An $O(|V||E|)$ Algorithm for Scheduling with AND/OR Precedence Constraints

Yefim Dinitz* Paz Carmi* Shahar Golan Omri Liba*
Guy Rozenwald

April 19, 2011

Abstract

The paper studies scheduling with AND and OR precedence constraints. The events are vertices of a non-negatively weighted digraph $G = (V, E, d)$; the precedence relation is defined by its edges, whose weights represent delays. An event of type AND may be scheduled only after *all* its preceding events, with the corresponding delays after them. An event of type OR may be scheduled only after *at least one of* its preceding events, with the corresponding delay after it. The early schedule is in question. A few algorithms solving various cases of this problem are known. We present an algorithm for the general case, where the precedence graph may have an arbitrary structure and zero weight cycles are allowed. It runs in time $O(|V||E|)$. In addition, we present a necessary and sufficient condition for infeasibility of a problem instance, in structural terms of graph G.

*Dept. of Computer Science, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105, Israel. E-mail:{dinitz,carmip,liba}@cs.bgu.ac.il .

1 Introduction

The classic PERT scheduling problem is defined by a directed graph $G = (V, E, d)$, $d : E \to \mathbf{R}^+$. (In what follows, we say "graph" meaning a directed graph with non-negatively weighted edges.) Its vertices represent events, while its edges represent the precedence relation between them: if there is an edge from vertex u to vertex v (we say: *u precedes v*), the event v may happen not earlier than by delay of $d(u, v)$ after u happens. The problem goal is finding the earliest non-negative time-schedule, which obeys the precedence relation (i.e., finding *early times* for events). It is accepted that if for some event no feasible time exists, then the time *infinity* is assigned to it. The classic motivation for PERT is finding the early time-schedule for a large project, where vertices are intermediate events (e.g., the starting or finishing time of a project component).

The linear time algorithm for the basic case of *positive weights*, using topological sorting of vertices, is one of the classic optimization algorithms; we will refer to it by name PERT. (Here and on, see e.g., [3] for a description of classic algorithms.) We say that a *problem instance is feasible* if there exists a finite solution, for it. For the basic case, it is known that this is the case if and only if the precedence graph is acyclic. For a graph with cycles, all vertices lying on a cycle, together with vertices reachable from them, should be assigned the early time infinity. The same holds for the case, when edge weights may be zero, but each cycle has a positive weight. For the general non-negative weight case, when cycles consisting of zero weight edges only (henceforth called "zero cycles") are allowed, the classic algorithm should be extended (see Section 4.3.1).

The common sense of a zero delay precedence is "not earlier than". Such a precedence often happens in human interaction. In particular, it is usual announcing events in groups. In physical macro-world, a precedence with zero delay is not so natural. However, it may be quite appropriate to a micro-world description, since quantum events naturally happen in groups. In particular, in future, quantum computation may become a source of precedence relations with zero delays.

The AND/OR setting of PERT with non-negative delays is defined in [5, 4] as follows. The precedence graph, henceforth called *AND/OR graph*, has vertices of two types. A vertex with precedence conditions as above is considered as an *AND vertex*. For an *OR vertex* v, its time should be not earlier than by $d(u, v)$ after *at least one* vertex u preceding v. In this paper, this problem is referred to as the *AND/OR problem*, for short. The graph

may be general: containing cycles and not necessarily bipartite.

A motivation given to the AND/OR problem in [5, 4] is the following New-Product-New-Technology problem. Certain new technologies, which supply certain new products to the market and which need certain new products, are considered. An AND vertex corresponding to a new technology represents the event of its launching. At that time, each new product required by it should be already present at the market for a certain time period. A new technology supplies each product produced by it to the market after a certain delay after its launching. An OR vertex corresponding to a new product represents the event of its first appearance at the market. The early schedule of this technology-product system is in question. (The assumption relaxing the motivating problem is that the *amount* of any new product at the market is not taken into account.) For example, see Figure **??**.

A priory lower time bounds for new technologies are modeled as follows. An auxiliary AND vertex *origin* is introduced; it has no predecessors, and thus its early time is 0. For each AND vertex v with an a priory bound t, an edge $(origin, v)$ with delay t is added. Suppose now that some new products can be obtained without a help of any new technology (e.g., by exporting it), from a certain time moment and on. This is modeled by using edges from *origin* to the OR vertices corresponding to the products as above. Note that in New-Product-New-Technology problem, the precedence graph is almost bipartite, where only vertex *origin* is an exception.

More motivation and applications can be found in [8, 2, 9]. Paper [9] presents an extensive study of the AND/OR problem.

By definition, the pure case of AND vertices only is the classic PERT problem. It is interesting that the case of OR vertices only, except for the source AND vertex s without incoming edges, is essentially equivalent to another corner-stone optimization problem: finding the shortest path lengths from s to other vertices. The basic variant of the classic Dijkstra algorithm solves it in time quadratic in $|V|$, while using appropriate data structures provides time bounds almost linear in the size of G (see a review in, e.g., [3]).

The algorithms PERT and Dijkstra, studied in the basic courses on algorithms, are of somewhat similar and of somewhat different nature. Surprisingly, for the hybrid AND/OR problem as above, there exists an algorithm, which is combined of these two classic algorithms and runs in their summary time. It works if there is no cycle consisting of edges with zero delays only,

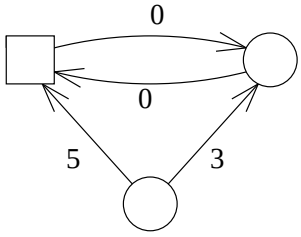

Figure 1: Circles are AND vertices, while squares are OR vertices. No one out of the two upper vertices cannot be assigned the time label 3 separately, only both of them together.

in the precedence graph. (Henceforth, such a cycle is called *zero cycle*.)

The above-mentioned hybrid algorithm was suggested independently in [8] and in [5, 4]. The problem considered by Knuth in [8] is presented in another language, and is equivalent to a particular case of the AND/OR problem. A reduction of the general AND/OR problem to the problem of [8] is also possible, but it does not preserve neither the correctness proof, given in [8], nor the algorithm running time bounds. The correctness proof of the algorithm [5, 4], though given at its presentations, was never published. It is provided in this paper.

The general (non-negative weight) case of the AND/OR problem, which allows zero cycles, is more involved. From the algorithmic point of view, the nature of the precedence with zero cycles does not allow extending the self-supporting set of vertices with optimal labels (S) vertex by vertex, as in algorithms Dijkstra, PERT, and the above-mentioned hybrid algorithm for the restricted AND/OR problem. It may happen that some vertices with the same early time support each other, and hence could be added to S only together. For an example, see Figure 1. After initializing S by $\{s\}$, with the early time label 0, and scanning s, OR vertex a may be labeled by 5 and AND vertex b by 3. The only possible single vertex extension of S is by a with the final label 5, and after that also b gets the final label 5. However, adding both vertices a and b to S with time 3 is also feasible, which forms the optimal solution.

