Finding the best root node strategy for the approximation of the time-indexed bound in min-sum scheduling

Yunpeng Pan and Leyuan Shi

***Abstract*— We identify the best root node strategy for the approximation of the time-indexed bound in min-sum scheduling by sorting through various options that involve the primal simplex, dual simplex, and barrier methods for linear programming, the network simplex method for network flow problems, and Dantzig-Wolfe decomposition and column generation.**

***Note to Practitioners*— Min-sum scheduling problems are frequently encountered in manufacturing or service environments where job/entity-based accounting metrics are of prime interest. Examples of such metrics are the average earliness-tardiness of jobs through a plant and the average flow/residence time of patients through a health care facility. Recently, a new and fast method for computing lower bounds for such problems has been introduced. When used in branch-and-bound, the lower bound method has the characteristic of requiring large initial computational expense at the root node and quite marginal subsequent expenses at the other nodes. This experimental work aims to address the question of how to reduce the large initial computational expense.**

***Index Terms*— min-sum scheduling; MMT; time-indexing; simplex; barrier; column generation**

I. Introduction

The paper is concerned with finding provably optimal solutions to nonpreemptive min-sum scheduling problems. Examples are the total weighted tardiness problem [1], [5], [14], the total weighted completion time problem [3], [4], [13], and the total weighted earliness-tardiness problem [15], [16].

All known strong bounds are based on time discretization, which implies integrality of processing time and pseudopolynomial problem size. The strongest of those bounds is one provided by the linear programming (LP) relaxation of the time-indexed integer programming formulation of scheduling problems [8], [17], [19]. We refer to this bound as the *TI bound* hereafter. (In [19] polyhedral techniques are proposed to strengthen the bound, but it is beyond the scope of this paper.) However, the TI bound is extremely time-consuming to compute for $n > 30$ and $p_{\max} > 10$, where n and $p_{\max}$ stand for the number of jobs and the maximum processing time, respectively. Even though advanced solution information is used to warm start the solution of LP during branch-and-bound, algorithms using the TI bound have very large running time.

This apparent drawback appears to have motivated the work in [3], [16], both proposing to mend the situation using Lagrangian relaxation. Meanwhile, another novel lower bound called the *max-min transportation (MMT) lower bound* recently has been introduced by Pan and Shi [14]. The bound has been shown to possess some very nice properties. For the root node, the MMT bound is equal to the TI bound. For subsequent nodes, the MMT bound is a fast approximation of the TI bound; more precisely, it entails the solution of a transportation problem, as opposed to the time-consuming LP problem. By considering the total weighted tardiness problem ($1||\sum w_jT_j$), Pan and Shi [14] established experimentally that the approximation is not only fast, but also able to retain the superior quality of the TI bound to a large extent. They showed that a branch-and-bound algorithm using the MMT bound on average generate 2–4 times more nodes than one using the TI bound, while the required time per lower bound evaluation is reduced to below 1%. Consequently, all benchmark problems from OR-Library (`http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html`) with as many as 100 jobs were solved to optimality for the first time.

The motivation behind the present work is the fact that on average, the computation of MMT at the root node is much more time-consuming than the computation at the rest of the nodes. As indicated by the computational experience reported in [14] on the total weighted tardiness problem, it is 120, 68, and 3 times more expensive than the remainder of the total computation time for $p_{\max} = 100$ and $n = 40, 50, 100$, respectively. Therefore, the reduction of root node time will have a visible impact on the overall running time of the branch-and-bound algorithm. To this end, let us first explain the steps entailed by the root node computation.

II. The MMT bound

In the nonpreemptive min-sum scheduling problem, we are given a set of n jobs to be processed on a single machine that can only process one job at a time. Job j ($j = 1, \ldots, n$) takes time p_j to complete. Upon its completion at time t_j, job j incurs a cost of $f_j(t_j)$, where $f_j(\cdot)$ is an arbitrary function. The objective is to find a processing sequence for these jobs such that the total cost, $\sum_{j=1}^{n} f_j(t_j)$, is minimized. For convenience, we let $P = \sum_{j=1}^{n} p_j$. Assume that jobs $1, \ldots, n$ are to be processed during time window $[0, T]$. We discretize the time window and refer to the time interval $[t, t+1)$ as time period t; hence, $t = 0, \ldots, T-1$.

