A Fast Scheduling Algorithm in AND-OR Graphs

GEORGE M. ADELSON-VELSKY
Department of Mathematics and Computer Science,
Bar-Ilan University, Ramat Gan, ISRAEL
velsky@macs.biu.ac.il

ALEXANDER GELBUKH
Center for Computing Research,
National Polytechnic Institute, Mexico City, MEXICO
gelbukh@cic.ipn.mx, www.gelbukh.com

EUGENE LEVNER
Department of Computer Science,
Holon Institute of Technology, Holon, ISRAEL
levner@hait.ac.il

Abstract. We present a polynomial-time algorithm for scheduling tasks in AND-OR graphs. Given the total number p of arcs and the number n of nodes, the complexity of the algorithm is $O(np)$, which is superior to the complexity of previously known algorithms.

Key-Words: AND-OR graphs, scheduling, routing, polynomial-time algorithms.

1 Introduction

In this paper, we consider a task scheduling 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 depicted by an *AND-node* if its execution can be started after all its preceding tasks have been solved; on the other hand, a starting point of a task is depicted by an *OR-node* if it can be started as soon as just 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.

Scheduling problems in AND-OR graphs have many real-world applications. Among many others, we can mention the work by De Mello and Sanderson [5] who have applied the scheduling problems for the planning of robotic assembling systems; Gillies and Liu [7] and Adelson-Velsky and Levner [1] employed AND-OR graphs for real-time scheduling of tasks in computer communication systems. Along with various technological applications, the scheduling problems in AND-OR graphs arise in mathematical analysis of extremal problems in context-free grammars [8], hypergraphs [4], and games [10].

The problem considered generalizes the classical shortest-path and critical-path problems in graphs.

While vast literature is devoted to the shortest-path problem and the critical-path problem in standard graphs (see, e.g., [3], and the numerous references therein), not much have been done for efficiently solving the path-finding problems in AND-OR graphs. A special case in which arc lengths are strictly positive has been elegantly solved by Dinic [6]. Another special case - in which AND-OR graphs are bipartite, arc lengths are non-negative and zero-length cycles are allowed - has been independently investigated by Adelson-Velsky and Levner [1, 2], and Mohring et al. [8], who have proposed different combinatorial algorithms of the same complexity, $O(pp')$, where p is the total number of arcs, and p' the number of arcs entering AND-nodes.

In this paper, we extend the polynomial-time algorithms in [1, 2, 8] to the general (= not-bipartite) AND-OR graphs and improve their complexity. Given the total number n of nodes, the time complexity of the new algorithm is $O(np)$, which is superior to the complexity of the previous algorithms.

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

2 Problem Formulation

The input to the scheduling problem under consideration is $<G, s, \tau>$, where $G = (V, E)$ is a directed graph with nodes of both types, 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 selected node, called the *start*, whose occurrence time is given: $t(s) = t_0$.

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 *starting times* $t(v_j)$, for all $v_j \in V$, satisfying the following conditions:

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

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

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

$$t(v_j) \geq 0, \text{ for all } v_j. \tag{4}$$

Here $P(v)$ denotes the set of nodes - immediate predecessors to v. 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 assembled onto one *super-start*). The problem above is called Problem **P**. The problem turns into the critical path problem if the set O is empty, and the shortest path problem if A is empty.

We start our considerations with the following graph transformations which permit to present the constraints (4) in graph form, and which do not violate the problem size order:

(a) if our graph has an OR-node u whose sons v_j are OR-nodes, then a new AND-node, $\underline{u}$, is added with $\tau(u, \underline{u}) = 0$, while arcs (u, v_j) are replaced by $(\underline{u}, v_j)$, with $\tau(\underline{u}, v_j) = \tau(u, v_j)$, and

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

Due to the preliminary transformations above, we may assume that any OR-node in G has a preceding AND-node; in particular, the G does not contain zero-length cycles consisting of OR-nodes only. Notice that in contrast to the graph model considered in [1, 2], in this paper the arcs entering OR-nodes are allowed to be of non-zero length.

