Multi-Agent Path Planning with Constraints on Visitation Order

AAAI 2015 Submission 2327

Abstract

We consider path planning problems with multiple cooperating agents, each with its own start and goal position, the optimization target being the last time of arrival. The agents are coupled together by constraints on the order in which certain regions are visited. Such constraints can encode constructive interactions, such as when agents need to complete actions like opening doors for one another, as well as destructive interactions, where one action precludes another thereafter. We show how this class of constraints can describe a variety of practical domains in maintenance, surveillance, and assembly. Next we show that, unlike classical graph search, visitation order constrained planning is NP-hard, regardless of approximation factor. Nonetheless, we present a planner VOA* which guarantees completeness with an approximation factor equal to the number of agents. Its performance is optimized using a trie-like data structure. Finally, we evaluate the planner's performance experimentally.

Introduction

The simplest motion planning domains, where the only state is an agent's position, reduce to explicit shortest-paths problems on graphs. On the other hand, high-level domain representations such as STRIPS lead to exponential-size graphs, which are NP-hard in general and demand different techniques to incorporate the high-level information encoding the graph's structure. Here we are interested in a middle ground, often occupied by multi-task and/or cooperative multi-agent planning problems. The agents still move between locations on an explicit graph, but some aspect of their history is relevant toward determining which future moves are allowed. If multiple agents plan jointly, they may help or hinder one another.

A simple and natural example is that of robot A pressing a switch that opens a door which robot B must pass through. In this example, imagine either that robot B is incapable of pressing the switch due to physical limitations, or that the switch happens to lie conveniently on robot A's path so that the collaboration results in faster plan execution. Thus, the N-agent problem cannot simply be solved by N independent planners running in parallel. Nor would be want to search with a single planner over the joint state space of all N agents, as that would result in exponential blowup even when there is a small constant number of constraints.

The class of problems we consider consist of a separate start and goal position for each agent, along with constraints on the visitation order of certain sets of nodes. For instance, to describe a door that needs to be opened, we will have a set of nodes corresponding to the trigger which opens the door, and a second set corresponding to the door itself. The constraint simply states that the second set cannot be visited at an earlier point in time than the first. Other types of constraints can be described similarly: for instance, we may require that the second set is visited after the last time the first set is visited, as might be the case if the first set corresponds to using a coffee machine which might be cleaned up after its final use by the end of the day.

As the number of constraints grows, exponential blowup seems unavoidable in the worst case, as we will show that even obtaining a bounded suboptimal solution is NP-hard. Nonetheless, our algorithm VOA* easily handles small instances, and is also efficient on a class of instances where it is clear from the graph topology that the constraints can only interact in a few specific orders.

Related Works (still kind of sloppy, revise)

Multi-agent planning is a growing field of research, as the emergent complexity of multi-agent systems raises the need for more advanced AI in both cooperative and competitive applications (Van der Hoek and Wooldridge 2008). Previous approaches for combining multiple individual plans into one consistent multi-agent plan include (Georgeff 1988) and (De Weerdt, Ter Mors, and Witteveen 2005). Our approach is different in that, by restricting the allowable constraints to a set that we very often find in practice, we are able to use simpler graph search methods in place of logical methods or complex representations such as STRIPS.

We take the well-studied A* planning method (Hart, Nilsson, and Raphael 1968), and extend it to multi-agent systems which interact via a set of constraints. To our knowledge, the particular class of constraints we consider is new to the field of multi-agent path planning. It is essentially a very restricted subset of temporal logic (Gabbay et al. 1994) in which every variable is monotonic: either constant, starts false and at some point shifts to becoming true thereafter, or starts true and shifts to becoming false.

Different constraint formulations have been explored in the context of multi-agent planning. CBS, MA* and variants (Ferner, Wagner, and Choset 2013)(Sharon et al. 2015) adapt the dimensionality of the search to depend only on the number of agents which are actually in conflict. Our approach, in constrast, does all the low-level planning independently, but at the cost of a recombination step to merge the resulting plans. Perhaps the closest prior domain to ours is one on which the DPC algorithm (Bhattacharya, Likhachev, and Kumar 2010) was applied. Here, multiple tasks need to be distributed among robots with time-parametrized constraints on their maximum distance. Our ordering constraints generalize the task distribution problem, but on the other hand we drop the distance constraints, lending our problem to more discrete methods.

