An Effective Algorithmic Framework for Near Optimal Multi-Robot Path Planning

Jingjin Yu[1] and Daniela Rus[2]

[1] Computer Science, Rutgers University, jingjin.yu@rutgers.edu
[2] CSAIL, Massachusetts Institute of Technology, rus@csail.mit.edu

Abstract. We present a centralized algorithmic framework for solving multi-robot path planning problems in general, two-dimensional, continuous environments while minimizing globally the task completion time. The framework obtains high levels of effectiveness through the composition of an optimal discretization of the continuous environment and the subsequent fast, near-optimal resolution of the resulting discrete planning problem. This principled approach achieves orders of magnitudes better performance with respect to both speed and the supported robot density. For a wide variety of environments, our method is shown to compute globally near-optimal solutions for 50 robots in seconds with robots packed close to each other. In the extreme, the method can consistently solve problems with hundreds of robots that occupy over 30% of the free space.

1 Introduction

We study the problem of planning collision-free paths for multiple labeled disc robots operating in two-dimensional, multiply-connected, continuous environments (*i.e.*, environments with holes). The *primary goal* of this work is to develop a practical, extensible framework toward the efficient resolution of multi-robot path planning (MPP) problems, in which the robots are densely packed, while simultaneously seeking to minimize *globally the task completion time*. The framework is composed of two key algorithmic components, executed in an sequential order. Using the example illustrated in Fig. 1(a), first, we compute the configuration space for a single robot, over which an optimal lattice structure is overlaid (Fig. 1(b)). Using the lattice structure as a roadmap, each start (resp., goal) location is assigned to a nearby node of the roadmap as its unique discrete start (resp., goal) node, which translates the continuous problem into a discrete one (Fig. 1(c)). Then, a state-of-the-art discrete planning algorithm is applied to solve the roadmap-based problem near-optimally (Fig. 1(d)). Through the tight composition of these two algorithmic components, our framework proves to be highly effective in a variety of settings, pushing the boundaries on optimal multi-robot path planning to new grounds in terms of the number of robots supported and the allowed robot density.

Related work. MPP finds applications in a wide spectrum of domains such as navigation [2, 26], manufacturing and assembly [14], warehouse automation [40], computer video games [27], and microfluidics [8]. Given the important role it holds in robotics-related applications, MPP problems has received considerable attention in robotics research with dedicated study on the subject dating back at least three decades [25], in

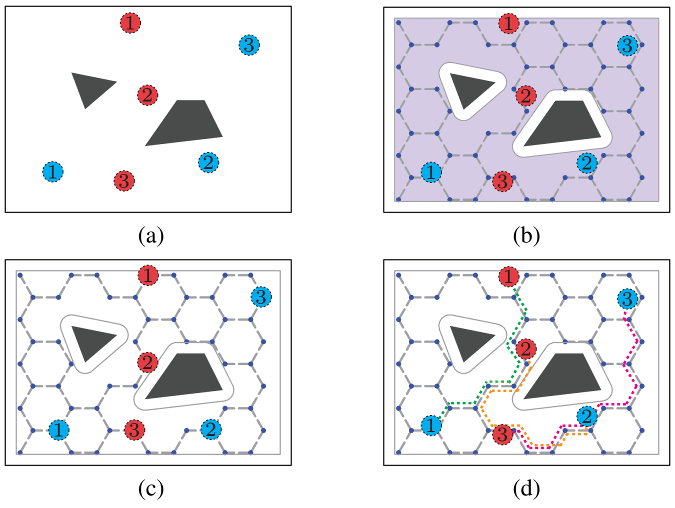

Fig. 1. An illustrative example of our algorithmic framework. a) A problem instance with three disc robots. The start and goal locations are indicated by the blue and red labeled discs, respectively. b) The configuration space (shaded area) for a single robot and the fitted hexagonal lattice. The blue circles are the start positions, and the red circles are the goal positions. c) The discrete abstraction of the original problem. d) Solution to the original continuous problem.

which a centralized approach is taken that considers all robots as a single entity in a high dimensional configuration space. Because the search space in such problems grows exponentially as the number of robots increases linearly, a centralized approach [25], although complete, would be extremely inefficient in practice. As such, most ensuing research take the approach of decomposing the problem. One way to do this is by assigning priorities to the robots so that robots with higher priority take precedence over robots with lower priority [4, 5]. Another often adopted partitioning method is to plan a path for each robot separately without considering robot-robot interaction. The paths are then *coordinated* to yield collision free paths [3, 20]. Following these initial efforts, the decomposition scheme is further exploited and improved [7, 18, 22, 34, 36, 37]. Many of the mentioned works also consider optimality in some form. We emphasize that, since finding feasible solution for MPP is already PSPACE-hard [9], *i.e.*, no polynomial-time complete algorithm may even exist for such problems unless P = PSPACE, computing globally near-optimal solution for a large number of robots is extremely challenging.

