Exact Anytime Multi-Agent Path Finding Using Branch-and-Cut-and-Price and Large Neighborhood Search

Edward Lam[1], **Daniel Harabor**[1], **Peter J. Stuckey**[1], **Jiaoyang Li**[2]

[1]Monash University, Australia
[2]Carnegie Mellon University, USA
{edward.lam, daniel.harabor, peter.stuckey}@monash.edu, jiaoyangli@cmu.edu

Abstract

Given a set of agents on a grid, the multi-agent path finding problem aims to find a path that moves each agent from its given start location to its target location such that they do not collide and that the sum of arrival times is minimized. LNS2 is a state-of-the-art algorithm for anytime, suboptimal solving. It is an upper-bounding algorithm that repeatedly adjusts an existing solution and, being a local search, is oblivious to optimality. BCP is a state-of-the-art algorithm for exact solving. It is a lower-bounding tree search that attempts to tighten the lower bound until a solution appears. As BCP operates on the lower bound, the first solution it finds is optimal or nearly optimal, and therefore has poor anytime behavior. This paper proposes to tightly couple LNS2 and BCP to achieve better anytime, suboptimal solving while retaining the optimality guarantee of BCP. Experiments indicate that the combination achieves better anytime behavior than BCP in general and better suboptimal performance than LNS2 on congested maps.

Introduction

Given a set of agents on a grid, each with a unique start location and target location, the multi-agent path finding (MAPF) problem finds a coordinated plan, which moves every agent from its start to its target while avoiding collisions with other agents. *Exact* or *optimal* MAPF algorithms guarantee to return a collision-free plan that has the minimum overall cost (e.g., sum-of-individual-costs or makespan). However, because MAPF is NP-hard (Yu and LaValle 2013), instances with increasing numbers of agents quickly become intractable. Attempting to solve large instances using exact algorithms often results in timeout failure. In contrast, *suboptimal* algorithms relax or even disregard optimality guarantees and instead attempt to find low-cost feasible solutions quickly. A third approach, which combines these strengths, is the family of *anytime* MAPF algorithms. These approaches aim to find feasible solutions quickly and then better/cost-improving solutions given more time; eventually converging to a local minima or, in some cases, converging to optimality.

BCP (Lam et al. 2022) is a state-of-the-art exact algorithm for MAPF and it is anytime in principle. BCP performs a tree search over a database of paths. The nodes in the search tree are explored using a best-first selection strategy, which prioritizes nodes with the lowest lower bound estimate. It repeatedly solves nodes with the lowest lower bound estimate to gradually push up the lower bound until a solution is found. As BCP selects nodes with the lowest lower bound, the first solution it finds often provides an upper bound that is close to the lower bound. But this solution is found very late in the search, after a large number of nodes have been explored, and hence leads to poor anytime performance.

LNS2 (Li et al. 2022) is a state-of-the-art algorithm for suboptimal MAPF. Starting from an infeasible solution, LNS2 repeatedly replans subsets of agents to find plans with fewer conflicts, eventually converging to a conflict-free plan. A related algorithm, MAPF-LNS (Li et al. 2021a), proceeds similarly but starts from a feasible solution, which it tries to improve while time remains. In this paper, we refer to their combination (LNS2 first, then MAPF-LNS while time allows) simply as LNS2. This approach does not maintain any lower bound and is a pure upper-bounding anytime algorithm.

BCP and LNS2 tackle the problem from orthogonal points of view. This paper proposes to exploit these two viewpoints by invoking LNS2 throughout BCP's search tree. LNS2 is warm-started using solutions from BCP, and BCP's database of paths is enlarged with paths from LNS2. At all times during the search, BCP provides a lower bound and upper bound, and LNS2 provides an upper bound. The incrementally-improving upper bounds from LNS2 and the incrementally-improving lower bounds from BCP yield a combination that displays significantly better anytime behavior.

Experimental results on 3,850 instances across 12 maps demonstrate that the coupling often finds tighter lower and upper bounds than standalone BCP and LNS2 but especially on congested maps. It also proves optimality on hardly any fewer instances than BCP.

Background and Literature Review

A wide range of new algorithms have been proposed recently to tackle MAPF, each with different performance characteristics and guarantees. We review a small group of works that are most closely related.

