Decentralized prioritized planning in large multirobot teams

Prasanna Velagapudi, *Student Member, IEEE*, Katia Sycara, *Fellow, IEEE*, and Paul Scerri

***Abstract*— In this paper, we address the problem of distributed motion planning for large teams of hundreds of robots in constrained environments. We introduce two distributed prioritized planning algorithms: an efficient, complete method which is shown to converge to the centralized prioritized planner solution, and a sparse method in which robots discover collisions probabilistically. Planning is divided into a number of iterations, during which every robot simultaneously and independently computes a planning solution based on other robots' path information from the previous iteration. Paths are exchanged in ways that exploit the cooperative nature of the team and a statistical phenomenon known as the "birthday paradox". Performance is measured in simulated 2D environments with teams of up to 240 robots. We find that in moderately constrained environments, these methods generate solutions of similar quality to a centralized prioritized planner, but display interesting communication and planning time characteristics.**

I. INTRODUCTION

Much previous research has been done on the problem of cooperative path planning. Unfortunately many proposed approaches do not scale to teams of hundreds of robots in constrained environments. One approach which has been shown to be effective for reasonably large teams is *prioritized planning* [1]. In this approach, robots sequentially plan paths according to a prioritization function. However, this means robots must plan paths in order, resulting in a linear increase in overall planning time with the number of robots. Prioritized planning is also centralized, creating a potential computational and communication bottleneck, as well as a single point of failure.

However, in many domains, the strict ordering of sequential planning is likely to be unnecessarily expensive because not all robots need to avoid all other robots. Online prioritized approaches, such as [2] and [3], take advantage of this property by determining which sets of robots need to be planned sequentially by detecting interactions via local observations. In this paper, we describe an approach that distributes prioritized planning, allowing each robot to plan at the same time, then look for collisions between paths and require lower priority robots to replan. The intuition behind this is that if robot paths are not dependent on all other paths, the number of replanning iterations might be quite low and overall planning time will be reduced because no single node needs to plan for each robot in sequence. We prove that distributing the planning in this way still converges to the same result as the centralized planner. Experimental results support the intuition, showing a dramatic reduction in the number of planning iterations. However, because the length of an iteration for the distributed algorithm is governed by the slowest planner, there is not always a large gain in overall planning time.

For a robot to know that its path is collision free, its path must be checked against the paths of every other robot. If each robot sends their path to every other robot during each planning iteration, though, the use of communication bandwidth may become prohibitively high. However, many robots do not need to change paths during an iteration. Thus, simply having robots assume that paths have not changed if they do not receive a message containing a new path can reduce communication dramatically. Empirically, we find this simple change reduces message traffic by 80% while maintaining the performance of the full version.

Moreover, we observe that it is not necessary for a robot to detect every collision with its path itself. In a cooperative team, it is sufficient that *some* robot detects the collision in the planned paths and communicates with the robots involved. Due to the "birthday paradox", it is possible to communicate newly planned paths with only a few other robots and have extremely high probability of some robot being able to detect every possible collision. Surprisingly, adding this strategy to the algorithm does not result in an overall reduction of communication, because it results in more planning iterations being required for convergence. We believe that this is due to robots using paths with which they are not yet colliding to effectively preempt possible collisions. By not sending paths to everyone, the robots lose this ability and more iterations are required. However, in sparse environments leveraging the birthday paradox is shown to be useful.

II. RELATED WORK

The problem of multi-robot path planning has been extensively studied for a number of years. For a detailed summary of the literature, we refer readers to [4]. Previous approaches can be generally divided into coupled and decoupled strategies. Coupled planners (e.g. [5], [6], [7], [8]) combine the DOFs of each robot into a single high-dimensional composite robot and plan in this joint space. These approaches can theoretically find optimal solutions for multi-robot planning problems, but are restrictive in the number of robots for which they can plan, as the complexity of planning grows exponentially with number of robots. Thus, while they provide the highest-quality solutions overall, they are generally intractable for large teams.

Decoupled approaches plan for each robot independently, then adjust the plans in various ways to account for the paths of other robots. The two most common such approaches are path coordination and prioritized planning. The former

P. Velagapudi, K. Sycara, and P. Scerri are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA {pkv, katia, pscerri}@cs.cmu.edu

decomposes the planning problem into a spatial path planning problem and a temporal velocity planning problem [9]. Robots first plan paths in space that avoid static obstacles, but ignore dynamic constraints such as other robots. Then, coupled [10], [11], [12], [13], prioritized [1], [14], or reactive [15] methods are used on the simpler problem of solving for velocity profiles that allow the robots to avoid collisions.