With an appropriate set of transportation costs, the following transportation problem is a relaxation to the scheduling problem:

$$\begin{aligned} (\mathrm{P}(c)) \quad \min \quad & \sum_{j=1}^{n} \sum_{t=0}^{T-1} c_{jt} x_{jt} \\ \text{s.t.} \quad & \sum_{t=0}^{T-1} x_{jt} = p_j, \ \forall j = 1, \ldots, n, \end{aligned} \tag{1}$$

Dr. Yunpeng Pan is now with CombineNet, Inc., Fifteen 27th Street, Pittsburgh, PA 15222 USA (email: ypan@combinenet.com). Prof. Leyuan Shi is with the ISyE department, University of Wisconsin-Madison, Madison, WI 53706 USA, and also with the Center for Intelligent and Networked Systems, Tsinghua University, Beijing, P. R. China (e-mail: leyuan@engr.wisc.edu).

$$\sum_{j=1}^{n} x_{jt} \le 1,\ \forall t = 0, \ldots, T-1, \tag{2}$$
$$x_{jt} \ge 0,\ \forall j = 1, \ldots, n; t = 0, \ldots, T-1,$$

where $x_{jt} = 1$ if job j is assigned to period t, and $x_{jt} = 0$ otherwise; c_{jt} is the transportation cost. Let $LB(c)$ denote the optimal value of the minimization problem. For completeness, we also write down the dual problem:

$$\begin{aligned}(\text{DP}(c)) \quad \max \quad & \sum_{j=1}^{n} p_j u_j + \sum_{t=0}^{T-1} v_t \\ \text{s.t.} \quad & u_j + v_t \le c_{jt},\ \forall j{=}1, \ldots, n; t{=}0, \ldots, T-1, \\ & u_j \text{ free}, \forall j = 1, \ldots, n, \\ & v_t \le 0, \forall t = 0, \ldots, T-1.\end{aligned}$$

This type of relaxation was first introduced by Lawler [10] and was also adopted in [6], [8], [9], [15]. One transportation problem distinguishes itself from another by assuming a unique set of transportation costs, i.e., a unique cost structure. In fact, Lawler [10] did not address the question of what cost structure to use, and his followers improvised with their own heuristically chosen cost structures. As a result, these cost structures lead to suboptimal bounds. In practice, these suboptimal bounds can be substantially worse than the one obtained with the optimal cost structure.

The MMT bound proposed in [14] is by definition, the strongest bound among the class of lower bounds provided by transportation problem relaxations whose c_{jt}'s satisfy the following condition:

$$\sum_{s=t}^{t+p_j-1} c_{js} \le f_j(t+p_j),\ \forall j = 1, \ldots, n; t = 0, \ldots, T-p_j, \tag{3}$$

Therefore, if we let $S = \{c | c \text{ satisfies condition (3)}\}$, the MMT bound is defined as $LB^* = \max_{c \in S}\{LB(c)\}$. According to [14], LB^* can be obtained by solving an LP problem in (u, v, c) variables:

$$\begin{aligned}(\text{P1}) \quad \max \quad & \sum_{j=1}^{n} p_j u_j + \sum_{t=0}^{T-1} v_t \\ \text{s.t.} \quad & u_j + v_t - c_{jt} \le 0,\ \forall j = 1, \ldots, n, \\ & \qquad\qquad t = 0, \ldots, T-1, \quad (4) \\ & (3), \\ & u_j \text{ free}, \forall j = 1, \ldots, n \\ & v_t \le 0, \forall t = 0, \ldots, T-1, \\ & c_{jt} \text{ free}, \forall j = 1, \ldots, n; t = 0, \ldots, T-1.\end{aligned}$$

By eliminating the free c variables in the above LP, we get a reduced LP (for more details, refer to [14]):

$$\begin{aligned}(\text{P1}') \quad \max \quad & \sum_{j=1}^{n} p_j u_j + \sum_{t=0}^{T-1} v_t \\ \text{s.t.} \quad & p_j u_j + \sum_{s=t}^{t+p_j-1} v_s \le f_j(t+p_j),\ \forall j = 1, \ldots, n, \\ & \qquad\qquad t = 0, \ldots, T-p_j, \quad (5) \\ & u_j \text{ free}, \forall j = 1, \ldots, n, \\ & v_t \le 0, \forall t = 0, \ldots, T-1.\end{aligned}$$

