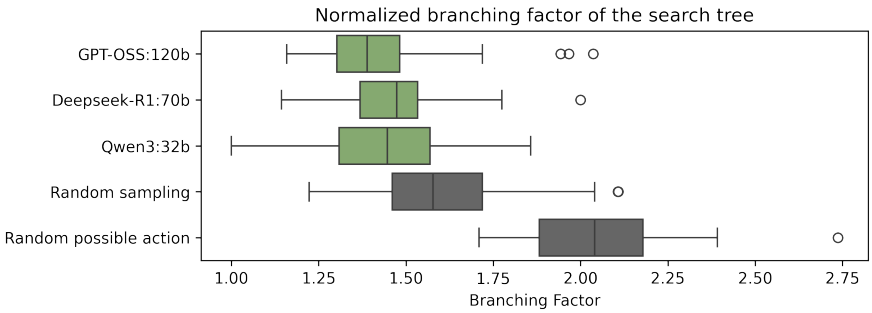


Figure 5: Normalized branching factor across methods, showing that the LLM agent explores fewer nodes relative to solution length, suggesting improved search efficiency.

The plot shows a comparison of the different methods in a greedy selection scenario. An error-free expansion would generate a branching factor of 1, which means that the best action is always taken. It can be seen that the LLM agents are able to make better decisions than the random methods. However, it requires significantly more time for evaluation and reasoning.

5 Conclusion

The Guided Domain Solver can solve the Sokoban game in an optimal way. Thereby the problem space is searched by the MCTS in an efficient, targeted manner. This initial application demonstrates the powerful combination of MCTS, LLM agent and Knowledge Graph. It paves the way for further applications in other domains that require a balance between flexibility and compliance with constraints. In future work, the method could also be extended to allow the LLM agent to execute its own queries in the Knowledge Graph. In addition, the method could be extended to continuous action spaces or used for continuous domain states.

References

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002. ISSN 0885-6125, 1573-0565. doi: 10.1023/A:1013689704352. URL https://link.springer.com/10.1023/A:1013689704352.

Jinheon Baek, Alham Fikri Aji, and Amir Saffari. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering, 2023. URL http://arxiv.org/abs/2306.04136.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2024. URL https://arxiv.org/abs/2306.09884.

Yihan Cao, Yanbin Kang, Chi Wang, and Lichao Sun. Instruction mining: Instruction data selection for tuning large language models, 2024. URL http://arxiv.org/abs/2307.06290.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models, 2023. URL http://arxiv.org/abs/2307.03109.

Xinqi Chen, Erci Xu, Dengyao Mo, Ruiming Lu, Haonan Wu, Dian Ding, and Guangtao Xue. MasterPlan: A reinforcement learning based scheduler for archive storage. *ACM Transactions on Architecture and Code Optimization*, 22(1):1–25, 2025. ISSN 1544-3566, 1544-3973. doi: 10.1145/3708542. URL https://dl.acm.org/doi/10.1145/3708542.

Thomas H. Cormen, Charles Eric Leiserson, Ronald Linn Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, third edition edition, 2009. ISBN 978-0-262-03384-8 978-0-262-27083-0.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers (eds.), *Computers and Games*, pp. 72–83. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-75538-8.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang

Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL http://arxiv.org/abs/2501.12948.

Muhammad Usman Hadi, Qasem Al Tashi, Rizwan Qureshi, Abbas Shah, Amgad Muneer, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, and Seyedali Mirjalili. Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects, 2023. URL https://www.techrxiv.org/doi/full/10.36227/techrxiv.23589741.v3.

Shibo Hao, Bowen Tan, Kaiwen Tang, Bin Ni, Xiyan Shao, Hengzhe Zhang, Eric P. Xing, and Zhiting Hu. BertNet: Harvesting knowledge graphs with arbitrary relations from pretrained language models, 2023. URL http://arxiv.org/abs/2206.14268.

Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136. URL http://ieeexplore.ieee.org/document/4082128/.

Nan Hu, Yike Wu, Guilin Qi, Dehai Min, Jiaoyan Chen, Jeff Z. Pan, and Zafar Ali. An empirical study of pre-trained language models in simple knowledge graph question answering, 2023. URL http://arxiv.org/abs/2303.10368.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Delong Chen, Wenliang Dai, Ho Shu Chan, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023. ISSN 0360-0300, 1557-7341. doi: 10.1145/3571730. URL http://arxiv.org/abs/2202.03629.

Amanda Kau, Xuzeng He, Aishwarya Nambissan, Aland Astudillo, Hui Yin, and Amir Aryani. Combining knowledge graphs and large language models, 2024. URL http://arxiv.org/abs/2407.06564.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping, 2024. URL http://arxiv.org/abs/2402.14083.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks, 2021. URL http://arxiv.org/abs/2005.11401.

