Anytime Multi-Agent Path Finding using Operation Parallelism in Large Neighborhood Search

Extended Abstract

Shao-Hung Chan
University of Southern California
Los Angeles, USA
shaohung@usc.edu

Zhe Chen
Monash University
Melbourne, Australia
zhe.chen@monash.edu

Dian-Lun Lin
University of Wisconsin-Madison
Madison, USA
dianlun.lin@wisc.edu

Yue Zhang
Monash University
Melbourne, Australia
yue.zhang@monash.edu

Daniel Harabor
Monash University
Melbourne, Australia
daniel.harabor@monash.edu

Sven Koenig
University of Southern California
Los Angeles, USA
skoenig@usc.edu

Tsung-Wei Huang
University of Wisconsin-Madison
Madison, USA
tsung-wei.huang@wisc.edu

Thomy Phan
University of Southern California
Los Angeles, USA
thomy.phan@usc.edu

ABSTRACT

Multi-Agent Path Finding (MAPF) is the problem of finding a set of collision-free paths for multiple agents in a shared environment while improving the solution quality. The state-of-the-art anytime MAPF algorithm is based on Large Neighborhood Search (MAPF-LNS), which is a combinatorial search algorithm that iteratively destroys and repairs a subset of collision-free paths. In this paper, we propose *Destroy-Repair Operation Parallelism for MAPF-LNS (DROP-LNS)*, a parallel framework that performs multiple destroy and repair operations concurrently to explore more regions of the search space and improve the solution quality. Unlike MAPF-LNS, DROP-LNS is able to exploit multiple threads during the search. The results show that DROP-LNS outperforms the state-of-the-art anytime MAPF algorithms, namely MAPF-LNS and LaCAM*, with respect to solution quality when terminated at the same runtime.

KEYWORDS

Multi-Agent Path Finding; Anytime Algorithm; Parallelism

ACM Reference Format:
Shao-Hung Chan, Zhe Chen, Dian-Lun Lin, Yue Zhang, Daniel Harabor, Sven Koenig, Tsung-Wei Huang, and Thomy Phan. 2024. Anytime Multi-Agent Path Finding using Operation Parallelism in Large Neighborhood Search: Extended Abstract. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 3 pages.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

1 INTRODUCTION

A *Multi-Agent Path Finding (MAPF)* instance contains an undirected and unweighted graph with a set of agents moving from their start to goal vertices [3]. At each timestep, an agent can move to an adjacent vertex or wait at its current vertex. A path p_i for an agent a_i is a sequence of vertices indicating where agent a_i is at each timestep. The cost $c(p_i)$ of path p_i is the number of timesteps to move from its start vertex to its goal vertex. The shortest path with cost d_i for an agent a_i is the path with the minimum cost while ignoring collisions with other agents. The *solution* of a MAPF instance is a set of collision-free paths, one for each agent. The objective is to minimize the *suboptimality ratio*, defined as

$$\textit{suboptimality ratio} = \frac{\sum_{a_i \in A} (c(p_i) - d_i)}{\sum_{a_i \in A} d_i}, \tag{1}$$

where $c(p_i) - d_i$ is the *delay* for agent a_i. The lower the suboptimality ratio, the better the solution quality. However, solving MAPF optimally is NP-hard, which limits the scalability [4].

Anytime algorithms are promising approaches to scalable MAPF. *Large Neighborhood Search (LNS)* is the leading anytime MAPF algorithm (MAPF-LNS) [1]. Starting with a set of collision-free paths, MAPF-LNS maintains the best-known solution $\mathbf{P}$ and iteratively selects a subset of agents as the *neighborhood*, destroys their paths, and repairs them while keeping the paths of the other agents unchanged, resulting in an updated solution P_{new}. To select the neighborhood at each iteration, MAPF-LNS uses a set of *destroy heuristics*, one with a *weight*, and selects one of the destroy heuristics at random, with their probabilities proportional to their weights. After MAPF-LNS repairs the paths, it updates the weight of the selected destroy heuristic according to the difference in the sum of path costs (SOC) of solutions $\mathbf{P}$ and P_{new}. MAPF-LNS replaces the best-known solution with solution P_{new} if the latter has a lower suboptimality ratio than the former. However, the repair operations can be time-consuming, and MAPF-LNS may struggle to lower the suboptimality ratio on large-scale MAPF instances. Thus,

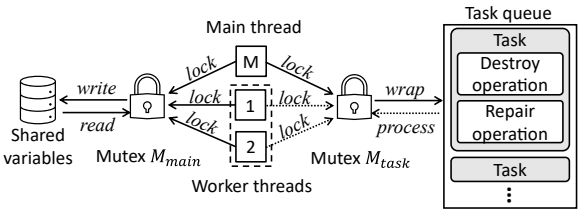

Figure 1: Illustrative example of the DROP-LNS framework with a main thread "M" and two worker threads "1" and "2". Arrows indicate the actions of each thread.

