SCHEDULING TWO GROUPS OF JOBS WITH INCOMPLETE INFORMATION*

Guochuan ZHANG[1] **Xiaoqiang CAI** **C.K. WONG**[2]

Department of Mathematics, Zhejiang University
Hangzhou 310027, P.R. China
Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Department of Computer Science and Engineering, The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
zgc@math.zju.edu.cn *xqcai@se.cuhk,edu.hk* *wongck@cse.cuhk.edu.hk*

Abstract

In real world situations, most scheduling problems occur neither as complete off-line nor as complete on-line models. Most likely, a problem arises as an on-line model with some partial information. In this article, we consider such a model. We study the scheduling problem $P(n_1,n_2)$, where two groups of jobs are to be scheduled. The first job group is available beforehand. As soon as all jobs in the first group are assigned, the second job group appears. The objective is to minimize the longest job completion time (makespan). We show a lower bound of 3/2 even for very special cases. Best possible algorithms are presented for a number of cases. Furthermore, a heuristic is proposed for the general case. The main contribution of this paper is to discuss the impact of the quantity of available information in designing an on-line algorithm. It is interesting to note that the absence of even a little bit information may significantly affect the performance of an algorithm.

Keywords: Machine scheduling, worst-case analysis, on-line algorithm

1. Introduction

In classical scheduling models, it is assumed that full information on the jobs to be scheduled is available in advance. Such a situation is termed off-line scheduling. In many applications, however, this is not realistic because decisions have to be made before information on the jobs is known. In contrast to off-line scheduling, such a situation is termed on-line scheduling. In most

* Research partially supported by a Hong Kong Government RGC Earmarked Grant. Ref. No. CUHK356/96E

1.Part of this work was done while visiting The Chinese University of Hong Kong. Research partially supported by National 973 Fundamental Research Project of China and National Natural Science Foundation of China (19801032)

2On leave from IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A on-line scheduling problems, it is assumed that jobs become available one by one. The information on the next job is not known until the current job is assigned. As soon as a job has been assigned, it cannot be moved again. For a survey of the results of various on-line paradigms reported in the literature, we refer to Sgall（1998）.

In this article, we consider a scheduling model with two groups of jobs, where information on the jobs in the first group is known whereas information on those jobs in the second group is not available beforehand. Specifically, the problem can be stated as follows: A number of jobs are to be processed by m identical parallel machines. Each job needs the processing of one and only one machine. The decision to assign (dispatch) a job to a machine is irrevocable, i.e., as soon as such a decision is made, the job will be dispatched and the decision cannot be changed any more. The jobs can be classified into two groups. The particulars of the first group of jobs are completely known to the scheduler and jobs can be assigned immediately. The second group of jobs, however, is not available (or released) until all jobs in the first group have been assigned. The objective is to schedule all jobs such that the maximum completion time (makespan) is minimized. For brevity, we use L_1 and L_2 to denote the first and second groups of jobs, respectively. Furthermore, we denote the model described above by $P(n_1, n_2)$, where n_1 and n_2 are the numbers of jobs in L_1 and L_2, respectively. Assume that n_1 and n_2 are known beforehand.

The $P(n_1, n_2)$ model is a generalization of the traditional model of off-line scheduling. If n_1=0 or n_2=0, $P(n_1, n_2)$ reduces to the traditional off-line scheduling problem of minimizing the makespan on a number of parallel machines (see, e.g., Gary and Johnson 1978). If the first group of jobs has been scheduled and the goal is to optimally schedule the second group of jobs, the problem becomes the one considered by Lee (1991), named parallel machines scheduling with nonsimultaneous machine available time. On the other hand, the $P(n_1, n_2)$ model is also a new variant of on-line scheduling over a list (see, for example, Sgall 1998), where jobs are ordered in some list (of which the scheduler is not aware) and are released one by one. The $P(n_1, n_2)$ model addresses the situations where some jobs are given but some are unknown. It can thus be viewed as a model in between off-line and on-line scheduling,