Recent years have witnessed a great many new approaches being proposed for solving MPP. One such method, reciprocal velocity obstacles [33, 35], which can be traced back to [12], explicitly looks at velocity-time space for coordinating robot motions. In [8], mixed integer programming (MIP) models are employed to encode the interactions between the robots. A method based on network-flow is explored in [10]. In [21], similar to our framework upon a first look, an A^*-based search is performed over a discrete roadmap abstracted from the continuous environment. However, the authors addressed a much narrower class of problems for which they can bound the computation cost but cannot guarantee the solution optimality. It is also unclear how the complex geometric problem of efficiently computing a discrete roadmap from the continuous environment is resolved in the paper. In [29], discrete-RRT (d-RRT) is proposed for the efficient search of multi-robot roadmaps. Lastly, as a special case of MPP in continuous domains, efficient algorithms are proposed [30, 32] for interchangeable robots (*i.e.*, in the end, the only requirement is that each goal location is occupied by an arbitrary robot). At the same time, discrete (*e.g.*, graph-based) MPP has also been a subject of active investigation. This line of research originates from the mathematical study of the 15-puzzle and related *pebble motion* problems [15, 39]. Since then, many heuristics augmenting the A^* algorithm have been proposed for finding optimal solution, *e.g.*, [24, 31, 38], to name a few. These heuristics essentially explore the same decoupling idea used in the continuous case to trim down the search space. A method based on network-flow also exists here [42]. Some of these discrete solutions, such as [15], have helped solving continuous problems [16, 28].

Contribution. Our work brings two contributions toward solving MPP effectively and optimally. First, we introduce a two-phase framework that allows any roadmap building (*i.e.*, discretization) method to be combined with any suitable discrete MPP algorithm for solving continuous MPP problems. The framework achieves this by imposing a partial collision avoidance constraint during the roadmap building phase while preserving path near-optimality. Second, we deliver a practical integrated algorithmic implementation of the two-phase framework for computing near optimal paths for a large number of robots. We accomplish this by combining *(i)* a fast algorithm for superimposing dense regular lattice structures over a bounded two-dimensional environment with holes and *(ii)* an integer linear programming (ILP) based algorithm for computing near-time-optimal solutions to discrete MPP [41]. To the best of our knowledge, we present the first such algorithm that can quickly plan near optimal, continuous paths for hundreds of robots densely populated in multiply-connected environments[3].

Paper organization. The rest of the paper is organized as follows. We formulate the MPP problem in Section 2. In Section 3, we describe the overall algorithmic framework architecture and the first component of the framework on roadmap-based problem construction. In Section 4, we describe how the second component of the framework may be realized. In Section 5, we demonstrate the effectiveness of our framework over a variety of environments. We hold an extensive discussion and conclude in Section 6.[4]

2 Problem Statement

Let $\mathscr{W}$ denote a bounded, open, multiply-connected (*i.e.*, with holes), two-dimensional region. We assume that the boundary and obstacles of $\mathscr{W}$ can be approximated using

[3] Warehousing systems from Kiva Systems[40] can work effectively with hundreds of robots. However, these robots essentially live on a grid within a structured environment.

[4] Due to limited space, detailed description of the ILP algorithm (Section 4) and larger versions of some figures are included in the online material available at `http://arxiv.org/abs/1505.00200`. An accompanying video demonstrating our algorithm and software developed in this paper are available from the corresponding author's website.

polygons with an overall complexity of m (*i.e*, there are a total of m edges). There are n unit disc robots residing in $\mathscr{W}$. These robots are assumed be omnidirectional with a velocity v satisfying $|v| \in [0,1]$. Let $\mathscr{C}_f$ denote the free configuration space for a single robot (the shaded area in Fig. 1(b)). The centers of the n robots are initially located at $S = \{s_1, \ldots, s_n\} \subset \mathscr{C}_f$, with goals $G = \{g_1, \ldots, g_n\} \subset \mathscr{C}_f$. For all $1 \leq i \leq n$, a robot initially located at s_i must be moved to g_i.

In addition to planning collision-free paths, we are interested in optimizing path quality. Our particular focus in this paper is minimizing the *global task completion time*, also commonly known as *makespan*[5]. Let $P = \{p_1, \ldots, p_n\}$ denote a feasible path set with each p_i a continuous function, defined as

$$p_i : [0, t_f] \to C_f, p_i(0) = s_i, p_i(t_f) = g_i.$$

The *makespan* objective seeks solutions that minimize t_f. In other words, let $\mathscr{P}$ denote the set of all solution path sets, the task is to find a path set with t_f close to

$$t_{min} := \min_{P \in \mathscr{P}} t_f(P). \tag{1}$$

We emphasize that the aim of this work is a method for quickly solving "typical" problem instances with many robots and high robot density (*i.e.*, the ratio between robot footprint and the free space is high) with optimality assurance. By *typical*, we mean that: *(i)* the start location and goal locations are reasonably separated, *(ii)* a start or goal location is not too close to static obstacles in the environment, and *(iii)* there are no narrow passages in the environment that cause the discretized roadmap structure to have poor connectivity. More formally, we assume that assumptions *(i)* and *(ii)*, respectively, take the forms [6]

$$\forall 1 \leq i, j \leq n, \quad |s_i - s_j| \geq 4, \; |g_i - g_j| \geq 4 \tag{2}$$

and

$$\forall p \in \{S \cup G\}, \quad |p - q| \leq \sqrt{5} \Rightarrow q \in \mathscr{W}. \tag{3}$$

For *(iii)*, the discretized roadmap should capture the topology of the continuous environment well. To be more concrete, see Figure 2(a). In this environment, there are two holes. The lattice graph, after contraction of faces that do not contain any obstacles, does not have any holes. We expect the discrete roadmap to be connected and have number of holes (after face contraction) equal to the number of holes of the continuous environment (*e.g.*, Fig. 1(d)).

