ELSEVIER

Discrete Applied Mathematics 87 (1998) 149–158

DISCRETE APPLIED MATHEMATICS

A parametric critical path problem and an application for cyclic scheduling

Eugene Levner [a,*], Vladimir Kats [b]

[a] *Center for Technological Education Holon affiliated with Tel-Aviv University, CTEH, 52 Golomb Street, Holon 58102, Israel*

[b] *Ben Gurion University of the Negev, Beer Sheva 84105, Israel*

Received 14 June 1997; received in revised form 30 January 1998; accepted 16 February 1998

Abstract

The paper addresses a problem of finding critical paths in PERT networks (digraphs) with variable arc lengths depending on a parameter. By equipping the Bellman–Ford label-correcting algorithm with variable vectorial labels depending on the parameter, we derive its version that solves the problem in $O(mn^2)$ time, for all possible parameter values (where m stands for the number of arcs, and n is the number of nodes in the digraph). An application related to cyclic scheduling of tasks in a robotic cell is considered.

Keywords: Networks/graphs, parametric shortest path problem; Analysis of algorithms, distance algorithms, critical-path algorithms; Cyclic scheduling, robotic scheduling

1. Introduction

Let $G=(V,E)$ be a PERT network (finite, not necessarily acyclic, digraph) with n nodes and m arcs. The network describes a project, in which some arcs $e \in A$ are identified with *jobs* and the nodes are identified with *events* (i.e., starting and finishing points of jobs of the project). For each arc $e \in A$, its *length* $w(e)$ is given. Some arc lengths are constants (positive, negative or zero) while some other arc lengths are assumed to be given functions depending on a parameter. Given two fixed nodes s and f, the *parametric critical path* (*CP*) *problem* is to find the longest ('*critical*') finite paths from s to f for *all* possible values of the parameter.

The parametric critical path problem and closely related parametric shortest path problem have received considerable attention in the last few years because of their many applications to several fields such as staff scheduling [3, 6], multifacility location [4], network optimization [13], design of parallel computer architectures [2], robotic

* Corresponding author. E-mail: levner@barley.cteh.ac.il.

0166-218X/98/$19.00
PII S0166-218X(98)00054-7

flowshop scheduling [5, 8, 10], and periodic jobshop scheduling [9]. In view of these applications, development of fast algorithms for the parametric path problems deserves special attention.

In the paper by Karp and Orlin [6], some arc lengths in a graph have a parameter subtracted from them, and the shortest paths are to be found for every parameter value, λ. Two strongly polynomial algorithms are derived, one running in $O(n^3)$ time and producing the desired shortest paths simultaneously for all values of λ, and the second running in $O(nm \log n)$ time and generating the paths for one value of λ at a time as λ increases from $-\infty$. The second algorithm is improved by Young et al. [16] to $O(nm + n^2 \log n)$ time. Another $O(n^3)$-time algorithm developed by Ioachim and Soumis [5] also examines parameter values one after another, but, unlike the second Karp–Orlin algorithm, it carries out the search in the 'opposite direction', that is, it decreases λ step by step, starting from some unfeasible value until λ becomes feasible.

The Karp–Orlin shortest path problem can be generalized naturally to the case in which the length of arc e is taken to be $w(e) = a(e) - b(e)\lambda$, $b(e)$ being non-negative integers. With this modification, essentially the second algorithm by Karp and Orlin [6] is employed, its running time becoming pseudopolynomial, $O(\max_{e \in E} b(e)(nm \log n))$ [1, 15]. Another pseudopolynomial algorithm for this problem using special data structures and running in $O(\max_{e \in E} b(e)(nm + n^2 \log n))$ time, is obtained by Hartmann and Orlin [3]. An interesting pseudopolynomial algorithm is proposed by Ribeiro and Minoux [14] for a more general shortest path problem in which arc lengths $w(e) = a(e) + b(e)\lambda$, where $a(e)$ and $b(e)$ can take on any values provided that graph G contains no negative cycle when $\lambda = 0$. This algorithm uses shortest-paths trees and solves the problem to optimality in $O(\max_e b(e)(nm \log n))$ time.

