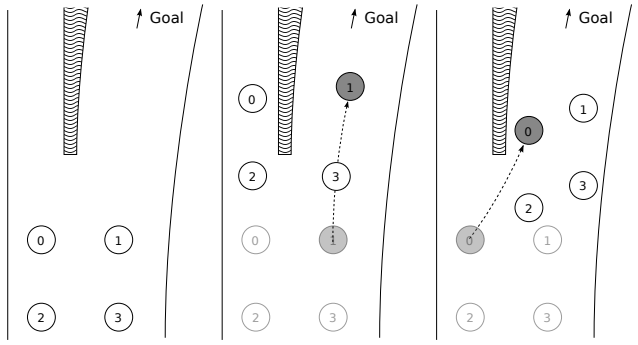
Planning for multi-agent teams with leader switching

Siddharth Swaminathan Mike Phillips Maxim Likhachev

***Abstract*— *Follow-the-leader* based approaches have been popular for the control of multi-robot teams for their ability to drive many with few. Typically, in these methods you select a single leader, generate a plan for it, while all other agents follow this leader using their individual controllers. However, there are many scenarios where this approach can lead to highly suboptimal behavior or even failure in the presence of clutter. In this work, we present a planning approach that automatically figures out when to switch leaders on the way to the goal while minimizing a given cost function that penalizes leader switching and deviations from the desired formation. To deal with the increased dimensionality of the problem we show how a recently developed algorithm, MHA* (multi-heuristic A*) can be extended to support planning for a team of robots. We also provide explicit cost minimization and guarantee that paths found are within a user- chosen factor of optimality with respect to the graph modeling the planning problem. Experimentally, we found that allowing for dynamic leader-switching leads to a significant increase in finding feasible plans for multi-robot teams ranging up to 21 robots.**



(a) Initial state (b) Leader 1 (c) Leader 0

Fig. 1. Leaders are shown in dark grey. Dotted arrow shows the path taken by the leader. In this scenario, using robot 1 as the leader cause separation of the team, while using a different leader such as robot 0 drives the whole team toward the goal.

I. INTRODUCTION

Multi-agent teams have many advantages over robots working alone. A team is more robust to individual agent failure, can allow for greater sensor range, and better communication coverage. They are therefore of great interest in the areas of disaster and emergency response, exploration tasks, and surveillance [1], [2], [3].

Motion planning is essential in cluttered environments to ensure agents always reach their goals without getting stuck and in an efficient and collision free manner. However, as teams get larger, planning for all agents quickly becomes intractable as the state space increases exponentially in the number of agents. In order to scale to controlling larger numbers of agents, *follow-the- leader* methods are used to control many agents while focusing on only a few [4].

An obvious method for planning using a single-leader is to only consider paths where there is sufficient clearance around the leader for the formation to be maintained [5]. However, this approach only works in relatively benign environments since it does not allow the team's formation to change when navigating in tight quarters and therefore is incomplete.

One way to resolve this is to allow the formation to deform (e.g, followers avoid obstacles) while following the leader. However, there are cases where the choice of leader affects whether followers get trapped or separated from the group. In Figure 1a, we see a formation heading toward a goal in the upper right. In Figure 1b, when leader 1 drives toward the goal, the group gets separated by the dividing obstacle. However, by using a different leader, such as robot 0 (Figure 1c), the planner can take advantage of formation compliance in order to keep the team together while approaching the goal. We therefore note that the choice of leader greatly impacts the success of the overall planner.

In this work we propose a planner which allows for leader switching. This allows us to avoid many of the failure cases of the single-leader method while still scaling significantly better than planning for all agents independently. This is largely because a follower that becomes trapped when following one leader can often be freed more easily by following a different leader (or by becoming the leader itself).

Our method applies heuristic search techniques which convert the motion planning problem into a graph where states in the configuration space become vertices in the graph and motions between two states become edges connecting corresponding vertices in the graph. Graph search methods can then find a path connecting the start and goal state for the team. Furthermore, the user can provide an arbitrary cost function which the planner will optimize such as minimizing distance traveled, leader switching, or deviation from the desired formation.

To deal with the increased dimensionality, we employ a recent search algorithm called MHA* which can use a collection of heuristics, each with their own strengths, in a principled manner [6]. The chosen heuristics can be arbitrary and as long as one heuristic function is admissible and consistent, the overall planner still provides bounds on worst

This research was sponsored by the ONR DR-IRIS MURI grant #N00014-09-1-1052.

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA

case solution quality. In this paper we extend MHA* to not only have several searches each with their own heuristic, but to also allow each to use a different set of actions. This is especially beneficial in our domain of multi-robot planning.

In our experimental results we show how leader switching and the use of multiple heuristics by MHA* provides significantly better planner performance. Specifically, we have observed a 5 times speedup and increase in the planner success rate from 61% to 97%.

II. RELATED WORK

Some of the most ubiquitous methods for controlling multi-agent systems are potential methods [7], [8], [9], [10]. These use attractors to pull agents toward goals or a leader they are tracking and repulsion forces which can be used to push agents away from obstacles or other agents. These methods can also be used to compel agents into formations while being non-rigid in order to avoid collisions with obstacles.

