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

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

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

Sven Koenig
University of Southern California
Los Angeles, USA
skoenig@usc.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 minimizing the sum of travel times. The current state-of-the-art anytime algorithm for MAPF is based on Large Neighborhood Search (LNS), called 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 LNS (DROP-LNS)*, a parallel framework that performs multiple destroy and repair processes simultaneously to explore a larger searching space under a limited time budget. Unlike MAPF-LNS, DROP-LNS is able to exploit parallelized hardware to improve the solution quality. We extend DROP-LNS to two alternatives and conduct experimental evaluations to compare the performance. The results show that DROP-LNS significantly outperforms the state-of-the-art.

KEYWORDS

Multi-Agent Path Finding, Anytime Algorithm, Parallelism

1 INTRODUCTION

A wide range of real-world applications can be formulated as *Multi-Agent Path Finding (MAPF)* problem such as autonomous warehouse [6] and unmanned aerial vehicles [1]. MAPF aims to find a set of collision-free paths, each from an assigned start location to a goal location, for multiple agents in a shared environment while minimizing the sum of travel time [5]. However, solving MAPF optimally is NP-hard, which limits the scalability of many algorithms [7]. MAPF is based on a undirected and unweighted graph, where k agents navigate from their start vertices to their goal vertices. At each discretized timestep, an agent is allowed to move to an adjacent vertex or wait at its current vertex. A path for an agent is a sequence of vertices indicating where the agent is at each timestep, with the path cost being the total timesteps for the agent to move from its start vertex to its goal vertex. The objective of MAPF is to minimize the suboptimality ratio, defined as sum of path cost/sum of individual shortest paths − 1.

Anytime algorithms are promising approaches to scalable MAPF. In particular, *Large Neighborhood Search (LNS)* is the leading anytime algorithm in solving MAPF (MAPF-LNS) [3]. Starting with a feasible solution from any existing suboptimal MAPF algorithm, MAPF-LNS iteratively selects a subset of agents, destroys their paths, and repairs them while keeping the other paths fixed. However, the repairing operations can be time-consuming, and MAPF-LNS may struggle to improve its solution on large-scale instances. In this work, we propose *Destroy-Repair Operation Parallelism for LNS (DROP-LNS)*, a parallel framework that performs multiple destroy and repair processes simultaneously to explore a larger searching space in a limited time budget. Unlike MAPF-LNS, DROP-LNS is able to exploit parallelized hardware to improve the solution quality. We also evaluate different parallelism variants and show that DROP-LNS trades off between productivity and synchronization to achieve better performance, where the former describes the concurrent execution of tasks, and the latter describes the access to the best-known solution to guide the search toward better quality [1].

2 DESTROY-REPAIR OPERATION PARALLELISM FOR MAPF-LNS

DROP-LNS uses a *main thread* and a set of *worker threads* to parallelize the search. The core idea is to wrap pairs of destroy and repair operations as *tasks* via 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 SOC found so far, the weights for each destroy heuristic, and a task queue with a fixed capacity. To avoid data racing, DROP-LNS uses two *mutexes* to ensure that only one thread can modify (1) the task queue and (2) any other shared variables at a time, respectively, as shown in Figure 1.

DROP-LNS first uses the main thread to initialize a feasible solution via LaCAM [4]. The main thread then fills the task queue

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

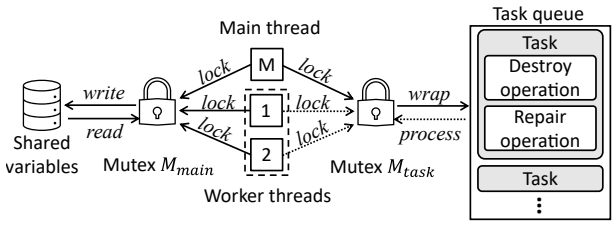

Figure 1: Illustrative example of the DROP-LNS framework with a main thread "M" and two worker threads "1" and "2". Arrows are the actions from each thread.

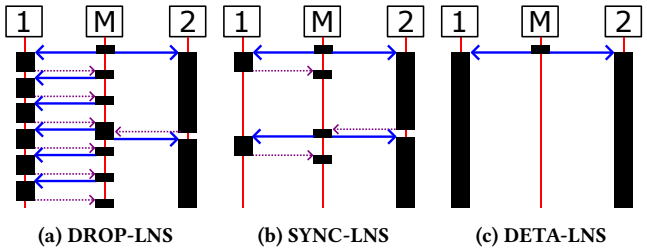

Figure 2: Conceptual timelines of parallelism variants. Black blocks indicate the *productivity*, and red lines indicate the idle time. Blue arrows indicate events where a worker thread receives tasks, and purple dotted arrows indicate events where a worker thread returns a solution.

with the fixed capacity of tasks until the time budget runs out. An idle worker thread tries to access the task queue and, if successful, pops a task from it. Before executing a task, the worker thread tries to access and copy the shared variables to its private memory, i.e., the best-known solution **P** and weights for destroy heuristics. Then, it performs a pair of destroy and repair operations to the copied best-known solution based on the copied weight of each destroy heuristic, resulting in an updated solution, denoted as P_{new}. After the operations, the worker thread tries to access the shared variables. If the updated SOC is lower than that from the shared variables, then the best-known solution is replaced with the one in the private memory. The weight value corresponding to the destroy heuristic from the private memory is updated according to the SOC difference between **P** and P_{new}. That is, the worker threads of DROP-LNS update the best-known solution asynchronously, as illustrated in Figure 2a.