DEFINITIONS. A set of values $\{t(v_j)\}$, $j = 1,\ldots, n$, satisfying inequalities (1)-(4) is called a *feasible solution* to 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, starting times,* and is denoted by $\{t^*(v_j)\}$.

Denote the graph obtained after the transformations by Γ, and the problem of finding the optimal occurrence times $\{t^*(v_j)\}$ in Γ subject to (1)-(4) by **Π**. Obviously, the problems **P** and **Π** are equivalent.

3 New Polynomial-Time Algorithm

A new algorithm is based on the previous algorithm suggested by the first and third authors of this paper in [1,2], differing from the latter in the following two aspects: (i) it uses another labeling procedure which permits us to improve the algorithm complexity; (ii) it uses a more sophisticated graph reduction procedure which permits to treat general AND-OR graphs rather than the bipartite graphs only. The labeling procedure is made bi-colored in order to simplify the presentation of the algorithm.

The basic scheme of the algorithm is the following. At each iteration, the algorithm finds a node with the smallest starting time (at this point, the algorithm is similar to the classic Dijkstra algorithm). However, in contrast to Dijkstra's [3], Knuth's [8] or Dinic's [6] algorithms, our algorithm is *not* greedy: it first discovers and labels all nodes with *not-minimal* current starting times, and only after that it reveals that the remaining (= not-yet-labeled) nodes gain minimum-time labels. At termination, the algorithm either provides the minimal starting time $t^*(v_j)$ for all nodes, or announces that Problem **Π** has no feasible solution.

For every node $v \in V$, the algorithm maintains a time label $t(v)$ and a status $St(v) \in \{$*uncolored*, *red*, *black*$\}$. All nodes start out *uncolored* and later become *red* or *black.* Initially, $t(v) = \infty$ if $v \neq s$; $t(s) = t_0$.

At each iteration, the underlying graph is reduced to a smaller one. Let Γ_h denote the graph derived at the end of the hth iteration ($h = 1, 2,\ldots$). The main idea behind the labeling procedure is to guarantee that in the Γ_h all nodes labeled *red* will have the time labels (= the occurrence times) *greater* than the earliest occurrence time in Γ_h; the nodes labeled *black* will have the same time labels $t(v)$ *equal* to the *earliest* occurrence time among the nodes of Γ_h.

Each iteration uses four sub-procedures: Node_Painting, Node_Selection, Node_Sorting, and Graph_Reduction. Consider a cuurnt iteration, say *h.*

First, Node_Sorting sorts all nodes v_i in **F**(s) in non-decreasing order of their weights $\tau(s,v_i)$.

Next, we use Node_Painting, consisting of three steps, S1-S3. The first step, S1, performs the *initial labeling* of the nodes, after which two other steps are repeatedly carried out one after another: S2, S3, S2, S3, etc. At each step, yet *uncolored* nodes are painted *red*, until at some instant no *uncolored* node can be labeled *red*, neither at Step *S*2 nor at Step *S*3.

Let **F**(s) denote the set of all nodes v which are the heads of the arcs leaving s.

Step S1. Paint *red* each *uncolored* AND-node v_j in the graph Γ_h which is the head of a *positive-length* arc (v_i, v_j) whose tail v_i is *not* the start node s.

Step S2. Paint *red* each *uncolored* OR-node v_j in Γ_h such that <u>*all*</u> immediate predecessors to v_j have been labeled *red* during the previous steps of the current iteration.

Step S3. Assign the status $St(v) = \{red\}$ to each *uncolored* AND-node v_j, such that it has <u>*at least one*</u> immediate predecessor labeled *red* during the previous steps of the current iteration.

