Mathematical Problems in Engineering, 2002, Vol. 8(4–5), pp. 283–293 Taylor & Francis
Taylor & Francis Group

On Fast Path-Finding Algorithms in AND–OR Graphs

GEORGE M. ADELSON-VELSKY[a,*], ALEXANDER GELBUKH[b,†] and EUGENE LEVNER[c,‡]

[a]*Department of Mathematics and Computer Science, Bar-Ilan University, Ramat Gan, Israel;* [b]*Center for Computing Research, National Polytechnic Institute, Mexico City, Mexico;* [c]*Department of Computer Science, Holon Institute of Technology, Holon, Israel*

(Received 7 February 2002; In final form 6 June 2002)

We present a polynomial-time path-finding algorithm in AND–OR graphs Given p arcs and n nodes, the complexity of the algorithm is $O(np)$, which is superior to the complexity of previously known algorithms.

Key words: AND–OR graphs; Extremal paths; Scheduling; Routing; Polynomial-time algorithms

1 INTRODUCTION

We address a mathematical programming problem with min–max inequalities that is reduced to finding extremal paths in AND–OR graphs. We precede the paper by a short note related to the history of the problem – we recall some graph-theoretical results that have been obtained in the halls of the Institute for Control Problems (ICP), Moscow, USSR, in the 1970s.

In 1970, Dinic [17] has improved the famous Ford-Fulkerson method for finding the maximum flow in a network by augmenting the flow along many shortest paths and delicately reconstructing the network, it has been the first *polynomial-time* max-flow algorithm. An even faster algorithm for the max-flow problem has been developed by Karzanov [23] who has introduced intermediate "pre-flows" that do not satisfy the conservation constraints at each node. Over the past years, a long sequence of more and more efficient max-flow algorithms, polishing and improving Dinic's and Karzanov's algorithms, has been proposed in the literature (see, *e.g.*, [7, 11, 22]). In 1978, Dinic [18] has addressed the shortest path problem in graphs and suggested a bucket-based version of Dijkstra's algorithm, today the bucket implementations proved to be most effective on many practical problems Arlazarov *et al.* [12] have devised a new construction for the transitive closure of a directed graph. Known in the West literature as "Four Russians' Algorithm", this efficient construction is widely used in matrix multiplication and other applications [10]. Weisfeiler, Adelson-Velsky and their collaborators [31] have investigated the classical graph isomorphism problem and designed an ingenious enumeration method based on an algebraic generalization of the arc length notion.

* E-mail: velsky@macs.biu.ac.il;
† E-mail: gelbukh@cic.ipn.mx;
‡ E-mail: levner@hait.ac.il

ISSN 1024-123X print; ISSN 1563-5147 online © 2002 Taylor & Francis Ltd

Adelson-Velsky [1] has studied general critical (longest) paths with non-linear arc lengths arising in a project management context. Levner [25, 26] has proposed to use the critical paths in graphs for solving scheduling problems, over the past years the approach became widespread in the scheduling theory (see, *e.g.*, [29]) Adelson-Velsky, Arlazarov, and their collaborators have created a chess-playing computer program KAISSA, which won the first world computer-chess championship, they have suggested fast and simple ways for evaluating the game states (*i.e.*, the node weights in game trees) and discarding non-perspective paths in the trees [2–6].

Today, the subject of path optimization in graphs had exploded, compared with its state in the 1970s. Nevertheless, the algorithmic ideas derived by the ICP people almost three decades ago are still vital and up to date.

The mathematical programming problem considered in this paper can be formulated as a path-finding problem in weighted directed AND–OR graphs in which arcs are identified with tasks while nodes represent their starting and finishing endpoints. A starting point of a task is represented by an *AND-node* if its execution can be started after all its preceding tasks have been solved, and by an *OR-node* if it can be started as soon as any one of its preceding tasks is solved. The time needed to execute a task is represented by an *arc length*. The problem that emerges, is to implement all the tasks in the graph in minimum time.

This type of problems has many real-world applications. Their study has been started by Elmaghraby [20] in the early 1960s. Crowston [14], and De Mello and Sanderson [15] have applied the problems in AND–OR graphs for the planning of production systems, Gillies and Liu [21] and Adelson-Velsky and Levner [8] have used them for scheduling of tasks in computer communication systems. The AND–OR graphs appear to be a powerful tool for the automatic text processing and problem solving in the artificial intelligence [28]. Routing problems in AND–OR graphs arise in mathematical analysis of extremal problems in context-free grammars [24], games [32], and hypergraphs [13].