Further, if we replace $p_j u_j$ with a new variable u'_j, and v_t with v'_t in (P1′) according to

$$u'_j = p_j u_j,\ \forall j = 1, \ldots, n; \text{ and } v'_t = v_t,\ \forall t = 0, \ldots, T-1,$$

it is not difficult to verify that (P1′) is the dual to the following LP that defines the TI bound:

$$\begin{aligned}(\text{TI}) \quad \min \quad & \sum_{j=1}^{n} \sum_{t=0}^{T-p_j} f_j(t+p_j) y_{jt} \\ \text{s.t.} \quad & \sum_{t=0}^{T-p_j} y_{jt} = 1,\ \forall j = 1, \ldots, n, \quad (6) \\ & \sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} y_{js} \le 1,\ \forall t = 0, \ldots, T-1, \\ & y_{jt} \ge 0,\ \forall j = 1, \ldots, n; t = 0, \ldots, T-p_j,\end{aligned}$$

where variables $y_{jt} = 1$ if job j is to start in time period t, and $y_{jt} = 0$ otherwise.

III. Root node computation in [14]

In [14], the computational task at the root node is accomplished in two phases: In Phase I, the following sparse reformulation of (TI) is solved using the primal simplex method:

$$\begin{aligned}(\text{TIS}) \quad \min \quad & \sum_{j=1}^{n} \sum_{t=0}^{T-p_j} f_j(t+p_j) y_{jt} \\ \text{s.t.} \quad & (6), \\ & \sum_{j=0}^{n} y_{j0} = 1, \quad (7) \\ & \sum_{j=0}^{n} y_{jt} - y_{j(t-p_j)} = 0,\ \forall t = 1, \ldots, T-1, \quad (8) \\ & y_{jt} \ge 0,\ \forall j = 0, \ldots, n; t = 0, \ldots, T-p_j,\end{aligned}$$

where the y_{0t} variables can be viewed as slack variables. Suppose that we have obtained an optimal primal solution y_{jt} $(j = 0, \ldots, n; t = 0, \ldots, T-p_j)$ along with dual variable values u''_j $(j = 1, \ldots, n)$, v''_0, and v''_t $(t = 1, \ldots, T-1)$ associated with constraints (6), (7), and (8), respectively. The restriction of the primal solution to index set $(j = 1, \ldots, n; t = 0, \ldots, T-p_j)$ is clearly optimal to (TI). The dual variable values associated with the constraints of (TI) are not explicit, but can be reconstituted according to the formula:

$$u_j = u'_j/p_j = u''_j/p_j,\ \forall j = 1, \ldots, n, \text{ and}$$
$$v_t = v'_t = \begin{cases} v''_t - v''_{t+1}, & \forall t = 0, \ldots, T-2, \\ v''_t, & t = T-1. \end{cases}$$

In Phase II, it is assumed that optimal u and v values have been computed. With u_j and v_t fixed, constraints (4) of (P1) reduce to lower bound constraints on the c_{jt} variables. Any feasible vector c that satisfies (3) and the lower bound constraints, together with u and v, would optimally solve (P1).

Hence, the task on hand is a trivial feasibility problem. Furthermore, the interaction among jobs through the constraints are gone, so the feasibility problem is naturally decomposed into to n smaller problems. Since there are more than one feasible solution, a secondary optimization criterion is imposed on c, resulting in n small LP problems (one for each j) of the form (see [14] for more details):

$$\max \quad \sum_{t=0}^{T-p_j} \sum_{s=t}^{t+p_j-1} c_{js} = \sum_{t=0}^{p_j-1} (t+1)c_{jt} + \sum_{t=p_j}^{T-p_j-1} p_j c_{jt} + \sum_{t=T-p_j}^{T-1} (T-t)c_{jt} \tag{9}$$

$$\text{s.t.} \quad \sum_{s=t}^{t+p_j-1} c_{js} \le f_j(t+p_j),\ \forall t = 0, \ldots, T-p_j, \tag{10}$$

$$c_{jt} \ge u_j + v_t,\ \forall t = 0, \ldots, T-1. \tag{11}$$

Additionally, [14] has discussed how to fix some of the variables in these LPs using the primal solution to (TIS) prior to solving them. This variable fixing is carried out whenever applicable in the present study.

