On the Tour Towards DPLL(MAPF) and Beyond

Pavel Surynek[1 [0000−0001−7200−0542]]

Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Praha 6, Czech Republic
pavel.surynek@fit.cvut.cz

Abstract. We discuss milestones on the tour towards DPLL(MAPF), a *multi-agent path finding* (MAPF) solver fully integrated with the Davis–Putnam-Logemann–Loveland (DPLL) propositional satisfiability testing algorithm through *satisfiability modulo theories* (SMT). The task in MAPF is to navigate agents in an undirected graph in a non-colliding way so that each agent eventually reaches its unique goal vertex. At most one agent can reside in a vertex at a time. Agents can move instantaneously by traversing edges provided the movement does not result in a collision. Recently attempts to solve MAPF optimally w.r.t. the sum-of-costs or the makespan based on the reduction of MAPF to propositional satisfiability (SAT) have appeared. The most successful methods rely on building the propositional encoding for the given MAPF instance lazily by a process inspired in the SMT paradigm. The integration of satisfiability testing by the SAT solver and the high-level construction of the encoding is however relatively loose in existing methods. Therefore the ultimate goal of research in this direction is to build the DPLL(MAPF) algorithm, a MAPF solver where the construction of the encoding is fully integrated with the underlying SAT solver. We discuss the current state-of-the-art in MAPF solving and what steps need to be done to get DPLL(MAPF). The advantages of DPLL(MAPF) in terms of its potential to be alternatively parametrized with MAPF^R, a theory of continuous MAPF with geometric agents, are also discussed.

Keywords: multi agent path finding (MAPF), propositional satisfiability (SAT), Davis–Putnam-Logemann–Loveland (DPLL), satisfiability modulo theories(SMT)

1 Introduction

In *multi-agent path finding* (MAPF) [11,18,19,20,22,24,30] the task is to navigate agents from given starting positions to given individual goals. The standard version of the problem takes place in undirected graph $G = (V, E)$ where agents from set $A = \{a_1, a_2, ..., a_k\}$ are placed in vertices with at most one agent per vertex. The initial configuration of agents in vertices of the graph can be written as $\alpha_0 : A \to V$ and similarly the goal configuration as $\alpha_+ : A \to V$. The task of navigating agents can be expressed as a task of transforming the initial configuration of agents $\alpha_0 : A \to V$ into the goal configuration $\alpha_+ : A \to V$.

Movements of agents are instantaneous and are possible across edges into neighbor vertices assuming no other agent is entering the same target vertex. This formulation

permits agents to enter vertices being simultaneously vacated by other agents. Trivial case when a pair of agents swaps their positions across an edge is forbidden in the standard formulation. We note that different versions of MAPF exist where for example agents always move into vacant vertices [26]. We usually denote the configuration of agents at discrete time step t as $\alpha_t : A \rightarrow V$. Non-conflicting movements transform configuration α_t *instantaneously* into next configuration α_{t+1}. We do not consider what happens between t and $t + 1$ in this discrete abstraction. Multiple agents can move at a time hence the MAPF problem is inherently parallel.

In order to reflect various aspects of real-life applications variants of MAPF have been introduced such as those considering *kinematic constraints* [9], *large agents* [12], or *deadlines* [15] - see [14] for more variants.

1.1 Lazy Construction of SAT Encodings

This paper summarizes the development SMT-CBS [28], a novel optimal MAPF algorithm that **unifies** two major approaches to solving MAPF optimally: a **search-based** approach represented by *conflict-based search* (CBS) [19] and a **compilation-based** approach represented by reducing MAPF to propositional satisfiability (SAT) [4] in the MDD-SAT algorithm [29]. The SMT-CBS algorithm rephrases ideas of CBS in the terms of *satisfiability modulo theories* (SMT) [5] at the high-level. While at the low-level it uses the SAT encoding from MDD-SAT.

