Delayed Proofs

For easier reference, we copy the execution procedure and the general algorithm framework below.

Procedure 1: Execute ($\mathcal{G} = (\mathcal{V}, \mathcal{E})$)
Lines highlighted in blue are activated to compute the cost of a TPG, and can be omitted for the mere purpose of execution.

---

1 Define a counter $cost$;
2 **Function** $\text{INIT}_{\text{EXEC}}(\mathcal{G})$
3 $\quad cost \leftarrow 0$;
4 $\quad$ Mark all vertices in $\mathcal{V}_0 = \{v_0^i : i \in \mathcal{A}\}$ as satisfied;
5 $\quad$ Mark all remaining $v \in (\mathcal{V} \setminus \mathcal{V}_0)$ as unsatisfied;
6 **Function** $\text{STEP}_{\text{EXEC}}(\mathcal{G}, i)$
7 $\quad$ **if** $\forall k : v_k^i$ satisfied **then**
8 $\quad\quad$ **return** NULL;
9 $\quad cost \leftarrow cost + 1$;
10 $\quad v \leftarrow v_k^i : v_k^i$ unsatisfied and $\forall k' < k, v_{k'}^i$ satisfied;
11 $\quad$ **forall** $(u, v) \in \mathcal{E}$ **do**
12 $\quad\quad$ **if** u unsatisfied **then**
13 $\quad\quad\quad$ **return** NULL
14 $\quad$ **return** v
15 **Function** $\text{EXEC}(\mathcal{G})$
16 $\quad \text{INIT}_{\text{EXEC}}(\mathcal{G})$;
17 $\quad$ **while** there exists unsatisfied vertex in $\mathcal{V}$ **do**
18 $\quad\quad$ Define a set $\mathcal{S} \leftarrow \varnothing$;
19 $\quad\quad$ **forall** agent $i \in \mathcal{A}$ **do**
20 $\quad\quad\quad$ Add $\text{STEP}_{\text{EXEC}}(\mathcal{G}, i)$ into $\mathcal{S}$;
21 $\quad\quad$ **forall** $v \in \mathcal{S}$ **do**
22 $\quad\quad\quad$ **if** $v \neq$ NULL **then**
23 $\quad\quad\quad\quad$ Mark v as satisfied;
24 $\quad$ **return** $cost$;

Proofs for Section Temporal Plan Graph

Lemma 1. Executing $\mathcal{G}$ encounters a deadlock if and only if there exists a cycle in $\mathcal{G}$.

Proof. Assuming that a deadlock is encountered in the t^{th} iteration of the while-loop (on line 17 of Procedure 1). Let $\mathcal{V}'$ denote that set of all vertices that are unsatisfied, which is non-empty by the definition of deadlock. Assume towards contradiction that $\mathcal{G}$ is acyclic, then there exists a topological ordering (i.e. a linear ordering of its vertices such that for every directed edge (u, v), u comes before v in the ordering) of $\mathcal{V}'$. In this case, $\mathcal{S}$ must contain the first vertex v_{first} in the ordering of $\mathcal{V}'$, because it satisfies both conditions in $\text{STEP}_{\text{EXEC}}$:

1. $v_{\text{first}} = \arg\min_k (v_k^i$: agent $i \in A$, v_k^i is unsatisfied), and
2. For all $(u, v_{\text{first}}) \in \mathcal{E}$, u is satisfied.

This contradicts the deadlock condition that $\mathcal{S} = \{\text{NULL}\}$. □

Algorithm 2: Replanning
HEURISTIC, TERMINATE, CYCLEDETECTION, and BRANCH are modules that will be specified later. $\mathcal{X}$ denotes some auxiliary information accompanying a TPG, whose format is defined by the set of modules.

Input: TPG $\mathcal{G}_{\text{root}} = (\mathcal{V}, \mathcal{E}_1, (\mathcal{S}_{\mathcal{E}2}, \mathcal{N}_{\mathcal{E}2}))$
Output: TPG $\mathcal{G}_{\text{result}}$