IV. The best root node strategy

What has been described in [14] is one out of many possible options to compute the optimal cost structure for the MMT bound. Due to limitations of experimental conditions at that time, a public-domain LP package called QSopt [2] was used. For this study, we switched to CPLEX 9.13 callable library [7], which provides an array of LP solvers including a barrier solver. Experiments were conducted on a Pentium IV 2.8 GHz desktop computer with 1 GB RAM and running a Linux OS, which is essentially the same computing platform as that of [14], except for an upgrade of OS to the latest version. GNU g++ 3.4 compiler was used to compile our C++ code.

The test problems that we used are the 375 benchmark instances of the total weighted tardiness problem from OR-Library with $n \in \{40, 50, 100\}$ and $p_{\max} = 100$. For each n, there are 125 instances, which are categorized according to two generating parameters rdd $\in \{0.2, 0.4, 0.6, 0.8, 1\}$ and tf $\in \{0.2, 0.4, 0.6, 0.8, 1\}$; hence, they form 25 problem groups, each containing 5 instances. When applied to solve these instances, solvers and algorithms were allowed to run as long as it took them to finish.

In our experiments, we considered the primal simplex, dual simplex, and barrier methods for solving the relevant LP problems, as well as the different pricing rules (for simplex) and crossover (for barrier). We also investigated the use of the network simplex method for solving a reformulation of the Phase II problem described above. Particular attention was paid to the sparsity of LP problems, since sparsity is a key factor that affects the numerical stability and running time of both the simplex and barrier methods on large-scaled problems.

The first thing that we tried and subsequently ruled out was to solve (P1′) directly. Regardless of which LP solver was used, the solution process was unbearably slow. This behavior of LP solvers can be attributed to the fact that the LP had to be augmented with $(nT-P+n)$ logical/slack variables due to the inequality constraints (5). As a result, we end up with an $(nT-P+n)\times(nT-P+n+2n+T)$ coefficient matrix, significantly larger than that associated with the dual problem (TI), which has $(n+T)$ rows and $(nT-P+n)$ columns. The enormous size of (P1′) seems to be too big a difficulty to overcome, even though we also manually applied row transformations to eliminate nonzeros in the coefficient matrix. Had we attempted to deal with (P1)—the LP before reduction—we would have ended up with an even larger problem. Therefore, the other options that we tested all followed the two-phased scheme.

We will discuss the two phases separately because our experience has confirmed that the choice of solver and solver configuration in Phase I has little impact on the computation time of Phase II. For example, the Phase II time for the 40-job instances changed by less than 5% when we switched the Phase I method from primal simplex to barrier.

The sparse LP (TIS) turned out to the most appropriate formulation to work with during Phase I. Sparsity brought benefits such as low memory requirements and fast solver execution. Our focus was therefore turned to comparing different solvers in solving (TIS). To see how the primal simplex, dual simplex, and barrier methods fare against each other, we configure them to their best settings for the current problem:

- For all three solvers, CPLEX presolve and aggregator options were switched off because they were not useful to this particular LP.
- For primal simplex, we used a feasible integer solution as a starting solution (it is not difficult to derive a starting basis from the feasible integer solution, but CPLEX directly takes a starting solution as input). The feasible integer solution was found using three simple heuristics for the total weighted tardiness problem (see [14]). Primal pricing was set to "auto," which means CPLEX automatically decides which pricing algorithm to use.
- For dual simplex, no starting solution was provided to the solver because it was not straightforward to come up with a nontrivial dual feasible solution. Dual pricing was also set to "auto."
- For barrier, it is best to run the solver without crossover. Unlike LP-based branch-and-bound that must take *basic* solutions for warm start, our method does not have such a requirement. We consider this a big advantage because crossover sometimes takes more time than barrier iterations.