In some cases, there isn't a specific formation that must be held, but only proximity to neighbors in order to maintain connectivity. In these cases, previous work has shown the benefits of knowing when to explicitly severe connections between agents to allow a team to part around obstacles [11]. However, larger concave objects will still give these methods trouble and planning will be required to provide claims on completeness.

A common technique for handling large numbers of agents is leader-following. Leader-following is a commonly used technique as it allows many agents to be guided by only a few [12]. When using these methods the leader is driven by teleoperation or a given path (such as from some planner), and then follower agents use controllers to maintain visibility, formation, or just a distance from the leader [13], [12]. An interesting variant on leader-following is shepherding, where the aim is still to get all agents to some goal while driving the leaders, however, in this case the followers are sheep and therefore repel away from the leaders instead of moving toward them [14], [15], [16].

Another more constrained way to control many agents is by using formations. These methods generally assume a path has been provided by a motion planner for a leader or point in the formation [17], [18]. The work in [19] allows for the team of agents to change between several formations in order to navigate among different obstacles. However they still assume a path is given for the leader robot. Finally, in this recent work [20], a team of micro quadrotors uses formations while flying among obstacles. Smooth paths are generated by using mixed-integer linear programming to avoid obstacles, but the method assumes that the high-level waypoints of a plan are provided.

On the planning side of things, perhaps one of the most common multi-agent problems is to move a large number of agents to goal states. In its typical form, the agents are not moving together and the goals are often far apart. There are two common variants of this problem, those where each agent has a specific goal it must reach [21] and those where the planner is allowed to choose which goals are assigned to which robot in order to provide better solutions [22]. In contrast, the planning problem we are dealing with has a set of agents moving together as a cohesive team.

The following works are the most relevant to ours. All of them are motion planning for a team of agents with a common goal region. Ideally these agents need to move in a cohesive fashion. In [23], a PRM [24] is constructed to provide global guidance to the goal. Agents use flocking techniques to move along the path generated by the PRM toward the goal while avoiding collisions with each other. However, cohesion among the agents is weak and formations are not used. Here, the authors use a two pass algorithm [25]. First, a global path is found for a team-enclosing but deformable rectangle. The paths of the agents are found by keeping them inside the rectangle at all times using controllers that repulse from the walls of the rectangle and from the other agents. This method keeps the agents together but the planner is incomplete as the team as a whole can only deform into different sized rectangles. It also is unable to maintain formations. In this method [26], a path for a single agent is found. Then the path is widened into a corridor using the path clearance. Behavior-based methods are then used to have agents move along the corridor, avoid collisions with each other, and avoid the edges of the corridor. The method keeps the team together but does not enforce a formation. In contrast, our approach tries to hold formation but allows it to deform around obstacles to increase success rate. The planner includes this formation error into the cost function and minimizes it on a global scale.

Finally, [12] looks at the benefits of leader switching but uses it for teleoperation and not in a planning context as we do. Unlike all of the above mentioned work, this paper studies planning for multi-robot teams while explicitly reasoning when to switch leaders.

III. BACKGROUND

A. Definitions and Assumptions

The motion planning problem involves finding a continuous, collision free, and (ideally) low cost, path through configuration space, that connects the start and goal states (s_{start}, s_{goal}). A point in configuration space (called a configuration or state) represents a complete specification of all degrees of freedom relevant to the planning problem. In our case, this will be the position of each agent and information on who is currently the leader.

The algorithms presented in this paper represent the motion planning problem as a graph search. Therefore, we discretize the configuration space and convert it to a graph. $G(V, E)$ is a graph modeling the original motion planning problem. V is the set of vertices, each representing a discretized cell of configuration space and E is the set of edges that connects pairs of vertices. The edges in E represent *motion primitives*, which are short kinematically feasible motions that can be applied at any state. $c(u, v)$ is the cost of the edge from vertex u to vertex v. This cost function can be defined as the user desires. When an edge

$u \to v$ violates a constraint, such as colliding with obstacles, or when agents separate beyond tolerance, its cost $c(u, v)$ is set to infinite.

A* like methods assume a heuristic function $h(s)$ is provided which estimates the distance from vertex s to s_{goal}. The heuristic is typically computed using a relaxed version of the problem which is easier to compute, such as estimating the distance when ignoring certain constraints (for example, ignoring obstacles) or by solving a lower dimensional planning problem by dropping some of the degrees of freedom. A heuristic is said to be *admissible* if it never overestimates the remaining cost to the goal. A heuristic is *consistent* if it also satisfies the triangle inequality $h(u) \leq c(u, v) + h(v) \forall u, v \in V$.

A* works by iteratively *expanding* the state with the lowest priority $f(s)$ where $f(s) = g(s) + h(s)$ until the goal has the minimum priority. The g-value, $g(s)$ of a state maintains the lowest accumulated cost from s_{start} to s that the planner has found so far. The f-value, $f(s)$ represents the cost of a path from start to goal that passes through s. Expanding a state s means that we compute the successors of s (the states where the motion primitives of s end) update the g-values of any states for which we found a lower cost to, and then put these states into the $OPEN$ list which holds the states that A* should consider for expansion. When the heuristic is admissible, A* is guaranteed to find an optimal cost solution. If the heuristic is consistent, A* will do this without expanding any state more than once.