Remark. We provide these assumptions only to suggest situations in which our framework is expected to perform well. In our evaluation, these assumptions are not enforced. We in fact greatly relax (2) (from 4 to 2.5) and do not enforce (3) at all. We also give an efficient subroutine for restoring connectivity when assumption *(iii)* is not

[5] Note that our algorithmic framework also applies to other time- and distance-based optimality objectives through the use of an appropriate discrete planning algorithm.

[6] (2) and (3) are unit-less given the unit disc robot assumption. If the robots have radius r, the right side of the inequalities from (2) and (3) should be scaled by a multiplicative factor of r.

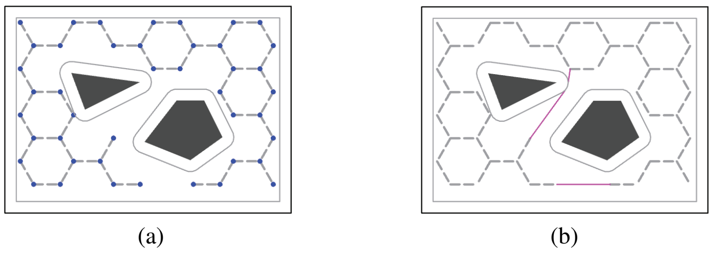
Fig. 2. a) An environment with a discretization that does not capture its original topology. b) The roadmap after restoring connectivity (the operations are performed automatically from our code), which then captures the topology of the original environment.

satisfied. For example, the routine, when applied to the example in Fig. 2(a), yields the result in Fig. 2(b), which is a screen capture from our program. We also emphasize that, given that optimal MPP is an extremely challenging task computationally [9] and our focus on method effectiveness, we do not consider the problem from the angle of solution completeness.

3 Algorithmic Framework Architecture and Roadmap-Based Discrete Problem Construction

We solve the proposed problem using an algorithmic framework with two algorithmic components–discretization of the continuous problem followed by resolution of the roadmap-based problem. The overall framework contains four sequential procedures:

(i) select and overlay a regular lattice structure over the configuration space,
(ii) restore environment connectivity lost in the discretization process,
(iii) snap start and goal locations to roadmap nodes to create a discrete problem on the roadmap, and
(iv) solve the discrete MPP problem optimally or near-optimally.

We note that, when compared with motion planning methods such as PRM [13] and RRT [17], our framework, looking somewhat similar on the surface, is in fact rather different. In methods like PRM and RRT, the discretization deals with the configuration space encompassing all degrees of freedom of the target system. Our approach, on the other hand, performs a careful, mostly uniform discretization of the configuration space for a single robot with two degrees of freedom. In doing so, we trade probabilistic completeness for the faster computation of near-optimal solutions. In the rest of this section, we describe the first key component of our algorithmic framework–the construction of the roadmap-based discrete problem, which subsumes the first three algorithmic procedures of the overall framework.

3.1 Lattice Selection and Imposition

Appropriate lattice structure selection In selecting the appropriate lattice structure, we aim to allow the packing of more robots simultaneously on the resulting roadmap and obtain the structure fast. Clearly, if an insufficient number of nodes exists in the roadmap, the resulting discrete problem can be crowded with robots, which is difficult to solve and may not even have a solution. On the other hand, to allow a clean separation between the roadmap building phase and the discrete planning phase of the framework, the nodes cannot be too close to each other, *e.g.*, two robots occupying two different nodes should not be in collision. Moreover, it is desirable that two robots moving on different edges in parallel will not collide with each other.

Considering all these factors together, we resort to adopting uniform tilings of the plane [23]. A uniform tiling of the plane is a regular network structure that can be repeated infinitely to cover the entire two-dimensional plane. Due to the regularity of uniform tilings, it is computationally easy to overlay a tiling pattern over $\mathscr{C}_f$. Choosing such a tiling then relieves us from selecting each node for the roadmap individually. Over the 11 uniform tilings[7] of the plane [23], we computed the density of robots supported by each. To allow concurrent moves of robots on nearby edges, take square tiling as an example, a square must have a side length of $4/\sqrt{2}$ to avoid potential collision incurred by such moves (see, *e.g.*, Fig. 3(a)). Indeed, it is straightforward to show that the closest inter-robot distance is reached when two robots are in the middle of two edges connecting to the same node. For hexagonal tilings, this results in a minimum side length of $4/\sqrt{3}$ (Fig. 3(b)).

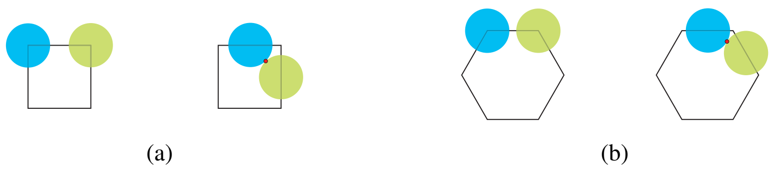

Fig. 3. Minimum distance between robots. To ensure no collision when executing a discrete plan, the distance between two lattice nodes must be $4/\sqrt{2}+\varepsilon$ for square tilings (a) and $4/\sqrt{3}+\varepsilon$ for hexagonal tilings (b). At exactly $4/\sqrt{2}$ (resp. $4/\sqrt{3}$) the robots will touch when reaching the midpoint of the edge. The contact point is shown as a red doc in both figures.