Contradicting to common wisdom, dual simplex performed very poorly on this LP; hence, the results obtained with dual simplex are omitted here. Table I shows the CPU time results for primal simplex ("p-spx") and barrier with no crossover ("b-nc"). "avg" (resp."max") stands for the average (resp. maximum) CPU time in seconds. The rows of the table that report zero second correspond to instances that can be solved with the earliest-due-date (EDD) rule because all jobs can be finished by their due dates. The results indicate that barrier is a clear winner against primal simplex on all but some of the relatively easy instances with tf $\in \{0.2, 0.4\}$ (they

also correspond to easy instances for the ensuing branch-and-bound). Furthermore, the strength of barrier relative to primal simplex appears to grow as problem size n increases. For example, the difference is as much as over 20 times in the categories of $n = 100, \text{rdd} \in \{0.2, 0.4, 0.6\}, \text{tf} = 1$.

We also looked into the feasibility of applying Dantzig-Wolfe decomposition and column generation to (TIS) or its equivalence (TI). At the first glance, it seemed promising because Van den Akker et al. [18] previously have been able to speed up the solution of (TI) using this technique. While column generation can be effective on some problem instances, it can also fail to converge on others even after a large number of iterations, i.e., "tailing-off," as observed in [5]. A practical approach to this problem is to terminate column generation early and settle for an approximate solution. The topic of utilizing such an approximation solution is beyond the scope of this paper, and is further developed in a sequel [12].

Granted that the column generation terminates without tailing-off, some extra work is needed to utilize the solution information obtained. It is worth nothing that column generation generally yields a *nonbasic* optimal primal solution to (TIS), and does not provide all the information needed to recover an optimal dual solution to (TIS), which is what we really need for Phase II. In our preliminary experiments, we tried to use the nonbasic solution found by column generation as an initial solution to start primal simplex on (TIS). This was a crossover operation since the optimal objective value had already been obtained. The crossover took too much time and thus was not competitive with the barrier method. A more effective method is to utilize the dual solution obtained, which is discussed in detail in [12].

Next, we investigated alterative approaches to computing the Phase II problem defined by (9)–(11). Note that constraints (10) define an *interval matrix*—a matrix in which the elements are 0's or 1's with the latter appearing consecutively in each column [11]. Therefore, the problem can be reformulated as an equivalent network flow problem. The reformulation is best illustrated using an example. To simplify the notation, let us omit the job index j in the Phase II problem and assume that $p_j = 3$ and $T = 7$. After the addition of slack variables $s_0, \ldots, s_4$, constraints (10) become ones shown in Table (III). Furthermore, we keep the first row fixed, subtract one row from the one immediately below for every two consecutive rows, and duplicate the last row and flip the sign, and finally, we get those in Table (IV)

These equations clearly define the conservation-of-flow constraints in a network flow problem. With the addition of dummy nodes and auxiliary arcs, the network flow problem can be represented by the network diagram shown in Figure 1.

We compared the approach in [14], which applies the dual simplex method to the Phase II problem, with our new approach, which applies the network simplex method to the network flow reformulation. The results are reported in Table II. The Phase II CPU time was registered as 0 if an instance was solved with the EDD rule, or if the lower bound met the upper bound after Phase I. The CPU seconds shown reflects the total time spent in solving all n problems—one for each job $j = 1, \ldots, n$—after a Phase I using barrier with no crossover. "d-spx" and "n-spx" stand for dual simplex and network simplex. These results indicate that speedup by a factor of 10 or more is achieved by using network simplex in conjunction with the network flow reformulation. As problem size n increases, the advantage of network simplex over dual simplex widens to as much as being 31 times faster.

V. Conclusion

Our findings in this study suggest that the best root node strategy is to apply the barrier method with no crossover to the sparse formulation (TIS) in Phase I, and apply the network simplex method to the network flow formulation in Phase II. Adopting this strategy can speed up the root node computation by 10–20 times on the majority of difficult instances. Consequently, there will be a substantial reduction in the overall solution time of branch-and-bound if the work at the root accounts for most of the computation, as in the case of the total weighted tardiness problem considered here.

An interesting new phenomenon was observed in our experiments. It was understood that generally there is not a unique optimal cost structure. Different LP solvers indeed resulted in different transportation costs, which in turn led to different numbers of nodes in branch-and-bound trees. It was really a pleasant surprise when we found that the cost structure resulted from barrier with no crossover, i.e., the best Phase I method that we identified, all in all generated significantly smaller trees than the other solvers and solver configurations, including the barrier methods with primal and dual crossovers. The reason behind the observed efficiency is yet to be investigated.