Weighted A* is a common variant of A* which prioritizes states according to $f(s) = g(s) + \varepsilon h(s)$ for $\varepsilon > 1$. By inflating the heuristic term the planner becomes more goal focused at the expense of cost minimization. In practice, weighted A* finds solutions significantly faster than A* (often by orders of magnitude). The algorithm is no longer optimal, however if the heuristic is admissible, it is guaranteed to find a solution with cost no larger than ε times the cost of an optimal solution [27]. If the heuristic is consistent, this guarantee holds even when not allowing re-expansions [28].

IV. ALGORITHM

The problem of planning for a multi-agent formation with leader switching consists of both continuous and discrete components. The continuous component comes from planning a path for N agents through a continuous configuration space, while the discrete component comes from finding a sequence of leaders over each step in the path. The robot formation is described by specifying desired offsets between the agents. We index the robots with the set $I = \{1, \ldots, N\}$. For brevity, we conflate the robots with their indices. We denote the set of available leaders by L and note that $L \subseteq I$.

A. Problem Representation

The planning problem is represented as finding a (close-to-optimal) path in a graph $G = (V, E)$, where V is a set of vertices in G and E is the set of its edges. At any given state $s \in V$, we represent the position $P_i(s), 1 \leq i \leq N$ of each of the agents and the index $l(s) \in L$ of the current leader. Let the configuration space of agent i be represented by C_i. Formally, we define the state space as:

$$[C_1 \times C_2 \times \ldots \times C_N \times L]$$

1) Edges: We assume that each robot i has a set of motion primitives M_i that characterizes its movement capabilities for planning. We call this the *action set*. The fully coupled action space would generate the cartesian-product of each individual agent's possible actions. This grows exponentially in the number of leaders. To make the problem scale with the number of agents, we adopt a leader-following approach. At any state s, we consider motion primitives for all leaders i. For $i \neq l(s)$, a leader switch occurs before execution of the motion. When a leader executes a motion primitive, the rest of the robots follow a fixed policy that is dependent on the position and possibly motion of the leader, as described in the next section. The action space thus scales linearly in the number of leaders.

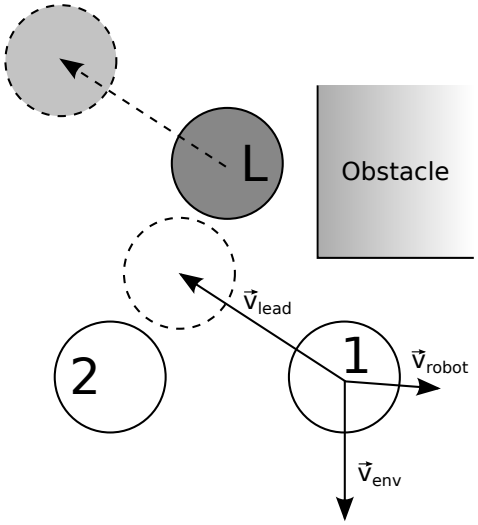


Fig. 2. Policy Generation. The leader is shown in dark grey. The dashed arrow is the motion primitive. The dashed light grey circle is the position of the leader in the successor state. The dashed white circle is the desired position of follower 1. The solid lines represent each of the velocity vectors that are used to determine the policy of robot 1.

2) Policy Generation: The policy, in general, is a motion that each follower (non-leader) takes in response to a motion executed by the leader. While there is flexibility in what policies follower robots should execute, in our implementation, we generate a policy through the weighted sum of velocity vectors returned from three different behaviors. This is illustrated in Figure 2. Every agent has a desired position relative to the origin O of the formation. Let ${}^iT_O$ represent the desired transform from the origin to robot i.

Leader-follow $\vec{v}_{lead}$ **:** This behavior tries to maintain the shape of the formation. The current position of the origin O is computed by applying the inverse transform ${}^lT_O^{-1}$ from the position of the leader robot l. We then compute the desired pose for each robot i, which is obtained by applying the transform ${}^iT_O$ from the origin. A velocity vector is computed in the direction of this desired pose.

Environment $\vec{v}_{env}$ **:** This behavior pushes the robots away from obstacles in the environment. A velocity vector that directs the robot away from the nearest obstacle is used. The magnitude of this component is computed by using a negative exponential gradient that decays away from the obstacles, so that we get a smooth, but exponentially increasing repulsion as the robot moves closer to the obstacle.

Inter-robot $\vec{v}_{robot}$ **:** This behavior pushes robots away from each other. For a robot i the resultant robot velocity is the sum of repulsive velocities from all other robots j.

The relative weights of each of these behaviors is domain-specific and controls several aspects of the formation, e.g., the rigidity and distance maintained from obstacles.

3) Edge Costs: The cost $c(s, s')$ of each edge $e \in E$ represents how expensive it is for the formation to transition from the source state s to the successor s'. While this function is domain specific, one reasonable combination of costs is:

$$c(s,s') = \begin{cases} c_m(s,s') + c_e(s,s') & l(s') = l(s) \\ c_m(s,s') + c_e(s,s') + c_l & l(s') \neq l(s) \end{cases},$$

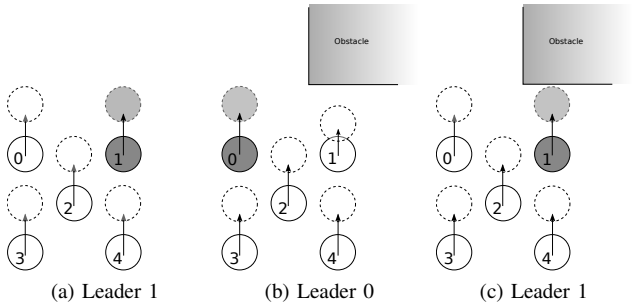
where $c_m(s, s')$ is the motion cost, $c_e(s, s')$ is the cost associated with the formation error, and c_l is the cost of changing the leader. The motion cost is the sum of the distances moved by the agents in that step. The formation error cost penalizes the deviation of the current state from the desired formation. The change leader cost represents how expensive it is to change the leader of the formation from the current state to the next state. For instance, changing the leader might involve communicating to the entire team, which could potentially be an expensive process.