Noting the NP-hardness results that have been established for the off-line scheduling problem with parallel machines (see Gary and Johnson 1978, 1979), we know that $P(n_1, n_2)$ is NP-*hard in the strong sense* in general (Gary and Johnson 1978), and NP-*hard in the ordinary sense* in the two-machine case (Gary and Johnson 1979). Furthermore, a pseudo-polynomial algorithm can be derived if the number of machines is fixed. Many heuristic results have been developed. Among them, the most popular approach is *list scheduling*. In a *list scheduling* algorithm, jobs are placed into a list, often in an arbitrary order. The algorithm always schedules the first available job on the list of unscheduled jobs whenever a machine is idle. If a list scheduling algorithm works with an arbitrary list of jobs, we name this algorithm LS. Since LS requires no knowledge of active and future jobs. It is obviously an on-line algorithm. If processing times of the jobs are known and the job list is sorted in non-increasing order of processing times, then such a list scheduling algorithm is known as LPT (Longest Processing Time job first), which works much better than LS. Clearly, LPT is an off-line algorithm since it sorts all jobs in some order, based on the known information, before scheduling them. In the next section we will further discuss the two algorithms LS and LPT and the possibility of applying them to our problem $P(n_1, n_2)$.

The rest of this paper is organized as follows. Section 2 analyzes LPT-class algorithms, which are shown no better than LS algorithm in terms of worst-case performance when applied to our problem. Section 3 provides a lower bound 3/2 for the problem. This lower bound remains true even for some special cases. In Section 4, we design and analyze on-line algorithms for these special cases, respectively. All algorithms have worst-case ratios matching the lower bound, and therefore they are the best possible. Section 5 investigates the general case. Several strategies are analyzed. Conclusions are given in Section 6.

2. Notation and Preliminary Analysis

Let $C^*(L)$ and $C_A(L)$ denote, respectively, the makespan given by an optimal algorithm and the makespan produced by an approximation algorithm A for an input job list L. The *worst-case* ratio ρ_A of algorithm A is defined as

$$\rho_A = \sup_L \{C_A(L)/C^*(L)\} \tag{1}$$

Clearly, $\rho_A \geq 1$. The worst-case ratio is the usual measure for the quality of an approximation algorithm for scheduling problems. The smaller the ratio, the better the approximation algorithm.

To measure the quality of on-line algorithms, we usually adopt the so-called *competitive ratio*, where $C^*(L)$ is provided by an off-line optimal algorithm and $C_A(L)$ is given by an on-line algorithm A. In this paper, for convenience we use the term “worst-case ratio” for both off-line and on-line algorithms. For simplicity, in the following instead of $C^*(L)$ and $C_A(L)$, we usually use C^* and C_A to denote the corresponding makespans without causing any confusion.

In the first paper on the worst-case analysis of scheduling heuristics, Graham (1966) showed that algorithm LS has a worst-case ratio of $2-1/m$, where m is the number of machines. Faigle, Kern and Turan (1989) proved that for $m=2$ and 3, LS is the best possible; In other words, no on-line algorithms exist such that their worst-case ratios are less than $2-1/m$ for $m=2$ and 3. For $m\geq 4$, Chen, Van Vliet and Woeginger (1994) provided algorithms with better worst-case ratios than LS. However, similar to LS, their algorithms still approach 2 as m increases.

Now we turn to the off-line algorithm LPT. LPT was shown by Graham (1969) to have a worst-case ratio of $4/3-1/(3m)$. For more information on parallel machine scheduling, one can refer to a recent survey paper by Chen, Potts and Woeginger (1998). Since our problem $P(n_1, n_2)$ has two groups of jobs, one may suggest to apply the algorithm LPT twice. Such a heuristic, denoted by *Two-phrase* LPT, first applies LPT to the first job group; afterwards applies LPT again to the second job group. It may be expected that the *Two-phrase* LPT could perform like the algorithm LPT. However, the following instance shows that it is not better than

LS in terms of worst-case ratio, and consequently some other approaches should be sought to tackle the problem $P(n_1, n_2)$.

Assume that in an instance of $P(n_1, n_2)$ the first job group consists of $m(m-1)$ identical jobs, each with processing time 1 and the second job group only contains one long job with processing time m. Obviously, the makespan given by *Two-phrase* LPT is $2m-1$ while the optimal makespan is m. It implies that the worst-case ratio of *Two-phrase* LPT is not better than $2-1/m$, the same worst-case ratio as that of LS. This worst-case result even holds when L_2 contains only one job. In other words, LPT is of no help even for the problem $P(n_1, 1)$. We realize that we cannot simply apply LPT to the first job group (or try to assign the first job group optimally). To design better algorithms, we must save some space for possible jobs with long processing times appearing in the second job group, i.e., at least one machine should be reserved in some way to accommodate possible long jobs in the second job group. This will be the main idea to design the algorithms proposed in the sections below.

