Problem Compilation for Multi-Agent Path Finding: a Survey

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

Abstract

Multi-agent path finding (MAPF) attracts considerable attention in the artificial intelligence community. The task in the standard MAPF is to find discrete paths through which agents can navigate from their starting positions to individual goal positions. The combination of two additional requirements makes the problem computationally challenging: agents must not collide with each other and the paths must be optimal with respect to some objective. Two major approaches to optimal MAPF solving include dedicated search-based methods, and compilation-based methods that reduce a MAPF instance to an instance in a different formalism, for which an efficient solver exists. In this survey, we summarize major compilation-based solvers for MAPF using CSP, SAT, and MILP formalisms. We explain the core ideas of the solvers in a simplified and unified way while preserving the merit making them more accessible for a wider audience.

1 Introduction

Compilation is one of the most prominent technique used in computing. In the context of problem solving in artificial intelligence, compilation is represented by a reduction of an input instance from its source formalism to a different usually well established formalism for which an efficient solver exists. The reduction is often fast so that obtaining the instance in the target formalism and interpreting the solution back to the source formalism consumes only a small portion of the total time while the most of time is spent by the solver. The key idea behind using compilation in problem solving is that the solving process benefits from the advancements in the solver for the target formalism, often accumulated over decades.

The compilation-based solving approach has been applied successfully in solving combinatorial problems like *planning* [Ghallab *et al.*, 2004], *verification* [Bradley and Manna, 2007], or *scheduling* [Blazewicz *et al.*, 2004] where the target formalism is often represented by *Boolean satisfiability* (SAT) [Biere *et al.*, 2021], *mixed integer linear programming* (MILP) [Jünger *et al.*, 2010; Rader, 2010], *answer set programming* (ASP) [Lifschitz, 2019] or *constraint satisfaction* (CSP) [Dechter, 2003], all being at the NP-hard level w.r.t. the computational complexity perspective. Significant advancements have been achieved in compilation-based approaches in specific domains, namely in *multi-agent path finding* (MAPF) [Silver, 2005; Ryan, 2008; Standley, 2010] that we are focusing on in this survey.

The standard MAPF is the problem of finding collision-free paths for a set of agents from their starting positions to individual goal positions (Figure 1). Agents move in a discrete environment which is modeled as an undirected graph $G = (V, E)$, where vertices represent positions and edges the possibility of moving between the positions. Agents are often denoted $A = \{a_1, a_2, ..., a_k\}, k \leq |V|$, placed in vertices of the graph, moving instantaneously between vertices provided that there is always at most one agent in a vertex and no two agents traverse an edge in opposite directions [1].

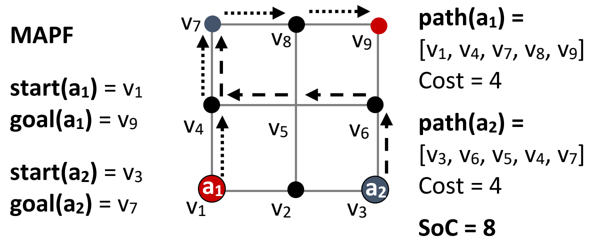

Figure 1: A multi-agent path finding (MAPF) instance with two agents a_1 and a_2 and its solution with the sum of costs (SoC) of 8 (paths for individual agents are shown to the right).

There are numerous applications of MAPF in warehouse logistics [Li *et al.*, 2020b], traffic optimization [Mohanty *et al.*, 2020], multi-robot systems [Preiss *et al.*, 2017], and computer games [Snape *et al.*, 2012] to name few.

Relatively simple formulation of MAPF is an important factor that made it accessible for various solving methods including compilation-based approaches. The simultaneous existence of diverse methods for MAPF has fostered mutual cross-fertilization and deeper understanding. Cur-

[1]Different movement rules exist such as permitting the move into a vacant vertex only. There is also large body of works dealing with related problems like *pebble motion* in graphs [Kornhauser *et al.*, 1984], *token swapping*, and *token permutation* in graphs [Bonnet *et al.*, 2018] using similar movement primitives.

rent state-of-the-art compilation-based solvers for MAPF go even beyond the standard single shot **reduction-solving-interpretation** loop coined for classical planning by the SATPlan algorithm [Kautz and Selman, 1992] where SAT has been used as the target formalism. Intensive cross-fertilization between the dedicated search-based methods for MAPF, that solve the problem directly, and compilation techniques led to numerous improvements in encodings of MAPF in target formalisms but also in how the solver is used, treating it less like a black-box.

The effort culminated recently in a combination of compilation and **lazy** collision elimination introduced in Conflict-based search algorithm (CBS) [Sharon *et al.*, 2015] resulting in approaches that construct the target encoding lazily in close cooperation with the solver. The solver in these lazy schemes suggests solutions for incomplete encodings of the input instance that do not specify it fully. After checking the interpreted solution against the original specification, that is if it is a valid MAPF solution (agents do not collide, do not jump, do not disappear, and do not appear spontaneously, etc.), the high-level part of the MAPF solver suggests a refinement of the encoding and the process is repeated. This lazy scheme has been implemented using SAT [Surynek, 2019], CSP [Gange *et al.*, 2019], and MILP [Lam *et al.*, 2019] as target formalisms while important differences exist between these approaches.