Our paper is devoted to another generalization of the Karp–Orlin path problem motivated by a real-life application in robotic scheduling. The latter gives rise to the problem version in which the arc lengths are taken to be $w(e) = a(e) \pm \lambda$, where λ is a parameter, and $a(e)$ are any real numbers. Thus, while the Karp–Orlin problem deals with the coefficients $b(e)$ in $w(e) = a(e) + b(e)\lambda$, equal to 0 or -1 only, we will treat the version where those coefficients are 0, -1 or 1.

The main contribution of this paper is the design of a strongly polynomial algorithm for the parametric critical path problem which is of another type than the Karp–Orlin algorithms. While the Karp–Orlin algorithms can be seen to be parametric extensions of the Dijkstra algorithm, we derive a parametric modification of the Bellman–Ford (BF) label-correcting algorithm.

The proposed algorithm for the parametrical CP problem has the following features:

(1) Like the first Karp–Orlin algorithm, it finds the critical paths '*all-at-once*' for *all* possible parameter values (its time complexity being $O(n^2 m)$).
(2) It finds all feasible values of parameter λ, that is, such values λ for which there is no positive-length cycle in the original graph G with respect to arc lengths $w(e) = a(e) \pm \lambda$.
(3) It does not require the knowledge of an initial feasible value of parameter λ.

(4) The algorithm is applied to a cyclic robotic scheduling problem, and improves the best previous running time for the latter problem.

2. The CP problem: some properties

In PERT network (digraph) $G=(V,E)$, a positive arc length denotes the duration of the job identified with the arc. Some other arcs of positive lengths are used to represent *not-before* relations denoting that some event, j, can occur (for example, some job can be started or can be finished) not before than in the given amount of time w_{ij} after some other event i occurs. Similarly, the *not-later* relation denotes that some event, j, is to occur not later than the given time, w_{ij}, before some other event i occurs. The latter relations are represented by arcs of negative length. A zero-length arc just indicates that a job preceeds some other job. Varying lengths, $w(e)=a(e)\pm\lambda$, appear when the operation durations depend on a parameter λ, for example, the calendar time or cycle time (see [11], for further details on the PERT model with negative lengths).

The following well-known properties of the CP problem will be used below.

Property 1. Given $\lambda=\lambda_0$, a finite critical path from s to f in G exists iff, for that λ_0, there is no positive-length cycle in the graph G achievable from s.

Definition. Given two fixed nodes, s and f, a value λ for which a finite critical path from s to f in G exists, is called *feasible*.

Property 2. The domain of all feasible λ values is an interval $\Lambda=[\lambda_{\min},\lambda_{\max}]$ (where the cases $\lambda_{\min}=-\infty$ and/or $\lambda_{\max}=\infty$ are possible), or the empty set.

(The latter property is due to the fact that all problem constraints are linear inequalities.)

Now, we can define exactly *the parametric critical-path (CP) problem* as follows:

The Parametric Critical Path Problem. (1) Given a finite, not necessarily acyclic, digraph $G=(V,E)$, to find interval $\Lambda=[\lambda_{\min},\lambda_{\max}]$ of all feasible values of parameter λ.

(2) The above interval $\Lambda=[\lambda_{\min},\lambda_{\max}]$ of all feasible λ values being known, to find, for each λ in $[\lambda_{\min},\lambda_{\max}]$, a critical path from s to f in graph G and its length.

If, for any real number λ, graph G has a positive cycle achievable from s, the parametric critical path problem is said to have no solution for this graph.

