# Multi-Agent Path Planning with Constraints on Visitation Order

**AAAI 2016 Submission 2663**


## 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, where one agent enables the action of another, as well as destructive interactions, where one action precludes another thereafter. We describe 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 to approximate. We present two planners, Fusion VOCA* and Greedy VOCA*. While the former guarantees completeness and a suboptimality factor equal to the number of agents, the latter is very efficient but may fail to yield a solution. Both planners augment the state space with a sequence of previously visited constraints, and this space is organized into a trie-like data structure. Finally, we compare the two planners experimentally, testing their scalability in terms of the number of constraints, agents and states.


## Introduction

The simplest motion planning domains, where the state is fully described by an agent's position in space, reduce to explicit shortest-paths problems on graphs upon discretization. On the other hand, high-level domain representations such as STRIPS compactly represent exponential-size graphs, which are NP-hard to search in general, and demand different techniques to incorporate the high-level information encoding the graph's structure. We explore a middle ground, often occupied by multi-task and/or cooperative multi-agent planning scenarios. 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 example is that of a 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 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 with a constant number of constraints.

We allow each agent to have a separate start and goal position, and specify constraints on the visitation order of certain sets of nodes. For instance, to describe a door that needs to be opened, we associate a set of nodes with the door-opening trigger and another set with the door itself, and insist that the trigger set is visited before the door. 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 needs to 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, the two VOCA* algorithms presented in this paper can efficiently handle a large fraction of randomly generated test cases, provided either they are small or it is clear from the graph topology that the constraints can only interact in a few specific orders.

## Related Works

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 set of allowable constraints, we are able to use simpler graph search methods in place of logical methods or complex representations such as STRIPS.

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. The DPC algorithm (Bhattacharya, Likhachev, and Kumar 2010) also studied certain types of constraints between multiple agents. 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.

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. Some of the work most closely related to our own occured in the context of a generalized vehicle routing problem (Karaman and Frazzoli 2011), for which techniques based on language automata and mixed-integer linear programming were developed. We focus on a more restricted class of problems, using classic graph search techniques and data structures.

## 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 $\pi_i$ **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 among its component plans. A planner's objective is to compute a valid joint plan of low cost; or in other words, to minimize the time at which all agents have reached their goals. A joint plan costing at most $\epsilon \geq 1$ times that of a minimum-cost valid joint plan is said to be **$\epsilon$-optimal**.

VO constraints impose certain order relations between the relative visit times of **regions** (sets of vertices) by agents. They come in four basic types:

- Door to Open: A "door" corresponds to some region 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". Thus, the door region is traversible only after visiting the trigger region.
- Door to Close: The door starts open, but is permanently "closed" the first time an agent visits the trigger region. Thus, vertices in the door region can only be traversed before visiting the trigger region.
- Use/Restore: If a "use" region corresponds to using, say, a coffee machine, a "restore" operation corresponds to cleaning it. The machine may be used and cleaned any number of times, but all uses must eventually be followed by a restore. Equivalently, no use can take place after the final restore.
- Sequence: Two regions must be visited, and the first visit of region 1 must precede the last visit of region 2. 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 requirement is that the receiving agent must finally see the email.

More formally, a VO constraint $\phi = \langle \phi^-, \phi^+, \tau \rangle$ is described by two regions $\phi^-, \phi^+ \subset \cup_i V_i$, along with a **constraint type** $\tau(\phi)$ describing the relationship $\phi$ imposes on the relative visit times of $\phi^-$ and $\phi^+$. 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 $\pi_i$ 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 $\phi$. These definitions conrrespond to the preceding intuitions: looking at O constraints, for instance, $\phi^+$ is the door region while $\phi^-$ is the trigger.

