An Efficient and Complete Approach for Cooperative Path-Finding

Ryan Luna and **Kostas E. Bekris**
University of Nevada, Reno
1664 N. Virginia St., MS 171
Reno, NV 89557 USA
{rluna, bekris}@cse.unr.edu

Introduction

Cooperative path-finding requires the computation of a set of compatible paths for multiple agents operating on a discrete roadmap. The goal of this problem is to navigate each agent to their unique target vertices without simultaneously occupying the same vertex or edge in the roadmap as any other agent. Such a formulation has applications in warehouse management, transportation networks, (dis)assembly, robotic mining, space exploration, and computer games.

The problem of cooperative path-finding has been extensively studied in the literature, with coupled techniques that attempt to intelligently prune the search space into something more tractable while maintaining completeness, as well as decoupled techniques that plan for each agent independently and utilize sophisticated heuristics in order to avoid collisions along these paths.

Coupled search techniques have been employed to separate large problems into fully coupled sub-problems (van den Berg et al. 2009), or segment the roadmap into smaller topologies with known characteristics (Ryan 2007) to solve the composite problem sequentially.

Decoupled techniques employ heuristics that prioritize agents (Erdmann and Lozano-Perez 1986), or tune the velocities on precomputed paths (Kant and Zucker 1986). Modern approaches consider dynamic prioritization and windowed search (Silver 2005), or domain restriction to guarantee completeness and tractability (Wang and Botea 2009).

Contribution

This work provides a novel, algorithmically complete approach for cooperative path-finding that efficiently computes sequential solutions. The algorithm is complete for a general class of problems: instances in which there are at most $n - 2$ agents on a roadmap of n vertices. Experimental results show that the technique is able to compute solutions many orders of magnitude faster than a traditional A^* implementation while maintaining completeness, and in times highly competitive with a state-of-the-art decoupled planner. The approach also makes no assumption regarding the topology of the roadmap, and does not depend on tunable parameters in order to solve a problem effectively.

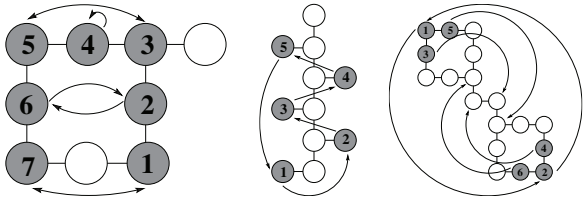

Figure 1: Small cooperative path-finding instances difficult for coupled and decoupled techniques. From left: Loopchain, string, connector.

Push and Swap Approach

The proposed PUSH_AND_SWAP technique utilizes a set of simple and computationally cheap primitives, PUSH and SWAP, to solve cooperative path-finding problems in a sequential manner. As shown in Algorithm 1, the proposed method simply iterates over the set of agents and first *pushes* the agent towards its target, given a set of positions that must remain static. If the agent a is not able to push to its target, it must *swap* positions with the adjacent agent along its shortest path. This loop of *pushing* and *swapping* continues until agent a is at its target. Once this occurs, the target is added into the list of static positions (line 7), and the loop repeats for the next agent. Algorithms for the PUSH and SWAP primitives are omitted for brevity, but are briefly discussed below.

Algorithm 1 PUSH_AND_SWAP($\mathcal{A}, \mathcal{S}, \mathcal{T}$)

```
1: U ← ∅
2: for all a ∈ A do
3:     while S[a] ≠ T[a] do
4:         if PUSH(a, U) == FALSE then
5:             if SWAP(a) == FALSE then
6:                 return FALSE (i.e., Failure)
7:     U ← U ∪ T[r]
8: return TRUE (i.e., Success)
```

Push primitive: The push primitive navigates an agent along its shortest path, potentially detouring other agents in the process. In short, when an agent is blocking the shortest path of the pushing agent a, PUSH attempts to move the agent blocking a away from its current position, allowing a to advance along its shortest path. There are restrictions, however, on what can be "pushed". The set $\mathcal{U}$ contains vertices of agents that have already planned and are at their tar-

Problem	Coupled A*	WHCA*(5)	Push-and-Swap
Loop-Chain	∞	∞	2.99
String	207.1	1.04	0.775
Connector	∞	∞	3.57

Table 1: Computation time in milliseconds for the benchmarks in Figure 1. ∞ represents no solution computed.

get position; these agents should not be moved. If agent a does not reach its target because it cannot make progress by "pushing", it returns false, indicating a SWAP is necessary.