In particular, the above example shows that Bellman-Ford algorithm applied to an AND/OR problem might converge to a non-optimal solution. Indeed, after relaxations on edges (s, a), (s, b), and (a, b), we arrive at the non-optimal solution considered above, where no relaxation is possible. The specifics as above makes development of an effective algorithm managing

the general case a new challenge.

The question of existence of a polynomial time algorithm for the general non-negative weight AND/OR problem, when zero cycles are allowed, was open for about a decade. Adelson-Velsky and Levner suggested an algorithm solving it in time $O(|E|^2) = O(|V|^4)$ [2]. Moehring et al. suggested an algorithm running in time $O(|V||E|)$ for a special case of the AND/OR problem [9]. In addition, a reduction from the general case to that case is provided in [9]. However, it increases the number of vertices from $|V|$ to $\Omega(|V| + |E|)$. As a result, the running time bound increases to $O(|E|^2) = O(|V|^4)$, in the terms of the original AND/OR precedence graph.

Two algorithms solving the AND/OR problem in time $O(|V||E|) = O(|V|^3)$ were announced in 2001: our algorithm, first presented in [6], and that of Adelson-Velsky et al. [1], which is more complicated than our one. The correctness proof was a real problem for both of them. The proof provided in [1] has essential gaps. Also for our algorithm, the correctness proof remained ugly for years. The algorithm presentation and the proof presented in this paper, which seem to us natural, were found only in 2011.

In addition, this paper presents a necessary and sufficient condition for infeasibility of an AND/OR problem instance, in structural terms of graph G (see Section 5).

2 Definitions and the Zero-Cycle-Free Case

An AND/OR problem instance is a directed weighted *AND/OR graph* $G = (V, E, d)$, where $d : E \to \mathbf{R}^+$ is the delay function, so that the vertex set V is divided into two disjoint subsets V^{AND} and V^{OR}. A non-negative (time) function T on E (a *solution*) is *feasible* if it satisfies the following constraints:

$$T(v) \geq \max_{(u,v)\in E} \{0, T(u) + d(u, v)\}, \text{ for all } v \in V^{AND}, \tag{1}$$

$$T(v) \geq \min_{(u,v)\in E} \{\infty, T(u) + d(u, v)\}, \text{ for all } v \in V^{OR}. \tag{2}$$

It is easy to check that the minimum of a set of feasible solutions is a feasible solution as well. Let us define function T^* by assigning to each vertex $v \in V$ the infimum of $T(v)$ over the set of all feasible solutions T. Note that this set is non-empty, since $T \equiv \infty$ is feasible.

Claim 2.1 (i) *The solution T^* is feasible.*
(ii) *The solution T^* satisfies all conditions* (1) *and* (2) *as equalities.*

Proof sketch: Item (i) is valid since the minimum of a set of feasible solutions is feasible. If item (ii) is wrong at a vertex v, then we can decrease $T^*(v)$ by a small amount without violating any constraint,—a contradiction to T^* being the infimum. □

The solution T^* will be referred to as *optimal*, the value $T^*(v)$ as the *early time* of v, and constraints (1) and (2) w.r.t. T^* as *equalities*. The problem goal is, given a graph G, to find the optimal solution T^*, that is the vertex early times.

An edge (u, v) with $T^*(v) = T^*(u) + d(u, v)$ will be referred to as a *tight* edge. For an edge (u, v), we call u its *tail*, and v its *head*.

In the remainder of this section, we consider the case when G has *no zero cycle*. Let us first remind the classic algorithms for the pure AND or OR cases, mentioned in Introduction.

Algorithm AND(G) /* PERT */
compute vertex in-degrees $indeg(v)$, for all $v \in V$
$T(v) \leftarrow 0$, $T^*(v) \leftarrow \infty$, for all $v \in V$
while (there is a vertex $u \in V$ with $indeg(u) = 0$)
    $T^*(u) \leftarrow T(u)$
    for each edge $(u, v) \in E$
        $indeg(v) \leftarrow indeg(v) - 1$
        $T(v) \leftarrow \max\{T(v), T(u) + d(u, v)\}$

Algorithm OR(G,s) /* Dijkstra */
$T^*(v) \leftarrow T(v) \leftarrow \infty$, for all $v \in V$
$T(s) \leftarrow 0$
$S \leftarrow \emptyset$; $m \leftarrow 0$
while $(m \neq \infty)$
    set m to $\min_{v \in V \setminus S}\{T(v), \infty\}$
    if $m \neq \infty$
        choose $u : T(u) = m$
        $T^*(u) \leftarrow T(u)$; $S \leftarrow S \cup \{u\}$
        for each edge $(u, v) \in E$
            $T(v) \leftarrow \min\{T(v), T(u) + d(u, v)\}$

The algorithm of [8, 5, 4] solving the no-zero-cycle AND/OR problem is as follows:

Routine Scan(u)

for each edge $(u,v) \in E$
  $indeg(v) \leftarrow indeg(v) - 1$
  if $v \in V^{AND}$
    $T(v) \leftarrow \max\{T(v), T(u) + d(u,v)\}$
  else $T(v) \leftarrow \min\{T(v), T(u) + d(u,v)\}$

Algorithm AND/OR(G)
compute vertex in-degrees $indeg(v)$, for all $v \in V$
$T(v) \leftarrow 0$, for all $v \in V^{AND}$; $T(v) \leftarrow \infty$, for all $v \in V^{OR}$
$T^*(v) \leftarrow \infty$, for all $v \in V$
$S \leftarrow \{\emptyset\}$; $m \leftarrow 0$

while $(m \neq \infty)$
  /* Phase AND */
  /* comment: works for both vertex types AND and OR */
  while (there is a vertex $u \in V$ with $indeg(u) = 0$)
    $T^*(u) \leftarrow T(u)$, $S \leftarrow S \cup \{u\}$
    Scan(u)

  /* Phase OR */
    set m to $\min_{v \in V^{OR} \cap (V \setminus S)}\{T(v), \infty\}$
    if $m \neq \infty$
      choose $u : T(u) = m$
      $T^*(u) \leftarrow T(u)$, $S \leftarrow S \cup \{u\}$
      Scan(u)

Theorem 2.2 *For any AND/OR graph $G = (V, E, d)$ without zero cycles, algorithm AND/OR correctly computes the early times. It can be implemented in time $O(|E| + |V^{OR}| \log |V^{OR}|)$.*

Proof: Note that updating at routine Scan maintains the properties:

$$T(v) = \max_{(u,v) \in E, u \in S}\{0, T(u) + d(u,v)\}, \text{ for all } v \in V^{AND}, \text{ and} \qquad (3)$$

$$T(v) = \min_{(u,v) \in E, u \in S}\{\infty, T(u) + d(u,v)\} \geq 0, \text{ for all } v \in V^{OR}. \qquad (4)$$

We prove, by induction on the moments of adding vertices v to S, that $T^*(v)$ is assigned the correct value. If v is added to S during some Phase AND, then all its predecessors, if any, are already in S, with the correct T^* labels. Hence, by equalities (1) and (3), it is correct also at v.