3. The modified Bellman–Ford algorithm

Let $t(e)$ and $h(e)$ denote, respectively, the tail and head of arc $e\in E$. Define for each node $v\in V$, the set of outward arcs $F(v)=\{e\in E\mid t(e)=v\}$, and the set of inward arcs, $B(v)=\{e\in E\mid h(e)=v\}$. We assume that $B(s)=F(f)=\emptyset$. (Otherwise, we add

to G new start and finish nodes, denoted s_0 and f_0, respectively, and zero-length arcs, (s_0, s) and (f, f_0), such that $B(s_0) = F(f_0) = \emptyset$.) Without the loss of generality, we consider the PERT networks without multiple arcs; otherwise, we can eliminate them by inserting an auxiliary node inside each multiple arc.

Let $p^i(v)$ denote the distance label of node v found at the ith pass of the algorithm. While carrying out label corrections in the algorithm below, we will present each variable label as the maximim of several functions of λ: $p(v) = \max(f_1, f_2, \ldots, f_r)$ where all f_i are linear functions of λ. If we have two labels $p_1(v) = \max(f_1, \ldots, f_r)$ and $p_2(v) = \max(g_1, \ldots, g_q)$ then, obviously, $\max(p_1(v), p_2(v)) = \max(f_1, \ldots, f_r, g_1, \ldots, g_q)$. Operation $p(v) + f(\lambda)$, where $f(\lambda)$ is a function of λ (in particular, a constant), is a standard operation of adding two functions. The similar terms, $A + k\lambda$ and $B + k\lambda$ (where A and B are constants) can be reduced: $\max(A + k\lambda, B + k\lambda) = A + k\lambda$, if $A \geqslant B$.

3.1. *The modified Bellman–Ford algorithm*

Step 0: (Initialization). Enumerate all the arcs of E in an arbitrary order.
Assign labels $p^0(s) = 0$, $Pred(s) = \emptyset$ and $p^0(v) = -\infty$ to all other nodes v of G.
Step 1: (Label Correction). **for** $i := 1$ to $n - 1$
do for each arc $e = (t(e), h(e))$ in E
do

$$p^i(h(e)) := \max(p^{i-1}(h(e)), p^{i-1}(t(e)) + w(e)). \tag{1}$$

Fix node-predecessor label $Pred(h(e))$ [details of this operation are explained in Section 6].

Step 2: (Finding All Feasible λ or Displaying '*NO SOLUTION*').
for each arc $e = (t(e), h(e))$ in E solve the functional inequality (2) below with respect to λ:

$$p^{n-1}(t(e)) + w(e) \leqslant p^{n-1}(h(e)). \tag{2}$$

Let Λ be the set of all λ values satisfying the latter inequalities for all $e \in E$.
If $\Lambda \neq \emptyset$ then return Λ and stop.
Otherwise return '*NO SOLUTION*' and stop.

Remark 1. The above algorithm, like the first Karp–Orlin algorithm [6], does not need an initial value λ to be specified in (1). At termination, the algorithm either produces the whole set Λ of all feasible λ, or reveals that $\Lambda = \emptyset$.

Remark 2. The above algorithm can be revised so that not to consider every arc in E during every pass. We can order the arcs by the tail nodes so that all arcs with the same tail node appear consecutively on an arc-list. Thus while scanning arcs we will consider one node at a time, say, node v, with label $p(v) > -\infty$ and scan only the arcs e such that $t(e) = v$. One way to implement this approach is to store only those 'candidate nodes' in a node-list whose labels $p(v)$ change in a pass and examine

this list in the first-in, first out order in the next pass. This approach known to be very effiicient in practice, for any fixed λ value, has, however, the same worst-case complexity as examining all the arcs in any particular order (see [1]).

4. Analysis of the modified BF algorithm

Lemma 1. *At the termination of the algorithm, the final label $p^{n-1}(v)$ for any node v, found at the $(n-1)$st pass, has no more than $2n-1$ components in its vector argument.*

Proof. We will prove by induction on the number of passes, k. For the basis we take $k=1$. We have the following labels of the 0th pass: $p^0(s)=0$, and $p^0(v)=-\infty$ for all other nodes v of V. At the first pass over all arcs of G, any node v adjacent to s achieves a new label: either $p^1(v)=\text{Const.}$, or $p^1(v)=\text{Const.}+\lambda$, or $p^1(v)=\text{Const.}-\lambda$; the labels of all other nodes do not change. Thus, for $k=1$ the required statement holds.