Problem Formulation

A visitation order (VO)-constrained planning problem with N agents and K constraints is formulated in terms of N weighted directed graphs $\{V_i, E_i, start_i, goal_i\}_{i=1}^N$ and K **VO constraints** $\{\phi_i\}_{i=1}^K$. The vertex set V_i contains the discrete positions accessible to agent i, including the special start and end positions $start_i, goal_i \in V_i$. Each edge $(u, v, w) \in E_i \subset V_i \times V_i \times [0, \infty)$ connects u to v; it takes at least w time units to traverse, but slower travel is permitted e.g. if one agent is pausing to wait for another. When such an edge exists, we may abuse notation and write $c(u, v) = w$. Otherwise, $c(u, v) = \infty$. We say the problem is **undirected** if $c(u, v) = c(v, u)$ for all u, v. Otherwise, it is **directed**.

A **plan** for agent i is a sequence of ordered pairs $\pi_i = \langle (v_0, t_0), (v_1, t_1), \ldots, (v_m, t_m) \rangle$ such that $v_0 = start_i$, $v_m = goal_i$, $t_0 \geq 0$, and for all j: $v_j \in V_i$ and $t_j \geq t_{j-1} + c(v_{j-1}, v_j)$. We say that π_i **visits** v_j at **time** t_j and has **arrival time** t_m. An assignment of plans $\{\pi_i\}_{i=1}^N$ to all N agents is called a **joint plan**, and is **valid** if it **satisfies** all K of the VO constraints. The **cost** of a joint plan is the maximum arrival time of its component plans. A planner's objective is to compute a valid joint plan of low cost. If the cost is at most $\epsilon \geq 1$ times that of a minimum-cost valid joint plan, then we say the computed plan is ϵ-optimal.

VO constraints impose certain order relations between the relative visit times of pairs of regions. They come in four basic types:

- Openable door: A door corresponds to some region (set of vertices) in one or more of the graphs. The door starts closed, but can be opened by visiting an associated region corresponding to the door's trigger. Once opened, the door region will become traversible.
- Closable door: This door starts open, but permanently closes the first time an agent visits the trigger region. Vertices cannot be visited while they contain closed doors.
- Restore: Some region corresponds to a restore operation, such as cleaning a coffee machine. Another region would then correspond to using the coffee machine. Any time the machine is used, it must then later be restored. Equivalently, the final use must take place no later than the final restore.
- Sequence: Some visit of the first region must precede some visit of the second region. Imagine for instance that one agent must send a piece of information, e.g. an email, to another. The email may be sent multiple times, and the inbox may be checked multiple times, but the crucial point is that the receiving agent must finally see the email.

More formally, a VO constraint $\phi = \langle \phi^-, \phi^+, type \rangle$ is described by two vertex sets $\phi^-, \phi^+ \subset \cup_i V_i$, along with a **constraint type** describing the relationship ϕ imposes on the relative visit times of ϕ^- and ϕ^+. To understand the constraint types, fix a joint plan $\{\pi_i\}_{i=1}^N$. For any vertex set $S \subset \cup_i V_i$, let $t_{min}(S)$ denote the very first time at which any π_i visits any of the vertices in S. Likewise, let $t_{max}(S)$ denote the last time S is visited. The four constraint types are defined by the following inequalities:

- O (open) constraint: $t_{\min}(\phi^-) \leq t_{\min}(\phi^+)$
- C (close) constraint: $t_{\max}(\phi^-) \leq t_{\min}(\phi^+)$
- R (restore) constraint: $t_{\max}(\phi^-) \leq t_{\max}(\phi^+)$
- S (sequence) constraint: $t_{\min}(\phi^-) \leq t_{\max}(\phi^+)$

When the corresponding inequality is met, we say that $\{\pi_i\}_{i=1}^N$ satisfies ϕ. Intuitively, an O constraint corresponds to a set of doors ϕ^+, which start closed but can be permanently opened by triggering any one of the switches in ϕ^-. In a C constraint, it is the ϕ^- vertices which should be thought of as doors, which are open until permanently barred shut by triggering a switch in ϕ^+. In an R constraint, visiting ϕ^- is like using an appliance; after the final use, ϕ^+ must be visited at least once in order to "clean up". Finally, the S constraint simply dictates that some instance of ϕ^- take place before some instance of ϕ^+.

