

# Analyzing Execution Traces – Critical-Path Analysis and Distance Analysis

## Online Appendix: Proofs

Martijn Hendriks<sup>\*†</sup>      Jacques Verriet<sup>\*</sup>      Twan Basten<sup>‡\*</sup>  
 Bart Theelen<sup>\*</sup>      Marco Brassé<sup>§</sup>      Lou Somers<sup>‡§</sup>

This is Online Resource 1 that belongs to the article “Analyzing Execution Traces – Critical-Path Analysis and Distance Analysis” in the International Journal on Software Tools for Technology Transfer (STTT) published by Springer.

**Theorem 1.** *Let  $f$  be an execution, and let  $\epsilon_1, \epsilon_2 \in \mathbb{R}^+ \cup \{\infty\}$  such that  $\epsilon_1 \leq \epsilon_2$ . Then  $\mathcal{G}_{\epsilon_1}(f) \subseteq \mathcal{G}_{\epsilon_2}(f)$ .*

*Proof.* Suppose that this does not hold, i.e.,  $\mathcal{G}_{\epsilon_1}(f) \not\subseteq \mathcal{G}_{\epsilon_2}(f)$ . The element that is in  $\mathcal{G}_{\epsilon_1}(f)$  and not in  $\mathcal{G}_{\epsilon_2}(f)$  is not a vertex because both task graphs have the same set of tasks as vertices. Therefore, it must be an arc, i.e.,  $(t, t')$  is an arc in  $\mathcal{G}_{\epsilon_1}(f)$  and not in  $\mathcal{G}_{\epsilon_2}(f)$ . By Def. 4,  $close_{\epsilon_1}(t, t')$  and this arc is not redundant. This means that there is no other path from  $t$  to  $t'$  via the  $close_{\epsilon_1}$  relation. By definition, we also have that  $close_{\epsilon_2}(t, t')$  because  $\epsilon_1 \leq \epsilon_2$ . However, because  $(t, t')$  is not an arc of  $\mathcal{G}_{\epsilon_2}(f)$ , we must conclude that it is redundant. Because the gap between  $t$  and  $t'$  is smaller than or equal to  $\epsilon_1$ , however, this would imply that  $(t, t')$  is also redundant in  $\mathcal{G}_{\epsilon_1}(f)$ , which is a contradiction. Therefore, the statement holds.  $\square$

**Theorem 2.** *Let  $G = (T, \rightarrow, d)$  be a task graph, let  $f$  be an execution of  $G$ , and let  $\mathcal{G}_{\infty}(f) = (T, \rightarrow_{\infty}, d)$ . If  $t \rightarrow^+ t'$ , then  $t \rightarrow_{\infty}^+ t'$ .*

*Proof.* Assume that  $t \rightarrow^+ t'$ . By Def. 3, we have that  $t'$  starts the moment that  $t$  ends or later than that. Thus,  $close_{\infty}(t, t')$  and therefore  $t \rightarrow_{\infty}^+ t'$ .  $\square$

**Proposition 1.** *Algorithm 1 adds the tasks (in line 6) according to a topological order of the resulting arc relation.*

*Proof.* A sequence  $v_1 v_2 \dots v_n$  of tasks is ordered topologically according to the arc relation  $\rightarrow$  if it holds that  $v_p \rightarrow_n v_q$  implies that  $p < q$ . Now consider the

---

<sup>\*</sup>Embedded Systems Innovation by TNO, Eindhoven, The Netherlands

<sup>†</sup>[martijn.hendriks@tno.nl](mailto:martijn.hendriks@tno.nl)

<sup>‡</sup>Eindhoven University of Technology, Eindhoven, The Netherlands

<sup>§</sup>Océ Technologies B.V., Venlo, The Netherlands

sequence of tasks that are added to  $T$  in line 6. Clearly, every task is added exactly once, because a task has only a single start event. Now consider the addition of task  $v_i$  to  $T$ . Line 7 of the algorithm adds  $(v_k, v_i)$  to the arc relation, for every  $v_k \in F_{i-1}$  that is  $\epsilon$ -close to  $v_i$ . We must show that  $v_k$  has already been added earlier to  $T$ . Line 17 has added  $v_k$  to some  $F_j$  with  $j < i$  when its end event was processed. By definition, the start event of  $v_k$  has also been processed in an earlier iteration and therefore  $k < i$ .  $\square$

