Mathematica
Balkanica

New Series Vol.23, 2009, Fasc. 3-4

Solving the Earliness Tardiness Scheduling Problem by DC Programming and DCA

Hoai An Le Thi [1], *Nguyen Quang Thuan* [1], *Nguyen Huynh Tuong* [2], *Tao Pham Dinh* [3]

Time-index formulation for the earliness tardiness scheduling problem has received a great attention from many researchers because lower bound obtained by linear relaxation is rather good. Much work is devoted to tackle its upper bound. In this paper, we consider this formulation by additionally proposing a deadline for each job. We also propose an approach based on DC (Difference of Convex functions) programming and DCA (DC Algorithm) to find upper bound efficiently for this problem. The results obtained are promising.

1. Introduction

Time-indexed formulation [22] is a well-known way to formulate a single machine earliness tardiness scheduling problem as a mixed integer problem. It is based on time-discretization. More obviously, time is divided into time periods and the binary variable x_{it} indicates for each job i and each time period t whether job i is completed at period t. The main advantage of this formulation is that the lower bound provided by linear relaxation is strong, stronger than the lower bounds provided by the linear relaxation of many other integer programming formulations ([22],[1]).

Much research has been carried out for this formulation. Most of them use heuristic methods or local searches to find the upper bound. In single machine scheduling problems, several authors have studied a special case in which every job has the same due date (called common due date case). For this special case, Van den Akker, Hoogeveen and Velde [24] solved instances up to 125 jobs (the common due date is assumed to be very large). Most recently, Sourd settled instances up to 1000 jobs [18]. However, for the general case, only instances up to 50 jobs can be solved ([18], [20], [26]).

To the best of our knowledge, one has not mentioned a deadline for each job in the time-indexed formulation yet. Such problem is encountered in practice when there is a limit on computation time for each job. Moreover, as shown by the recent surveys of T'kindt and Billaut [23] and Hoogeveen [3], most of research effort was devoted to earliness tardiness problems with one machine. In this paper, we consider the earliness tardiness problem in a parallel machine environment with deadline of each job. According to classical scheduling notation [2], it is denoted by $(P|r_i, \tilde{d}_i| \sum \alpha_i E_i + \beta_i T_i)$. We consider it and propose a new approach based on DC programming and DCA.

DC programming and DCA were introduced by Pham Dinh Tao in 1985 and have been extensively developed by Le Thi Hoai An and Pham Dinh Tao since 1994. The DCA has been successfully applied to real world non convex programs in various fields of applied sciences ([10], [13], [14]). It is one of the rare efficient algorithms for non smooth non convex programming which allows solving large-scale DC programs.

The original time-indexed formulation is transformed to an equivalent DC program by using exact penalty technique [11]. The resulting DC program is then handled by DCA. The instances for test are taken from Sourd and Kedad-Sidhoum's data [19]. Of course, these instances do not contain the deadlines $\tilde{d}_i$. So, we need to generate additionally $\tilde{d}_i$ from these instances. We test instances up to 80 jobs. The results show that our proposed approach is efficient for solving this scheduling problem.

The rest of the paper is organized as follows. In Section 2, we give a time-indexed formulation for the earliness tardiness scheduling problem. Section 3 is dedicated to the solution method via DC programming, DCA. In section 4, we provide the description of the numerical experiments and report the analysis of the results obtained in this work. Section 5 concludes the paper with conclusions and future research directions.

2. Formulation

Let $I = \{1, 2, \ldots, n\}$ be the set of n independent jobs which are scheduled on m identical parallel machines without any pre-emption. For each job i, let $\alpha_i, \beta_i, p_i, r_i, d_i, \tilde{d}_i$ be the earliness penalty, the tardiness penalty, the processing time, the release date, the due date and the deadline of job i respectively. Denote c_{it} as the processing cost of job i if it is completed at time t. The scheduling horizon is denoted by $T = \min(\max(\max_{i=1}^n r_i; \max_{i=1}^n d_i) + \sum_{i=1}^n p_i; \max_{i=1}^n \tilde{d}_i)$.

We have the time-indexed formulation (TI) as follows:

$$\min \sum_{i=1}^{n} \sum_{t=r_i+p_i}^{\tilde{d}_i} c_{it} x_{it} \tag{2.1}$$

$$\sum_{t=r_i+p_i}^{\tilde{d}_i} x_{it} = 1, \quad \forall i \in [1, n] \tag{2.2}$$

$$\sum_{i=1}^{n} \sum_{s=t}^{t+p_i-1} x_{is} \leq m, \quad \forall t \in [1, T] \tag{2.3}$$

$$x_{it} = 0, \quad \forall i \in [1, n], \quad t \in [1, r_i + p_i) \cup (\tilde{d}_i, T] \tag{2.4}$$

$$x_{it} \in \{0, 1\}, \quad \forall i \in [1, n], \quad t \in [0, T] \tag{2.5}$$

where the binary variable x_{it} for each job i and time period t indicates whether job i completes at time t ($x_{it} = 1$) or not ($x_{it} = 0$). The objective function (2.1) is to minimize the sum of all the processing costs. The assignment constraints (2.2) state that each job has to be executed exactly once, and the capacity constraints (2.3) state that at most m jobs can be handled at any time period t. Equations (2.4) specify that each job i can not be executed at some period t.