After obtaining the required side length parameters for all 11 tilings, the maximum robot density allowed by these tilings can then be computed. We compute the density by assuming that all nodes of the regular tiling patterns are occupied by robots and compute the ratio between the area occupied by robots and the free space when it is unoccupied. For an infinite lattice with no obstacles, the hexagonal tiling is the best with about 45% density, followed by the square tiling with roughly 39% density. Triangular tilings have

[7] These tilings are: triangular, trihexagonal, square, elongated triangular, hexagonal, truncated square, truncated trihexagonal, truncated hexagonal, snub square, rhombitrihexagonal, snub hexagonal.

a density of only 23%. This leads us to choose hexagonal lattices as the base structure of the discrete roadmap.

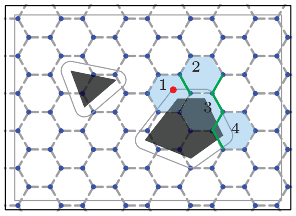

Fig. 4. Efficient computation of the hexagonal lattice that falls inside $\mathscr{C}_f$.

Imposing the lattice structure After deciding on the lattice structure, we need a procedure for imposing the structure on $\mathscr{C}_f$. Essentially, every edge must be checked to determine whether it is entirely contained in the free configuration space $\mathscr{C}_f$. Note that if this is performed naively, *i.e.*, performing collision checking of each edge with all obstacles, the overall complexity is on the order of $O(mA)$, in which m is the complexity of the workspace and A is the area contained in the outer boundary. The naive approach quickly becomes time-consuming as either m or A grows.

To complete this step efficiently, we start by making an arbitrary alignment between a sufficiently large piece of the infinite hexagonal lattice and the continuous environment (Fig. 4). Then, we look at one C-space obstacle (including the outer boundary) at a time. For each obstacle, we pick an arbitrary vertex on the boundary (red dot in Fig. 4) and locate the hexagon from the lattice it belongs to (in case of the example in Fig. 4, the shaded hexagonal with the label “1”). We then follow the obstacle boundary and find all (green) edges of the lattice that intersect the boundary. The edges found this way do not belong to $\mathscr{C}_f$ and the final discrete graph structure; moreover, they partition the lattice into pieces that are either completely inside $\mathscr{C}_f$ or completely outside $\mathscr{C}_f$. This allows us to efficiently check whether the rest of the lattice edges belong to $\mathscr{C}_f$. To do so, we start with a vertex that is within $\mathscr{C}_f$ that also belongs to one of these green edges and perform a breath first search over the lattice structure, now with all the green edges deleted. All edges found this way must be long to $\mathscr{C}_f$. We repeat this until all vertices of the lattice that fall inside $\mathscr{C}_f$ are exhausted. Note that this BFS is a discrete search without performing geometric computation over real numbers, which can be done much faster than edge intersection checks. In the end, we obtain an output sensitive algorithm that typically takes time between $\Theta(\sqrt{A})$ and $\Theta(A)$, depending the total length of obstacle boundaries. In practice, using the said method, the computation time used by this step is trivial in comparison to the time it takes to do the discrete planning.

Restore Configuration Space Connectivity We now address how we may ensure that the topology of $\mathscr{C}_f$ is preserved in the discrete roadmap. Essentially, we must locate places where connectivity in the continuous environment is lost. We illustrate our algorithmic solution for doing so using an example. For the problem given in Fig. 3(a), for each C-space obstacle, it is straightforward to obtain the smallest cycle on the lattice enclosing the obstacle (*e.g.*, the green and red cycles in Fig. 5). Then, for each pair of obstacles, we check whether the corresponding enclosing cycles share non-trivial interior and if so, locate a minimum segment on the overlapping section (em e.g., the red segments between the two orange nodes in Fig. 5). Using *visibility graph* [19], we may then restore the lost connectivity and obtain the roadmap shown in Fig. 3(b). Most of the computation time in this step is spent on computing the visibility graph itself, which takes time $O(m \log m + E)$ [6], with m being the complexity of the environment and E being the number of edges in the resulting visibility graph.

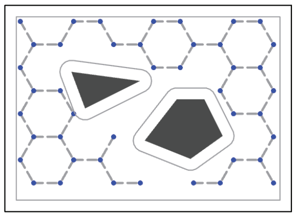

Fig. 5. Smallest cycles fully surrounding the two $\mathscr{C}_f$ obstacles.

Remark. In the process of restoring connectivity, it is possible that the resulting roadmap cannot guarantee that simultaneous movements of disc robots are collision-free. Without getting into details, we mention that this issue can be fully addressed by sacrificing some time optimality.

We also note that the preservation of the connectivity or topology of the continuous environment can be crucially important. A better connected environment has a more diverse set of candidate paths, making the resulting problem easier to solve. Perhaps more importantly, the preservation of the connectivity of $\mathscr{C}_f$ is essential to preserving path optimality. For a roadmap built from an overlaid square lattice, given a shortest path $p \subset \mathscr{C}_f$ between two points, due to the *strong equivalence* between the Euclidean metric and the Manhattan metric, the shortest path p and the corresponding shortest path p' on the square lattice-based roadmap are within a constant factor multiple of each other for any reasonably long path p (that is, $length(p) \ll 1$ does not hold). The same argument applies to the roadmap-based hexagonal lattices. Without obstacles, the ratio $length(p')/length(p)$ over a long path p is bounded by $\sqrt{2}$ for square lattices and roughly the same for hexagonal lattices. The ratio is largely the same when obstacles are present. On the other hand, if the connectivity of $\mathscr{C}_f$ is not preserved, then it becomes possible that $length(p')/length(p)$ is arbitrarily large. An example is given in Fig. 6.