Notice that if neither of ϕ^+, ϕ^- are ever visited, then by the usual definition of $\min, \max$ of empty sets as $\infty, -\infty$ respectively, ϕ is satisfied iff it is not of type S. For $v \in \cup_i V_i$, we will use the notation $v^+ = \{\phi \mid v \in \phi^+\}$ to refer to the set of constraints ϕ such that $v \in \phi^+$, and likewise $v^- = \{\phi \mid v \in \phi^-\}$. To prevent technical ambiguities, we further require that for each v, either v^+ or v^- is empty. In other words, $(\cup_{i=1}^K \phi_i^+) \cap (\cup_{i=1}^K \phi_i^-) = \emptyset$.

Later in this section, we will show how to convert R and S constraints into O and C constraints. The following theorem then states that without loss of generality, every joint plan can be expressed in terms of its component plans with all timestamps removed. This fact will prove helpful when we want to have agents plan independently, without knowing when and for how long they'll have to wait for one another.

Theorem 1. *Suppose all constraints are of type O or C, and let $\{\pi_i\}_{i=1}^N$ be a joint plan with the visit times ommitted. Then there exists a greedy algorithm to check whether some assignment of visit times exists which would make the joint plan feasible and, if so, it will find one with minimum cost.*

Proof. (Sketch): We propose the greedy algorithm which moves the agents in parallel along their respective plans at maximum speed, taking time equal to the edge costs when possible, but pausing when necessary to respect a constraint. If, while following a fixed path, we come across a door and it's open, clearly nothing is lost by passing through it at the earliest opportunity. Likewise if we see a trigger that opens a door, it's best to press it as soon as possible. What if we come across a trigger that would close doors? In this case, we should wait until all corresponding doors that will be encountered by the other agents have been passed. After that, we may proceed. Thus, we never get stuck as a result of a bad choice. If the algorithm does get stuck, failing to take every agent to its goal, then the joint plan cannot be made valid. On the other hand, if every agent reaches its goal, this algorithm obtains the smallest possible visit times. $\square$

Of course, the conclusion also holds if all constraints are of type C or R, by simply reversing time and tracing paths from goal to start. Next, we show how to eliminate the additional constraint types with only a constant-factor blowup in the size of the problem description. In order to describe the requisite constructions, we develop an additional piece of notation:

Theorem 2. *Any VO-constrained planning problem can be reduced to one involving only O and C constraints, such that every plan in one problem can be efficiently transformed into an equal-cost plan in the other.*

Proof. First, we sketch how to get rid of all R constraints. For clarity, we refer to the coffee machine use/clean terminology: v **uses** ϕ if $v \in \phi^-$ and $type(\phi) = R$; v **cleans** ϕ if $v \in \phi^+$ and $type(\phi) = R$. First, give each agent a new start node $start'_i$ which cleans all R constraints. Place a zero-cost directed edge from $start'_i$ to the original start $start_i$. Now, from every pair v, ϕ such that v cleans ϕ, draw bidirectional 0-cost edges from v to a new vertex v', and from v' to a new vertex v''. Add a C constraint that closes access to ϕ^- upon access to v', and an O constraint that opens access to some $goal_i$ upon access to v''. Accessing v' and v'' is tantamount to deciding on the final clean for the coffee machine ϕ, after which further use of ϕ is forbidden.