3 PARALLELISM VARIANTS OF MAPF-LNS

We also implement two parallelism variants: SYNC-LNS and DETA-LNS. SYNC-LNS selects the same number of neighborhoods as the worker threads and performs a pair of destroy and repair operations individually on each worker thread. After all the worker threads complete their own destroy and repair operations, SYNC-LNS updates the best-known solution and the weight value by comparing the solution with the lowest SOC and the selected destroy heuristic. That is, it synchronizes solutions and weights at each iteration by selecting the one with the lowest SOC, as illustrated in Figure 2b. Inspired by [2] that processes LNS using four CPUs, DETA-LNS

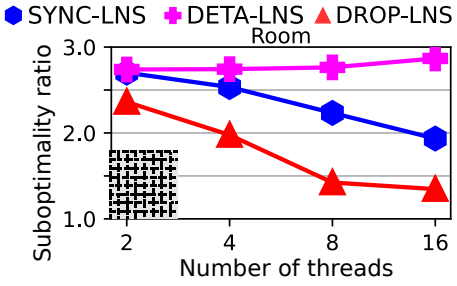

Figure 3: Average suboptimality ratios among all instances with 300 agents solved by SYNC-LNS, DETA-LNS, and DROP-LNS with 2, 4, 8, and 16 threads.

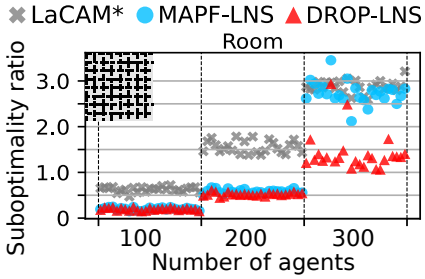

Figure 4: Suboptimality ratios of instances solved by LaCAM*, MAPF-LNS, and DROP-LNS. Instances are grouped by the number of agents.

"detaches" one MAPF-LNS process individually on a worker thread in parallel with equal weights of each destroy heuristic. When the time budget runs out, it selects the solution with the lowest SOC among all solutions generated by the worker threads. That is, DETA-LNS never synchronizes solutions and weights until the time budget runs out, as shown in Figure 2c.

4 EMPIRICAL EVALUATION

We use a 4-neighbor grid map of size 32×32 from the MAPF benchmark suite [5]. DROP-LNS outperforms SYNC-LNS and DETA-LNS while maintaining its performance as the number of threads increases, and outperforms the state-of-the-art algorithms for anytime MAPF, namely LaCAM* and MAPF-LNS, as shown in Figures 3 and 4 respectively. Please refer to our full paper in https://arxiv.org/abs/2402.01961 for more details.

5 CONCLUSION

In this work, we presented DROP-LNS, a parallel framework that performs multiple destroy and repair operations concurrently to explore more regions of the search space within a limited time budget, while the currently best-known solution is updated asynchronously to maintain the productivity of worker threads. The empirical evaluations show that DROP-LNS outperforms state-of-the-art anytime algorithms such as MAPF-LNS and LaCAM* in the MAPF benchmark. Future work includes developing more sophisticated mechanisms for synchronization and extensions to anytime bounded-suboptimal algorithms and parallel algorithms using GPU.

REFERENCES

[1] Florence Ho, Ana Salta, Ruben Geraldes, Artur Goncalves, Marc Cavazza, and Helmut Prendinger. 2019. Multi-Agent Path Finding for UAV Traffic Management. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 131–139.

[2] Florian Laurent, Manuel Schneider, Christian Scheller, Jeremy Watson, Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Konstantin Makhnev, Oleg Svidchenko, Vladimir Egorov, Dmitry Ivanov, Aleksei Shpilman, Evgenija Spirovska, Oliver Tanevski, Aleksandar Nikov, Ramon Grunder, David Galevski, Jakov Mitrovski, Guillaume Sartoretti, Zhiyao Luo, Mehul Damani, Nilabha Bhattacharya, Shivam Agarwal, Adrian Egli, Erik Nygren, and Sharada Mohanty. 2021. Flatland Competition 2020: MAPF and MARL for Efficient Train Coordination on a Grid World. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. 275–301.

[3] 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.

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

[5] 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.

[6] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine* (2008), 9–20.

[7] 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.