Notice that if neither of $\phi^+, \phi^-$ are ever visited, then by the usual definition of $\min, \max$ of empty sets as $\infty, -\infty$ respectively, $\phi$ is satisfied iff $\tau(\phi) \neq S$. For $v \in \cup_i V_i$, we will use the notations $v^+ = \{\phi \mid v \in \phi^+\}$ and $v^- = \{\phi \mid v \in \phi^-\}$ to refer to constraints associated with a particular vertex. To prevent technical ambiguities, we further require that for each $v$, at least one of $v^+$ or $v^-$ is empty. Equivalently, $(\cup_{i=1}^K \phi_i^+) \cap (\cup_{i=1}^K \phi_i^-) = \emptyset$.

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 allow much of the planning to be done separately by agent, 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 an 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 a door? In this case, we should wait until all agents' encounters with the relevant door have 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 arrival times. □

Next, we show how to eliminate the redundant constraint types with only a constant-factor blowup in the size of the problem description.

**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 appliance use/restore terminology: $v$ **uses** $\phi$ if $\tau(\phi) = R$ and $v \in \phi^-$; $v$ **restores** $\phi$ if $\tau(\phi) = R$ and $v \in \phi^+$ . First, give each agent a new start node $start'_i$ which restores all R constraints. Place a zero-cost directed edge from $start'_i$ to the original start $start_i$. Now, from every pair $v, \phi$ such that $v$ restores $\phi$, draw bidirectional zero-cost edges from $v$ to a new vertex $v'$, and from $v'$ to a new vertex $v''$. Add a C constraint that blocks access to $\phi^-$ upon accessing $v'$, and an O constraint that blocks access to some $goal_i$ until $v''$ is accessed. Accessing $v'$ and $v''$ is tantamount to deciding on the final restore on $\phi$, after which further use of $\phi$ is forbidden.

Finally to get rid of all S constraints, consider each pair $v, \phi$ such that $\tau(\phi) = S$ and $v \in \phi^+$. Connect $v$ bidirectionally to new vertices $v'$ followed by $v''$ as before. Now, add an O constraint which opens $v'$ whenever $\phi^-$ 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. □

Since R and S constraints can be represented in terms of O and C, we consider the latter pair as the only types without loss of generality. This allows us to use the greedy algorithm from Theorem 1.

## A Sampling of Real-World Interpretations

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.

More 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 of which opens one of $k$ O doors blocking the goal. Taking one of the copies activates a C trigger, blocking access to the agent's remaining $k-1$ copies, thus preventing the same agent from being counted twice. We can also require, for instance, that one agent from each of several assigned groups scan the item, by giving different agents control over different "doors".

In a military or video game application similar to the main example in (Karaman and Frazzoli 2011), one may be presented with a choice of routes to take, but also a series of enemy bases which block all routes within their respective ranges. Each base effectively blocks its neighboring routes via a C constraint. Thus, it is necessary to destroy or disable some of the bases before taking a route.

## 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. In the worst case, we may only solve instances with few constraints; for larger instances, we can only hope to take advantage of some structure or regularity in the realistic problem instances of 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. □

The above construction can be modified to have $|\phi^+| = |\phi^-| = 1$ and $|v^+ \cup v^-| \leq 1$: 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 opening doors while 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 triggers must all be pressed to open the goal, we can view VOC planning as a generalization of the Metric Traveling Salesman Problem. Since the latter admits polynomial-time constant-factor approximations (Christofides 1976), we might expect to find the same 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 $\phi$.*
- *$|v^+ \cup v^-| \leq 1$ 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 "main path" of cost 0 from $start$ to $goal$, blocked at the goal end 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 precisely the doors of the main path that correspond to elements of $S_i$. Thus, it costs 1 time time unit to open all the doors corresponding to the elements of each set $S_i$. □

## Algorithms

Part of the difficulty in planning with these temporal constraints is that the set of actions available to one agent does not depend solely on its current position: it also depends on past positions visited by the agent and its collaborators. Perhaps the most obvious approach, then, is to plan directly over the joint state space, using a method such as A*. However, this yields absurdly many states and state transitions. Therefore, we opt for an approach where each agent runs an independent planner, and then the results are joined according to Theorem 1.