The time-indexed formulation is very simple, and it can be used to model many types of single machine scheduling problems. Different objective functions can be modeled by appropriate choices of cost coefficients. In this paper, for the earliness tardiness scheduling problem, we can compute $c_{it} = \max\{\alpha_i(d_i - t); \beta_i(t - d_i)\}$.

The main disadvantage of the time-indexed formulation is its size. For instances with many jobs or jobs with large processing times, the number of binary variable nT becomes very large.

3. Solution method via DC programming and DCA

DC Programming and DCA are becoming quite popular with many successful applications in different fields. In order to give the reader an easy understanding of the theory of DC Programming and DCA, we first outline this theory before applying it to the time-indexed formulation of earliness tardiness scheduling problem.

3.1. DC programming and DCA

Let $\Gamma_0(\mathbb{R}^n)$ denote the convex cone of all lower semi-continuous proper convex functions on $\mathbb{R}^n$. The vector space of DC functions, $DC(\mathbb{R}^n) = \Gamma_0(\mathbb{R}^n) - \Gamma_0(\mathbb{R}^n)$, is quite large to contain most real life objective functions and is closed under all operations usually considered in optimization.

Consider the general DC program

$$\alpha = \inf\{f(x) := g(x) - h(x) : x \in \mathbb{R}^n\} \quad (P_{dc})$$

with $g, h \in \Gamma_0(\mathbb{R}^n)$. Such a function f is called a DC function; $g - h$ is a DC decomposition of f while the convex functions g and h are DC components of f.

Notice that a constrained DC program whose feasible set C is convex can always be transformed to an unconstrained DC program by adding the indicator function χ_C of C ($\chi_C(x) = 0$ if $x \in C, +\infty$ otherwise) to the function g.

Let

$$g^*(y) := sup\{\langle x, y\rangle - g(x) : x \in \mathbb{R}^n\}$$

be the conjugate function of g. We have the dual program of (P_{dc})

$$\alpha = \inf\{h^*(y) - g^*(y) : y \in dom(h^*)\}$$

which can be written, with the convention $+\infty - (+\infty) = +\infty$, as:

$$\alpha = \inf\{h^*(y) - g^*(y) : y \in Y\} \quad (D_{dc})$$

We can see perfect symmetry between primal and dual DC programs, that is the dual to (D_{dc}) is exactly (P_{dc}).

If g or h are polyhedral convex functions then (P_{dc}) is called a polyhedral DC program, which plays a main role in non convex programming (see [9] and references therein), and enjoys interesting properties (from both theoretical and practical viewpoints) concerning the local optimality and the convergence of the DCA.

DC programming investigates the structure of the vector space $DC(\mathbb{R}^n)$, DC duality and optimality conditions for DC programs. The complexity of DC programs resides, of course, in the lack of practical global optimality conditions. We have the following necessary local optimality conditions for primal DC programs, by symmetry the dual version can be easily obtained (see [9], [10], [13], [14] and references therein):

$$\partial h(x^*) \cap \partial g(x^*) \neq \emptyset$$

(such a point x^* is called a critical point of $g - h$ or for (P_{dc})), and

$$\emptyset \neq \partial h(x^*) \subset \partial g(x^*). \tag{3.1}$$

The condition (3.1) is also sufficient for many important classes of DC programs. In particular it is sufficient for the following cases quite often encountered in practice:

◇ In polyhedral DC programs with h being a polyhedral convex function (see [9], [10], [13], [14] and references therein). In this case, if h is differentiable at a critical point x^*, then x^* is actually a local minimizer for (P_{dc}). Since a convex function is differentiable everywhere except for a set of measure zero, one can say that a critical point x^* is almost always a local minimizer for (P_{dc}).

◇ In the case where the function f is locally convex at x^* ([8], [10]).

The transportation of global solutions between (P_{dc}) and (D_{dc}) is expressed by:

Property 3.1:

$$[\bigcup_{y^* \in \mathcal{D}} \partial g^*(y^*)] \subset \mathcal{P}, \quad [\bigcup_{x^* \in \mathcal{P}} \partial h(x^*)] \subset \mathcal{D} \tag{3.2}$$

where $\mathcal{P}$ and $\mathcal{D}$ denote the solution sets of (P_{dc}) and (D_{dc}) respectively.

Under technical conditions, this transportation also holds for local solutions of (P_{dc}) and (D_{dc}) (For example see [8], [10], [13] for more information).

Property 3.2: Let x^* be a local solution to (P_{dc}) and let $y^* \in \partial h(x^*)$. If g^* is differentiable at y^* then y^* is a local solution to (D_{dc}). Similarly, let y^* be a local solution to (D_{dc}) and let $x^* \in \partial g^*(y^*)$. If h is differentiable at x^* then x^* is a local solution to (P_{dc}).