Finally, we would like to comment on the use of the barrier method with crossover at the root node in LP-based branch-and-bound that works with (TI) or (TIS). The barrier method has demonstrated a clear edge over the simplex method on large-scaled LP problems, and doing a crossover generally takes much less time than running primal or dual simplex from scratch. With an advanced basis recovered by crossover, dual simplex can be applied to subsequent nodes in the standard manner. This could be a simpler and more efficient alternative to the Dantzig-Wolfe decomposition and column generating approach of [18]. Should it turn out to be true, we would then not have to deal with the complications of intertwining column generation and valid inequalities.

References

[1] T. S. Abdul-Razaq, C. N. Potts, L. N. van Wassenhove, A survey of algorithms for the single machine total weighted tardiness scheduling problem, Discrete Appl. Math. 26 (1990) 235–253.

[2] D. Applegate, W. Cook, S. Dash, M. Mevenkamp, QSopt reference manual version 1.0, (2003), `http://www.isye.gatech.edu/~wcook/qsopt/`.

[3] P. Avella, M. Boccia, B. D'Auria, Near-optimal solutions of large-scale single-machine scheduling problems, INFORMS J. Comput. 17 (2) (2005) 183–191.

[4] H. Belouadah, Marc E. Posner, C. N. Potts, Scheduling with release dates on a single machine to minimize total weighted completion time, Discrete Appl. Math. 36 (3) (1992) 213–231.

[5] L.-P. Bigras, M. Gamache, G. Savard, Time-indexed formulations and the total weighted tardiness problem, G-2005-30, Les Cahiers du GERAD, 2005.

[6] K. Bülbül, P. Kaminsky, C. Yano, Preemption in single machine earliness/tardiness scheduling, J. Sched. (2005), Forthcoming.

TABLE I

CPU TIME IN SECONDS FOR PHASE I PROBLEM

rdd	tf	$n = 40$				$n = 50$				$n = 100$			
		p-spx		b-nc		p-spx		b-nc		p-spx		b-nc	
		avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
0.2	0.2	1.8	2.4	1.8	3.5	4.9	6.2	5.2	7.3	18.5	23.2	39.9	53.0
	0.4	9.7	13.1	7.6	11.4	18.9	29.9	13.9	22.6	142	162	153	227
	0.6	10.3	13.6	6.9	9.8	32.6	44.1	18.3	21.9	404	527	233	272
	0.8	24.1	41.4	9.1	11.3	40.6	47.6	15.6	17.3	2080	2474	158	253
	1	32.3	50.0	5.0	8.0	55.9	83.3	10.7	14.2	1577	2108	61.2	124
0.4	0.2	0.5	1.4	0.2	0.6	0.8	3.2	0.4	1.8	9.7	17.3	1.2	2.8
	0.4	8.2	13.8	5.9	8.3	13.6	20.9	10.8	20.4	91	144	91.9	132
	0.6	13.5	20.0	7.0	9.6	30.7	46.7	16.5	22.5	355	675	220	258
	0.8	27.2	33.5	9.8	12.5	38.1	51.7	13.9	16.4	1722	3344	112	166
	1	23.4	43.2	4.2	5.8	62.8	81.8	11.7	17.4	2071	2436	67.5	139
0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.4	7.8	11.5	3.8	7.0	13.2	26.1	7.1	11.3	141	238	52.5	83.4
	0.6	24.5	41.9	8.6	12.7	54.5	66.4	17.4	25.4	317	501	160	190
	0.8	31.0	45.8	7.8	10.4	52.0	106	15.7	23.1	763	1190	126	187
	1	26.6	50.3	6.2	9.0	37.8	51.1	10.6	13.4	2039	2317	56.4	88.2
0.8	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.4	14.3	38.2	1.9	3.7	23.0	36.4	6.4	13.6	106	283	5.4	11.2
	0.6	19.0	31.5	7.2	10.1	54.6	84.0	14.1	19.3	443	757	92.7	174
	0.8	26.7	33.3	7.0	7.9	38.4	52.5	12.2	18.5	621	1083	109	141
	1	31.1	55.1	7.3	9.5	47.9	64.8	12.1	15.6	1263	1830	105	169
1	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.4	1.7	8.3	0.5	2.4	14.6	47.5	3.1	6.5	2.4	10.7	0.9	4.1
	0.6	28.2	37.4	6.0	8.4	51.7	79.8	12.6	24.4	584	1055	97.1	148
	0.8	22.2	29.8	7.7	10.7	35.1	39.2	12.7	15.5	432	555	86.6	117
	1	27.0	31.1	6.8	8.5	48.6	63.9	11.2	17.0	1146	1756	80.7	151