### Fusion VOCA*

In order to truly account for all possible plans, an agent's "state" must include not only its present position, but also its history of interactions with every constraint. And since individual plans may have to be interleaved in unpredictable ways, we require the history to be saved as a sequence documenting the order in which constraints were triggered en route to the state under consideration. Indeed, each agent searches over an augmented graph whose vertices, called **states**, are elements of $V_i \times H$. Here, an element of $H$ is a (possibly empty) sequence of distinct regions of the form $\phi^+$ or $\phi^-$ for some constraint $\phi$. Thus, an agent's state is a combination of its present position and its history of encounters with constrained regions. In a forward search, the start state is $(start_i, \emptyset)$, and goals are any state of the form $(goal_i, hist)$ for $hist \in H$. Conversely, in backward search, the "history" sequence can be thought of as encoding planned future encounters: the search starts at $(goal_i, \emptyset)$, and ends at $(start_i, hist)$ for any $hist \in H$.

Each time a goal state $(goal_i, hist)$ is expanded, we can retrieve a plan for agent $i$ by following saved backpointers as in Dijkstra or A*. At this time, we compare the new plan with every combination of known plans from the remaining $N-1$ agents, using the method of Theorem 1. Note that the $N$ parallel searches can be interleaved in any order; our implementation simply extracts states from the $N$ priority queues in round-robin fashion.

**Theorem 5** (Completeness with Bounded Suboptimality)**.** *Fusion VOCA* (based on $w$-weighted A*) terminates, and returns a valid $Nw$-optimal joint plan, if one exists.*

*Proof.* Let $g^*$ be the optimal valid joint plan cost, and $g_i^*$ be the optimal distance in agent $i$'s graph to the goal state whose history sequence matches that of the joint optimal solution. 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 $w$-weighted A*, the goal is expanded with cost at most $wg_i^*$, so the greedy algorithm will recover this joint plan with cost not exceeding $w\sum_i g_i^* \leq w\sum_i g^* = Nwg^*$. 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 $f(s') = g(s') + wh(s') \leq w(g^*(s') + h(s')) \leq wg_i^*$. At termination, $jointCost \leq N \min_{i=1}^N f_{min,i}$, so $jointCost \leq Nwg_i^* \leq Nwg^*$. □

### Greedy VOCA*

This algorithm simply plans for each agent $1, \ldots, N$ in turn, greedily making choices and committing to them. Since these choices might not have a continuation into a valid joint plan, this approach is necessarily incomplete. On the other hand, it is fast because it only searches for one path to the goal per agent.

When planning for agent $i$, the algorithm must maintain its commitments to the plans it found for agents $1, \ldots, i-1$, all while optimistically assuming that agents $i+1, \ldots, N$ will pose no obstacles and conveniently open any doors reachable from their respective graphs. It does this by essentially storing, along with each state in agent $i$'s search graph, a plausible set of simultaneous positions for the agents $1, \ldots, i-1$. These positions are derived from the pre-determined paths for these agents, according to a variant of the greedy algorithm of Theorem 1. That is, agents move forward as necessary to accomodate agent $i$'s moves.

The state history that's saved is a sequence that interleaves moves from all $i$ agents.

While Greedy VOCA* makes no global guarantees, it will find a $w$-optimal path for itself (not accounting for joint waiting times), consistent with its optimistic assumption on the later agents and its commitments on prior agents. Thus, we might expect comparable path qualities provided the algorithm succeeds. Greedy VOCA* is generally fast but incomplete: to increase its success rate, we can restart it as many times as we like with a random ordering of the agents, performing the $N$ searches along this new ordering instead.