Based on local optimality conditions and duality in DCA, the idea of DCA is quite simple: each iteration k of DCA approximates the concave part $-h$ by its affine majorization (that corresponds to taking $y^k \in \partial h(x^k)$) and minimizes the resulting convex function (that is equivalent to determining $x^{k+1} \in \partial g^*(y^k)$).

```
Generic DCA scheme
Initialization: Let $x^0 \in \mathbb{R}^n$ be a best guess, $k \leftarrow 0$.
Repeat
    Calculate $y^k \in \partial h(x^k)$
    Calculate $x^{k+1} \in \arg\min\{g(x) - h(x^k) - \langle x - x^k, y^k\rangle : x \in \mathbb{R}^n\}$   $(P_k)$
    $k \leftarrow k+1$
Until convergence of $\left\{x^k\right\}$.
```

Convergence properties of the DCA and its theoretical basis are described in ([8], [10], [13], [14]). However, it is worthwhile to report the following properties that are useful in the next (for simplicity, we omit here the dual part of these properties):

- DCA is a descent method *without line search*, say the sequence $\{g(x^k) - h(x^k)\}$ is decreasing.
- If $g(x^{k+1}) - h(x^{k+1}) = g(x^k) - h(x^k)$ then x^k is a critical point of $g - h$. In this case, DCA terminates at k^{th} iteration.
- If the optimal value α of problem (P_{dc}) is finite and the infinite sequence $\{x^k\}$ is bounded, then every limit point $\widetilde{x}$ (of this sequence is a critical point of $g - h$.
- DCA has a linear convergence for general DC programs. Especially, for polyhedral DC programs the sequences $\{x^k\}$ contain finitely many elements and the algorithm convergences to a solution in a finite number of iterations.

The construction of DCA involves DC components g and h but not the function f itself. Hence, for a DC program, each DC decomposition corresponds to a different version of DCA. Since a DC function f has an infinite number of DC decompositions which have crucial impacts on the qualities (speed of convergence, robustness, efficiency, globality of computed solutions,...) of DCA, the search of a "good" DC decomposition is important from algorithmic points of view. Moreover, despite its local character, DCA with a good initial point can converge to global solutions. Finding a "good" initial point is then also an important stage of DCA. How to develop an efficient algorithm based on the generic DCA scheme for a practical problem is thus a judicious question to be studied, and the answer depends on the specific structure of the problem being considered.

In the past years DCA has been successfully applied in several works of various fields among them Learning Machines, Financial optimization, Supply chain management, telecommunication, etc (see e.g. [8] - [10], [6], [12], [14], [15], [17], [16] and references therein).

3.2. DCA for solving (TI)

By using an exact penalty result, we can reformulate the aforementioned (TI) in the form of a concave minimization program. The exact penalty technique aims at transforming the original (TI) into a more tractable equivalent problem in the DC optimization framework. Let $K := \{x \in \mathbb{R}^{nT} : x \text{ satisfies } (2.2)(2.3) \text{ and } (2.4)\}$. The feasible set of (TI) will be $S = \{x : x \in K, x \in \{0,1\}^{nT}\}$.

Let us consider the function $p : \mathbb{R}^{nT} \to \mathbb{R}$ defined by

$$p(x) = \sum_{i=1}^{n}\sum_{t=1}^{T} \min\{x_{it}, 1 - x_{it}\}. \tag{3.3}$$

It is clear that $p(x)$ is concave and finite on K, $p(x) \geq 0 \quad \forall x \in K$ and

$$\{x : x \in S\} = \{x : x \in K, p(x) \leq 0\}. \tag{3.4}$$

Hence problem (TI) can be rewritten as

$$\min\Big\{\sum_{i=1}^{n} \sum_{t=r_i+p_i}^{\tilde{d}_i} c_{it}x_{it} : x \in K, p(x) \leq 0\Big\}. \tag{3.5}$$

The following theorem is in order.

Theorem 3.1. *Let K be a nonempty bounded polyhedral convex set, f be a finite concave function on K and p be a finite nonnegative concave function on K. Then there exists $\eta_0 \geq 0$ such that for $\eta > \eta_0$ the following problems have the same optimal value and the same solution set:*

$$(P_\eta) \qquad \alpha(\eta) = \min\{f(x) + \eta p(x) : x \in K\}$$

$$(P) \qquad \alpha = \min\{f(x) : x \in K, p(x) \leq 0\}.$$

Furthermore

- *If the vertex set of K, denoted by $V(K)$, is contained in $x \in K : p(x) \leq 0$, then $\eta_0 = 0$.*

- *If* $p(x) > 0$ *for some* x *in* $V(K)$, *then* $\eta_0 = \min\Big\{\frac{f(x)-\alpha(0)}{S_0} : x \in K, p(x) \leq 0\Big\}$, *where* $S_0 = \min\Big\{p(x) : x \in V(K), p(x) > 0\Big\} > 0$.