The problem under consideration generalizes the shortest path and critical path problems in graphs. Several fast methods for its solution have been suggested in the literature. A polynomial algorithm has been suggested by Dinic [19] who has elegantly solved a special case when the graph has no zero-length cycles. Another special case – in which arc lengths are non-negative, AND–OR graphs are bipartite, arcs leading to the OR-nodes are of zero length, and zero-length cycles are permitted – has been solved by Adelson-Velsky and Levner [8, 9] in $O(pp')$ time (p' is the number of arcs entering the AND-nodes and p the total number of arcs). Mohring *et al.* [27] have solved an equivalent problem in bipartite graphs, with the same complexity. The present paper gives a full description of an improved, $O(np)$-time algorithm for general (not necessarily bipartite) AND–OR graphs with n nodes, and provides the proof of its correction.

The paper is organized as follows: in Section 2 we define the problem, Section 3 presents a new polynomial algorithm, Section 4 analyzes its properties, Section 5 concludes the paper.

2 PROBLEM FORMULATION

The input to the scheduling problem under consideration is $\langle G, s, \tau \rangle$, where $G = (V, E)$ is a directed graph, V is the node-set, $|V| = n$, E is the arc-set, $|E| = p$, $\tau = \tau(v_i, v_j)$ is an arc length function, and s is a node called the *start* whose occurrence time is given by $t(s) = t_0$.

We assume that $V = A \cup O \cup \{s\}$, A being the set of AND-nodes and O the set of OR-nodes. The problem is to find the earliest *occurrence times* $t(v_j)$, for all $v_j \in V$, satisfying the following conditions:

$$t(s) = t_0, \tag{1}$$

$$t(v_j) \geq \max_{v_i \in P(v_j)} (t(v_i) + \tau(v_i, v_j)) \quad \text{if } v_j \in A, \tag{2}$$

$$t(v_j) \geq \min_{v_i \in P(v_j)} (t(v_j) + \tau(v_i, v_j)) \quad \text{if } v_j \in O, \tag{3}$$

$$t(v_j) \geq t_0, \quad \text{for all } v_j. \tag{4}$$

Here $P(v)$ denotes the set of nodes that are immediate predecessors to v. Relations (2) formally describe the conditions occurring in the AND-nodes of standard PERT/CPM project management models, whereas relations (3) formally state that just one of the preceding activities for node v_j may be finished before the v_j occurs. The problem turns into the critical path problem if O is empty, and into the shortest path problem if A is empty. Without the loss of generality, we assume that $P(v)$ is non-empty for any node v, $v \neq s$ (otherwise, we would have several start nodes which could be glued together into a single start node). We will denote this problem by **P**.

Conditions (2) and (3) are represented in the graph G by the arcs from v_i to v_j of length $\tau(v_i, v_j)$. We start our considerations with the following graph transformations that permit us to present the constraints (4) in graph form (without violating the problem size order).

(a) if the graph has an OR-node u having immediate successors v_j of OR type, then we add a new AND-node $\bar{u}$ with $\tau(u, \bar{u}) = 0$, and the arcs (u, v_j) are replaced by the arcs $(\bar{u}, v_j)$ with $\tau(\bar{u}, v_j) = \tau(u, v_j)$ (see Fig. 1 below, where the squares denote OR-nodes and the circles AND-nodes),

(b) we add arcs (s, v_j) of zero length leading from the start s to all AND-nodes v_j in G (including those added in the previous transformation).

Due to these transformations, we may assume that any OR-node in G has a preceding AND-node or s, in particular, G does not contain zero-length cycles consisting only of OR-nodes. It should be noted that in contrast to the graph model in [8, 9], in this paper the arcs entering OR-nodes are allowed to be of *non-zero* length. If the graph has an AND-node u having immediate successors v_j of AND type, then we add a new OR-node $\bar{u}$ with $\tau(u, \bar{u}) = 0$, and the arcs (u, v_j) are replaced by the arcs $(\bar{u}, v_j)$ with $\tau(\bar{u}, v_j) = \tau(u, v_j)$, similar to the transformation (a). As a result, a general (non-bipartite) graph is reduced to a bipartite graph with the number of nodes increased not more than twice. Hence, without the loss of generality, from now on we will consider the *bipartite* graphs only.

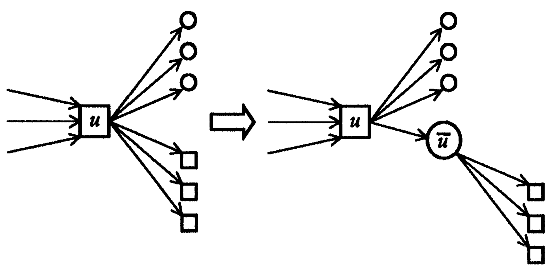

FIGURE 1 The graph transformation.

DEFINITIONS *A set of values* $\{t(v_j)\}$, $j = 1, \ldots, n$, *satisfying inequalities* (1) *to* (4) *is called a feasible solution to the problem* **P***. The feasible solution providing the minimum values* $t(v_j)$ *for all* v_j *among all feasible solutions, is called optimal, or earliest, occurrence times, and is denoted by* $\{t^*(v_j)\}$.