We summarize how the research in compilation for MAPF came to this point and describe major compilation-based approaches in a simplified and unified way so that core ideas are easily identifiable and accessible for a wider audience.

This survey aims to complement existing surveys on the MAPF topic, namely [Felner *et al.*, 2017] that focuses on MAPF from the point of view of heuristic search, and [Stern, 2019] that focuses more on the MAPF problem itself and its generalizations.

2 An Overview of Target Formalisms

We will first give a brief overview of popular formalisms that are often used as the target of the compilation process.

Constraint satisfaction problem (CSP) is a tuple (X, D, C) where X is a finite set of variables, D is a finite domain of values that can be assigned to variables, and C is a set of constraints in the form of arbitrary relations over the variables, that is a constraint is a subset of the Cartesian product $D \times D \times ...D$ (the arity corresponds to how many variables participate in the constraint). A solution of CSP is an evaluation of all variables by values from D such that all constraints from C are satisfied.

In MAPF, we can express positions of agents a_i at time step t using a variable $X_i^t \in V$ (indicating that the domain is V) [Ryan, 2010]. Then constraints expressing the MAPF rules such as: $X_i^t \neq X_i^{t+1} \rightarrow \{X_i^t, X_i^{t+1}\} \in E$, agents use edges (do not skip); and $Distinct(X_1^t, X_2^t, ..., X_k^t)$, agents do not collide in vertices, etc. are added.

The CSP paradigm provides powerful complete search algorithms coupled with constraint propagation and domain filtering techniques that prune the search space [Dechter, 2003]. One of the most significant advantages of CSP in contrast to other paradigms are *global constraints*, dedicated filtering algorithms often based on matchings for special relations like $Distinct$ (variables should take different values) or more general cardinality constraints, that enforce consistency across large sets of variables [Régin, 1994]. Using global constraints it is easy to detect that $Distinct(X_1^t, X_2^t, X_3^t)$ cannot be satisfied when $D = \{v_1, v_2\}$ but it is hard to see it looking on the individual inequalities $X_i^t \neq X_j^t$ separately.

A *linear program* (LP) is a finite list of linear inequalities plus linear objective, the task is to minimize the objective such that the inequalities hold. Geometrically, the inequalities define a polytope and the objective function defines a gradient so the task is to find some boundary point of the polytope that minimizes the gradient. Formally, the task is to minimize $c^\top x$ subject to $Px \leq b$, $x \geq 0$, where x is a real vector representing the decision variables, P is a matrix of real coefficients of linear inequalities, and b is another real-valued vector. An optimal solution to linear program can be found in polynomial time however LP is not suitable for discrete decision problems like MAPF due to possible fractional assignment of decision variables.

That is why often some or all decision variables are declared to be integers turning LP into an *mixed integer linear program* (MILP) or *integer linear program* (ILP) respectively. Introducing integer decision variables improves the expressive power for discrete problems but on the contrary in makes the problem NP-hard. The important advantage of MIP is natively supported arithmetic hence optimizing with respect to various cumulative objectives used in MAPF can be modeled easily when MILP is used as the target formalism.

Boolean satisfiability (SAT) problem consists in deciding whether there exists a truth-value assignment of variables that satisfies a given Boolean formula. The formula is often specified using the *conjunctive normal form* (CNF), it is a conjunction of clauses where each clause is a disjunction of literals, where a literal is either a variable or a negation of a variable. SAT is often considered to be a canonical NP-complete problem [Cook, 1971]. A formula in CNF can be regarded through the CSP formalism so that clauses represent individual constraints. Such view actually catalyses mutual cross fertilization between solving algorithms for SAT and CSP. Modern SAT solvers are based on the *conflict-driven clause learning* algorithm [Eén and Sörensson, 2003; Audemard and Simon, 2018] that implements constraint propagation and back-jumping techniques known from CSP.

The significant challenge when SAT is used as the target formalism is bridging the original representation of the problem and the yes/no environment of SAT. In the context of MAPF, it is often difficult to handle the objectives that use arithmetic.

The disadvantage in the lack of expressiveness, caused by the fact that Boolean formulae are often far from being human readable, is well balanced by the efficiency of SAT solvers enabled by numerous techniques such as learning, restarts, etc.

The difficulty of encoding problems in the SAT formalism at the low-level led to the development of various high-level languages that enable encoding of problems more intuitively.

One such notable higher level approach used in MAPF is *answer set programming* (ASP) [Erdem *et al.*, 2013]. The problem in ASP is described as a logic program that is automatically translated to a Boolean formula. The formula is solved by the SAT solver and the outcome is propagated back at the logic program level where it is translated to the human readable form.