Unlike the original CBS that resolves conflicts between agents by branching the search, SMT-CBS refines the propositional model with a disjunctive constraint to resolve the conflict. SMT-CBS hence does not branch at the high-level but instead incrementally extends the propositional model that is consulted with the external SAT solver similarly as it has been done in MDD-SAT. In contrast to MDD-SAT where the propositional model is fully constructed in a single-shot, the propositional model is being built lazily in SMT-CBS as new conflicts appear.

The hypothesis behind the design of SMT-CBS is that in many cases we do not need to add all constraints to form the *complete propositional model* while still be able to obtain a conflict-free solution. Intuitively we expect that such cases where the *incomplete propositional model* will suffice are represented by sparsely occupied instances with large environments. The expected benefit in contrast to MDD-SAT is that incomplete model can be constructed and solved faster. On the other hand we expect that the superior performance of MDD-SAT in environments densely populated with agents will be preserved as SMT-CBS will quickly converge the model towards the complete one.

1.2 Towards DPLL(MAPF) / CDCL(MAPF)

SMT-CBS represents a milestone towards an optimal MAPF solver where the SAT solver and the high-level construction of the propositional model are fully integrated, an algorithm we denote DPLL(MAPF). DPLL(T) [16] commonly denotes an algorithm integrating the SAT solver [2] with a decision procedure for the conjunctive fragment of some first-order theory T. Similarly as in SMT-CBS the SAT solver decides what literals in a given formula should be set to $TRUE$ in order to satisfy the formula. Subsequently, the decision procedure for the conjunctive fragment checks if the suggested

truth value assignment is consistent with T. In our case, the theory is represented by movement rules of MAPF and the formula encodes a question if there is a solution of given MAPF of specified value of the objective.

DPLL stands here for the standard search-based SAT solving procedure (Davis-Putnam-Logemann–Loveland) [6] but in fact we use its more modern variants known as CDCL (Conflict-Driven Clause Learning) [21], hence precisely our algorithm should be denoted CDCL(MAPF) however we will keep the more common notation DPLL(MAPF) used in the literature.

The **organization** of the paper is as follows: We first introduce MAPF formally. Then the combination of CBS and MDD-SAT, the recent optimal MAPF algorithm SMT-CBS is recalled. On top of this we discuss the future perspective of DPLL(MAPF) in more details.

2 Multi-Agent Path Finding Formally

The *Multi-agent path finding* (MAPF) problem [22,18] consists of an undirected graph $G = (V, E)$ and a set of agents $A = \{a_1, a_2, ..., a_k\}$ such that $|A| \leq |V|$. Each agent is placed in a vertex so that at most one agent resides in each vertex. The placement of agents is denoted $\alpha : A \rightarrow V$. Next we are given initial configuration of agents α_0 and goal configuration α_+.

At each time step an agent can either *move* to an adjacent vertex or *wait* in its current vertex. The task is to find a sequence of move/wait actions for each agent a_i, moving it from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ such that agents do not *conflict*, i.e., do not occupy the same vertex at the same time nor cross the same edge in opposite directions simultaneously. The following definition formalizes the commonly used movement rule in MAPF. An example of MAPF instance is shown in Figure 1.

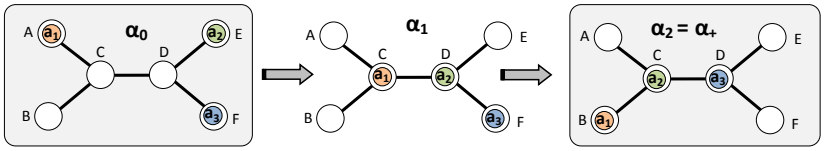

Fig. 1: A MAPF instance with three agents a_1, a_2, and a_3. A two-step solution is shown too.