Prioritized planning, introduced in [16], is another, often complementary approach to path coordination. In it, trajectories are generated sequentially according to some prioritization of the robots. Higher priority robots form obstacles in space-time for lower priority robots. This greedy and incomplete strategy is often effective in practice [17], [18]). Searching or optimizing over the prioritization can yield improved solutions [19], [20], [21], but often, simple heuristics perform quite well [1]. However, a major problem with prioritized planning is that paths must be computed in sequence, which scales poorly to large numbers of robots.

Several works have tried to address this by identifying simplified graph structures online for which prioritized planning can be done. In [22], robots are divided into small coupled cliques and plan to execute in order, with each group waiting until the last finishes. This allows solutions in certain highly constrained settings, but can lead to long execution times as cliques must execute in sequence rather than simultaneously. In [23], the map itself is partitioned into subgraphs, simplifying the space over which prioritized planning takes place. In [2], dynamic networks are formed as robots move through the environment. When new members are added, trajectories are exchanged and checked for collisions. Each robot then executes a prioritized replan on its own probabilistic roadmap [24], with the best plan used by the entire group. In [3], robots locally sense conflicts, and use a prioritization scheme to determine whether to add the corresponding dynamic obstacles to their maps. The approach presented in this paper is similar to this latter technique, however it makes no assumptions between connectivity and spatial locality, it attempts to resolve all conflicts at planning time rather than during execution, and it addresses much larger team sizes.

III. PROBLEM

Suppose we have a team of n robots $R = \{r_1, \ldots, r_n\}$. These robots are traversing a binary obstacle map O. Each robot r_i has some start location s_i and some goal location g_i. We consider this team over a time window $[t_i, t_f]$. Our objective is to find a set of trajectories $\Pi = \{\pi_1, \ldots, \pi_n\}$ such that for each r_i, $\pi_i(t_i) = s_i$ and $\pi_i(t_f) = g_i$. This set of trajectories should not allow any two robots to collide as defined by some function COLLISIONCHECK() that compares two path sets Π_1 and Π_2 and returns a boolean indicating whether they ever occupy the same space at the same time.

We are also given a deterministic independent planning function PLAN() that takes an obstacle map O, a start s and goal g, and a set of obstacle trajectories (of other robots) Π_{obs}, and returns a collision-free trajectory π from s to g, if such a path is possible. Robots are assumed to have complete peer-to-peer communication over the team, as well as a distributed synchronization mechanism allowing them to wait for all teammates to reach a certain point in algorithm execution.

IV. ALGORITHMS

In this section, we define the prioritized planning algorithms used in this paper. We begin with a brief overview of canonical prioritized planning, as described in [16]. From this, we define a straightforward distributed algorithm that yields the same result. Next, we show that the communication necessary for this distributed algorithm can be reduced by reasoning about the absence of messages and discovering which robots share dependencies. Finally, we propose a variant that detects dependencies using a probabilistic method, potentially reducing communication further and relaxing constraints on robots' knowledge of the team.

A. Prioritized planning

Prioritized planning is a decoupled planning strategy first introduced in [16]. The underlying concept is as follows: given a team of robots and an obstacle map, assign a priority to each robot in the team. Then, plan for the robots sequentially, in the order of their priorities. As each robot is assigned a path, map that path to a dynamic obstacle in the map that subsequent robots must plan around. Assuming that independent planning for each robot is successful, a set of collision-free paths can be generated in exactly n planning cycles.

B. Distributed prioritized planning

The complete distributed prioritized planner is a simple variant of the conventional prioritized planner as applied to multi-robot problems. Planning is divided into a number of iterations, during which every robot simultaneously and independently computes a planning solution based on other robots' path information from the previous iteration. At the end of each iteration, each robot transmits its current plan to every other robot, to be used in subsequent planning stages.

Theorem IV.1. *Given complete communication of robot paths in every iteration and a deterministic path planner, in n iterations, the complete distributed prioritized planner will output the same solution as a centralized prioritized planner.*

Proof: This can be proved using induction. For a team of size 1 (e.g. $R = \{r_1\}$), the centralized planner will output some planned path π_1. On the first iteration, the complete distributed planner will output the same independent path.

