A General Formal Framework for Pathfinding Problems with Multiple Agents*

Esra Erdem and **Doga G. Kisa** and **Umut Oztok** and **Peter Schüller**
Faculty of Engineering and Natural Sciences
Sabancı University
34956 İstanbul, Turkey
{esraerdem,dgkisa,uoztok,peterschueller}@sabanciuniv.edu

Abstract

Pathfinding for a single agent is the problem of planning a route from an initial location to a goal location in an environment, going around obstacles. Pathfinding for multiple agents also aims to plan such routes for each agent, subject to different constraints, such as restrictions on the length of each path or on the total length of paths, no self-intersecting paths, no intersection of paths/plans, no crossing/meeting each other. It also has variations for finding optimal solutions, e.g., with respect to the maximum path length, or the sum of plan lengths. These problems are important for many real-life applications, such as motion planning, vehicle routing, environmental monitoring, patrolling, computer games. Motivated by such applications, we introduce a formal framework that is general enough to address all these problems: we use the expressive high-level representation formalism and efficient solvers of the declarative programming paradigm Answer Set Programming. We also introduce heuristics to improve the computational efficiency and/or solution quality. We show the applicability and usefulness of our framework by experiments, with randomly generated problem instances on a grid, on a real-world road network, and on a real computer game terrain.

Introduction

Pathfinding for a single agent is the problem of planning a route from an initial location to a target location in an environment, going around obstacles. Pathfinding for multiple agents (**PF**) also aims to plan such routes for each agent, but subject to various constraints, such as no self-intersecting paths, no intersection of paths/plans, no crossing/meeting each other, no waiting idle, restrictions on the length of each path/plan and on the total length of paths/plans, and requirements on visiting multiple target locations. **PF** also has variations for finding optimal solutions where the goal is to minimize the maximum path/plan length, the sum of path/plan length, the number of target locations visited, etc. Some of these **PF** problems have been studied in the literature, but under different names. For instance, if it is required that the plans of the agents do not interfere with each other then **PF** is called multi-agent pathfinding problem. If it is required that each agent visits multiple target locations then **PF** is called multi-robot routing problem. Assuming that agents are homogeneous and move one unit at a time, deciding whether a solution of at most k moves to multi-agent pathfinding exists is NP-complete (Ratner and Warmuth 1986); optimization variants of both problems are NP-hard (Lagoudakis et al. 2005; Surynek 2010).

Despite their difficulty, **PF** problems have played a significant role in different applications, such as motion planning (Bennewitz, Burgard, and Thrun 2002), vehicle routing and traffic management (Svestka and Overmars 1998; Dresner and Stone 2008; Pallottino et al. 2007), air traffic control (Tomlin et al. 1998), computer games (Silver 2005), disaster rescue (Kitano et al. 1999; Jennings, Whelan, and Evans 1997), formation generation for multi-agent networks (Smith, Egerstedt, and Howard 2009), patrolling and surveillance (Machado et al. 2002; Xu and Stentz 2011).

In these applications, **PF** is solved subject to various relevant constraints. For instance, in computer games such as Warcraft III, the routes of agents may intersect with each other as long as the agent's plans do not interfere with each other (i.e., the agents can be at the same location in different time steps, but not at the same time). On the other hand, in environmental coverage or surveillance, it may be required that the routes of agents do not intersect with each other at all so that more areas are covered in a shorter amount of time (note also the limited power supplied by batteries of robots). In disaster rescue, it may be required that certain parts of the world are checked by some agent (i.e., they should be covered by some route) since it is more probable for some people to be there. To prevent an agent to do all the work, a restriction may be specified on the length of the route or the duration of the plan for each agent. Furthermore, the maximum plan length or the sum of plan lengths may be minimized to save some battery power.

We introduce a formal framework that is general enough to solve many variations of **PF** (including the ones mentioned above) declaratively, with the possibility of guaranteed optimality with respect to some criteria based on plan lengths, and with the possibility of embedding heuristics. This framework is based on Answer Set Programming (ASP) (Lifschitz 2008; Brewka, Eiter, and Truszczynski 2011)—a knowledge representation and reasoning paradigm with an expressive formalism and efficient solvers. The idea of ASP is to formalize a given problem as a "program" and to solve the problem

*This work is partially supported by TUBITAK Grant 111E116. Peter Schüller is supported by TUBITAK 2216 Fellowship.

by computing models (called "answer sets" (Gelfond and Lifschitz 1991)) of the program using "ASP solvers", such as CLASP (Gebser et al. 2007). The expressive formalism of ASP allows us to easily represent variations of **PF**, as well as sophisticated heuristics to improve the computational efficiency and/or quality of solutions. Such a flexible elaboration tolerant (McCarthy 1998) general framework is useful in studying and understanding variations of **PF** and different heuristics in different applications. Deciding whether an ASP has an answer set for nondisjunctive programs is NP-complete (Dantsin et al. 2001), therefore ASP is expressive enough for solving many **PF** problems.