3. A Lower Bound

Throughout the rest of this paper we always assume that $n_1 \geq 2$, $n_2 \geq 1$ and $n_1+n_2 \geq m+1$ (m is the number of machines). Obviously, if $n_1 \leq 1$ or $n_2=0$, the problem $P(n_1, n_2)$ is equivalent to the off-line problem; if $n_1+n_2 \leq m$, we can arrange each job to a single machine which results in an optimal schedule.

Lemma 1 *For any given positive integer $n_1 \geqslant 2$ and $n_2 \geqslant 1$, where $n_1+n_2 \geqslant m+1$ (m is the number of machines), there does not exist any on-line algorithm with worst-case ratio less than 3/2 for problem $P(n_1, n_2)$.*

Proof. Let ε be an arbitrarily small positive number. Consider any heuristic H with the following instances. The first job group L_1 only contains two kinds of jobs: One is called long jobs, each with processing time 1 while the other is called short jobs, each with processing time ε. If $n_1 \leq m$, let L_1 contain n_1 long jobs; otherwise, let L_1 have m long jobs and n_1-m short jobs. In the case where two long jobs are put together to a machine by H, the next job group L_2 becomes available and it only consists of n_2 short jobs, each with processing time ε. Then we come to the following result: the makespan produced by H is at least 2 while the optimal makespan is close to 1. If H assigns long jobs to different machines, L_2 contains k huge jobs, each with processing time 2 and n_2-k short jobs. If $n_1 \leq m$, $k=m+1-n_1$; otherwise, $k=1$. In this case, H produces a schedule with a makespan no better than 3 while the optimal value is close to 2. It implies that the worst-case ratio of any heuristic H is at least 3/2. ■

Note that Lemma 1 holds for any fixed n_1 and n_2 as long as $n_1 \geq 2$, $n_2 \geq 1$ and $n_1+n_2 \geq m+1$. In particular, the following corollaries hold. Recall that in problem $P(n_1,1)$, it is known to the scheduler that the second job group consists of only one job, but the processing time of such a job is not known before the first job group has been assigned.

Corollary 2 *There does not exist any on-line algorithm with worst-case ratio less than 3/2 for problem $P(n_1,1)$, where $n_1 \geqslant m$.*

Recall that for the off-line problem, a pseudo-polynomial algorithm exists for any fixed number of machines. However, it follows from Corollary 2 that, if the information on the last job (only one job!) is unknown beforehand, no on-line algorithm can produce a makespan less than 3/2 times that of the optimum makespan in terms of worst-case performance.

Corollary 3 *There does not exist any on-line algorithm with worst-case ratio less than 3/2 for the problem $P(n_1, n_2)$. If* $2\le n_1\le m$ and $n_1+n_2\ge m+1$.

Note that $P(n_1, n_2)$ is equivalent to the off-line problem, if L_1 has only one job. But if L_1 has one more job, i.e., $n_1=2$, the worst-case performance of an on-line algorithm becomes much worse. From the two corollaries, we realize that in terms of worst-case performance, the on-line problems with partial information can differ significantly from the off-line problems even if the partial information available is almost complete.

It is well known that algorithm LS has a worst-case ratio 3/2 for $m=2$. We thus get that for $m=2$, LS is still a best possible algorithm for our problem. Throughout the rest of this paper, we always assume that $m\ge 3$. Denote jobs by J_1, $J_2,\dots, J_n$ and their respective processing times by $p_1, p_2,\dots, p_n$, where n is the total number of jobs in two groups, i.e., $n=n_1+n_2$.

In the next section we will give, respectively, two on-line algorithms for problems $P(n_1,1)$ and $P(n_1,n_2)$ $(2\le n_1\le m)$ and show that their worst-case ratios are exactly 3/2.

4. Special Cases

We first consider the special case $P(n_1,1)$. As discussed above, at least one machine (denoted as *a waiting machine*) should be lightly loaded for L_1.

Algorithm LPT$_W$ (LPT with a waiting machine)

Step 1. Reindex the jobs of L_1 in non-increasing order of processing times. If $n_1\le m$, assign all jobs each in a machine; otherwise, assign the first m jobs each in a machine.