Now suppose that the theorem holds for all teams of size n or smaller. Consider a team $R = \{r_1, \ldots, r_{n+1}\}$ of size $(n+1)$. Without loss of generality assume the robots are ordered by priority (e.g. $p_1 > p_2 > \ldots > p_{n+1}$. Then consider the subteam $R' = \{r_1, \ldots, r_n\}$. Since any path generated by r_{n+1} will simply be ignored by robots in R', R' will match the centralized solution in n iterations, by the inductive hypothesis. Now, in iteration $(n+1)$, robot r_{n+1} will receive paths $\{\pi_1, \ldots, \pi_n\}$ which exactly match the

centralized planner solution. Thus, because both prioritized planners are using the same deterministic path planner, with the same map and same obstacle paths, they must return the same path π_{n+1}. □

C. Reduced distributed prioritized planning

We can drastically reduce the number of messages communicated by taking advantage of the static prioritization of robots. Suppose the robots exchange messages containing priority along with their path: $m_i = \langle i, p_i, \pi_i \rangle$. Then, a receiving robot can compare the priority value to its own priority, and determine if the sender robot has a higher or lower priority. If priorities are static, the receiving robot can maintain this knowledge in subsequent rounds. If the priority is lower, the receiving robot would simply ignore the path in any iteration.

In order to reduce the number of messages, r_i maintains a dependency list D_i and a path cache Π_{cache}. The list is initialized to contain all of the robots in the team, and the cache is initially empty. In each iteration, when a robot r_i receives a message $m_j = \langle j, p_j, \pi_j \rangle$, it compares the priority to its own, p_i. If $p_j > p_i$, the robot will remove the sender r_j from its dependency list and add π_j to Π_{cache}. If $p_j < p_i$, it is discarded, but r_j remains on the dependency list D_i. At the end of each iteration, if a robot has replanned, it sends its new plan to all robots on its dependency list.

In the first iteration, this leads to identical behavior to the original distributed planner. However, in subsequent rounds, robots will only communicate to robots that have a lower priority. In addition, if robots do not change their plans, no further communication to lower priority robots is necessary, as these robots have the previous plan already cached.

Algorithm 1 REDUCED DISTRIBUTED PLANNER(i)

```
1:  D_i = {1, ..., n}
2:  Π_cache = ∅
3:  π_i ← PLAN (O, ∅)
4:  p_i ← COMPUTEPRIORITY (π_i)
5:  m_i ← ⟨i, p_i, π_i⟩
6:  SENDTOTARGETS (m_i, D_i)
7:  repeat
8:      M_recv ← RECEIVEMESSAGES ()
9:      for all ⟨j, p_j, π_j⟩ ∈ (M_recv) do
10:         if p_j > p_i then
11:             Π_cache ← Π_cache ∪ {π_j}
12:         else
13:             D_i ← D_i \ {j}
14:     isCollided ← COLLISIONCHECK (π_i, Π_cache)
15:     if !isCollided then
16:         π_i ← PLAN (O, Π_cache)
17:         m_i ← ⟨i, p_i, π_i⟩
18:         SENDTOTARGETS (m_i, D_i)
19:     everyoneDone ← WAITFOROTHERS (!isCollided)
20: until everyoneDone
```

D. Sparse distributed prioritized planning

It is possible to further reduce communications by exploiting the cooperative nature of the team in conjunction with a statistical phenomenon known as the "birthday paradox". In the birthday paradox, in a set of randomly chosen people, the probability that two will have the same birthday grows rapidly as the set size is increased. We can use this in a communication algorithm by considering "birthdays" to be collisions, and people to be robot paths. Since we have a cooperative team, we can rely on other agents to check our paths for collisions with the other paths they know about. If each robot sends its path to a set of randomly chosen teammates, then each robot has received its random set of "people" (paths), within which it can search for a common "birthday" (pairwise path collision). The exact probability of this detection grows rapidly with the number of teammates each robot shares its path with, and the function is formally computed later in this section.

We therefore introduce decentralized collision detection to form the *sparse distributed prioritized planner* (Alg. 2) with the aim of reducing the number of exchanged messages while still detecting potential path collisions with a high probability. Robots start with an empty dependency list D_i. Then, complete exchange of paths is replaced by a pair of communication stages. In the first stage, when each robot computes its path, it sends out a message to λ random teammates (lines 5 and 25 in Alg. 2). Then, in the second stage, it takes the received messages from its teammates, as well as its own path, and checks every pair of paths for possible collisions (lines 7-10 in Alg. 2). For any pairwise collisions are detected, the robot forwards the opposing messages directly to the two conflicting robots (i.e. $m_i \rightarrow r_j$, $m_j \rightarrow r_i$), thus alerting them both to the conflict (lines 12-13 in Alg. 2). When a conflict is received, the higher priority robot adds the originator of the message to its dependency list, ensuring the lower priority robot will be alerted upon a plan change. As long as the probability of collision detection is sufficiently high, the sparse algorithm seems as though it will quickly converge to the behavior of the reduced planner, while not requiring an initial iteration of complete path exchange.

1) Statistical Properties of Collision Communication: The effectiveness of collision communication depends on the probability that a particular collision will be detected in a team. This detection probability corresponds to the probability that two colliding path messages will both be received by a single robot anywhere in the team. We can compute this by considering the complementary probability that two paths that collide will *not* both be received by any robot in the team.

