Towards Search-Free Multi-Agent Path Finding

Adi Botea
IBM Research, Dublin, Ireland

Daniel Harabor, Ko-Hsin Cindy Wang
NICTA* and the Australian National University

Introduction

Multi-agent path finding (MAPF) has received a growing interest in recent years. Approaches to this problem range from centralized search (e.g., A*) in the combined state space of all units to decoupled methods that split the computation into several searches. For example, algorithms such as FAR (Wang and Botea 2008) trade completeness and optimality for speed and scalability. FAR performs one individual search for each unit, ignoring the presence of other units. Very little additional search is required. Other algorithms are more search intensive, searching, for example, in a larger, combined state space. Not surprisingly, even the fastest existing MAPF algorithms are CPU intensive, especially on instances of growing size.

The goal of this work is a MAPF method that performs *no or very little search* at runtime. The key idea is taking advantage of recent results in compressing all-pairs shortest path (APSP) data. SILC (Sankaranarayanan, Alborzi, and Samet 2005) and CPDs (compressed path databases) (Botea 2011) are techniques with powerful compression capabilities, allowing to store in memory APSP data corresponding to graphs of a considerable size. Given *any* start–target pair, APSP can quickly provide the next move towards the target, with no runtime search. Such a property is very useful in MAPF, where it can eliminate not only a set of initial path searches, but also subsequent searches (re-planning) caused by factors such as collisions.

Our contributions are as follows: We introduce MARS (Multi-Agent Ring Slidable), an algorithm that combines ideas from the MAPP algorithm (Wang and Botea 2009) and CPDs (Botea 2011) to eliminate expensive runtime searches. We define a class of instances where MARS is complete. We prove theoretical properties of the algorithm. To the best of our knowledge, this could be the first work that aims at eliminating runtime search in MAPF.

MAPP, the selected baseline algorithm, is complete on a subclass of instances called SLIDABLE (Wang and Botea 2009). The idea of using CPDs in MAPF is not limited to MAPP. CPDs could possibly be studied in combination with other MAPF algorithms as well, such as FAR (Wang and Botea 2008) or Push and Swap (Luna and Bekris 2011).

*NICTA is funded by the Australian Government.

Problem Description

We assume the input graph is undirected. An instance with n units consists of n start–target pairs such that no units share their start (or target). A move to an adjacent location l is possible iff l is empty and no other unit plans to move to l at the current moment. Given two arbitrary locations (nodes) l and t in the graph, CPD(l, t) returns the first move along an optimal path from l to t. We do not require solution optimality.

MARS Algorithm

We define a class of instances on which our algorithm is complete. It is an adaptation of the SLIDABLE class.

Definition 1 (RING-SLIDABLE class)**.** *An instance belongs to the class* RING-SLIDABLE *if it satisfies the following conditions:*

1. *There are no adjacent targets.*
2. *For any three consecutive locations $l_1 - l_2 - l_3$, except triples where l_1 or l_2 is a target, there is an alternate path Ω such that Ω contains no target and does not contain l_2.*
3. *In the initial state, each unit has a blank on an adjacent position that is not a target.*
4. *Targets and initial positions do not overlap.*

Definition 2 (Active move of u)**.** *Let l and t be the current location and the target of a unit u. Let l' be CPD(l, t) and l'' be CPD(l', t). The active move of u from location l is CPD(l, t), except for the following situation. When the CPD(l, t) move leads to a foreign target (i.e., l' is a foreign target), the active move will take the unit along the alternate path $\Omega(l, l'')$. As soon as u reaches l'', it reverts back to CPD-provided active moves.*

Compared to MAPP, MARS differs in two important ways: CPDs quickly identify moves towards each unit's target location without explicit state-space search; and we allow on-the-fly navigation around foreign targets.

Definition 3. *The* local zone *of an active unit u contains two adjacent locations: the current location of u, loc(u), and the previous location pre(u), unless u is in its initial state.*[1]

[1]This is similar to the private zone in MAPP. Our definition could easily be relaxed to allow seeking a blank on any adjacent position, not only behind a unit.

Algorithm 1 shows our method in pseudocode. Initially, all units belong to the set A of active units. A unit is either active or solved. They become solved as soon as they reach their target. Active units are totally ordered. The active unit with the highest priority is called the master unit $\bar{u}$.