We show the applicability and effectiveness of our ASP-based framework (from the point of view of computational efficiency and quality of solutions) by experiments with randomly generated problem instances on a grid, on a real-world road network, and with a computer game terrain.

Pathfinding for Multiple Agents

We consider a general pathfinding problem (i.e., **PF**) where multiple agents need to find paths from their respective starting locations to their goal locations, ensuring that paths do not collide with static obstacles and that no two agents collide with each other, and view **PF** as a graph problem as follows.

Input:

- a graph $G = (V, E)$,
- a positive integer k,
- a function h that maps every positive integer $i \leq k$ to a pair (v_i, u_i) of vertices in V,
- a set $O \subseteq V$, and
- a function g that maps every positive integer $i \leq k$ to a positive integer l_i.

Output: For every positive integer $i \leq k$ with $h(i) = (v_i, u_i)$ and $g(i) = l_i$,

- a path $P_i = \langle w_{i,1}, \ldots, w_{i,n_i} \rangle$ for some $n_i \leq l_i$ from $w_{i,1} = v_i$ to $w_{i,n_i} = u_i$ in G where each $w_{i,j} \in V \setminus O$,
- a function f_i that maps every nonnegative integer less than or equal to l_i to a vertex in P_i such that,
  (i) for every $w_{i,j}$, $w_{i,j'}$ in P_i and for every nonnegative integer $t < l_i$, if $f_i(t) = w_{i,j}$ and $f_i(t{+}1) = w_{i,j'}$ then $w_{i,j'} = w_{i,j}$ or $w_{i,j'} = w_{i,j+1}$; and
  (ii) for different paths P_i and P_j and positive integers $t_i \leq l_i$, $t_j \leq l_j$, if $f_i(t_i) = f_j(t_j)$ then $t_i \neq t_j$.

Intuitively, graph G characterizes the environment (e.g., a game terrain) where the agents move around, positive integer k denotes the number of agents, function h describes the initial and goal locations of agents, set O denotes the parts of the environment covered by the static obstacles, and function g specifies the maximum plan length l_i for each agent i. A path P_i in G from an initial location v_i to a goal location u_i characterizes the path that the agent i plans to traverse. The accompanying function f_i denotes which vertices in the path are visited when; the conditions on f_i makes sure that (i) the agent visits consecutive vertices in P_i at consecutive time steps, or waits at a vertex (e.g., to give way to other agents), and that (ii) no two agents meet at the same place (collision).

An upper bound l_i on the plan length is specified as part of the **PF** problem to avoid that one agent does all the work in case some parts of the environment should be visited by some agents. One can specify a very large l_i for each agent to discard a tight bound on path lengths.

We define variations of **PF** by restricting solutions further using the following *constraints* (ASP can handle these and more constraints, which we omit for space reasons):

C No path P_i has a cycle.
This constraint can be useful in cases where robotic agents shall not visit the same part of the environment many times for a more efficient use of their batteries.

I No two different paths P_i and P_j $(i < j)$ intersect with each other.
This constraint can be useful in cases where it is sufficient if only one robotic agent visits each part of the environment, also for a more efficient use of their batteries.

V Some specified vertices should be visited by some path P_i.
This constraint can be useful, for instance, in a disaster rescue scenario, where it is essential that certain parts of the environment are visited by some agent.

W Waiting of agents is not allowed (e.g., to minimize idle time): for every path P_i and for every nonnegative integer $t_i < l_i$, if $f_i(t_i) \neq u_i$, i.e., agent i has not yet reached its goal u_i, then $f_i(t_i) \neq f_i(t_i{+}1)$.

X Agents cannot "pass through" each other (switch place): for every two different paths P_i and P_j and for every nonnegative time step $t < \min(l_i, l_j)$, if $f_i(t) = w_x$, $f_i(t{+}1) = w_{x'}$, and $f_j(t) = w_{x'}$ then $f_j(t{+}1) \neq w_x$.

L The sum of the plan lengths is less than or equal to a given positive integer: for each agent i, the smallest time step t_i such that $f(t_i) = u_i$ denotes the length of its plan. Formally the sum of plan lengths is $\sum_{i=1}^{k} \min\{t_i : f(t_i) = u_i\}$.
This constraint can be useful in cases where we want to minimize the total time (and energy) spent by agents.

Arbitrary combinations of these constraints can be considered; for instance multi-agent pathfinding problems consider **X**. Some of them, e.g., (Standley 2010; Standley and Korf 2011; Sharon et al. 2011; Yu and LaValle 2013) focus on finding solutions where the sum of plan lengths is as small as possible (as suggested by **L**). Problems studied by patrolling/surveillance applications (Machado et al. 2002) do not consider **X**, but since they focus on finding plans that ensure some parts of the environment are visited by some agent, they consider **V** and sometimes **I** (Xu and Stentz 2011). Multi-robot routing problems (Lagoudakis et al. 2005; Kishimoto and Sturtevant 2008) where robots move one unit at a time also consider **V**, for some allocation of target locations to robots, as well as **L** to minimize total path lengths.