Suppose we have a team A of n robots using collision communication with γ messages being sent randomly in the first iteration. Let us consider some pair of colliding robots $a, b \in A$ that send out messages m_a and m_b. robot a sends m_a to some set of robots N_a, while robot b sends m_b to some set of robots M_b. From the definition of collision communication, $|N_a| = |N_b| = \gamma$. A collision will not be

Algorithm 2 SPARSE DISTRIBUTED PLANNER(i)

$D_i = \emptyset$
$\pi_i \leftarrow$ PLAN $(O, \emptyset)$
$p_i \leftarrow$ COMPUTEPRIORITY (π_i)
$m_i \leftarrow \langle i, p_i, \pi_i \rangle$
SENDTORANDOM (m_i, γ)
repeat
  $M_{recv} \leftarrow$ RECEIVEMESSAGES $()$
  for all $\langle j, p_j, \pi_j \rangle \in (M_{recv} \cup m_i)$ **do**
    for all $\langle k, p_k, \pi_k \rangle \in (M_{recv} \cup m_i)$ **do**
      $isCollided \leftarrow$ COLLISIONCHECK (π_j, π_k)
      if isCollided **then**
        SENDTOTARGET (m_j, k)
        SENDTOTARGET (m_k, j)
  $M_{recv} \leftarrow$ RECEIVEMESSAGES $()$
  for all $\langle j, p_j, \pi_j \rangle \in (M_{recv})$ **do**
    if $p_j > p_i$ **then**
      $\Pi_{cache} \leftarrow \Pi_{cache} \cup \{\pi_j\}$
    else
      $D_i \leftarrow D_i \setminus \{j\}$
  $isCollided \leftarrow$ COLLISIONCHECK (π_i, Π_{cache})
  if $!isCollided$ **then**
    $path \leftarrow$ PLAN (O, Π_{cache})
    $m_i \leftarrow \langle i, p_i, \pi_i \rangle$
    SENDTOTARGETS (m_i, D_i)
    SENDTORANDOM $(m_i, \max(0, \gamma - |D_i|))$
  $everyoneDone \leftarrow$ WAITFOROTHERS $(isCollided)$
until $everyoneDone$

detected if $N_a \cap N_b = \emptyset$. Thus, we can simply consider the possible ways of choosing the members of N_b from the set $A - N_a$ against the possible ways of choosing the members of N_b from A.

$$P(N_a \cap N_b = \emptyset) = \frac{\binom{n-\gamma}{\gamma}}{\binom{n}{\gamma}}$$

This implies that the probability of detecting a collision is:

$$P(N_a \cap N_b \neq \emptyset) = 1 - \frac{\binom{n-\gamma}{\gamma}}{\binom{n}{\gamma}} \quad (1)$$

We can also consider a lower bound on the collision detection probability by computing the probability of detecting a collision if teammates are chosen randomly, with replacement. In this case, the probability that some robot c will receive a message is $P(c \in N_a) = 1 - \left(\frac{n-1}{n}\right)^{\gamma}$. Since a and b will have their own messages, and would not send their messages to the same teammates multiple times, it is clear that this is a lower bound on the actual probability. Now, the chance of some robot c *not* detecting a collision is

$$P(c \notin N_a \cap N_b) = 2\left(\frac{n-1}{n}\right)^{\gamma} - \left(\frac{n-1}{n}\right)^{2\gamma}$$

This means that, since detection will occur unless every robot doesn't detect the collision, we have the following:

$$P(N_a \cap N_b \neq \emptyset) = 1 - \left(2\left(\frac{n-1}{n}\right)^{\gamma} - \left(\frac{n-1}{n}\right)^{2\gamma}\right)^{n} \quad (2)$$

An interesting property emerges if γ is selected to be of the form $\gamma = k \cdot \sqrt{n}$. In Figure 1, we plot the resulting collision probabilities with various k. As the team size increases, the exact probability and the lower bound converge together toward an asymptote. This suggests that using $\gamma \sim \sqrt{n}$ allows team-size-invariant tuning of collision detection probability. While the method is still probabilistic and can miss collisions, the results in Figure 1 suggest that values of $k > 2$ have extremely high probabilities of collision detection, implying that sparse distributed planning is likely detect colliding paths efficiently in large teams.