Initialize an empty priority queue $\mathcal{Q}$;
$h_{\text{root}} \leftarrow \text{HEURISTIC}(\mathcal{G}_{\text{root}}, \mathcal{X}_{init})$;
$\mathcal{Q}.push((\mathcal{G}_{\text{root}}, \mathcal{X}_{init}), 0, h_{\text{root}})$;
while $\mathcal{Q}$ is not empty **do**
$\quad ((\mathcal{G}, \mathcal{X}), g, h) \leftarrow \mathcal{Q}.pop()$;
$\quad (g', \mathcal{X}', (v_k^i, v_s^j)) \leftarrow \text{BRANCH}(\mathcal{G}, \mathcal{X})$;
$\quad$ **if** $\text{TERMINATE}(\mathcal{G}, \mathcal{X}')$ **then**
$\quad\quad fix$ all edges in $\mathcal{S}_{\mathcal{E}2}$ of $\mathcal{G}$;
$\quad\quad$ **return** $\mathcal{G}$;
$\quad \mathcal{G}_{\text{f}} \leftarrow fix(\mathcal{G}', (v_k^i, v_s^j))$;
$\quad$ **if not** $\text{CYCLEDETECTION}(\mathcal{G}_{\text{f}}, (v_k^i, v_s^j))$ **then**
$\quad\quad h_f \leftarrow \text{HEURISTIC}(\mathcal{G}_{\text{f}}, \mathcal{X}')$;
$\quad\quad \mathcal{Q}.push((\mathcal{G}_{\text{f}}, \mathcal{X}'), g + g', h_f)$;
$\quad \mathcal{G}_{\text{r}} \leftarrow reverse(\mathcal{G}', (v_k^i, v_s^j))$;
$\quad$ **if not** $\text{CYCLEDETECTION}(\mathcal{G}_{\text{r}}, (v_{s+1}^j, v_k^i))$ **then**
$\quad\quad h_f \leftarrow \text{HEURISTIC}(\mathcal{G}_{\text{f}}, \mathcal{X}')$;
$\quad\quad \mathcal{Q}.push(\mathcal{G}_{\text{r}}, \mathcal{X}'), g + g', h_r)$;
throw exception "No solution found";

Proposition 2 (Collision-Free)**.** Let $\mathcal{G}$ be a TPG constructed from a MAPF solution. Assuming $\mathcal{G}$ is executed as in Procedure 1 and an agent i is moved to its k^{th} location l_k^i at timestep t iff vertex v_k^i is satisfied in the t^{th} iteration of the while-loop on line 17, any two agents i, j never collide.

This proposition is similar to lemma 4 in (Hönig et al. 2016) with slightly different terms. We include a proof for completeness.

Proof. Assume towards contradiction that i and j collide because they are at the same location at the same timestep t, i.e. after the t^{th} iteration, $v_k^i = \arg\max_k(v_k^i : v_k^i$ satisfied), $v_s^j = \arg\max_s(v_s^j : v_s^j$ satisfied), and $l_k^i = l_s^j$. Either edge (v_{k+1}^i, v_s^j) or (v_{s+1}^j, v_k^i) should be in $\mathcal{E}$. But $(v_{k+1}^i, v_s^j) \notin \mathcal{E}$ as otherwise v_s^j cannot be satisfied since v_{k+1}^i is unsatisfied; similarly $(v_{s+1}^j, v_k^i) \notin \mathcal{E}$ as otherwise v_k^i cannot be satisfied since v_{s+1}^j is unsatisfied. This shows a contradiction.

If they collide because i leaves a location at a timestep t, and j enters the same location at timestep t, then $l_{k-1}^i = l_s^j$ and vertices v_k^i and v_s^j are satisfied exactly in the t^{th} iteration of the loop. However, this is impossible since either (v_k^i, v_s^j) or (v_{s+1}^j, v_{k-1}^i) is in $\mathcal{E}$, but the out-going vertex in neither of these edges are satisfied before the t^{th} iteration. □

Corollary 3. Let $\mathcal{G}$ be a collision-free TPG. If we replace an arbitrary Type 2 edge (v^j_{s+1}, v^i_k) in it with (v^i_{k+1}, v^j_s), the TPG remains to be collision-free.

Proof. This is observed from the fact that the proof of the above proposition argues the non-existence of two pairs of edges $(v^i_{k+1}, v^j_s) - (v^j_{s+1}, v^i_k)$ and $(v^i_k, v^j_k s) - (v^j_{s+1}, v^i_{k-1})$. Since in both pairs, the two edges are equal upon the replacement, the proof remains exactly the same after we perform an arbitrary replacement. □

Proposition 4. Let $\mathcal{G}$ be a TPG constructed from a MAPF solution $\mathcal{P}$, the cost of $\mathcal{G}$ is no greater than the sum of travel time for agents following $\mathcal{P}$.

We rely on the following lemma to prove this proposition.

Lemma A. If an arbitrary agent i is planned to move to location l^i_k at a time $t = t^i_k$ in $\mathcal{P}$, then vertex v^i_k is either satisfied or can be satisfied in the t^{th} iteration of the while-loop.

Proof (of Lemma A). We induct on t. When $t = 0$, this holds by the functionality of $\text{INIT}_{\text{EXEC}}$. For $t > 0$, we consider the non-trivial case that the index k of v^i_k is non-zero. For the Type 1 edge of v^i_k, since $t^i_{k-1} \leq t^i_k - 1 = t - 1$, v^i_{k-1} is satisfied by an inductive hypothesis. Let v^j_s be an arbitrary Type 2 in-neighbor of v^i_k, by construction of Type 2 edges, $t^j_{s-1} < t^i_k$, so $t^j_{s-1} \leq t - 1$. This shows that all in-neighbors of v^i_k must be satisfied after the $(t-1)^{\text{th}}$ iteration, thus v^i_k can be satisfied in the t^{th} iteration if it has not been satisfied yet. □

Proof (of Proposition 4). Lemma A shows that if $\mathcal{P}$ plans an agent i to enter its goal location at time t^i_{zi}, then all vertices of i are satisfied after the $(t^i_{zi})^{\text{th}}$ iteration, i.e. agent i contributes to $cost$ by at most t^i_{zi} units. Therefore $cost \leq \sum_{i \in \mathcal{A}} t^i_{zi}$, which is the sum of travel times of all agents in $\mathcal{P}$. □

Corollary 5 (Deadlock-Free)**.** If a TPG $\mathcal{G}$ is constructed from a MAPF solution $\mathcal{P}$, then it is deadlock-free.

