# A New Constraint Satisfaction Perspective on Multi-Agent Path Finding: Preliminary Results

Extended Abstract

Jiangxing Wang
University of Southern California
jiangxiw@usc.edu

Jiaoyang Li
University of Southern California
jiaoyanl@usc.edu

Hang Ma
University of Southern California
hangma@usc.edu

Sven Koenig
University of Southern California
skoenig@usc.edu

T. K. Satish Kumar
University of Southern California
tkskwork@gmail.com

## ABSTRACT

In this paper, we adopt a new perspective on the Multi-Agent Path Finding (MAPF) problem and view it as a Constraint Satisfaction Problem (CSP). A variable corresponds to an agent, its domain is the set of all paths from the start vertex to the goal vertex of the agent, and the constraints allow only conflict-free paths for each pair of agents. Although the domains and constraints are only implicitly defined, this new CSP perspective allows us to use successful techniques from CSP search. With the concomitant idea of using matrix computations for calculating the size of the reduced domain of an uninstantiated variable, we apply Dynamic Variable Ordering and Rapid Random Restarts to the MAPF problem. In our experiments, the resulting simple polynomial-time MAPF solver, called Matrix MAPF solver, either outperforms or matches the performance of many state-of-the-art solvers for the MAPF problem and its variants.

**ACM Reference Format:**
Jiangxing Wang, Jiaoyang Li, Hang Ma, Sven Koenig, and T. K. Satish Kumar. 2019. A New Constraint Satisfaction Perspective on Multi-Agent Path Finding: Preliminary Results. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

## 1 INTRODUCTION

The Multi-Agent Path Finding (MAPF) problem is the problem of assigning conflict-free paths to agents from their start vertices to their goal vertices such that an optimization objective (like the makespan or sum-of-costs) is minimized. It is important for many real-world problems [11, 12, 20, 22]. Despite the fact that the MAPF problem is NP-hard to solve optimally [24], many practical MAPF solvers have been developed in the past few years [4–7, 9, 13–15, 17, 19, 21, 23]. Many variants of the MAPF problem are also NP-hard to solve optimally and can be solved by extending these MAPF solvers, for example, MAPF with deadlines [10] and MAPF with large agents [8].

In this paper, we adopt a new perspective on the MAPF problem and view it as a Constraint Satisfaction Problem (CSP). A variable corresponds to an agent, its domain is the set of all paths from the start vertex to the goal vertex of the agent, and the constraints allow only conflict-free paths for each pair of agents. Although the domains and constraints are only implicitly defined, Dynamic Variable Ordering (DVO) [2] can still be successfully applied since it only requires knowledge of the cardinality of the reduced domain of an uninstantiated variable, which can be obtained using matrix multiplication on the adjacency matrix of the graph. We use this insight to develop a simple polynomial-time prioritized MAPF solver [14, 16]. Moreover, we use Rapid Randomized Restart (RRR) strategies [3] to make our Matrix MAPF solver probabilistically complete.

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, 1817189, and 1837779, as well as a gift from Amazon.

## 2 MATRIX MAPF SOLVER

**Algorithm 1** MatrixMAPFSolver: A MAPF solver that uses randomized runs for a given makespan.

```
Input:
 1: Graph $G = (V, E)$
 2: AgentList $A \leftarrow \{a_1, a_2 \ldots a_N\}$
 3: Makespan $T$
 4: Maxruns $R$
Output:
 5: PathSet $P$
 6: 
 7: MatrixList Movement[$0 \ldots T-1$]
 8: MatrixList State[$0 \ldots T$]
 9: for $r = 1 \ldots R$ do
10:     $P \leftarrow \{\}$
11:     for $t = 0 \ldots T-1$ do
12:         Movement[$t$] $\leftarrow Adj(G)$    / adjacency matrix of $G$ /
13:     end for
14:     State[0] $\leftarrow I$    / identity matrix /
15:     $A\_current \leftarrow A$
16:     while $A\_current$ is not empty do
17:         for $t = 0 \ldots T-1$ do
18:             State[$t+1$] $\leftarrow$ State[$t$] $\cdot$ Movement[$t$]
19:         end for
20:         $MinNumPath \leftarrow \min_{a_i \in A\_current}$ State[$T$]$(s_i, g_i)$
21:         if $MinNumPath < 1$ then
22:             Break
23:         end if
24:         $a \leftarrow \operatorname{argmin}_{a_i \in A\_current}$ State[$T$]$(s_i, g_i)$
25:         $p \leftarrow$ PathPlanningForSingleAgent($a$, Movement, State)
26:         $P \leftarrow P \cup \{p\}$
27:         Movement $\leftarrow$ ModifyMovement(Movement, $p$)
28:         $A\_current \leftarrow A\_current \setminus \{a\}$
29:     end while
30:     if $P$ contains conflict-free paths for all agents then
31:         return $P$
32:     end if
33: end for
34: return Failure
```