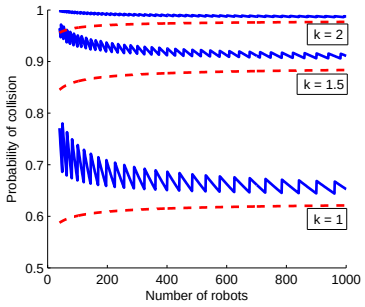
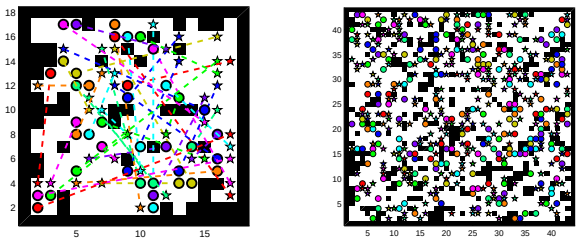


Fig. 1. Collision detection probability as team size increases when $\gamma = k \cdot \sqrt{n}$. The solid line is the exact probability, and the dashed line is the lower bound. The sawtooth shape of the exact probability function is because the actual number of messages sent must be an integer.

V. EXPERIMENTAL DESIGN

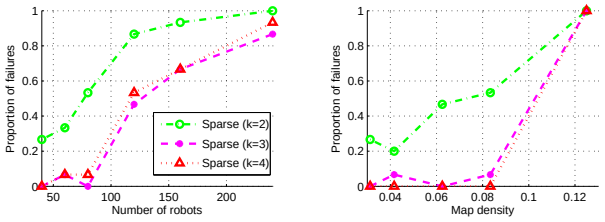
Detailed experiments were performed to evaluate the performance of these algorithms. In this section we describe the experimental setup used and in the following section we describe the corresponding results. Experiments were carried out in a 2D grid simulation on binary obstacle maps. Maps were generated by iterating through cells in row-first order using a cellular automaton model, where occupancy was determined as a probabilistic function of the upper and left cells on the map. This yielded randomized maps containing partially-connected obstacles. Figures 2(a) and 2(b) show two examples of these maps. Start locations were uniformly randomly sampled from the free cells in the map, and goal locations were uniformly randomly sampled from a square region 60 cells wide, centered at the start location.

Robots executed motion plans over 80 discrete time steps. Each time step, the plans could instruct the robot to move one cell horizontally or vertically, or remain in place. Collisions were defined as multiple robots occupying the same map cell in a given time step, or multiple robots exchanging positions with one another over consecutive time steps. For this paper, an A* planner with a Manhattan distance heuristic was used, but the method could be adapted to other planning approaches such as D* [25], RRTs [26] or PRMs [24].



(a) Small map (b) Large map

Fig. 2. Two typical maps used in experiments. Circles denote robot start positions while stars denote goals. In the smaller map, dotted lines connect associated start and goal positions.



(a) Failure rates over team size (b) Failure rates over density

Fig. 3. Rates of failure of the sparse planners to converge to a collision-free solution in 20 iterations. A k-value of 2 ($\gamma = 2\sqrt{n}$ messages) is insufficient in most cases, but there is little difference between $k = 3$ and $k = 4$. Increasing either map density or team size dramatically increases failure rates in all cases.

Two problem sets were generated to test the performance of the planners, a team-size dataset and a map-density dataset. 15 problem instances were generated for each parameter set and tested using a centralized prioritized planner to verify that a solution existed and to establish a baseline for performance. In the first dataset, the number of robots in the team was varied between 40 and 240. In order to keep the problem difficulty similar across different team sizes, the maps were sized such that the density (the ratio of robots to map cells) was held constant at 0.125. In the second dataset, the number of robots was fixed at 240, while the density of the map varied between 0.125 and 0.03125. Descriptions of the two problem sets can be seen in Table I, along with some measures of map difficulty explained next.

TABLE I

PARAMETERS FOR THE TWO PROBLEM SETS USED IN EXPERIMENTS.

TEAM-SIZE DATASET						
Team size	40	60	80	120	160	240
Map density	0.125					
Map size	18	22	26	31	36	44
Avg. collisions	75.5	139.5	237.9	420.8	658.3	1050.9
Avg. blocked	30.6	47.4	67.1	104.2	143.4	214.9

MAP-DENSITY DATASET					
Team size	240				
Map density	0.03125	0.04166	0.06250	0.08266	0.1250
Map size	88	76	62	54	44
Avg. collisions	262.3	352.0	535.7	721.5	1030.4
Avg. blocked	145.0	161.6	184.6	201.7	215.3

A. Establishing the difficulty of the problem instances

A simple experiment was done to measure the difficulty of the generated path planning problem instances. For each problem instance, an independent planner coupled with a simple stop-and-wait reactive controller was tested on the map. Plans were generated independently for each robot and the number of collisions between these paths was computed. Then robots attempted to follow their paths, stopping if they detected that they were entering an occupied cell. This was iterated until all robots were either at their goals or blocked by another robot. In all problem instances, this reactive planning failed to get every robot to its destination. The number of collisions and the number of robots that ended up blocked were averaged for each parameter set, giving a relative measure of the difficulty of the problem instance. These results are included in Table I.

