Multi-agent Path Planning and Network Flow

Jingjin Yu and Steven M. LaValle

Abstract This paper connects multi-agent path planning on graphs (roadmaps) to network flow problems, showing that the former can be reduced to the later, therefore enabling the application of combinatorial network flow algorithms, as well as general linear program techniques, to multi-agent path planning problems on graphs. Exploiting this connection, we show that when the goals are permutation invariant, the problem always has a feasible solution path set with a longest finish time of no more than $n+V-1$ steps, in which n is the number of agents and V is the number of vertices of the underlying graph. We then give a complete algorithm that finds such a solution in $O(nVE)$ time, with E being the number of edges of the graph. Taking a further step, we study time and distance optimality of the feasible solutions, show that they have a pairwise Pareto optimal structure, and again provide efficient algorithms for optimizing each of these practical objectives.

1 Introduction

Consider the problem illustrated in Fig. 1, which inspired the authors to pursue this research. As an exercise (26-1 in [8]), the *escape problem* is to determine, given $m \leq n^2$ evaders placed on m different points of an $n \times n$ grid, whether there are m vertex disjoint paths from these m locations to m different points on the boundary of the grid. Intended as a demonstration of applications of *maximum flow* algorithms (Ch. 26 of [8]), it undoubtedly mimics multi-agent[1] path planning problems on graphs. Intrigued by the elegant network flow based solution to the escape problem, we wonder: How tightly are these two classes of problems intertwined and how we may take advantage of the relationship?

Jingjin Yu
Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, e-mail: jyu18@uiuc.edu

Steven M. LaValle
Department of Computer Science, University of Illinois, Urbana-Champaign, e-mail: lavalle@uiuc.edu

[1] We use *agent* instead of *robot* since the method applies to scenarios such as evacuation planning.

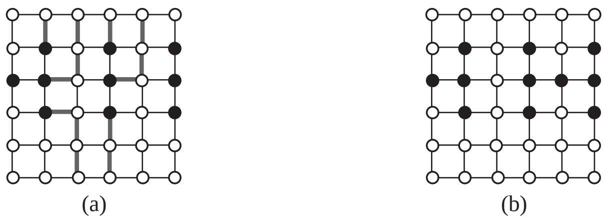


Fig. 1 Examples of the escape problem on a 6×6 grid. The black discs are the initial evader locations. The goal is to plan disjoint paths for the evaders to reach different vertices on the boundary of the grid. a) An instance with solution given as the bold edges. b) An instance without a solution.

In this paper, we explore and exploit the connection between multi-agent path planning on *collision-free unit-distance graphs* (or CUGs, see Section 2 for the definition) and network flow. We begin by showing that multi-agent path planning on CUGs is closely related to a class of problems called *dynamic network flow* or *network flow over time*. We then focus on the permutation invariant multi-agent path planning problem on CUGs (by permutation invariant, we mean that goals are not pre-assigned to agents. Instead, we only require that each goal is reached by a unique agent), establishing that such problems always have solutions. To solve the problem algorithmically, an adapted maximum flow algorithm is provided which plans collision free paths for all agents with worst time complexity $O(nVE)$, in which n is the number of agents, V is the number of vertices of the CUG and E is the number of edges of the CUG. Moreover, we guarantee that the last agent takes time no more than $n+V-1$ to reach its goal, assuming that agents travel at unit speed. Next, we construct efficient algorithms for obtaining temporally and spatially optimal solutions. For example, our algorithm for shortest overall time has running time $O(nVE\log V)$. We also show that these temporal and spatial objectives cannot be optimized simultaneously (i.e., they have a *Pareto optimal* structure).

As a universal subroutine in multi-agent systems, collision-free path planning for multiple agents finds applications in tasks spanning assembly [19, 33], evacuation [5, 40], formation control [3, 38, 42, 44, 45], localization [15], object transportation [31, 41], search and rescue [21], and so on. Given its importance, path planning for multi-agent systems has remained as a subject of intense study for many decades. Due to the vast size of the available literature, we only mention a most related subset of the research in this field and refer the readers to [6, 26, 28] and the references therein for a more comprehensive review of the subject.

When all agents are treated as a single agent with a high dimensional configuration space, the problem can be solved using cylindrical algebraic decomposition [7] or Canny's roadmap algorithm [4], in theory. Such *coupled* approaches suffer from the curse of dimensionality; even when sampling based methods [23, 27] are used, instances involving only a small number of agents can be computationally challenging. This difficulty prompts the study of methods that seek to explore local features whenever possible to avoid working with too many agents at a time. Among these, *decoupled* planning is the most popular, which generally performs coordination of robot motion after deciding a path for each robot [18, 22, 34, 37, 43, 49]. In contrast, priority based methods force an order on agents to significantly reduce the search space [11, 47]. Some more recent works using decoupling heuristics include applying optimal decoupling techniques to exploit problem instances with low degrees of coupling [48], using *push-and-swap* primitives to avoid unnecessary exploration of search space [30], and heuristics aimed at performance guarantees (completeness is lost) [50].

Our algorithmic efforts in this paper focus on the *permutation invariant* multi-agent path planning problem on CUGs. Such formulations, in both discrete and continuous forms, are extensively studied as formation control problems [3, 38, 42, 44, 45], among others. On research that appears mostly related to this aspect of our paper, a discrete grid abstraction model for formation control was studied in [32]. To plan the paths, a three-step process was used in [32]: 1) Target assignment, 2) Path allocation, 3) Trajectory scheduling. Although it was shown that the process always terminates, no characterization of solution complexity was offered. In contrast, we provide very efficient algorithms that solve a strictly more general class of problems with optimality assurance. On the continuous side, a novel *formation space* approach was employed to represent the entire formation of robot teams with a single polynomial of which the roots correspond to the unassigned configurations for the robots in the formation [24].