Once we establish that the roadmap preserves the near-optimality on path length, the same applies to time optimality. Given the preservation of near-optimality of individual paths, it does not directly imply that an optimal solution to the abstracted discrete problem also preserves optimality with respect to the original continuous problem, in terms of time or distance. However, our computational experiments show that this is generally the case when $\mathscr{C}_f$ has good connectivity.

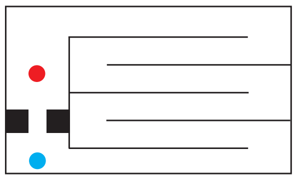

Fig. 6. Suppose that the start and goal locations are at the center of the blue and the red discs, respectively. If the robot does not find the narrow passage on the left, it then needs to travel through a winding path on the right. By extending the width of the environment, we can make the winding path arbitrarily long when compared to the shortest path.

3.2 Snapping Start and Goal Locations to Roadmap Nodes

After the full roadmap is built, each start or goal location in $S \cup G$ must be associated with a nearby roadmap node. We call this process *snapping*. For the snapping step, for each $s_i \in S$, we simply associate s_i with the closest roadmap node that s_i can reach without colliding with another $s_j \in S$. The same process is performed for all $g_i \in G$ With the separation assumptions (2) and (3), this is almost always possible. In particular, (2) implies that each hexagon from the lattice contains (roughly) at most one start and one goal location. Therefore, the number of nodes on the roadmap is at least twice the number of robots. In rare cases when conflicts do happen, we may apply the rearrangement algorithms (*e.g.*, [30]) to perform the snapping step without incurring much penalty on time optimality. The completeness of this step is guaranteed by (2) and (3).

With the snapping process complete, a discrete abstraction of the original continuous problem is obtained. For our example, this leads to the scenario captured in Fig. 1(c). If we are not interested in optimality, the discrete problem may be attempted using a non-optimal but polynomial time algorithm [15, 44]. As stated in the individual subsections, the computation required in this section can be carried out using low-degree polynomial time algorithms. The relative time used for this portion is trivial as compared to the time required for solving the roadmap-based discrete problem.

4 Fast, Near-Optimal Discrete Path Planning

After a high quality roadmap is obtained with near-optimality guarantees on time and distance (*e.g.*, an optimality-preserving reduction from continuous space to discrete space), one may then freely choose an algorithm for finding solutions to the discrete abstraction (Fig. 1(c) in our example). Whereas an arbitrary number of globally optimal objectives can be conjured, four objectives are perhaps most natural. These four objectives minimize the maximum or the total *arrival time* or *travel distance*. Viewing from the angle of service provider (*e.g.*, delivery drones) and end user (*e.g.*, customers), minimizing the total distance or time allows the service provider to minimize energy cost or overall vehicle fleet usage. On the other hand, minimizing the maximum time or distance promises a more uniform service quality among customers. If minimizing the total arrival time or the total distance is the goal, then discrete search methods such as

ID [31] can be applied. Here, we focus on the minimum *makespan* (*i.e.*, maximum arrival time or task completion time). We describe an effective method for minimizing the makespan [42, 41], which is also a good proxy to minimizing the maximum travel distance. The method is an ILP-based one with an optimal baseline algorithm, augmented with near-optimal heuristics to improve the computational performance.

The baseline, ILP model-based algorithm We first describe how an ILP model is obtained [42]. The key idea is to perform *time expansion* over the discrete (spatial) roadmap and then build the ILP model over the resulting directed *space-time* graph. This step essentially sequentially chains some T copies of the spatial roadmap together. Locally, for a hexagonal roadmap, the space-time graph has the structure given in Fig. 7. Now, if the discrete MPP allows a solution within T time steps, then there is a corresponding solution on the space-time graph in the form of *n vertex disjoint paths*. An ILP model can then be readily build to find these vertex disjoint paths. Each solution found on the space-time graph can be easily mapped back to yield a feasible solution to the original MPP problem. To ensure optimality, a conservative initial estimate of T, the number of steps for doing time expansion, is used. This T is then gradually increased until a first feasible solution is found, which is also a minimum makespan solution.

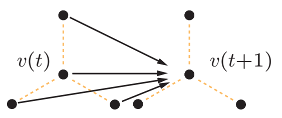

Fig. 7. In a single time expansion step, a node's neighbors (including the node itself) at time step t are connected to the node at time step $t+1$.

***k*-way split heuristic** As finding minimum makespan solutions to discrete MPP is NP-hard [43], we observe the time in solving the corresponding ILP models grows exponentially as the size of the model grows. This lead us to a heuristic that breaks a large ILP model into multiple small ones along the *time line*. If the problem is broken in k pieces, we call it a k-way split. Using 2-way split as an example, first, individual paths for the robots are computed . Then the mid-point of these paths are used to divide the discrete MPP problem into two sub-problems, with these mid-points serve as the goals of the first sub-problem and the start locations of the second sub-problem. If there are mid-points that overlap, randomization is used to find alternative locations. Last, each sub-problem is solved individually, after which we stitch the solutions together. Note that, because the division is over time, there are no interactions between two sub-problems. In a k-way split, the original ILP model is effectively divided into k equal sized pieces. Solving these k pieces is usually much less time consuming than solving the single, larger ILP model. The heuristic, however, does not preserve true optimality on makespan but rather yields near-optimal solutions.