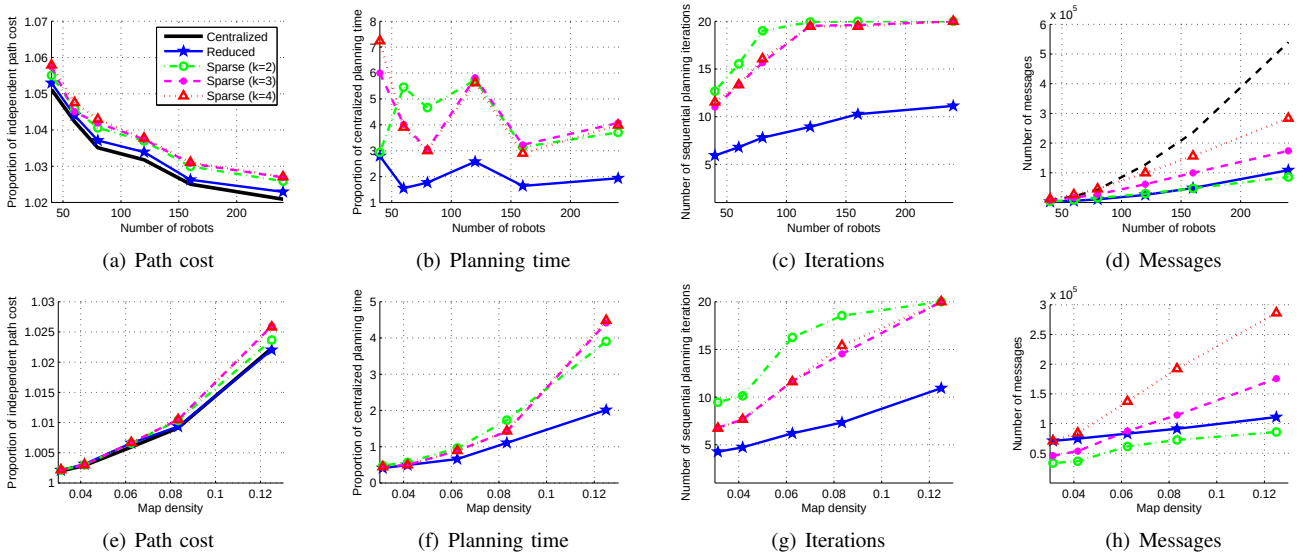
B. Selecting the prioritization

In this paper, initial local planning time is used as a prioritization for the robots. Previous work [1] showed that a path-length heuristic performed well in many environments, with the intuition that robots with shorter paths have time to move around other robots. The best-first nature of the underlying A* planner provides a similar intuition for using planning time. If the planner is taking a long time to search, it is because the most direct paths for the robot cannot be taken. A comparison of the paths generated by centralized prioritized planners using path length or planning time heuristic revealed similar performance. Variance in relative path cost between the two over both data sets was minimal ($\sigma^2 = 8.87 \times 10^{-5}$). However, the planning time of distributed planners can be improved with this time heuristic because their running time is limited by the speed of the slowest planning operation every iteration.

VI. RESULTS

The centralized, reduced distributed, and three sparse distributed planners were run on the two problem sets. The total path costs, planning time, number of iterations (for the distributed algorithms), and number of messages used were recorded for each of the problem instances. For planning time, the concept of *wall-clock time* was used to estimate running time in an actual team of n robots. Under wall-clock time, operations that can be parallelized, such as individual path planning in the distributed algorithms, are combined by taking the maximum of the running times, as this is the amount of time it would take a team of robots to compute these operations in parallel. The results over the two datasets can be seen in Figure 3 and Figure 4, with the first dataset (varying team size) on the top, and the second dataset (varying map density) on the bottom.

An iteration limit of 20 was imposed on all distributed planners, based on the performance of the complete distributed planner on the first dataset. Unfortunately, as can be seen in Figure 3, the sparse planners often failed to converge to a solution within this limit when the map was dense or



(a) Path cost (b) Planning time (c) Iterations (d) Messages

(e) Path cost (f) Planning time (g) Iterations (h) Messages

Fig. 4. Average results of the centralized and distributed planners on the team-size and map-density datasets. Results for the team-size dataset are displayed on the top row, while results for the map-density are on the bottom row. Each point represents 15 runs.

there were large numbers of robots. However, the reduced distributed planner did not have any failures in the problem instances. This is interesting as statistically, sparse planners with $k \geq 3$ detect well over 99% of trajectory collisions in the team. Indeed, failure rates are virtually identical between the $k = 3$ and $k = 4$ cases. This suggests that collision detection alone is insufficient to achieve good convergence. The reduced distributed planner differs in that it updates every robot with every relevant path, suggesting that some of these non-colliding paths are useful in preemptively avoiding collisions and significantly improving convergence. For the remainder of the results, non-converging sparse planning runs were included in the numerical results with the statistics taken when the planners were interrupted at 20 iterations. However, this means that the measurements in Figure 4 such as planning time, number of messages, and number of iterations are *underestimates*, as the actual performance might have been worse given additional iterations.