Finally to get rid of all S constraints, consider each pair v, ϕ such that $v \in \phi^+$ and $type(\phi) = S$. Connect v bidirectionally to new vertices v' followed by v'' as before. Now, add an O constraint which opens v' whenever ϕ^- is visited, and another O constraint to open the goal when v'' is visited. It's easy to check that this encoding has the desired effect. $\square$

Since R and S constraints can be represented in terms of O and C, we consider the former as the only types without loss of generality. This allows us to use the greedy algorithm according to Theorem [X].

Example Applications

In distributed assembly, it is often necessary for one agent to complete its task before another agent can begin the next step. This is easily encoded by a type O constraint. In maintenance, a type R constraint can indicate that certain restoration tasks must follow equipment use in order to return everything to its original state.

Finally, some interesting encodings are possible in security and surveillance. If, say, k agents must scan the same object, this can be encoded by giving each agent k copies of the vertex where the scanning takes place, each opening one of k O doors blocking the goal. Taking one of the copies blocks the agent' s remaining $k-1$ copies using an O constraint, thus preventing the same agent from being counted twice. We can also require, for instance, that one agent of each type scan the item, by simply giving different agents control over different "doors".

Hardness Results

In unconstrainted single-agent graph planning, the main difficulty stems from the graph being extremely large, often too large even to store in memory. If the graph were reasonably compact, say with only a million edges, then Dijkstra's algorithm finds the optimal path very efficiently. We might hope that something similar is true in the presence of VO constraints. However, we show here that unless P = NP, the complexity of VO-constrained planning is superpolynomial in K, the number of constraints. This is the case even if there is only one agent, unit edge costs, and we allow an arbitrarily large constant approximation factor. Thus, we may only hope to solve instances with few constraints in the worst case, and more complex instances if we are able to use the structure of realistic problem instances in a particular domain.

Theorem 3. *Determining the existence of a VO-constrained plan is NP-complete, even in the restricted case where there is only one agent and at most one of the following conditions holds:*

- *All constraints have type O.*
- *The graph is undirected.*

Proof. First, we reduce 3-SAT to the directed version of the problem with only O constraints. Let's say there are n variables $\{x_i\}_{i=1}^n$ and m clauses $\{x_{a(i,1)} \vee x_{a(i,2)} \vee x_{a(i,3)}\}_{i=1}^m$. In addition to $start$ and $goal$, create the vertices $\{u_i\}_{i=0}^m$, $\{v_{i,1}, v_{i,2}, v_{i,3}\}_{i=1}^m$, as well as a vertex for each variable and its negation. Draw the edges $(start, x_1)$, $(start, \neg x_1)$, (x_i, x_{i+1}), $(x_i, \neg x_{i+1})$, $(\neg x_i, x_{i+1})$, $(\neg x_i, \neg x_{i+1})$, (x_n, u_0), $(\neg x_n, u_0)$, $(u_{i-1}, v_{i,j})$, $(v_{i,j}, u_i)$, $(u_m, goal)$. Finally, make O constraints so that $x_{a(i,j)}$ is needed to open $v_{i,j}$. Notice that u_i is reachable from u_{i-1} iff one of the variables from the i'th clause was triggered. Furthermore, it is impossible to trigger both a variable and its negation.

To handle the undirected case, follow a similar construction but use C constraints to block backtracking once a variable or its negation has been triggered. This effectively directs the graph as before. $\square$

The above construction can be modified to have $|\phi^+| = |\phi^-| = 1$ and $|v^+ \cup v^-| \le 1$: simply replace the vertex corresponding to x_i with a sequence of trigger nodes on a linear path, one for each clause containing x_i. Thus, even isolated constraints suffice to yield NP-completeness.

On the other hand, if only O constraints appear AND the graph is undirected, then greedily exploring the graph in flood-fill fashion will always lead to a solution if one exists. Nonetheless, we may examine hardness of approximation. By considering the case where a set of randomly scattered triggered must all be pressed to open the goal, we can view this problem as a generalization of the Metric Traveling Salesman Problem. Since the latter admits polynomial-time constant-factor approximations (Christofides 1976), we might expect the same to hold here. The following theorem trashes such hopes.