We delay the literature review on network flow, from which we devise our time expansion construction for multi-agent path planning, to Section 3. The basic idea of applying time expansion to robotics problem is far from new [11, 36]. To the best of our knowledge, however, the research presented here is an original attempt at proposing a general time expansion technique, connecting it to network flow, and making full use of the benefits that come with this approach. We also note that our exact and complete algorithms all come with low constants in their respective worst case time complexity because they are derived from well studied combinatorial algorithms[2]. Our simulation result, which we omit due to the length limit, confirms this assertion.

There are three main contributions. First, we formally establish the link between multi-agent path planning on graphs and network flow, showing how multi-agent path planning can be reduced to network flow problems, thereby enabling the potential application of powerful tools from combinatorial optimization to path planning for multiple agents in a principled way. Second, for the planning problem in which agents do not have pre-specified goals, we give fast and complete algorithms for finding collision free path sets that deliver every agent to a different goal. Third, we study time and distance optimality of the feasible solutions to the aforementioned problem, show that they have a pairwise Pareto optimal structure, and again provide efficient algorithms for optimizing each of these practical objectives.

The rest of the paper is organized as follows. In Section 2, we define two multi-agent path planning problems on CUGs. Section 3 starts with a quick review of

[2] A *combinatorial algorithm* is an algorithm that only adds, subtracts, and compares values; no multiplication and division operations are allowed (i.e., ordered *group* operations versus ordered *field* operations) [17].

network flow problems and then proceeds to show the reduction from multi-agent path planning on CUGs to network flow. Concentrating our efforts on the permutation invariant multi-agent path planning problem, Section 4 begins with a key construction that allows us to tightly bound the time steps required for a time-expanded network to have a feasible solution, which in turn enables efficient algorithms. Section 5 takes a further step and studies solution optimality on three natural objectives, showing the objectives have a Pareto optimal structure. We conclude in Section 6. We omit all proofs due to their length, which will be included in an extended version of this paper.[3]

2 Multi-agent Path Planning Problems on Collision-free Unit-distance Graphs

Let $G = (V,E)$ be a connected, undirected, simple graph (i.e., no multi-edges), in which $V = \{v_i\}$ is its vertex set and $E = \{(v_i, v_j)\}$ is its edge set. Let $A = \{a_1, \ldots, a_n\}$ be a set of agents with initial and goal locations on G given by the injective maps $x_I : A \to V$ and $x_G : A \to V$, respectively. Note that A is essentially an index set; $x_I(A)$ and $x_G(A)$ are the set of initial and goal locations, respectively. We require that $x_I(A)$ and $x_G(A)$ be disjoint. For convenience, we let $n = |A|$ and use V,E to denote the cardinality of the sets V,E, respectively, since the meaning is usually clear from the context. Let σ be a bijection that acts on the elements of x_G, a *feasible path* for a single agent a_i is a map $p_i : \mathbb{Z}^+ \to V$ with the following properties[4]: 1. $p_i(0) = x_I(a_i)$. 2. There exists a smallest $k_{\min} \in \mathbb{Z}^+$ such that $p_i(k_{\min}) = (\sigma \circ x_G)(a_i)$ for some fixed σ. That is, the end point of the path p_i is some goal vertex. 3. For any $k \geq k_{\min}$, $p_i(k) \equiv (\sigma \circ x_G)(a_i)$. 4. For any $0 \leq k < k_{\min}$, $(p_i(k), p_i(k+1)) \in E$ or $p_i(k) = p_i(k+1)$. Intuitively, think of the domain of the paths as discrete time steps. We say that two paths p_i, p_j are in *collision* if there exists $k \in \mathbb{Z}^+$ such that $p_i(k) = p_j(k)$ (meet) or $(p_i(k), p_i(k+1)) = (p_j(k+1), p_j(k))$ (head-on). Given a path p, let E_p denote the set of edges of the form $(p(k), p(k+1))$ for all applicable $k \in \mathbb{Z}^+$ when $p(k) \neq p(k+1)$. If $p(k) = p(k+1)$, the agent stays at vertex $p(k)$ during the time interval $[k, k+1]$.

As mentioned, in this paper, we work with a specific type of graph called the collision-free unit-distance graph (CUG): A CUG is a connected, undirected graph G satisfying the following: 1. Every edge is of unit length; 2. Given any two distinct edges (u_1, v_1) and (u_2, v_2) of G with $u_1 \neq u_2, v_1 \neq v_2$, two disc shapes (or spherical for 3D or more) agents of radius less than 1/2 traveling at unit speed through these edges (starting simultaneously at u_1, u_2, respectively) will never collide. For example, a connected 2D grid with holes is a CUG. Since subgraphs of 2D grids are easy to draw and visualize, we generally use subgraphs of 2D grids when we create examples in this paper. With the above setup, the *multi-agent path planning on CUGs* problem is defined as follows.

[3] For proofs, see `http://msl.cs.uiuc.edu/~jyu18/subm/proofs.pdf`.

[4] In this paper, we let $\mathbb{Z}^+ := \mathbb{N} \cup \{0\}$.