We show a part of the ASP program for MAPF to illustrate how the ASP paradigm works, see [Erdem *et al.*, 2013] for the full program. We have atoms $path(i,t,v)$ whose interpretation is that agent $a_i \in A$ is at vertex v at time step t. These atoms are defined recursively using the rules of a *logic program*:

$$path(i,0,v) \leftarrow start(i,v) \tag{1}$$

$$1\{path(i,t+1,u), path(i,t+1,v) : edge(u,v)\}1 \leftarrow path(i,t,v) \tag{2}$$

The program says that agents' paths start at starting positions 1 and then either traverse an edge or wait in a vertex 2. Ones around the rule specify the number of atoms selected for the answer of the rule, the lower and the upper bounds are specified. Here one of the two instances of *path* is selected.

ASP allows for specifying constraints which are the rules without the head, eliminating answers that satisfy the body. Constraints can be used to eliminate conflicts in vertices for MAPF as follows:

$$\leftarrow path(i,t,v), path(j,t,v) \tag{3}$$

The ASP program is solved via reduction to SAT and by calling the SAT solver. The important advantage of ASP in contrast to plain SAT-based approach is that ASP provides high level language to describe problems and one does not need to construct the target Boolean formula.

3 From Classical Planning to MAPF

One of the pioneering works that used compilation for problem solving is the SATPlan algorithm reducing the classical planning problem to Boolean satisfiability [Kautz and Selman, 1992]. Classical planning is the task of finding a sequence of actions that transforms a given initial state of some abstract world to a desired goal state. States are described as finite sets of atoms. Actions can add and remove atoms in states provided their preconditions are satisfied.

SATPlan is the important milestone in non-trivial compilation schemes since there is no one-to-one correspondence between the planning instance and the target SAT instance. Instead, a bounded decision variant of the input problem is introduced in which the number of time steps is bounded. The SAT solver is consulted with a sequence of yes/no questions on the bounded variant, whether there exists a plan consisting of a specified number of time steps t_{max}.

3.1 Time Expansion

Another important concept significantly used by SATPlan is a *time expansion* that enables representing states at individual time steps inside the target formalism. Decision variables representing atoms are indexed with time steps up to t_{max}; the variable indexed with $0 < t \leq t_{max}$ is set to $TRUE$ if and only if the corresponding atom holds at time step t.

Algorithm 1: Framework of the SATPlan algorithm.

```
1 SATPlan(planning problem $\mathcal{P}$)
2     $t_{max} \leftarrow 1$
3     while $TRUE$ do
4         $F \leftarrow$ encode-SAT($\mathcal{P}$,$t_{max}$)
5         $assignment \leftarrow$ consult-SAT-Solver($F$)
6         if $assignment \neq UNSAT$ then
7             $plan \leftarrow$ interpret($\mathcal{P}$,$assignmnet$)
8             return $plan$
9         $t_{max} \leftarrow t_{max} + 1$
```

In addition to state variables, there are action variables for each time step t indicating whether given action takes place at t. Constraints ensure that if an action variable at t is set to $TRUE$ then precondition atoms must hold in state variables at time step t and effects must hold in state variables at time step $t+1$.

Consider an action that moves agent a from vertex v_1 to vertex v_2: $precond(move) = \{at(a,v_1)\}$, $effect^+(move) = \{at(a,v_2)\}$, $effect^-(move) = \{at(a,v_1)\}$ for which we need to add the following constraints $\forall t \in \{1,...,t_{max}-1\}$:

$$move^t \rightarrow at(a,v_1)^t \wedge at(a,v_2)^{t+1} \wedge \neg at(a,v_1)^{t+1} \tag{4}$$

Such encoding can be easily constructed as well as read off in the interpretation phase. The simplified pseudo-code of SATPlan is shown as Algorithm 1.

3.2 Encodings of MAPF

The early compilation-based approaches for MAPF were rooted in the CSP formalism [Ryan, 2010]. Even early SAT-based approaches used CSP-like variables for expressing the positions of agents with domains consisting of all vertices [Surynek, 2012]. These decision variables were represented as a vector consisting of the logarithmic number of bits (Boolean variables) with respect to the size of the domain *log-space encoding* [Prestwich, 2003]. The disadvantage of log-space encodings in the context of MAPF seems to be weak Boolean constraint propagation and the fact that *reachability analysis* does not prune out variables from the encoding (removing values from decision variables corresponds to forbidding certain combination of settings of bit vectors but the Boolean variables representing the bits remain).

This is why all following SAT-based compilations of MAPF used *direct encoding* [Walsh, 2000] where there is a single Boolean variable $\mathcal{X}^t_{i,v}$ for each agent $a_i \in A$, each vertex $v \in V$, and relevant time step t. Despite these decision variables are somewhat redundant they provide a good support for constraint propagation and the reachability analysis can completely remove them from the formula.

4 Milestones in MAPF Compilation

Although modern SAT solvers are powerful they alone are not sufficient and the way how the MAPF instance is presented to the SAT solver is equally important. From my personal perspective, there are two milestones that made the compilation for MAPF a competitive alternative to search-based methods: **(1)** reachability analysis and **(2)** lazy refinements.