In this section, we describe our Matrix MAPF solver. For the outer loop [Line 9-33], our Matrix MAPF solver tries at most $R$ (in our experiments: 15) randomized runs to solve the MAPF problem within a specified makespan $T$. In each iteration [Line 16-29], the most constrained agent, namely the one with the fewest available conflict-free paths from its start vertex $s_i$ to its goal vertex $g_i$, is identified [Line 20]. For this agent, a path of length $T$ is chosen at random [Line 25]. This path is treated as a dynamic obstacle for all future agents [Line 27] by modifying Movement[$t$]: Some entries in Movement[$t$] ($t = 1 \dots T - 1$) are set to 0 based on $p[t]$ and $p[t-1]$ so that movements of all future agents that create collisions with the movements of this agent are prohibited. If the most constrained agent has no paths available from its start vertex to its goal vertex, then the current run is terminated unsuccessfully [Line 22] and the next run is initiated.

## 3 EXPERIMENTS

In this section, we compare our MATRIX MAPF solver against state-of-the-art solvers for MAPF and MAPF with deadlines [10]. All experiments are conducted on a 2.50 GHz Intel Core i5-3210M laptop running Ubuntu 16.04 with 6 GB RAM. All results are averaged over 100 MAPF instances on 2-D 4-neighbor grids of size $10 \times 10$. Each cell is blocked independently with a given probability. The start and goal vertices of each agent are selected randomly from the unblocked cells such that no two agents have the same start vertex or the same goal vertex. A time limit of 100 seconds is used for the "MAPF" experiments, and a time limit of 60 seconds is used for the "MAPF with Deadlines" and "Makespan Minimization" experiments.

### 3.1 MAPF

In this section, we find feasible solutions. We compare our Matrix MAPF solver against four well-known MAPF solvers with a variety of (or missing) optimization objectives: Cooperative A* [14] (without a strategy for ordering agents), Push and Swap [9], and ECBS for sum-of-costs [1] with suboptimality bounds 2 and 3. Cooperative A* is provided with a makespan limit of $L + 0.25 \times N$, and our Matrix MAPF solver is provided with a fixed makespan of $L + 0.25 \times N$, where $L$ is the largest distance between any two vertices in the graph and $N$ is the number of agents.

| Obstacle Density/Agents | | 10%/20 | 10%/30 | 10%/40 | 20%/20 | 20%/30 | 20%/40 |
|---|---|---|---|---|---|---|---|
| Matrix | Success Rate | **100** | **99** | **100** | **99** | **94** | 45 |
| | Makespan | 18.48 | 21.65 | 24.43 | **20.10** | **26.98** | **66.19** |
| | Runtime | 0.118 | 1.220 | **0.982** | **1.357** | **7.382** | 61.636 |
| Push and Swap | Success Rate | 91 | 52 | 16 | 72 | 27 | 12 |
| | Makespan | 34.53 | 65.47 | 91.47 | 49.07 | 84.25 | 97.06 |
| | Runtime | 9.005 | 48.008 | 84.008 | 28.007 | 73.011 | 88.022 |
| Cooperative A* | Success Rate | 74 | 22 | 1 | 22 | 1 | 0 |
| | Makespan | 37.03 | 81.61 | 99.16 | 81.41 | 99.17 | 100 |
| | Runtime | 26.059 | 78.028 | 99.002 | 78.017 | 99.001 | 100 |
| ECBS(2) | Success Rate | **100** | **99** | 99 | 98 | 89 | 35 |
| | Makespan | **16.33** | **20.39** | **23.24** | 21.87 | 31.79 | 73.37 |
| | Runtime | **0.002** | **1.024** | 2.490 | 3.084 | 13.458 | 71.522 |
| ECBS(3) | Success Rate | **100** | **99** | 98 | 97 | 93 | **46** |
| | Makespan | 16.69 | 22.42 | 26.55 | 24.18 | 31.69 | 68.00 |
| | Runtime | 0.003 | 1.034 | 3.205 | 4.510 | 9.713 | **61.344** |