Definition 1. Valid movement in MAPF. *Configuration α' results from α if and only if the following conditions hold:*

(i) $\alpha(a) = \alpha'(a)$ *or* $\{\alpha(a), \alpha'(a)\} \in E$ *for all* $a \in A$ *(agents wait or move along edges);*

(ii) for all $a \in A$ *it holds* $\alpha(a) \neq \alpha'(a) \Rightarrow \neg(\exists b \in A)(\alpha(b) = \alpha'(a) \wedge \alpha'(b) = \alpha(a))$ *(no two agents crosses an edge in opposite directions);*

(iii) and for all $a, a' \in A$ *it holds that* $a \neq a' \Rightarrow \alpha'(a) \neq \alpha'(a')$ *(no two agents share a vertex in the next configuration).*

Solving MAPF is to find a sequence of configurations $[\alpha_0, \alpha_1, ..., \alpha_\mu]$ such that α_{i+1} results using valid movements from α_i for $i = 1, 2, ..., \mu - 1$, and $\alpha_\mu = \alpha_+$. A *feasible solution* of a solvable MAPF instance can be found in polynomial time [32,11]; precisely the worst case time complexity of most practical algorithms for finding feasible solutions is $\mathcal{O}(|V|^3)$ [13,31].

We are often interested in optimal solutions. In case of the *makespan* [26] we just need to minimize μ in the aforementioned solution sequence. For introducing the *sum-of-costs* objective [8,23,20] we need more notation as follows:

Definition 2. Sum-of-costs objective *is the summation, over all k agents, of the number of time steps required to reach the goal vertex. Denoted ξ, where $\xi = \sum_{i=1}^{k} \xi(path(a_i))$ and $\xi(path(a_i))$ is an individual path cost of agent a_i connecting $\alpha_0(a_i)$ and $\alpha_+(a_i)$ calculated as the number of edge traversals and wait actions.* [1]

We note that finding a solution that is optimal (minimal) with respect to either the makespan or the sum-of-costs objective is NP-hard [17,25].

3 Unifying Search-based and Compilation-based Approaches

Before introducing SMT-CBS, a unification between the search-based and the compilation-based approach we will briefly discuss both approaches themselves.

3.1 Conflict-based Search

CBS is a representative of **search-based approach**. CBS uses the idea of resolving conflicts lazily; that is, a solution of MAPF instance is not searched against the complete set of movement constraints that forbids collisions between agents but with respect to initially empty set of collision forbidding constraints that gradually grows as new conflicts appear. The advantage of CBS is that it can find a valid solution before all constraints are added.

The high-level of CBS searches a *constraint tree* (CT) using a priority queue in breadth first manner. CT is a binary tree where each node N contains a set of collision avoidance constraints $N.constraints$ - a set of triples (a_i, v, t) forbidding occurrence of agent a_i in vertex v at time step t, a solution $N.paths$ - a set of k paths for individual agents, and the total cost $N.\xi$ of the current solution.

The low-level process in CBS associated with node N searches paths for individual agents with respect to set of constraints $N.constraints$. For a given agent a_i, this is a standard single source shortest path search from $\alpha_0(a_i)$ to $\alpha_+(a_i)$ that avoids a set of vertices $\{v \in V | (a_i, v, t) \in N.constraints\}$ whenever working at time step t. For details see [19].

CBS stores nodes of CT into priority queue OPEN sorted according to the ascending costs of solutions. At each step CBS takes node N with the lowest cost from OPEN and checks if $N.paths$ represent paths that are valid with respect to MAPF movements

[1] The notation $path(a_i)$ refers to path in the form of a sequence of vertices and edges connecting $\alpha_0(a_i)$ and $\alpha_+(a_i)$ while ξ assigns the cost to a given path.

