A reinforced Lagrangean relaxation for non-preemptive single machine problem. Application to the earliness-tardiness common due date criterion.

Francis Sourd
CNRS – LIP6 laboratory, Paris, France
e-mail: Francis.Sourd@lip6.fr

Keywords: Time-indexed formulation, Lagrangean relaxation, earliness-tardiness scheduling, common due date.

1 Introduction

The time-indexed formulation [3, 6] is a well-known way to formulate a single-machine scheduling problem as a mixed-integer scheduling problem (MIP). It is based on time discretization: time is divided into unary *periods* (or *time slots*). The main advantage of this integer program is that the linear relaxation is very good. However, as the number of time-points is pseudo-polynomial in the size of the input, the resulting MIP is usually very large and is not directly tractable by MIP solvers (such as ILOG CPLEX) for large instances.

Therefore, much research effort was devoted to approaches based on this formulation, which enable to fastly derive good upper and lower bounds. Most approaches are either Lagrangean relaxations or column generation methods. In this paper, we consider the Lagrangean relaxation approach. In the litterature, it has been illustrated that we can either relax the resource constraints or the constraints that force each job to be executed once. We are going to consider this latter relaxation, which was also considered by Péridy et al. [5].

In this paper, we first present a generic approach to integrate the presence of dominance rules in the relaxation scheme. Indeed, dominance rules are very useful to solve most scheduling problems, especially when a sum of penalties has to be minimized. We also apply this approach to solve the one-machine scheduling problem with generic earliness-tardiness penalties and an arbitrary common due date. We show that all the instances of OR-Library [2] can be solved: whether the due date is large or not, instances with 1000 tasks can be solved in about 1000s. This new algorithm significantly improves the approach of van den Akker et al. [1] which reports solving instances with up to 125 jobs (and the approach is limited to a large common due date).

Section 2 presents the reinforced Lagrangean lower bound. Section 3 is devoted to its specialization to solve the common due date problem and experimental results are given.

2 Reinforced Lagrangean relaxation

We first formally introduce the scheduling problem and its time-indexed MIP formulation. Let $\mathcal{J} = \{1, \dots, n\}$ be the set of jobs to be scheduled on a single machine. Let $p_j \in \mathbb{N}$ denote the processing time of job j and let c_{jt} be the *processing cost* of j if it completes at time t. We also assume that we know the scheduling horizon, denoted by T, thus we consider the time-periods $1, 2, \cdots, T$. For the simplicity of the presentation, we introduce neither release dates nor precedence constraints but the approach can be easily adapted to deal with these constraints. Let x_{jt} be a binary variable equal to 1 if the task j completes at time t and 0 otherwise. We also have the variable x_{0t} which is equal to 1 if and only if the machine is idle between $t-1$ and t.

$$
\begin{aligned}
\min \quad & \textstyle\sum_{j\in\mathcal{J}} \sum_{t=1}^{T} c_{jt} x_{jt} & & (1)\\
\text{s.t.} \quad & \textstyle\sum_{t=1}^{T} x_{jt} = 1 & \forall\, j \in \mathcal{J} & \quad (2)\\
& \textstyle\sum_{j=0}^{n} \sum_{s=t}^{t+p_i-1} x_{js} = 1 & \forall\, t \in [1, T] & \quad (3)\\
& x_{jt} \in \{0,1\} & \forall\, j \in \mathcal{J},\ \forall\, t \in [r_j, T-p_j] & \quad (4)
\end{aligned}
$$

The objective function (1) indicates that the sum of all the processing costs is to be minimized. Equations (2) ensure that each job is proceeded once. Inequalities (3), also refered to as *resource constraints*, state that at most one job can be handled at any time. At some time slots, the load of the machine can be null, which means that the machine is idle ($x_{0t} = 1$). We now consider a well-known dominance rule and we code it as a linear constraint that we add to our MIP as a redundant constraint. The role of this redundant constraint is precisely to reinforce the Lagrangean relaxation that will be introduced next.