Problem 1. [Multi-agent Path Planning on CUGs] Given a 4-tuple (G, A, x_I, x_G) in which G is a CUG, find a set of paths $P = \{p_1, \ldots, p_n\}$ such that p_i's are feasible paths for respective agents a_i's with σ being the identity map and no two paths p_i, p_j are in collision.

We require the graph to be a CUG so that it is suitable for multi-agent path planning. We formalize the rationale in the following lemma.

Lemma 1. *Let p_i, p_j be two paths that are not in collision (as a partial solution to Problem 1). Then two disc shaped agents*[5] *of radius less than 1/2, starting at the same time and moving along these respective paths with unit speed, will never collide.*

Lemma 1 shows that a solution to Problem 1 provides a path set for disc agents with unit diameter in A to reach their respective goals without a collision. It is easy to see that not all instances of this problem are solvable. Furthermore, since Problem 1 is a generalization of multi-agent path planning on 2D grids and it is NP-hard to optimally (i.e., using least number of moves) solve the $N \times N$ extension of the 15-puzzle [39], it is NP-hard to find the shortest total path length for Problem 1. If we remove the assumption that all agents must reach their respective goals and allow permutation invariant paths (i.e., as long as each goal gets occupied by a unique agent in the end), Problem 1 becomes the *permutation invariant multi-agent path planning on CUGs* problem.

Problem 2. [Permutation Invariant Multi-agent Path Planning on CUGs] Given a 4-tuple (G, A, x_I, x_G) in which G is a CUG, find a set of paths $P = \{p_1, \ldots, p_n\}$ such that p_i's are feasible paths for respective agents a_i's for an arbitrary (but fixed) permutation σ and no two paths p_i, p_j are in collision.

Problem 2 models the problem in which multiple identical or indistinguishable agents need to be deployed for serving requests at different locations (for example, formation control). This problem always has a solution: We simply plan and execute one path at a time and use more "remote" goal vertices earlier to avoid possible blocking of later paths; a formal result on the existence of such a choice of paths is given in Section 4.

3 Multi-agent Path Planning on CUGs and Network Flow

3.1 Network Flow

In this subsection we give a brief review of network flow problems and algorithms pertinent to our problems. For surveys on network flow, see [2, 14]. We start with the classic static network flow problems.

Static Network Flow. A *network* $\mathcal{N} = (G, u, c, S)$ consists of a directed graph $G = (V, E)$ with $u, c : E \to \mathbb{Z}^+$ as the maps defining the capacities and costs on edges,

[5] Or spherical agents with radius less than 1/2, for dimensions higher than 2.

respectively, and $S \subset V$ as the set of sources and sinks. We let $S = S^+ \cup S^-$, with S^+ denoting the set of sources and S^- denoting the set of sink vertices. For a vertex $v \in V$, let $\delta^+(v)$ (resp. $\delta^-(v)$) denote the set of edges of G going to (resp. leaving) v. A feasible (static) S^+, S^--flow on this network $\mathscr{N}$ is a map $f : E \to \mathbb{Z}^+$ that satisfies edge capacity constraints,

$$\forall e \in E, \quad f(e) \leq u(e), \tag{1}$$

the flow conservation constraints at non terminal vertices,

$$\forall v \in V \backslash S, \quad \sum_{e \in \delta^+(v)} f(e) \; - \sum_{e \in \delta^-(v)} f(e) = 0, \tag{2}$$

and the flow conservation constraints at terminal vertices,

$$\sum_{v \in S^+} \Big(\sum_{e \in \delta^-(v)} f(e) \; - \sum_{e \in \delta^+(v)} f(e) \Big) = \sum_{v \in S^-} \Big(\sum_{e \in \delta^+(v)} f(e) \; - \sum_{e \in \delta^-(v)} f(e) \Big). \tag{3}$$

The quantity on either side of (3) is called the *value* of the flow.

The classic (single-commodity) *maximum flow* problem asks the question: Given a network $\mathscr{N}$, what is the maximum value of flow that can be pushed through the network (i.e., seeking to maximize F)? The *minimum cost maximum flow* problem further requires the flow to have minimum total cost among all maximum flows. That is, we want to find the flow among all maximum flows such that the quantity

$$\sum_{e \in E} c(e) \cdot f(e) \tag{4}$$

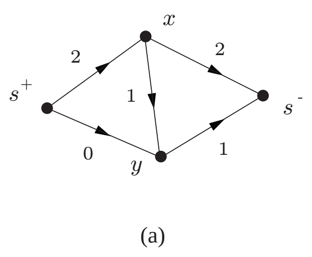
is minimized. Given integer inputs, integer maximum flow always exists, and many polynomial time algorithms exist for finding such a solution [10, 16]. The minimum cost maximum flow problem is equivalent to the *minimum cost circulation* problem, which is also solvable in polynomial time [46].