An interesting variation of **PF** aims to reach all goals as soon as possible. We call this problem **TPF**, it minimizes maximum plan length over all agents, formally it minimizes $\max_{i=1}^{k} \min\{t_i : f(t_i) = u_i\}$.

Solving PF in ASP

We represent a **PF** problem as a program P in ASP, whose answer sets correspond to solutions of the problem. We refer readers to (Lifschitz 2008; Brewka, Eiter, and Truszczynski 2011), for syntax and semantics of programs.

We describe the input $I = (G, k, h, O, g)$ of a **PF** instance by a set F_I of facts: $edge(v, u)$ represents edges $(v, u) \in E$; $start(i, v)$ and $goal(i, u)$ represent start and goal vertices of each agent $i \le k$ (i.e., $h(i) = (v, u)$); finally $clear(v)$ represents that $v \in V \setminus O$.

The output (P_i, f_i) of **PF** characterizes for each agent i a path plan such that the agent reaches the goal location from its initial location and avoids obstacles. We represent path plans by atoms of the form $path(i, t, v)$ which specify that at time step t, agent i is at vertex v, formally $f_i(t) = v$.

The ASP program P defines path plans recursively. The first vertex visited by agent i at step 0 is its initial location v:

$$path(i, 0, v) \leftarrow start(i, v).$$

If a vertex v is visited by the agent i at step t ($0 \le t < l_i$, recall $g(i) = l_i$), then either the agent waits at v or it moves along an edge (v, u) to an adjacent vertex u:

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

We ensure that agents do not go through obstacles:

$$\leftarrow path(i, t, v), not\ clear(v)$$

and that each agent i reaches its goal v:

$$\leftarrow goal(i, v), not\ visit(i, v)$$
$$visit(i, v) \leftarrow path(i, t, v)$$

where $visit(i, v)$ describes that the path of agent i contains v. We also ensure that agents do not collide with each other:

$$\leftarrow path(i, t, v), path(i', t, v) \quad (v \in V, 1 \le i < i' \le k, 0 \le t \le l_i, l_j)$$

The ASP program P is sound and complete.

Theorem 1. *Given a problem instance $I = (G, k, h, O, g)$, for each answer set S of $P \cup F_I$, the set of atoms of the form $path(i, t, v)$ in S encodes a solution (P_i, f_i) ($1 \le i \le k$) to the* **PF** *problem. Conversely each solution to the* **PF** *problem corresponds to a single answer set of $P \cup F_I$.*

The atoms in P that include time steps only depend on atoms of previous time steps. So we can use the splitting set theorem and the method proposed in (Erdogan and Lifschitz 2004) iteratively at each time step to eventually show (by induction) that the answer sets for $P \cup F_I$ characterize obstacle- and collision-free paths for agents.

Solving Variations of PF in ASP

To solve variations of **PF** in ASP, we simply add to the main program P described above, a set of ASP rules which encode the relevant constraints. For instance, to solve multi-agent pathfinding, we add to P the ASP constraints describing $\underline{\mathbf{X}}$. It is important to emphasize here that, we do not modify the rules of the program P; in that sense, our formulation of **PF** in ASP is elaboration tolerant (McCarthy 1998). Constraints are formulated as follows:

$\underline{\mathbf{C}}$ (i.e., no cycles in a path) can be expressed in ASP by the following constraint for each agent $i \in 1 \dots k$:

$$\leftarrow 2\{path(i, 0, v), \dots, path(i, l_i, v)\}, not\ goal(v). \quad (v \in V)$$

These constraints ensure that, for every agent i, no non-goal vertex in the path P_i is visited twice or more by the agent i.

$\underline{\mathbf{I}}$ (i.e., no intersection of paths) is encoded as

$$\leftarrow visit(i, v), visit(i', v) \quad (v \in V, 1 \le i < i' \le k)$$

which ensures that no two agents i, i' visit the same vertex v.

$\underline{\mathbf{V}}$ (i.e., visit specified vertices) is represented in ASP by

$$\leftarrow required(v), \{visit(1, v), \dots, visit(k, v)\}0 \quad (v \in V)$$

where $required(v)$ describes the vertices to be visited.

