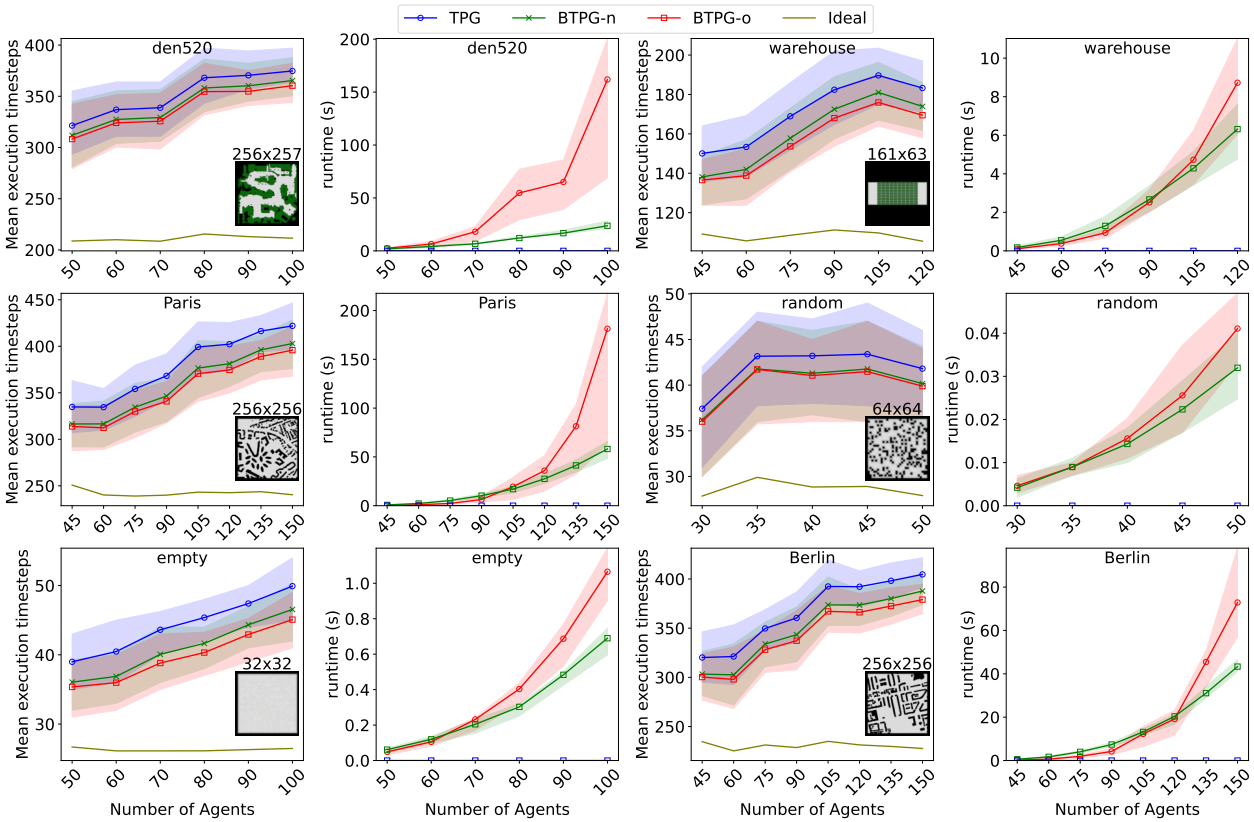

Figure 7: Performance of TPG, BTPG-nave, and BTPG-optimized. The shaded area represents the interquartile range (25%-75%) of the data distribution. The runtime of BTPG-n and BTPG-o is the time for the two algorithms to finish finding their respective bidirectional pairs. Even though BTPG-o takes longer to finish finding bidirectional pairs, it finds around 2x more bidirectional pairs than BTPG-n, so it is actually more efficient than BTPG-n (see Figure 9).