Algorithm 1: CBS algorithm for MAPF solving

```
1  CBS (G = (V, E), A, α0, α+)
2      R.constraints ← ∅
3      R.paths ← {shortest path from α0(ai) to α+(ai)|i = 1, 2, ..., k}
4      R.ξ ← Σ_{i=1}^{k} ξ(N.paths(ai))
5      insert R into OPEN
6      while OPEN ≠ ∅ do
7          N ← min(OPEN)
8          remove-Min(OPEN)
9          collisions ← validate(N.paths)
10         if collisions = ∅ then
11             return N.paths
12         let (ai, aj, v, t) ∈ collisions
13         for each a ∈ {ai, aj} do
14             N'.constraints ← N.constraints ∪ {(a, v, t)}
15             N'.paths ← N.paths
16             update(a, N'.paths, N'.constraints)
17             N'.ξ ← Σ_{i=1}^{k} ξ(N'.paths(ai))
18             insert N' into OPEN
```

rules - that is, $N.paths$ are checked for collisions. If there is no collision, the algorithms returns valid MAPF solution $N.paths$. Otherwise the search branches by creating a new pair of nodes in CT - successors of N. Assume that a collision occurred between agents a_i and a_j in vertex v at time step t. This collision can be avoided if either agent a_i or agent a_j does not reside in v at timestep t. These two options correspond to new successor nodes of N - N_1 and N_2 that inherit the set of conflicts from N as follows: $N_1.conflicts = N.conflicts \cup \{(a_i, v, t)\}$ and $N_2.conflicts = N.conflicts \cup \{(a_j, v, t)\}$. $N_1.paths$ and $N_1.paths$ inherit paths from $N.paths$ except those for agents a_i and a_j respectively. Paths for a_i and a_j are recalculated with respect to extended sets of conflicts $N_1.conflicts$ and $N_2.conflicts$ respectively and new costs for both agents $N_1.\xi$ and $N_2.\xi$ are determined. After this, N_1 and N_2 are inserted into the priority queue OPEN.

The pseudo-code of CBS is listed as Algorithm 1. One of crucial steps occurs at line 16 where a new path for colliding agents a_i and a_j is constructed with respect to the extended set of conflicts. $N.paths(a)$ refers to path of agent a.

3.2 Compilation to Propositional Satisfiability

The major alternative to CBS is represented by **compilation** of MAPF to propositional satisfiability (SAT) [29,26]. The idea follows SAT-based planning [10] where the existence of a plan for a fixed number time steps is modeled as SAT. We similarly construct propositional formula $\mathcal{F}(\xi)$ such that it is satisfiable if and only if a solution of a given MAPF of sum-of-costs ξ exists. Moreover, the approach is constructive; that is, $\mathcal{F}(\xi)$ exactly reflects the MAPF instance and if satisfiable, solution of MAPF can be

reconstructed from satisfying assignment of the formula. We say $\mathcal{F}(\xi)$ to be a *complete propositional model* of MAPF.

Definition 3. (complete propositional model). *Propositional formula $\mathcal{F}(\xi)$ is a* complete propositional model *of MAPF Σ if the following condition holds:*

$$\mathcal{F}(\xi) \textit{ is satisfiable} \Leftrightarrow \Sigma \textit{ has a solution of sum-of-costs } \xi.$$

Being able to construct such formula $\mathcal{F}$ one can obtain optimal MAPF solution by checking satisfiability of $\mathcal{F}(\xi_0)$, $\mathcal{F}(\xi_0 + 1)$, $\mathcal{F}(\xi_0 + 2)$,... until the first satisfiable $\mathcal{F}(\xi)$ is met (ξ_0 is the lower bound for the sum-of-costs calculated as the sum of lengths of shortest paths). This is possible due to monotonicity of MAPF solvability with respect to increasing values of common cumulative objectives. Details of $\mathcal{F}$ are given in [29].

The advantage of the SAT-based approach is that state-of-the-art SAT solvers can be used for determining satisfiability of $\mathcal{F}(\xi)$ [3].

4 Combining SMT and CBS

A natural relaxation from the complete propositional model is an *incomplete propositional model* where instead of the equivalence between solving MAPF and the formula we require an implication only.