$\underline{\mathbf{W}}$ (i.e., no waiting) can be formalized as

$$\leftarrow path(i, t, v), path(i, t{+}1, v), not\ goal(i, v) \quad (v \in V, 1 \le i \le k, 0 \le t < l_i).$$

$\underline{\mathbf{X}}$ (i.e., no swapping places) can be represented by the following constraints for pairs of distinct agents i, j:

$$\leftarrow path(i, t, v), path(i, t{+}1, w), path(j, t, w), path(j, t{+}1, v) \quad (i \ne j, (v, w) \in E, 0 \le t < l_i, l_j).$$

$\underline{\mathbf{L}}$ (i.e., the total plan length is restricted by a positive integer z) can be formalized in ASP as follows

$$\leftarrow totalPlanLength(t) \quad (t > z)$$

where $totalPlanLength(t)$ ("the sum of all plan lengths is t") is defined as follows:

$$totalPlanLength(t) \leftarrow sum\langle\{x : planLength(i, x)\}\rangle = t$$
$$planLength(i, t) \leftarrow path(i, t, v), goal(i, v), path(i, t{-}1, v'), not\ goal(i, v') \quad (0 < t \le l_i, v, v' \in V, 1 \le i \le k).$$

Here the aggregate sum is used to find the total plan lengths; and $planLength(i, t)$ describes the plan length for agent i.

We can formalize **TPF** in ASP, by simply adding to the formulation of **PF** in ASP (i.e., program P), the rules above that define $planLength(i, x)$ and the following rules:

$$maxPlanLength(t) \leftarrow max\langle\{x : planLength(i, x)\}\rangle = t$$
$$\#minimize\ [\, maxPlanLength(t) = t\,].$$

The optimization variant of multi-agent pathfinding, which minimizes the maximum plan length, can be represented by adding Constraint $\underline{\mathbf{X}}$ to the formulation of **TPF** in ASP.

A similar optimization variant of multi-robot routing with target vertices (described by atoms $target(v)$) can be represented by adding to the formulation of **TPF** in ASP the Constraint $\underline{\mathbf{V}}$ and the rules

$$1\{goal(i, v) : target(v)\}1 \leftarrow \quad (1 \le i \le k)$$
$$required(v) \leftarrow target(v), \{goal(1, v), \dots, goal(k, v)\}0.$$

that allocate the given target vertices to agents.

Experimental Results

We performed experiments with various randomly generated instances of **PF** to be able to understand

- how the input parameters affect the computation time and solution quality (i.e., average path plan length);
- how adding constraints to the main problem formulation affects the computation time and solution quality;
- how various heuristics affect the computation time and solution quality.

Table 1: **PF** and **TPF** on random 25×25 grid graphs with a variable number of obstacles o and agents k.

o %	k	Grounding sec	First Solution sec (plan length)	Optimal Solution sec (plan length)
10	5	0.70	0.27 (30.4)	7.60 (27.4)
	10	1.88	0.84 (31.2)	13.39 (29.7)
	15	3.43	1.39 (32.9)	18.97 (31.6)
	20	6.27	3.66 (33.4)	30.78 (32.6)
20	5	0.39	0.16 (28.0)	4.21 (26.0)
	10	1.59	0.68 (31.5)	16.12 (29.6)
	15	3.51	1.88 (38.0)	20.71 (36.0)
	20	5.90	4.13 (35.8)	31.35 (33.8)
40	5	0.67	0.28 (58.8)	7.12 (54.5)
	10	2.10	1.19 (63.8)	18.07 (59.6)
	15	3.78	4.99 (62.0)	29.19 (57.8)
	20	7.28	10.58 (69.7)	45.30 (66.6)

We randomly generated problem instances of **PF** on three sorts of graphs: randomly generated grid graphs, a real road network, and a computer game terrain.

- The grid graphs are of size 25×25. We varied the number of agents $k = 5, 10, 15, 20$ and the percentage of grid points covered by obstacles $o = 10, 20, 40$.
- The road network (Xu and Stentz 2011) is a graph with 769 vertices and 1130 edges. We considered $k = 5, 10, 15, 20$.
- The computer game terrain (Sturtevant 2012) is a map (called battleground.map) of a computer game Warcraft III. It is a 512×512 grid-based graph which consists of 262144 vertices and 523264 edges, where 92268 of the vertices are not covered by obstacles. In our experiments, for $k = 5, 10, 15, 20, 25$, we sampled k start and goal configurations with Euclidean distance of at most 25.

For grid graphs and the road network, we used a timeout of 1000 CPU seconds and a memory limit of 4GB. For the larger computer game dataset, we limited memory usage to 10GB.

We used the ASP solver CLASP (Version 2.1.1) with the grounder GRINGO (Version 3.0.5) on a machine with four 1.2GHz Intel Xeon E5-2665 8-Core Processors and 64GB RAM. We used CLASP in single-threaded mode with the command line `--configuration=handy` as this configuration performs best in the majority of cases. CPU times are reported in seconds. In tables, each row gives averages over 10 randomly generated instances, for CPU times and plan lengths.

We assumed that the maximum plan length for every agent is identical: for every i, $g(i) = l_i = l$. In our experiments, we started with $l = 30$, and increased it by 10 (until $l = 80$) if the solver proved that there exists no plan for that l. We repeated this until $l = 80$ and above that we considered the problem unsolved.