Reachability analysis Another useful, optimality preserving heuristic is reachability analysis. The basic idea here is to truncate edges from the space-time graph that are unreachable, based on the start and goal locations of each individual robot.

5 Computational Evaluation

We implemented the roadmap building phase in C++ using CGAL [1]. The discrete path planning module, written in Java, uses Gurobi [11] as the ILP solver. The experiments were carried out on an Intel i7-4850HQ laptop PC.

For evaluation, we tested of our algorithmic framework over five distinct environments. The first one is a simple square with a side length of 35 (recall that the robots are unit discs), with no internal obstacles. The rest of the environments have the same bounding square but contain different obstacle setups. We randomly select start and goal locations for all our tests. These environments, along with a typical 50-robot problem instance, are illustrated in Fig. 8.

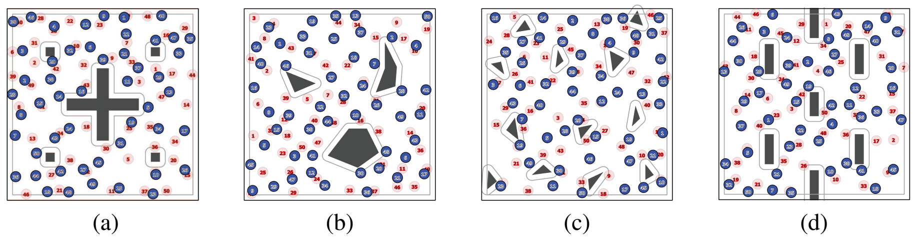

Fig. 8. Environments with obstacles and 50 start and goal locations. The labeled blue discs mark the start locations and the labeled pink discs mark the goal locations. Zoom-in on the digital version of the paper for details. a) Plus. b) (Halloween) Jack. c) Triangles. d) Bars.

5.1 Performance in Bounded, Obstacle-Free Environment

We first characterize how our framework performs in terms computation speed and solution optimality, as k-way split heuristic is used with different values of k. For this task, we carry out two sets of computations. The first set, covered in this subsection, focuses on bounded, obstacle-free environment. For this environment, we let the number of robots vary between 10 to 100 and evaluate the performance of the framework with the baseline algorithm (*i.e.*, a single sub-problem), 2-way split (*i.e.*, two sub-problems), 4-way split, and 8-way split. For each choice of the number of robots and the heuristic, 10 test cases are randomly generated sequentially and solved. The average running time and optimality ratio is plotted in Fig. 9. Note that our computation of the optimality ratio is conservative. To compute this ratio, we find the shortest distance between each pair of start and goal locations and use the maximum of these distances as the estimate of optimal time (since the robot has maximum speed of 1). We then obtain the optimality ratio by dividing the actual task completion time by the estimated value.

From the experiments, we observe that the baseline algorithm actually performs quite well for up to 40 robots in the absence of obstacles. With that said, both 2-way and 4-way splits do much better without losing much optimality–all three achieves optimality ratio between 1.2 to 1.6 in our experiments. With the 8-way split, sacrificing some

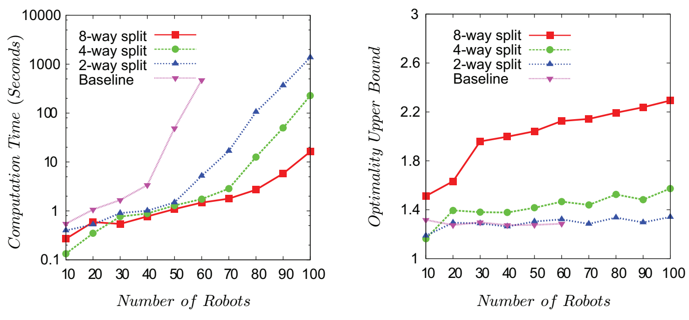

Fig. 9. Performance of framework with various choices of heuristics for a square environment without internal obstacles. [left] Computation time. [right] Optimality ratio.

optimality, we were able to consistently solve problems with 100 robots in 10 seconds on average. Such settings correspond to robots occupying over 25% of the free space, a setting that has never been attempted before in optimal multi-robot path planning. With 8-way split, problems with 125 robots in the same environment, which corresponds to a robot density over 31.4%, can be comfortably solved in about 15 minutes. We note that, if robot density is around 20%, our method can readily solve problems with over 300 robots (in a larger environment).

5.2 Performance in Bounded Environment with Obstacles

The second set of experiments shifts the focus to an environment with obstacles. For this we use the "Jack" environment. We choose this environment because it is in fact a relatively difficult setting as many shortest paths have to pass through the middle, causing conflicts. The experimental result, for 5 to 50 robots, is plotted in Fig. 10, which is consistent with our first set of experiments. We note that obstacles, while affecting the computation time, do not heavily impact the optimality of the result.