Definition 4. (incomplete propositional model). *Propositional formula $\mathcal{H}(\xi)$ is an* incomplete propositional model *of MAPF Σ if the following condition holds:*

$$\mathcal{H}(\xi) \textit{ is satisfiable} \Leftarrow \Sigma \textit{ has a solution of sum-of-costs } \xi.$$

A close look at CBS reveals that it operates similarly as problem solving in *satisfiability modulo theories* (SMT) [5]. SMT divides satisfiability problem in some complex theory T into an abstract propositional part that keeps the Boolean structure of the decision problem and a simplified decision procedure $DECIDE_T$ that decides fragment of T restricted on *conjunctive formulae*. A general T-formula Γ is transformed to a *propositional skeleton* by replacing atoms with propositional variables. The SAT solver then decides what variables should be assigned $TRUE$ in order to satisfy the skeleton - these variables tells what atoms hold in Γ. $DECIDE_T$ then checks if the conjunction of atoms assigned $TRUE$ is valid with respect to axioms of T. If so then satisfying assignment is returned. Otherwise a conflict from $DECIDE_T$ (often called a lemma) is reported back and the skeleton is extended with a constraint forbidding the conflict.

The above observation led us to the idea to rephrase CBS in terms of SMT. The abstract propositional part working with the skeleton will be taken from MDD-SAT provided that only constraints ensuring that assignments form valid paths interconnecting starting positions with goals will be preserved. Other constraints for collision avoidance will be omitted initially. This will result in an *incomplete propositional model*.

The paths validation procedure will act as $DECIDE_T$ and will report back the set of conflicts found in the current solution. Hence axioms of T will be represented by the movement rules of MAPF. We call the resulting algorithm SMT-CBS and it is shown in pseudo-code as Algorithm 2.

Algorithm 2: SMT-CBS algorithm for MAPF solving

```
1  SMT-CBS (Σ = (G = (V, E), A, α0, α+))
2      conflicts ← ∅
3      paths ← {path*(ai) a shortest path from α0(ai) to α+(ai)|i = 1, 2, ..., k}
4      ξ ← Σ_{i=1}^{k} ξ(paths(ai))
5      while TRUE do
6          (paths, conflicts) ← SMT-CBS-Fixed(conflicts, ξ, Σ)
7          if paths ≠ UNSAT then
8              return paths
9          ξ ← ξ + 1
10 SMT-CBS-Fixed(conflicts, ξ, Σ)
11     H(ξ) ← encode-Basic(conflicts, ξ, Σ)
12     while TRUE do
13         assignment ← consult-SAT-Solver(H(ξ))
14         if assignment ≠ UNSAT then
15             paths ← extract-Solution(assignment)
16             collisions ← validate(paths)
17             if collisions = ∅ then
18                 return (paths, conflicts)
19             for each (ai, aj, v, t) ∈ collisions do
20                 H(ξ) ← H(ξ) ∪ {¬X_v^t(ai) ∨ ¬X_v^t(aj)}
21                 conflicts ← conflicts ∪ {[(ai, v, t), (aj, v, t)]}
22         return (UNSAT,conflicts)
```

The algorithm is divided into two procedures: SMT-CBS representing the main loop and SMT-CBS-Fixed solving the input MAPF for fixed cost ξ. The major difference from the standard CBS is that there is no branching at the high-level. The high-level SMT-CBS roughly correspond to the main loop of MDD-SAT. The set of conflicts is iteratively collected during the entire execution of the algorithm. Procedure *encode* from MDD-SAT is replaced with *encode-Basic* that produces encoding that ignores specific movement rules (collisions between agents) but in contrast to *encode* it encodes collected conflicts into $\mathcal{H}(\xi)$.

The conflict resolution in the standard CBS implemented as high-level branching is here represented by refinement of $\mathcal{H}(\xi)$ with disjunction (line 20). The presented SMT-CBS can eventually build the same formula as MDD-SAT but this is done lazily in SMT-CBS.