Proof. If $\mathcal{G}$ contains a deadlock, then its execution would enter the while-loop for infinitely many iterations, and in each iteration, $cost$ strictly increases. Thus $cost = \infty$. Yet the sum of travel time of $\mathcal{P}$ is always finite, contradicting Proposition 16. □

Proofs for Section Switchable TPG

Theorem 6. Let $\mathcal{G}$ be a switchable TPG constructed as in Construction 1, there always exists a finite-cost, collision-free standard TPG that can be produced from $\mathcal{G}$.

Proof. One naive solution $\mathcal{G}_{\text{naive}}$ is produced by fixing all switchable edges in $\mathcal{G}$. $\mathcal{G}_{\text{naive}}$ has a finite cost (i.e. is deadlock-free) because by Corollary 5, an initial TPG $\mathcal{G}_0$ constructed from a MAPF solution is deadlock-free. And by Lemma 1, a TPG has a deadlock iff it contains a cycle, so it suffices to argue that step 2 and 3 in Construction 1 does not introduce a new cycle. This holds since step 2 has no effect once we fix all switchable edges. Step 3 behaves as expanding a pre-existing edge (v^i_{k-1}, v^i_k) into a line of connecting edges. If a cycle exists in $\mathcal{G}_{\text{naive}}$, it either involves edges in this line or not. In the latter case, this cycle would exist exactly in $\mathcal{G}_0$, which is impossible. In the former case, the entire line has to be contained in this cycle, in which case (v^i_{k-1}, v^i_k) along with the remaining component of this cycle would form a cycle in $\mathcal{G}_0$, which is impossible.

Executing $\mathcal{G}_{\text{naive}}$ is collision-free because the exact same proof of Proposition 2 shows that two agents cannot collide if none of them is the delayed agent i or if the most-recently satisfied vertex of i is not a dummy vertex. So we may assume without loss of generality that agent i and j collide when i has already entered location l^i_{k-1}. However, such a collision is impossible since any vertex v^j_s for $j \neq i$ corresponding to the same location cannot be satisfied before v^i_k is satisfied. □

Proofs for Section Algorithm

Lemma 7. Let $\mathcal{G}_{\text{switch}}$ be a switchable TPG and $\mathcal{G}$ be an arbitrary standard TPG produced from $\mathcal{G}_{\text{switch}}$. The partial cost of $\mathcal{G}_{\text{switch}}$ is no greater than the cost of $\mathcal{G}$.

Proof. Let $\mathcal{G}_{\text{red}}$ be the reduced standard TPG of $\mathcal{G}_{\text{switch}}$ that contains only its non-switchable edges. Consider running Procedure 1 on $\mathcal{G}$ and $\mathcal{G}_{\text{red}}$, respectively. Since an edge appears in $\mathcal{G}_{\text{red}}$ must also appear in $\mathcal{G}$, we can inductive show that in any call to $\text{STEP}_{\text{EXEC}}$, if a vertex v can be marked as satisfied in $\mathcal{G}$, then it can be marked as satisfied in $\mathcal{G}_{\text{red}}$ as well. Therefore the total timestep to satisfy all vertices in $\mathcal{G}_{\text{red}}$ cannot exceed that in $\mathcal{G}$. □

Proofs for Section Graph-based Modules

Theorem 11. Given a TPG, compute the longest path from vertex v^i_0 to vertex v^i_{zi} for each $i \in \mathcal{A}$. Taking the sum of lengths of all such longest paths, this equals the cost of this TPG.

Proof. We again refer back to Procedure 1. Fix a longest path from vertex v^i_0 to vertex v^i_{zi}. We prove by induction that the distance from v^i_0 to a vertex v^j_s on this longest path is equal to the number of iterations required in the while-loop (line 17) in Procedure 1 to satisfy v^i_k. In the base case, the distance from v^i_0 to itself is indeed 0. In the inductive step, assuming v^j_s is satisfied in the $t-1^{\text{th}}$ iteration, and the longest path from v^i_0 to v^j_s is $t-1$. Then the next vertex $v^{j'}_{s'}$ on the longest path is satisfied in the t^{th} iteration, because:

- #iterations $\geq t$ since v^j_s is a in-neighbor of $v^{j'}_{s'}$ which needs to be satisfied before $v^{j'}_{s'}$.
- #iterations $\leq t$ since otherwise there must be another in-neighbor of $v^{j'}_{s'}$ that is not yet satisfied in the $t-1^{\text{th}}$ iteration, which is going to compose a longer path than the one we look at.

Therefore the cost computed by Procedure 1 which equals the sum of iterations for all agents to reach their goal vertex is exactly the sum of lengths of longest paths □