we propose *Destroy-Repair Operation Parallelism for MAPF-LNS (DROP-LNS)*, a parallel framework that performs multiple destroy and repair operations concurrently to lower the suboptimality ratio, while the best-known solution is updated asynchronously to maintain the productivity of threads, i.e., the runtime to process tasks. We show that DROP-LNS has lower suboptimality ratios than the state-of-the-art anytime MAPF algorithms when terminated at the same runtime.[1]

2 DESTROY-REPAIR OPERATION PARALLELISM FOR MAPF-LNS (DROP-LNS)

Figure 1 shows an illustrative example of the DROP-LNS framework. DROP-LNS uses a *main thread* and a set of *worker threads* to parallelize the search. The key idea is to bundle pairs of destroy and repair operations into *tasks* in the main thread and assign these tasks to idle worker threads. DROP-LNS maintains *shared variables* that can be modified by any thread, including the *best-known solution* with the minimum suboptimality ratio found so far, the weights for the destroy heuristics, and a task queue with a fixed capacity. To avoid data racing, DROP-LNS uses two *mutexes* M_{task} and M_{main} to respectively ensure that only one thread can modify (1) the task queue and (2) any other shared variables at a time.

The main thread of DROP-LNS first finds a set of collision-free paths and then, during the search, maintains the task queue with a constant capacity of tasks. An idle worker thread tries to access the task queue and, if successful, pops a task from it. Before executing the task, the worker thread tries to access and copy the shared variables to its private memory, i.e., the copied best-known solution **P** and the copied weights for the destroy heuristics. Then, it performs a pair of destroy and repair operations on the copied best-known solution based on the copied weights, resulting in an updated solution P_{new}. After the operations, the worker thread tries to access the shared variables. The best-known solution, which may be updated by other worker threads during the search and thus may no longer be **P**, is replaced with the updated solution P_{new} if the latter has a lower suboptimality ratio than the former. The weight of the destroy heuristic selected to generate solution P_{new} is updated according to the difference in the SOC of the copied solution **P** and the updated solution P_{new}. That is, the worker threads of DROP-LNS update the best-known solution asynchronously.

[1]This extended abstract is a short version of https://arxiv.org/abs/2402.01961

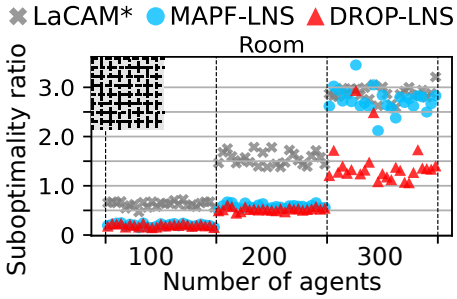

Figure 2: Suboptimality ratios of instances solved by LaCAM*, MAPF-LNS, and DROP-LNS with 8 worker threads. Instances are grouped by their numbers of agents.

DROP-LNS parallelizes the search by wrapping pairs of destroy and repair operations as concurrently executable tasks while maintaining the best-known solution and the weights for destroy heuristic selection. Compared to performing destroy and repair operations sequentially on a single worker thread, parallelism can efficiently exploit high-quality solutions for further improvement and explore different regions in the search space. Both aspects increase the chance of finding solutions with lower suboptimality ratios than MAPF-LNS.

3 EMPIRICAL EVALUATION

We use a 4-neighbor grid map `room-32-32-4` of size 32×32, denoted as `Room`, from the MAPF benchmark suite [3]. We terminate all the anytime MAPF algorithms when the runtime reaches 60 seconds. For MAPF-LNS and DROP-LNS, we generate the initial solution via LaCAM [2] and set the number of agents in a neighborhood to 16. We implement all algorithms in C++ (compiled with GCC-11.3.0) and run experiments on CentOS Linux and an AMD EPYC 7302 16-core processor with 16 GBs of memory.

As shown in Figure 2, DROP-LNS with 8 worker threads has lower suboptimality ratios than the state-of-the-art anytime MAPF algorithms, namely LaCAM* [2] and MAPF-LNS, especially on MAPF instances with 300 agents. Please refer to our full paper https://arxiv.org/abs/2402.01961 for details.

4 CONCLUSION

In this work, we presented DROP-LNS, a parallel framework that performs multiple destroy and repair operations concurrently to improve the solution quality. The empirical evaluations show that under the same runtime when the user interrupts for the solution, DROP-LNS reaches lower suboptimality ratios than the state-of-the-art anytime MAPF algorithms such as MAPF-LNS and LaCAM*. Future work includes developing more sophisticated mechanisms for synchronization, extensions to anytime bounded-suboptimal MAPF algorithms, and parallel MAPF algorithms using GPU.

ACKNOWLEDGMENTS

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1817189, 1837779, 1935712, 2121028, 2112533, and 2321786, as well as a gift from Amazon Robotics.

REFERENCES

[1] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. 2021. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 4127–4135.

[2] Keisuke Okumura. 2023. Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 243–251.

[3] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 151–159.

[4] Jingjin Yu and Steven M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 1443–1449.