When additional structure is put on S, additional questions arise. If we limit the supply (resp. demand) of the source (resp. sink) vertices, we obtain a type of the flow problem called the *transshipment* problem. To formalize this, let $d : V \to \mathbb{Z}$ be the supplies on the vertices of G. Given a vertex $v \in V$, a positive $d(v)$ suggests that the vertex has positive supply ($v \in S^+$) and a negative one suggests that the vertex has positive demand ($v \in S^-$). For all other vertices v, $d(v) = 0$. The basic version of the transshipment problem asks for a feasible flow through the network that also respects the supply/demand requirements

$$\begin{aligned} \forall v \in S^+, \quad & \sum_{e \in \delta^-(v)} f(e) \; - \sum_{e \in \delta^+(v)} f(e) = d(v), \\ \forall v \in S^-, \quad & \sum_{e \in \delta^+(v)} f(e) \; - \sum_{e \in \delta^-(v)} f(e) = d(v). \end{aligned} \tag{5}$$

The transshipment problem becomes the *evacuation* problem when $|S^-| = 1$ and the demand of the single sink vertex is equal to the total supply of the source vertices. The transshipment problem and the evacuation problem, as special cases of the maximum flow problem, can be solved with maximum flow algorithms mentioned above. If we instead require that vertices of S^+, S^- are paired up as $(s_1, s'_1), \ldots, (s_k, s'_k)$ and that commodity of type i can be injected only into s_i and taken out at s'_i, we get the *multi-commodity flow* problem. Optimality questions as these from the single-commodity case can be asked here as well. Unlike in the single commodity case, finding integer maximum flow for multi-commodity problems is NP-hard in general and MAX SNP-hard (NP-hard to approximate below a certain multiple of optimal flow value) even for some simple restrictions [9].

Dynamic Network Flow. If we consider that flowing commodities through edges takes some time to complete, the problem becomes a *dynamic network flow* problem, which sometimes is also called *network flow over time*. There are two common variations of the dynamic network flow model: *Discrete time* and *continuous time*. In a *discrete time* model, flows enter and exit from vertices at integer time steps $t = 0, 1, \ldots, T$. For a given edge $e = (u, v) \in E$, we may view the cost $c(e)$ as the time that is required to pass an amount of flow (not exceeding the capacity) from the tail u to the head v of the edge e. Therefore, we may interpret a (static) flow



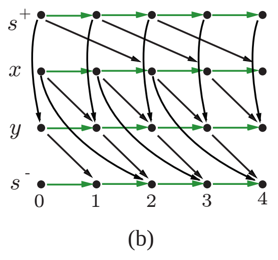


Fig. 2 a) The (static) flow network with source s and sink t. The numbers on the edges are the costs/time delay for passing through these edges. We may assume that the capacities are all unit capacities. b) The time-expanded network with 5 copies of the original vertices ($T = 4$). All edges have unit capacity. There is a forward edge between two vertices u and v at time steps t and t', respectively (e.g. x at $t = 0$ and y at $t' = 1$), if one of the following is true: 1. $e = (u, v)$ is an edge of the static network with $c(e) = t' - t$ (the black edges, which retain the costs as $c(e)$'s); 2. u, v are the same vertex of the static network and $t' - t = 1$ (the green edges, which have unit costs). The green edges are also called *holdover* edges since traveling through a green edge is the same as the agent not actually moving.

network $\mathscr{N}$ as a dynamic one without any change of notations. In the closely related *continuous time* model, which we do not use in this paper, a *flow rate* is assigned to each edge, designating how fast a unit of flow can pass through the edge. The constraints imposed in the static network flow model generally apply to dynamic network flow models, except that dynamic network flow further requires that at any time, the flow passing through any edge cannot exceed the edge capacity.

Given a dynamic flow network, a question similar to the single-commodity maximum flow problem is the following: Starting at $t = 0$, what is the maximum units of flow the can reach the sinks on or before time $t = T$? It turns out that this problem can be solved using static flow algorithms such as Edmonds-Karp [10] over a *time-expanded network*. For example, given the dynamic flow network in Fig. 2(a), its time-expanded network with $T = 4$ is given in Fig. 2(b). To compute a flow over the time expanded network, we first add a *super source* and connect it using outgoing edges to all copies of source vertices at $t = 0$, and add a *super sink* and connect all copies of sink vertices for all t to it using outgoing edges (the super source, super sink, and additional edges are not shown in Fig. 2(b)). With this construction, a static flow on the time-expanded network corresponds to a dynamic flow on the dynamic flow network. In particular, assuming a sufficiently large T, we can state the following [13].

Lemma 2. *A flow for a dynamic flow network $\mathscr{N}$ is feasible if and only if the corresponding static flow on the time-expanded network of $\mathscr{N}$ is feasible.*

Using a time-expanded network comes with a caveat. The standard maximum flow algorithms have time complexity depending polynomially on T and are therefore *pseudopolynomial* in general. For a special class of problems, the *quickest transshipment problem*, of which the goal is finding the quickest feasible flow for a transshipment problem over a dynamic network, strongly polynomial time algorithm[6] exists [20]. However, the algorithm requires calling subroutines (for example, submodular function optimization routines) that are not *combinatorial* algorithms and also has with large constant terms when it comes to asymptotic time complexity.

3.2 Equivalence Between Multi-agent Path Planning on CUGs and Maximum Network Flow

In this subsection, we establish a reduction from the problems of our interest to multi-commodity network flow. For illustration purposes, we use the simple graph G in Fig. 3(a), with initial locations $\{s_i^+\}, i = 1,2$ and goal locations $\{s_i^-\}, i = 1,2$. An instance of Problem 1 is given by $(G, \{a_1, a_2\}, x_I : a_i \mapsto s_i^+, x_G : a_i \mapsto s_i^+)$. To apply maximum flow algorithms, we construct from G a time-expanded directed graph G', part of which is shown in Fig. 3(c). We construct Fig. 3(c) as follows.