Let us now consider the moment of assigning early time m to an OR vertex u. We first prove that $T^*(u) \leq m$. Let T' be equal T^* at S and at u, and be ∞ at $(V \setminus S) \setminus \{u\}$. Solution T' is feasible at $(V \setminus S) \setminus \{u\}$ trivially, at any AND or OR vertex $v \in S$ by the algorithm, and at u by equality (4). By feasibility of T', holds $T^*(u) \leq m$, as required.

We now prove that for any vertex v_0 in $V \setminus S$ (including $v_0 = u$), holds $T^*(v_0) \geq m$, which will suffice. Assume, to the contrary, that $T^*(v_0) < m \leq \infty$. If v_0 is an *OR vertex*, by equality (2), there exists a vertex v_1, such that $T^*(v_1) + d(v_1, v_0) = T^*(v_0)$. Since $T^*(v_0) < m$, edge (v_1, v_0) does not participate in computing m in the Phase OR; therefore, $v_1 \in V \setminus S$. By non-negativity of d, we have $T^*(v_1) < m$. If v_0 is an *AND vertex*, by the finishing condition of the while loop at the previous Phase AND, there exists its predecessor v_2 in $V \setminus S$. By inequality (1) and non-negativity of d, $T^*(v_1) \leq T^*(v_1) + d(v_1, v_0) \leq T^*(v_0) < m$.

That is, in both cases, v_1 is as v_0: belongs to $V \setminus S$ and has the early time less than m. We continue to build, as above, a sequence $v_0, v_1, v_2, \ldots$ in $V \setminus S$ with non-increasing values of $T^*(v_i)$. Since V is finite, we must arrive at a cycle. Evidently, T^* is a constant on this cycle. Hence, all its edges have weight zero,—a contradiction to the assumption of the theorem. (See Figure **??** for illustration.)

Let us turn to the running time analysis. We denote by E^{AND} and E^{OR} the sets of edges incoming AND and OR vertices, respectively. Initialization is done in a linear time. The algorithm operations at its Phases AND require time $O(|E^{AND}|)$, similarly to algorithm PERT. The operations at Phases OR are similar to those of algorithm Dijkstra, as well as the specification for a data structure: to answer Extract-Min query $|V^{OR}|$ times, while updating by Decrease-Key query $|E^{OR}|$ times. Choosing the Fibonacci heap implementation, we obtain the running time bound $O(|E^{AND}|) + O(|E^{OR}| + |V^{OR}| \cdot \log |V^{OR}|) = O(|E| + |V^{OR}| \cdot \log |V^{OR}|)$. □

Since T^* is at least m at $V \setminus S$, as proved, the following property holds:

Corollary 2.3 *For any AND/OR graph without zero cycles, Algorithm AND/OR assigns early times to OR vertices in a non-decreasing order.*

3 Algorithm AND/OR Unweighted

Let us pass to the general case, where zero cycles are allowed. In this section, we suggest a linear time algorithm for an auxiliary *unweighted AND/OR*

problem. That algorithm serves as a sub-routine in our main algorithm. The problem and the algorithm may also be of a certain interest by themselves.

Let us extend a bit the notion of an AND/OR problem. Suppose that, in addition to its regular instance, a priory time bounds $b(v)$ are given: lower ones to (some of) AND vertices v, and upper ones to (some of) OR vertices v (the lower bound 0 and the upper bound ∞ are equivalent to the absence of a bound). Note that the arising problem (G, b) is straightforwardly equivalent to the following AND/OR problem $(G, b)^{origin}$: We add to G a new AND vertex *origin* without in-coming edges (hence, its final label is 0); for any original vertex v with a given time bound, we add a new edge $(origin, v)$ with delay $b(v)$. Henceforth, we will identify (G, b) with $(G, b)^{origin}$, referring to it as to an AND/OR problem.

Consider an AND/OR graph G, whose all edges have zero delays; that is, the edges provide an *unweighted* precedence relation. Assume that in addition, a priory time bounds $b(v)$ are given to some vertices v. We call this setting an *unweighted AND/OR problem*.

In order to "vizualize" solving this problem and facilitate the algorithm correctness proof, let us consider the following auxiliary unweighted AND/OR problem instance $(G, b)^{ruler}$, equivalent to (G, b). Let B be the set of distinct values of the partial function $b : V \to \mathbf{R}^+$, together with the value ∞. For each $t \in B$, we add to G an auxiliary AND vertex z^t, with the a priory time bound t. Clearly, the early time of z^t is t. For every original vertex v with the given bound $b(v)$, we cancel its bound, replacing it by an additional edge $(z^{b(v)}, v)$ with a zero delay. Besides, for any OR vertex v, we add edge (z^{∞}, v) with a zero delay. The equivalence is easy.

Remark: This construction may be thought of as creating a sort of *ruler*, whose vertices have fixed early times; unweighted precedence edges from those vertices provide the time bounds as required.

In the following algorithm, we assume that set B is given *sorted* in the decreasing order. At any algorithm moment, vertices not given T^* label yet are referred to as *unlabeled*.

AND/OR Unweighted(G, b, B)

construct $(G, b)^{ruler}$
compute $indeg(v)$, for all $v \in V^{OR}$
for each vertex z^t, $t \in B$, in the decreasing order of t
  $T^*(z^t) \leftarrow t$; $Q \leftarrow \{z^t\}$
  while (Q is non-empty)

pop vertex u from Q
for each edge $(u, v) \in E$
if v is an unlabeled AND vertex
$T^*(v) \leftarrow t$; add v to Q
if v is an OR vertex /* always unlabeled */
$indeg(v) \leftarrow indeg(v) - 1$
if $indeg(v) = 0$
$T^*(v) \leftarrow t$; add v to Q

for each unlabeled vertex v
$T^*(v) \leftarrow 0$

Remark: Early time zero may be given partly at the last main for-each iteration and partly after it. After the main for-each loop, it is given, in particular: (i) to all AND vertices without in-coming edges and (ii) to all OR vertices lying on cycles consisting of OR vertices and zero edges only.

Proposition 3.1 *Algorithm AND/OR Unweighted correctly computes early times, in a time linear in the graph size.*

Proof: The proof is by induction on the iterations of the main for-each loop.

Basis: We prove for the first iteration (that using z^∞) by induction on vertex labeling in it. Giving T^* label ∞ to the AND vertices is obviously correct. Labeling by ∞ an OR vertex is also correct, since all edges in-coming it are from z^∞ or vertices given label ∞ previously.

Induction step: Suppose that T^* labels are given correctly at some first iterations. Consider the state at the beginning of the next iteration, which uses vertex z^t. By the algorithm, the following property (*) holds:

- There is no edge from a labeled vertex to an unlabeled AND vertex.
- There is at least one edge to any unlabeled OR vertex from another unlabeled vertex.

Let us prove that at the beginning of the iteration using vertex z^t, $T^*(v)$ is *at most* t for any unlabeled vertex v. Let us label by t all unlabeled vertices. By the sorting of B, the a priory time bounds of vertices z^x are satisfied. By property (*) and the assumption that all edges are zero ones, the current labeling satisfies inequalities (1) and (2). That is, we arrived

at a feasible solution. Hence, it is an upper bound to the early times, as required.