4.1 Reachability Analysis

The disadvantage of SAT-based compilation for planning is often the big size of the target formula since variables for all ground atoms and all ground actions must be included. Reachability analysis via *planning graphs* [Blum and Furst, 1997] mitigates this difficulty. The planning graph is a structure representing time expansion of the set of possibly valid atoms across discrete time steps. Starting with the initial state at the first time step, atoms for the next time step are generated first by considering all applicable actions in the previous time step and by adding their positive effects together with atoms from the previous time step. To reduce the growth of the set of atoms, *mutual exclusions* (mutex) for pairs of atoms and actions are introduced. Whenever two atoms are mutex they cannot satisfy the preconditions of an action and the action is hence pruned out.

Reachability analysis analogous to planning graphs in the domain of MAPF is done via *multi-valued decision diagrams* (MDDs) [Andersen *et al.*, 2007]. MDDs were surprisingly not used in MAPF compilation for the first time but they were originally applied in the Increasing-cost Tree Search algorithm (ICTS) [Sharon *et al.*, 2013]. The idea of MDD is to represent all paths of the specified cost.

The compilation introduced in the MDD-SAT algorithm [Surynek *et al.*, 2016] is analogous to the more advanced variant of SATPlan with planning graphs [Kautz and Selman, 1999]. Both algorithms use *eager compilation*, that is, the resulting Boolean formula is constructed in one shot and the SAT solver is regarded as a black-box. The incremental scheme to find a step-optimal plan is also similar to SATPlan.

The role of planning graphs for reachability analysis is substituted by *multi-valued decision diagrams* (MDDs). MDD$_i$ is constructed for each agent $a_i \in A$ and contains a copy of the underlying graph G for every relevant time step where nodes that are not reachable because they are too far from the starting vertex or from the goal vertex are removed. Directed edges in MDDs represent move and wait actions, that is, they interconnect nodes at consecutive time steps in MDD whose corresponding vertices are either identical or connected in G. A directed path in MDD$_i$ corresponds to a plan for agent a_i.

Decision Boolean variables $\mathcal{X}_{i,v}^t$ and $\mathcal{E}_{i,u,v}^t$ are introduced for each node (v,t) and edge $[(u,t);(v,t+1)]$ in MDD$_i$ respectively. The variables are $TRUE$ iff the agent uses the corresponding vertex or edge at given time step.

To encode MAPF rules, constraints are introduced over these variables. We list here few examples only, for the complete list of constraints see [Surynek *et al.*, 2016]. Collisions in vertices can be ruled out by the following constraint for every $v \in V$ and time step t:

$$\sum_{a_i \in A \mid (v,t) \in MDD_i} \mathcal{X}_{i,v}^t \leq 1 \tag{5}$$

The following constraints ensure that directed paths are taken in MDDs. If agent a_i appears in $u \in V$ at time step t then it must leave through exactly one edge connected to u:

$$\mathcal{X}_{i,u}^t \rightarrow \bigvee_{[(u,t);(v,t+1)] \in MDD_i} \mathcal{E}_{i,u,v}^t \tag{6}$$

Algorithm 2: SMT-CBS: MAPF via SAT.

```
1  SMT-CBS(MAPF $\mathcal{M}$)
2      $SoC \leftarrow$ lower-Bound($\mathcal{M}$)
3      $conflicts \leftarrow \emptyset$
4      while $TRUE$ do
5          $paths \leftarrow$ Bounded-SMT-CBS($\mathcal{M}$, $SoC$)
6          if $paths \neq UNSAT$ then
7              return $paths$
8          $SoC \leftarrow SoC + 1$
9  Bounded-SMT-CBS($\mathcal{M}$,$SoC$)
10     $F \leftarrow$ encode-SAT($\mathcal{M}$,$SoC$,$conflicts$)
11     while $TRUE$ do
12         $assignment \leftarrow$ consult-SAT-Solver($F$)
13         if $assignment \neq UNSAT$ then
14             $(paths, conflicts') \leftarrow$ check($\mathcal{M}$,$assignment$)
15             if $conflicts' = \emptyset$ then
16                 return $paths$
17             for each $(a_i, a_j, v, t) \in conflicts'$ do
18                 $F \leftarrow F \cup (\neg\mathcal{X}_{i,v}^t \vee \neg\mathcal{X}_{i,v}^t)$
19             $conflicts \leftarrow conflicts \cup conflicts'$
20         return $UNSAT$
```

$$\sum_{(v,t+1) \mid [(u,t);(v,t+1)] \in MDD_i} \mathcal{E}_{i,u,v}^t \leq 1 \tag{7}$$

4.2 Lazy Refinements

Conflict-based search (CBS) [Sharon *et al.*, 2015] is currently the most popular approach for MAPF. It is due to its simple and elegant idea which enables to implement the algorithm relatively easily, its good performance, and openness to various improvements via using heuristics.