Since we cannot create an infinite time-expanded network, we need to specify the required number of time steps. For now assume that this number is some sufficiently large T (that is, if a flow with value F is achievable with an arbitrarily long time expansion, then F is also achievable with only T time steps). After fixing T, we create $2T+1$ copies of vertices from G, with indices $0, 1, 1', \ldots$, as shown in Fig. 3(c). For each vertex $v \in G$, we denote these copies $v(0) = v(0)', v(1), v(1)', v(2), \ldots, v(T)'$. For each edge $(u,v) \in G$ and time steps $t, t+1$, $0 \le t < T$, we then add the gadget

[6] An algorithm is a strongly polynomial algorithm if: 1. The number of operations in the arithmetic model of computation is bounded by a polynomial in the number of integers in the input instance, and 2. The space used by the algorithm is bounded by a polynomial in the size of the input [17].

shown in Fig. 3(b) between $u(t)', v(t)'$ and $u(t+1), v(t+1)$ (arrows from the gadget are omitted from Fig. 3(c) since they are too small to draw). This gadget ensures that two agents cannot travel in opposite directions on an edge in the same time step. For the gadget, we assign unit capacity to all edges, unit cost to the horizontal middle edge, and zero cost to the other four edges. To finish the construction of Fig. 3(c), for each vertex $v \in G$, we add one edge between every two successive copies (i.e., we add the edges $(v(0), v(1)), (v(1), v(1)'), \ldots, (v(T), v(T)')$). These correspond to the green and blue edges in Fig. 3(c). For all green edges, we assign them unit capacity and cost; for all blue edges, we assign them unit capacity and zero cost.

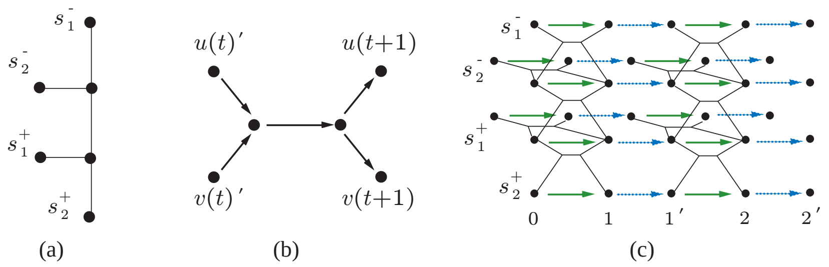


Fig. 3 a) A simple CUG G. b) A gadget for splitting an undirected edge through time steps. c) Part of the time-expanded network ($T = 2$).

The graph Fig. 3(c) is the main piece of G', which is mostly done with the exception of the set S. To create S^+, for each source vertex $u \in \{s_i^+\}$, we create a new vertex u' and add that to S^+. We then add the edge $(u', u(0))$ to G'. The edge has unit capacity and zero cost. To create S^-, for each $v \in \{s_i^-\}$, we add two vertices v', v'' and the edge (v', v'') to G' with unit capacity and zero cost. We then connect all copies of $v \in G'$ to v' (e.g. add edges $(v(0), v'), \ldots, (v(T)', v')$) and let these edges have unit capacity and zero cost. The set of all v'''s is the set S^-. The network $\mathcal{N}' = (G', u, c, S^+ \cup S^-)$ is now complete; we have now reduced Problem 1 to an integer maximum multi-commodity flow problem on $\mathcal{N}'$ with each agent from A as a single type of commodity.

Theorem 3. *Given an instance of Problem 1 with input parameters (G, A, x_I, x_G), there is a bijection between its solutions (with maximum number of time steps up to T) and the integer maximum multi-commodity flow solutions of flow value n on the time-expanded network $\mathcal{N}'$ constructed from (G, A, x_I, x_G) with T time steps.*

Since integer maximum multi-commodity flow is NP-hard, the above construction does not directly offer an efficient solution to Problem 1. Nevertheless, with backtracking, it is not hard to design complete algorithms that search the time-expanded network $\mathcal{N}'$ for a feasible solution. Our preliminary analysis shows that when the problem instance is not particularly hard (i.e., when there are not many narrow passages in the combined configuration space), a solution can be found relatively quickly. We plan to study this problem in more detail in future work. Alternatively, we may readily obtain an integer linear programming problem from $\mathcal{N}'$ and apply heuristics such as *branch and bound* method [25], for which many heavily optimized numerical packages are readily available.

Moving to Problem 2, allowing an arbitrary permutation σ to act on x_G means that we may treat all agents as a single type of commodity. Theorem 3 then implies that Problem 2 is equivalent to the quickest transshipment problem, which is solvable in polynomial time using subroutines for optimizing submodular functions. In the next section, we show that we can do better by bounding the required time steps for finding a feasible solution to Problem 2 and then apply more standard combinatorial algorithms for network flow to solve it.

4 Efficient Combinatorial Algorithms for Permutation Invariant Multi-agent Path Planning on CUGs

If we choose to apply combinatorial network flow algorithms over the time-expanded network to find solutions to Problem 2, the first priority is to determine the required number of time steps necessary to find a solution; otherwise we cannot declare that the algorithm is complete. We now provide a tight bound on T. Let (G,A,x_I,x_G) be an instance of Problem 2. We first prove some intermediate results on path sets over G. To distinguish these paths from the solution path set, denote them as $Q = \{q_1,\ldots,q_n\}$. For convenience, $head(q_i)$, $tail(q_i)$, and $len(q_i)$ denote the start vertex, end vertex, and length of q_i, respectively. With a slight abuse of notation, $V(\cdot), E(\cdot)$ denote the vertex set and edge set of the input parameter, which can be either a path, q_i, or a set of paths, such as Q. An *intersection* between two paths is a maximal consecutive sequence of vertices and edges common to the two paths. A *standalone* goal vertex is a vertex $v \in x_G(A)$ such that there is a single path $q \in Q$ containing v. To start off, we want a path set Q with the following properties:

Property 1. For all $1 \le i \le n$, $head(q_i) \in x_I(A)$ and $tail(q_i) \in x_G(A)$. For any two paths q_i, q_j, $head(q_i) \neq head(q_j)$ and $tail(q_i) \neq tail(q_j)$.