Algorithm 1 MARS

```
1:  while A ≠ ∅ do
2:    for all u ∈ A in decreasing order of priority do
3:      if destination of active move is empty then
4:        make active move
5:      else if can bring a blank in front along Ω then
6:        bring blank in front
7:        make active move
8:      else
9:        do nothing
10:     if u just reached target then
11:       remove u from A {u is solved}
12:       if u is the master and A ≠ ∅ then
13:         label a different active unit as the master ū
14:         make sure ū has a blank on an alternate path
```

At line 5, Ω is the alternate path for the triple pre(u), loc(u), and next(u), the next location indicated by the active move. A blank can be brought in front of u along Ω iff: Ω contains a blank at a position p; and no position in Ω between next(u) and p, including next(u) and p, belongs to the local zone of a unit with a higher priority than u. Bringing a blank in front of u is performed by pushing units one by one along Ω from next(u) towards p, similarly to the way the blank travels in the sliding-tile puzzle.

Line 14 makes sure that a freshly selected master unit can make its first active move, since it guarantees that the alternate path Ω associated with pre($\bar{u}$), loc($\bar{u}$) and next($\bar{u}$) will contain at least one blank. One simple way to ensure the property stated on line 14 is to perform the so-called reverse repositioning (Wang and Botea 2009). Reverse repositioning undoes, in reverse global order, moves of the active units, but not moves of the solved units. It can easily be proven that a position with the desired property for $\bar{u}$ can always be reached with 0 or more reverse repositioning moves. The idea is that, in the worst case, all active units could revert to their original positions, where they have a blank next to them (to bring to front if needed) according to Definition 1.

Lemma 1. *In the nested for loop, every iteration that processes $\bar{u}$ allows it to make an active move towards the target. Hence $\bar{u}$ reaches the target in a finite number of iterations.*

Proof. According to condition 3 in Definition 1 and line 14 in Algorithm 1, $\bar{u}$ can always make the first active move. After each active move, the position behind $\bar{u}$ is empty and no other unit can occupy it, since it belongs to the local zone of $\bar{u}$. Therefore, Ω has at least one blank position. Furthermore, all other units whose local zone could possibly intersect with Ω have a lower priority. As a result, all conditions for bringing a blank at the front along Ω (if needed) are satisfied. □

We remark that the ability to reach the target is not restricted to the master unit at hand. Many units could reach their target without having the master unit status.

Lemma 2. *After reaching its target, a unit does not interfere with the rest of the problem.*

Proof. All moves are either active moves (lines 4, 7), or blank travel moves along alternate paths Ω (line 6), or reverse versions of such moves (line 14). All these explicitly avoid foreign targets. □

Theorem 1. *Algorithm* MARS *terminates on a* RING-SLIDABLE *instance, producing a valid solution.*

Proof. Every unit with the master status gets solved, after which it does not interfere with the rest of the problem. Hence there will be no active units after a finite number of iterations. □

Discussion and Future Work

The pseudocode presented earlier assumes that the CPD and the alternate paths Ω are readily available. CPDs can be precomputed once per map and reused over all instances on that map (Botea 2011). The alternate paths could be precomputed as well. However, given a new instance, those alternate paths that intersect a target would have to be recomputed (Wang and Botea 2009). The runtime search in MARS could be as small as re-computing such a subset of alternate paths. This could potentially be performed quickly, given that the two end points of a alternate path are so close to each other. In effect, one advantage of MARS is the speed. The price to pay is using memory for the CPD.

The definition of RING-SLIDABLE instances requires the existence of alternate paths for every triple of adjacent locations. In contrast, SLIDABLE (Wang and Botea 2009) imposes a similar condition only for triples along a precomputed path of a unit. This suggests that the RING-SLIDABLE class might be more restricted than SLIDABLE.

In the future, we plan to implement and evaluate this method in terms of speed, solution quality, and completeness range, measured and the percentage of units and percentage of instances that satisfy the RING-SLIDABLE conditions.

References

Botea, A. 2011. Ultra-fast Optimal Pathfinding without Runtime Search. In *Proceedings of AIIDE-11*, 122–127.

Luna, R., and Bekris, K. E. 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *IJCAI-11*, 294–300.

Sankaranarayanan, J.; Alborzi, H.; and Samet, H. 2005. Efficient query processing on spatial networks. In *ACM workshop on Geographic information systems*, 200–209.

Wang, K.-H. C., and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *ICAPS*, 380–387.

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