Let Γ be the graph obtained from G after the transformations (a) and (b) are done. Let $\mathbf{\Pi}$ denote the problem of finding the optimal occurrence times $\{t^*(v_j)\}$ in the new graph Γ subject to (1)–(4). Obviously, the problems **P** and $\mathbf{\Pi}$ are equivalent.

3 ALGORITHM

The new algorithm is based on the previous algorithm presented in [8, 9], differing from the latter in the following aspects: (i) the new algorithm is designed to treat general AND–OR graphs with possibly *non-zero arcs* entering OR-nodes, (ii) a revised labeling procedure permits to improve the algorithm complexity.

The algorithm works iteratively. At each iteration, it finds a node with the smallest occurrence time. In this respect, it is similar to the algorithms by Dijkstra [16] and Dinic [19]. However, in contrast to Dijkstra's or Dinic's algorithms, our algorithm is *not greedy* in the sense that, at each step, it does not make an immediate choice that looks best at the moment. In fact, it always behaves in a complementary, "anti-greedy", mode, namely, at each iteration, it first finds all the nodes whose occurrence times are *non-minimal* and paints them *red*, as a consequence, all the *unpainted* nodes are proved to gain the minimum occurrence times.

At each iteration, the graph is reduced to a smaller one. Let Γ_h denote the graph derived at the end of the $(h-1)$th iteration $(h = 1, 2, \ldots)$. For every node v in Γ_h the algorithm assigns a time label $t(v)$ and maintains a status $\mathrm{St}(v) \in \{\mathit{uncolored}, \mathit{red}, \mathit{black}\}$. All nodes start out *uncolored* and later become *red* or *black*. Initially, $t(s) = t_0$, the time labels for all other nodes are not defined. At termination, the algorithm either provides the minimum occurrence times for all nodes, or reveals that the given problem has no solution.

Each iteration consists of three procedures: Node-Painting, Node-Sorting, and Graph-Reduction. Consider an iteration h. Let $\mathrm{Out}(s)$ denote the set of all nodes v which are the heads of the arcs leaving s. Before the first iteration starts, we sort all nodes v_i in $\mathrm{Out}(s)$ in non-decreasing order of their lengths $\tau(s, v_i)$. For each OR-node, v_j, we compute its in-degree and assign its value to a variable r_j. We will re-evaluate (namely, decrement) the r_j-values so that when a current r_j becomes 0 this will mean that *all* the predecessors to node v_j are painted *red*.

Procedure Node-Painting consists of three steps, S1 to S3, where S3 is performed repeatedly. At each iteration, during these steps each arc is scanned at most once, and some yet-not-colored nodes are painted *red*, until at some instant no uncolored node can be painted *red*.

Intuitively, the main idea behind the painting procedure at Steps S1–S3 below is to paint *red* all the nodes in Γ_h which have the occurrence times *greater* than the earliest occurrence time possible in Γ_h, as a consequence, all the unpainted nodes will have the time labels $t(v)$ *equal* to the *earliest* occurrence time among the nodes of Γ_h. Further, we will paint *red* all the not-yet-painted AND- and OR-nodes taking into account the following rules.

(i) Any AND-node may be painted *red* if any of its predecessors is *red*, and
(ii) any OR-node may be painted *red* if *all* of its predecessors are already *red*

Now we can describe the steps of the painting procedure at each iteration.

Step S1 [*Initial painting*] For each positive-length arc (v_i, v_j), $v_i \neq s$, consider its head v_j. If it is an AND-node then paint it *red*. If it is an OR-node then decrement r_j by 1, when r_j becomes 0, paint v_j *red*.

Step S2 [*Initial painting continued*] For each zero-length arc (v_i, v_j) such that its tail v_i is red, consider its head v_j. If it is an AND-node then paint it *red*. If it is an OR-node then decrement r_j by 1, when r_j becomes 0, paint v_j *red*. When a node is painted *red*, all its leaving arcs of zero length are added to a FIFO queue to be examined and processed in the same manner later at this step.

Two cases are possible after the initial painting at Steps S1–S2 is finished.

Case C1 All nodes are painted *red*. This means that the initial graph has a cycle of positive length, and moreover, the times of the nodes in the cycle will become infinitely large when the algorithm runs (infinitely long) further. In this case, the problem has no feasible solution, so the algorithm reveals this fact and stops.

Case C2 Some nodes (or, possibly, all of them) are not painted *red*. Then an unpainted node of maximum distance from s (that is, of maximal $\tau(s, v_i)$-value) is chosen as a *pivot*, and the algorithm goes to Step S3.

Step S3 [*Choosing a pivot and painting*] Among the unpainted nodes in Out(s), choose an AND-node v^{**} of maximum length $\tau(s, v,)\tau^{**} = \tau(s, v^{**}) = \max_{i \in \mathrm{Out}(s)} \tau(s, v_i)$, and paint it *red*. Then paint *red* the not-yet-painted nodes, as described in the rules (i)–(ii) just above starting from the node v^{**}. If there are still not painted nodes in Out(s), then the operation just described is executed with the next maximal node in Out(s). That is, the algorithm chooses the (next) maximal not-yet-painted node in Out(s), paints it *red* and applies the rules (i)–(ii). This "descend" is repeated until all nodes in Out(s) become painted *red*. When all nodes in Out(s) become *red*, select the last node considered as v^{**}, and denote it as v^*.