	den520		warehouse		Paris		random		empty		Berlin	
	BTPG-n	BTPG-o	BTPG-n	BTPG-o	BTPG-n	BTPG-o	BTPG-n	BTPG-o	BTPG-n	BTPG-o	BTPG-n	BTPG-o
Mean imp.	5.9%	**8.9%**	12.1%	**17.9%**	10.6%	**14.6%**	12.9%	**15.2%**	14.5%	**20.9%**	9.6%	**14.6%**
Median imp.	5.7%	**8.1%**	12.3%	**17.8%**	10.5%	**14.2%**	10.3%	**12.2%**	14.0%	**20.0%**	9.2%	**14.2%**
Max imp.	14.0%	**19.3%**	25.0%	**35.3%**	20.7%	**25.4%**	50.0%	**63.6%**	37.5%	**42.9%**	21.3%	**24.7%**
Min imp.	1.1%	**2.9%**	2.7%	**6.7%**	4.1%	**7.0%**	0.0%	0.0%	3.3%	**7.4%**	3.2%	**7.6%**
# Type-2 edges	26,738		15,084		24,401		1,151		2,898		27,922	
# Singleton edges	2,172		1,051		2,688		146		817		2,265	
# Bi-Pairs found	663	**1,008**	368	**532**	1,254	**1,725**	60	**76**	248	**360**	1018	**1,409**
# Used Bi-Pairs	38	**68**	36	**57**	82	**125**	6	6	25	**40**	68	**112**
BTPG runtime (s)	23.72	161.88	6.31	8.72	58.12	181.33	0.03	0.04	0.69	1.06	43.32	72.87
MAPF runtime (s)	1.93		9.44		1.12		19.80		1.11		9.90	

Table 1: Statistics of BTPG-nave/optimized. The number of agents for the six maps selected for the statistics are 100, 120, 150, 50, 100, and 150, respectively. All data in the second block are averages of 10 scenarios for each map; Bi-Pairs: bidirectional pairs; imp.: improvement; Singleton edges: type-2 edges that cannot be grouped.

6 Empirical Evaluation

Our experiments work as follows. First, we use the optimal MAPF solver CBSH2-RTC (**?**) to obtain the MAPF plan for each given MAPF instance. We have in total 3,900 simulations, consisting of six different benchmark maps from (**?**), as shown in Figure 7, each map with five to eight different numbers of agents, and each number of agents with ten random instances. The largest number of agents for each map is determined by the largest number of agents CBSH2-RTC can solve within 2 minutes. Then, we convert each MAPF plan into a TPG (for which the runtime is negligible). Next, we construct BTPG using our proposed algorithms BTPG-naive and BTPG-optimized. Last, we simulate the TPG and BTPG execution policies with 10% agents having a 30% chance of being delayed by 5 timesteps at each non-delayed timestep. Each instance is simulated with 10 random seeds,

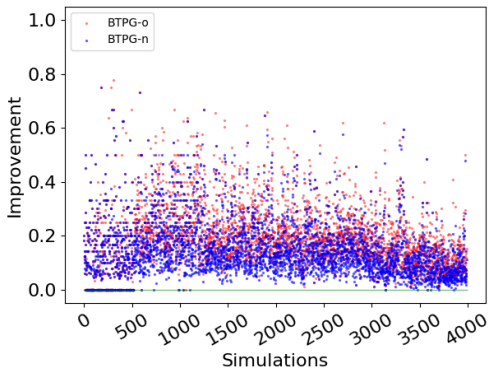

Figure 8: Improvement of BTPG over TPG per instance. We find no instances with negative improvements and 3.6%, 23.3%, 41.9%, and 31.2% of instances with no improvement, 0-10% improvement, 10-20% improvement, and >20% improvement across all simulations, respectively.

resulting in 3,900 simulations per algorithm. All algorithms were implemented in C++ [1], and all experiments were run on a PC with a 3.60 GHz Intel i7-12700KF CPU and 32 GB of RAM.