**Lemma 1.** *An invariant that holds in line 20 (at the end of each loop iteration) of Alg. 1 is that for all  $t \in D_i$  there exists a task  $t' \in F_i$  such that  $t \rightarrow_i^+ t'$ .*

*Proof.* Clearly, this holds for the first loop iteration that necessarily processes a start event. Now suppose that the property holds for all iterations up to the  $i$ -th loop iteration. If the  $i$ th event is a start event, then  $D_i = D_{i-1}$ ,  $F_i = F_{i-1}$  and  $\rightarrow_i \supseteq \rightarrow_{i-1}$ . Therefore, the property also holds for  $i$ . If the  $i$ th event is an end event  $(v_i, t_i, \mathbf{e})$ , then a set of vertices is moved from  $F_{i-1}$  to  $D_i$ . Every vertex that is moved, is a predecessor of  $v_i$ , which itself is added to  $F_i$ . Therefore, the property also holds for  $i$ .  $\square$

**Lemma 2.** *An invariant that holds in line 20 (at the end of each loop iteration) of Alg. 1 is that  $t \not\rightarrow_i^+ t'$  for all  $t, t' \in F_i$ .*

*Proof.* Clearly, this holds for the first loop iteration that necessarily processes a start event. Suppose that it holds for the loop iterations up to  $i$ . If a start event is processed in iteration  $i$ , then the arc relation is extended with arcs from tasks in  $F_{i-1}$  to the new task. Furthermore,  $F_i = F_{i-1}$ . Therefore, the lemma still holds. If an end event is processed in iteration  $i$ , then the arc relation is not changed:  $\rightarrow_i = \rightarrow_{i-1}$ . The task of the end event,  $v_i$ , however, is added to  $F_{i-1}$  in order to create  $F_i$ . However, all tasks in  $F_{i-1}$  that are predecessors of  $v_i$  are removed at the same time. Therefore, the only remaining way to have a relation between tasks in  $F_i$  is that  $v_i$  is predecessor of some other task in  $F_{i-1}$ , say  $w$ . Now suppose that that is the case. This means that the arc  $v_i \rightarrow w$  has been added earlier by line 7 in the algorithm in an iteration  $j < i$ . This iteration has thus processed the start event of task  $w$ . This implies that  $v_i \in F_{j-1}$ , which means that the end event of  $v_i$  had to be processed before iteration  $j$ . This contradicts the fact that iteration  $i$  processes the end event of  $v_i$ .  $\square$

**Theorem 3.** *Let  $f$  be an execution, and let  $\tau$  be a trace of  $f$ . Algorithm 1 computes  $\mathcal{G}_\epsilon(f)$  from  $\tau$ .*

*Proof.* We must prove that  $(T_n \rightarrow_n, d_n) = \mathcal{G}_\epsilon(f)$ , which equals  $((T, close_\epsilon)^-, d)$  by Def. 4. Clearly,  $d_n = d$ , and  $T_n = T$  because the trace contains exactly one start and one end event for each task. Line 6 updates the set of tasks, and lines 8 and 15 take care of the duration execution function.

Next, we must prove that (i)  $\rightarrow_n$  encodes the same reachability relation as  $close_\epsilon$  and (ii) that  $\rightarrow_n$  contains no redundant edges. From (ii) follows that  $(T_n, \rightarrow_n)$  is transitively reduced. Together with (i), which states that  $\rightarrow_n$  and  $close_\epsilon$  encode the same reachability relation, we can then conclude that

$(T_n \rightarrow_n) = (T, \text{close}_\epsilon)^-$ . In the remainder of this proof, we omit the subscript  $n$  in the arc relation  $\rightarrow_n$ .