BCP (Lam et al. 2022) is among the leading methods for optimal MAPF. This approach combines branch-and-bound (e.g., Rader 2010) together with advanced techniques from integer programming. BCP maintains a database of paths. At every node of the tree, BCP uses column generation (e.g.,

Lübbecke and Desrosiers 2005) to find better paths to add into the database, and solves a continuous relaxation of the problem of choosing optimal paths for the agents. The continuous relaxation (e.g., Rader 2010) ignores the discreteness of the problem and allows each agent to use multiple paths on the condition that the proportions of all used paths sum to 100%. A solution (to the continuous relaxation) in which at least one agent uses more than one path is a *fractional solution*. Occasionally, solving the continuous relaxation will yield an *integer solution*, in which every agent uses exactly one path (with 100% proportion), and hence, this integer solution is a valid MAPF plan. Otherwise, BCP branches, creating two children nodes in which the fractional solution is removed. Eventually, an integral solution is found at a leaf node. The search in BCP is free to solve nodes in any order and integral solutions can appear anywhere in the tree. BCP keeps track of the best integral upper bound and the search continues until the upper- and lower-bounds meet, or until the available time is exhausted. On termination, BCP returns the best known integral solution. These characteristics make BCP an *anytime* algorithm.

After branching, both of BCP's child nodes inherit the lower bound from their parent. A node selection heuristic then directs the search to the next node to solve. The default node selection heuristic is best-first search, which favors nodes with the lowest lower bound estimate coming from the continuous relaxation. BCP can also be used with depth-first search, which favors plunging deep into the tree where integral solutions reside. Given enough time, best-first search produces trees with fewer nodes than depth-first search but it can take a long time to find the first solution because the focus is shallower. Often, the majority of BCP's run-time is spent solving the root node in order to establish a global lower bound. Because solutions appear deep in the tree and because BCP spends a lot of time at the root node, the first solution is often found very late into the search or not at all within a given time limit. Thus, while BCP is an anytime algorithm *in principle*, its anytime performance *in practice* is extremely poor.

Another popular algorithm for optimal MAPF is Conflict-based Search (CBS) (Sharon et al. 2015). Similar to BCP, this algorithm also uses a two-level strategy. It comprises a high-level tree search, used for conflict resolution, and a low-level graph search, used for solving constrained single-agent path finding problems. Some variants, such as CBSH2-RCT (Li et al. 2021b), further improve performance by adding additional pruning and bounding techniques. Others, such as Lazy CBS (Gange, Harabor, and Stuckey 2019) implement lazy clause generation (Ohrimenko, Stuckey, and Codish 2009) from constraint programming. In the CBS framework, the first solution found is usually optimal, which means there is little opportunity to integrate an anytime component.

For computing suboptimal solutions, two types of algorithms are considered in the literature: bounded-suboptimal and unbounded suboptimal. Bounded suboptimal algorithms, such as EECBS (Li, Ruml, and Koenig 2021), a leading technique from the CBS family, relax the optimality criteria of exact algorithms. Instead, these approaches guarantee to return a solution whose cost is not more than some fixed (multiplicative or additive factor) larger than optimal. Bounded suboptimal methods can run much faster than exact methods and they tend to succeed more often.

Unbounded suboptimal algorithms trade away all guarantees in exchange for further performance improvements. LNS2 (Li et al. 2022) is among the leading techniques of this type. LNS2 begins by finding an initial solution using prioritized planning (Silver 2005), which plans each agent in turn and makes no attempt to resolve the conflicts. After an initial solution is found, LNS2 chooses a subset of agents and replans these agents one-at-a-time using prioritized planning (Li et al. 2021a). This process is repeated until terminated by a time limit or iteration limit. LNS2 progressively improves the upper bound with each successive solution, which makes it an anytime algorithm. However, LNS2 makes no attempt to compute a lower bound. Nevertheless, LNS2 is shown to find near-optimal and best-known solutions for a wide range of problems.

Integrating BCP and LNS2

Solutions to the continuous relaxation in BCP are often fractional and permit each agent to use multiple paths on the condition that every path is used with a fractional proportion that all sum to 100%. BCP then calls subroutines, known as *primal heuristics*, that take a fractional solution and adjust it according to a programmed strategy in the hope of finding an integral solution. BCP currently runs the primal heuristics built into its underlying integer programming solver, which are applicable to many other problems beyond MAPF. Because of their generality, these primal heuristics have no understanding of the MAPF problem and only operate on the paths already existing in BCP's database. Basically, the built-in primal heuristics randomly assign one existing path to each agent and hope that this produces a valid MAPF plan. These primal heuristics cannot generate new paths to complement the existing paths. In the proposed coupling, LNS2, which understands the MAPF problem, is developed into a primal heuristic that can generate new paths.