For the inductive step, we assume that after the $(k-1)$th pass, any node label has the following expression:

$$p^{k-1}(u)=\max(C^{(-(k-1))}-(k-1)\lambda,\ldots,C^{(-1)}-\lambda,\\ C^{(0)},C^{(1)}+\lambda,\ldots,C^{(k-1)}+(k-1)\lambda),$$

where $C^{(-(k-1))},\ldots,C^{(-1)},C^{(0)},C^{(1)},\ldots,C^{(k-1)}$ are real numbers. Some of the terms under the max-operator can be absent. At the kth scanning of the arcs of E at Step 1, all distance labels are corrected according to (1), that is, each time when an arc $e=(u,v)$ is examined, the intermediate label $p^k(v)=p^{k-1}(u)+w(e)$ of node v, can become: either $p^k(v)=p^{k-1}(u)+\text{Const.}$, or $p^k(v)=p^{k-1}(u)+(\text{Const.}+\lambda)$, or $p^k(v)=p^{k-1}(u)+(\text{Const.}-\lambda)$.

Clearly, after carrying out the elementary component-wise operations of adding a linear function to the max-function $p^{k-1}(u)$, the length (i.e., the number of arguments under the max-operator) of any of expressions $p^k(v)$ is not longer than $2k+1$. Further, any expression of the form $p^k(v)=\max(p^{k-1}(v),p^k(v))$ also has the length not longer than $2k+1$ because all the similar terms under the max-operator may be reduced, as it was indicated in the previous section. In other words, label $p^k(v)$ of any node v at the kth pass has the following form: $p^k(v)=\max(C^{(-k)}-k\lambda,\ldots,C^{(-1)}-\lambda,C^{(0)},C^{(1)}+\lambda,\ldots,C^{(k)}+k\lambda)$. Taking $k=n-1$, the proof is complete. □

5. Complexity of the algorithm

Theorem 2. *The complexity of the parametric version of the Bellman–Ford algorithm is $O(n^2m)$.*

Proof. Step 0 takes $O(n)$ time. At Step 1, in each of $n-1$ passes over all arcs of graph G, the algorithm corrects labels according to (1). At any of m arcs, this operation consists in adding a linear function (or a constant) $w(e)$ to $(2n-1)$-segment piecewise-linear function $p^{i-1}(t(e))$ and subsequent component-wise comparing the obtained max-function with another max-function, $p^{i-1}(h(e))$, each of them having a vector operand with at most $2n-1$ components (see Lemma); all these operations can be performed, for any arc, in $O(n)$ time. Hence, in order to derive the final algebraic expressions $p^{n-1}(v)$, for all v, at Step 1, it takes at most $O(n^2m)$ time.

At Step 2, the algorithm solves m algebraic inequalities (2). For this purpose, it finds the intersections of each pair of max-functions in (2). Since $p^{n-1}(t(e))+w(e)$ and $p^{n-1}(h(e))$ are piecewise-linear convex functions with at most $O(n)$ segments each, calculating all points λ at which the two functions intersect, can be carried out in $O(n)$ time. Indeed, each of the two functions can be defined on an ordered set of segments with the coordinate of their left ends increasing. In linear time, we can obtain the ordered merged set of all segments from both functions. Passing the latter ordered set of segments from the left to the right, we first find the smallest point λ where the functions intersect.