5 From SMT-CBS to DPLL(MAPF) and Beyond

Although the performed experimental evaluation presented in [28] shows that SMT-CBS significantly outperforms both the CBS algorithm and MDD-SAT we conjecture that there is still room for improving the idea behind SMT-CBS. SMT-CBS as shown

in Algorithm 2 implements very basic variant of SMT-based problem solving. More advanced SMT-based algorithms for deciding formulae in first order theories integrates the SAT solver and $DECIDE_T$ more tightly. $DECIDE_T$ is invoked not only for the fully assigned formulae (line 13 produces a full assignment) but also for partial assignments. Such algorithms are usually denoted DPLL(T) [16].

Whenever the SAT-solving search loop in DPLL (or CDCL) assigns new propositional variable, $DECIDE_T$ is invoked to check if the extended partial assignment is consistent with respect to T (or MAPF in our case). If so the main loop continues with assigning the next variable. Otherwise $DECIDE_T$ return a lemma forbidding the current assignment, translated in MAPF terms this corresponds to a case when $DECIDE_{MAPF}$ discovers a conflict and returns the conflict elimination constraint.

$DECIDE_T$ ($DECIDE_{MAPF}$) can do much more in the consistent case. The procedure can derive new assignments that is to perform a form of MAPF constraint propagation. Such propagation could work in tandem with the standard *unit propagation* [7] integrated in the CDCL-based SAT solvers. Moreover DPLL(MAPF) algorithm can be further parametrized with variable and value selection heuristics that can take into account axioms of the MAPF theory.

5.1 Implications for MAPF with Continuous Time and Geometric Agents

Recently generalizations of MAPF considering continuous time and geometric agents have appeared [1]. The continuous variant denoted MAPF$^{\mathcal{R}}$ assigns each vertex of the underlying graph from the standard MAPF coordinates in the Euclidean space. Agents are geometric objects such as circles, spheres, polygons etc. and can move along straight lines connecting vertices of the underlying graph (agents cannot move outside these predefined lines). For simplicity agents are assumed to have constant speeds but the algorithmic concepts generalize for non-constant models too.

Collisions between agents are defined as any overlap between their bodies. The task is to find a collision free temporal plan. An adaptation of CBS algorithm denoted CBS$^{\mathcal{R}}$ for the continuous variant has been proposed [1]. We rephrased CBS$^{\mathcal{R}}$ in terms of SMT similarly as it has been done with CBS. The resulting algorithm SMT-CBS$^{\mathcal{R}}$ [27] differs from SMT-CBS mostly in the aspect of decision variable generation. In case of the standard MAPF, decision variables can be determined statically in advance but this is not applicable in the continuous version since the continuity has potential to spawn infinitely many decisions. Decision variables need to be generated dynamically in response to discoveries of new conflicts in MAPF$^{\mathcal{R}}$.

Naturally the future step in the development of SMT-CBS$^{\mathcal{R}}$ is DPLL(MAPF$^{\mathcal{R}}$) where also partial assignments will be checked for consistency.

6 Discussion and Conclusion

We recalled the new MAPF solving method called SMT-CBS that combines search-based CBS and SAT-based MDD-SAT through concepts from satisfiability modulo theories (SMT). Although SMT-CBS represent state-of-the-art in optimal MAPF solving as shown in [28] we identified certain deficiencies of the algorithm. Namely the fact

that it checks for consistency with respect to MAPF rules only fully assigned propositional encodings. We therefore theoretically suggest DPLL(MAPF), a new algorithm that will also check for consistency partial assignments. We further suggest future work in which DPLL(MAPF) will be generalized for MAPF with continuous time and geometric agents, an analogous next step beyond the existing SMT-CBS$^{\mathcal{R}}$ algorithm [27].

Acknowledgements

This research has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S.

