Multi-Agent A*

AAAI 2015 Submission 2327

Abstract

abstract

Notes

PAD* notes: should we check last element of CLOSED instead of first in OPEN?

For $i = 1, \ldots, N$, agent i has a graph defined by vertex set V_i and edge weights $c_i : V_i \times V_i \to \mathbb{R}^+$. We consider constraints Φ, each defining a relation on the relative visit times of two sets of nodes $\Phi^-, \Phi^+ \subset \cup_i V_i$. There are four constraint types:

- $\min \le \min$ (opening a door)
- $\max \le \min$ (closing a door)
- $\max \le \max$ (restoration)
- $\min \le \max$ (sequence)

Given individual plans for each agent, an optimal multi-agent plan can be constructed greedily forward in time, when all constraints are of the opening or closing kind. An optimal multi-agent plan can be constructed greedily backward in time when all constraints are of the closing or restoration kind. Since restoration and sequence constraints can be represented in terms of opens and closes, we consider the former as the only types without loss of generality.

Representing restoration contraints in terms of opens and closes: suppose B restores (i.e. cleans after) A. Then create a copy B' of B. Let B' close A, and B' open any agent's goal node. To deal with the boundary condition where neither A nor B are visited, the start node should have a similar copy, reachable at zero cost.

Representing sequence contraints in terms of opens: suppose (A, B) are a mandated sequence. Then create a copy B' of B. Let A open B', and B' open any agent's goal node.

NP-completeness proof:

Let's say there are m clauses and the variables are $x_1, x_2, \ldots, x_n$. Draw the edges $(start_1, x_1)$, $(start_1, \neg x_1)$, (x_i, x_{i+1}), $(x_i, \neg x_{i+1})$, $(\neg x_i, x_{i+1})$, $(\neg x_i, \neg x_{i+1})$, $(x_n, goal_1)$, $(\neg x_n, \neg goal_1)$. For the other agent, draw the edges $(start_2, c_0)$, $(c_{i-1}, req_{i,j})$, $(req_{i,j}, c_i)$, $(c_m, goal_2)$. Here, $req_{i,j}$ is a door that needs to be opened by the node corresponding to the j'th variable of the i'th clause.

$g_{s_1,\ldots,s_N}(s_i)$ is the minimum time by which agent i can reach state s_i, assuming every agent j will go to s_j along some known path (following $bp(s_j)$, say) and wait there forever. Note that the state space for each agent is the Cartesian product of its graph's vertex set, and the set of partial permutations of constraints.

Define $f(s_1, \ldots, s_N) = \max_i g_{s_1,\ldots,s_N}(s_i) + \epsilon h(s_i)$. This key is impractical to compute over all possible tuples. Further approximations will need to be made. Note that $(s_1, \ldots, s_N)$ range over the goal nodes as well as the $OPEN$ list.

Even if we could compute f, does it guarantee $N\epsilon$ optimality? To prove that, it would suffice to show that we only expand nodes whose individual-agent g-values are ϵ-optimal. However, the logic fails because passing through a door-opening node can result in a sudden drastic decrease in the $g_{s_1,\ldots,s_N}$ value, due to constraints being lifted. To remedy this, instead of defining g by "wait-forever" semantics, we could treat doors as being opened after the partial paths to $(s_1, \ldots, s_N)$, at some lower bound time.

A lower-bound for $f(s_1, \ldots, s_n)$ could be computed as $f(s_j)$ by building a coarse search tree at the level of constraint sequences, where a macro-expansion consists of searching toward each possible next constraint node.

Want to prove that $g(s)$ is ϵ-optimal at expansion (when its jointPriority value is minimal), and thus the joint paths are $N\epsilon$ optimal. Suppose $g(s) > \epsilon g^*(s)$. As induction hypothesis, let $s' \in OPEN$ be such that $g(s') \le \epsilon g^*(s')$. Fix a priority-minimizing tuple $(s_1, \ldots, s, \ldots, s_n)$ for s. Now consider the tuple $(s_1, \ldots, s', \ldots, s_n)$. If its priority were less, it would follow that s' will be expanded before s. But, path to s might open doors much sooner than path to s'!

Pending further insights, it seems the state is forced to remember sequences of contrained nodes instead of merely sequences of constraints. This way, we guarantee prefix-suboptimality, and as a bonus get global ϵ-suboptimality without the extra factor of N.

Algorithm 1 $MultiAgentA^*HighLevel()$

```
∀i : s_i := start_i
while g(goal_1, ..., goal_N) > f(s_1, ..., s_N) do
  ∀i : expand(s_i)
  (s_1, ..., s_N) = argmin_{s_1,...,s_N} f(s_1, ..., s_N)
end while
```

Algorithm 2 $MultiAgentA^*()$

```
∀i : s_i := start_i
OPEN := {start_1, ... start_N}
while jointPriority(goal_1, ..., goal_N) > OPEN.fmin() do
  expand(OPEN.extractMin())
end while
```

Algorithm 3 $expand(s)$

```
for all s' ∈ successors(s) do
  if g(s') > g(s) + c(s.node, s'.node) then
    g(s') := g(s) + c(s.node, s'.node)
    bp(s') := s
    insert s' in OPEN
  end if
end for
```

Algorithm 4 $f(s)$

```
f := ∞
for all tuples s_{1...N} ∈ OPEN that include s do
  f := min(f, jointPriority(s_{1...N}))
end for
return f
```

Algorithm 5 $successors(u, seq_u)$

```
succlist := ∅
for all v such that c(u, v) < ∞ do
  seq_v := seq_u
  for all min constraints Φ^± ∉ seq_u with v ∈ Φ^± do
    append Φ^± to seq_v
  end for
  for all max constraints Φ^± with v ∈ Φ^± do
    remove Φ^± from seq_v
    append Φ^± to seq_v
  end for
  push (v, seq_v) onto succlist
end for
return succlist
```

Algorithm 6 $greedyCost(\pi_1, \ldots, \pi_n)$

```
∀Φ : trigger(Φ) := Φ is open or Φ^- is absent in π_{1...N}
∀ı ∈ 1...N : u_i := v_i := start_i
events := {0 : {1...N}}
while events is not empty do
  (time, ids) := extractMin(events)
  for all i ∈ ids do
    u_i := v_i
    for all constraints Φ^- with v ∈ Φ^- do
      if Φ is an open constraint or Φ^- appears in no
      suffix of π_{1...N} after u_{1...N} then
        trigger(Φ) := true
      end if
    end for
  end for
  for all i ∈ 1...N with u_i = v_i ≠ goal_i do
    v_i := successor of u_i on π_i
    for all constraints Φ^+ with v ∈ Φ^+ do
      if ¬trigger(Φ) then
        v_i := u_i and continue with the next i
      end if
    end for
    events(time + c(u_i, v_i)).insert(i)
  end for
end while
if u_{1...N} = goal_{1...N} then
  return time
else
  return ∞
end if
```

Algorithm 7 $jointPriority(s_1, \ldots, s_N)$

```
build π_{1...N} by following bp() from s_{1...N}
for all i ∈ 1...N such that s_i.node ≠ goal_i do
  extend π_i with a zero cost edge to a node that opens
  all doors the agent can open, followed by goal_i at cost
  εh(s_i)
end for
return greedyCost(π_1, ..., π_n)
```