TABLE II

CPU TIME IN SECONDS FOR PHASE II PROBLEM

rdd	tf	$n = 40$				$n = 50$				$n = 100$			
		d-spx		n-spx		d-spx		n-spx		d-spx		n-spx	
		avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
0.2	0.2	0.6	1.5	0.10	0.22	1.3	4.3	0.14	0.38	15.9	19.5	1.13	1.25
	0.4	7.1	11.5	0.65	0.81	8.3	16.9	0.68	1.18	106	115	3.38	5.27
	0.6	6.4	8.0	0.67	0.81	18.9	21.2	1.39	1.49	162	188	6.95	7.88
	0.8	9.7	16.0	0.98	1.31	14.6	16.5	1.30	1.43	138	174	6.39	7.00
	1	3.4	5.6	0.51	0.76	7.7	14.4	0.88	1.42	61.8	100	4.19	5.46
0.4	0.2	0.0	0.0	0.00	0.00	0.4	1.9	0.06	0.29	0.4	2.0	0.08	0.40
	0.4	4.5	7.3	0.48	0.65	7.5	16.0	0.64	1.11	75.7	108	3.16	3.84
	0.6	6.5	9.2	0.78	0.97	17.9	25.9	1.29	1.65	170	211	6.23	6.92
	0.8	9.8	15.0	1.01	1.37	11.8	16.1	1.24	1.44	113	128	5.86	6.82
	1	2.3	3.9	0.40	0.60	9.8	14.3	1.13	1.37	68.0	106	4.53	6.12
0.6	0.2	0.0	0.0	0.00	0.00	0.0	0.0	0.00	0.00	0.0	0.0	0.00	0.00
	0.4	3.4	6.4	0.43	0.72	7.1	12.0	0.76	1.05	59.7	89.5	2.90	3.93
	0.6	9.7	16.7	1.00	1.42	21.1	34.2	1.56	1.92	160	199	5.91	7.22
	0.8	7.2	9.3	0.93	1.09	14.2	30.8	1.37	2.23	96.5	113	5.39	6.18
	1	4.1	7.9	0.60	0.92	8.6	12.5	1.08	1.30	79.2	94.6	4.98	5.50
0.8	0.2	0.0	0.0	0.00	0.00	0.0	0.0	0.00	0.00	0.0	0.0	0.00	0.00
	0.4	0.6	2.4	0.13	0.49	3.0	12.9	0.25	0.91	4.6	11.6	0.64	1.32
	0.6	7.2	10.0	0.84	1.09	16.0	22.3	1.37	1.58	118	158	5.38	6.25
	0.8	6.1	8.4	0.80	1.02	10.0	12.9	1.14	1.47	96.0	127	5.48	6.48
	1	5.1	7.6	0.68	0.90	10.1	12.8	1.18	1.40	80.0	106	5.00	5.66
1	0.2	0.0	0.0	0.00	0.00	0.0	0.0	0.00	0.00	0.0	0.0	0.00	0.00
	0.4	0.4	2.1	0.07	0.37	2.6	5.7	0.36	0.76	0.6	3.0	0.13	0.65
	0.6	6.3	9.0	0.80	1.02	13.4	26.3	1.20	1.81	99.7	153	5.58	7.36
	0.8	6.6	9.7	0.84	1.11	11.6	15.2	1.21	1.38	77.8	89.8	5.17	5.44
	1	5.6	8.4	0.78	0.95	9.8	14.6	1.11	1.39	82.7	101	5.16	5.80

TABLE III

CONSTRAINTS WITH SLACK VARIABLES (A)

$$
\begin{array}{ccccccccccccccc}
c_0 & +c_1 & +c_2 & & & & & +s_0 & & & & & & = & f(3) \\
 & +c_1 & +c_2 & +c_3 & & & & & +s_1 & & & & & = & f(4) \\
 & & +c_2 & +c_3 & +c_4 & & & & & +s_2 & & & & = & f(5) \\
 & & & +c_3 & +c_4 & +c_5 & & & & & +s_3 & & & = & f(6) \\
 & & & & +c_4 & +c_5 & +c_6 & & & & & +s_4 & & = & f(7).
\end{array}
$$