(i  $\Rightarrow$ ) Suppose that  $v \rightarrow v'$ . We prove  $\text{close}_\epsilon(v, v')$ . The arc  $v \rightarrow v'$  is only added if task  $v$  is in  $F_{i-1}$  and the start event of  $v'$  is processed in iteration  $i$  (see line 7). Task  $v$  being element of  $F_{i-1}$  implies that the end event of task  $v$  has been processed in iteration  $i-1$  or earlier. Because the events are ordered by their timestamp, we have that  $\text{end}(v) \leq \text{start}(v')$ . Furthermore, in line 7 we also see that the distance between the end event of  $v$  ( $t_k$ ) and the start event of  $v'$  ( $t_i$ ) is at most  $\epsilon$ . Therefore,  $\text{close}_\epsilon(v, v')$ .

(i  $\Leftarrow$ ) Suppose that  $\text{close}_\epsilon(v, v')$ . We prove  $v \rightarrow^+ v'$ . By definition,  $\text{end}(v) \leq \text{start}(v') \wedge \text{start}(v') - \text{end}(v) \leq \epsilon$ . In the execution trace, the end event of  $v$  is processed before the start event of  $v'$ . Consider the iteration  $i$  of the algorithm which processes the start event of  $v'$ . There are two cases: either  $v \in F_{i-1}$  or  $v \in D_{i-1}$ . In the first case, line (7) adds the arc  $v \rightarrow v'$ . In the second case, we know from Lem. 1 that some  $v'' \in F_{i-1}$  exists such that  $v \rightarrow^+ v''$ . Furthermore, line (7) adds the arc  $v'' \rightarrow v'$ , and therefore  $v \rightarrow^+ v'$ .

(ii) Consider an arc  $v \rightarrow v'$  and suppose that this arc is redundant, i.e., there is a  $v'' \neq v$  and  $v'' \neq v'$  such that  $v \rightarrow^+ v''$  and  $v'' \rightarrow v'$ . Let  $i$  be the iteration that adds the arc  $v \rightarrow v'$ . This happens when processing the start event of  $v'$  and implies that  $v \in F_{i-1}$ . This same iteration must then also add the arc  $v'' \rightarrow v'$ , and therefore  $v'' \in F_{i-1}$ . However, Lem. 2 tells us that  $v$  and  $v''$  must be unrelated, i.e.,  $v \not\rightarrow^+ v''$ . This contradicts our assumption. Therefore, such a  $v''$  does not exist, and hence  $v \rightarrow v'$  is not redundant.  $\square$

**Theorem 4.** Consider a trace  $\tau$  of an execution  $f$ . Algorithm 1 uses  $\mathcal{O}(\text{length}(\tau) \cdot w(\mathcal{G}_\epsilon(f)))$  set operations.

*Proof.* There are  $2 \cdot |T|$  iterations, and each iteration applies a number of set operations that is linear in the size of  $F_i$ . Lemma 2 shows that  $F_i$  is an anti-chain, hence  $|F_i| \leq w(\mathcal{G}_\epsilon(f))$ . Since  $\text{length}(\tau) = 2 \cdot |T|$ , the total number of set operations is  $\mathcal{O}(\text{length}(\tau) \cdot w(\mathcal{G}_\epsilon(f)))$ .  $\square$

**Lemma 3.**  $\text{start}^-(t) \leq \text{start}^+(t)$  for all  $t \in T$ .

*Proof.* By induction on the reversed topologically ordered task sequence, i.e., from  $t_n$  to  $t_1$ . By line 4 in Alg. 2 we have that  $\text{start}^-(t_n) = \text{start}^+(t_n)$ . Now suppose that it holds up to task  $j$ :  $\text{start}^-(t_k) \leq \text{start}^+(t_k)$  for all  $j \leq k \leq n$ , and consider  $t_i$ . The maximum start time of  $t_i$  is determined by its successors. Consider one of the successors, say  $t_j$ , that determines it. Then we have that  $\text{start}^+(t_i) = \text{start}^+(t_j) - d_i$  and  $\text{start}^-(t_j) \geq \text{start}^-(t_i) + d_i$ . The latter equation gives  $\text{start}^-(t_i) \leq \text{start}^-(t_j) - d_i$ . Application of the induction hypothesis gives that  $\text{start}^-(t_i) \leq \text{start}^+(t_j) - d_i$ . The first equation above then gives that  $\text{start}^-(t_i) \leq \text{start}^+(t_j) - d_i = \text{start}^+(t_i)$ .  $\square$

**Lemma 4.** If a task  $t_j$  is critical, then at least one successor  $t_k$  (if it exists) is critical, and at least one predecessor  $t_i$  (if it exists) is critical. Furthermore, these all are consecutive, which is to say that  $\text{start}^-(t_i) + d(t_i) = \text{start}^-(t_j)$  and  $\text{start}^-(t_j) + d(t_j) = \text{start}^-(t_k)$ .