From the compilation perspective, the CBS algorithm should be understood as a lazy method that tries to solve an underspecified problem and relies on being lucky to find a correct solution even using this incomplete specification. The completeness of the lazy approach is ensured by incremental additions of refinements that eliminate counterexamples obtained from unlucky solving. Similar concepts of incomplete problem specification and refinement can be found e.g. in logic-based Benders's decomposition [Ciré *et al.*, 2013] or in Counter Example Abstraction Refinement (GEGAR) [Clarke *et al.*, 2000; Clarke, 2003].

The refinement in the CBS algorithm is carried out by a branching scheme. If the candidate solution is incorrect in terms of MAPF rules, that is, if it is a counterexample, then the search branches for each possible refinement of discovered MAPF rule violation and the refinement is added to the problem specification in each branch. Concretely, the MAPF rule violations are conflicts of pairs of agents such as collision of $a_i \in A$ and $a_j \in A$ in v at time step t and the refinements are conflict avoidance constraints for single agents in the form that $a_i \in A$ should avoid v at time step t (for a_j analogously).

While in CBS, the branching scheme must be explicitly implemented, in the compilation-based approach, we can just eliminate MAPF rule violation by adding a new constraint into the problem specification and leave branching to the solver for the target formalism. The CBS inspired SAT-based compilation algorithm SMT-CBS is shown as Algorithm 2. In SMT-CBS, a conflict can be eliminated by adding the following disjunctive constraint (line 18):

$$\neg \mathcal{X}_{i,v}^{t} \vee \neg \mathcal{X}_{i,v}^{t} \tag{8}$$

In this way, the encoding of MAPF (or any other problem) is built dynamically and eventually may end up by the complete specification of the problem as done in MDD-SAT (or in SATPlan in the context of classical planning).

This approach is commonly known as *lazy encoding* and is often used in *satisfiability modulo theories* (SMT) [Katz *et al.*, 2016] as well as in LP, ILP, and MILP where it corresponds to the *row generation* technique [Muter *et al.*, 2013]. In SMT, we are interested in decision procedures for some complex logic theory T, that is decomposed into the Boolean part given to the SAT solver and decision procedure for the conjunctive fragment of T, denoted $DECIDE_T$. The SAT solver and $DECIDE_T$ cooperate in solving logic formula in T. The SAT solver chooses literals to be set $TRUE$ and $DECIDE_T$ checks if the chosen combination of literals is feasible and suggests refinements. In this sense, CBS is analogous to this approach where the role of $DECIDE_T$ is represented by conflict checking procedure (line 14).

As shown by experiments a solution or a proof of that it does not exists is often found for incomplete specification (that is, well before all constraints are added to the encoding). Surprisingly intuitive explanation why this is possible comes from the geometry of linear programming. The finite set of inequalities define a polytope of feasible solutions as an intersection of half-spaces, the boundary of the polytope is made by planes. An optimal feasible solution is an element of some of the planes defining the boundary but not an element of all of them (in other words, some inequalities are satisfied because the optimal solution is deep inside their half-space). Hence, to specify the optimal solution, one does not need all the constraints. Similarly if there is no solution, the polytope is empty. Again this may happen by intersecting only some of the half-spaces.

5 Beyond Simple Time Expansion

Modern MAPF compilation techniques to CSP and MILP do not rely on explicit time expansion. Instead, the target encoding or the parts of it are generated on demand.

5.1 Lazy Conflict-based Search

Lazy-CBS [Gange *et al.*, 2019] uses CSP as a target formalism but in contrast to SMT-CBS it uses **variables** to establish interaction between the search spaces of individual agents rather than disjunctive clauses as done in SMT-CBS. From the LP perspective, this approach is analogous to *column generation* [Desaulniers *et al.*, 2005]. A simplified pseudo-code of Lazy-CBS is shown as Algorithm 3.

Algorithm 3: Lazy CBS: MAPF via CSP.

```
1  Lazy-CBS(MAPF $\mathcal{M}$)
2      $costs = (c_1, c_2, ..., c_n) \leftarrow$ lower-Bounds($\mathcal{M}$)
3      $variables \leftarrow \emptyset$
4      while $TRUE$ do
5          $paths \leftarrow$ Bounded-LCBS($\mathcal{M}$, $costs$)
6          if $paths \neq FAIL$ then
7              return $paths$
8          $(c_1, c_2, ..., c_n) \leftarrow$ update-Bounds($costs$, $variables$)
9  Bounded-LCBS($\mathcal{M}$,$costs$)
10     $(X, D, C) \leftarrow$ encode-CSP($\mathcal{M}$,$costs$,$variables$)
11     while $TRUE$ do
12         $evaluation \leftarrow$ consult-CSP-Solver($(X, D, C)$)
13         if $evaluation \neq FAIL$ then
14             $(paths, conflicts) \leftarrow$ check($\mathcal{M}$,$evaluation$)
15             if $conflicts = \emptyset$ then
16                 return $paths$
17             for each $(a_i, a_j, v, t) \in conflicts$ do
18                 $X \leftarrow X \cup \{p_{v,t} \in A\}$
19                 $variables \leftarrow variables \cup \{p_{v,t}\}$
20         return $FAIL$
```