Steps S2 and S3 iterate as long as some *uncolored* nodes can be painted *red.* When, at some instant no *uncolored* node can be labeled *red* neither at Step *S*2 nor at Step *S*3, we say that the *first phase* of Node_Painting is finished and the algorithm starts its second phase as will be described below.

Two cases are possible at the termination of the first phase:

Case **C1**. All nodes are painted *red.* This means that the initial graph has a cycle of positive length, and moreover, the time labels of certain AND-node(s) in the cycle will be infinitely large, i.e., unbounded from below. Thus, in this case the problem has no feasible solution.

Case **C2**. Some of the nodes (or, possibly, all of them) cannot be painted *red.*

Two sub-cases are possible:

Case **C2.1**. All the unpainted nodes cannot be reached from *start s* through directed paths in G (clearly, this might happen for OR-nodes only). Then any time label assigned to such OR-nodes will be feasible, and, hence, we will assign the minimally possible time label, $t(s) = t_0$, to all of them.

Case **C2.2**. There are unpainted nodes reachable from *start s* by directed paths in G. Then choose, among them, the node v^{**} in **F**(s) with the maximal "weight": $\tau^{**} = \tau(s,v^{**}) = \tau(s,v_i) = \max_{i\in \mathbf{F}(s)} \tau(s,v_i)$, paint it red, and paint not-yet-painted nodes, by repeatedly using Steps S2 and S3, starting from the chosen node v^{**}. Repeat Steps S2 and S3 as long as some *uncolored* nodes can be painted *red.* When, at some instant no *uncolored* node can be labeled *red* neither at Step *S*2 nor at Step *S*3, we will say that the *second phase* (but not yet the Node_Painting at the considered iteratopm *h*) of the algorithm is finished.

At each iteration, the further behavior of the algorithm depends on whether <u>*all*</u> or <u>*not all*</u> nodes in Γ_h are painted *red.*

If there are nodes not painted *red* at the termination of the second stage, then, again, we choose among them, the node v^{**} in **F**(s) with the maximal "weight": $\tau^{**} = \tau(s,v^{**}) = \tau(s,v_i) = \max_{i\in \mathbf{F}(s)} \tau(s,v_i)$, paint it *red*, and paint *red* not-yet-painted nodes, by repeatedly using Steps S2 and S3, starting from the newly-chosen node v^{**}. Repeat Steps S2 and S3 as long as some *uncolored* nodes can be painted *red.* The second phase is repeated until, at some instant all nodes in **F**(s) are labeled *red.*

The Node_Painting at the current iteration h is finished. Then, Node_Selection selects the last chosen node, and denote it as v^{***}. It has the minimum time in Γ_h and may be deleted from the graph, being saved in a special file, ***FINAL*** $= \{v, t(v)\}$. (This file will be retrieved at the last iteration of the algorithm). The deleted node is painted *black* (this coloring is done in order to simplify the further proofs).

Before the next iteration starts, Graph_Reconstruction is applied: Any arc in Γ_h having a *black* head is deleted. The time labels $t(v)$ for the sons of v^{***} are defined as follows:

$t(v) := \max(t(v), t_0 + \tau(s, v^{***}) + \tau(v_i, v_j))$ if v is an AND-node;

$t(v) := \min(t(v), t_0 + \tau(s, v^{***}) + \tau(v_i, v_j))$ if v is an OR-node.

The arc (v^{***}, v) (with a black *tail* and a red *head*) it is replaced by a new arc, (s, v), of length $t(v)$.

The v^{***} may be either an AND-node or an OR-node. Thus, at each iteration either AND-node or OR-node from the initial graph G is deleted. The number of nodes in F(s) changes dynamically from iteration to iteration, (at some iterations it may increase), but can never exceed the total number of nodes, n.

At each iteration, the number of nodes (and arcs) in the graph Γ_h decreases by at least one. Then the next iteration, consisting of the four sub-procedures described above is applied to the obtained graph Γ_{h+1}. Observe that this construction does not mimic the structure of Dijkstra's shortest path algorithm.