B. Weighted A based Approach*

Weighted A* has been applied to a variety of path planning problems. Here, we show how weighted A* can be applied to our problem. When a state s is removed from $OPEN$ for expansion, the algorithm generates successors using each of the possible leaders $k \in L$. This means that for every expansion, it produces $\sum_{i \in L} |M(i)|$ number of successors, where $M(i)$ is the set of motion primitives that can be taken by leader i. Hence, the branching factor of the search tree grows linearly with the potential number of leaders.

To boost the search efficiency, we apply two optimizations: lazy evaluation of edges and pruning with state dominance.

1) Lazy Weighted A:* The Lazy Weighted A* planner [29] is based on the Weighted A* algorithm. It only evaluates edges when the planner intends to use them. This is done by expanding states optimistically. Each successor s is assumed to be valid and when computing its g-value (and consequently its priority), the cost of the edge in-between the state that is being expanded and s is set to the lowest possible cost. Only when an optimistic state is chosen for expansion does the edge to it gets evaluated which may result in it going back into $OPEN$ with a larger cost (to be expanded later), being thrown out if invalid, or being expanded upfront if its priority still remains the minimum in $OPEN$. This saves a lot of computational effort because successors that are generated but not expanded are never evaluated, avoiding a lot of expensive collision checking. In our domain, we set the optimistic cost of an edge to the pre- computed cost of the motion primitive that was used to generate the edge.



(a) Leader 1 (b) Leader 0 (c) Leader 1

Fig. 3. Pruning with state dominance. Leaders are shown in dark grey. The positions of the robots in the successor state are shown as dashed circles. (a) Free space - Successors produced are the same irrespective of leader; (b,c) The same motion primitive with leaders 0 and 1 produce two different states.

2) Pruning with State Dominance: For many search problems, it is possible to identify states which cannot contribute to the optimal solution and can therefore be pruned from the search tree without compromising on search optimality or completeness [30]. As an example, let us consider a mobile robot that is powered by a battery. We wish to plan a path while optimizing over costs and the battery level. Since the battery is a limited resource, we can say that state s_i is always better than another state s_j at the same position, if it has more battery power available and the cost is at least as good. We then say that state s_i *dominates* state s_j.

In our problem, we identify a state dominance rule that enables us to prune away certain states. Consider state s and two of its successors s' and s''. Also, assume that both s' and s'' have the same positions for all robots, that is, $P_i(s') = P_i(s'') \ \forall \ i$. Furthermore, assume that edge $s \rightarrow s'$ preserves the leader, while edge $s \rightarrow s''$ has the leader switched. That is $l(s) = l(s')$ and $l(s) \neq l(s'')$. In such a case, we can prune away the successor s'' because $g(s'') = g(s') + c_l$ due to the cost of changing the leader, and any successor of s'' that could have been produced via subsequent expansion can also be produced by expanding state s' (and at the same cost). Figure 3a shows the expansion of a state without any formation error in free space, that is, in the absence of any nearby obstacles. We observe that irrespective of which leader is used to expand the state, the same set of successors is produced. This is because the policies for the followers are the result of only the motion of the leader, which is just to keep up with the leader's motion primitive. Therefore, the behavior produced is analogous to applying the same motion primitive to all the agents. This lets us efficiently prune away the successors of state expansions with alternate leaders for which an expansion with the current leader would produce a cheaper path. Figures 3b and 3c show an example where pruning is not applicable. In these figures, the same motion primitive is applied to two different agents as leaders. This produces two different successor states, and

therefore neither successor can be pruned.

3) Heuristic: The performance of A*-based search algorithms is highly dependent on the quality of the heuristic function used. Further, the heuristic needs to be admissible to guarantee bounds on the suboptimality of the solution. These constraints make it hard to compute informative heuristics in many domains. Typically, A*-based search algorithms are applied to path planning for a single agent. However, for our problem, we have multiple agents that are potentially heterogeneous. Computing an efficient, admissible heuristic, therefore, can be challenging. We tackle this problem by computing several individually admissible heuristics for each potential leader, and subsequently combining them by taking their maximum, as described later in section V-B.