*Proof.* By induction on the reversed topologically ordered task sequence, i.e., from  $t_n$  to  $t_1$ . First, consider  $t_n$  (which is critical by definition). It has no successors. Consider a predecessor, if it exists, that determines  $start^-(t_n)$ , say  $t_i$ . It holds that they are consecutive:  $start^-(t_n) = start^-(t_i) + d(t_i)$  by definition (see Alg. 2). Furthermore, we have that

$$\begin{aligned} start^+(t_i) &\leq start^+(t_n) - d(t_i) \\ &= start^-(t_n) - d(t_i) \\ &= start^-(t_i). \end{aligned}$$

Lemma 3 gives that  $start^+(t_i)$  cannot be lower than  $start^-(t_i)$  and therefore  $start^+(t_i) = start^-(t_i)$ , which is to say that  $t_i$  is a critical predecessor.

Now assume that the lemma holds for  $t_{i+1} \cdots t_n$ . We prove the lemma for  $t_i$ . The proof that  $t_i$  has a critical and consecutive predecessor is equivalent to the proof above. Now we show that it has a critical and consecutive successor. Therefore, let  $t_j$  be the successor that determines the latest start of  $t_i$ , i.e.,  $start^+(t_i) = start^+(t_j) - d(t_i)$ . Because we assumed that  $t_i$  is critical, we have that minimum and maximum start times are equal and therefore  $start^-(t_i) = start^+(t_j) - d(t_i)$ . Then we have that

$$\begin{aligned} start^-(t_j) &\geq start^-(t_i) + d(t_i) \\ &= start^+(t_j). \end{aligned}$$

Lemma 3 gives that  $start^-(t_j)$  cannot be larger than  $start^+(t_j)$  and therefore  $start^-(t_j) = start^+(t_j)$ , which makes  $t_j$  critical. Substitution gives that  $start^-(t_i) + d(t_i) = start^-(t_j)$ , which makes them consecutive.  $\square$

**Lemma 5.** *The  $start^-$  functions as computed by Alg. 2 for  $G$  and for  $\mathcal{G}_0(exec(G))$  are equal.*

*Proof.* The function computed for  $G$  is denoted by  $start_1^-$  and the one computed for  $\mathcal{G}_0(exec(G))$  is denoted by  $start_2^-$ . First, note that the  $start$  function that is part of  $exec(G)$  as defined in Def. 3 is equal to  $start_1^-$ .

We use induction on the topologically ordered task sequence, i.e., from  $t_1$  to  $t_n$  in  $G$ . Because a source task has no predecessors, we have that  $start^-(t_1) = 0$ . Definition 3 also gives  $start(t_1) = 0$ . Since tasks have a strictly positive execution time, there is no  $t'$  such that  $close_0(t', t_1)$  and hence  $t_1$  is also a source in  $\mathcal{G}_0(exec(G))$ . Therefore,  $start_2^-(t_1) = 0$ .

Now suppose that  $start_1^-(t_i) = start_2^-(t_i)$  for all  $i < m$  and consider the valuation of  $t_m$ . Let  $t_k$  be the predecessor of  $t_m$  that defines its start time in  $G$ :  $start_1^-(t_m) = start_1^-(t_k) + d(t_k)$ . Thus,  $start(t_m) = end(t_k)$  and hence  $close_0(t_k, t_m)$ , which implies that  $t_k$  also is a predecessor of  $t_m$  in  $\mathcal{G}_0(exec(G))$ . By the induction hypothesis we have that  $start_2^-(t_k) = start_1^-(t_k)$ . Therefore,  $start_2^-(t_m) \geq start_2^-(t_k) + d(t_k)$  (see Alg. 2). Substitution gives  $start_2^-(t_m) \geq start_1^-(t_k) + d(t_k)$ .