We now prove that the early time is *at least* t for any vertex v labeled at the current iteration, by induction on vertex labeling in it. This is straightforward by the algorithm if v is an AND vertex. This is correct if v is an OR vertex, since by the algorithm, *all* edges in-coming v are from vertices given previously T^* labels greater or equal than t. This is the end of the induction step.

Now we finish the correctness proof by showing that at the end of the main for-each loop, for any unlabeled vertex v, holds $T^*(v) = 0$. Let us label all unlabeled vertices by zero. Notice that after the last iteration of the while loop, property (*) holds. By property (*) and the assumption that all edges are zero ones, the current labeling satisfies inequalities (1) and (2). That is, we arrived at a feasible solution. It is optimal, since labels should be non-negative.

The linear running time bound, if B is given sorted, is evident. □

4 Algorithm for the General Case

Now we pass to the general AND/OR scheduling problem. Let us define a vertex *layer* L_t as the set of vertices v with the early time $T^*(v) = t$. In the following discussion, we consider the set of layers as their sequence sorted by t. In this sense, we will use notions of the next and previous layers, a layer prefix/suffix, meaning a prefix/suffix of the layer sequence, resp., etc. For a layer set $\mathcal{L}$, we denote $V(\mathcal{L}) = \cup_{L \in \mathcal{L}} L$. Algorithm AND/OR General presented below works in iterations. In its basic version, each iteration provides the early times exactly to the *entire next layer*. Note that also algorithm AND/OR Unweighted provides early times layer by layer, but in the opposite order: from later to earlier ones.

At each iteration, AND/OR General executes AND/OR Unweighted on an auxiliary graph on the set of unlabeled vertices. The construction of that graph guarantees that its first layer inherits the early time of the first unlabeled layer of G, while the early times in the following layers may only grow. As a result, the last, earliest layer discovered by AND/OR Unweighted in the auxiliary graph is the next layer of G, with the same early time. The extended version of AND/OR General uses, in addition, another way to provide T^* labels to vertices, similar to that used by algorithm PERT; that version has a lower running time bound.

In Section 4.1, we give an abstract algorithm. Its correctness is proved in Section 4.2. In Section 4.3, we give an extended version of the abstract algorithm and prove its correctness. An implementation and the running time analysis are provided in Section 4.4. Some additional fastening is suggested in Section 4.5.

4.1 Algorithm

Let E^{zero} be the set of zero edges in G, and $G^{zero} = (V, E^{zero})$. For a graph H, we denote by $H(U)$ the subgraph of H induced by its vertex subset U. Given a vertex subset S, let $b^*_S(v)$ be the function on $V \setminus S$ equal $\max_{u \in S,\, (u,v) \in E}\{0; T^*(u)+d(u,v)\}$ for AND vertices and $\min_{u \in S,\, (u,v) \in E}\{\infty; T^*(u)+d(u,v)\}$ for OR vertices. (The sense of $b^*_S(v)$ is the time bound implied on v by the edges in-coming it from S.) The following is the basic version of the algorithm.

AND/OR General(G)
$S \leftarrow \emptyset$
while $(S \neq V)$
    for each vertex $v \in V \setminus S$
        $b(v) \leftarrow b^*_S(v)$
    for each AND vertex v in $V \setminus S$ with in-coming non-zero edges from $V \setminus S$
        $b(v) \leftarrow \infty$

    AND/OR Unweighted($G^{zero}(V \setminus S), b, B$)

    for each vertex u that got the minimal early time m in $G^{zero}(V \setminus S)$
        $T^*(u) \leftarrow m$ /* in G */
        $S \leftarrow S \cup \{u\}$

4.2 Correctness Proof

Let us establish a few auxiliary properties:

Lemma 4.1 *For any AND/OR problem instance, if delays of some edges are increased, then:*

1. *Early times nowhere decrease.*

2. *Assume that delays were increased for non-zero edges only, connecting between vertices in some layer suffix $\mathcal{L}$. In the layers before $\mathcal{L}$ and in the first layer of $\mathcal{L}$, the early times remain the same.*

Proof: Item 1 is straightforward. Indeed, any solution feasible w.r.t. the new instance is feasible also w.r.t. the original one. Therefore, the early times at the new instance are at least those at the original one.

Item 2: Denote the first layer of $\mathcal{L}$ by $L_{t_{min}}$, and $(V \setminus V(\mathcal{L})) \cup L_{t_{min}}$ by V'. Let T' be equal T^* at V' and ∞ at the rest of the vertices. If $v \in V'$ is an AND vertex, by feasibility of T^*, no edge in-comes v from vertices in $V \setminus V'$, and any edge in-coming v from $L_{t_{min}}$ is a zero one (then, $v \in L_{t_{min}}$). Hence, the state at v is the same w.r.t. T' and w.r.t. T^*, that is T' satisfies equality (1) at v. By feasibility of T^*, for any OR vertex v in V', there exists a tight edge, which either in-comes it from a vertex in $V \setminus V(\mathcal{L})$, or in-comes it from $L_{t_{min}}$ and is a zero one (then, $v \in L_{t_{min}}$). Similarly, T' satisfies equality (2) at v.

Solution T' is obviously feasible for the new instance in $V \setminus V'$. Summarizing, T' is a feasible solution. By item 1, the early times for the new instance should be between their T^* and T' labels. Since T^* and T' coincide at V', there is no change in early times at V'. □

For any instance A of the AND/OR problem and any vertex subset S, let A_S denote the instance with S removed and time bounds b^*_S added at $V \setminus S$. (Equivalently, A_S is obtained by contracting S to *origin*, with replacing the bunch of edges from S to v by edge $(origin, v)$ with delay $b^*_S(v)$, for every $v \in V \setminus S$.)

Lemma 4.2 *The early times of A and A_S coincide on $V \setminus S$.*

Proof: Let us denote the early times of A_S by T^*_S. It is straightforward that the restriction of T^* to $V \setminus S$ is a feasible solution to A_S, and hence $T^* \geq T^*_S$. Consider the extension of T^*_S to V, denoted $T^{*\prime}$, as T^* on S and $\min\{T^*_S, T^*\}$ on $V \setminus S$. $T^{*\prime}$ is feasible w.r.t. A at vertices in $V \setminus S$ as the minimum of two feasible solutions. It is feasible also at vertices in S, since for any edge (u, v) in-coming S from $V \setminus S$, $T^{*\prime}(u)$ is at most $T^*(u)$. Its feasibility implies $T^*_S \geq T^*$, and this suffices. □

We base on the following statement:

Lemma 4.3 *For any set S of labeled vertices, with correct T^* labels, let $L_{t_{min}}$ be the first layer that is not contained in S entirely. Then, by applying the while iteration of algorithm AND/OR General to S, T^* labels are assigned to all vertices in $L_{t_{min}} \setminus S$ and only to them, and those labels are correct.*

Proof: Let us consider three AND/OR problem instances: the current one A, A_S, and A_S^∞, obtained from A_S by changing the delays of all the non-zero edges between vertices in $V \setminus S$ to ∞. Their early times are denoted T^*, T_S^*, and $T_S^{\infty *}$, respectively. By Lemma 4.1(1), Lemma 4.2, and the definition of the layers, $T_S^{\infty *} \geq T_S^* = T^* > t_{min}$ on $(V \setminus S) \setminus L_{t_{min}}$. By Lemma 4.1(2) and Lemma 4.2, $T_S^{\infty *} = T_S^* = T^* = t_{min}$ on $L_{t_{min}}$. Therefore, exactly $L_{t_{min}} \setminus S$ is the first layer of A_S^∞, labeled by t_{min}.