5.3 Evaluation of Overall Framework Performance

Our last set of experiments is aimed at showing the overall effectiveness of our framework. For this purpose we select the splitting heuristic automatically. Roughly, we do this by increasing k (in a k-way split) to keep each time expansion with 10 time steps, which we have found to strike a good balance between speed and optimality. For the set of environments illustrated in Fig. 8, the experimental result is plotted in Fig. 11. Our method is able to consistently solve all instances with an average solution time from 0.5 to 10 seconds while providing good optimality assurance on minimum makespan. The two spikes in Fig. 11(a) at 40 robots are due to the switching to 8-way split at 45 robot for these two environments.

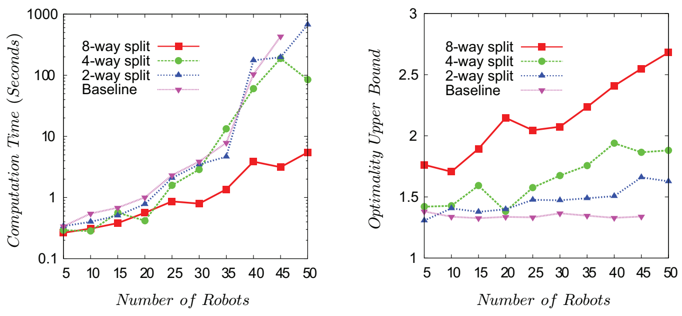

Fig. 10. Performance of our algorithmic framework with various choices of heuristics for the "Jack" environment. [left] Computation time. [right] Optimality ratio.

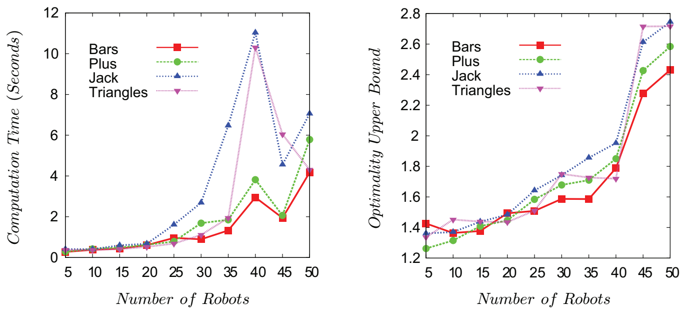

Fig. 11. Performance of the overall framework in a wide variety of environments. [left] Computation time. [right] Optimality ratio.

6 Conclusion

In this paper, we present an algorithmic framework for tackling the multi-robot path planning problem in continuous, multiply-connected environments. Our framework partitions the planning task into two phases. In the first phase, the configuration space is tiled with a carefully selected regular lattice pattern, taking into account robot-robot collision avoidance. The imposed lattice is then processed to yield a roadmap that preserves the connectivity of the continuous configuration space, which is essential for achieving near optimality in the final solution. Snapping the robots and their goal locations to the roadmap then transforms the initial continuous planning problem to a discrete planning problem. In the second phase, the discrete planning problem can be solved using any graph-based multi-robot path planning algorithms, after which the solution can be readily used in continuous domains. With a good optimal planner for discrete MPP, our overall algorithm can consistently solve large problem instances with tens to hundreds of robots in seconds to minutes.

As we make an important first step here toward a generic framework for near-optimal multi-robot path planning in continuous domains with obstacles, we also bring about many natural next steps. We discuss a few of these here, which we plan to fully explore in our future research.

Nonholonomic constraints. An important issue not addressed in this paper is path planning for nonholonomic robots. We briefly touch upon this issue here. Our algorithmic framework supports quite naturally nonholonomic robots that are small-time locally controllable (STLC) with reasonable minimum turning radius. Essentially, to apply our method to a nonholonomic robot, the robot only need the capability to: *(i)* move from its start location to a nearby roadmap node with a given orientation, *(ii)* trace any path on the roadmap without incurring collision, and *(iii)* move from a roadmap node to a nearby goal location (with an arbitrary orientation). A car-like robot, or any robot that is STLC, possesses the first and the third capabilities. Then, as long as the robot has a minimum turning radius of 2, it can follow any path on a hexgonal lattice without violating its nonholonomic constraints (see Fig. 12). More importantly, multiple robots may move concurrently in such a manner without causing collisions. The introduction of nonholonomic constraints does not significantly affect optimality.

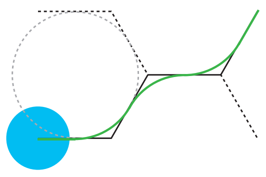

Fig. 12. A car-like robot with a mininum turning radius of 2 can trace any given path on a hexagonal lattice with side length $4/\sqrt{3}$ without violating its nonholonomic constraints or colliding with other robots.

Decentralized planner. The current implementation of our framework yields a centralized algorithm. It is possible, however, to make the algorithm decentralized at the global scale. For example, we may simply let each robot perform planning individually using a method such as reciprocal velocity obstacle (RVO) based algorithm and engage locally our centralized method as the density of robots surpass some critical threshold. Note that, as the density of robots increases, RVO-based or repulsion-force-based methods generally do not have optimality guarantees and may also create deadlocks.

Optimality of hexagonal lattice in general environments. While we have shown that a hexagonal lattice structure yields the optimal tiling in the absence of obstacles, it is unclear whether this holds well when there are obstacles in the bounded environment. In future work, we plan to study this through simulation under various obstacle settings. We will also characterize the performance using lattice structures other than hexagonal ones. The reason behind this is that, although hexagonal lattice allows the highest density, each node is only 3-connected. Square lattices, for example, has a 4-connected structure, which facilitates the discrete planning phase. Generally, discrete MPP problems with higher connectivity are easier to optimally solve.