In its default configuration, LNS2 is initialized using a solution found by prioritized planning. In the proposed coupling, LNS2 is initialized using the paths selected by BCP for each agent with highest proportion. LNS2 is then run until a time limit. In the world of metaheuristics, this corresponds to a restart strategy, which helps with escaping local minima. If LNS2 finds a solution whose cost is better than the current upper bound in BCP, the new solution overwrites the incumbent plan. Furthermore, paths in the LNS2 solution that do not yet exist in BCP are also added to the database, allowing BCP to choose them in future iterations.

Using this bidirectional communication, solutions to the continuous relaxation from BCP are used to initialize LNS2 and solutions from LNS2 are used to generate paths and solutions for BCP. While this concept is simple to understand, there are major challenges in the implementation.

Primal heuristics must respect the decisions made by branching. For example, if the solver requires an agent to use one particular edge due to branching, solutions from primal heuristics must respect this decision. BCP runs two branching rules for creating children nodes in its search tree. The first

branching rule stipulates that an agent's arrival time must be either before or after a given time. The second branching rule requires an agent to either visit an edge or avoid an edge. LNS2 currently has no mechanism to enforce the path length for an agent nor to enforce the use of an edge. We allow LNS2 to ignore the branching decisions in BCP and therefore cannot haphazardly add paths found by LNS2 into BCP's database. Instead, we hack into the underlying integer programming solver and force it to backtrack to the root node where no decisions are made, add the paths, and then repropagate down from the root node to the current node. Implementing this backtracking and repropagation is tricky as the underlying integer programming solver does not natively support this functionality.

Experimental Results

Experiments are run on 3,850 instances over 12 maps. Ten instances are run for each map and each number of agents. The maps are categorized into five classes: three warehouses, one city, two maps from computer games, four randomly generated squares and two randomly generated mazes. More agents are added to the first warehouse map 10x30-w5 because the original instances are too small. The other maps are retrieved from the Moving AI repository (Stern et al. 2019). All instances are run for 60 seconds.

The following five anytime algorithms are compared:

- **BCP BFS**: Running BCP with best-first search.
- **BCP DFS**: Running BCP with depth-first search.
- **LNS2**: Running LNS2.
- **BCP then LNS2**: Running BCP for half the time (30 seconds) and then running LNS2 for the remaining half, with no communication between the two.
- **BCP with LNS2**: Running LNS2 for 30 seconds to initialize the path database of BCP at the root node, and then running LNS2 initialized with the highest-proportion paths from BCP for 0.2 seconds at all later nodes in BCP.

All variants of BCP are exact and will find an optimal solution eventually. If BCP terminates with a suboptimal solution, it is because it has not yet found an optimal solution within the time limit. All solvers are implemented in C++ and are available from the authors' websites.

Figure 1 plots the optimality gap averaged over the ten instances. The optimality gap

$$\frac{\text{upper bound} - \text{lower bound}}{\text{upper bound}}$$

measures the relative difference between the lower bound and upper bound. If an algorithm fails to find a solution to any of the ten instances, it is shown as ∞ in the plots. For standalone LNS2, the lower bound for the purpose of computing the optimality gap is the sum-of-costs of the individually-optimal paths of the agents.

BCP is the weakest and begins to fail at the fewest number of agents. Depth-first search sometimes performs better than best-first search but still fails at medium-size instances. Pursuing the lower bound is just not productive use of time given the short time limit.

Map	BCP BFS	BCP DFS	LNS2	BCP then LNS2	BCP with LNS2
10x30-w5	**10%**	9%	0%	9%	9%
warehouse-10-20-10-2-1	33%	32%	5%	33%	**33%**
warehouse-10-20-10-2-2	**65%**	63%	11%	61%	63%
Berlin_1_256	**57%**	54%	12%	52%	52%
brc202d	26%	**26%**	1%	24%	23%
ost003d	33%	**34%**	4%	31%	31%
empty-8-8	**98%**	93%	34%	96%	97%
empty-32-32	**32%**	**32%**	9%	31%	31%
random-32-32-10	28%	28%	5%	28%	**29%**
random-32-32-20	**19%**	19%	3%	19%	19%
maze-128-128-1	16%	16%	11%	17%	**18%**
maze-128-128-2	46%	45%	15%	45%	**46%**
All	**38%**	37%	8%	36%	36%

Table 1: Percentage of instances solved optimally. The most solved instances are shown in bold. The trivial lower bound is used to determine optimality for LNS2.