Theorem 4. *Fix $\alpha < 1$. It is NP-hard to find an $\alpha ln(K)$-optimal VO-constrained plan, even when ALL of the following conditions hold:*

- *There is only $N = 1$ agent.*
- *All edges have zero or unit cost.*
- *All constraints have type O.*
- *The graph is undirected.*
- $|\phi^+| = |\phi^-| = 1$ *for all constraints ϕ.*
- $|v^+ \cup v^-| \leq 1$ *for all vertices v.*

Proof. This follows from the inapproximability of SET-COVER (Moshkovitz 2012). Let $\{S_i\}_{i=1}^n$ be a collection of sets. Draw a 0-main path of cost 0 from $start$ to $goal$, blocked by doors corresponding to all the elements of $\cup_{i=1}^n S_i$. For each set S_i, draw an edge of cost 1 connecting the main path to a component corresponding to all the elements of S_i, all edges within the component having cost 0. The vertices of this component are triggers which open the doors on the main path that correspond to elements of S_i. Thus, it costs 1 time time unit to open the doors corresponding to the elements of each set S_i. □

Approach

Part of the difficulty in planning with these temporal constraints is that the set of available actions does not depend solely on the current position: it also depends on past positions visited by the agent and its collaborators. Perhaps the most obvious approach, then, is to plan over the joint state space. That is, do an A* search over the graph whose vertices correspond to N-tuples as well as a history bitmask documenting which of the sets $\phi^\pm$ have been visited. If we search about V positions per agent, and there are K constraints, then naively this corresponds to a complexity of about $V^N 4^K$, or $V^N 2^K$ once we notice that only ϕ^- visits need be remembered for O constraints and ϕ^+ visits for C constraints. Due to the above hardness results, we are willing to tolerate the 2^K term in hopes of K being small. However, V^N is certainly not acceptable, as V must be quite large if we want to search graphs of substantial size! While we don't expect such a planner to fully explore the Cartesian product of all the agents' configuration spaces, this still allows far too much repeat work on each individual agent's states.

Another rather naive approach would be to greedily flood-fill positions which are not occuped by any C constraints, since O's correspond to doors which start closed, and opening them can never hurt completeness. While this deals with O's with low computational effort, the resulting paths may do substantial wasted work, and so we any semblance of an A* optimality guarantee. Indeed, Theorem [X] shows that we cannot hope for a reasonable suboptimality bound from an efficient greedy approach. Worse, C's cannot be processed greedily. We could choose to close a door or not, nondeterministically, i.e. try one approach, recurse, and then backtrack if it fails. Or we could branch in such cases, exploring both possibilities in a manner that trades dynamically between breadth and depth like A*. If we also treat the O's this way in place of the greedy approach, a bounded suboptimal algorithm starts to take form.

Therefore, we opt for an approach where each agent runs an independent planner, and then the results are joined according to Theorem [X]. Since individual plans may have to be interleaved in unpredictable ways, we save not only a bitmask but a sequence documenting the order in which constraints were triggered en route to the state under consideration. Indeed, we plan over a new notion of "state" which contains both a position and a sequence encoding the relevant history. The complexity of this representation is $K!N$ per agent, or $K!VN$ overall. Since VN is the total size of the configuration spaces, this is essentially linear when the number of constraints is sufficiently small. For convenient, we search backward from the goal. Thus, every time agent i's planner expands a state $(pos, hist)$ such that $pos = start_i$, the corresponding path is compared with paths from the other agents in the greedy manner described above. Unfortunately, this does present a bottleneck in that we're potentially trying up to $(K!)^N$ path tuples, where each agent may try any sequence of constraints.

Analysis (needs rewriting)

$g_{s_1,\ldots,s_N}(s_i)$ is the minimum time by which agent i can reach state s_i, assuming every agent j will go to s_j along some known path (following $bp(s_j)$, say) and wait there forever. Note that the state space for each agent is the Cartesian product of its graph's vertex set, and the set of partial permutations of constraints.