Property 2. Each path q_i is a shortest path between $head(q_i)$ and $tail(q_i)$ on G.

Property 3. The total length of the path set Q is minimal.

Property 4. If we orient the edges of every path $q_i \in Q$ from $head(q_i)$ to $tail(q_i)$, no two paths share a common edge oriented in different directions.

Properties 1 and 2 are merely restrictions to have the initial and goal vertices paired up using shortest paths. Property 3 requires the total length of these paths to be minimal. Property 4, which is implied by Property 3, lends to show that the paths can be oriented to form a *directed acyclic graph*.

Lemma 4. *There exists a set of paths $Q = \{q_1,\ldots,q_n\}$ that satisfies Properties 1-4.*

The technique from the proof of Lemma 4 can be generalized to show that oriented paths cannot form any directed cycles, which in turn implies the existence of a standalone goal vertex.

Proposition 5. *A path set Q that satisfies Properties 1-3 induces a directed acyclic graph (DAG) structure on $E(Q)$.*

Corollary 6. *A path set Q that satisfies Properties 1-3 has a standalone goal vertex.*

The existence of a standalone goal vertex allows the construction of a path set which decomposes into paths that can be sequentially scheduled without colliding into each other. We characterize such a path set as one with an additional property.

Lemma 7. *There exists a path set Q satisfying Properties 1-4 and the following additional property:*

Property 5. Let $Q_i := \{q_i, \dots, q_n\}$. For any $1 \leq i \leq n$, restricting to Q_i, among all possible paths connecting an initial location (of Q_i) to a standalone goal location (of Q_i) using oriented edges from $E(Q_i)$, q_i is one shortest such.

If we schedule agents using a path set Q satisfying properties 1-5, there can never be cases where two agents block each other, as a direct consequence of Lemma 4. There is still the possibility that one agent blocks another. The following theorem shows that such blocking can be minimized.

Theorem 8. *Given an instance of Problem 2 with input parameters (G, A, x_I, x_G) and let ℓ be the largest pairwise distance between a member of $x_I(A)$ and a member of $x_G(A)$,*

$$\ell = \max_{\forall u \in x_I(A), v \in x_G(A)} dist(u, v). \tag{6}$$

A time-expanded network $\mathscr{N}'$ with $T = n + \ell - 1$ is necessary and sufficient for a feasible solution to Problem 2 to exist.

Since ℓ cannot be larger than V, the number of vertices of G, the following corollary is immediate.

Corollary 9. *For every instance of Problem 2, a feasible solution exists.*

In particular, the construction in the proof of Lemma 4 yields a complete (may not be efficient) algorithm for Problem 2. In addition to confirming that any maximum flow algorithm over the time expanded network $\mathscr{N}'$ with $T = n + \ell - 1$ is a also complete algorithm, Theorem 8 enables us to show that such algorithms are efficient.

Theorem 10. *Problem 2 is solvable usign a combinatorial algorithm in strongly polynomial time.*

Using the Ford-Fulkerson algorithm [12], the time complexity is $O(nVE)$. In practice, even better running times are possible. If G is a planar graph, we have $E \sim O(V)$ and $\ell \sim O(V^{\frac{1}{2}})$. The time complexity then becomes $O(\mathrm{MF}(n, V(n+V^{\frac{1}{2}} - 2), V(n+V^{\frac{1}{2}} - 2))) \sim O(\mathrm{MF}(n, V(n+V^{\frac{1}{2}}), V(n+V^{\frac{1}{2}})))$. Since in our case $n < V$, Ford-Fulkerson gives us the running time $O(nV(n+V^{\frac{1}{2}})) = O(n^2V + nV^{\frac{3}{2}})$.

5 Optimal Solutions

In this section, we present optimal solutions for the permutation invariant multi-agent path planning problem. After introducing several temporal and spatial objectives of practical importance, we apply techniques from network flow to obtain optimal solutions for these objectives. Since these objectives are different from the basic version of Problem 2, we provide bounds on T again to obtain strongly polynomial algorithm for them. Lastly, we show that these objectives possess a Pareto optimal structure and they cannot be optimized simultaneously.

5.1 Optimizing over the Feasible Solutions

Having found *feasible* solutions to Problem 2, we turn the focus to the *optimality* of these solutions for practical purposes. As mentioned in Section 2, we intend to use the formulation as a model for scenarios such as multi-robot servicing. For many applications, time optimality is a top priority. Optimizing over the feasible solutions to Problem 2 (that is, we require that all goals are reached), there are two natural criteria for measuring time optimality:

Objective 1. Minimizing the average time it takes for all agents to reach their goals.

Objective 2. Minimizing the time it takes for the last agent to reach its goal.

