Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

# Compiling Cost-Optimal Multi-Agent Pathfinding to ASP

**Rodrigo N. Gómez,**[1] **Carlos Hernández,**[2] **Jorge A. Baier**[1,3]
[1] Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Santiago, Chile
[2] Departamento de Ciencias de la Ingeniería, Universidad Andrés Bello, Santiago, Chile
[3] Instituto Milenio Fundamentos de los Datos, Chile

## Introduction

Multi-Agent Pathfinding (MAPF) over grids is the problem of finding $n$ non-conflicting paths that lead $n$ agents from a given initial cell to a given goal cell. Sum-of-costs-optimal MAPF, or simply cost-optimal MAPF, in addition, minimizes the total number of actions performed by each agent before stopping at the goal. Being a combinatorial problem in nature, a number of compilations from MAPF to Satisfiability (SAT) (Surynek et al. 2016) and Answer Set Programming (ASP) exist (Erdem et al. 2013; Gebser et al. 2018). Here we propose and evaluate a new compilation of MAPF over grids to ASP. Unlike existing compilations we are aware of, both to SAT and to ASP, our encoding is the first that produces a number of clauses that is *linear* on the number of agents. In addition, the clauses that allow representing the optimization objective are also efficiently written, and do not depend on the size of the grid. Like makespan-optimal approaches, our algorithm searches for cost-optimal solutions with increasing makespan. When a solution is found a provably correct upper bound on the maximum makespan at which a true cost-optimal solution exists is computed, and the solver is rerun once more.

## Answer-Set Programming

Answer-set programming (ASP) (Lifschitz 2008) is a logic-based framework for knowledge reasoning and constraint optimization. Optimization problems are modeled with an ASP *program* over a set of *atoms* $\mathcal{P}$, such that a *model* of the program, which is a subset of $\mathcal{P}$, encodes an optimal solution. An ASP program can be viewed as composed by a set of *rules*, a set of *constraints*, and *optimization statements*. Each rule specifies under which conditions certain atoms should become part of the model. In our compilation, we use two types of rules. The first type, *basic rules*, that have the form $p \leftarrow B$, where $p \in \mathcal{P}$ and $B \subseteq \mathcal{P}$, which intuitively specify that when all atoms in $B$ are part of a model, so is $p$. The second type, henceforth referred to as *singleton rules*, have the form $1 = |A| \leftarrow B$, where $A, B \subseteq \mathcal{P}$, and specify that whenever $B$ is contained in the model, only one element of $A$ should be. Constraints of the form $\leftarrow C$, where $C \subseteq \mathcal{P}$, express $C$ cannot be contained in a model. Finally an optimization statement, of the form $minA$ (resp. $maxA$) instructs the solver to minimize (resp. maximize) the number of atoms in $A$ that appear in the model.



## From MAPF to ASP

Like compilations of planning to satisfiability, we use atoms of the form $at(a, x, y, t)$ to specify that an agent $a$ is at cell $(x, y)$ at *time instant* $t$, and our encoding considers instances of these atoms for $t \in \{0, \ldots, T\}$, where $T$ is the makespan of the solution sought. Atoms of the form $exec(a, m, t)$ specify that agent $a$ executes $m$ at time $t$.

**Effects of moves** To encode the effect of agent $a$ performing move $m$ we use a basic rules of the form $at(a, x, y, t+1) \leftarrow at(a, x', y', t), exec(a, m, t) \cup B$, where $B$ specifies the way $(x, y)$ and $(x', y')$ are related depending on $m$.

**Parallel move execution** To encode that each agent performs exactly one move at each time instant, we use singleton rules, expressing that set $\{exec(a, m, t) : m \in Moves\}$ has cardinality 1 for each agent $a$ and each time instant $t$.

**Conflict avoidance** Here we want to express that the paths do not have conflicts. We need to express that (1) no pair of different agents are ever at the same cell and (2) there is no pair of adjacent agents that *swap* cells after performing a single (parallel) move. We do this by adding constraints to the program. We experimented with two approaches.

*Quadratic conflict encoding*. The first approach is similar to what is done in MDD-SAT (Surynek et al. 2016) and ASPRILO (Gebser et al. 2018). We write the constraints $\leftarrow on(a, x, y, t), on(a', x, y, t)$, for every pair of different agents $(a, a')$, every time instant $t$, and every grid cell $(x, y)$. Similarly, we write constraints for disallowing swaps; these are omitted for space, but we need one for each pair of different agents. This yields a representation that is quadratic on the number of agents, and linear on the size of the map.

*Linear conflict encoding*. Here we introduce atoms of the form $rt(x, y, t)$ to specify that the edge between $(x, y)$ and $(x+1, y)$ is traversed by *some* agent at time $t$. $lt(x+1, y, t)$, on the other hand, represents that an agent moved from $(x + 1, y)$ to $(x, y)$ a time $t$. In addition, atom $st(x, y, t)$ represents some agent at $(x, y)$ at time $t$ performed a wait action at $t$. The dynamics of these predicates are defined using basic rules like $lt(x, y, t) \leftarrow exec(a, \text{right}, t), on(a, x, y, t)$ for every agent $a$, cell $(x, y)$, and time instant $t$. Their definition is thus linear in the number of agents and the size