Lazy-CBS starts with finding individual shortest paths which are found by the CSP solver (line 2). Then similarly as in SMT-CBS, the paths are validated with respect to potential collisions (line 14). If there is no collision, then the path represent valid optimal solution of MAPF. Otherwise, there is a collision, say between agents a_i and a_j in vertex v at time step t. The algorithm introduces a fresh CSP variable $p_{v,t} \in A$ plus corresponding constraints indicating which agent is permitted to occupy v at time step t (line 18-19). Clearly, the need to evaluate the $p_{v,t}$ variable with a single value effectively eliminates the collision being found (the collision would imply evaluation of the variable with two values). After introducing all variables for eliminating the conflicts, the CSP solver is consulted to find new paths w.r.t. $p_{v,t}$ variables.

In addition, to the conflict elimination mechanism, the Lazy-CBS algorithm introduces cost bounds $c_1, c_2, ..., c_n$ on individual agents rather than having the cumulative cost bound. The individual cost bounds are updated similarly as done in the ICTS algorithm, that is, if the CSP solver concludes that no solution can be found for say agents a_i and a_j to simultaneously achieve their goals within costs c_i and c_j respectively, the costs bounds are increased by 1 (line 8).

It is important to note, that the state of the CSP solver can be kept between its invocations (line 12). Hence if a new $p_{v,t}$ variable does not affect already existing agent's path, then the path is kept.

5.2 BCP: Branch Cut and Price

MILP as the target formalism allows for reasoning about both integer and real-valued decision variables which opens opportunities to use decision variables with completely different meaning than in CSP and SAT. Bounding linear objectives is also easier in MILP than in SAT.

The model suggested in the Branch Cut and Price algorithm (BCP) [Lam *et al.*, 2019], simplified pseudo-code shown as Algorithm 4, considers a large but finite pool of paths $\Pi(a_i)$ for each agent $a_i \in A$ connecting its start and goal vertex (line 2). Decision variables $\lambda_{i,\pi} \in [0, 1]$ determine the proportion of path $\pi \in \Pi(a_i)$ being selected by the agent. Constraints ensure that agents use at least one path and the overall cost of path is minimized resulting in the following initial linear program LP:

$$min \sum_{a_i \in A} \sum_{\pi \in \Pi(a_i)} \lambda_{i,\pi} cost(\pi) \tag{9}$$

$$\sum_{\pi \in \Pi(a_i)} \lambda_{i,\pi} \geq 1 \quad \forall a_i \in A \tag{10}$$

$$\lambda_{i,\pi} \geq 0 \quad \forall a_i \in A, \forall \pi \in \Pi(a_i) \tag{11}$$

The program represents an incomplete encoding of the input instance as collisions between the agents are not forbidden initially. Collision resolution is done lazily similarly as in SMT-CBS and Lazy CBS. Once a vertex collision in $v \in V$ is detected in current (fractional) solution $\{\lambda_{i,\pi}\}$, that is the following inequality holds for v:

$$\sum_{a_i \in A} \sum_{\pi \in \Pi(a_i)} s_v^\pi \lambda_{i,\pi} > 1, \tag{12}$$

where $s_v^\pi \in \{0, 1\}$ indicates selection of vertex v by path π (1 means the vertex is selected), corresponding collision elimination constraint for vertex v is included into the encoding; that is, the following constraint is added (lines 23-24):

$$\sum_{a_i \in A} \sum_{\pi \in \Pi(a_i)} s_v^\pi \lambda_{i,\pi} \leq 1, \tag{13}$$

Then cost penalties are introduced for agents using vertex v where the collision happened and the pool of paths is extended with at least one better path for each agent considering the paths' costs and the penalties (lines 25-27). Compared to the SAT-based approach, the extension of the set of candidate paths is more general, more fine grained, and provides a room to integrate heuristics that include more promising paths first. The SAT-based compilation increases the bound of the objective instead, which allows for considering more paths satisfying the bound in future iterations, but such extension adds many new paths all at once without distinguishing if some of them are more promising than others.

On the other hand, the MILP-based approach compared to SAT-based compilation requires to deal with fractional solutions (lines 15-19), which adds non-trivial complexity to the high level solving process. As shown in Algorithm 4, elimination of fractionality of solutions requires to implement *branch-and-bound* search represented in the code by a non-deterministic choice (line 17).

Experiments show that the method performs well even in its vanilla variant. This can be attributed to the strength of the target MILP solver and to the design of decision variables that are suitable for it.