TABLE IV

CONSTRAINTS WITH SLACK VARIABLES (B)

$$
\begin{array}{rrrrrrrrrrrrcl}
c_0 & +c_1 & +c_2 & & & & & +s_0 & & & & & = & f(3) \\
-c_0 & & & +c_3 & & & & -s_0 & +s_1 & & & & = & f(4)-f(3) \\
 & -c_1 & & & +c_4 & & & & -s_1 & +s_2 & & & = & f(5)-f(4) \\
 & & -c_2 & & & +c_5 & & & & -s_2 & +s_3 & & = & f(6)-f(5) \\
 & & & -c_3 & & & +c_6 & & & & -s_3 & +s_4 & = & f(7)-f(6) \\
 & & & & -c_4 & -c_5 & -c_6 & & & & & -s_4 & = & -f(7).
\end{array}
$$

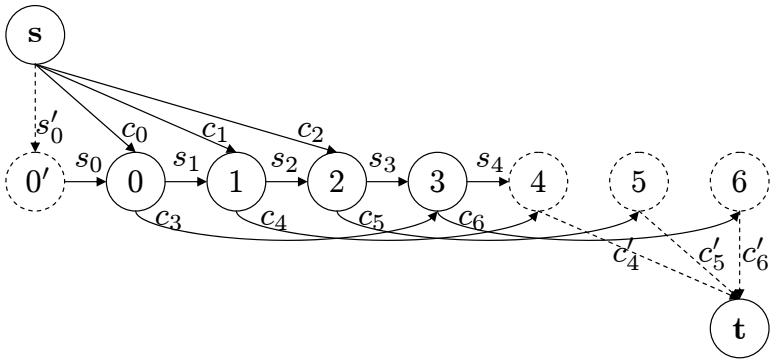

Fig. 1. Network flow reformulation of Phase II problem

[7] CPLEX, CPLEX callable library, version 9.1, 2005, CPLEX: A division of ILOG.

[8] M. E. Dyer, L. A. Wolsey, Formulating the single machine sequencing problem with release dates as a mixed integer program, Discrete Appl. Math. 26 (1990) 255–270.

[9] L. Gelders, P. R. Kleindorfer, Coordinnating aggregate and detailed scheduling in the one-machine job shop: Part I. Theory, Oper. Res. 22 (1974) 46–60.

[10] E. L. Lawler, On scheduling problems with deferral costs, Management Sci. 11 (2) (1964) 280–288.

[11] G. L. Nemhauser, L. A. Wolsey, Integer and combinatorial optimization, John Wiley, NY, 1988.

[12] Y. Pan, Dual relaxations of the time-indexed ilp formulation for min-sum scheduling problems, (2006), Submitted for publication. Available online: http://www.optimization-online.org/DB_HTML/2006/08/1438.html.

[13] Y. Pan, L. Shi, Dual constrained single machine sequencing to minimize total weighted completion time, IEEE Trans. Automat. Sci. & Eng. 2 (4) (2005) 344–357.

[14] ———, On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems, Math. Program., Ser. A (2006), Forthcoming. Available online: http://dx.doi.org/10.1007/s10107-006-0013-4.

[15] F. Sourd, S. Kedad-Sidhoum, The one-machine problem with earliness and tardiness penalties, J. Sched. 6 (2003) 533–549.

[16] ———, An efficient algorithm for the earliness-tardiness scheduling problem, (2005), Working paper. Online: http://www.optimization-online.org/DB_FILE/2005/09/1205.pdf.

[17] J. P. Sousa, L. A. Wolsey, A time indexed formulation of non-preemptive single machine scheduling problems, Math. Program. 54 (1992) 353–367.

[18] J. M. van den Akker, C. A. J. Hurkens, M. W. P. Savelsbergh, Time-indexed formulations for machine scheduling problems: Column generation, INFORMS J. Comput. 12 (2) (2000) 111–124.

[19] J. M. van den Akker, C. P. M. van Hoesel, M. W. P. Savelsbergh, A polyhedral approach to single-machine scheduling problems, Math. Program. 85 (1999) 541–572.