In a feasible schedule S, let us assume that job i completes at t and that job j starts at t (i or j may be equal to 0). The cost of the two adjacent jobs is $c_{it} + c_{j,t+p_j}$. If we swap the pair of jobs (the start times of the other jobs are left unchanged), their cost is then $c_{i,t+p_j-p_i} + c_{j,t+p_j}$ and, if the latter cost is less than the former, S is clearly dominated. Such dominated schedules can be removed from the solution polyhedron of our MIP by adding the linear constraints

$$
x_{it} + x_{j,t+p_j} \leq 1 \qquad \text{if } \begin{cases} c_{it} + c_{j,t+p_j} > c_{i,t+p_j-p_i} + c_{j,t+p_j} \\ c_{it} + c_{j,t+p_j} = c_{i,t+p_j-p_i} + c_{j,t+p_j} \text{ and } i \geq j \end{cases} \tag{5}
$$

We now consider the Lagrangean relaxation of the equations (2). Lagrangean multipliers $(\lambda_1, \ldots, \lambda_n)$ are introduced and, for given values of these multipliers, the Lagrangean problem is to solve

$$\begin{array}{ll} \min & \left(\sum_{j\in\mathcal{J}} \sum_{t=1}^{T} (c_{jt} - \lambda_j) x_{jt}\right) + \sum_j \lambda_j \\ \text{s.t.} & \qquad (3),\ (4) \text{ and } (5) \end{array} \tag{6}$$

We can show that the problem is solved as a shortest path in the following directed acyclic graph. The nodes are indexed by the pair (i, t) and the arcs are defined between the nodes (i, t) and $(j, t + p_j)$ unless a dominance given by equation (5) is defined (we set $p_0 = 1$ to deal with idle time). The length of the arc is set to $c_{jt} - \lambda_j$ if $j > 0$ or is null if $j = 0$. As the graph contains $O(n^2T)$ arcs, the Lagrangean problem is solved in $O(n^2T)$ time. In order to maximize the lower bound $L(\lambda)$ given by the relaxed problem, we classicaly use a subgradient method to find good multipliers.

3 The common due date problem

We test our approach to minimize the total weighted earliness and tardiness (TWET) with arbitrary weights. All jobs have the same due date d. Each job has an earliness penalty α_i and a tardiness penalty β_i, that is $c_{it} = \max(\alpha_i(d - t), \beta_i(t - d))$. According to some strong dominance properties [4], there are two classes of dominant schedules. Basically, either there is some idle time before the start of the schedule (we say d is large) or there is no idle time in the schedule (d is restrictive). In both cases, the tardy jobs are scheduled in the order of the non-decreasing p_i/β_i starting at d and the early jobs are scheduled backward before d in the order of the non-decrease p_i/α_i. In the first case, each job is wholly scheduled either before or after d while in the second case, there may be a *straddling* job that starts before d and completes after d.

Using these dominances, we can show that the Lagrangean problem can be solved in $O(nT)$ time. We compute a lower bound for each of the two cases and take the lower one. When d is large, it corresponds to the Lagrangean relaxation presented by van den Akker et al.[1]. When d is restrictive, we use a forward and a backward dynamic programming to schedule the minimal cost of the early and tardy jobs respectively and we combine them by enumerating all the possible straddling jobs. We also implemented a simple Lagrangean heuristic, that builds a feasible solution using the results of the dynamic programs.

In all our experiments, the algorithm remarkably converges in the sense that the best lower bound found is always equal to the upper bound given by the heuristic. In other words, no branching was necessary to prove the optimality of the instances (but it would be in theory). These results corroborate the observation of van den

n	$h = 0.2$			$h = 0.4$			$h = 0.6$			$h = 0.8$		
	% solved	Avg time	Max time	% solved	Avg time	Max time	% solved	Avg time	Max time	% solved	Avg time	Max time
50	100%	0,12	0.15	100%	0.15	0,28	100%	0,14	0.21	100%	0,12	0.17
100	100%	0,67	0.94	100%	0,85	0.99	100%	1,08	1.92	100%	1,02	1.69
200	100%	5,90	7.40	100%	7,19	8.72	100%	7,83	10.9	100%	7,56	10.2
500	100%	64,4	78.7	100%	90,4	105	100%	98,9	139	100%	101,1	150
1000	100%	611	850	100%	794	1027	100%	915	1321	100%	918	1394