LNS2 paired with the trivial sum-of-costs lower bound already obtains tight optimality gaps. This is a consequence of the MAPF problem because shuffling the order of agents within the prioritized planning step of LNS2 or inserting waits into an agent's path does not worsen a solution too much. Therefore, the optimal cost is likely to be within a few wait actions away from the trivial lower bound. Overall, LNS2 performs remarkably well due to this characteristic.

The sequential BCP then LNS2 solver sometimes outperforms LNS2, especially on instances with lower agent counts, where collisions are easier to resolve. The plots for BCP then LNS2 and standalone LNS2 cross over in warehouse-10-20-10-2-1, warehouse-10-20-10-2-2, Berlin_1_256, ost003d, empty-32-32, random-32-32-10 and random-32-32-20, suggesting that resolving conflicts at high agent counts in order to compute a lower bound is prohibitively expensive.

The integrated BCP and LNS2 algorithm essentially dominates the other algorithms on all but two maps. The combined algorithm performs worse than LNS2 on the large game maps brc202d and ost003d, which have fewer obstacles and less congestion. The combined solver generally achieves the lowest optimality gaps on the other maps, which are smaller and/or have more obstacles, increasing congestion. The maze-128-128-1 map is arguably the hardest because the optimality gap blows up at the fewest number of agents. BCP fails at the fewest number of agents on this map and the sequential solver performs almost identically to LNS2, indicating that BCP makes no contribution. In contrast, the integrated algorithm dominates the other solvers, demonstrating that the improvement is due to the synergy between LNS2 and BCP.

These observations lead to the conclusion that the combined solver generally outperforms the other algorithms on maps with high congestion (e.g., 10x30-w5, warehouse-10-20-10-2-1, warehouse-10-20-10-2-2 and maze-128-128-1), whereas LNS2 performs better on larger maps with lower congestion (e.g., brc202d and ost003d), where the trivial lower bound is more accurate.