Define $f(s_1, \ldots, s_N) = \max_i g_{s_1,\ldots,s_N}(s_i) + \epsilon h(s_i)$. This key is impractical to compute over all possible tuples. Further approximations will need to be made. Note that $(s_1, \ldots, s_N)$ range over the goal nodes as well as the $OPEN$ list.

Even if we could compute f, does it guarantee $N\epsilon$ optimality? To prove that, it would suffice to show that we only expand nodes whose individual-agent g-values are ϵ-optimal. However, the logic fails because passing through a door-opening node can result in a sudden drastic decrease in the $g_{s_1,\ldots,s_N}$ value, due to constraints being lifted. To remedy this, instead of defining g by "wait-forever" semantics, we could treat doors as being opened after the partial paths to $(s_1, \ldots, s_N)$, at some lower bound time.

A lower-bound for $f(s_1, \ldots, s_n)$ could be computed as $f(s_j)$ by building a coarse search tree at the level of constraint sequences, where a macro-expansion consists of searching toward each possible next constraint node.

Want to prove that $g(s)$ is ϵ-optimal at expansion (when its jointPriority value is minimal), and thus the joint paths are $N\epsilon$ optimal. Suppose $g(s) > \epsilon g^*(s)$. As induction hypothesis, let $s' \in OPEN$ be such that $g(s') \leq \epsilon g^*(s')$. Fix a priority-minimizing tuple $(s_1, \dots, s, \dots, s_n)$ for s. Now consider the tuple $(s_1, \dots, s', \dots, s_n)$. If its priority were less, it would follow that s' will be expanded before s. But, path to s might open doors much sooner than path to s'!

Pending further insights, it seems the state is forced to remember sequences of contrained nodes instead of merely sequences of constraints. This way, we guarantee prefix-suboptimality, and as a bonus get global ϵ-suboptimality without the extra factor of N.

Theorem 5 (Completeness with Bounded Suboptimality). *VOA* terminates, and returns a valid $N\epsilon$-optimal joint plan, if one exists.*

Proof. Let g^* be the optimal valid joint plan cost, and g_i^* be the optimal distance to the corresponding goal history state, looking only at agent i. To see that termination is guaranteed, note that since each state is expanded at most once, eventually, each agent expands its optimal goal state. As in weighted A*, the goal is expanded with cost at most ϵg_i^*, so the greedy algorithm will recover this joint plan with cost not exceeding $\epsilon \sum_i g_i^* \leq \epsilon \sum_i g^* = N\epsilon g^*$. Therefore, the termination condition holds.

Now, assuming the termination condition, we want to bound the optimality factor on our plan. Based on the above, we know that at least one of the agents has not expanded its optimal goal state s. Let s' be the earliest state in the open list on the optimal path from $start$ to s. Then $f(s') \leq \epsilon g_i^*$. But $jointCost \leq N \min_{i=1}^N f_{min,i}$, so $jointCost \leq N\epsilon g_i^* \leq N\epsilon g^*$. □

Trie Stuff

A **state** is the search space of agent i is an ordered pair (v, seq) of vertex $v \in V_i$ and history sequence seq. States are accessed by following a trie according to the sequence of constraints named in seq. Then, v is reached simply by looking it up in a map stored in seq's trie node.

Algorithm 1 $search()$

```
$jointCost = \infty$
while $N \min_{i=1}^N f_{min,i} < jointCost$ do
    choose $i \in \{1, \dots, N\}$ such that $OPEN_i$ is not empty
    $(pos, hist_i)$ := remove from the front of $OPEN_i$
    if $pos = start_i$ then
        for all tuples $(hist_1, \dots, hist_{i-1}, hist_{i+1}, \dots, hist_N)$
        do
            $jointCost := \min(jointCost, jointPriority($
                $(start_i, hist_i), \dots, (start_N, hist_N)))$
        end for
    end if
    $expand(pos, hist)$
end while
```

Algorithm 2 $expand(pos, hist)$

```
mark $(pos, hist)$ as CLOSED
for all $(pos', hist') \in successors(pos, hist)$ do
    if $g(pos', hist') > g(pos, hist) + c(pos, pos')$ then
        $g(pos', hist') := g(pos, hist) + c(pos, pos')$
        $bp(pos', hist') := (pos, hist)$
        if $(pos', hist')$ is not CLOSED then
            insert $(pos', hist')$ in $agent.OPEN$
        end if
    end if
end for
```