Acknowledgment This work was supported in part by ONR projects N00014-12-1-1000 and N00014-09-1-1051, and the Singapore-MIT Alliance on Research and Technology (SMART) Future of Urban Mobility project. We are grateful for the funding support that we receive. We also thank the reviewers for their insightful comments that helped significantly improve the quality of the paper.

References

1. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.
2. J. Alonso-Mora. *Collaborative Motion Planning for Multi-Agent Systems*. PhD thesis, ETH Zurich, March 2014.
3. Z. Bien and J. Lee. A minimum-time trajectory planning method for two robots. *IEEE T. Robot. Autom.*, 8(3):414–418, 1992.
4. S. J. Buckley. Fast motion planning for multiple moving robots. In *ICRA*, 1989.
5. M. A. Erdmann and T. Lozano-Pérez. On multiple moving objects. In *ICRA*, 1986.
6. S. K. Ghosh, A. Maheshwari, S. P. Pal, S. Saluja, and C. V. Madhavan. Characterizing and recognizing weak visibility polygons. *Comput. Geom.*, 3(4):213–233, 1993.
7. R. Ghrist, J. M. O'Kane, and S. M. LaValle. Computing Pareto Optimal Coordinations on Roadmaps. *IJRR*, 24(11):997–1010, 2005.
8. E. J. Griffith and S. Akella. Coordinating multiple droplets in planar array digital microfluidic systems. *IJRR*, 24(11):933–949, 2005.
9. J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-Hardness of the "Warehouseman's Problem". *IJRR*, 3(4):76–88, 1984.
10. A. F. van der Stappen I. Karamouzas, R. Geraerts. Space-time group motion planning. In *WAFR*, 2012.
11. Gurobi Optimization Inc. Gurobi optimizer reference manual, 2015.
12. K. Kant and S. Zucker. Towards efficient trajectory planning: The path velocity decomposition. *IJRR*, 5(3):72–89, 1986.
13. L. E. Kavraki, P. Svestka, J-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE T. Robot. Autom.*, 12(4):566–580, 1996.
14. R. A. Knepper and D. Rus. Pedestrian-inspired sampling-based multi-robot collision avoidance. In *RO-MAN*, 2012.
15. D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS*, 1984.
16. Athanasios Krontiris, Qandeel Sajid, and Kostas Bekris. Towards using discrete multiagent pathfinding to address continuous problems. *AAAI-WoMP*, 2012.
17. S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, Oct 1998. Computer Science Department TR 98-11.
18. S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE T. Robot. Autom.*, 14(6):912–925, December 1998.
19. T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.
20. P. A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *ICRA*, 1989.
21. M. Peasgood, C. Clark, and J. McPhee. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE T. Robot.*, 24(2):283–292, 2008.

22. J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. In J.-D. Boissonat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algo. Found. Robot. V*, pages 221–237. Springer-Verlag, Berlin, 2002.
23. W. Robert. *The Geometrical Foundation of Natural Structure. A Source Book of Design*. Dover, 1978.
24. M. R. K. Ryan. Exploiting subgraph structure in multi-robot path planning. *J. Arti. Intel. Res.*, 31:497–542, 2008.
25. 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*, 2(3):46–75, 1983.
26. J. Snape, S. J. Guy, J. van den Berg, and D. Manocha. Smooth coordination and navigation for multiple differential-drive robots. In *Exp. Robot.* Springer, 2014.
27. J. R. Snape. *Smooth and collision-free navigation for multiple mobile robots and video game characters*. PhD thesis, University of North Carolina at Chapel Hill, 2012.
28. K. Solovey and D. Halperin. *k*-color multi-robot motion planning. In *WAFR*, 2012.
29. K. Solovey, O. Salzman, and D. Halperin. Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. In *WAFR*, 2014.
30. K. Solovey, J. Yu, O. Zamir, and D. Halperin. Motion planning for unlabeled discs with optimality guarantees. In *RSS*, 2015.
31. T. Standley and R. Korf. Complete algorithms for cooperative pathfinding problems. In *IJCAI*, 2011.
32. M. Turpin, N. Michael, and V. Kumar. Concurrent assignment and planning of trajectories for large teams of interchangeable robots. In *ICRA*, 2013.
33. J. van den Berg, M. C. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*, 2008.
34. J. van den Berg and M. Overmars. Prioritized motion planning for multiple robots. In *IROS*, 2005.
35. J. van den Berg, J. Snape, S. J. Guy, and D. Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *ICRA*, 2011.
36. J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *RSS*, 2009.
37. P. Švestka and M. H. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23(3):125–152, 1998.
38. G. Wagner and H.. Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *IROS*, 2011.
39. R. M. Wilson. Graph puzzles, homotopy, and the alternating group. *J. Combinat. Theo. (B)*, 16:86–96, 1974.
40. P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Mag.*, 29(1):9–19, 2008.
41. J. Yu and S. M. LaValle. Fast, near-optimal computation for multi-robot path planning on graphs. In *AAAI*, 2013. Late breaking papers.
42. J. Yu and S. M. LaValle. Planning optimal paths for multiple robots on graphs. In *ICRA*, 2013.
43. J. Yu and S. M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, 2013.
44. J. Yu and D. Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning. In *WAFR*, 2014.