Note that if an edge with the infinite delay in-comes an AND vertex, then its early time is infinity. If it in-comes an OR vertex, then its early time remains the same if that edge is removed. Therefore, the early times and the layers are the same at A_S^∞ and at the instance $(G^{zero}(V \setminus S), b)$ that algorithm AND/OR Unweighted works on at the iteration.

By the above and since algorithm AND/OR Unweighted processes A_S^∞ labeling layer by layer in the inverse order, it assigns, at its last iteration, labels $T_S^{\infty *} = t_{min}$ exactly to the first layer $L_{t_{min}} \setminus S$ of A_S^∞, as required. □

Proposition 4.4 *At the end of each iteration of the basic version of algorithm AND/OR General, the early times are assigned exactly at the next layer, and they are correct.*

Proof: The proof is by induction on the iterations. The induction assumption is that at the beginning of the current iteration, the set S is formed by the vertices of some prefix of the layer sequence, with correct early times T^*. The basis $S = \emptyset$ is trivially correct. The induction step is straightforward from Lemma 4.3. □

4.3 Extended version

4.3.1 Handling Zero AND Cycles

Let us call a cycle consisting of AND vertices and zero edges only a *zero AND cycle*. We begin with a reduction of the general AND/OR problem to the case, where there is no zero AND cycle in the precedence graph. Note that the same reduction works for the classic PERT problem.

Observe that all vertices laying on a zero AND cycle C should have the same early time. Hence, we may contract them to a single AND super-vertex v_C, process the obtained graph, and then expand v_C, copying its early time to all original vertices lying on C.

Contraction as above should assign properly delays of edges incident to v_C. The bunch of the edges, if any, from vertex u to any $v \in C$ is replaced by a single edge (u, v_C) with the maximal delay. The bunch of the edges, if any, from all $v \in C$ to an AND vertex u is replaced by a single edge (v_C, u) with the maximal delay. That of the edges to an OR vertex u is replaced by (v_C, u) with the minimal delay. Notice that if there are non-zero edges between vertices on C, then all the vertices on C should have early time infinity. We model this by adding a self-loop (v_C, v_C) with delay 1.

It is easy to see that the above holds not only for the vertices of any zero AND cycle, but also for the vertices of any non-singleton strongly connected component of $G^{zero}(V^{AND})$. This is because two vertices lie on a cycle, in a graph, if and only if they belong to the same its strongly connected component.

The reduction first finds all strongly connected components of the graph $G^{zero}(V^{AND})$, e.g., by algorithm Kosaraju-Sharir. Then, all non-singleton ones are processed as above. The reduction correctness and its linear running time are obvious.

4.3.2 Algorithm and Its Correctness

The following is the extended version of the algorithm:

AND/OR General(G)

pre-process zero AND cycles
$S \leftarrow \emptyset$

/* Phase AND */
while (there is a vertex $u \in V^{AND}$ without in-coming edges)
    $T^*(u) \leftarrow T(u)$
    $S \leftarrow S \cup \{u\}$

while $(S \neq V)$
    for each vertex $v \in V \setminus S$
        $b(v) \leftarrow b^*_S(v)$

for each AND vertex v in $V \setminus S$ with non-zero in-coming edges from $V \setminus S$
$b(v) \leftarrow \infty$

AND/OR Unweighted($G^{zero}(V \setminus S), b, B$)

for each vertex u that got the minimal T^* label m in $G^{zero}(V \setminus S)$
$T^*(u) \leftarrow m$ /* in G */
$S \leftarrow S \cup \{u\}$

/* Phase AND */
while (there is a vertex $u \in V^{AND}$ without in-coming edges)
$T^*(u) \leftarrow T(u)$
$S \leftarrow S \cup \{u\}$

post-process zero AND cycles

Proposition 4.5 *Each iteration of the extended version of algorithm AND/OR General assigns the early times at least at the next layer not labeled entirely. The assigned early times are correct.*

Proof: The correctness of the first Phase AND is obvious. The same holds for each next phase AND, assumed the previously given early times are correct. A proof similar to that of Proposition 4.4 establishes that the part of each main iteration preceding its Phase AND completes the correct early times at the next layer not labeled entirely. The following Phase AND may assign correct early times to more vertices. □

4.4 Implementation and running time

4.4.1 Implementation

Following is the suggested implementation of the extended version of algorithm AND/OR General. We denote by $indeg^+(v)$ the number of non-zero edges in-coming v. Priority queue Q is implemented as a balanced tree.

Routine Scan(u)
for each edge $(u, v) \in E$
if $v \in V^{AND}$
if $T(v) < T(u) + d(u, v)$
$T(v) \leftarrow T(u) + d(u, v)$

$ChangeKey(Q, v, T(v))$
$indeg(v) \leftarrow indeg(v) - 1$
if $indeg(v) = 0$
$V^0 \leftarrow V^0 \cup \{v\}$
if $d(u, v) > 0$
$indeg^+(v) \leftarrow indeg^+(v) - 1$
if $indeg^+(v) = 0$
$V^+ \leftarrow V^+ \setminus \{v\}$
else /* $v \in V^{OR}$ */ **if** $T(v) > T(u) + d(u, v)$
$T(v) \leftarrow T(u) + d(u, v)$
$ChangeKey(Q, v, T(v))$

AND/OR General(G)

pre-process zero AND cycles
compute $indeg(v)$ and $indeg^+(v)$, for all $v \in V^{AND}$
$V^0 \leftarrow \{v \in V^{AND} : \ indeg(v) = 0\}$
$V^+ \leftarrow \{v \in V^{AND} : \ indeg^+(v) > 0\}$
$T(v) \leftarrow 0$, for all $v \in V^{AND}$
$T(v) \leftarrow \infty$, for all $v \in V^{OR}$
$Q \leftarrow (V, T)$

/* Phase AND */
while (V^0 is non-empty)
choose a vertex u from V^0
$T^*(u) \leftarrow T(u)$
$Q \leftarrow Q \setminus \{u\}$
Scan(u)

while ($Q \neq \emptyset$)
for each vertex v in Q
$b(v) \leftarrow T(v)$
for each vertex v in V^+
$b(v) \leftarrow \infty$

form B by passing on Q and V^+
AND/OR Unweighted($G^{zero}(V \setminus S), b, B$)

for each vertex u that got the minimal T^* label m in $G^{zero}(V \setminus S)$
$T^*(u) \leftarrow m$ /* in G */

$Q \leftarrow Q \setminus \{u\}$
Scan(u)

/* Phase AND */
while (V^0 is non-empty)
choose a vertex u from V^0
$T^*(u) \leftarrow T(u)$
$Q \leftarrow Q \setminus \{u\}$
Scan(u)

post-process zero AND cycles

We now show that the above algorithm is a proper implementation of algorithm AND/OR General. It is easy to see that V^0 is maintained as the set of all AND vertices with no in-coming edge. Hence, the loop in each Phase AND is arranged properly. In addition, V^+ is maintained as the set of all AND vertices with at least one in-coming non-zero edge. For all vertices v in $V \setminus S$, executions of Scan maintain the value $T(v)$ be equal b^*_S. Hence, the bounds b are assigned the proper values.