To ensure that efficient paths were being generated, the cumulative path costs of each solution was compared to that of a set of independently planned shortest paths for each robot from start to goal. In both of the problem sets, the cumulative path costs of the solutions (Figures 4(a) and 4(e)) found by the planners were less than 6% higher than the costs of the shortest paths, suggesting that the prioritized solutions were very efficient. As map density increased, there was a slight increase in the relative path costs of all the prioritized planners, as would be expected from an increase in planning difficulty. Interestingly, as team size increased, there was a slight decrease in the relative path cost for all the prioritized planners, suggesting that the planners were producing more optimal paths in larger teams.

Since all of the planners generated similarly efficient solutions, the next measure of performance was planning time. Ideally, the distributed planners, because they divided computational load over the team of robots, would be able to outperform centralized planning in large teams. However, as team size was varied, there were no clear trends in the planning times, and the distributed planners took longer than the centralized planner. Figure 4(b) shows the times as normalized by the wall-clock time taken by a centralized prioritized planner. Part of this result is related to the characteristics of the underlying path planner. The A* planner takes widely varying amounts of time on these maps in the best and worst cases. For example, in the first problem instance of 240 robots, A* takes 0.0047s for one robot and 4.4822s for another. While this means that the latter robot will be prioritized higher initially, this order of magnitude difference in planning times means that often, the distributed approaches are bounded in time by a few iterations in which they replan a path that takes A* a long time, which a centralized prioritized planner only has to plan once.

The results on the density varying dataset (Figure 4(f)) show a more clear trend. As density increases, relative planning time for the distributed planners also increases. At the lowest densities, the distributed planners are to resolve paths in less than half the time taken by the centralized planner. This shows that when the environment is sparse, the parallelization was more helpful. However, the high failure rate of the sparse planners can once again be clearly seen, as the planning times increase dramatically as density increases. The reduced distributed planner increases at a more controlled rate.

The potential for the distributed planners to reduce planning time, specifically the reduced distributed planner, is particularly evident when considering the number of iterations taken for convergence across problem sets, as seen in Figures 4(c) and 4(g). While the sparse planners are able to converge relatively quickly in small teams and low densities, they begin to take many more iterations as density and team size is increased. The reduced distributed planner, on the other hand, converges across all cases with a relatively

small number of iterations, considering the team sizes. In the 240-robot problem instances, (the right side of Figure 4(c) and all of Figure 4(g)), it finds a solution on average in fewer than 12 iterations, with its absolute maximum over all trials being 16 iterations. This compares to the 240 iterations required for a centralized prioritized planner, so if these communications and worst-case planner behaviors are acceptable, this approach may be useful.

Finally, the purpose of the optimizations introduced by the reduced and sparse distributed planners was to reduce the amount of necessary communications between robots. This was reflected in the total number of messages used by the various planners. For the team-size dataset (Figure 4(d)), a plot of the number of messages used by the complete distributed planner is shown as a worst-case upper bound. By design, the sparse planners follow curves that are proportional to $n\sqrt{n}$. More interestingly, however, the reduced distributed algorithm uses far fewer messages than the sparse planners. In fact, in a 240-robot team, it uses only 17% of the messages used by the complete distributed planner. This can be attributed to two factors: the fast convergence of the algorithm, and the ability to depend on cached messages. Because the reduced distributed planner takes so few iterations to converge, it is able to reduce its total number of messages. In addition, however, the reduced distributed planner can reduce messages through the assumptions that it makes at the start: that not receiving a message means that a robot has not changed its path, and that everyone is dependent unless proven otherwise. This performance trend is also found in the density dataset (Figure 4(h)), where the reduced distributed planner once again uses very few messages.

VII. CONCLUSIONS AND FUTURE WORKS

In this paper, we present two distributed variants of prioritized planning for use with teams containing hundreds of robots. An efficient, complete method is presented which is shown to converge to the centralized prioritized planner solution. Then, a sparse method is proposed in which robots discover collisions probabilistically. In moderately constrained environments, it is shown that these methods generate solutions of similar quality to the centralized alternative, while taking similar amounts of time. However, in less constrained environments, it is shown that these methods can find solutions in as little as half the time of full prioritized planning. Surprisingly, convergence to collision-free solutions is found to be highly sensitive to the communication of paths involved in future conflicts.