C. Multi-queue approach

While the Weighted A* based approach described in the previous section performs reasonably well, it suffers from a couple of issues. The first issue is that there is only a single heuristic (even if it is based on a combination of heuristics) that guides the search. The search still suffers from local minima in the space of this heuristic. Recent work on Multi-Heuristic A* (MHA*) has shown that it is advantageous to treat the heuristics separately and use them to guide the search along different paths in the configuration space [6]. MHA* provides us with a principled approach to take advantage of several heuristics when they are available while also providing guarantees on the worst case suboptimality of the solution. Further, these heuristics can be inadmissible, which presents a plethora of options to compute heuristics for multiple agents. The second problem with the single-queue approach is that depending on how the costs are set up for switching leaders, the search might explore a huge number of states with the current leader before switching to an alternate leader. In a lot of scenarios, it is advantageous to be able to simultaneously explore actions led by different leaders. These issues motivate the need for a multiple-queue approach. We maintain separate queues that search the state space using different robots as leaders, but also share found paths between queues. In each queue, we maintain a robot as the leader and use a corresponding heuristic.

1) Generalized MHA:* We present a generalization of the Shared Multi-Heuristic A* algorithm (SMHA*).

SMHA* runs N independent weighted A* searches at the same time, each with a different heuristic. Specifically SMHA* has several $OPEN$ lists. $OPEN_i$ sorts its states according to $f_i(s) = g(s) + w_h h_i(s)$, where h_i is the i^{th} heuristic and w_h, being the weight for weighted A*. The different $OPEN$ lists takes turns expanding states in a round robin fashion (each choosing a state to expand according to its own internal priority f_i) . After an expansion from $OPEN_i$, the generated successors are put into all $OPEN$ lists so that $g(s), \forall s$ is always the same across all the searches. This effectively lets the different searches share path progress with each other. The intuition is that each heuristic has different local minima and by using each of them independently and sharing resulting successors, the different searches help each other get past local minima.

Algorithm 1 GENERALIZED MHA*

```
1:  procedure KEY(i, s)
2:      return g_i(s) + w_h * h_i(s)
3:  procedure UPDATESUCC(i, s, g)
4:      if s was already expanded from OPEN_i then
5:          continue
6:      if g_i(s) > g then
7:          g_i(s) = g
8:          Insert/Update s in OPEN_i with KEY(i, s)
9:          if SATISFIESGOAL(s) then
10:             g(s_goal) = g_i(s)
11: procedure TRANSFERFUNC(i, s)
12:     s'' ← modify s for queue i
13:     return (s'', cost to modify)
14: procedure EXPANDSTATE(i, s)
15:     Remove s from OPEN_i
16:     for all s' ∈ SUCC(s) do
17:         if s' was never seen in the i^th search then
18:             g_i(s') = ∞
19:         UPDATESUCC(i, s', g_i(s) + c(s, s'))
20:         UPDATESUCC(0, s', g_i(s) + c(s, s'))
21:         for j = 1, ..., n, j ≠ i do
22:             (s'_j, c_transfer) ← TRANSFERFUNC(j, s')
23:             if s'_j was never seen in the j^th search then
24:                 g_j(s'_j) = ∞
25:             UPDATESUCC(j, s'_j, g_i(s') + c_transfer)
26:             UPDATESUCC(0, s'_j, g_i(s') + c_transfer)
27: procedure MAIN()
28:     for i = 0, 1, ..., n do
29:         OPEN_i ← ∅
30:         g_i(s_start) = 0
31:         Insert s_start in OPEN_i with KEY(i, s_start)
32:     q ← 0
33:     while g(s_goal) > w_a * OPEN_0.MINKEY() do
34:         q ← (q + 1 mod n) + 1
35:         key_0 = OPEN_0.MINKEY()
36:         key_q = OPEN_q.MINKEY()
37:         if key_q ≤ w_a * key_0 then
38:             s ← OPEN_q.TOP()
39:             EXPANDSTATE(q, s)
40:         else
41:             s ← OPEN_0.TOP()
42:             EXPANDSTATE(0, s)
```

In addition to being effective at solving challenging planning problems, MHA* has strong theoretical properties. In addition to the N searches run by $OPEN_1 \ldots OPEN_N$, there is an *anchor search* which uses $OPEN_0$. The anchor heuristic is required to be admissible and consistent, in contrast to the other N heuristics, which have no constraints at all. The anchor is used to bound the solution cost to within a user-chosen factor $(w_h \times w_a)$ of the optimal solution cost.

We provide a generalized SMHA* which is particularly effective in multi-robot domains. In our case, each $OPEN_i$ represents a different robot (not counting the anchor). Specifically, $OPEN_i$ only contains states where robot i is the leader. When expanding state s from $OPEN_i$, a successor s' is generated only if $l(s') = l(s)$, i.e. leader is unchanged.

Now, the generalization of SMHA* comes from how we share successors to the other $OPEN$ lists. After generating a successor s' from an expansion in $OPEN_i$, the state s'_j is shared to $OPEN_j$ with an additional

transition cost $c_{transfer}$. This is done using the domain-specific function $(s'_j, c_{transfer})$ =TRANSFERFUNC(j, s'). This function allows states to be transformed (in configuration space as well as cost) before being inserted into another $OPEN$ list. SMHA* is the trivial case where $(s', 0)$ =TRANSFERFUNC(j, s'), which means that the state is being shared unmodified.

In our multi-robot problem, we stated that each $OPEN_i$ only contains states where i is the leader. For this setup to work without losing the benefits of the "path sharing" of SMHA*, the $TransferFunc$ is used. Specifically, a state s generated from $OPEN_i$, before being inserted into $OPEN_j$, will have its leader changed to j and its cost increased by c_l to indicate the leader switch.