Step 2. Let M_i be the current workload of Machine $i(i=1,\dots, m)$, where $M_1\ge M_2\ge \cdots \ge M_m$. If there are no unscheduled jobs of L_1, go to Step 4.

Step 3. Apply LS to the remaining jobs in L_1 to Machines $1,\dots, m-1$. The m-th machine is set to be a waiting machine.

Step 4. Assign the job of L_2 to Machine m.

Theorem 4 *The worst-case ratio of algorithm* LPT$_W$ *is not greater than 3/2. Moreover, it is a best possible on-line algorithm.*

Proof. We prove this theorem in the following two cases.

Case 1. The makespan C_{LPTW} is determined by a job of L_1. Without loss of generality, assume that the completion time of the last job J_{n_1} of L_1 is C_{LPTW}. Let s be the starting time of J_{n_1}. If $s=0$, then $C_{LPTW}= p_{n_1}$ and algorithm LPT$_W$ provides an optimal makespan. Now assume that $s>0$, thus $C^*\ge 2p_{n_1}$ due to the LPT rule. If $p_{n_1}>s/2$, then on the same machine, there is just one job scheduled before J_{n_1} due to the LPT rule. Hence

$$\frac{C_{LPTw}}{C^*}\le\frac{s+p_{n_1}}{2p_{n_1}}<3/2$$

Consider the case $p_{n_1}\le s/2$. Note that the total workload of Machines $1,\dots,m-1$, is not less than $(m-1)s+p_{n_1}$ and the workload of Machine m is p_m+p_n where p_n is the processing time of the job J_{n_1} in L_2. We have

$$C^*\ge((m-1)s+p_{n_1}+p_m+p_n)/m$$
$$\ge(m-1)s/m+2p_{n_1}/m$$

and

$$C_{LPTw}=s+p_{n_1}$$

Note that $m \geq 3$.

$$\frac{C_{LPTw}}{C^*} \leq \frac{m(s+p_{n_1})}{(m-1)s+2p_{n_1}} \leq 3/2$$

Case 2. The makespan $C_{LPT\mathrm{W}}$ is determined by the job J_n of L_2, i.e., $C_{LPT\mathrm{W}} = p_m + p_n$. Note that $C^* \geq p_n$. If $p_n \geq 2p_m$, then

$$\frac{C_{LPTw}}{C^*} \leq \frac{p_m + p_n}{p_n} \leq 3/2$$

Now we assume that $p_n < 2p_m$. In an optimal schedule, two of the jobs among $\{J_1, J_2, \ldots, J_m, J_n\}$ will be assigned on the same machine. Then $C^* \geq p_m + p_n$ or $C^* \geq 2p_m$.

Observe that $(p_m+p_n)/(2p_m) < 3/2$. Thus we get $C_{LPT\mathrm{W}}/C^* \leq 3/2$.

From Corollary 2, we have shown that $\mathrm{LPT_W}$ is a best possible algorithm. ■

In algorithm $\mathrm{LPT_W}$, we set the machine which has accepted job J_m to be the waiting machine. To see this is crucial, we give two remarks as follows.

Remark 1. If we set an empty machine as the waiting machine, then the algorithm will have a worst-case ratio of 2. This bound is reached by Instance (L_1, L_2) where L_1 consists of m jobs, each with processing time 1 and L_2 has only one job with processing time ε (an arbitrarily small positive number).

Remark 2. If we set the machine which has accepted job J_i ($i < m$) to be the waiting machine, then the algorithm will have a worst-case ratio not less than $2m/(m+1)$. This bound is reached by Instance (L_1, L_2), where L_1 consists of $m-1$ jobs, each with processing time 1 and m jobs, each with processing time $1/m$, where L_2 has only one job with processing time 1.

We now turn to another special case $P(n_1, n_2)$ where $2 \leq n_1 \leq m$.