The basic step consists in further passing the ordered merged set of segments and choosing the segment with the smallest left-end coordinate where the functions intersect again (and the sign of $p^{n-1}(t(e))+w(e)-p^{n-1}(h(e))$ changes). We repeat this step until all the segments of one of the two functions are passed over. Therefore, for any single arc, in linear time we can find all λ for which (2) holds. Thus, $O(nm)$ time is sufficient for solving inequalities (2), for all m arcs at Step 2. Hence, in $O(n^2m)$ time, the algorithm finds the interval $[\lambda_{\min}, \lambda_{\max}]$ and the piecewise-linear function $p^{n-1}(f)$ in node f which determines, for any λ in $[\lambda_{\min}, \lambda_{\max}]$, the length of a critical path from s to f as a function of parameter λ. The same time amount, $O(n^2m)$, is required if $\Lambda=\emptyset$ and the algorithm displays that the problem is inconsistent (that is, it has no solution for any λ). □

Correctness of the modified Bellman–Ford algorithm follows immediately from the fact that, for each fixed value of parameter λ from $\Lambda=[\lambda_{\min}, \lambda_{\max}]$, the algorithm turns into the standard Bellman–Ford algorithm for the problem with numerical arc lengths (see, e.g., [1]).

6. Finding critical paths

A special operation is required if we wish, along with computing the critical-path lengths, to fix the actual nodes of the critical paths as well. The labeling we use for this purpose is a vector extension of the standard Bellman–Ford method of keeping a predecessor node for each node of $V\backslash s$, in the critical path problem with numerical data.

While performing an operation of node labeling according to (1) at the kth pass at Step 1, in our parametric algorithm we set two $(2k+1)$-long vectorial attributes, π and ρ, to each node v of G. The ith component of $\rho(i=-k,\ldots,-1,0,1,\ldots,k)$ is the

number of a component-predecessor for the ith component of vector operand in $p(t(e))$, that is, it indicates which component in the vector operand of $p(t(e))$ has produced $p(h(e))$ after adding $w(e)$ to $p(t(e))$, in the case if the ith component of $p(h(e))$ has been changed after performing the addition. If the ith component of $p(h(e))$ is not changed then the ith component of ρ is not changed as well.

The ith component of $\pi(i=-k,\ldots,-1,0,1,\ldots,k)$ is the number of the node-predecessor, $t(e)$, for the ith component of vector operand in $p(h(e))$, that is, it indicates which node-tail has changed the ith component in the vector operand of $p(h(e))$.

The vector labels $\pi(v)$ and $\rho(v)$ together form a node-predecessor label, $Pred(v)$ which is associated with each node v. For each linear segment of the piecewise-linear function $p(v)$, the node-predecessor label $Pred(v)=(\pi(v),\rho(v))$ indicates a linear segment in distance label $p(u)$ of node u preceding node v in the critical path. Thus, the chain of segments-predecessors originating at node v runs backwards along a critical path from s to v, like a chain of nodes-predecessors in the standard Bellman–Ford algorithm restoring a critical path in the critical path problem with numerical data.

7. An application: a scheduling problem

The following problem is well known in the scheduling literature (see, for instance, [7, 8, 10, 12]).

Given several machines in a line, $M_1,M_2,\ldots,M_N$, where M_1 stands for the input/output storage, identical parts are to be processed successively through the machines in the same order $S=(M_1,M_2,\ldots,M_N)$. A single robot transports the parts between the machines. After arriving at M_i, $2\leqslant i\leqslant N$, a part is to be processed at this machine, for a time interval of no less than L_i and no more than U_i (otherwise the part will be defective), where both L_i and U_i are given constants, and then it must be transported to the next machine. It is assumed that the robot and the machines (except for M_1) can handle only one part at a time. The no-wait constraint is imposed requiring that after a part is processed by a machine it is not allowed to wait and must be immediately transferred by the robot to the next machine in S.

The robot's transporting operations are repeated periodically. Each repetition of the sequence of its moves is called a *cycle*, and its duration (which is equal to the time elapsing between two successive introductions of parts into the system) is called the *cycle time*.