The set of elements of Q is exactly $V \setminus S$, so also the main while loop is as required. The set B may be made ordered by scanning the balanced tree Q and by forcing the vertices in V^+ be at the beginning of B.

4.4.2 Running Time

In what follows, a *linear time* is a time linear in the size of the precedence graph, that is in $|V| + |E|$. By Section 4.3.1, pre- and post-processing zero AND cycles is done in a linear time. The rest of the initialization also can be done in a linear time.

There are $|E|$ executions of routine Scan. A $ChangeKey$ operation costs $O(\log |V|)$, while the rest of a routine Scan execution costs $O(1)$. Hence, all routine Scan executions cost $O(|E| \log |V|)$ in total.

The for-each and while iterations at the end of the main while iteration happen once per vertex, so they cost $O(|V|)$ in total, not including executions of routine Scan.

Let us bound now the uncounted yet cost of a single main while iteration. Setting b values costs $O(|V \setminus S|) = O(|V|)$. In B, all vertices of V^+ are at the beginning. Passing on the balanced tree Q in the sorted order of keys T may be done in $O(|V|)$ time. An execution of algorithm AND/OR Unweighted

costs a time linear in the size of the graph, that is $O(|V| + |E^{zero}|)$, which dominates the other addenda.

Clearly, there are no more than $|V|$ main while iterations, since at each iteration, at least one vertex is labeled. This implies the time bound $O(|E| \log |V|)+|V|^2+|V|\cdot|E^{zero}|)$ of the *basic version* of algorithm AND/OR General.

Lemma 4.6 *At the* extended version*, at least one OR vertex is labeled at each non-last while iteration.*

Proof: Assume to the contrary that at some while iteration, the set V' of vertices given the minimal label m during the execution of algorithm AND/OR Unweighted contains AND vertices only. Choose arbitrarily a vertex v_1 in V'. By the stopping condition of the last Phase AND, there exists an edge (v_2, v_1), where v_2 is unlabeled. Since before the iteration of AND/OR Unweighted localizing the layer V' property (*) holds, v_2 belongs to V'. If $d(v_2, v_1)$ is non-zero, then the inequality $m = T^*(v_2) \geq T^*(v_1) + d(v_2, v_1) > T^*(v_1) = m$ implies $m = \infty$, that is the current iteration of AND/OR General is the last one. So, at any its non-last iteration, (v_2, v_1) is a zero edge. Similarly, there exists an edge in-coming v_2 from a vertex in V', and so on. Since V' is finite, the constructed vertex sequence repeats its vertices, that is contains a cycle. By the above, that cycle is a zero AND cycle, a contradiction to preprocessing zero AND cycles made at the beginning of the algorithm AND/OR General. □

We thus proved that the extended version of algorithm AND/OR General has at most $|V^{OR}|$ main iterations. Hence, its running time is $O(|E|\cdot\log |V|+ |V^{OR}| \cdot |V| + |V^{OR}| \cdot |E^{zero}|))$. We result in the following theorem:

Theorem 4.7 *Algorithm AND/OR General (the extended version) solves the AND/OR problem in time* $O(|E|\cdot\log |V|+|V^{OR}|\cdot|V|+|V^{OR}|\cdot|E^{zero}|) = O(|V||E|)$.

Let us consider $|V^{OR}|$ and $|E^{zero}|$ as *parameters* of an AND/OR instance. Their values may be small for certain sub-classes of the AND/OR problem, which may decrease the running time bound as follows.

Corollary 4.8
- *If* $|V^{OR}|$ *is* $O(\frac{|E|\cdot\log |V|}{|V|})$ *and* $|V^{OR}| \cdot |E^{zero}|$ *is* $O(|E| \cdot \log |V|)$*, then the running time is* $O(|E| \cdot \log |V|)$.

- *If* $|E^{zero}|$ *is* $O(|V|)$*, then it is* $O(|E| \cdot \log |V| + |V|^2)$*.*
- *If* $|V^{OR}|$ *is* $O(\frac{|E| \cdot \log |V|}{|V|})$*, then it is* $O(|E| \cdot (|V^{OR}| + \log |V|))$*.*

4.5 More Fastening

This section shows that, under certain conditions, a single iteration of algorithm AND/OR General provides the correct labels not only to the single next layer, but also to a few following layers. The presentation in this section is a kind of sketch.

We base on Lemmas in Section 4.2, see notation there. Let d_1 denote the minimal delay of a non-zero edge from $L_{t_{min}}$ to $(V \setminus S) \setminus L_{t_{min}}$. Consider the layer $L_{t_{min2}}$ next after $L_{t_{min}}$, and assume that its early time t_{min2} is less than $t_{min} + d_1$. Let us pass on the proof of Lemma 4.1(2) as applied to vertices in $L_{t_{min2}}$, instead of $L_{t_{min}}$ (for this, we replace V' by $V_2' = V \setminus V(\mathcal{L})) \cup (L_{t_{min}} \cup L_{t_{min2}})$, $V(\mathcal{L}) \setminus L_{t_{min}}$ by $V(\mathcal{L}) \setminus (L_{t_{min}} \cup L_{t_{min2}})$, etc.), while taking into account that the early times do not change at $L_{t_{min}}$. It is easy to check that it proves correctly that the early times do not change at $L_{t_{min2}}$ as well. (For illustration, see Figure **??**.)

Assume now that the early time at the next layer *of the new instance* is less than $t_{min} + d_1$. Then, by Lemma 4.1(1), also $t_{min2} < t_{min} + d_1$ holds, and the above proof leads to the same result. As a consequence, we may extend the labeling phase of the iteration of algorithm AND/OR General by: (i) checking whether the early time $m2$ at the layer computed before the last by algorithm AND/OR Unweighted is less than $t_{min} + d_1$, and if so, (ii) copying also the labels $m2$ to G. Lemma 4.3 may be extended correspondingly, implying the extended versions of Propositions 4.4 and 4.5.

We may continue in a similar way: After assigning early times to layers $L_{t_{min}}$ and $L_{t_{min2}}$, we compute also the minimal delay, d_2, of a non-zero edge from $L_{t_{min2}}$ to $(V \setminus S) \setminus (L_{t_{min}} \cup L_{t_{min2}})$, and check whether the following, third layer of $(G^{zero}(V \setminus S), b)$ has a "safe" early time $t_{min3} < \min\{t_{min} + d_1, t_{min2} + d_2\}$. If so, we assign also at that layer the early time t_{min3}, in G, and so on up to the first "unsafe" layer. Then, we pass to the next main iteration of the algorithm.