Hongyun Liu, Peng Chen, Xue Ouyang, Hui Gao, Bing Yan, Paola Grosso, and Zhiming Zhao. Robustness challenges in reinforcement learning based time-critical cloud resource scheduling: A meta-learning based solution. *Future Generation Computer Systems*, 146:18–33, 2023. ISSN

0167739X. doi: 10.1016/j.future.2023.03.029. URL https://linkinghub.elsevier.com/retrieve/pii/S0167739X23001061.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey, 2025. URL http://arxiv.org/abs/2402.06196.

Ida Momennejad, Hosein Hasanbeig, Felipe Vieira, Hiteshi Sharma, Robert Osazuwa Ness, Nebojsa Jojic, Hamid Palangi, and Jonathan Larson. Evaluating cognitive maps and planning in large language models with CogEval, 2023. URL http://arxiv.org/abs/2309.15129.

OpenAI. gpt-oss-120b. https://huggingface.co/openai/gpt-oss-120b, August 2025. URL https://huggingface.co/openai/gpt-oss-120b. Hugging Face model repository.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599, 2024. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2024.3352100. URL http://arxiv.org/abs/2306.08302.

Sergey Pastukhov. Solving sokoban using hierarchical reinforcement learning with landmarks, 2025. URL http://arxiv.org/abs/2504.04366.

Sébastien Racanière, Theophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/9e82757e9a1c12cb710ad680db11f6f1-Paper.pdf.

Max-Philipp B. Schrader. gym-sokoban. https://github.com/mpSchrader/gym-sokoban, 2018.

John Schultz, Jakub Adamek, Matej Jusup, Marc Lanctot, Michael Kaisers, Sarah Perrin, Daniel Hennes, Jeremy Shar, Cannada Lewis, Anian Ruoss, Tom Zahavy, Petar Veličković, Laurel Prince, Satinder Singh, Eric Malmi, and Nenad Tomašev. Mastering board games by external and internal planning with language models, 2025. URL http://arxiv.org/abs/2412.12119.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL http://arxiv.org/abs/1712.01815.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters, 2024. URL http://arxiv.org/abs/2408.03314.

Richard S. Sutton and Andrew Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, second edition edition, 2020. ISBN 978-0-262-03924-6.

Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment, 2024. URL http://arxiv.org/abs/2402.12275.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL http://arxiv.org/abs/1706.03762.

Yu Wang, Nedim Lipka, Ryan A. Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. Knowledge graph prompting for multi-document question answering, 2023. URL http://arxiv.org/abs/2308.11730.

Haoyi Xiong, Jiang Bian, Yuchen Li, Xuhong Li, Mengnan Du, Shuaiqiang Wang, Dawei Yin, and Sumi Helal. When search engine services meet large language models: Visions and challenges, 2024. URL http://arxiv.org/abs/2407.00128.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL http://arxiv.org/abs/2505.09388.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced language representation with informative entities, 2019. URL http://arxiv.org/abs/1905.07129.

A Breadth-first search

Breadth-first search is a graph traversal algorithm that explores nodes in order of their depth. It visits all immediate neighbors before moving on to nodes at the next level (Cormen et al., 2009).

Algorithm 1 Breadth-First Search

Require: Graph $G = (V, E)$, start node s
Ensure: Visits all nodes reachable from s in BFS order

Initialize an empty queue Q
Mark all nodes as unvisited
Mark s as visited and enqueue it into Q
while Q is not empty **do**
  $u \leftarrow$ dequeue(Q)
  process node u
  for all neighbors v of u **do**
    if v is not visited **then**
      Mark v as visited
      Enqueue v into Q
    end if
  end for
end while

B Knowledge Graph Schemas

The nodes and their relationships are shown in the schemas of the Knowledge Graph. Following are the schemas of the Knowledge Graph for Sokoban.

B.1 Environment Nodes


```
{
  "Player": {
    "labels": [],
    "properties": {
      "id": {"unique": false, "indexed": false, "type": "INTEGER", "existence": false}
          ,
      "caption": {"unique": false, "indexed": false, "type": "STRING", "existence":
          false},
      "y": {"unique": false, "indexed": false, "type": "INTEGER", "existence": false},
      "x": {"unique": false, "indexed": false, "type": "INTEGER", "existence": false}
    },
    "relationships": {
      "ON_TOP_OF": {
        "direction": "out",
        "labels": ["Floor"],
```