Proof. The proof for the general case can be found in [11]. ■

From Theorem 3.1 we get, for a sufficiently large number η ($\eta > \eta_0$), the equivalent concave minimization problem to (3.5)

$$\min\left\{ f_\eta(x) := \sum_{i=1}^{n} \sum_{t=r_i+p_i}^{\tilde{d}_i} c_{it}x_{it} + \eta p(x) : x \in K \right\} \tag{3.6}$$

which is a DC program of the form

$$\min\{g(x) - h(x) : x \in \mathbb{R}^{nT}\} \tag{3.7}$$

where

$$g(x) = \chi_K(x); \; h(x) = -f_\eta(x) = -\sum_{i=1}^{n} \sum_{t=r_i+p_i}^{\tilde{d}_i} c_{it}x_{it} - \eta p(x).$$

We have successfully transformed an optimization problem with integer variables into its equivalent form with continuous variables. Notice that (3.7) is a polyhedral DC program where both g and h are polyhedral convex functions.

We now construct two sequences $\{x^k\}$ and $\{y^k\}$. For notational simplicity, we group all variables and all costs in column vectors $x = (x_{11}, x_{12}, ...x_{1T}, x_{21}, ..., x_{nT})^{\overline{T}}$, $c = (c_{11}, c_{12}, ...c_{1T}, c_{21}, ..., c_{nT})^{\overline{T}}$ then renumber them as follows $x = (x_1, x_2, ..., x_{nT})^{\overline{T}}$ and $c = (c_1, c_2, ..., c_{nT})^{\overline{T}}$ respectively where $\overline{T}$ denotes the transpose operator.

By definition, a sub-gradient $y^k \in \partial h(x^k)$ can be computed as follows:

$$y_i^k = \begin{cases} -c_i + \eta, & \text{if } x_i^k \geq 0.5; \\ -c_i - \eta, & \text{otherwise.} \end{cases} \quad i = 1..nT \tag{3.8}$$

By using y^k, we then compute x^k by solving the linear program:

$$x^{k+1} = \arg\min\{\chi_K(x) - \langle x - x^k, y^k \rangle : x \in \mathbb{R}^{nT}\} \tag{3.9}$$

or

$$x^{k+1} = \arg\min\{-\langle y^k, x \rangle : x \in K\}. \tag{3.10}$$

Our algorithm is summarized in **Algorithm 1**.

```
Algorithm 1: DCA applied to (3.7)
Initialization:
        Choose a initial point $x^0$, set $k = 0$;
        Let $\epsilon_1, \epsilon_2$ be sufficiently small positive numbers;
Repeat
        Compute $y^k$ via (3.8);
        Solve the linear program (3.10) to obtain $x^{k+1}$;
        $k \leftarrow k + 1$;
Until either $\|x^{k+1} - x^k\| \le \epsilon_1(\|x^k\| + 1)$ or $|f_\eta(x^{k+1}) - f_\eta(x^k)| \le \epsilon_2(|f_\eta(x^k)| + 1)$.
```

Theorem 3.2. *(Convergence properties of Algorithm **DCA**)*

*(i) **DCA** generates the sequence $\{x^k\}$ contained in $V(K)$ such that the sequence $\{f_\eta(x^k)\}$ is decreasing.*

(ii) The sequence $\{x^k\}$ converges to $x^* \in V(K)$ after a finite number of iterations.

(iii) The point x^* is a critical point of Problem (3.7).

(iv) Moreover, if $x^*_i \neq \frac{1}{2}$ for all $i = 1, \dots, nT$, then x^* is a local solution to (3.7).

(v) For a number η sufficiently large, if at iteration r we have $x^r \in \{0,1\}^{nT}$, then $x^k \in \{0,1\}^{nT}$ for all $k \ge r$.

Proof. i) and (iii) are direct consequences of the convergence properties of general DC programs while (ii) is a convergence property of a Polyhedral DC program. (iv) is due to the fact that the necessary local optimality condition $\partial h(x^*) \subset \partial g(x^*)$ is also sufficient (remember that (3.7) is a Polyhedral DC program with $g := \chi_K$ and $h := -f_{\eta)}$ being polyhedral functions), and such a condition is verified at x^* when h is differentiable at x^*, say $x^*_i \neq \frac{1}{2}$ for all $i = 1, \dots, nT$.

v) Let

$$\eta > \eta_1 := \max\left\{ \frac{c^T x - \xi}{S_0} : x \in V(K),\ p(x) \le 0 \right\},$$

where $\xi := \min\{c^T x : x \in V(K)\}$ and $S_0 := \min\{p(x) : x \in V(K), p(x) > 0\}$. Let $\{x^k)\} \subset V(K)$ $(k \ge 1)$ be generated by **DCA**. If $V(K) \subset \{0,1\}^{nT}$, then the assertion is trivial. Otherwise, let $x^r \in \{0,1\}^{nT}$ and $x^{r+1} \in V(K)$ be an optimal solution of the linear program (3.10). Then from (i) of this theorem we have

$$c^T x^{r+1} + tp(x^{r+1}) \leq c^T x^r + tp(x^r).$$

Since $p(x^r) = 0$, it follows

$$tp(x^{r+1}) \leq c^T x^r - c^T x^{r+1} \leq c^T x^r - \xi.$$

If $p(x^{r+1}) > 0$, then

$$\eta \leq \frac{c^T x^r - c^T x^{r+1}}{p(x^{r+1})} \leq \frac{c^T x^r - \xi}{S_0} \leq \eta_1$$

which contradicts the fact that $\eta > \eta_1$.
The proof is then complete.