### Data Structures

A **state** is the search space of agent $i$ is an ordered pair $(v, hist)$ of vertex $v \in V_i$ and history sequence $hist \in H$. States are organized into a trie and accessed by following edges corresponding to the sequence of constraints named in $hist$. Having arrived at the right trie node for $hist$, data corresponding to the state $(v, hist)$ is retrieved by looking it up in a map stored in this node. This data structure allows for efficient traversal of the state graph without having to repeatedly compute a hash function of the entire $hist$ sequence array. Furthermore, the trie node contains stores information common to all the states with the same constraint-oriented history, such as bitmasks representing the set of activated triggers.

### Heuristic

As our A* heuristic, we simply precomputed distances to the goal in the relaxed problem, where each agent plans on its own and passes freely through all constraints.

## Experiments

### Setup

We tested both versions of VOCA* with weight $w = 1$ on random 2D mazes with equal numbers of type O and C constraints. A "maze" consists of $N$ $r$x$r$ square grids, one for each agent. The grid is 8-connected, with diagonal moves costing $\sqrt{2}$ times as much as orthogonal moves. We used a standard DFS maze generator to fill the grid with a minimally connected network of thin passages. We then generated numbered a subset of the cells by consecutive positive integers, such that every numbered cell is reachable from the start by traversing only cells with smaller numbers. The highest-numbered cell becomes the goal.

We want to place constraints on some cells while maintaining existence of a path to the goal. To accomplish this, we build an "abstract" multi-agent plan by generating a sequnence of constraint events of the form "agent $i$ visits $\phi^+$ (or $\phi^-$)" in which $\phi^+$ and $\phi^-$ each occur once for each constraint $\phi$. The sequence is generated from beginning to end, at each step choosing randomly among the options that are consistent with the choices thus far. If the subsequence corresponding to a given agent has length $L$, then the $i$'th entry becomes a constraint on the maze cell numbered $num(goal)/(L+1)$.

| # Constraints | Median Time | IQR | % Solved |
|---|---|---|---|
| 6 | .0049/.0045 | .0029-.0258/.0019-.0215 | 92/84 |
| 8 | .0083/.0152 | .0030-.1337/.0032-.6245 | 86/78 |
| 10 | .4075/.0239 | .0055-$\infty$/.0042-.2114 | 68/82 |
| 12 | .1473/.0941 | .0046-$\infty$/.0220-$\infty$ | 64/74 |
| 14 | $\infty$/.2889 | .2009-$\infty$/.0098-3.987 | 42/76 |
| 16 | $\infty$/.2504 | .1832-$\infty$/.0200-2.1436 | 34/82 |

Table 1: 8 agents on 25x25 mazes

| # Agents | Median Time | IQR | % Solved |
|---|---|---|---|
| 6 | .0032/.0064 | .0015-.0307-.0014-.0186 | 88/86 |
| 8 | .0083/.0152 | .0030-.1337/.0032-.6245 | 86/78 |
| 10 | .1384/.0437 | .0052-$\infty$/.0125-$\infty$ | 68/58 |
| 12 | .0812/.0652 | .0138-$\infty$/.0243-$\infty$ | 70/64 |
| 14 | 3.427/2.948 | .0715-$\infty$/.0328-$\infty$ | 52/50 |
| 16 | $\infty$/$\infty$ | .0174-$\infty$/$\infty$-$\infty$ | 44/8 |

Table 2: 8 constraints on 25x25 mazes

Now a solution is guaranteed to exist, using only the numbered cells. To maintain this property while adding additional challenge, each cell without a number, whether it was originally an empty space or a wall, has a 0.25% chance of having each possible type of constraint placed on it.

Three parameters were varied: the number of constraints, the number of agents, and the maze dimensions. We varied each parameter individually while keeping the other two fixed, resulting in Tables 1-3. The control condition is 8 constraints (4 of each type), 8 agents, and 25x25 maze. The numbers on the left side of each table cell correspond to Fusion VOCA*, while the numbers on the right correspond to Greedy VOCA*. For each setting of the parameters, 50 mazes were randomly generated. Times are measured in seconds. Median times are computed as the mean of the 25th and 26th ranking result. The Interquartile Range is also given, from the 14th to the 37th result. Note that the IQT traps the true median with probability $2^{-50}\sum_{i=14}^{36}\binom{50}{i} > 99.9\%$. Each algorithm was given a maximum of 5 seconds to solve each maze on a 3.4 GHz Intel Core i7-2600 CPU.