Algorithm 4: Branch Cut and Price: MAPF via MILP.

```
1  BPC-MAPF(MAPF $\mathcal{M}$)
2      $\Pi \leftarrow$ initialize-Paths-Pool($\mathcal{M}$)
3      $constraints \leftarrow \emptyset$
4      while $TRUE$ do
5          $(paths, \Pi') \leftarrow$ BCP-Using-Path-Pool($\mathcal{M}$, $\Pi$)
6          if $paths \neq FAIL$ then
7              return $paths$
8          $\Pi \leftarrow \Pi'$

9  BPC-Using-Path-Pool($\mathcal{M}$,$\Pi$)
10     while $TRUE$ do
11         $LP \leftarrow$ encode-MILP($\mathcal{M}$, $\Pi$, $constraints$)
12         $\{\lambda_{i,\pi}\}_{a_i \in A, \pi \in \Pi(a_a)} \leftarrow$ consult-MILP-Solver($LP$)
13         $conflicts \leftarrow$ check($\mathcal{M}$,$\{\lambda_{i,\pi}\}$)
14         if $conflicts = \emptyset$ then
15             if $\exists \pi \in \Pi, a_i \in A$ such that
                 $\lambda_{i,\pi} > 0 \wedge \lambda_{i,\pi} < 1$ then
16                 $resolvers \leftarrow$ resolve-Fractions($\{\lambda_{i,\pi}\}$)
17                 let $r \in resolvers$ /* non-deterministic */
18                 $constraints \leftarrow constraints \cup \{r\}$
19                 return $(FAIL, \Pi)$
20             else
21                 $paths \leftarrow$ interpret($\mathcal{M}$,$\{\lambda_{i,\pi}\}$)
22                 return $(paths, \Pi)$
23         for each $c \in conflicts$ do
24             $constraints \leftarrow constraints \cup$ eliminate($c$)
25         for $i = 1, 2, ..., k$ do
26             $\Pi_{better} \leftarrow$ better-Paths($i$,$\{\lambda_{i,\pi}\}$,$constraints$)
27             $\Pi(a_i) \leftarrow \Pi(a_i) \cup \Pi_{better}$
```

6 SAT vs. CSP vs. MILP for MAPF

As the pseudo-codes of the SAT-based, the CSP-based, and the MILP-based compilation scheme suggest there are different opportunities how to enhance each approach with MAPF specific heuristics and pruning techniques such as *symmetry breaking* [Li *et al.*, 2020a] or *mutex reasoning* [Zhang *et al.*, 2020] etc. In this regard, MILP-based scheme is one the most open for integration of domain specific improvements as more decisions are made at the high-level, namely branching strategy where heuristics can be included, paths pool refinement that could further guide the search can be modified at the high level.

In contrast to this, SAT-based and CSP-based approaches, leave lot of decisions on general purpose solver into which it is difficult to include any MAPF specific heuristics without changing the implementation of the solver. Hence, the role of the solver is bigger as it solves without any intervention from the high-level the entire NP-hard component of the problem.

7 Conclusion

We believe that the existence of compilation-based MAPF solvers based on diverse formalisms calls for further cross-fertilization between them. There is still an open room for integrating MAPF specific heuristics in a more efficient way into the compilation framework. Moreover, we believe that the cooperation of the main MAPF solver and the solver for the target formalism can be much closer such as in the case of DPLL(T) solvers [Katz *et al.*, 2016].

Acknowledgments

This research has been supported by GAČR - the Czech Science Foundation, grant registration number 22-31346S.

References

[Andersen *et al.*, 2007] Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming - CP 2007, Proceedings*, volume 4741 of *LNCS*, pages 118–132. Springer, 2007.

[Audemard and Simon, 2018] Gilles Audemard and Laurent Simon. On the glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018.

[Biere *et al.*, 2021] A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability: Second Edition*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

[Blazewicz *et al.*, 2004] Jacek Blazewicz, Nadia Brauner, and Gerd Finke. Scheduling with discrete resource constraints. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[Blum and Furst, 1997] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[Bonnet *et al.*, 2018] Édouard Bonnet, Tillmann Miltzow, and Pawel Rzazewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018.

[Bradley and Manna, 2007] Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007.

[Ciré *et al.*, 2013] André A. Ciré, Elvin Coban, and John N. Hooker. Mixed integer programming vs. logic-based benders decomposition for planning and scheduling. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, 2013. Proceedings*, volume 7874 of *LNCS*, pages 325–331. Springer, 2013.

[Clarke *et al.*, 2000] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification, 12th International Conference, CAV 2000, Proceedings*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[Clarke, 2003] Edmund M. Clarke. Sat-based counterexample guided abstraction refinement in model checking. In *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction, Proceedings*, volume 2741 of *LNCS*, page 1. Springer, 2003.

[Cook, 1971] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[Dechter, 2003] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[Desaulniers *et al.*, 2005] G. Desaulniers, J. Desrosiers, and M.M. Solomon. *Column Generation*. Cahiers du GERAD. Springer, 2005.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Selected Revised Papers*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[Erdem *et al.*, 2013] Esra Erdem, Doga Gizem Kisa, Umut Öztok, and Peter Schüller. A general formal framework for pathfinding problems with multiple agents. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, 2013*. AAAI Press, 2013.