■

4. Experimental results

The Algorithm 1 has been coded in C and implemented on a Intel Core 2 CPU 1.86Ghz and RAM 2GB. The instances are based on the data of Sourd and Kedad-Sidhoume [19]. From r_i, p_i, d_i taken, we have generated randomly $\tilde{d_i} \in [\max(r_i + p_i; d_i), 1.1(\max(r_i) + \sum p_i)]$.

In order to evaluate the quality of the upper bound, we compare Algorithm 1 with MIP solver of CPLEX11.2. Linear programs in each iteration of Algorithm 1 and linear relaxation problem are also solved by CPLEX11.2.

We tested in turn with $n = 40$, $n = 60$ and $n = 80$ while m varies from 1 to 3. For each n and m, 10 instances are used. The first 5 instances have small processing times and the last 5 instances have large processing times.

We show the results in the three tables below corresponding to $m = 1$, $m = 2$ and $m = 3$ respectively. Some notations used in the results table:

- ◇ Val_{DCA}: upper bound obtained by Algorithm 1,
- ◇ it: number of iteration of Algorithm 1,
- ◇ $time_{DCA}$: computing time of Algorithm 1,
- ◇ Val_{CPLEX}: upper bound obtained by CPLEX after $time_{DCA}$ seconds,
- ◇ LB: lower bound obtained by linear relaxation,

◇ $GAP_1 = \frac{(Val_{DCA} - LB)}{Val_{DCA}} 100\%$,

◇ $GAP_2 = \frac{(Val_{CPLEX} - LB)}{Val_{CPLEX}} 100\%$,

◇ NA: CPLEX has not furnished the integer solution yet after $time_{DCA}$ seconds.

From the result tables, we observe that:

- For every instances, Algorithm 1 always furnishes the integer solutions with the small number of iteration.
- Algorithm 1 is fast.
- GAP_1s are small. It means that either the upper bounds obtained by Algorithm 1 are rather close to the optimal values, or these upper bounds may even be the optimal values. 46/90 test instances have $GAP_1 < 5\%$.
- With the same time, Algorithm 1 often furnishes the objective value better than CPLEX does (see the two columns Val_{DCA} and Val_{CPLEX}). It is also better in the average sense. Moreover, CPLEX has not found the integer solution yet in 10/90 cases. Visually, the charts from Fig. 1 show that Algorithm 1 is surpassing.

5. Conclusion

In this paper, we consider the time-indexed formulation for the earliness tardiness scheduling with deadline for each job. We also propose an efficient approach to solve this formulation. This approach is based on DC Programming and DCA. The computational results obtained show that this approach is efficient and original as it can give integer solutions while working in a continuous domain. The future research is directed towards building hybrids between DCA and Branch-and-Bound or Branch-and-Cut Algorithm to solve such problems.

References

[1] M.E. Dyer, L.A. Wolsey, Formulating the single machine sequencing problem with release dates as a mixed integer problem, *Discrete Applied Mathematics*, **26**, 1990, 255-270.

[2] R.E.Graham, E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, **4**, 1979, 287326.

Table 1: Comparison between Algorithm 1 and CPLEX with $m = 1$

$Instancen = 40$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1(%)	GAP_2 (%)
40.1	**43320**	4	0.750	**51710**	41174.520	4.95	20.37
40.2	**53320**	3	0.578	**84499**	52685.000	1.19	37.65
40.3	46310	4	0.609	45462	43785.885	5.45	3.69
40.4	**39647**	4	0.593	**41083**	37558.949	5.27	8.58
40.5	66052	5	1.203	56802	52473.135	20,56	7.62
40.6	138229	5	12.297	138130	136074.190	1.56	1.49
40.7	**127837**	5	7.188	**181543**	120406.267	5.81	33.68
40.8	92233	5	10.016	92200	85615.029	7.18	7.14
40.9	**127057**	5	12.812	**131828**	110880.352	12.73	15.89
40.10	**154296**	3	5.422	**203692**	149695.486	2.98	26.51
Average	**88830.1**		5.1468	**102694.9**	83034,901	6.52	19.14