The node v^* is painted *black*, the nodes connected with the v^* by zero-length arcs are painted *black* as well. All *black* nodes have the minimum time $t(v^*)$ in Γ_h. This finishes the procedure Node-Painting.

Remarks

(1) If the problem is unfeasible due to the presence of a positive-length cycle in a given graph, then at the beginning of some iteration all arcs (s, v_i) in a current graph Γ_h will be of the same length, and, after Steps S1–S2, all the nodes in Γ_h will be painted *red*.
(2) At Steps S1–S3 of each iteration, each arc is examined at most once.
(3) All the arcs examined at Steps S2–S3 are of length zero.
(4) All the nodes are reachable from the *start* s through directed paths in G. Indeed, for the AND-nodes this trivially follows from the preliminary transformation (b) described above, for the OR-nodes it follows from the fact that any such node is preceded by an AND-node or s (this follows from the transformation (a)). Therefore, in Case **C2**, Out(s) contains at least one unpainted node.

After the procedure Node-Labeling is finished, Graph-Reduction is applied. Let v' be an immediate successor of v^*, and v'' an immediate successor of v' (where, of course, v' is of

OR-type, and v'' of AND-type). The node v^* is removed from Γ_h, and the arcs incident to v^* are reconstructed as it is described below.

(a) If $\tau(v^*, v') = 0$, then (v^*, v') is removed from Γ_h, node v'' is made connected with s by an arc (s, v'') whose length is defined as follows. If there is no arc (s, v'') in Γ_h, then such an arc is inserted in the graph (instead of the removed arcs, (v^*, v') and (v', v'')), with $\tau(s, v'') = \tau(s, v^*) + \tau(v', v'')$. If there is an arc (s, v'') in Γ_h, then $\tau(s, v'')$ is recalculated as follows $\tau(s, v'') = \max(\tau(s, v''), \tau(s, v^*) + \tau(v', v''))$. After all nodes v'' are scanned and connected with s, OR-node v' is removed from Γ_h.
(b) If $\tau(v^*, v') > 0$, then a new AND-node w is added into each arc (v^*, v') of this kind, with $\tau(v^*, w) = \tau(v^*, v') > 0$ and $\tau(w, v') = 0$. Then (v^*, w) is removed from Γ_h, node w is made connected with s by an arc (s, w) whose length is defined as follows $\tau(s, w) = \tau(s, v^*) + \tau(v^*, v')$.
(c) If, at some iteration, an AND-node w preceding an OR-node v' have been added to Γ_h, and at some later iteration, another "new AND-node" w' preceding the same OR-node v' is to be added (according to the rule described in paragraph (b)), then only one of them, which provides a smaller distance from s to v', is left $\tau(s, v') = \min(\tau(s, w) + \tau(w, v'), \tau(s, w') + t(w', v'))$. Another node, together with its incident arcs, is removed.
(d) When all arcs leaving v^* are removed from Γ_h during the transformations (a)–(c) above, the node v^* and the arc (s, v^*) are removed as well.
(e) The new AND-nodes w are added to Out(s) and ordered so that all nodes v_i in Out(s) remain sorted in non-decreasing order of their lengths $\tau(s, v_i)$. Notice that the resulted graph is also bipartite.

During the graph reduction, the total number of nodes in Out(s) changes from iteration to iteration at first it may grow due to new AND-nodes added (but it never exceed n), and then this number decreases. The algorithm finishes when the set Out(s) is exhausted, all node times having been defined. Otherwise, a new iteration (starting from Step S1) is executed with the reduced graph Γ_h, with all its nodes made again unpainted. As a result of Graph-Reduction, either an OR-node is removed from the current graph Γ_h, or at least one "old" AND-node is substituted by a "new" AND-node (if the latter will be chosen as v^* at a later iteration then an OR-node will be removed). Thus, the total number of executions of the procedure Graph-Reduction cannot exceed $n_{\text{AND}} + n_{\text{OR}} = n$ (where, clearly, n_{AND} is the number of AND-nodes, and n_{OR} the number of OR-nodes in the initial graph G).

4 ALGORITHM ANALYSIS

At each iteration, either (i) an AND-node is removed from the graph Γ_h (see paragraph (d) in the previous section), or (ii) an OR-node is removed from the graph (see paragraph (a) in the previous section), or (iii) an AND-node all leaving arcs of which are positive is substituted by a new AND-node (or by several new AND-nodes) with a leaving arc of zero length. Clearly, the situation (iii) may happen not more than n_{AND} times where n_{AND} is the number of AND-nodes in G. Although the total number of arcs leading from the AND-nodes in Out(s) to their immediate OR-successors may be quadratic in n (and the number of new AND-nodes added to Out(s) even may grow at some iterations), $|\text{Out}(s)|$ never exceeds the total number n of nodes in G. This property is guaranteed by the rule in paragraph (c).