Now consider a predecessor of  $t_m$  in  $\mathcal{G}_0(exec(G))$ , say  $t_j$  (which not necessarily is a predecessor of  $t_m$  in  $G$ ). We thus have that  $close_0(t_j, t_m)$  and therefore

$start_1^-(t_j) + d(t_j) \leq start_1^-(t_m)$ . Substitution then gives  $start_1^-(t_j) + d(t_j) \leq start_1^-(t_k) + d(t_k) \leq start_2^-(t_m)$ . An arbitrary predecessor  $t_j$  thus does not overrule  $t_k$  for the value of  $start_2^-(t_m)$  and therefore  $start_2^-(t_m) = start_1^-(t_k) + d(t_k) = start_1^-(t_m)$ .  $\square$

**Theorem 5.** *A task that is marked critical (i.e., it has zero float) by Alg. 2 when run on task graph  $G$ , is also marked critical by Alg. 2 when run on  $\mathcal{G}_0(exec(G))$ .*

*Proof.* The function computed for  $G$  is denoted by  $start_1^-$  and the one computed for  $\mathcal{G}_0(exec(G))$  is denoted by  $start_2^-$ .

Consider the situation in which a task  $t_i$  is marked critical in  $G$  but not in  $\mathcal{G}_0(exec(G))$ . Using Lem. 4 we can construct a path  $\pi$  between  $t_i$  and the unique sink  $t_n$  in  $G$  consisting of consecutive critical tasks. Because the tasks are consecutive, they are related by the  $close_0$  relation. Furthermore, because tasks have a strictly positive execution time we can conclude that  $\pi$  is also a path in  $\mathcal{G}_0(exec(G))$ . The tasks on this path are clearly also consecutive (e.g., because  $start_1^- = start_2^-$  by Lem. 5).

Now pick the task  $t_j$  in this path that is not marked critical in  $\mathcal{G}_0(exec(G))$  and which is closest to  $t_n$  (this may be task  $t_i$ ). This is not  $t_n$  itself, which is critical by definition. Therefore, the non-critical  $t_j$  has a critical successor  $t_k$  (on the path  $\pi$ ). Thus,

$$\begin{aligned} start_2^+(t_j) &\leq start_2^+(t_k) - d(t_j) \\ &= start_2^-(t_k) - d(t_j). \end{aligned}$$

Substitution of  $start_2^-(t_k) = start_2^-(t_j) + d(t_j)$  (because  $t_j$  and  $t_k$  are consecutive) in the equation above gives that  $start_2^+(t_j) \leq start_2^-(t_j)$ . Lemma 3 gives us that  $start_2^+(t_j)$  cannot be lower than  $start_2^-(t_j)$  and therefore  $start_2^+(t_j) = start_2^-(t_j)$  which makes  $t_j$  critical in  $\mathcal{G}_0(exec(G))$ . This contradiction completes the proof.  $\square$

**Theorem 6.** *Let  $G$  be a task graph. If for any two paths  $\pi$  and  $\pi'$  in  $G$   $duration(\pi) \neq duration(\pi')$ , then the sets of critical tasks of  $G$  and  $\mathcal{G}_0(exec(G))$  are equal.*

*Proof.* We show that a task that is critical in  $\mathcal{G}_0(exec(G))$  is also critical in  $G$ . Together with Th. 5 this then proves the theorem.

The function computed for  $G$  is denoted by  $start_1^-$  and the one computed for  $\mathcal{G}_0(exec(G))$  is denoted by  $start_2^-$ . We use induction on the reversed topological order of tasks in  $\mathcal{G}_0(exec(G))$ , i.e., from  $t_n$  to  $t_1$ . The algorithm marks  $t_n$  as critical in  $\mathcal{G}_0(exec(G))$ . We infer that  $t_n$  is also the last task in a topological order of  $G$ . Therefore,  $t_n$  is also marked critical in  $G$ .