Our method gets a large speedup from a low branching factor. Recall, when expanding a state from $OPEN_i$, only successors that maintain leader i are generated. Successors with leader j that were ignored can still be generated by expanding the state $s_j(P(s_j) = P(s_i))$ from $OPEN_j$, where $P(s)$ is the position of all the robots at state s.

Finally, the theoretical guarantee on bounded suboptimality provided by MHA* is maintained by our generalized version using the same anchor search. To ensure this, every generated successor and transferred successor are shared with the anchor search. When a state s is expanded from the anchor, it generates successors under all potential leaders as in the Weighted A* approach.

In algorithm 1, the MAIN procedure is the same as that from standard MHA* [6]. MHA* selects an $OPEN$ list in a round robin fashion (line 34) and calls the EXPANDSTATE function. Lines 16-19, generates each successor and inserts it in the current queue (using UPDATESUCC). Lines 21-25 use the TRANSFERFUNC to modify the successor before inserting it in the other queues. Lines 20 and 26 ensure that every state is placed in the anchor queue. The UPDATESUCC procedure updates the g-value of the state and inserts it in the given $OPEN$ list (if applicable).

2) Heuristics: The strength of the Multi-Heuristic A* framework resides in the use of inadmissible heuristics that are often easier to design and more informative. This fits very well with our domain, because computing an admissible heuristic is a challenge for multi-agent teams. Recall that we have an open list $OPEN_i$ for each leader i. While not required, a reasonable choice of the heuristic h_i would be one that guides this search while prioritizing leader i reaching the goal. It is also possible to compute other inadmissible heuristics that are not tied to a particular leader. These introduce additional $OPEN$ lists which receive and generate successors in the same manner as the anchor search.

V. EXPERIMENTAL RESULTS

For our experiments, we look at path planning for deformable formations of multi -agent teams. The configuration space of each robot in the formation is 2D (x, y). The environment is generated by randomly placing box-shaped obstacles. We randomly sample for valid start and goal configurations of the desired formation from the environment.

For each generated trial, we ran three planners. The planning and parameters in computing individual robot policies were kept the same for all three planners. For Generalized MHA*, w_h was set to 15, and w_a to 1.5. This gives us a suboptimality bound of 22.5. The inflation for the heuristic was set to be the same for the single-queue approaches so that the suboptimality bound remained the same across all three methods. We set the cost to change the leader (c_l) to a value that is higher than the most expensive motion primitive. An experiment set comprises of 10 randomly generated environments with 10 trials each, which gives us a total of 100 random trials over a range of different environments. We ran two such experiment sets, one each for formations with 5 and 21 robots respectively. The planners are compared on the basis of several metrics such as success rates in finding feasible plans within a timeout (30s for the 5 robot formation and 60s for the 21 robot formation), planning times, expansions, and number of successors generated. For each pairwise comparison, we average the metrics over the trials where both the planners succeeded.

A. Single Leader

In this approach, we plan for the entire swarm using a single leader. We use the lazy weighted A* planner with a heuristic that was computed using a 2D Dijkstra search from the leader's location at the goal state.

B. Multiple leaders - Lazy Weighted A*

This is the planner presented in section IV-B. We compute a heuristic for each robot using a 2D Dijkstra search from its respective position at the goal. Each is a perfect heuristic for the robot in absence of the rest of the formation. Therefore, the heuristics are all admissible. We then take the maximum of all of these heuristics, which is known to give us an admissible heuristic. The intuition behind using the `max` function is that we prioritize the states based on the position of the robot that is farthest away from the goal according to the heuristic estimates. This prevents scenarios where the leader can keep progressing toward the goal, but a follower robot is stuck at a local minima. The heuristic for the follower robot would be higher and therefore, the search is guided to help this robot make progress.

C. Generalized MHA*

In our implementation, we have a search queue for each potential leader. For simplicity, let's assume that the leaders are numbered $1, 2, \ldots, |L|$. Therefore, queue i searches with the leader i. We compute $|I|$ admissible heuristics, one for each robot using a 2D Dijkstra search from its goal location. This gives us the heuristics $h_i, i = 1, 2, \ldots, |I|$. We then compute the maximum of these heuristics, h_{max}, which is also admissible. Essentially, this heuristic captures the robot that is farthest from the goal. The anchor search uses h_{max} for its search. In each individual leader's search, we prioritize the leader's heuristic to allow the search to follow this heuristic, but also ensure that no robot gets left behind too far. This is achieved by setting the heuristic for queue i