To summarize, after each iteration, the following states are possible:

(1) either all of the nodes are painted *red* after the steps S1 and S2 of Node-Painting, then the problem has no solution, or

(2) all AND-nodes of the initial graph G have been removed (so that their minimum times have been defined, and, therefore, the times for all OR-nodes are also defined), this means that the problem has been solved, or
(3) after Graph-Reduction terminates, set Out(s) is not yet empty, then the algorithm goes to the next iteration. Observe that this construction does not mimic the structure of Dijkstra's shortest-path algorithm.

Consider now the properties of the algorithm in more detail.

DEFINITIONS *Let $t(v_j)$ be the optimal solution to the problem Π. Any arc (v_i, v_j) such that $t(v_j) = t(v_i) + \tau(v_i, v_j)$ where $v_i \in P(v_j)$, is called critical, or binding. A path is called critical if it either consists of a single node or consists of critical arcs. Given the optimal solution to the problem Π and a node v_j, a critical sub-network $N_{\text{cr}}(v_j)$ originating at v_j is defined recursively as follows*: (1) *$v_j \in N_{\text{cr}}(v_j)$, and* (2) *if $v_k \in N_{\text{cr}}(v_j)$ and $t(v_j) = t(v_i) + \tau(v_i, v_j)$, then $v_i \in N_{\text{cr}}(v_j)$ and $(v_i, v_k) \in N_{\text{cr}}(v_j)$.*

We can see that the critical path in the problem Π may be neither the shortest nor the longest path in Γ (see Ref. [9], for details and examples).

The following theorem establishes the relationships between the critical paths and the optimal values of the decision variables, $t^*(v_j)$.

THEOREM 1 [9]

1. *If $v_i \in N_{cr}(v_j)$, then there exists a simple (i.e., acyclic) critical path $L = (v_i, \dots, v_j)$, starting at v_i and terminating at v_j, in which all nodes and arcs belong to $N_{cr}(v_j)$.*
2. *The length $\lambda(v_i, \dots, v_j)$ of the critical path L, defined as $\sum_{s=i}^{j-1} \tau(v_s, v_{s+1})$, equals $t^*(v_j) - t^*(v_i)$.*
3. *If the problem Π has a feasible solution, then the starting node s belongs to all critical sub-networks $N_{cr}(v_j)$, $j = 1, \dots, n$.*

The proof, which follows directly from the definitions of the critical sub-network and the critical path, is skipped here. (The detailed proof can be found in [9]).

Consider now an iteration h. Assign the labels {*red*, *black*} to the nodes of Γ_h as described in the previous section, and define the ranks of the *red*-labeled nodes as follows.

The first pivot v^{**} labeled *red* at the first run of Step S3 receives rank $k = 1$, when a node v is painted *red* at a current run of Step S3 through an arc (u, v), then the rank of v is set to the rank of u plus 1 (The AND-nodes receive odd ranks while the OR-nodes receive even ranks). The following two lemmas establish useful properties of the *black*- and *red*-labeled nodes.

LEMMA 1 (*"The times of red-labeled nodes $v \neq v^*$, is not minimal"*) *If the optimal solution to Problem Π exists, then the earliest occurrence time of any red-labeled node, except for v^* and those painted by v^*, is greater than $t_0 + \tau^* = t_0 + \min_{t \in \text{Out(s)}} \tau(s, v_i)$, where Out($s$) is the set of heads of the arcs leaving the start node s in Γ_h.*

Proof Any node painted *red* at Steps S1 and S2 evidently does not possess the minimum time because its time is at least the sum of the minimum time plus a positive arc length. The proof for the nodes painted at Step S3 is by induction on the rank k. Consider any iteration, say h. Let Λ_k denote the set of all *red*-labeled nodes of the rank k at that iteration.

Let us first verify the result for $k = 1$. For the node $v^{**} \in \Lambda_1$, we have

Since $v^{**} \neq v^*, s \in P(v^{**})$, and $\tau(s, v^{**}) > \tau^* = \tau(s, v_1) = \min_{t \in \text{Out(s)}} \tau(s, v_i)$, then $t(V^{**}) \geq t(s) + \tau(s, v^{**}) > t_0 + \tau(s, v_1)$.

Suppose that the required result is true for all the ranks not greater than k, that is, for the earliest occurrence times of the nodes v_i from $\cup_{s\leq k}\Lambda_s$, $t(v_i) > t_0 + \tau(s, v_1)$.