Experiments on Artificial Grid Graphs The goal of these experiments is to understand how the parameters of instances ($n \times n$: grid size, k: number of agents, o: percentage of obstacles) affect the computation time.

Table 1 shows results of our experiments, where we vary the number of agents and the percentage of blocked vertices in the grid graph. Every row in the table presents three CPU times: 1) for grounding, 2) for finding some solution (possibly non-optimal, but less than the given upper bound l on the plan length), and 3) for finding an optimal solution. The second CPU time (for finding the first solution) is included in the third CPU time (for an optimal solution). Plan lengths (solution quality) are presented in parentheses. For instance, with $o = 40\%$ of obstacles and $k = 20$ agents, GRINGO grounds the instances in 7.28 seconds, CLASP computes some solution to **PF** in 10.58 seconds with a plan length of 69.7; finding an optimal solution (of length 66.6) takes 45.30 seconds.

Table 2: **PF/TPF** with combinations of $\underline{\mathbf{C}}$,$\underline{\mathbf{W}}$,$\underline{\mathbf{X}}$ on random 25×25 grids with $k = 15$ agents and $o = 20\%$ obstacles.

C	W	X	Grounding sec	First Solution sec (length)	Optimal Solution sec (plan length)
			3.30	2.56 (34.8)	19.09 (34.4)
	x		3.71	12.98 (38.1)	37.52 (34.4)
		x	11.24	3.35 (36.7)	32.19 (34.4)
x			3.64	18.74 (39.0)	48.96 (34.4)
x	x		3.85	27.82 (39.0)	66.31 (34.4)
	x	x	11.62	18.30 (39.0)	70.02 (34.4)
x		x	11.61	34.33 (38.6)	103.28 (34.4)
x	x	x	11.59	33.32 (39.0)	116.13 (34.4)

We observe from Table 1 that increasing the number of agents increases both the grounding time and the solution time (first and optimal solution). The number of obstacles, on the other hand, primarily influences the plan length of an optimal solution, e.g., with 20 agents the average optimal plan length is 33.8 steps for 20% obstacles, compared to 66.6 steps for 40% obstacles. While the plan length increases with more obstacles, the time spent to find such a solution stays fairly stable between 10% and 20% obstacles, and increases to less than twice with 50% obstacles. Solution time is primarily determined by the number of agents.

Experiments with Constraints The goal of these experiments is to understand the effect of constraints on the computation time and the solution quality (average plan length). We considered variations of **PF** with all combinations of $\underline{\mathbf{C}}$, $\underline{\mathbf{W}}$, and $\underline{\mathbf{X}}$, according to which paths must not have cycles, agents are forbidden to wait idle, and head-on collisions of agents are forbidden, respectively. Table 2 shows the results of these experiments, sorted by the overall time to compute an optimal solution (i.e., by the sum of Grounding and Optimal Solution).

We observe from Table 2 that $\underline{\mathbf{C}}$ and $\underline{\mathbf{W}}$ marginally increase the grounding time; and, contrary to our expectations, $\underline{\mathbf{W}}$ and $\underline{\mathbf{C}}$ do not improve the quality of the initial solution, or reduce the solution time by constraining the problem more. $\underline{\mathbf{X}}$ significantly increases the grounding time; however, among all constraints, it has the least effect on the time for finding initial solutions. All constraints significantly increase the time to find optimal solutions; their combinations increase this time even more.

Which constraints to add to the main formulation should be decided on a case-by-case basis depending on the actual application. For instance, since graph representations are discrete abstractions of an environment, and the computed discrete paths characterize the continuous trajectories followed by the agents, $\underline{\mathbf{X}}$ may be ignored in some applications where the agents do not necessarily follow straight paths. It may not be

Table 3: **PF/TPF** with redundancy elimination (E) and circle heuristics (R) on random 25×25 grid graphs with $k=15$ agents and $o=20\%$ obstacles.

E	R	Grounding sec	First Solution sec (plan length)	Optimal Solution sec (plan length)
		3.48	2.44 (34.8)	19.32 (34.4)
x		16.71	5.23 (36.9)	56.24 (34.4)
	x	1.87	0.95 (37.1)	14.47 (34.5)†
x	x	17.18	2.11 (38.4)	30.33 (34.5)‡

† 2 instances unsatisfiable, ‡ 2 instances out of memory

possible for two agents to move in opposite directions on an edge (v, u) in the given graph, but it may be possible in the environment for one agent to move from v to u and the other agent to move from u to v following different continuous trajectories. On the other hand, for some other applications where robots move via narrow roads, $\underline{\mathbf{X}}$ may be required.

We also experimented with other constraints: Adding $\underline{\mathbf{I}}$ to the ASP formulation of **PF** increases the computation time in many problems. Note that since $\underline{\mathbf{I}}$ ignores the time of an agent visiting a vertex, it may be too strong for many **PF** applications. Adding $\underline{\mathbf{L}}$ to the ASP formulation of **PF** similarly increases the computation time in many problems.