### Results

The costs of solutions found by Fusion and Greedy VOCA* are very similar. The key empirical differences lie in their respective speeds and success rates. Fusion VOCA* seems to be more sensitive to the number of constraints; while Greedy VOCA* is more sensitive to the number of agents. Both aresensitive maze, but the fusion algorithm is a bit more so. Perhaps it should come as a bit of a surprise that the greedy algorithm, despite its incompleteness, manages to be competitive. On the other hand, it avoids the main bottleneck of the fusion algorithm: the latter sometimes spends inordinate amount of time not on expanding states, but on checking all possible goal $N$-tuples for a possible joint intervealing. Note also that both algorithms have very high-variance runtimes, which is why we consider the median a more meangful statistic here than the mean

| Maze Size | Median Time | IQR | % Solved |
|---|---|---|---|
| 15x15 | .0019/.0013 | .0015-.0028/.0005-.0040 | 98/86 |
| 25x25 | .0083/.0152 | .0030-.1337/.0032-.6245 | 86/78 |
| 35x35 | .0477/.0766 | .0047-$\infty$/.0238-$\infty$ | 68/72 |
| 55x55 | 1.256/2.944 | .0183-$\infty$/.2206-$\infty$ | 52/50 |
| 75x75 | $\infty$-$\infty$ | $\infty$-$\infty$ | 24/26 |
| 95x95 | $\infty$-$\infty$ | $\infty$-$\infty$ | 8/18 |

Table 3: 8 constraints and 8 agents

## Future Work

We presented two algorithms for planning with multiple-agents tied by visitation order constraints. Fusion VOCA* is a rigorous approach that guaranteed completeness with bounded suboptimality. Greedy VOCA*, on the other hand, trades completeness for speed, and succeeds in a good fraction of the cases we generated. Thus, it seems advisable to try the greedy approach first, and if it fails, fall back to the fusion approach.

When there are many agents, we found that the actual planning part of Fusion VOCA* is considerably faster than the fusion step which it invokes to test every new $N$-tuple of known paths. Although the testing for a single tuple is quick and greedy, the number of such tuples can be exponential in the number of agents $N$. It would be interesting to see if the latter bottleneck could be removed while preserving completeness and bounded suboptimality. It would also be worthwhile to develop better heuristics that somehow capture the interdependence between the agents in a constrained planning scenario.

**Algorithm 1** $FusionVOCA*$

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

**Algorithm 2** $GreedyVOCA*$

**for** $i$ in $1, \ldots, N$ **do**
  **while** $OPEN_i$ is not empty **do**
    $(pos, hist_i)$ := remove from the front of $OPEN_i$
    **if** $pos = goal_i$ **then**
      commit to this path for agent $i$
      break out of while loop
    **end if**
    $expand(pos, hist)$
  **end while**
  **if** no path found for agent $i$ **then**
    randomly permute the agents and start over
  **end if**
**end for**

**Algorithm 3** $expand(pos, hist)$

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 4** $successors(u, seq_u)$

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.

**Algorithm 5** $greedyCost(\pi_1, \ldots, \pi_n)$

$\forall \phi:\ trigger(\phi) := \tau(\phi) = O$ **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** $\tau(\phi) = O$ **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 6** $jointPriority((pos_1, hist_1), \ldots, (pos_N, hist_N))$

$\forall i$, 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** $greedyCost(\pi_1, \ldots, \pi_n)$


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.

Karaman, S., and Frazzoli, E. 2011. Linear temporal logic vehicle routing with applications to multi-uav mission planning. *International Journal of Robust and Nonlinear Control* 21(12):1372–1395.

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. 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.