Execution Time Figure 7 reports the *mean execution timesteps*, namely the average number of timesteps that each agent takes to reach its target location during execution. We include a lower bound called *Ideal*, which is the sum of timesteps in the original MAPF plan plus the total delay timesteps of the delayed agents, then divided by the number of agents. Essentially, *Ideal* represents an overly optimistic scenario where delayed agents do not cause any additional waits for other agents. As shown, BTPG-o consistently outperforms BTPG-n, which in turn consistently outperforms TPG, across all maps and all numbers of agents.

To quantify the execution time improvement of BTPG over TPG, we define *improvement* as $\frac{T_{TPG}-T_{BTPG}}{T_{TPG}-T_{Ideal}}$, where T_{TPG}, T_{BTPG}, and T_{Ideal} are the mean execution timesteps of TPG, BTPG, and Ideal, respectively. Figure 8 shows that our improvement is always non-negative over 3,900 simulations, even though, as mentioned in Section 4, we could have negative improvements in theory. The top block of Table 1 further provides detailed numbers on the improvements over the instances with the largest number of agents for each map. The median improvement of BTPG-o is in the range of 8-20%, with maximum values around 19-64%.

Bidirectional Pairs The middle block of Table 1 reports how many bidirectional pairs that our algorithms find and that are useful. While the TPG has thousands or even ten thousands of type-2 edges, only about 10% of these edges are not grouped and have the chance to be changed to bidirectional pairs. After BTPG-o evaluates these singleton type-2 edges, about 50% are changed to bidirectional pairs. We define a bidirectional pair as *used* if the agents select the reversed edge rather than the original TPG type-2 edge

[1] https://github.com/YifanSu1301/BTPG

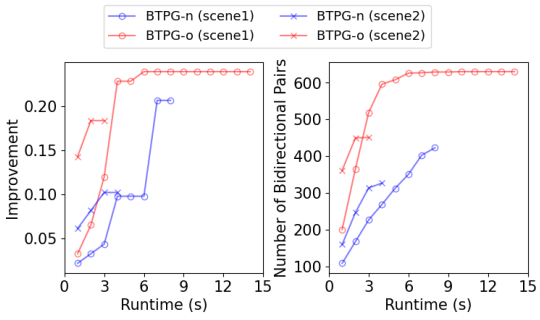

Figure 9: Anytime behavior of BTPG-n/o on `warehouse` with 120 agents. Scenes 1 and 2 are scenarios with the longest and shortest BTPG-o runtimes, respectively. The appendix has similar results on other maps.

during execution. We see that roughly 10% of bidirectional pairs are used. Although this is a small percentage over the total number of type-2 edges, these used bidirectional pairs are the contributors to the significant reduction in execution time that we reported above.

Runtime Both Figure 7 and the bottom block of Table 1 report the CPU runtime of our algorithms. As expected, BTPG-o is slower than BTPG-n. However, we find that the longer runtime of BTPG-o is due to finding more bidirectional pairs rather than inefficiency. Figure 9 plots the anytime behavior of both algorithms. For any given cut-off time, BTPG-o finds more bidirectional pairs and thus leads to better improvement than BTPG-n. BTPG-o is more efficient because its DFS can skip over more edges than BTPG-n.

7 Conclusion

We constructed a new graphical representation of passing orders in the MAPF plan, BTPG, by proposing the concept of bidirectional pairs. The main difference between BTPG and TPG lies in the fact that agents can switch the order of passing certain locations during execution. Two algorithms, BTPG-n and BTPG-o, are proposed to construct a BTPG. The results indicate that following BTPGs consistently outperform following TPGs by 8-20% when agents get delayed. Also, we show our proposed algorithms are anytime, and given a fixed time budget, BTPG-o outperforms BTPG-n. Overall, we convincingly show that allowing switching dependencies in a MAPF plan allows us to improve execution time without replanning.

Acknowledgments

The research is supported by the National Science Foundation (NSF) under grant number 2328671. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

Fuga saepe consequatur perferendis sed repellendus id incidunt in porro sapiente, voluptatibus praesentium beatae odio dolore dolores impedit, eum non praesentium?Expedita ullam suscipit rerum consequuntur ratione nihil minima magnam facilis unde dolore, enim quisquam excepturi sint