Now suppose that the theorem holds for all tasks  $t_{i+1} \cdots t_n$ . We show that it also holds for  $t_i$ . Therefore, assume that  $t_i$  is marked critical in  $\mathcal{G}_0(exec(G))$ . Lemma 4 gives that it has a successor that is critical and consecutive, say  $t_j$ . This task  $t_j$  is also marked critical in  $G$  by the induction hypothesis.

First, we prove that  $t_j$  is also a successor of  $t_i$  in  $G$ . By Lem. 5 and the observation that the execution as defined in Def. 3 gives the same start times as

Alg. 2, we have that  $start(t_i) + d(t_i) = start(t_j)$ . Suppose that  $t_i \not\rightarrow t_j$  in  $G$ . In that case, there must be another task  $t_k$  such that  $start(t_k) + d(t_k) = start(t_j)$ . The assumption that every path has a unique duration, however, implies that  $start^-(t) + d(t)$  is unique for every task  $t$ . This contradiction proves that  $t_i \rightarrow t_j$  in  $G$ .

Next, we prove that  $start_1^+(t_i) = start_1^-(t_i)$ . By definition  $start_1^+(t_i) \leq start_1^+(t_j) - d(t_i)$  and therefore,  $start_1^+(t_i) \leq start_1^-(t_j) - d(t_i)$  (because  $t_j$  is critical). By Lem. 3 we have that:

$$start_1^-(t_i) \leq start_1^+(t_i) \leq start_1^-(t_j) - d(t_i) \quad (1)$$

We can write  $start_1^-(t_i)$  as follows:

$$\begin{aligned} start_1^-(t_i) &= start_2^-(t_i) && \text{(By Lem. 5)} \\ &= start_2^-(t_j) - d(t_i) && (t_i \text{ and } t_j \text{ consecutive}) \\ &= start_1^-(t_j) - d(t_i) && \text{(By Lem. 5)} \end{aligned}$$

Substitution in 1 gives that  $start_1^+(t_i) = start_1^-(t_j) - d(t_i)$ . Because  $t_i$  and  $t_j$  are consecutive in  $\mathcal{G}_0(exec(G))$ , Lem. 5 gives that  $start_1^-(t_j) = start_1^-(t_i) + d(t_i)$ . Substitution then yields  $start_1^+(t_i) = start_1^-(t_i)$  and hence  $t_i$  is critical.  $\square$

**Theorem 7.** *Let  $G = (T, \rightarrow, d)$  be a task graph, let  $exec(G) = f$ , and let  $U \subseteq T$ . If the source tasks and the unique sink task in  $G$  are not elements of  $U$ , then a task  $t \in T \setminus U$  that is marked critical by Alg. 2 when run on  $G$ , is also marked critical by Alg. 3 when run on  $\mathcal{G}_{m(G,U)}(f \ominus U)$  and the gap function  $\delta^f$ .*

*Proof.* The function computed for  $G$  is denoted by  $start_1^-$  and the one computed for  $\mathcal{G}_{m(G,U)}(f \ominus U)$  is denoted by  $start_2^-$ . With our assumption that the source tasks and the unique sink task are not part of  $U$ , we can prove with a similar argument as in the proof of Lem. 5 that  $start_1^-(t) = start_2^-(t)$  for  $t \in T \setminus U$ .

Consider the situation in which a task  $t_i$  is marked critical in  $G$  but not in  $\mathcal{G}_{m(G,U)}(f \ominus U)$ . According to Lem. 4 we can construct a path  $\pi$  between  $t_i$  and the unique sink  $t_n$  in  $G$  consisting of consecutive critical tasks. By our choice of  $\epsilon = m(G, U)$  and the fact that  $start_1^-(t) = start_2^-(t)$  for  $t \in T \setminus U$  we can conclude that the sub sequence of  $\pi$  without tasks in  $U$  (denoted by  $\pi \ominus U$ ), say  $t_1 \cdots t_n$ , are consecutive modulo the gaps, which is to say that  $start_2^-(t_{i+1}) = start_2^-(t_i) + d(t_i) + \delta(t_i, t_{i+1})$  for  $1 \leq i < n$ .

