Time Optimal Multi-agent Path Planning on Graphs*

Jingjin Yu
Department of Electrical and Computer Engineering
University of Illinois, Urbana, IL 61801
jyu18@uiuc.edu

Steven M. LaValle
Department of Computer Science
University of Illinois, Urbana, IL 61801
lavalle@uiuc.edu

Introduction

Significant progress has been made in the area of multi-agent path finding/planning in the past decade (Silver 2005; van den Berg et al. 2009; Standley 2010; Luna and Bekris 2011; Wang and Botea 2011). In this work, we introduce a multi-agent path planning problem similar to that of (Standley 2010) and aim at maximizing parallelism among the agents. That is, we seek a feasible plan that minimizes the time it takes the last agent to reach its goal. To solve the problem, we convert it into an equivalent multiflow problem, from which an integer linear programming (ILP) problem is readily obtained. Our algorithm is complete.

Problem Formulation

Let $G = (V, E)$ be a connected, undirected, simple graph with vertex set $V = \{v_i\}$ and edge set $E = \{(v_i, v_j)\}$. Since the meaning is clear from context, V, E also denote the cardinality of these sets, respectively. Let $A = \{a_1, \ldots, a_n\}$ be a set of n agents, $n \leq V$. The agents must reside on distinct vertices of G at integer time steps beginning at time zero; each agent may move from a vertex to an adjacent vertex in one time step. An agent may also remains still between time steps. When applied to the agent set A, the term *move* denotes the location changes of all agents between two consecutive time steps. A move is *legal* if no two agents move along an edge in opposite directions (head-on collision). Note that the "distinct vertices" assumption precludes the possibility of two agents moving to the same vertex (meet collision). Viewing A as an index set, the initial and goal configuration of the agents are defined by the injective maps, $x_I, x_G : A \to V$, respectively. For a fixed $1 \leq i \leq n$, a *path* for agent a_i is a map $p_i : \mathbb{Z}^+ \to V$, in which $\mathbb{Z}^+ := \mathbb{N} \cup \{0\}$. Intuitively, view the domain of the paths as discrete time steps; the path set then directly maps to agent moves on G. We say that a path set $P = \{p_1, \ldots, p_n\}$ is *feasible* if:

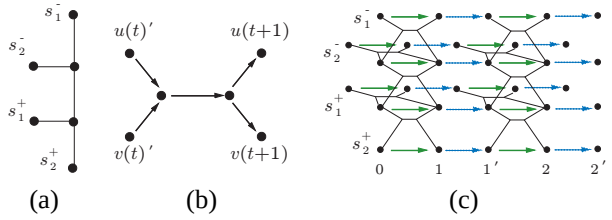
1. For each $1 \leq i \leq n$, $p_i(0) = x_I(a_i)$ and there exists a least $k_i^{\min} \in \mathbb{Z}^+$ such that $p_i(k) \equiv x_G(a_i)$ for $k \geq k_i^{min}$;
2. The paths map to legal moves of the agent set A on G.

Problem 1 (Multi-agent Path Planning) *Given a* 4*-tuple* (G, A, x_I, x_G)*, find a set of paths* $P = \{p_1, \ldots, p_n\}$ *that is feasible.*

We note two implied features of our formulation. First, our formulation allows multiple agents to move at the same time step as long as no collision occurs. On a graph, this allows agents on any circle to "rotate", provided that the agents' sizes are suitable (for unit distance grids, this requires the agent has radius no more than $\sqrt{2}/4$). Note that diagonal moves on grids are not allowed. Second, our formulation applies to arbitrary graphs including grids of arbitrary dimensions. As such, our algorithm does not depend on a L_1 distance heuristic and applies to a more general setting.

Algorithmic Solution Sketch

In (Yu and LaValle 2012), adapting a dynamic network flow approach, we convert a given graph into a flow network. We illustrate the process as follows. Given the graph G in Fig. 1(a) and a natural number T, we create $2T + 1$ copies of



(a) (b) (c)

Figure 1: a) A simple graph G. b) A gadget for splitting an undirected edge through time steps. c) Part of the time-expanded network ($T = 2$).

vertices from G, with indices $0, 1, 1', \ldots$, as shown in Fig. 1(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 $t, t + 1$, $0 \leq t < T$, we then add the gadget shown in Fig. 1(b) between $u(t)', v(t)'$ and $u(t + 1), v(t+1)$. This gadget ensures that two agents cannot travel in opposite directions on an edge in the same time step. To finish the construction, for each vertex $v \in G$, we add one edge between every two successive copies (i.e., we add the

*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.

edges $(v(0), v(1)), (v(1), v(1)'), \ldots, (v(T), v(T)')$). These correspond to the green and blue edges in Fig. 1(c) [1].