Let $v_j \in \Lambda_{k+1}$. Then we have

$$\text{If } v_j \in A \text{ and } v_i \in P(v_j) \cap \Lambda_k \text{ then } t(v_j) \geq t(v_i) + \tau(v_i, v_j) = t(v_i) > t_0 + \tau(s, v_1),$$
$$\text{If } v_j \in O \text{ and } v_i \in P(v_j) \cap (\cup_{s\leq k}\Lambda_s), \text{ then } t(v_j) = \min_{v_i \in P(v_j)} t(v_i) > t_0 + \tau(s, v_1)$$

Lemma is proved. ■

LEMMA 2 (*"The time of any black-labeled node is minimal in Γ_h"*) *Let Γ_h be a graph obtained at the hth iteration of the algorithm. If a feasible solution to* Π *in graph Γ_h exists, then the earliest occurrence time of the black-labeled node v^* is equal to* $t_0 + \tau^* = t_0 + \tau(s, v^*) = t_0 + \min_{t\in \mathrm{Out}(s)} \tau(s, v_i)$.

Proof The proof is based on three simple observations. First, notice that the choice of any pivot node v^{**} is done among those nodes in Out(s) of which the incoming arcs (v, v^{**}), $v \neq s$, are all of zero length (otherwise v^{**} cannot be of minimal "distance" (=time) from s in a current graph, and it is painted at Step S1). A pivot v^{**} being fixed, we will call "*lower nodes*" those nodes v in Out(s) for which $\tau(s, v) \leq \tau(s, v^{**})$, and "*upper nodes*" v for which $\tau(s, v_1) > \tau(s, v^{**})$.

Our second observation is as follows. If a v^{**}, being painted *red* at Step S3, causes that all "lower" nodes v in Out(s) are also painted *red* (while all the "upper" nodes in Out(s) have also been painted *red*, due to our algorithm, at some earlier iterations), then $t(v) \geq t(v^*)$, for all v, that is, v^{**} is correctly selected as the minimum-time node v^*. And, third, if after painting the nodes starting from a pivot v^{**}, some lower node v in Out(s) remains unpainted then either the latter node v will paint all the remaining not-yet-painted nodes in Out(s) (and, therefore will be the minimum-time node v^*) at the next step of the "descend" at the same iteration h, or a new unpainted node will be discovered, and the descend will continued until some pivot node v^{**} will paint all lower nodes in Out(s). This proves the lemma. ■

Notice that the procedure Graph-Reduction does not change the occurrence times of the nodes, and in this sense the routing problem in the reduced graph Γ_h is equivalent to the problem Π in the initial graph G.

Similar to [8, 9], the new algorithm operates with two main procedures, Node-Labeling and Graph-Reduction. The Node-Labeling – just as in [8, 9] – has the complexity O(p). After each run of Graph-Reduction at least one node of G will be removed, so the total number of runs of this procedure cannot exceed n, where n is the number of nodes in G, this is a point of departure from the algorithm in [8, 9] where the number of runs of Graph-Reduction required in the worst case is O(p). Thus, we are able to improve the algorithm complexity.

THEOREM 2 *The complexity of the algorithm is $O(np)$.*

Proof We first estimate how many operations each iteration h requires. For each node v_i in Γ_h, the input contains the list of arcs (v_i, v_j) leaving the node. In addition, for each OR-node, v_j, we compute its in-degree r_j.

At Steps S1–S2, the painting procedure includes examining all arcs of positive length in Γ_h and their successive arcs. The required time is at most O(p).

For Step S3 of Node-Painting (in total, during all their invocations at a given iteration h), the labeling procedure can be implemented by examining each arc in Γ_h *not more than once.*

Indeed, during all runs of Step S3 (within a certain fixed iteration), the algorithm will examine in turn each arc, say (v_i, v_j) that leaves a *red*-labeled node v_i (and that has not yet been examined). During each examination, the algorithm treats an OR-node as follows: it updates the value r_j for the head-node v_j decrementing it by 1, and when the new $r_j = 0$, the node v_j is painted *red*, as for an AND-node, the node is painted *red* immediately as soon its predecessor is *red*. Having been once examined at some run of Step S3, the arc (v_i, v_j) is not examined anymore at further runs of this step at the considered iteration. Thus, at any given iteration, during all repetitions of Step S3, the labeling procedure takes $O(p)$ operations. At each iteration, graph Γ_h is reduced by one node so that the total number of iterations is at most n, the number of nodes in the initial graph G. Thus, the overall time of the labeling procedure during all iterations is $O(np)$.

Now we estimate how many operations the reconstruction of the graph requires. It is sufficient to scan only the arcs leading to the immediate successors of the nodes from the set Out(s), and each arc is examined only once, their number is at most $O(p)$. Since the total number of iterations is $O(n)$, the Graph-Reduction requires at most $O(np)$ operations.

When a new AND-node is added to Out(s) or an AND-node changes its position in the ordered Out(s), the order can be maintained in $O(n)$ operations, and even in $O(\log n)$ if we use the balanced-tree data structure (the AVL-trees). This may happen only when an arc is removed from the current graph by procedure Graph-Reduction, *i.e.*, this happens at most p times. Thus, the total cost of maintaining Out(s) to be ordered is not greater than $O(np)$. Therefore, the total complexity of the suggested algorithm is $O(np)$. The claim is proved. ■.