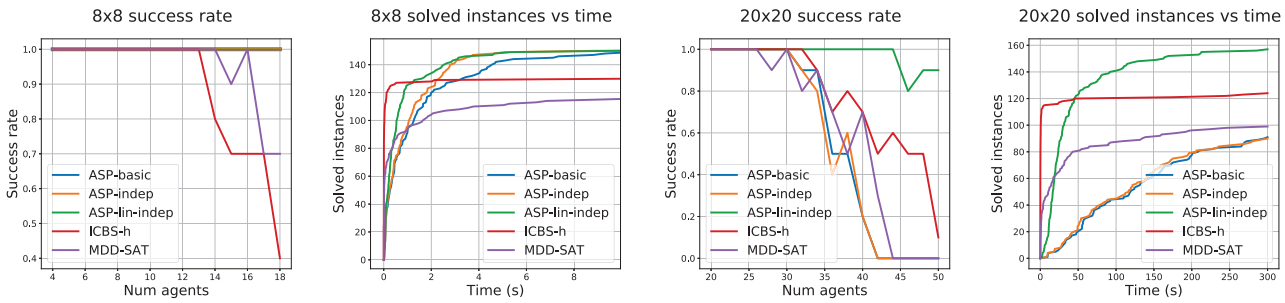


Figure 1: Success rate and number of instances solved versus time on $8 \times 8$ and $20 \times 20$ grids.

of the grid. Finally, for swaps, we express constraints like $\leftarrow rt(x, y, t), lr(x + 1, y, t)$ for every $(x, y)$ in the grid, and time instant $t$. The resulting number of constraints and rules is linear in the size of the map.

**Optimization** To find a cost-optimal solution, we need to encode the minimization of the number of actions executed by the agents. We tried with two different encodings.

*Grid-dependent penalty*. This encoding is similar to the approach used in MDD-SAT: the idea to minimize the actions performed by the agent at each cell. Specifically there is an atom $cost(a, t, 1)$ which specifies that agent $a$ has performed an action at time $t$. To define this atom, we need to specify three rule schemas. One of them is $cost(a, t, 1) \leftarrow on(a, x, y, t), notgoal(a, x, y)$, which establishes that moving the agent from a cell that is not the goal is penalized by one unit. There are three more rule types that we omit for space. The resulting number of rules and constraints grows linearly with the size of the grid and the number of agents. Finally, via an optimization statement, we minimize the number of atoms of the form $cost(a, t, 1)$ in the model.

*Grid-independent encoding*. Here we maximize the slack between the makespan $T$ and the time instant at which an agent has stopped at the goal. We define atoms of the form $at\_goal\_back(a, t)$ which specify that between time instants $t$ and $T$ agent $a$ is at the goal. The rule defining these atoms ensures that no action other than wait has been performed between $t$ and $T$, without making reference to the cells int the grid, resulting in an encoding that does not depend on the grid, and thus is linear in the number of agents.

**Achieving the goal** Via a constraint, we specify that no agent is away from its goal at time $T$.

**Obtaining an optimal solution** Let $T_0$ and $C_0$ be, respectively, the makespan and cost of the solution that is obtained by solving the problem independently for each agent, ignoring all others. To obtain an optimal solution we iterate from $T = T_0$ incrementing $T$ by one until a solution is found. Now let $T_0$ and $C_0$ be, respectively, the makespan and cost of the first solution found. Then we run the solver for one last time with makespan $T_1 + C_1 - 1 - C_0$, which we can prove will return an optimal solution.

## Empirical Evaluation

The objective of our preliminary evaluation was to evaluate combinations of our encodings and compare them to publicly available search-based solvers as well as MDD-SAT (Surynek 2014) (enc=mdd), on congested grids. We ran two state-of-the-art search algorithms: EPEA* (Goldenberg et al. 2014), and ICBS-h (Felner et al. 2018), but decided just to report on the latter, since it was the best performing.

We used Clingo 5.3 as the underlying ASP solver. Clingo was run with 4 threads in *parallel-mode*, and using *usc* as the optimization strategy. Experiments were run on a 3.40GHz Intel Core i5-3570K computer with 8GB of memory. We set a runtime limit of 5 minutes for all the problems.

We experimented on $8 \times 8$ and $20 \times 20$ randomly generated problems with 10% obstacles. For $8 \times 8$ (resp. $20 \times 20$) we generate 150 (resp. 160) problems with the number of agents in $\{4, \ldots, 18\}$ (resp. $\{20, 22, \ldots, 50\}$). Success rates, and number of problems solved versus time are shown in Figure 1, where ASP-basic is a basic encoding that uses quadratic conflict resolution and grid dependent penalties. ASP-indep uses quadratic conflict resolution and grid independent penalties. Finally, ASP-lin-indep uses linear conflict resolution and grid independent penalties. We conclude that in congested grids, ASP-lin-indep is superior to both search-based the other compilation-based approaches.

## References

Erdem, E.; Kisa, D. G.; Öztok, U.; and Schüller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*. AAAI Press.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, 83–87.

Gebser, M.; Obermeier, P.; Otto, T.; Schaub, T.; Sabuncu, O.; Nguyen, V.; and Son, T. C. 2018. Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming* 18(3-4):502–519.

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

Lifschitz, V. 2008. What is answer set programming? In *AAAI*, 1594–1597.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI*, 810–818. IOS Press.

Surynek, P. 2014. Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *ICTAI*, 875–882. IEEE Computer Society.