**Table 1: Results for the "MAPF" experiment. Obstacle density refers to the probability with which each cell is blocked, and success rate refers to the percentage of solved MAPF instances (that is, for which conflict-free paths are found for all agents). The makespan and runtime calculations use 100 as makespan and runtime, respectively, for each unsolved MAPF instance.**

From Table 1, we observe that, in terms of the success rate, our Matrix MAPF solver outperforms the other solvers in five of the six scenarios, but the ECBS variants perform at a similar level. In terms of the makespan, the ECBS(2) variant performs best in the three scenarios with obstacle density 10% (even though it is bounded-suboptimal with respect to the sum-of-costs), while our Matrix MAPF solver performs best in the three scenarios with obstacle density 20%. In terms of runtime, the ECBS variants and our Matrix MAPF solver each perform best in three of the six scenarios.

### 3.2 MAPF with Deadlines

In this section, we maximize the number of agents that reach their goal vertices within 15 time steps [10]. We use our Matrix MAPF solver by terminating successfully on Line 31 whenever a solution is found where all agents are instantiated and returning the solution with the most instantiated agents over all runs otherwise. We compare our Matrix MAPF solver against five MAPF solvers presented in [10]: CBS-DL, DBS, and MA-DBS with merge thresholds 0, 10, and 100.

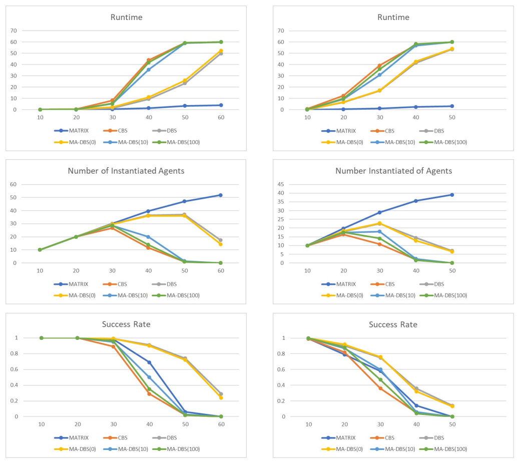

**Figure 1: Results for the "MAPF with Deadlines" experiment. The left column shows results for scenarios with obstacle density 10%. The right column shows results for scenarios with obstacle density 20%. The x-axes show the number of agents. Instantiated agents refer to agents that reach their goal vertices by the deadline, and success rate refers to the percentage of instances where all agents are instantiated.**

From Figure 1, we observe that, in terms of the runtime and the number of instantiated agents, our Matrix MAPF solver outperforms the other MAPF solvers. In terms of the success rate, it also mostly outperforms the other MAPF solvers, except for the DBS and MA-DBS(0) variants.