References

1. Andreychuk, A., Yakovlev, K., Atzmon, D., Stern, R.: Multi-agent pathfinding (MAPF) with continuous time. CoRR **abs/1901.05506** (2019), http://arxiv.org/abs/1901.05506
2. Audemard, G., Lagniez, J., Simon, L.: Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: SAT. pp. 309–317 (2013)
3. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: IJCAI. pp. 399–404 (2009)
4. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
5. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with SAT modulo theories. Constraints **17**(3), 273–303 (2012)
6. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962). https://doi.org/10.1145/368273.368557, https://doi.org/10.1145/368273.368557
7. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional horn formulae. J. Log. Program. **1**(3), 267–284 (1984). https://doi.org/10.1016/0743-1066(84)90014-1, https://doi.org/10.1016/0743-1066(84)90014-1
8. Dresner, K., Stone, P.: A multiagent approach to autonomous intersection management. JAIR **31**, 591–656 (2008)
9. Hönig, W., Kumar, T.K.S., Cohen, L., Ma, H., Xu, H., Ayanian, N., Koenig, S.: Summary: Multi-agent path finding with kinematic constraints. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017. pp. 4869–4873 (2017)
10. Kautz, H.A., Selman, B.: Unifying sat-based and graph-based planning. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 1999. pp. 318–325 (1999)
11. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: FOCS, 1984. pp. 241–250 (1984)
12. Li, J., Surynek, P., Felner, A., Ma, H., Koenig, S.: Multi-agent path finding for large agents. In: AAAI. AAAI Press (2019)
13. Luna, R., Bekris, K.E.: Push and swap: Fast cooperative path-finding with completeness guarantees. In: IJCAI. pp. 294–300 (2011)
14. Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hönig, W., Kumar, T.K.S., Uras, T., Xu, H., Tovey, C.A., Sharon, G.: Overview: Generalizations of multi-agent path finding to real-world scenarios. CoRR **abs/1702.05515** (2017), http://arxiv.org/abs/1702.05515

15. Ma, H., Wagner, G., Felner, A., Li, J., Kumar, T.K.S., Koenig, S.: Multi-agent path finding with deadlines. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 417–423 (2018)
16. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T). J. ACM **53**(6), 937–977 (2006)
17. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the N x N extension of the 15-puzzle is intractable. In: AAAI. pp. 168–172 (1986)
18. Ryan, M.R.K.: Exploiting subgraph structure in multi-robot path planning. J. Artif. Intell. Res. (JAIR) **31**, 497–542 (2008)
19. Sharon, G., Stern, R., Felner, A., Sturtevant, N.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
20. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. **195**, 470–495 (2013)
21. Silva, J.P.M., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Computers **48**(5), 506–521 (1999). https://doi.org/10.1109/12.769433, https://doi.org/10.1109/12.769433
22. Silver, D.: Cooperative pathfinding. In: AIIDE. pp. 117–122 (2005)
23. Standley, T.: Finding optimal solutions to cooperative pathfinding problems. In: AAAI. pp. 173–178 (2010)
24. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: ICRA 2009. pp. 3613–3619 (2009)
25. Surynek, P.: An optimization variant of multi-robot path planning is intractable. In: AAAI 2010. AAAI Press (2010)
26. Surynek, P.: Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. Ann. Math. Artif. Intell. **81**(3-4), 329–375 (2017)
27. Surynek, P.: Multi-agent path finding with continuous time viewed through satisfiability modulo theories (SMT). CoRR **abs/1903.09820** (2019), http://arxiv.org/abs/1903.09820
28. Surynek, P.: Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019. in press (2019)
29. Surynek, P., Felner, A., Stern, R., Boyarski, E.: Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: ECAI. pp. 810–818 (2016)
30. Wang, K., Botea, A.: MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. JAIR **42**, 55–90 (2011)
31. de Wilde, B., ter Mors, A., Witteveen, C.: Push and rotate: a complete multi-agent pathfinding algorithm. JAIR **51**, 443–492 (2014)
32. Wilson, R.M.: Graph puzzles, homotopy, and the alternating group. Journal of Combinatorial Theory, Series B **16**(1), 86 – 96 (1974)