$Instance = 60$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1(%)	GAP_2 (%)
60.1	**103391**	5	2.735	**142289**	99643.300	3.62	29.97
60.2	146876	4	1.297	119853	79851.000	45.63	33.38
60.3	**107608**	5	2.343	**192369**	100576.782	6.53	47.72
60.4	**85586**	4	1.765	**87561**	77452.772	9.50	11.54
60.5	101757	7	4.375	98281	90485.537	11.08	7.93
60.6	**305625**	5	37.829	**314041**	286025.176	6.41	8.92
60.7	**276135**	5	27.157	**320524**	242412.339	12.21	24.37
60.8	425790	9	46.360	395662	393970.541	7.47	0.43
60.9	235358	5	31.141	217826	216147.432	8.16	0.77
60.10	**228329**	5	26.500	**247768**	219715.290	3.77	11.32
Average	**201645.5**		18.150	**213617.4**	180628.017	10.42	15.44

$Instancen = 80$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1(%)	GAP_2 (%)
80.1	**151173**	4	7.297	**168084**	142212.646	5.93	15.39
80.2	**114351**	4	5.61	**117651**	106465.567	6.90	9.51
80.3	**197566**	5	4.266	**205009**	178696.958	9.55	12.83
80.4	**158556**	4	5.859	**174975**	148885.270	6.10	14.91
80.5	140620	4	5.781	139064	134850.896	4.10	3.03
80.6	436606	4	88.75	424506	404193.573	7.42	4.78
80.7	**371542**	6	66.178	**377821**	346234.718	6.81	8.36
80.8	**498690**	6	88.438	**537149**	479278.132	3.89	10.77
80.9	472060	5	92.094	468070	456885.464	3.21	2.39
80.10	**410939**	7	124.781	**444147**	366955.883	10.70	17.38
Average	**295210.3**		48.905	**305647.6**	276465,911	6.35	9.55

Table 2: Comparison between Algorithm 1 and CPLEX with $m = 2$

$Instancen = 40$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1**(%)**	GAP_2 (%)
40.1	**15558**	5	0.688	**15573**	15485.500	0.47	0.56
40.2	**40777**	3	0.390	**56197**	24458.800	40.02	56.48
40.3	**20962**	2	0.391	**20962**	**20962.000**	0.00	0.00
40.4	**10774**	4	0.391	**14968**	9703.958	9.93	35.19
40.5	**20635**	3	0.547	**21448**	20331.220	1.47	5.21
40.6	62267	4	5.594	61931	61603.429	1.07	0.53
40.7	47113	4	4.047	45646	45610.556	3.19	0.08
40.8	28297	3	3.359	27209	26973.167	4.68	0.87
40.9	**48362**	4	3.937	**106144**	44184.667	8.64	58.37
40.10	**711386**	3	2.015	**106541**	51365.920	27.79	51.79
Average	**36588.3**		2.136	**47661.9**	32067.920	9.73	20.90
$Instance = 60$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1**(%)**	GAP_2 (%)
60.1	58193	2	1.313	48210	41181.066	29.23	14.58
60.2	**32682**	4	1.109	**42708**	29000.546	11.26	32.10
60.3	**42980**	3	1.109	**106721**	29148.500	32.18	72.69
60.4	**42702**	3	0.907	**69280**	35206.188	17.55	49.18
60.5	**44850**	3	1.203	**48083**	42437.618	5.38	11.74
60.6	166901	4	11.782	148260	143213.800	14.19	3.40
60.7	**118189**	4	12.719	**118476**	114229.902	3.35	3.58
60.8	**235263**	3	5.750	NA	192238.617	18.29	NA
60.9	**76133**	3	5.313	NA	69745.571	8.39	NA
60.10	94210	4	13.140	94208	93931.810	0.30	0.29
Average	**75088.4**		5.410	**84493.3**	66043.680	14.18	23.45
$Instancen = 80$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1**(%)**	GAP_2 (%)
80.1	**63244**	5	4.156	**63543**	61884.476	2.15	2.61
80.2	**45346**	5	5.406	**45515**	40859.412	9.89	10.23
80.3	87370	5	5.140	86964	79537.072	8.97	8.54
80.4	**60761**	3	2.641	**88915**	58694.052	3.40	33.99
80.5	**62905**	3	2.672	**82472**	51179.486	18.64	37.94
80.6	**201140**	5	26.681	**206425**	194438.654	3.33	5.81
80.7	140571	5	26.250	140366	137070.218	2.49	2.35
80.8	**277727**	3	13.171	NA	238847.972	13.99	NA
80.9	**281009**	2	15.575	NA	192435.624	31.52	NA
80.10	**183856**	4	27.750	**189340**	177302.436	3.56	6.36
Average	**105649.1**		12.587	**112942.5**	100120.700	6.55	13.48