All the above is a heuristics, which may be more or less useful. However, also a new worst case bound may be established, using the above approach. Assume that some upper bound to the maximal finite early time, $\bar{T}$, is available, and let $d_{min} > 0$ be the minimal delay of a non-zero edge, in G. When using the above technique, we are sure that the minimal assigned early

time increases, from any iteration to the next one, at least by d_{min}. Hence, there are at most $\frac{\bar{T}}{d_{min}} + 2$ iterations, in any execution of the algorithm. As a result, the factor $|V^{OR}|$ could be replaced by $\min\{|V^{OR}|, \frac{\bar{T}}{d_{min}} + 2\}$, in the running time bound of Theorem 4.7. Note that the value $\frac{\bar{T}}{d_{min}}$ is *scalable*, that is does not depend on the time unit.

5 Characterization of infeasible AND/OR Graphs

We call a scheduling instance *feasible* if it admits a finite schedule, and *infeasible* otherwise. Recall that a PERT instance is infeasible if and only if there exists a positive cycle, in G. This characterization may be formulated in reachability terms as follows: *There is a non-zero edge, whose tail is reachable from its head, in G.* In this section, we generalize this criterion to AND/OR graphs.

Let W be an arbitrary set of edges and vertices of G, containing all edges out-going its vertices. Vertices in W are said be AND/OR reachable from W in no step. An AND vertex v is said be AND/OR reachable from W in one step if at least one edge in-coming v is in W. An OR vertex v is said be AND/OR reachable from W in one step if all edges in-coming v are in W (note that in particular, this holds for any W if there is no edge in-coming v, in G). A sequence $(W, w_1, w_2, \ldots, w_k = w)$ is called an *AND/OR path* from W to w in G if each w_i, $i = 1, 2, \ldots, k$, is reachable from $W \cup \{w_1, w_2, \ldots, w_{i-1}\} \cup \{(u, v) \in E : u = w_j, 1 \leq j \leq i-1\}$ in one step. Vertex w is said to be *AND/OR reachable* from W if there exists an AND/OR path from W to w.

Theorem 5.1 *An AND/OR graph $G = (V, E, d)$ is infeasible if and only if there exists a subset W of its non-zero edges, such that all tails of edges in W are AND/OR reachable from W.*

Proof: *If*: Let W be an edge set as in the statement. Assume, to the contrary, that no vertex has T^* label ∞. Denote the set of tails of edges in W by V_0, $\min_{v \in V_0} T^*(v)$ by $t_0 < \infty$, and $\min_{(u,v) \in W} d(u, v)$ by $\Delta > 0$. Notice that for all vertices w AND/OR reachable from W, holds $T^*(w) \geq t_0 + \Delta > t_0$ (this is easy to prove by induction on the steps in an AND/OR path). By the definition of W, this holds, in particular, for all vertices in V_0, a contradiction to the definition of t_0.

Only if: Let G be an infeasible AND/OR graph. Consider the execution of the basic version of algorithm AND/OR General on G. By

Proposition 4.4, its last iteration labels the entire $V \setminus S \neq \emptyset$ by ∞. By the algorithm, in this case, all vertices in $V \setminus S$ are AND/OR reachable in $(G^{zero}(V \setminus S), b)^{ruler}$ from $\{z^\infty\}$. In other words, all vertices in $V \setminus S$ are AND/OR reachable in $(G^{zero}(V \setminus S), b)^{origin}$ from the set of AND vertices v with $b(v) = \infty$ (note that edges from z^∞ to OR vertices do not matter for their AND/OR reachability). By the definition of function b, those are the AND vertices with at least one in-coming non-zero edge from $V \setminus S$. By the definition of AND/OR reachability, all vertices in $V \setminus S$ are AND/OR reachable in $(G(V \setminus S), b)^{origin}$ from the set W of non-zero edges of $G(V \setminus S)$.

Observe that in G, no edge in-comes any OR vertex $v \in V \setminus S$ from S, since otherwise, $T^*(v)$ could not be ∞. Note also that *origin* cannot be AND/OR reachable from W in $(G(V \setminus S), b)^{origin}$, since it has no in-coming edges.

Let us show that the AND/OR reachability of $V \setminus S$ from W holds also in G. Indeed, vertex *origin* is not in the game, while excluding it, AND/OR reachability of a vertex in one step in $G(V \setminus S)$ remains in G: for an AND vertex straightforwardly and for an OR one by the above observation. In particular, all tails of edges in W are AND/OR reachable from W in G, as required. □

An example see in Figure **??**. Let us set $W = \{(d, c), (c, f), (c, g)\}$. The set of the tails of its edges is $\{c, d\}$. Paths (W, c) and (W, f, g, d) are AND/OR paths, and they reach set $\{c, d\}$. Hence, the presented AND/OR instance is infeasible.

Note that the criterion of Theorem 5.1 is constructive: Suppose that solving a given AND/OR problem instance, e.g., by algorithm AND/OR General, revealed a non-empty set V^∞ of vertices with infinite early times. By the theorem proof, an infeasibility certificate W as in the theorem can be easily found as the set of non-zero edges between vertices in V^∞.

6 Discussion on Related Work

1. The AND/OR graph problem setting considered by Knuth in [8] arises from a certain problem on grammars. Correspondingly, the AND/OR graphs considered there are bipartite, where the parts are the sets of AND and OR vertices, and such that there is a *single out-going edge from each AND vertex*. The specifics of the motivation restricted the set of possible time-schedules considered in the paper: the events in any solution are essentially ordered

by the causal relation. In addition, the problem setting implies absence of zero cycles. The algorithm suggested in [8] is the same as in [5, 4].

Let us see how the above specifics restricts the generality of the arising AND/OR problem. A simple transformation of a general graph to an equivalent bipartite one, without enlarging the problem size, is suggested in [1]. Taking care of AND vertices with multiple out-going edges is more heavy. The reduction from the general AND/OR problem to the case of [8], known to us, leads to a substantial increase of the graph size, and thus of running time bounds.

The reduction takes care of each AND vertex, as follows: Let v be an AND vertex with out-going edges $(v, w_1), \ldots, (v, w_k)$. We add a new OR vertex w and AND vertices $v_1, \ldots v_k$, add a zero edge (v, w), and replace each edge (v, w_i) by the sequence of two edges: (w, v_i), $d(w, v_i) = d(v, w_i)$, and a zero edge (v_i, w_i). For an illustration, see Figure **??**.

As a result, the number of AND vertices, in the graph, increases by the total number of *edges* out-going AND vertices in the original graph. Correspondingly, any running time bound established for the AND/OR problem of [8] converts to a bound for the general AND/OR problem, where each $|V|$ term is replaced by $|E|$. For example, the bound $O(|E| + |V| \log |V|)$ of the algorithm of [8] (based on the bound of Dijkstra algorithm implemented with Fibonacci heaps), converts to a worse bound $O(|E| \log |V|)$ for the general case.

The property of events ordering by the causal relation is used in the correctness proof in [8]. This prevents applying the proof to the general AND/OR setting. That is, the algorithm correctness for the general case is not established in [8].

Remark: It is worth to mention that the setting of Knuth is more general than that in [5, 4] in the definition of the precedence restriction for an AND vertex. Instead of the maximum function only, a more general class of so called "superior" functions is considered in [8].