Experiments with Heuristics We utilized the "circle heuristic" (Erdem and Wong 2004) to improve the computational efficiency in terms of computation time and consumed memory. The circle heuristic identifies, for each agent, a subgraph of the given graph that is more "relevant" for that agent to search for a path: we introduce two "circles" with a given radius around the start and the goal positions of the agent, and require that the path connecting the start and the goal positions is contained in the union of these two circles. The radius can be defined as a constant, or a function of some distance between the start and the goal positions. By preprocessing, for each agent i, we identify the relevant edges (v, u) of the graph and represent them as facts of the form $\mathit{relevantEdge}(i, v, u)$; and replace in P all atoms of the form $\mathit{edge}(v, u)$ by $\mathit{relevantEdge}(i, v, u)$.

We also experimented with a "redundancy elimination" heuristic to eliminate certain redundant moves of agents; moving from vertex u to v via other vertices is not allowed if an edge (u, v) exists, except if the agent is waiting at u or v:

$$\begin{aligned} \leftarrow\ & \mathit{path}(i,t,u), \mathit{path}(i,t',v), \mathit{edge}(u,v),\\ & \mathit{not\ path}(i,t{+}1,u), \mathit{not\ path}(i,t'{-}1,v). \end{aligned}$$

where $0 \le t < t' < l_i$, $t{+}1 < t'$, $1 \le i \le k$, $v, u \in V$, and $v \neq u$. This heuristic intuitively removes redundancies in paths, and thus it is expected to improve the quality of solutions by making average plan lengths smaller. Note that there still may be redundancies if the specified maximum plan length is not small enough.

To analyze the effect of adding these heuristics, we considered randomly generated instances of **PF**, over 25×25 grid graphs with $o = 20\%$ obstacles and $k = 15$ agents. With the circle heuristic, for each agent i, we considered a radius of $\lceil \frac{ED_i}{2} \rceil + 3$ where ED_i is the Euclidean distance between the start and the goal locations of agent i.

Table 3 shows the results of our experiments. Each row in this table shows averages over the same set of instances used in our experiments with constraints (Table 2). Here E and R denote the redundancy elimination and the circle heuristics.

Table 4: **PF/TPF** with circle heuristics (R) on a road network.

k	Grounding sec	First Solution sec (plan length)	Optimal Solution sec (plan length)
5	1.02	0.16 (29.5)	4.97 (24.4)
10	1.39	0.59 (32.0)	11.65 (29.4)
15	3.30	1.14 (35.6)	18.16 (32.5)
20	4.42	1.68 (34.9)	22.35 (32.7)
25	7.60	4.13 (39.2)	37.73 (34.6)

Table 5: **PF$_{\underline{\mathbf{W}}}$/TPF** with circle heuristics (R) on game map.

k	Grounding sec	First Solution sec (plan length)	Optimal Solution sec (plan length)
5	22.94	0.29 (34.8)	9.07 (29.0)
10	56.48	0.81 (34.7)	23.14 (33.0)
15	109.09	1.25 (45.7)	29.45 (45.7)
20	80.76	1.14 (35.3)	26.71 (35.3)
25	119.90	1.82 (40.3)	40.48 (40.3)

We observe that, since the redundancy elimination heuristic adds further constraints to the problem, the grounding time increases. These constraints, furthermore, do not constrain the search space enough to allow the solver engine to find solutions faster; therefore, the solution time also increases. Contrary to what we expected, solution quality also becomes worse with redundancy elimination. The additional constraints seem to be misleading for the solver, resulting in worse solution quality and worse efficiency.

With the circle heuristic, only some parts of the graph are considered while computing a solution; therefore, this heuristic significantly reduces both the grounding time and the time to find an optimal solution. Recall, however, that with a small value of the radius, the circle heuristic is neither sound nor complete: the optimal solution found for **PF** with the circle heuristic may not be an optimal solution for **PF**; also there may be instances of **PF** that have some solutions, but using the circle heuristic eliminates all solutions.

Experiments on a Real Road Network The results of our experiments with randomly generated instances of **PF** on the road network, using the circle heuristics, are shown in Table 4.

We observe that, with an increasing number of agents, grounding time does not increase as fast as the time to prove optimality. Finding an initial solution is fast in these instances, and plan length averages show that initial solutions are often already optimal.

The time to prove optimality is greater than the time for finding an initial solution; and quality of initial and optimal solutions are close to each other. Therefore, on a road network, it might be advantageous to try to find some solution (rather than an optimal one).

We also observed that optimal solution lengths are as same as the lengths of optimal solutions computed without using the circle heuristic.

Experiments on a Real Game Terrain The results of experiments with randomized instances of **PF$_{\underline{\mathbf{W}}}$** on a game terrain, using the circle heuristics, are shown in Table 5.