Concluding this section, let us consider an illustrative example presented in Figure 2.

First, it should be noted that the greedy Dinic-type algorithm is unable to solve the instance. Indeed, it is known that the Dinic algorithm can label any AND-node only after all its predecessors are labeled (see [19]); hence, nodes v and u can never be labeled by the latter algorithm. At the same time, as we can easily see, this problem instance is solvable, and our proposed algorithm will solve the problem as follows. This example illustrates that the greedy algorithm cannot properly treat the AND–OR graphs with zero-length cycles.

Initialization St(s) = St(v) = St(u) = St(a) = St(b) = {*uncolored*}, $t(s) = 0$.

Iteration 1 *Steps* S1–S2 are void as there is no positive-length arc $(v_i, v_j), v_i \neq s$, in the instance considered. No node is painted *red*.

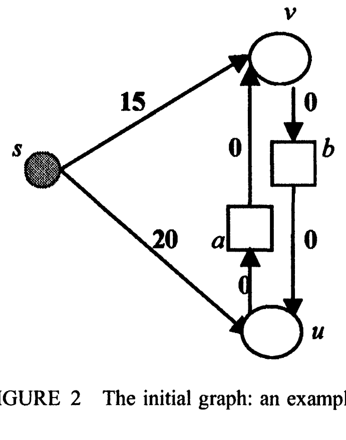

FIGURE 2 The initial graph: an example.

Step S3 [*Choosing a pivot and painting*] Choose the AND-node u as v^{**} since $\tau(s,u) = 20 = \tau^{**} = \max_{t\in \mathrm{Out}(s)} \tau(s, v_i)$, and paint it *red*. Then paint *red* the not-yet-painted nodes, starting from the node u in accord with the rules (i)–(ii) first a, then v, then b. All the nodes are painted *red*, so the node u is chosen as the pivot v^*. Then, the node u, as well as a, v, and b are all painted *black* and assigned the optimal occurrence times $t^*(u) = t^*(a) = t^*(u) = t^*(a) = 20$. The problem is solved.

5 CONCLUDING REMARKS

Our algorithm can find the earliest occurrence times in a more general problem, in which, along with the max-operator (see (2) in Section 2), there is the summation operation defined for some AND-nodes.

$$t(v_j) \geq \sum_{v_i\in P(v_j)} (t(v_t) + \tau(v_i, v_j)) \quad \text{for some } v_j \in A$$

Any superposition of the max-operator and the sum-operator is also allowed, since the superposition function in an AND-node can be decomposed into several new AND-nodes with just max-operators and sum-operators. It is worth of noticing that in the latter case there appears one more reason for introducing zero-length cycles such a cycle appears if an argument $t(v)$ is met several times, in different places, in the superposition function. Furthermore, the proposed algorithm can be easily modified to solve a generalized problem in which the max-operator in AND-nodes is substituted by superior functions (defined in Knuth [24]).

When arc lengths in the AND–OR graph in the considered problem are of arbitrary sign, the proposed algorithm is inapplicable. Although it is easy to construct a dynamic programming algorithm that solves the problem in pseudo-polynomial time [30], the question whether the problem is polynomial solvable, is still open.

Acknowledgement

The partial support from the Israel Ministry of Science and AFIRST, France (grant no MOS/AFIRST 98051-98), and the INTAS Foundation, Brussels, Belgium (grant no 96-0812) for the first and third authors, and from SNI and CONACyT, Mexico, for the second author is gratefully acknowledged. The authors thank E. A. Dinic and V. E. Levit for useful discussions.

References

[1] Adelson-Velsky, G. M. (1973) On some issues of project management, In: Friedman, A. (Ed.), *Studies on Discrete Mathematics.* Moscow: Nauka, pp. 105–134 (Russian).
[2] Adelson-Velsky, G. M. (1995) On design of software systems capable of self-improvement during the expert-machine dialogue, In: Levner, E. (Ed), *Intelligent Scheduling of Robots and Flexible Manufacturing Systems.* Proceedings of the International Workshop, Holon, Israel: CTEH Press, pp. 11–26.
[3] Adelson-Velsky, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovsky, A. A., and Uskov, A. V. (1970) Programming a computer to play chess, *Russian Mathematical Surveys*, **25**, 221–262.
[4] Adelson-Velsky, G. M., Arlazarov, V. L. and Donskoy, M. V. (1975) Some methods of controlling the tree search in chess programs, *Artificial Intelligence*, **6**(4), 361–371.
[5] Adelson-Velsky, G. M., Arlazarov, V. and Donskoy, M. (1977) On the structure of an important class of exhaustive problems and methods of search reduction for them, In: Clarke, M. R. B. (Ed.), *Advances in Computer Chess*, Vol. 1. Edinburgh: Edinburgh University Press, pp. 1–6.