Swap primitive: The swap primitive switches the position of an agent a with an agent b that is adjacent to it along the shortest path. One requirement for SWAP is that after execution, the only agents that have changed positions are a and b. This is necessary to ensure completeness. Maintaining this constraint is feasible by allowing all agents to move during the swap, and reversing the actions taken once the swap is complete. In order to maintain the swap, agent a will reverse using b's paths and vice-versa.

Completeness

PUSH_AND_SWAP provides completeness for problems in which there are at least two empty vertices in the roadmap. The theorem can be proven with the following lemmas:
1) The path-finding instance is solvable if and only if SWAP can transfer two agents to the vicinity of a vertex v with a degree at least 3 together with two empty vertices.
2) After each iteration of PUSH and SWAP, at least one agent will make progress along the shortest path to its target.
3) The PUSH and SWAP operators do not disturb agents already at their target vertices.
4) If the initial configuration is solvable, any permutation achieved using PUSH and SWAP operators will be solvable.

Results

The proposed technique was evaluated with a set of small benchmark problems, Figure 1, where it is possible to compare against a complete, centralized A^* implementation, as well as a modern and well known decoupled technique, WHCA* (Silver 2005), which can compute larger scale solutions in a reasonable amount of time. The computation times for these problems is seen in Table 1. All experiments were deemed a failure after 60 minutes of computation.

Another issue with cooperative path-finding techniques is scalability. Coupled planners become infeasible after just a few agents, and decoupled planners suffer from deadlocks because of inherent greediness. When compared with WHCA*, the proposed technique scales very well in a randomly populated 20x30 grid world with 20% obstacle coverage. Figure 2 shows a graph of the Push-and-Swap technique against WHCA* with two planning window sizes. It should be noted that Push-and-Swap never failed to compute a solution, and found solutions in times competitive with the smaller WHCA* window size. WHCA* suffered from deadlocks with 50 or more agents. [1]

[1] A detailed description of this work can be found at www.cse.unr.edu/robotics/pracsys/complete_multi_robot_planning

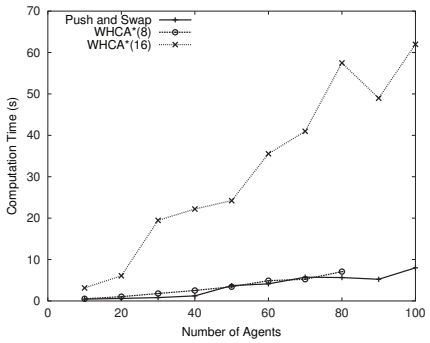

Figure 2: Computation time for agents placed randomly in a 20x30 grid. Values are averages of 20 runs.

Discussion

It is important to note that the solutions generated by PUSH_AND_SWAP are not optimal, and investigating whether or not an optimal set of paths can be computed using the two primitives in a reasonable amount of time is of particular interest. Depending on the particular problem domain, techniques that provide Pareto optimality (Ghrist, O'Kane, and LaValle 2005) may also be applicable.

In addition to optimality, it would also be desirable to provide completeness in all instances solvable using a sequential formulation. PUSH_AND_SWAP provides completeness in problems where there are at least two empty vertices in the roadmap. The classic "15-puzzle" problem can easily be formulated as cooperative path-finding, however this instance would only have one empty vertex. It may be possible to create a variant of SWAP to take advantage of redundant loops in the grid in order to solve such instances.

References

Erdmann, M., and Lozano-Perez, T. 1986. On multiple moving objects. In *IEEE Intern. Conference on Robotics and Automation (ICRA)*, 1419–1424.

Ghrist, R.; O'Kane, J. M.; and LaValle, S. M. 2005. Computing pareto optimal coordinations on roadmaps. *International Journal of Robotics Research* 24(11):997–1010.

Kant, K., and Zucker, S. 1986. Towards efficient trajectory planning: The path-velocity decomposition. *International Journal of Robotics Research (IJRR)* 5(3):72–89.

Ryan, M. R. K. 2007. Graph decomposition for efficient multi-robot path planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003–2008.

Silver, D. 2005. Cooperative pathfinding. In *The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'05)*, 23–28.

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

Wang, K.-H. C., and Botea, A. 2009. Tractable Multi-Agent Path Planning on Grid Maps. In *International Joint Conference on Artificial Intelligence IJCAI-09*, 1870–1875.