[Felner *et al.*, 2017] Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan R. Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017*, pages 29–37. AAAI Press, 2017.

[Gange *et al.*, 2019] Graeme Gange, Daniel Harabor, and Peter J. Stuckey. Lazy CBS: implicit conflict-based search using lazy clause generation. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pages 155–162. AAAI Press, 2019.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[Jünger *et al.*, 2010] Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010.

[Katz *et al.*, 2016] Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. Lazy proofs for dpll(t)-based SMT solvers. In Ruzica Piskac and Muralidhar Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD, 2016*, pages 93–100. IEEE, 2016.

[Kautz and Selman, 1992] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *10th European Conference on Artificial Intelligence, ECAI, 1992. Proceedings*, pages 359–363. John Wiley and Sons, 1992.

[Kautz and Selman, 1999] Henry A. Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 318–325. Morgan Kaufmann, 1999.

[Kornhauser *et al.*, 1984] Daniel Kornhauser, Gary L. Miller, and Paul G. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 241–250. IEEE Computer Society, 1984.

[Lam *et al.*, 2019] Edward Lam, Pierre Le Bodic, Daniel Damir Harabor, and Peter J. Stuckey. Branch-and-cut-and-price for multi-agent pathfinding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 1289–1296. ijcai.org, 2019.

[Li *et al.*, 2020a] Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. New techniques for pairwise symmetry breaking in multi-agent path finding. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, 2020*, pages 193–201. AAAI Press, 2020.

[Li *et al.*, 2020b] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20*, pages 1898–1900. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[Lifschitz, 2019] Vladimir Lifschitz. *Answer Set Programming*. Springer, 2019.

[Mohanty *et al.*, 2020] Sharada Prasanna Mohanty, Erik Nygren, Florian Laurent, Manuel Schneider, Christian Scheller, Nilabha Bhattacharya, Jeremy D. Watson, Adrian Egli, Christian Eichenberger, Christian Baumberger, Gereon Vienken, Irene Sturm, Guillaume Sartoretti, and Giacomo Spigler. Flatland-rl : Multi-agent reinforcement learning on trains. *CoRR*, abs/2012.05893, 2020.

[Muter *et al.*, 2013] undefinedbrahim Muter, ź. undefinedlker Birbil, and Kerem Bülbül. Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. *Math. Program.*, 142(1–2):47–82, 2013.

[Preiss *et al.*, 2017] James A. Preiss, Wolfgang Hönig, Gaurav S. Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017*, pages 3299–3304. IEEE, 2017.

[Prestwich, 2003] Steven D. Prestwich. Local search on sat-encoded colouring problems. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003, Selected Revised Papers*, volume 2919 of *LNCS*, pages 105–119. Springer, 2003.

[Rader, 2010] D.J. Rader. *Deterministic Operations Research: Models and Methods in Linear Optimization*. Wiley, 2010.

[Régin, 1994] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence, 1994*, pages 362–367. AAAI Press / The MIT Press, 1994.

[Ryan, 2008] Malcolm Ross Kinsella Ryan. Exploiting subgraph structure in multi-robot path planning. *J. Artif. Intell. Res.*, 31:497–542, 2008.

[Ryan, 2010] Malcolm Ryan. Constraint-based multi-robot path planning. In *IEEE International Conference on Robotics and Automation, ICRA 2010*, pages 922–928. IEEE, 2010.

[Sharon *et al.*, 2013] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.*, 195:470–495, 2013.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219:40–66, 2015.

[Silver, 2005] David Silver. Cooperative pathfinding. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 117–122. AAAI Press, 2005.

[Snape *et al.*, 2012] Jamie Snape, Stephen J. Guy, Jur van den Berg, Ming C. Lin, and Dinesh Manocha. Reciprocal collision avoidance and multi-agent navigation for video games. In *Multiagent Pathfinding, Papers from the 2012 AAAI Workshop, MAPF@AAAI 2012*, volume WS-12-10 of *AAAI Workshops*. AAAI Press, 2012.

[Standley, 2010] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010.

[Stern, 2019] Roni Stern. Multi-agent path finding - an overview. In *Artificial Intelligence - 5th RAAI Summer School, Russia, 2019, Tutorial Lectures*, volume 11866 of *LNCS*, pages 96–115. Springer, 2019.

[Surynek *et al.*, 2016] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 810–818. IOS Press, 2016.

[Surynek, 2012] Pavel Surynek. On propositional encodings of cooperative path-finding. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012*, pages 524–531. IEEE Computer Society, 2012.

[Surynek, 2019] Pavel Surynek. Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 1177–1183. ijcai.org, 2019.

[Walsh, 2000] Toby Walsh. SAT v CSP. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Proceedings*, volume 1894 of *LNCS*, pages 441–456. Springer, 2000.

[Zhang *et al.*, 2020] Han Zhang, Jiaoyang Li, Pavel Surynek, Sven Koenig, and T. K. Satish Kumar. Multi-agent path finding with mutex propagation. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, 2020*, pages 323–332. AAAI Press, 2020.