In the future, improved performance might be obtained with the use of a more efficient planning strategy than A*. As the presented algorithms spend most of their time re-solving paths to avoid new obstacles, incremental planners [27] or roadmaps [24] might tremendously improve running-time by reducing the time to replan. In addition, in real domains communication is unlikely to be lossless, and more work needs to be done to understand the impact of communications losses on performance.

VIII. ACKNOWLEDGMENTS

This research has been funded in part by the AFOSR MURI grant FA9550-08-1-0356. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

REFERENCES

[1] J. van den Berg and M. Overmars, "Prioritized motion planning for multiple robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005.(IROS 2005)*, 2005.

[2] C. Clark, S. Rock, and J. Latombe, "Motion planning for multiple mobile robots using dynamic networks," in *Proc. of ICRA*, 2003.

[3] L. Chun, Z. Zheng, and W. Chang, "A decentralized approach to the conflict-free motion planning formultiple mobile robots," in *Proc. of ICRA*, vol. 2, 1999.

[4] L. E. Parker, *Path Planning and Motion Coordination in Multiple Mobile Robot Teams*. Springer, 2009.

[5] D. Parsons and J. Canny, "A motion planner for multiple mobile robots," in *Proc. of ICRA*, 1990.

[6] J. Barraquand and J. Latombe, "Robot motion planning: A distributed representation approach," *IJRR*, vol. 10, no. 6, 1991.

[7] J. Schwartz and M. Sharir, "On the piano movers' problem: III. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers," *IJRR*, vol. 2, no. 3, 1983.

[8] P. Svestka and M. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, no. 3, 1998.

[9] K. Kant and S. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *IJRR*, vol. 5, no. 3, 1986.

[10] B. Aronov, M. De Berg, A. Van der Stappen, and P. Švestka, "Motion planning for multiple robots," *Discrete and Computational Geometry*, vol. 22, no. 4, 1999.

[11] Y. Guo and L. Parker, "A distributed and optimal motion planning approach for multiple mobile robots," in *Proc. of ICRA*, vol. 3, 2002.

[12] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *IJRR*, vol. 24, no. 4, 2005.

[13] P. O'Donnell and T. Lozano-Pérez, "Deadlock-free and collision-free coordination of two robotic manipulators," in *Proc. of ICRA*, 1989.

[14] R. Alami, F. Robert, F. Ingrand, and S. Suzuki, "Multi-robot cooperation through incremental plan-merging," in *Proc. of ICRA*, 1995.

[15] T. Simeon, S. Leroy, and J. Laumond, "Path coordination for multiple mobile robots: A resolution-complete algorithm," *IEEE Tran. on Robotics and Automation*, vol. 18, no. 1, 2002.

[16] M. Erdman and T. Lozano-Perez, "On Multiple Moving Objects," *Algorithmica*, vol. 2, 1987.

[17] C. Ferrari, E. Pagello, J. Ota, and T. Arai, "Multirobot motion coordination in space and time," *Robotics and Autonomous Systems*, vol. 25, no. 3-4, 1998.

[18] C. Warren, "Multiple robot path coordination using artificial potential fields," in *Proc. of ICRA*, 1990.

[19] K. Azarm and G. Schmidt, "Conflict-free motion of multiple mobile robots based ondecentralized motion planning and negotiation," in *Proc. of ICRA*, vol. 4, 1997.

[20] M. Bennewitz, W. Burgard, and S. Thrun, "Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots," *Robotics and Autonomous Systems*, vol. 41, no. 2-3, 2002.

[21] S. Buckley, "Fast motion planning for multiple moving robots," in *Proc. of ICRA*, 1989.

[22] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized Path Planning for Multiple Robots: Optimal Decoupling into Sequential Plans," in *Proc. of RSS*, 2009.

[23] M. Ryan, "Exploiting subgraph structure in multi-robot path planning," *Journal of Artificial Intelligence Research*, vol. 31, no. 1, 2008.

[24] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Tran. on Robotics and Automation*, vol. 12, no. 4, 1996.

[25] S. Koenig and M. Likhachev, "D* lite," in *Eighteenth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 2002.

[26] S. LaValle and J. Kuffner Jr, "Randomized kinodynamic planning," *IJRR*, vol. 20, no. 5, 2001.

[27] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy, "Incremental heuristic search in AI," *AI Magazine*, vol. 25, no. 2, 2004.