With the network, it is possible to setup an integer multiflow problem and solve the ILP with generic linear programming packages[2]. Note that we are interested in finding the minimal T such that a feasible integer solution exists. To do this, we start with $T = 1$ and double up until a feasible solution is found; the optimal T is then obtained by a binary search. The algorithm is always complete since it is equivalent to a breadth first search for a large T (i.e., if a solution does not exist, we will eventually know that).

Computational Results

Computing optimal solutions. Somewhat surprisingly (since ILP is in general NP-complete), our solution approach obtains time optimal solutions to many input in reasonable amount of time[3]. We highlight a few of these results here. The first example is a variant of the 8-puzzle. Given an arbitrary setup (e.g. Fig. 2(a)), the goal state is that of Fig. 2(b).

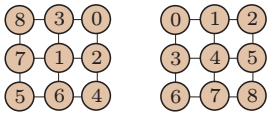


Figure 2: A variant of the 8-puzzle.

For this problem, our test runs show that it generally takes no more than 5 steps to return an arbitrary configuration to the desired goal configuration. For the given problem instance, 4 steps is enough. The 4 synchronous moves are shown in Fig. 3. For this problem instance, the total computation time is 0.1 second. Moving to a larger example with a 4×4 grid and 16 agents, a typical time optimal solution is obtained in 114.6 seconds. We also evaluated the algorithm on larger

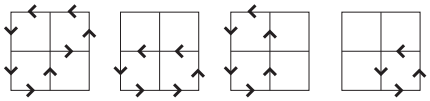


Figure 3: A 4-step solution from our algorithm.

problem instances. One such instance is a 20×15 grid graph with 20% of the vertices removed. For 20 agents with randomly selected start and goal locations (start and goal of different agents may overlap), it takes about 58 seconds (on average) to compute the time optimal solution, which takes a total of about 25 steps. When we bump the number of agents to 40, the computational time becomes about 9 minutes, and the optimal solution takes about 30 steps.

As a heuristic for solving larger problems. Since our algorithm works well on instances with limited sizes, we also exploit its use as a *generic heuristic* for helping solve large problem instances (32×32 grid with 20% vertices removed, 25-125 agents). By generic, we mean that our algorithm does not depend on specific graph and agent configuration; we simply apply it to resolve local conflicts on a neighborhood of size up to 8×8. Each instance is allowed to run a maximum of 10 seconds. The results, each as an average over 500 runs, are listed in Table 1, along with other statistics (we used Java and expect a 2-4x speedup from a C++ implementation).

Table 1: Evaluation of our algorithm as a generic heuristic.

	Number of Agents				
	25	50	75	100	125
Running time (s)	0.038	0.225	0.732	1.944	4.935
% goals reached	100.00	99.95	99.78	98.84	98.47
Ave. path length	24.61	24.84	25.03	25.52	26.26
Heu. path length	24.60	24.67	24.56	24.60	24.51
Length difference	0.01	0.17	0.47	0.92	1.75

Conclusion and Future Work

In this work, we optimally solve a version of the multi-agent path planning problem using an ILP formulation obtained from the time-expanded network of the original graph. We observe that the algorithm can provide time optimal paths for many reasonable sized, complex problems efficiently. Viewing its effectiveness on solving small problem instances, we successfully used the algorithm as a generic heuristic for quickly solving large problem instances. We note that many open questions remain, for example: 1. Could the flow network be built using fewer vertices/edges? 2. Can we improve the ILP formulation to make optimizers run faster? 3. What about different optimality criteria? 4. How hard is our formulation computationally? Is it NP-hard? NP-complete?

References

Luna, R., and Bekris, K. E. 2011. Push and swap: Fast cooperative path-finding with completeness guarantees. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 294–300.

Silver, D. 2005. Cooperative pathfinding. In *The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, 23–28.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *The Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 173–178.

van den Berg, J.; Snoeyink, J.; Lin, M.; and Manocha, D. 2009. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Proceedings Robotics: Science and Systems*.

Wang, K.-H. C., and Botea, A. 2011. Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research* 42:55–90.

Yu, J., and LaValle, S. M. 2012. Multi-agent path planning and network flow. In *The Tenth International Workshop on Algorithmic Foundations of Robotics*. to appear.

[1] More details and ongoing computational evaluation results can be found at `http://arxiv.org/abs/1204.3830`

[2] We used the Gurobi solver which is free for academic use. Available at `http://www.gurobi.com/`.

[3] The runs were performed on a Quad-core 3.0G machine with 8GB of memory. The program is written in Java.