A cycle of robot's route is denoted as $R=([1],[2],\ldots,[N])$, where the kth component of R denotes that, in each cycle, in its kth move in R, the robot moves a part from the machine numbered $[k]$ to machine $[k]+1$, and then runs (without a load) to machine $[k+1]$. Clearly, after servicing the last machine in the cycle, $[N]$, the robot starts its next cycle with the machine numbered $[1]$. Without the loss of generality, we may assume that $[1]=1$.

The following notation and terminology are used to formalize the problem:

d_i = the time required by the (loaded) robot to move a part from machine M_i to M_{i+1},

r_{ij} = the time required by the unloaded robot to run from M_i to M_j, where $2 \leqslant i \leqslant N$, $1 \leqslant j \leqslant N-1$,

t_k = the completion time of the kth operation in the processing sequence S which is performed within the 'first' cycle (i.e., during the interval $[0, T]$), $k = 1, \ldots, N$,

τ_k = the duration of the kth operation in the processing sequence S,

T = the cycle time.

Given the processing sequence S, robot's route R, and real numbers d_k, r_{ij}, L_k and U_k, the scheduling problem is to find the set $\mathbf{T}$ of all feasible cycle time values for which a cyclic schedule (that is, the operation durations τ_k lying within $[L_k, U_k]$ and completion times t_k, $k = 1, \ldots, N$) exists.

The minimal T from $\mathbf{T}$ will provide the minimal cycle time.

Let us denote by $\mathbf{A}$ the set of part-processing operations in S which being started in some robot cycle R are finished in the same cycle, and $\mathbf{B}$ those operations in S which being started in cycle R are finished in the next cycle. Then this cyclic scheduling problem can be reformulated as the following *linear programming* problem:

Problem P. Find all T
subject to:

$$L_k \leqslant \tau_k = t_k - t_{k-1} - d_{k-1} \leqslant U_k, \quad \text{if } k \in \mathbf{A},$$

$$L_k \leqslant \tau_k = T + t_k - t_{k-1} - dk - 1 \leqslant U_k, \quad \text{if } k \in \mathbf{B}$$

[the two-sided constraints on the processing times],

$$t_{[i]} + d_{[i]} + r_{[i]+1,[i+1]} \leqslant t_{[i+1]},$$

$$\text{for all } i = 1, 2, \ldots, N \text{ (where } r_{[i]+1,[N+1]} = r_{[i]+1,1}),$$

[the constraints on the traveling times of the unloaded robot],
and

$$t_1 = 0, \quad t_{[N+1]} = T.$$

We can see now that T plays the role of a parameter in this scheduling problem. We can construct a PERT network G in which there are N nodes corresponding to the completion times of operations from S plus one more node, $[N+1]$, corresponding to $t_{[N+1]} = T$.

Constraints of the form $L_k \leqslant t_k - t_{k-1} - d_{k-1}$ and $L_k \leqslant T + t_k - t_{k-1} - d_{k-1}$ are represented by the arcs leading from node $k-1$ to node k and having the length $L_k + d_{k-1}$ and $L_k + d_{k-1} - T$, respectively. Constraints of the form $t_k - t_{k-1} - d_{k-1} \leqslant U_k$ and $T + t_k - t_{k-1} - d_{k-1} \leqslant U_k$ are represented by the arcs leading from node k to node $k-1$ and having the length $-U_k - d_{k-1}$ and $T - U_k - d_{k-1}$, respectively. Constraints of the form $t_{[i]} + d_{[i]} + r_{[i]+1,[i+1]} \leqslant t_{[i+1]}$ are represented by the arcs leading from node $[i]$ to node $[i+1]$ and having the length $d_{[i]} + r_{[i]+1,[i+1]}$.

Since $T = t_{[N+1]}$, finding the feasible cycle time T is equivalent to finding the critical path length from node 1 to node $[N+1]$ in graph G. Since parameter T plays in this model the same role as parameter λ in the parametric critical path problem studied above, the minimal value of λ obtained as a solution of the critical path problem will be the minimal T we are searching for.