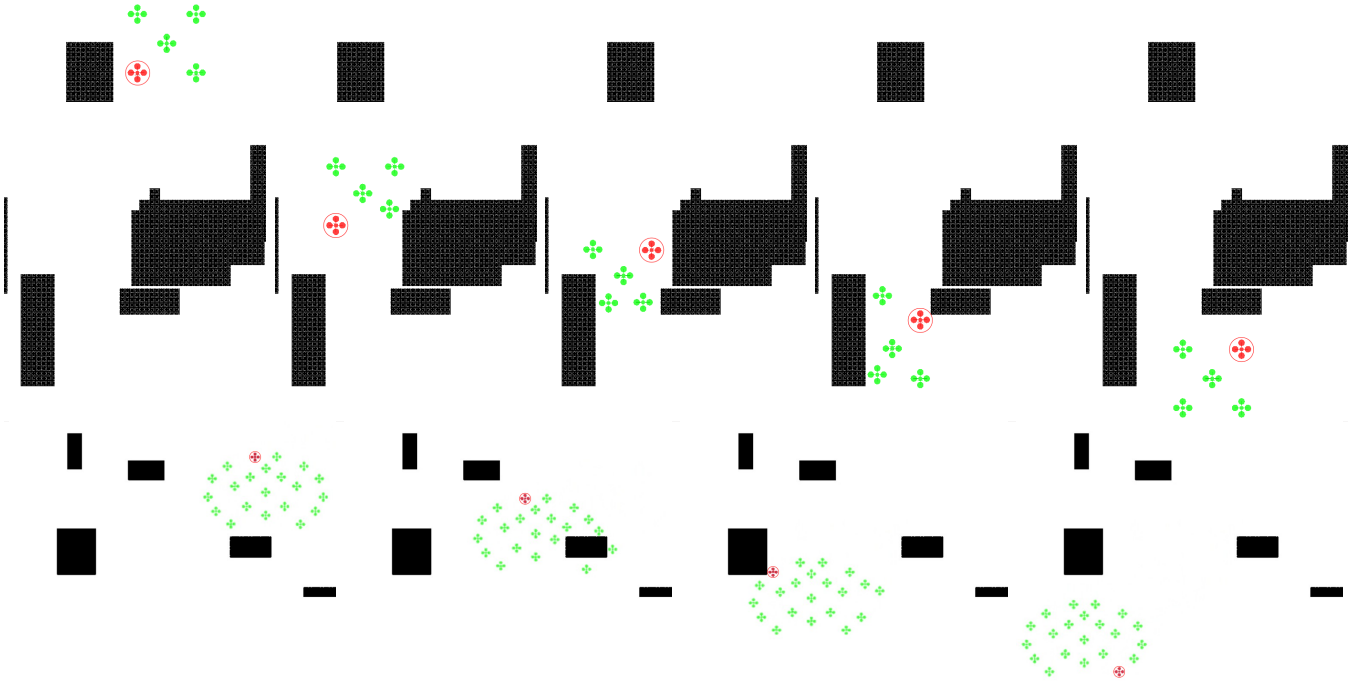


Fig. 4. Path generated by the planner for a team of 5 robots and for a team of 21 robots. The leader robot is encircled.

(which searches with leader i) to $\max(\eta \times h_i, h_{max}), \eta > 1$. For appropriately set η, q_i will be led by the leader's heuristic alone, unless a follower gets left behind.

In addition to the heuristics described above, a second set of heuristics is computed for the formation as a whole, introducing additional planner queues. At any state, we compute the centroid of the positions of all the robots. We then treat the formation as a circular robot centered about the centroid. Heuristics for several different radii ranging from the inscribed radius of the undeformed formation to its circumscribed radius are generated. Practically, this is implemented by inflating the obstacles by each of the different radii and then computing a Dijkstra search from the centroid of the goal position. These heuristics lead the entire formation around the obstacles, and tend to avoid narrow gaps. When a state is expanded from these queues, choice of leader is not restricted. We treat it like in the the Weighted A* based planner; that is, we generate the successors from all potential leaders. The search automatically chooses the leader with the least cost. For our experiments, we computed two such heuristics, one for the inscribed radius of the formation and another for the circumscribed radius.

In figure 4, we show snapshots from the paths produced by generalized MHA* for the 5-robot formation and the 21-robot formation. In the first case, the search is initially led by the robot on the bottom left. However, as the formation enters the narrow passage, the robot on the top right takes over as the leader to drive the robots in front through the narrow gap. The formation deforms smoothly as the leader rounds the corner, and the entire formation restores its original shape as it emerges on the other side.

Table I compares generalized MHA* with the single leader method. The success rate increases dramatically by allowing leader switching and by using multiple heuristics to guide the search. The planning time is reduced about 5x, because the search quickly progresses out of local minima. This trend is reflected in the average number of expanded states.

TABLE I

GENERALIZED MHA* VS SINGLE LEADER SEARCH

	Generalized MHA*	Single leader search
success rate	97%	61%
average planning time	0.62	3.27
average expansions	297.30	6381.17
average solution cost	24167.43	25031.10

TABLE II

GENERALIZED MHA* VS WEIGHTED-A*

	Generalized MHA*	Weighted A*
success rate	97%	81%
average planning time	0.59	0.62
average expansions	278.07	302.48
average solution cost	26772.33	28126.63
average leader changes	0.27	2.62
average generated successors	10765.58	36665.37

TABLE III

GENERALIZED MHA* AND WEIGHTED-A* STATS FOR A LARGE FORMATION (21 ROBOTS)

	Generalized MHA*	Weighted A*
success rate	80%	70%
average planning time	2.13	2.13
average expansions	131.60	106.88
average solution cost	124501.78	140441.07
average leader changes	24.87	59.97
average generated successors	7107.24	28222.81

Table II presents a comparison of generalized MHA* and weighted A* with leader switching. Both planners have better success rates than the single leader search. The weighted A* based method generates around 3.4x the number of successors. This is expected because every expansion produces successors from every potential leader.