2. The class of AND/OR graphs considered by Moehring et al. in [9] is symmetric, in a sense, to the class considered in [8]: there is a *single edge out-going each OR vertex*. It is mentioned in [9] that this specifics does not restrict the generality of the results established in [9], since there is a simple reduction from the general case to that one. Indeed, a reduction similar to that described above converts a general AND/OR graph to an AND/OR graph in that class.

However, similarly to the case of [8], that reduction increases the number

of OR vertices by the number of edges out-going all OR vertices. Therefore, using that reduction spoils the running time bounds of algorithms developed for the model of [9]. In particular, an algorithm solving the general AND/OR problem (i.e., with zero cycles are allowed) is suggested in [9]. The running time bound $O(|V||E|)$, established for it in [9], becomes $O(|E|^2)$, when combined with the reduction from the general AND/OR problem.

3. Following paper [9], V. Kääb studied the influence of small changes made to edge delays, in an AND/OR graph, to the schedule makespan [7, Chapter 2]. Four types of critical sets were defined and thoroughly studied, in this context.

In the model of [7], the AND/OR graph is general, but edge delays are restricted be all positive. In what follows, we show that all *propagation properties* of sufficiently small changes made to edge delays, in an AND/OR graph, remain the same if zero edge delays are allowed, but zero cycles are not. This equivalence extends the results of [7, Chapter 2] to this less restricted model. It may be useful also in other contexts.

Recall that an edge (u, v) is said be *tight* if $T^*(v) = T^*(u) + d(u, v)$. By definition, non-tight edges do not matter for defining early times. As well, sufficiently small delay changes at non-tight edges do not influence early times ([7, Remark 2.2.6]). Based on this, we may remove all non-tight edges from the AND/OR graph, while remaining with the same early times and the same propagation properties of sufficiently small delay changes. Let the resulting AND/OR graph be denoted $G^{tight} = (V, E^{tight}, d)$.

Notice that G^{tight} is *acyclic*. Indeed, since edge delays are non-negative, early times may only grow along any path, and thus should be the same on any cycle, a contradiction to the assumed absence of zero cycles. It is easy to see that the delay values themselves do not matter at all, when studying propagation of small delay changes, only the structure of G^{tight}. Indeed, let $\epsilon(u, v)$ denote the increase of $d(u, v)$, for any $(u, v) \in E^{tight}$, and $\Delta^*(v)$ denote the corresponding increase of $T^*(v)$, for any $v \in V$. Since for all $(u, v) \in E^{tight}$ holds $T^*(v) = T^*(u) + d(u, v)$, the inequality

$$(T^*(v) + \Delta^*(v)) = \max_{(u,v)\in E^{tight}} \{0, (T^*(u) + \Delta^*(u)) + (d(u, v) + \epsilon(u, v))\}$$

is equivalent to

$$\Delta^*(v) = \max_{(u,v)\in E^{tight}} \{0, \Delta^*(u) + \epsilon(u, v)\} \ ,$$

for any $v \in V^{AND}$. A similar equivalence holds also for any $v \in V^{OR}$. That

is, $\Delta^*(v)$ are the early times in the AND/OR graph (V, E^{tight}, ϵ), irrelatively to the original edge delays.

Therefore, we may "play" with the original edge delays as we wish: Notice that *any* assignment of early times T'^* to the vertices defines the edge delays of all edges of G: $d'(u,v) = T'^*(v) - T'^*(u)$, so that set of tight edges remains the same, and hence the change propagation properties remain the same. In particular, we may assign early times so that a part of the resulting delays would be zero (and even negative, in general), retaining the same propagation properties. As a boundary case, we may define *all* early times and thus *all* edge delays of tight edges be zero, with the same result.

Probably, the study of delay change propagation properties of AND/OR graphs, and thus of criticality in the sense of [7, Chapter 2], may be facilitated, or at least made easier to explain, by concentrating on acyclic unweighted graphs (V, E^{tight}).

Let us consider the opposite direction: Given any AND/OR problem instance without zero cycles, we may compute the early times (e.g., by algorithm AND/OR General), find all tight edges, and construct the subgraph G^{tight}. Let the vertices be numbered in a topological order of G^{tight}, so that $V = \{v_i\}_{1 \le i \le |V|}$. Let us change all edge delays: $d^+(v_i, v_j) \leftarrow d(u,v) + (j-i) \cdot \delta$, in the entire G. Note that the new AND/OR graph G' has positive edge delays only, as in [7, Chapter 2]. If δ is sufficiently small, then $(G')^{tight}$ remains the same as G^{tight}, and thus the delay change propagation properties remain the same. Therefore, the results of [7, Chapter 2] are valid also for the AND/OR problem with zero delays allowed but without zero cycles.

Remark: Note that from the algorithmic point of view, computing a concrete change propagation in G^{tight} is trivial. Computing early times $\Delta^*(v)$, given the delay changes $\epsilon(u,v)$, may be done by consequently applying equalities (1) and (2) in a topological order of G^{tight}, in a linear time, with no relation to the original edge delays.

Acknowledgements

The authors are grateful to Eugene Levner for his permanent interest to the work and useful discussions, and to Seth Pettie for his valuable comments.

References

[1] G. M. Adelson-Velsky, A. Gelbukh, and E. Levner. A fast scheduling algorithm in AND-OR graphs. Mathematical Problems in Engineering, Hindawi Publishing Corporation, **8** (4/5), pp. 283293 (2003).

[2] G. M. Adelson-Velsky, and E. Levner. Project Scheduling in AND-OR Graphs: A Generalization of Dijkstra's Algorithm. *Mathematics of Operations Research* **27** (3), 504–517 (2002).

[3] T. Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms*, McGraw-Hill, 2001.

[4] E. A. Dinic. The fastest algorithm for the PERT problem with AND- and OR-vertices (the new-product-new technology problem). In *Proc. of the Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization (IPCO'90), Waterloo, Canada*, R. Kannan and W. R. Pulleyblank eds., Univ. of Waterloo Press, 1990, pp. 185-187.

[5] E .A. Dinic, A. B. Merkov, and I. A. Tejman. Coordination analysis and computing of early periods of launching for a set of new technologies, in: *Models and Methods for Forecast of the Science-Technology Progress*, V. V. Tokarev ed., Moscow, 1984, 125–131 (in Russian).

[6] Y. Dinitz, P. Carmi, S. Golan, and G. Rozenwald. An $O(|V||E|)$ Algorithm for Scheduling with AND/OR Precedence Constraints. A talk at the Seminar of Dept. of Computer Science, Holon Institute of Technology, Israel, January 2002.

[7] V. Kääb. Scheduling with AND/OR-Networks. Ph.D. thesis, Technical University of Berlin, 2003. Available at ftp://ftp.math.tu-berlin.de/pub/Preprints/combi/dissertation-kaeaeb.pdf .

[8] D. E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters* **6** (1977), no.1, pp.1-5.

[9] R. H. Moehring, M. Skutella, and F. Stork. Scheduling with AND/OR precedence constraints. *SIAM J.of Computing* 33 (2) pp. 393–415 (2004).