Table 3: Comparison between Algorithm 1 and CPLEX with $m = 3$

$Instancen = 40$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1**(%)**	GAP_2 (%)
40.1	**11620**	3	0.375	**16433**	11424.400	1.68	30.48
40.2	**25558**	2	0.328	**26110**	21653.000	15.28	17.07
40.3	**19112**	2	0.328	**19290**	18533.000	3.03	3.92
40.4	**7011**	2	0.281	**7011**	**7011.000**	0.00	0.00
40.5	**16087**	2	0.375	**24556**	14356.000	10.76	41.54
40.6	**47505**	3	1.985	**60117**	47360.000	0.31	21.22
40.7	36744	2	3.812	36169	36012.000	1.99	0.43
40.8	20488	3	3.453	20474	20336.500	0.74	0.67
40.9	**35566**	2	2.218	**35566**	**35566.000**	0.00	0.00
40.10	**39393**	2	1.610	**82527**	35893.071	8.88	56.51
Average	**25908.4**		1.477	**32825.3**	24814.500	4.27	17.18

$Instance = 60$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1**(%)**	GAP_2 (%)
60.1	**35028**	2	0.937	**48131**	33831.000	3.42	29.71
60.2	**23927**	2	0.703	**24280**	22015.667	7.99	9.33
60.3	**20095**	2	0.938	NA	19489.500	3.01	NA
60.4	**31956**	3	0.828	NA	31819.000	0.43	NA
60.5	**54859**	4	1.609	**54859**	36387.136	33.67	33.67
60.6	**127663**	5	6.546	**167944**	125582.667	1.63	25.22
60.7	98859	4	6.719	98661	94465.500	4.44	4.25
60.8	**165403**	3	5.156	**208397**	165206.750	0.12	20.72
60.9	**53307**	3	4.484	**82853**	52855.200	0.85	36.21
60.10	**82501**	2	5.000	NA	82501.000	0.00	NA
Average	**79863.71**		3.736	**97875**	75763.420	7.45	22.73

$Instancen = 80$	Val_{DCA}	it	$time_{DCA}$	Val_{CPLEX}	LB	GAP_1**(%)**	GAP_2 (%)
80.1	**51322**	3	1.968	NA	50627.375	1.35	NA
80.2	**33228**	3	1.578	NA	32177.167	3.16	NA
80.3	81255	3	2.109	74933	67514.000	16.91	9.90
80.4	**53244**	2	1.640	NA	51148.500	3.94	NA
80.5	**37242**	3	2.297	**58271**	37105.393	0.37	36.32
80.6	166973	4	17.234	165230	164133.774	1.70	0.66
80.7	**108115**	3	12.093	**166474**	106583.680	1.42	35.98
80.8	215892	3	14.078	215892	215809.000	0.04	0.04
80.9	**152880**	2	11.532	**152979**	152766.500	0.07	0.14
80.10	**159153**	3	11.375	**203835**	157810.358	0.84	22.58
Average	**131644.3**		10.103	**148230.6**	128817.500	3.05	15.09

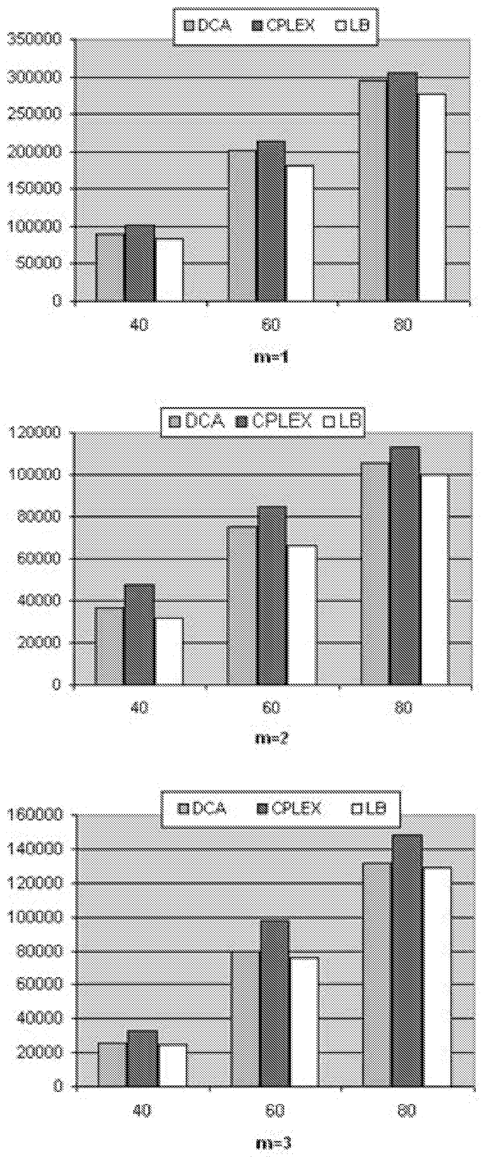

Figure 1: Diagram comparing average objective values

[3] J. A. Hoogeveen, Multicriteria scheduling, *European Journal of Operational Research*, **167**, 2005, 592623.

[4] J. A. Hoogeveen, S. L. Van de Velde, A branch and bound algorithm for single machine earliness tardiness scheduling with idle time, *INFORMS Journal on Computing*, **8**, 1996, 402-412.

[5] Y. D. Kim, C. Yano, Minimizing mean tardiness and earliness in single machine scheduling problems with unequal due dates, *Naval Research Logistics*, **41**, 1994, 913-933.