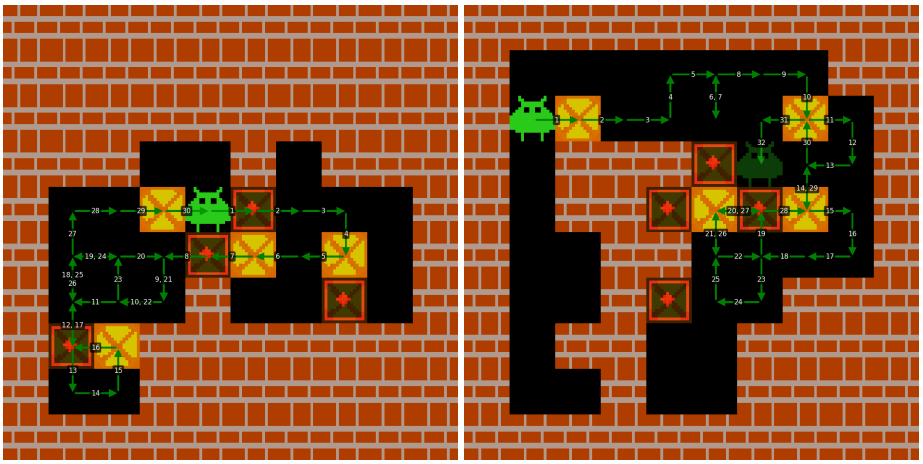
```
        "properties": {}
      },
      "CAN_MOVE": {
        "direction": "out",
        "labels": ["Action"],
        "properties": {}
      }
    }
  },
  "Box": {
    "labels": [],
    "properties": {
      "id": {"unique": false, "indexed": false, "type": "INTEGER", "existence": false}
          ,
      "caption": {"unique": false, "indexed": false, "type": "STRING", "existence":
          false},
      "y": {"unique": false, "indexed": false, "type": "INTEGER", "existence": false},
      "is_on_target": {"unique": false, "indexed": false, "type": "BOOLEAN", "
          existence": false},
      "x": {"unique": false, "indexed": false, "type": "INTEGER", "existence": false}
    },
    "relationships": {
      "ON_TOP_OF": {
        "direction": "out",
        "labels": ["Floor"],
        "properties": {}
      },
      "SHOULD_GO_TO": {
        "direction": "out",
        "labels": ["Floor"],
        "properties": {}
      }
    }
  },
  "Floor": {
    "labels": [],
    "properties": {
      "id": {"unique": false,"indexed": false,"type": "INTEGER","existence": false},
      "has_box_target": {"unique": false,"indexed": false,"type": "BOOLEAN","existence
          ": false},
      "caption": {"unique": false,"indexed": false,"type": "STRING","existence": false
          },
      "y": {"unique": false,"indexed": false,"type": "INTEGER","existence": false},
      "x": {"unique": false,"indexed": false,"type": "INTEGER","existence": false}
    },
    "relationships": {
      "ON_TOP_OF": {
        "direction": "in",
        "labels": ["Player", "Box"],
        "properties": {}
      },
      "CAN_GO_TO": {
        "direction": "out",
        "labels": ["Floor", "Floor"],
        "properties": {}
      },
      "SHOULD_GO_TO": {
        "direction": "in",
        "labels": ["Box"],
        "properties": {}
      }
    }
  }
}
```

B.2 Monte Carlo Tree Search Nodes


```
{
  "Path": {
    "labels": [],
    "properties": {
      "id": {"unique": false, "indexed": false, "type": "INTEGER", "existence": false}
          ,
      "possible_actions": {"unique": false, "indexed": false, "type": "LIST", "
          existence": false},
      "reward": {"unique": false, "indexed": false, "type": "FLOAT", "existence":
          false},
      "trajectory": {"unique": false, "indexed": false, "type": "LIST", "existence":
          false},
      "done": {"unique": false, "indexed": false, "type": "BOOLEAN", "existence":
          false},
      "value": {"unique": false, "indexed": false, "type": "FLOAT", "existence": false
          },
      "caption": {"unique": false, "indexed": false, "type": "STRING", "existence":
          false},
      "parent_id": {"unique": false, "indexed": false, "type": "INTEGER", "existence":
           false}
    },
    "relationships": {
      "MOVE": {
        "direction": "out",
        "labels": ["Path"],
        "properties": {
          "id": {"indexed": false, "type": "INTEGER", "existence": false, "array":
              false},
          "caption": {"indexed": false, "type": "STRING", "existence": false, "array":
               false}
        }
      }
    }
  }
}
```

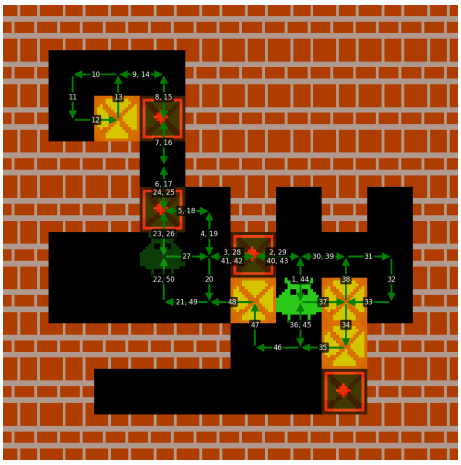

C Solved Sokoban games

In the following Figure 6 are several examples of Sokoban games that were solved using the Guided Domain Solver. In each case, the game was solved with the minimum number of steps.