### 3.3 Makespan Minimization

In this section, we minimize the makespan. We use our Matrix MAPF solver [Line 9-34] to find a small makespan by initializing $T$ with a lower bound on the makespan (in our experiments: 0) and incrementing it (by 1) after each failed attempt [Line 34]. We compare our Matrix MAPF solver against optimal solutions obtained with a SAT-based MAPF solver [18].

| Obstacle Density/Agents | 10%/20 | 10%/30 | 10%/40 | 20%/20 | 20%/30 | 20%/40 |
|---|---|---|---|---|---|---|
| Available MAPF Instances | 100 | 99 | 98 | 70 | 66 | 64 |
| Success Rate | 100.0 | 100.0 | 98.0 | 100.0 | 98.5 | 56.3 |
| Optimality Rate | 99.0 | 94.9 | 64.3 | 87.1 | 50.0 | 4.7 |
| Suboptimality Rate | 0.08 | 0.89 | 6.88 | 1.11 | 15.57 | 67.05 |

**Table 2: Results for the "Makespan Minimization" experiment. Optimality rate refers to the percentage of MAPF instances for which our Matrix MAPF solver produces the optimal makespans, and suboptimality rate = 100 × (found makespan−optimal makespan)/optimal makespan averaged over all solved MAPF instances for which our Matrix MAPF solver finds a makespan.**

From Table 2, we observe that our Matrix MAPF solver always finds the optimal makespans for MAPF instances with 10% obstacle density and 20 or 30 agents. For the other scenarios, it still finds low makespans. As the obstacle density and the number of agents increase, it finds larger and larger numbers of suboptimal makespans and these makespans get larger and larger.

## REFERENCES

[1] M. Barer, G. Sharon, R. Stern, and A. Felner. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *SoCS*.

[2] X. Chen and P. Van Beek. 2001. Conflict-directed backjumping revisited. *Journal of Artificial Intelligence Research* 14 (2001), 53–81.

[3] L. Cohen, G. Wagner, D. Chan, H. Choset, N. Sturtevant, S. Koenig, and T. K. S. Kumar. 2018. Rapid randomized restarts for multi-agent path finding solvers. In *SoCS*.

[4] B. de Wilde, A. W. ter Mors, and C. Witteveen. 2013. Push and rotate: Cooperative multi-agent path planning. In *AAMAS*. 87–94.

[5] E. Erdem, D. G. Kisa, U. Oztok, and P. Schueller. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*. 290–296.

[6] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. C. Holte, and J. Schaeffer. 2014. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research* 50 (2014), 141–187.

[7] M. M. Khorshid, R. C. Holte, and N. R. Sturtevant. 2011. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *SoCS*.

[8] J. Li, P. Surynek, A. Felner, H. Ma, T. K. S. Kumar, and S. Koenig. 2019. Multi-agent path finding for large agents. In *AAAI*.

[9] R. Luna and K. E. Bekris. 2011. Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*. 294–300.

[10] H. Ma, G. Wagner, A. Felner, J. Li, T. K. S. Kumar, and S. Koenig. 2018. Multi-agent path finding with deadlines. In *IJCAI*. 417–423.

[11] H. Ma, J. Yang, L. Cohen, T. K. S. Kumar, and S. Koenig. 2017. Feasibility study: Moving non-Homogeneous teams in congested video game environments. In *AIIDE*. 270–272.

[12] R. Morris, C. Pasareanu, K. Luckow, W. Malik, H. Ma, T. K. S. Kumar, and S. Koenig. 2016. Planning, scheduling and monitoring for airport surface operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*.

[13] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[14] D. Silver. 2005. Cooperative pathfinding. In *AIIDE*. 117–122.

[15] T. S. Standley. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*. 173–178.

[16] N. R. Sturtevant and M. Buro. 2006. Improving collaborative pathfinding using map abstraction.. In *AIIDE*. 80–85.

[17] P. Surynek. 2009. A novel approach to path planning for multiple robots in bi-connected graphs. In *ICRA*. 3613–3619.

[18] P. Surynek. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*. 564–576.

[19] P. Surynek. 2015. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *IJCAI*. 1916–1922.

[20] M. M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. 2015. CoBots: Robust symbiotic autonomous mobile service robots.. In *IJCAI*. 4423.

[21] K. Wang and A. Botea. 2011. MAPP: A scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research* 42 (2011), 55–90.

[22] P. R. Wurman, R. D'Andrea, and M. Mountz. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29, 1 (2008), 9.

[23] J. Yu and S. M. LaValle. 2013. Planning optimal paths for multiple robots on graphs. In *ICRA*. 3612–3617.

[24] J. Yu and S. M. LaValle. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*. 1444–1449.