Using the modified Bellman–Ford algorithm described above, we can solve the latter critical path problem in $O(N^3)$ time (since the number of arcs is $O(N)$ in the constructed network). Notice that mathematical programming formulations for the scheduling problem have been known before in the literature (see, e.g., [7, 10, 12]). However, all the previous studies of the problem known to us have resulted in either heuristic or, at best, weakly polynomial algorithms. If problem P has one-sided constraints on the processing times then it can be solved in $O(N^2 \log N)$ time by reducing to the Karp–Orlin parametric path problem with binary $b(e)$ [8, 9].

8. Concluding remarks

The proposed algorithm remains valid and solves to optimality a more general parametric critical path problem with arc lengths $w(e) = a(e) + b(e)\lambda$, where $a(e)$ are any real numbers, and $b(e)$ are any integers. However, the length of the vector label $p(v)$ will be then at most $2b(n-1)+1$, where $b = \max_{e \in E} b(e)$, and, in this case, the algorithm becomes pseudopolynomial. It would be interesting to find other special cases of the parametric critical path problem solvable in polynomial or pseudopolynomial time.

Acknowledgements

The authors thank Boris V. Cherkassky, Evgeny G. Gol'shtein and Arkady S. Nemirovsky for their helpful comments and the anonymous referees for their constructive suggestions.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1993, pp. 164–165.

[2] C. Hanen, A. Munier, Cyclic scheduling on parallel processors: an overview, in: P. Chretienne, E.G. Coffman, Jr., J.K. Lenstra, Z. Liu (Eds.), Scheduling Theory and its Applications, Wiley, Chichester, 1995, pp. 193–226.

[3] M. Hartmann, J.B. Orlin, Finding minimum cost to time ratio cycles with small integral transit times, Networks 23 (1993) 567–574.

[4] T. Ichimori, A shortest path approach to a multifacility minimax location problem with rectilinear distances, J. Oper. Res. Soc. Japan 28 (1985) 269–284.

[5] I. Ioachim, F. Soumis, Schedule efficiency in a robotic production cell, Int. J. Flexible Manufact. Systems 7 (1995) 5–26.

[6] R.M. Karp, J.B. Orlin, Parametric shortest path algorithms with an application to cyclic staffing, Discrete Appl. Math. 3 (1981) 37–45.

[7] V.B. Kats, An exact optimal cyclic scheduling algorithm for multi-operator service of a production line, Automat. Remote Control 43 (1982) 538–542.
[8] V. Kats, E. Levner, Polynomial algorithms for scheduling of robots, in: Proc. Workshop on Intelligent Scheduling of Robots and Flexible Manufacturing Systems, CTEH Press, Holon, Israel, 1996, pp. 77–100.
[9] T.E. Lee, M.E. Posner, Performance measures and schedules in periodic job shops, Oper. Res. 45 (1997) 72–91.
[10] L. Lei, Determining the optimal starting times in a cyclic schedule with a given route, Comput. Oper. Res. 20 (1993) 807–816.
[11] E.V. Levner, A.S. Nemirovsky, A network flow algorithm for just-in-time project scheduling, European J. Oper. Res. 79 (1994) 167–175.
[12] L.W. Phillips, P.S. Unger, Mathematical programming solution of a hoist scheduling problem, AIIE Trans. 2 (1976) 219–225.
[13] G.G. Polak, On a parametric shortest path problem from primal-dual multicommodity network optimization, Networks 22 (1992) 283–295.
[14] C.C. Ribeiro, M. Minoux, A heuristic approach to hard constrained shortest path problems, Discrete Appl. Math. 10 (1985) 125–137.
[15] C. Yang, D. Jin, A primal–dual algorithm for the minimum average weighted length circuit problem, Networks 21 (1991) 705–712.
[16] N.E. Young, R.E. Tarjan, J.B. Orlin, Faster parametric shortest path and minimum-balance algorithms, Networks 21 (1991) 205–221.