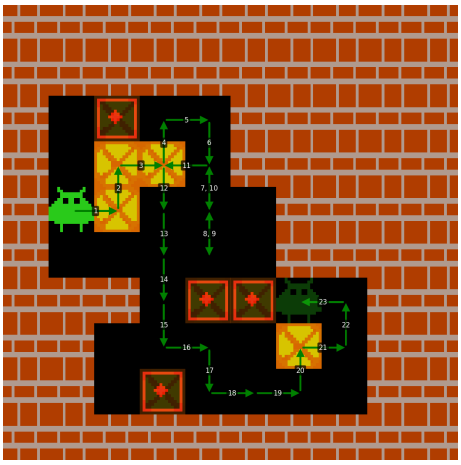


(a) Solved example from Bonnet et al. (2024).

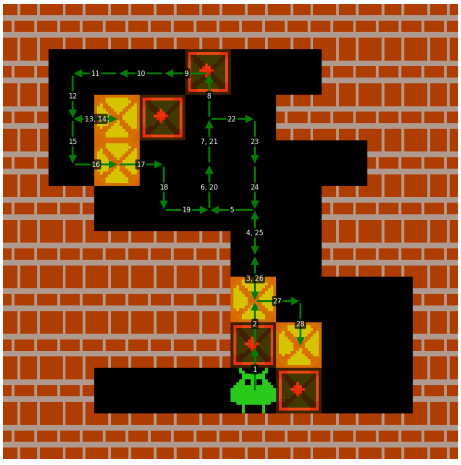
(b) Solved generated Sokoban game.



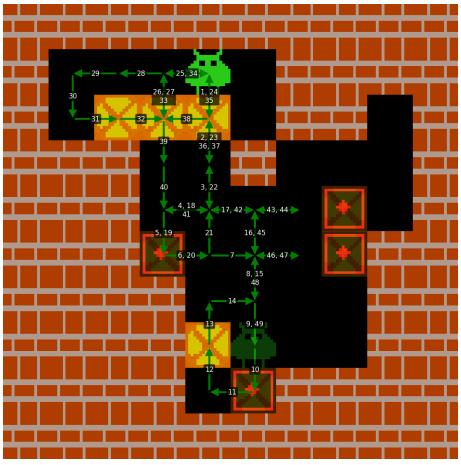
(c) Solved generated Sokoban game.



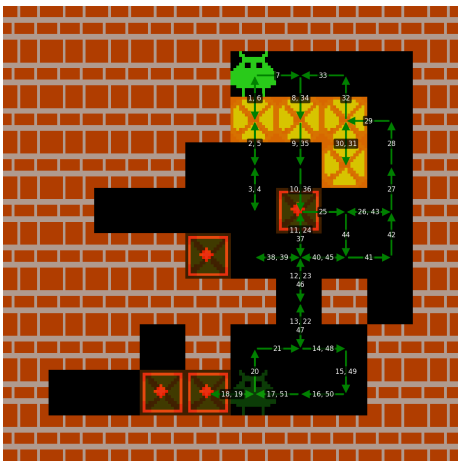
(d) Solved generated Sokoban game.



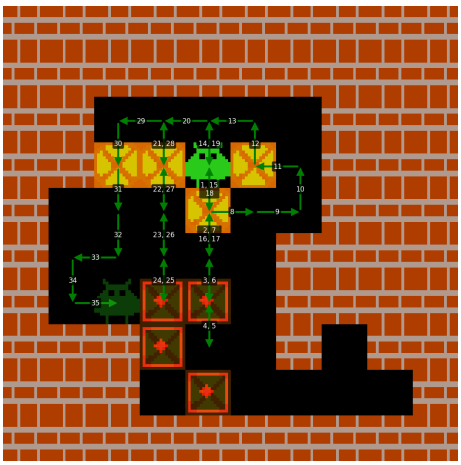
(e) Solved generated Sokoban game.



(f) Solved generated Sokoban game.



(g) Solved generated Sokoban game.



(h) Solved generated Sokoban game.

Figure 6: A subset of optimally solved Sokoban games. The images show the start of the game. Numbered green arrows indicate the actions that lead to the solution of the game. The final state of the game is shown in the form of a translucent overlay.

D USE OF LARGE LANGUAGE MODELS

During the writing of this paper, LLMs were used as a general-purpose assist tool. Enhancement suggestions from the Overleaf assistant were used to polish the writing style. In addition, DeepL Translator was used for translations or for finding synonyms.