In terms of agents (robots or people) serving requests, Objective 1 seeks to minimize the average time before a request gets served, which is arguably the most efficient approach. By Theorem 3, optimizing over Objective 1 is equivalent to finding a maximum flow on the time-expanded network $\mathcal{N}'$ such that the total cost of the flow is at minimum. This is exactly the minimum cost maximum flow objective stated in (4), which is equivalent to finding the *minimum cost circulation* on an slightly augmented network, for which polynomial time algorithms exist (polynomial in T). To show that Objective 1 can be optimized in polynomial time, we bound T with respect to the problem input as follows.

Theorem 11. *There exists an optimal solution for Objective 1 in a time-expanded work with* $T = n(n-1)/2 + \ell$.

Theorem 11 implies that the number of edges of the time-expanded graph is at most $O(E(n^2+V))$. The total complexity is at most $O((n^2+V)VE\log V)$ [1]. The second objective, minimizing the time that its last goal is reached, provides a lower bound on the time that is required to reach all goals. Solutions optimizing this objective are useful in providing worst servicing time estimate or guarantee. Solutions to the quickest transshipment problem [20] yield optimal solutions to this objective. However, we can avoid using submodular function optimization routines if we have a polynomial bound on T, which is provided in the following corollary of Theorem 8.

Corollary 12. *There exists an optimal solution for Objective 2 in a time-expanded work with* $T = n + \ell - 1$.

To see that Corollary 12 is true, note that $T = n + \ell - 1$ is sufficient for finding a feasible solution, which must have completion time as large as that of a solution to Objective 2. With the bound on T, running $\log T$ rounds (via binary search) of maximum flow over time-expanded network with different time horizon then gives us an optimal solution to Objective 2. The running time is then bounded by $O(\mathrm{MF}(n, V^2, VE)\log V)$, which is strongly polynomial. In particular, with Ford-Fulkerson, the running time becomes $O(nVE\log V)$. After time optimality, another very useful solution property to optimize is the total distance traveled by the agents, i.e., spatial optimality:

Objective 3. Minimizing the total distance traveled by the agents on G.

Because we work with a CUG, if an agent a_i actually moves along path p_i between time steps t and $t+1$, $p_i(t)$ must be different from $p_i(t+1)$. These correspond to the black edges in the time-expanded network (see Fig. 3(b)). Thus, to optimize this objective, we can find the shortest total distance traveled by all agents via setting the cost of the holdover edge (green edges in Fig. 3(b)) to zero and then running minimum cost maximum flow algorithm over the time-expanded network. The method is again strongly polynomial, with complexity $O(V^2 E \log V)$, due to the following corollary.

Corollary 13. *There exists an optimal solution for Objective 3 in a time-expanded work with $T = n + \ell - 1$.*

5.2 Pareto Optimality Between the Objectives

From the discussion in the previous subsection, we observe that each of the three objectives is of practical importance. At this point, one might be tempted to seek solutions that optimize multiples of these objectives simultaneously. We show that this is not possible for each pair of these objectives. In the following theorem, we say that two objectives are *compatible* if and only if they can be optimized simultaneously. Otherwise, we say the objectives are *incompatible*.

Theorem 14. *Over the feasible solutions to Problem 2, Objectives 1-3 are pairwise incompatible.*

6 Conclusion, Future Work, and Open Problems

In this paper, we established the close link between two classes of problems: Multi-agent path planning on CUGs and network flow. Focusing on the permutation invariant versions of the multi-agent path planning problem, we proved a tight bound on the number of time steps necessary and sufficient for a feasible path set to exist in the time-expanded network, enabling efficient algorithmic solutions to these problems. We then explored optimality issues, demonstrating that the time-expansion bound generally carry over to yield strongly polynomial algorithms for optimizing these practical objective functions. Interestingly, each pair of these objectives cannot be optimized simultaneously.

Given our study, an immediate question or criticism is the applicability of the results to problems beyond CUGs. After all, real agents, whether robots or people, do not always live on a discrete graph. To answer this question, we have research under way that explores the idea of overlaying the the CUGs on the actual workspace. That is, we may first create a roadmap over the workspace that captures the connectivity and then discretize the roadmap over which the statement of Lemma 1 continues to hold (as long as the edges are close to unit length the angle between two edges is obtuse, similar version of Lemma 1 can be stated) [35]. A basic solution (Fig. 4(a)) may be to put a grid on the roadmap and delete vertices inside or close to obstacles. To overcome the issue of the inherited Manhattan metric of grids, we may adapt the grid to align with the geodesics of the environment. For example, for a two dimensional workspace with polygonal obstacles, we can arrange the grid edges to follow edges of the visibility graph [29] of the environment when possible. When clearance is tight, we may start with a maximum clearance roadmap [35] and add the vertices carefully (see Fig. 4(b)). Note that because the workspace is often two dimensional, these preparations can be computed relatively efficiently.

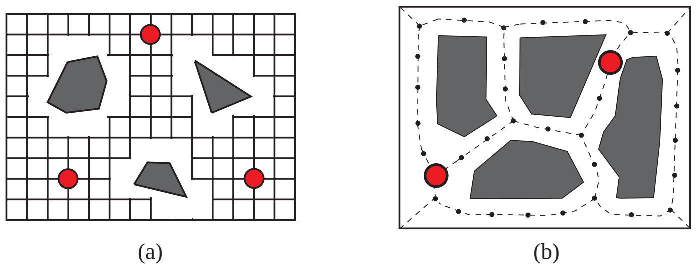


Fig. 4 a) Overlaying grid on a workspace with obstacles. b) Adapting a roadmap to obtain a graph that can be used with our multi-agent path planning algorithms.