Theorem 5 *The worst-case ratio of algorithm Two-phrase LPT for $P(n_1, n_2)(2 \leq n_1 \leq m)$ is not greater than 3/2. Thus it is a best possible on-line algorithm.*

Proof. Let $C_{\max}$ denote the makespan produced by algorithm *Two-phrase* LPT. Clearly, after L_1 is assigned, each machine has at most one job to be scheduled. Without loss of generality, the completion time of the last job J_n (of L_2) is $C_{\max}$. Let s be the starting time of J_n. Then $C_{\max} = s + p_n$. Note that $C^* \geq p_n$ and $C^* \geq s + p_n/m$. When $p_n \geq 2s$ or $s \geq 2p_n$, it follows that $C_{\max}/C^* \leq 3/2$. In the following we only consider the case where $s/2 < p_n < 2s$. It implies that some other jobs are assigned to the same machine as J_n is. From LPT rule, among L_2, J_n is one of the shortest jobs. Assume that J_n is assigned to Machine i.

Case 1. On Machine i, a job from L_2 is scheduled before J_n. Denote this job by J_i. Immediately before J_n is scheduled, each machine contains either a job from L_2 or a job of L_1 with processing time not less than p_i. Note that $p_i \geq p_n$. There are at least $m+1$ jobs with processing time not less than p_n. Thus $C^* \geq 2p_n$, Since $s < 2p_n$, $C_{\max}/C^* \leq 3/2$ holds.

Case 2. On Machine i, a job from L_1 is scheduled before J_n. Such a job is denoted by J_j. Clearly $p_j = s$. Before J_n is assigned, every machine must have a job from L_1 with a processing time not less than s or from L_2 with a processing time not less than p_n. Optimally scheduling those jobs will result in a makespan $C^* \geq 2s$ or $C^* \geq 2\,p_n$. Note that $(s+p_n)/2s < 3/2$ and $(s+p_n)/(2p_n) < 3/2$. Thus $C_{\max}/C^* < 3/2$.

From Corollary 3, we conclude that Two-phrase LPT is a best possible algorithm. ■

5. The General Case

From Lemma 1, algorithm LS is the best possible for m=2. In this section, we will only consider the case $m \geq 3$ and extend algorithm LPTw to the general problem $P(n_1, n_2)$.

Algorithm GLPTw

Step 1. Reindex the jobs of L_1 in non-increasing order of processing times. If $n_1 \leq m$, assign all jobs each in a machine; otherwise, assign the first m jobs each in a machine.

Step 2. Let M_i be the current workload of Machine i (i=1,…, m) where $M_1 \geq M_2 \geq \cdots \geq M_m$. If there are no unscheduled jobs of L_1, go to Step 4.

Step 3. Apply LS to the remaining jobs in L_1 on Machines 1,…, $m-1$. The m-th machine is set to be a waiting machine.

Step 4. Apply LPT to job group L_2.

Theorem 6 *For problem $P(n_1, n_2)$, the worst-case ratio of algorithm GLPTw is $2-1/(m-1)$ for $m \geq 3$.*

Proof. Omitted. ■

Corollary 7 *Algorithm GLPTw is a best possible on-line algorithm for m=3.*

Proof. It follows from the fact that the worst-case ratio of GLPTw is 3/2 for m=3. ■

Note that for $m \geq 4$, algorithm GLPTw does not work. The point is that only one waiting machine is set. However if we set more waiting machines to accept few jobs, the worst-case ratio can not be improved. It can be observed by considering the following instance: Group L_1 consists of a large number of tiny jobs. Suppose that the number of waiting machines is k. Then L_2 comes with k+1 long jobs. One can select appropriate job length to get a worst-case bound. From the above observation, we realize that the crucial point is how to "aptly" schedule the first job group. We have to keep the largest difference of the workloads among the machines in an appropriate range. For the second job group we can always apply LPT. Below may be a possible way to get a better algorithm. Let α be a parameter.

Find a near optimal makespan C_1 for group L_1.

Let $q = C_1$.

Consider the following multiple Knapsack problem: Given m knapsacks with capacity of $C_1(1+\alpha)$ and a set of items $L_1 \cup L_0$, where L_0 is a set of m identical dummy items with size q. Find near optimal solution for this problem.

If all items can be packed into those knapsacks, remove those dummy items and arrange the remaining items (jobs) of each knapsack to a machine; otherwise decrease the size g of dummy items and back to 3.

Based on the schedule for L_1, apply LPT to L_2.

Finally we guess that there exists an on-line algorithm with a worst-case ratio 3/2 for the general case.