Theorems.

1. *Let* Γ_h *be a graph obtained at the h-th iteration of the algorithm. If a feasible solution to* $\mathbf{\Pi}$ *in graph* Γ_h *exists, then the earliest occurrence time of any black node is equal to* $\tau^* = \tau(s, v_1) = \min_{i\in \mathbf{F}(s)} \tau(s, v_i)$.
2. *If the optimal solution to Problem* $\mathbf{\Pi}$ *exists, then the earliest occurrence times of the red nodes are larger than* $\tau^* = \min_{i\in \mathrm{F}(s)} \tau(s, v_i)$, *where* $\mathbf{F}(s)$ *is the set of all heads of the arcs leaving node s in* Γ_h.
3. *If the optimal solution* $\{t^*(v)\}$ *to Problem* $\mathbf{\Pi}$ *exists, and* $\tau^* = \min_{i\in \mathrm{F}(s)} \tau(s, v_i)$ *at the beginning of an iteration of the algorithm, then, after performing the steps of each iteration, all the nodes with* $t^*(v) > \tau^*$ *will be labeled red.*
4. *The complexity of the algorithm is O(np).*

The proofs immediately follow from the algorithm description above.

The algorithm runs faster if at some stage the reduced graph becomes acyclic, or if all arc lengths in the reduced graph become positive.

Our algorithm is a modification of the algorithm in [1, 2] that solves the above problem (1)-(4). Similar to the algorithm in [1], the new algorithm operates with two main procedures, Node_ Labeling and Graph_Reduction. Node_Labeling - as in [1,2] - has the complexity $O(p)$. However, after each run of Node_Labeling at least one *node* of G may be removed, so the total number of runs of the internal cycle can reach n, where n is the number of nodes in G; this is a point of departure from the algorithm in [1, 2], where $O(p)$ time is required in the internal cycle in the worst case. We improved the complexity to $O(np)$.

Acknowledgements

The work was done under partial support of SNI and CONACyT, Mexico, for the second author.

References

1. Adelson-Velsky, G.M., and E. Levner (1999a). Routing information flows in networks: A generalization of Dijkstra's algorithm, *Proceedings of the International Conference "Distribu-ted Computer Communication Networks"*, November 9-13, 1999, Tel-Aviv University, Israel, 1-4.
2. Adelson-Velsky, G.M., and E. Levner (1999b). *Finding extremal paths in AND-OR graphs.* Technical report, Holon Academic Institute of Technology, Holon, Israel, 35 pp.
3. Ahuja, R.K., T.L. Magnanti, J.B. Orlin (1993). *Network Flows. Theory, Algorithms and Applications*, Prentice Hall, Englewood Cliffs.
4. Ausiello, G., A. D'Atri, D. Sacca (1983), Graph algorithms for functional dependency manipulation, *Journal of ACM*, 30, 752-766.
5. De Mello, L.S.H., and A.C. Sanderson (1990). AND/OR graph representation of assembly plans, *IEEE Transactions on Robotics and Automation,* vol. 6, no.2, 188-199.
6. 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, Waterloo, 185-187.
7. Dijkstra, E.W. (1959). A note on two problems in connexion with graphs, *Numerische Mathematik,* 1, 269-271.
8. Gillies, D. and J. Liu (1995), Scheduling tasks with AND/OR precedence constraints, *SIAM Journal on Computing* 24(4), 787-810.
9. Knuth, D. (1977), A generalization of Dijkstra's algorithm, *Information Processing Letters*, 6,1-5.
10. Mohring, R.H, M. Skutella and F. Stork (2000), *Scheduling with AND/OR Precedence Con straints,* Technical Report No. 689/2000, Technische Universitat Berlin, August 2000, 26 pp.
11. Zwick, U., and M. Patterson (1996), The complexity of mean payoff games on graphs, *Theoretical Computer Science,* 158, 343-359.