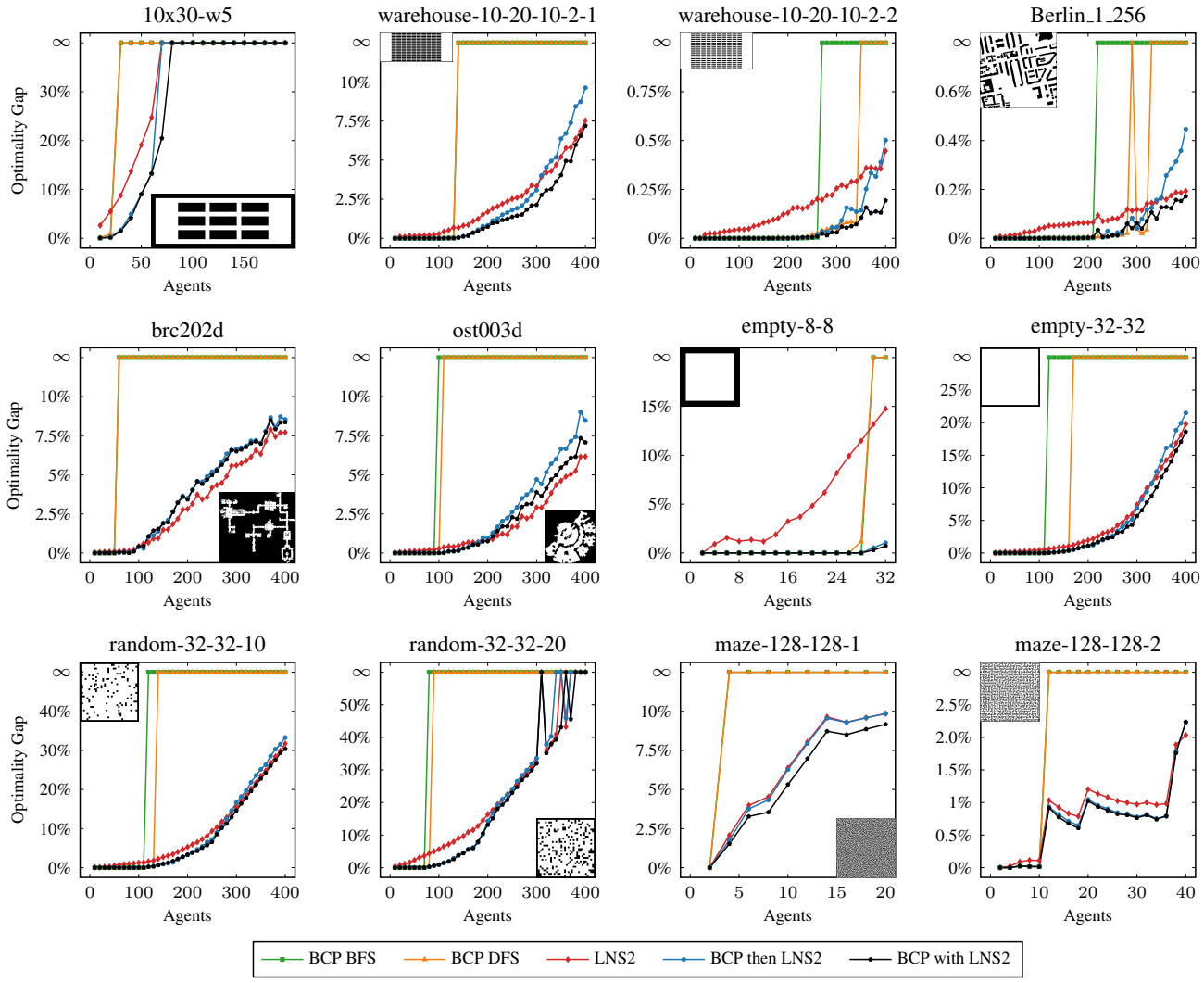

Figure 1: Optimality gap for each map and each number of agents averaged over ten instances. Lower is better. An infinite gap is shown if a solver fails to find an upper bound to at least one of the ten instances with the given number of agents.

Table 1 shows the percentage of instances solved optimally by each algorithm. BCP with best-first search finds the most optimal solutions, skewed by the instances with fewer agents. The combined solver finds more optimal solutions than BCP on warehouse-10-20-10-2-1, random-32-32-10, maze-128-128-1 and maze-128-128-2, but overall finds fewer optimal solutions than BCP. LNS2 finds a surprising number of optimal solutions even when just using the trivial lower bound, i.e., the trivial lower bound is optimal and LNS2 can find a solution with this cost.

Conclusions

This paper proposes to integrate BCP and LNS2, two leading algorithms for optimal and suboptimal MAPF respectively. While BCP itself is an anytime algorithm in theory, it exhibits poor anytime behavior in practice because the first few solutions that it finds are near-optimal and occur very late in the search. LNS2 offers no proof of optimality but can find low-cost solutions very quickly. The proposed combination first calls LNS2 to find an initial solution and warm-starts BCP with this solution. Then, LNS2 is called throughout the search tree of BCP, and is itself warm-started from the fractional solutions found by BCP.

Compared against standalone BCP and LNS2, and running BCP and LNS2 for half the time in sequence, the integrated algorithm achieves better optimality gaps on most maps and loses to LNS2 only on two large game maps with many open spaces and hence lower congestion. The combined algorithm is even competitive with standalone BCP in proving optimality, only closing 2% fewer instances despite spending a large amount of time on upper bounding. The results also show that the trivial lower bound is acceptable in most cases because the agents can often move a few cells away or wait a few timesteps, increasing the total cost by a small amount and hence degrading the gap by only a few percentages.

Acknowledgments

We are grateful to our anonymous reviewers, whose comments have improved this paper. This research is supported by the Australian Research Council under Discovery Grants DP190100013 and DP200100025 and by a gift from Amazon.

References

Gange, G.; Harabor, D.; and Stuckey, P. 2019. Lazy CBS: Implict Conflict-Based Search Using Lazy Clause Generation. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS)*, volume 29, 155–162.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research*, 144: 105809.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P.; and Koenig, S. 2021a. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021b. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301: 103574.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 12353–12362.

Lübbecke, M. E.; and Desrosiers, J. 2005. Selected topics in column generation. *Operations Research*, 53(6): 1007–1023.

Ohrimenko, O.; Stuckey, P. J.; and Codish, M. 2009. Propagation via lazy clause generation. *Constraints*, 14(3): 357–391.

Rader, D. J. 2010. *Deterministic operations research: models and methods in linear optimization*. John Wiley & Sons.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219: 40–66.

Silver, D. 2005. Cooperative Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 117–122.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Yu, J.; and LaValle, S. M. 2013. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, 3612–3617. IEEE.