6. Concluding Remarks

In this article, we discussed the impact of the quantity of available information in designing an on-line algorithm. It is interesting to note that the absence of even a little bit information may significantly affect the performance of an algorithm. We studied the scheduling problem $P(n_1, n_2)$, where two groups of jobs are to be scheduled. The first job group is available beforehand. As soon as all jobs in the first group are assigned, the second job group appears. A lower bound 3/2 was given even for two special cases $P(n_1,1)$ and $P(n_1, n_2)(2\leq n_1\leq m)$. Two on-line algorithms LPTw and *Two-phrase* LPT were provided and were shown to match the lower bound for the two special cases, respectively. Finally, the general problem was considered.

References

[1] Chen, B., A. Van Vliet and G.J. Woeginger, "New lower and upper bounds for on-line scheduling", *Operations Research Lette*rs, Vol. 16, pp221-230, 1994.

[2] Chen, B., C.N. Potts and G.J. Woeginger, "A review of machine scheduling: Complexity, algorithms and approximability", D.Z. Du and P. Pardalos, *Handbook of Combinatorial Optimization*, Kluwer Academic Publishers, 1998.

[3] Faigle, U., W. Kern and G. Turan, "On the performance of on-line algorithms for partition problems" , *Acta Cybernetica*, Vol. 9, pp107-119, 1989

[4] Graham, R.L., "Bounds for certain multiprocessing anomalies", *Bell System Technical J.*, Vol. 45, pp1563-1581, 1966.

[5] Graham, R.L., "Bounds on multiprocessing timing anomalies", *SIAM Journal on Applied Mathematics*, Vol. 17, pp416-429, 1969.

[6] Gary, M.R., and D.S. Johnson, "Strong NP-completeness results: motivation, examples and implications", *Journal of ACM*, Vol. 25, pp499-508, 1978.

[7] Gary, M.R., and D.S. Johnson, *Computers and Intracability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[8] Lee, C.Y., "Parallel machines scheduling with nonsimultaneous machine available time", *Discrete Applied Mathematics*, Vol. 30, pp53-61, 1991.

[9] Sgall, J., "On-line scheduling", In A. Fiat and G, J. Woeginger, Online Algorithms - The state of the art, *Lecture Notes in Computer Science*, Vol. 1442, pp196-231, 1998.

Guochuan Zhang is Professor of Department of Mathematics, Zhejiang University, China. He received his Ph.D. in Operations Research from Academia Sinica (Beijing) in 1995, and was awarded Alexander-von-Humboldt Research Fellowship in 2000. His current research interests are in on-line algorithms and approximation algorithms for hard optimization problems. He has published papers in journals such as Naval Research logistics, Operations Research Letters, Discrete Applied Mathematics, etc.

Xiaoqiang Cai is Professor and Chairman of the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. He received his Ph.D from Tsinghua University, Beijing, in 1988. His current research interests include scheduling models and applications, time-varying network optimization, and portfolio optimization. He has published over 50 papers in journals such as Management Science, Operations Research, Naval Research Logistics, IIE Transactions, IEEE Transactions, Discrete Applied Mathematics, European Journal of Operational Research, etc. He is on the editorial board of IIE Transactions on Scheduling and Logistics, Journal of Scheduling, and several other journals.

C.K. Wong is Professor of Computer Science and Engineering at The Chinese University of Hong Kong and served as Chairman of the Department of Computer Science and Engineering from 1995 – 1997. He received the B.A. degree (First Class Honors) in mathematics from University of Hong Kong in 1965, and the M.A. and Ph.D. degrees in mathematics from Columbia University in 1966 and 1970, respectively. He is a Fellow of the Institute of Electrical and Electronic Engineers (IEEE) and a Fellow of the Association for Computing Machinery (ACM). He had ever been Chair of the IEEE Computer Society Technical Committee, Editor of "IEEE Transactions on Computers", Associate Editor of "IEEE Transactions on VLSI Systems, and an Editorial Board Member of the international journal "Fuzzy Sets and Systems" and "Networks". He is also the founding Editor-in-Chief of the international journal "Algorithmica" and an Advisory Board Member of "ACM Journal of Experimental Algorithmics". His current research is mainly focused on algorithms, in particular, VLSI design algorithms, Steiner tree problems in non-Euclidean metrics and scheduling problems in various settings. He holds four U.S. patents and has published more than 200 papers and two books. He received an Outstanding Invention Award, an Outstanding Technical Achievement Award and four Invention Achievement Awards from IBM. At the 1995 IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD' 95), he received a best paper award for his work on FPGA design.