[6] Adelson-Velsky, G. M. Arlazarov, V. L. and Donskoy, M. V. (1988) *Algorithms for Games.* New York: Springer-Verlag.

[7] Adelson-Velsky, G. M., Dinic, E. A. and Karzanov, A. V. (1975) *Flow Algorithms*, Nauka: Moscow, p. 120 (Russian).

[8] Adelson-Velsky, G. M. and Levner, E. (1999) Routing information flows in networks. A generalization of Dijkstra's algorithm. *Proceedings of the International Conference Distributed Computer Communication Networks*, November 9–13, 1999, IPPI RAN Press: Tel-Aviv University, Tel-Aviv–Moscow, pp. 1–4.

[9] Adelson-Velsky, G. M. and Levner, E. (1999) *Finding Extremal Paths in AND-OR Graphs. A Generalization of Dijkstra's Algorithm.* Technical Report, Holon Academic Institute of Technology, Holon, Israel, pp. 35.

[10] Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974) *The Design and Analysis of Computer Algorithms*, Addison-Wesley.

[11] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993) *Network Flows. Theory, Algorithms and Applications*, Englewood Cliffs: Prentice Hall.

[12] Arlazarov, V. L., Dinic, E. A., Kronrod, M. A. and Faradzev, I. A. (1970) On economical construction of the transitive closure of a directed graph, *Soviet Math. Doklady*, **11**(5), 1209–1210.

[13] Ausiello, G., D'Atri, A. and Sacca, D. (1983) Graph algorithms for functional dependency manipulation, *Journal of ACM*, **30**, 752–766.

[14] Crowston, W. (1970) Decision CPM. Network reduction and solution, *Operations Research Quarterly*, **21**(40), 435–444.

[15] De Mello, L. S. H. and Sanderson, A. C. (1990) AND/OR graph representation of assembly plans, *IEEE Transactions on Robotics and Automation*, **6**(2), 188–199.

[16] Dijkstra, E. W. (1959) A note on two problems in connexion with graphs, *Numerische Mathematik*, **1**, 269–271.

[17] Dinic, E. A. (1970) Algorithm for solution of a problem of maximal flow in a network with power estimation, *Soviet Math. Doklady*, **11**, 1277–1280.

[18] Dinic, E. A. (1978) Economical algorithm for finding shortest paths in a network, In: Popkov, Ju. S. and Shmulyan, B. L. (Eds.), *Transportation Modeling Systems*, Moscow: Institute for System Studies, pp. 36–44 (Russian).

[19] Dinic, E. A. (1990) The fastest algorithm for the PERT problems with AND- and OR-nodes, *Proceedings of the Workshop on Combinatorial Optimization*, Waterloo: University of Waterloo Press, pp. 185–187.

[20] Elmaghraby, S. (1964) An algebra for the analysis of generalized activity networks, *Manag. Science*, **10**(3).

[21] Gillies, D. and Liu, J. (1995) Scheduling tasks with AND/OR precedence constraints, *SIAM Journal on Computing*, **24**(4), 787–810.

[22] Goldberg, A. V., Tardos, E. and Tarjan, R. E. (1989) *Network Flows Algorithms*, Technical Report STAN-CS-89-1252, Stanford University.

[23] Karzanov, A. V. (1974) Determining the maximal flow in a network by the methods of preflows, *Soviet Math. Doklady*, **15**, 434–437.

[24] Knuth, D. (1977) A generalization of Dijkstra's algorithm, *Information Processing Letters*, **6**, 1–5.

[25] Levner, E. V. (1969) Optimal planning of processing parts on a number of machines, *Automation and Remote Control*, **12**, 1972–1979.

[26] Levner, E. V. (1973) A network approach to scheduling problems, In: Zipkin, Ya. Z. (Ed.), *Modern Problems of Control Theory.* Moscow: IPU Press, pp. 43–54 (Russian).

[27] Mohring, R. H., Skutella, M. and Stork, F. (2000) *Scheduling with AND/OR Precedence Constraints*, Technical Report No 689/2000, Technische Universitat Berlin, August 2000, p. 26.

[28] Nilsson, N. J. (1980) *Principles of Artificial Intelligence.* Palo Alto: Tioga Publ. Co.

[29] Pinedo, M. (1995) *Scheduling: Theory, Algorithms, Systems*, Prentice Hall.

[30] Schwiegelshohn, U. and Thiele, L. (1999) Dynamic min-max problem, *Discrete Event Dynamic Systems*, **9**, 111–134.

[31] Weisfeiler, B. (Ed.) (1976) *On Construction and Identification of Graphs, Lecture Notes in Mathematics*, Vol. 558, Springer Verlag.

[32] Zwick, U. and Patterson, M. (1996) The complexity of mean payoff games on graphs, *Theoretical Computer Science*, **158**, 343–359.