Now we pick a task  $t_j$  in  $\pi \ominus U = t_1 \cdots t_n$  that is not marked critical in  $\mathcal{G}_0(exec(G))$  and for which holds that the next task  $t_k$  is marked as critical. This can be done, because the unique sink task  $t_n$  is marked as critical by Alg. 3 and is not part of  $U$ . Thus,

$$\begin{aligned} start_2^+(t_j) &\leq start_2^+(t_k) - (d(t_j) + \delta(t_j, t_k)) \\ &= start_2^-(t_k) - (d(t_j) + \delta(t_j, t_k)). \end{aligned}$$

Substitution of  $start_2^-(t_k) = start_2^-(t_j) + d(t_j) + \delta(t_j, t_k)$  (because  $t_j$  and  $t_k$  are consecutive modulo the gaps) in the equation above gives us the following:

$start_2^+(t_j) \leq start_2^-(t_j)$ . Similar to the proof of Lem. 3, we can prove that  $start_2^+(t_j) \geq start_2^-(t_j)$  and therefore  $start_2^+(t_j) = start_2^-(t_j)$  which makes  $t_j$  critical. This contradiction completes the proof.  $\square$

**Proposition 2** (Triangle inequality). *The graph edit distance satisfies the triangle inequality:  $d_{\text{ged}}(G_1, G_2) + d_{\text{ged}}(G_2, G_3) \geq d_{\text{ged}}(G_1, G_3)$ .*

*Proof.* We first show that  $A\Delta C \subseteq (A\Delta B) \cup (B\Delta C)$  for all sets  $A, B, C$ . Consider some  $x \in A\Delta C$ . It holds that  $(x \in A \vee x \in C) \wedge x \notin A \cap C$ . We know that  $x \in B \vee x \notin B$ . We consider the four combinations of the two disjunctions in our set of premises. (i)  $x \in A \wedge x \in B$ . We know that  $x \notin A \cap C$ . Now suppose that  $x \in B \cap C$ . Then  $x \in B \wedge x \in C$ . The combination with  $x \in A$  then gives that  $x \in A \cap C$ , which gives a contradiction. Therefore,  $x \notin B \cap C$ . The combination with  $x \in B$  gives  $x \in B\Delta C$ . (ii)  $x \in A \wedge x \notin B$ . Thus,  $x \in A\Delta B$ . (iii)  $x \in C \wedge x \in B$ . We have that  $x \notin A \cap C$  and therefore  $x \notin A$ . Thus,  $x \in A\Delta B$ . (iv)  $x \in C \wedge x \notin B$ . Thus,  $x \in B\Delta C$ . Concluding,  $A\Delta C \subseteq (A\Delta B) \cup (B\Delta C)$ . This gives  $|A\Delta C| \leq |(A\Delta B) \cup (B\Delta C)|$ , which implies  $|A\Delta C| \leq |A\Delta B| + |B\Delta C|$ . Application of this inequality to the graph edit distance proves the lemma.  $\square$

**Theorem 8.** *Execution distance is a pseudo-metric.*

*Proof.* First, the distance is positive or zero by Def. 6. Second,  $d(f, f) = 0$  clearly holds by Def. 6, since an execution has a unique task graph, and symmetric differences between identical sets are empty. Third,  $d(f_1, f_2) = d(f_2, f_1)$  because symmetric differences are symmetric:  $A\Delta B = B\Delta A$ . Fourth, the triangle inequality holds by Prop. 2.  $\square$

**Theorem 9.** *The time complexity to compute the execution distance from two traces is  $\mathcal{O}(|T| \cdot w)$ , where  $T$  is the largest of the traces' task sets and  $w$  is the maximum of the widths of the two task graphs.*

*Proof.* We can construct the two task graphs from the traces in time  $\mathcal{O}(|T| \cdot w)$  according to Th. 4. The number of vertices of these graphs is equal to the number of tasks in the corresponding traces. However, because we are dealing with transitively reduced directed acyclic graphs, we know that the number of arcs is bound by  $|T| \cdot w$ . This can be understood from the observation that in a transitive reduction, every vertex has at most  $w$  direct successors, which follows from Dilworth's theorem [1]. Therefore, computation of the execution distance using Def. 6 takes at most  $\mathcal{O}(|T| \cdot w)$  set inclusion checks.  $\square$

## References

- [1] R.P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.