We observe at Table 5 that, due to the larger grid size, the problem instances are also bigger in terms of number of facts that describe the map. Therefore grounding consumes a lot of time for this setting. On the other hand, for applications where the environment (e.g., game terrain map) does not change, we can do grounding only once and reuse the ground ASP program for different problem instances in the same environment.

In this domain, an initial solution is found in a second, whereas finding an optimal solution and proving its optimality take a significant amount of time. Fortunately, the average plan length of initial solutions does not differ much from the average plan length of optimal solutions. Therefore, computing only the initial solution might be sufficient in practice.

Furthermore, we observed that optimal solution lengths are the same when they are computed with and without using the circle heuristics.

Related Work

Our formal framework for **PF** is general enough to solve variations of pathfinding problems with multiple agents, including multi-agent pathfinding (MAPF) and multi-robot routing. Many of these variants have been studied in the literature; thus a comprehensive comparison with the existing approaches is not possible within limited space. Therefore, we briefly discuss related work on MAPF, and report some preliminary experimental results.

Most of the existing solutions to MAPF apply some sort of A* search algorithm, with *decoupled pathfinding* or *centralized pathfinding* approach. In the former approach (Silver 2005; Dresner and Stone 2008; Wang and Botea 2008; Jansen and Sturtevant 2008), a path is computed for each agent independently; in case a conflict occurs (e.g., two agents attempt to move to the same location), it is resolved by replanning one of the conflicting agents' route. Although this approach could be used to solve large MAPF instances quickly, it lacks the optimality and completeness guarantees. The latter approach (Ryan 2008; Surynek 2009; Standley 2010) considers the multi-agent system as a single-agent system by combining state spaces of each agent into one state space and then use a search algorithm to find paths for all agents simultaneously. Although the centralized approach can guarantee optimality and completeness, it is not as efficient (in terms of computation time) as the decoupled approach for large problems. More recently, some decoupled pathfinding algorithms (Luna and Bekris 2011; Wang and Botea 2011) are introduced to guarantee completeness for some graphs; and some (Standley and Korf 2011) optimality. Some centralized planners (Ryan 2010; Khorshid, Holte, and Sturtevant 2011) use heuristics to find suboptimal solutions to improve computational efficiency. Our approach to MAPF is centralized, guarantees various sorts of optimality (thanks to elaboration tolerant representation of **PF** and various constraints), soundness and completeness (Theorem 1). As observed from experiments, with some heuristics, the computational efficiency can be improved also.

It is important to emphasize here that, unlike our ASP-based approach, the search-based methods above do not provide a formal framework; and thus it is hard to ensure and verify various properties (e.g., constraints mentioned above) over paths unless the modeling of the problem and the implementation of the algorithm are modified for each case.

One of the closest related work to ours is by Yu and LaValle (2013): like our approach, the authors introduce a formal framework for solving MAPF to minimize overall plan length on arbitrary graphs with a centralized approach, but using integer linear programming (ILP) instead of ASP. Differently to Yu and Lavalle, our approach is general enough to solve variations of **PF**, some of which are not (or not easily) representable in ILP (e.g., acyclicity constraints). We compared the ILP approach to ours using 180 randomly generated **TPF** instances of 25x25 grid graphs with 0-40% obstacles, and with either 10 or 20 agents (10 instances for each configuration). We used 1000 seconds timeout and report averages and standard deviation for finding optimal solutions. We observed that memory usage was higher for ILP (<10GB) than for ASP (<4GB). With 10 agents, ILP found optimal solutions faster (3 seconds) than ASP (11 seconds); average plan length was 27 steps. With 20 agents, ILP did not return a solution for 7 of the 180 instances and solved the remaining instances in 42 seconds on average (deviation 74 seconds), while ASP timed out only for 2 of 180 instances and found optimal solutions for the other instances in 50 seconds on average (deviation 36 seconds); average plan length was 30 steps. We observed that an increased amount of obstacles degrades both ILP and ASP performance, however ILP performance degrades much stronger. This is because ILP is based on linear optimization with additional support for boolean variables, while ASP is well-suited for finding solutions to highly constrained boolean search spaces.

Although the approaches are quite different, we also compared our (centralized, complete, optimal) ASP-based approach with the state-of-the-art (decoupled, incomplete, nonoptimal) MAPF solver MAPP (Wang and Botea 2011) with some randomly generated instances of the game Baldur's Gate, described in (Wang and Botea 2011). Preliminary results confirm our expectations (as also observed in previous studies comparing decoupled and centralized approaches): in terms of computation time MAPP performs better than our approach as the number of agents and the grid size increases; on the other hand, some problems with multiple conflicts cannot be solved by MAPP while they can be solved by our approach. A more detailed comparison is ongoing work.