The choice and number of leaders depend on formation shape. More leaders implies more queues in the planner, but better control over the formation. For the larger formation, 8 leaders were used. The results are presented in Table III. Success rates for both methods remain high. Similar trends are observed in planning times, expansions, and number of successors generated. Thus, generalized MHA* scales well with the number of robots.

VI. CONCLUSIONS

In this work, we studied the problem of motion planning for multi-agent formations; in particular, planning in the context of follow-the-leader approaches while allowing for leader switching. To this end, we have proposed several planning approaches that dealt well with the increased complexity of planning when leader switching is allowed. The experimental results demonstrate that explicitly reasoning over when to switch the leader in the planning process results in drastic improvements in the success rate of finding feasible paths and significant reductions in planning times.

In the future, we plan to test our approach on a small team of real UAVs. In addition, we would like to explore planning under the constraints of maintaining line-of-sight. In the presented algorithm the formation points were attached to robots (leaders). However, this need not be the case. Instead, any set of virtual leader points in the formation could be chosen for the formation to deform about. A potential advantage of attaching them to physical robots is that during execution, an actual leader-follower control method could be employed, which could eliminate the need for all agents to be performing localization at all times.

REFERENCES

[1] J. Butzke, K. Daniilidis, A. Kushleyev, D. D. Lee, M. Likhachev, C. Phillips, and M. Phillips, "The university of pennsylvania magic 2010 multi-robot unmanned vehicle system," *Journal of Field Robotics*, vol. 29, no. 5, pp. 745–761, 2012.

[2] D. Vallejo, P. Remagnino, D. N. Monekosso, L. Jimnez, and C. Gonzalez-Morcillo, "A multi-agent architecture for multi-robot surveillance." in *ICCCI*, ser. Lecture Notes in Computer Science, N. T. Nguyen, R. Kowalczyk, and S.-M. Chen, Eds. Springer, 2009.

[3] N. Schurr, J. Marecki, M. Tambe, and P. Scerri, "The future of disaster response: Humans working with multiagent teams using defacto," in *In AAAI Spring Symposium on AI Technologies for Homeland Security*, 2005.

[4] S. Carpin and L. Parker, "Cooperative leader following in a distributed multi-robot system," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, 2002.

[5] T. Barfoot and C. Clark, "Motion planning for formations of mobile robots," *Robotics and Autonomous Systems*, 2004.

[6] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a*," in *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[7] V. Gazi, "Swarm aggregations using artificial potentials and sliding-mode control," *Robotics, IEEE Transactions on*, 2005.

[8] V. Gazi and K. Passino, "A class of attraction/repulsion functions for stable swarm aggregations," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 3, Dec 2002.

[9] T. Balch and M. Hybinette, "Social potentials for scalable multi-robot formations," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, 2000, pp. 73–80 vol.1.

[10] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 6, pp. 926–939, Dec 1998.

[11] P. Ghosh, J. Gao, A. Gasparri, and B. Krishnamachari, "Riverswarm: Topology-aware distributed planning for obstacle encirclement in connected robotic swarms," in *First Workshop on Robotic Sensor Networks 2014 (RSN2014)*, 2014.

[12] X. C. Ding, M. Powers, M. Egerstedt, and R. Young, "An optimal timing approach to controlling multiple uavs," in *American Control Conference, 2009. ACC '09.*, June 2009, pp. 5374–5379.

[13] D. Panagou and V. Kumar, "Cooperative visibility maintenance for leader-follower formations in obstacle environments," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 831–844, 2014.

[14] C. Vo, J. F. Harrison, and J.-M. Lien, "Behavior-based motion planning for group control," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009.

[15] J.-M. Lien, S. Rodriguez, J.-P. Malric, and N. M. Amato, "Shepherding behaviors with multiple shepherds," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 3402–3407.

[16] J.-M. Lien, B. Bayazit, R. T. Sowell, S. Rodriguez, and N. M. Amato, "Shepherding behaviors," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 4. IEEE, 2004, pp. 4159–4164.

[17] J. P. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*. IEEE, 1998.

[18] M. B. Egerstedt and X. Hu, "Formation constrained multi-agent control," 2001.

[19] J. P. Desai, J. P. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 6, pp. 905–908, 2001.

[20] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, 2013.

[21] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 2011.

[22] M. Turpin, N. Michael, and V. Kumar, "Capt: Concurrent assignment and planning of trajectories for multiple robots," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 98–112, 2014.

[23] O. B. Bayazit, J.-M. Lien, and N. M. Amato, "Better group behaviors using rule-based roadmaps," in *Algorithmic Foundations of Robotics V*. Springer, 2004, pp. 95–112.

[24] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, 1996.

[25] A. Kamphuis and M. H. Overmars, "Motion planning for coherent groups of entities," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 4. IEEE, 2004, pp. 3815–3822.

[26] ——, "Finding paths for coherent groups using clearance," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 2004, pp. 19–28.

[27] I. Pohl, "First results on the effect of error in heuristic search," *Machine Intelligence*, vol. 5, pp. 219–236, 1970.

[28] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press, 2003.

[29] B. Cohen, M. Phillips, and M. Likhachev, "Planning single-arm manipulations with n-arm robots," in *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[30] T. Fujino and H. Fujiwara, "An efficient test generation algorithm based on search state dominance," in *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, July 1992, pp. 246–253.