Many interesting open problems remain. Although finding a distance optimal solution to Problem 1 using a time-expanded network is impractical due to its intrinsic hardness, the network flow approach might still produce efficient methods that yield basic feasible solutions since the time-expanded network has a forward only structure. In addition, approximation algorithms on integer multi-commodity flow could lead to better heuristics for optimal solution search. Along this line, we only touched the most essential results in the field of network flow, which are but the tip of the iceberg. It would not be surprising that results from the vast amount of network flow literature could be readily carried over to tackle path planning problems, either as we proposed in this paper or in some other forms. As an example, for Problem 2, since Objectives 1-3 are all of practical concerns but incompatible, it is desirable to seek solutions that provide performance guarantees on each of these objectives. Network flow methods, closely relate to linear programming, appear to be promising tools for such parametric optimization tasks.

Acknowledgments This work was supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

References

1. R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. Finding minimum-cost flows by double scaling. *Mathematical Programming*, 53:243–266, 1992.
2. J. E. Aronson. A survey on dynamic network flows. *Annals of Operations Research*, 20(1):1–66, 1989.
3. T. Balch and R. C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transaction on Robotics and Automation*, 14(6):926–939, 1998.
4. J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.
5. L. G. Chalmet, R. L. Francis, and P. B. Saunders. Network models for building evacuation. *Management Science*, 28(1):86–105, 1982.

6. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, 2005.
7. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proceedings Second GI Conference on Automata Theory and Formal Languages*, pages 134–183, Berlin, 1975. Springer-Verlag. Lecture Notes in Computer Science, 33.
8. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (2nd Ed.)*. MIT Press, Cambridge, MA, 2001.
9. Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. Minimal Multicut and Maximal Integer Multiflow: A Survey. *European Journal of Operational Research*, 162:55–69, 2005.
10. J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.
11. M. A. Erdmann and T. Lozano-Pérez. On multiple moving objects. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1419–1424, 1986.
12. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Research Memorandum RM-1400, The RAND Corporation*, November 1954.
13. L. R. Ford. and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.
14. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, New Jersey, 1962.
15. D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autom. Robots*, 8(3):325–344, June 2000.
16. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 136–146, New York, NY, USA, 1986. ACM.
17. M. Grötschel and A. Schrijver L. Lovász. *Complexity, Oracles, and Numerical Computation*. Springer, 1988.
18. Y. Guo and L. E. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2612–2619, 2002.
19. D. Halperin, J.-C. Latombe, and R. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26(3-4):577–601, 2000.
20. B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):36–62, 2000.
21. J. S. Jennings, G. Whelan, and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *Proceedings IEEE International Conference on Robotics & Automation*, 1997.
22. K. Kant and S. Zucker. Towards efficient trajectory planning: The path velocity decomposition. *International Journal of Robotics Research*, 5(3):72–89, 1986.
23. 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 & Automation*, 12(4):566–580, June 1996.
24. S. Kloder and S. Hutchinson. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics*, 22(4):650–665, 2006.
25. A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.
26. J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.
27. S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.
28. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.
29. T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

30. R. Luna and K. E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conferences in Artificial Intelligence (IJCAI-11)*, pages 294–300, Barcelona, Spain, 16-22 July 2011.
31. M. J. Matarić, M. Nilsson, and K. T. Simsarian. Cooperative multi-robot box pushing. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 556–561, 1995.
32. D. Miklic, S. Bogdan, R. Fierro, and S. Nestic. A discrete grid abstraction for formation control in the presence of obstacles. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3750–3755, 2009.
33. B. Nnaji. *Theory of Automatic Robot Assembly and Programming*. Chapman & Hall, 1992.
34. P. A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 484–489, 1989.
35. C. O'Dúnlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.
36. M. Peasgood, C. Clark, and J. McPhee. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, 24(2):283–292, 2008.
37. 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, *Algorithmic Foundations of Robotics V (WAFR 2002)*, pages 221–237. Springer-Verlag, Berlin, 2002.
38. S. Poduri and G. S. Sukhatme. Constrained coverage for mobile sensor networks. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.
39. D. Ratner and M. W. Finding. A shortest solution for the N $\times$ N extension of the 15-puzzle is intractable. In *Proceedings AAAI National Conference on Artificial Intelligence*, pages 168–172, 1986.
40. S. Rodriguez and N. M. Amato. Behavior-based evacuation planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 350–355, 2010.
41. D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 235–242, 1995.
42. B. Shucker, T. Murphey, and J. K. Bennett. Switching rules for decentralized control with simple control laws. In *American Control Conference*, July 2007.
43. T. Siméon, S. Leroy, and J.-P. Laumond. Path coordination for multiple mobile robots: A resolution complete algorithm. *IEEE Transactions on Robotics & Automation*, 18(1), February 2002.
44. B. Smith, M. Egerstedt, and A. Howard. Automatic deployment and formation control of decentralized multi-agent networks. In *Proceedings IEEE International Conference on Robotics and Automation*, 2008.
45. H. Tanner, G. Pappas, and V. Kumar. Leader-to-formation stability. *IEEE Transactions on Robotics and Automation*, 20(3):443–455, Jun 2004.
46. É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.
47. J. van den Berg and M. Overmars. Prioritized motion planning for multiple robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.
48. J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Proceedings Robotics: Science and Systems*, 2009.
49. P. Švestka and M. H. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23:125–152, 1998.
50. K.-H. C. Wang and A. Botea. Tractable multi-agent path planning on grid maps. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1870–1875, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.