Discussion and Conclusion

We have introduced a general formal framework to solve various pathfinding problems with multiple agents (**PF**), using ASP. We have shown that, due to the expressive formalism of ASP, we can easily represent **PF** and its variations subject to different constraints on the paths, and heuristics to improve computational efficiency and quality of solutions. Such a flexible elaboration tolerant framework is important in studying and understanding **PF** and its applications in different domains (e.g., motion planning, vehicle routing, environmental monitoring, patrolling/surveillance, computer games). In particular, that our framework can be applied to any sort of graphs (e.g., not necessarily grid graphs or trees) is advantageous for various robotic applications.

References

Bennewitz, M.; Burgard, W.; and Thrun, S. 2002. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems* 41(2–3):89–99.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33(3):374–425.

Dresner, K. M., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res. (JAIR)* 31:591–695.

Erdem, E., and Wong, M. D. F. 2004. Rectilinear steiner tree construction using answer set programming. In *Proc. of ICLP*, 386–399.

Erdogan, S. T., and Lifschitz, V. 2004. Definitions in answer set programming. In *Proc. of LPNMR*, 114–126.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*, 260–265.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Jansen, R., and Sturtevant, N. 2008. A new approach to cooperative pathfinding. In *Proc. of AAMAS*, 1401–1404.

Jennings, J.; Whelan, G.; and Evans, W. 1997. Cooperative search and rescue with a team of mobile robots. In *Proc. of ICAR*, 193–200.

Khorshid, M. M.; Holte, R. C.; and Sturtevant, N. R. 2011. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Proc. of SOCS*.

Kishimoto, A., and Sturtevant, N. R. 2008. Optimized algorithms for multi-agent routing. In *Proc. of AAMAS*, 1585–1588.

Kitano, H.; Tadokoro, S.; Noda, I.; Matsubara, H.; Takahashi, T.; Shinjou, A.; and Shimada, S. 1999. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Proc. of IEEE SMC*, 739–746. IEEE Computer Society.

Lagoudakis, M. G.; Markakis, E.; Kempe, D.; Keskinocak, P.; Kleywegt, A. J.; Koenig, S.; Tovey, C. A.; Meyerson, A.; and Jain, S. 2005. Auction-based multi-robot routing. In *Proc. of Robotics: Science and Systems*, 343–350.

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

Luna, R., and Bekris, K. E. 2011. Efficient and complete centralized multi-robot path planning. In *Proc. of IROS*, 3268–3275.

Machado, A.; Ramalho, G.; Zucker, J.-D.; and Drogoul, A. 2002. Multi-agent patrolling: An empirical analysis of alternative architectures. In *Proc. of MABS*, 155–170.

McCarthy, J. 1998. Elaboration tolerance. In *Proc. of Commonsense*.

Pallottino, L.; Scordio, V. G.; Bicchi, A.; and Frazzoli, E. 2007. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics* 23(6):1170–1183.

Ratner, D., and Warmuth, M. K. 1986. Finding a shortest solution for the n $\times$ n extension of the 15-puzzle is intractable. In *Proc. of AAAI*, 168–172.

Ryan, M. 2008. Exploiting subgraph structure in multi-robot path planning. *J. Artif. Intell. Res. (JAIR)* 31:497–542.

Ryan, M. 2010. Constraint-based multi-robot path planning. In *Proc. of ICRA*, 922–928.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The increasing cost tree search for optimal multi-agent pathfinding. In *Proc. of IJCAI*, 662–667.

Silver, D. 2005. Cooperative pathfinding. In *Proc. of AIIDE*, 117–122.

Smith, B. S.; Egerstedt, M.; and Howard, A. 2009. Automatic generation of persistent formations for multi-agent networks under range constraints. *Mob. Netw. Appl.* 14(3):322–335.

Standley, T. S., and Korf, R. E. 2011. Complete algorithms for cooperative pathfinding problems. In *Proc. of IJCAI*, 668–673.

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proc. of AAAI*.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*.

Surynek, P. 2009. An application of pebble motion on graphs to abstract multi-robot path planning. In *Proc. of ICTAI*, 151–158.

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *Proc. of AAAI*.

Svestka, P., and Overmars, M. H. 1998. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems* 23(3):125–152.

Tomlin, C.; Pappas, G. J.; Member, S.; Member, S.; and Sastry, S. 1998. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *IEEE Transactions on Automatic Control* 43:509–521.

Wang, K.-H. C., and Botea, A. 2008. Fast and memory-efficient multi-agent pathfinding. In *Proc. of ICAPS*, 380–387.

Wang, K.-H. C., and Botea, A. 2011. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *J. Artif. Intell. Res. (JAIR)* 42:55–90.

Xu, L., and Stentz, A. 2011. An efficient algorithm for environmental coverage with multiple robots. In *Proc. of ICRA*, 4950–4955.

Yu, J., and LaValle, S. M. 2013. Planning optimal paths for multi robots on graphs. In *Proc. of ICRA*.