Algorithm 3 $successors(u, seq_u)$

```
succlist := $\emptyset$
for all $v$ such that $c(u, v) < \infty$ do
    $seq_v := seq_u$
    for all min constraints $\phi^{\pm} \notin seq_u$ with $v \in \phi^{\pm}$ do
        append $\phi^{\pm}$ to $seq_v$
    end for
    for all max constraints $\phi^{\pm}$ with $v \in \phi^{\pm}$ do
        remove $\phi^{\pm}$ from $seq_v$
        append $\phi^{\pm}$ to $seq_v$
    end for
    push $(v, seq_v)$ onto succlist
end for
return succlist
```

References

Bhattacharya, S.; Likhachev, M.; and Kumar, V. 2010. Multi-agent path planning with multiple tasks and distance constraints. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 953–959. IEEE.

Christofides, N. 1976. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document.

De Weerdt, M.; Ter Mors, A.; and Witteveen, C. 2005. Multi-agent planning: An introduction to planning and coordination. In *In: Handouts of the European Agent Summer*. Citeseer.

Ferner, C.; Wagner, G.; and Choset, H. 2013. Odrm* optimal multirobot path planning in low dimensional search spaces. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 3854–3859. IEEE.

Gabbay, D. M.; Hodkinson, I.; Reynolds, M.; and Finger, M. 1994. *Temporal logic: mathematical foundations and computational aspects*, volume 1. Clarendon Press Oxford.

Georgeff, M. 1988. Communication and interaction in multi-agent planning. *Readings in distributed artificial intelligence* 313:125–129.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Moshkovitz, D. 2012. The projection games conjecture and the np-hardness of ln n-approximating set-cover. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer. 276–287.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015.

Algorithm 4 $greedyCost(\pi_1, \ldots, \pi_n)$

```
$\forall\phi$ : $trigger(\phi) := type(\phi) = O$ **or** $\phi^-$ is absent in $\pi_{1\ldots N}$
$\forall 1 \in 1 \ldots N$ : $u_i := v_i := start_i$
$events := \{0 : \{1 \ldots N\}\}$
**while** $events$ is not empty **do**
    $(time, ids) := extractMin(events)$
    **for all** $i \in ids$ **do**
        $u_i := v_i$
        **for all** $\phi \in v^-$ **do**
            **if** $type(\phi) = O$ **or** $\phi^-$ appears in no suffix of $\pi_{1\ldots N}$ after $u_{1\ldots N}$ **then**
                $trigger(\phi) := true$
            **end if**
        **end for**
    **end for**
    **for all** $i \in 1 \ldots N$ with $u_i = v_i \neq goal_i$ **do**
        $v_i :=$ successor of $u_i$ on $\pi_i$
        **for all** $\phi \in v^+$ **do**
            **if** $\neg trigger(\phi)$ **then**
                $v_i := u_i$ and continue with the next $i$
            **end if**
        **end for**
        $events(time + c(u_i, v_i)).insert(i)$
    **end for**
**end while**
**if** $u_{1\ldots N} = goal_{1\ldots N}$ **then**
    **return** $time$
**else**
    **return** $\infty$
**end if**
```

Algorithm 5 $jointPriority((pos_1, hist_1), \ldots, (pos_N, hist_N))$

```
$\forall i$, build $\pi_i$ by following $bp()$ from $(pos_i, hist_i)$
**for all** $i \in 1 \ldots N$ such that $pos_i \neq start_i$ **do**
    extend $\pi_i$ with a zero cost edge to a node that opens all doors the agent can open, followed by $start_i$ at cost $\epsilon h(s_i)$
**end for**
**return** $greedyCost(\pi_1, \ldots, \pi_n)$
```

Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Van der Hoek, W., and Wooldridge, M. 2008. Multi-agent systems. *Foundations of Artificial Intelligence* 3:887–928.