[6] N. Krause, Y. Singer, Leveraging the margin more carefully, *Proceedings of the twenty-first international conference on Machine learning ICML 2004*. Banff, Alberta, Canada, 2004, 63. ISBN:1-58113-828-5.

[7] Le Thi Hoai An, D.C. Programming for Solving a Class of Global Optimization Problems via Reformulation by Exact Penalty, *Lecture Notes in Computer Science, Springer Berlin*, 2003, 87-101.

[8] Le Thi Hoai An, Pham Dinh Tao, DC Programming: Theory, Algorithms and Applications. The state of art, *The first international workshop on global constrained optimization constrained Satisfaction (Cocos'02), Valbonne-Sophia Antipolis, France*, October 2-4, 2002, 28 p.

[9] Le Thi Hoai An, Pham Dinh Tao, Solving a class of linearly constrained indefinite quadratic problems by DC algorithms,*Journal of Global Optimization*,**11**, 1997, 253-285.

[10] Le Thi Hoai An, Pham Dinh Tao, The DC (difference of convex functions) Programming and DCA revisited with DC models of real world non convex optimization problems, *Annals of Operations Research*, **133**, 2005, 23-46.

[11] Le Thi Hoai An, Pham Dinh Tao, Le Dung Muu, Exact Penalty in DC Programming, *Vietnam Journal of Mathematics*, **2**(2), 1999, 169-178.

[12] Y. Liu, X. Shen, H. Doss, Multi category ψ-Learning and Support Vector Machine: Computational Tools, *Journal of Computational and Graphical Statistics*, **14**, 2005, 219-236.

[13] Pham Dinh Tao, Le Thi Hoai An, Convex analysis approach to DC program ming: Theory, Algorithms and Applications, *Acta Mathematica Vietnamica, dedicated to Professor Hoang Tuy on the occassion of his 70th birthday*, **22**(1), 1997, 289-357.

[14] Pham Dinh Tao, Le Thi Hoai An, DC optimization algorithms for solving the trust region subproblem, *SIAM J. Optimization*, **8**, 1998, 476-505.

[15] C. Ronan, S. Fabian, W. Jason, B. Léon, Trading Convexity

for Scalability, *Proceedings of the 23rd international conference on Machine learning ICML 2006.* Pittsburgh, Pennsylvania, 2006, 201-208, ISBN:1-59593-383-2.

[16] T. Schüle , C. Schnörr , S. Weber , Variational reconstruction with DC-programming, In: G.T. Herman and A. Kuba (Eds) *Advances in Discrete Tomography and Its Applications,* (Birkhäuser, Boston), 2007.

[17] T. Schüle , C. Schnörr , S. Weber and J. Hornegger, Discrete tomography by convex-concave regularization and d.c. programming, *Discrete Applied Mathematics*, **151**, 2005, 229-243.

[18] F. Sourd, New exact algorithms for one machine earliness tardiness scheduling, *INFORMS Journal of Computing*, **21**, 2009, 167-175

[19] F. Sourd, S. Kedad-Sidhoum, Set of instances for Earliness-tardiness scheduling with distinct due dates, *http://www-poleia.lip6.fr/∼sourd/project/et/*

[20] F. Sourd , S. Kedad-Sidhoum A faster branch-and-bound algorithm for the earliness tardiness scheduling problem, *Journal of Scheduling*, **11**, 2008, 49-58.

[21] F. Sourd, S. Kedad-Sidhoum, The one machine problem with earliness and tardiness penalties, *Journal of Scheduling*, **6**, 2003, 533-549.

[22] J. P. Sousa, L. A. Wolsey, A time-indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, **54**, 1992, 353-367.

[23] V. T'kindt, J.C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*, second ed., Springer-Verlag, 2006.

[24] Van den Akker, M., H. Hoogeveen, S. Van de Velde, Combining column generation and lagrangean relaxation to solve single machine common due date problem, *INFORMS Journal on Computing*, **14**, 2002, 35-51.

[25] Van den Akker M., C.A.J. Hurkens, M.W.P. Savelsbergh, Time-indexed formulations for machine scheduling problems: column generation, *INFORMS Journal on Computing*, **12**, 2000, 111-124.

[26] H. Yau, Y. Pan, L. Shi New solution approaches to the general single machine earliness tardiness problem, *Automation Science and Engineering, IEEE Transactions*, **5**(2), 2008, 349-360.

[1] *Laboratory of Theoretical and Applied Computer Science*
University Paul Verlaine of Metz
Île du Saulcy, Metz 57045, FRANCE
E-MAIL: {lethi,thuan}@univ-metz.fr

[2] *Laboratory of Informatics*
University of Franois Rebalais,
Tours FRANCE
E-MAIL: nguyen1846@yahoo.com

[3] *Laboratory of Modeling, Optimization and Operations Research*
National Institute for Applied Science of Rouen
Mont Saint Aignan 76131, FRANCE
E-MAIL: pham@insa-rouen.fr