Table 1: CPU time (in seconds) to solve the instances of the OR-library

Akker et al. [1] and extend them to the general common due date problem. However, as their approach —which is in fact mainly based on column generation— can solve instances with up to 125 jobs, our "pure" Lagrangean approach is significantly faster and requires less memory. Moreover, our approach is not limited to the large due date case and we were able to solve all the instances of the OR-Library which contains instances with up to 1000 jobs (see Table 1).

References

[1] M. van den Akker, H. Hoogeveen and S. van de Velde (2002). Combining Column Generation and Lagrangean Relaxation to Solve a Single-Machine Common Due Date Problem. *INFORMS Journal on Computing* **14**, 35–51.

[2] J.E.Beasley (1990). OR-Library: distributing test problems by electronic mail, *Journal of the Operational Research Society* **41** 1069–1072. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

[3] M.E. Dyer and L.A. Wolsey (1990). Formulating the single machine sequencing problem with release dates as a mixed integer problem, *Discrete Applied Mathematics* **26**, 255–70.

[4] N.G. Hall, W. Kubiak and S.P. Sethi (1991). Earliness-tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. *Operations Research* **39**, 847-856.

[5] L. Péridy, E. Pinson and D. Rivreau (2003). Using short-term memory to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research* **148**, 591–603.

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

10 jobs

n	k	h=0.2		h=0.4		h=0.6		h=0.8	
10	1	**1882**	*0,03*	**1009**	*0,02*	**841**	*0,01*	**818**	*0,02*
10	2	**1001**	*0,02*	**615**	*0,01*	**615**	*0,01*	**615**	*0,02*
10	3	**1586**	*0,01*	**917**	*0,02*	**793**	*0,01*	**793**	*0,02*
10	4	**2169**	*0,02*	**1180**	*0,01*	**815**	*0,02*	**803**	*0,02*
10	5	**1149**	*0,01*	**619**	*0,02*	**521**	*0,02*	**521**	*0,01*
10	6	**1469**	*0,02*	**891**	*0,02*	**755**	*0,01*	**755**	*0,02*
10	7	**2102**	*0,02*	**1362**	*0,01*	**1083**	*0,02*	**1083**	*0,02*
10	8	**1680**	*0,02*	**1011**	*0,01*	**610**	*0,02*	**540**	*0,01*
10	9	**1520**	*0,01*	**857**	*0,02*	**582**	*0,02*	**554**	*0,01*
10	10	**1847**	*0,02*	**1097**	*0,01*	**709**	*0,02*	**671**	*0,01*
Av CPU			*0,02*		*0,02*		*0,02*		*0,02*
Max CPU			*0,03*		*0,02*		*0,02*		*0,02*

20 jobs

n	k	h=0.2		h=0.4		h=0.6		h=0.8	
50	1	**4333**	*0,02*	**3066**	*0,02*	**2986**	*0,03*	**2986**	*0,03*
50	2	**8338**	*0,03*	**4797**	*0,03*	**3176**	*0,02*	**2980**	*0,03*
50	3	**6141**	*0,02*	**3820**	*0,03*	**3583**	*0,02*	**3583**	*0,02*
50	4	**9188**	*0,02*	**5118**	*0,02*	**3317**	*0,04*	**3040**	*0,02*
50	5	**4164**	*0,02*	**2465**	*0,02*	**2173**	*0,03*	**2173**	*0,02*
50	6	**6440**	*0,03*	**3533**	*0,02*	**3010**	*0,02*	**3010**	*0,02*
50	7	**10340**	*0,03*	**6172**	*0,03*	**4104**	*0,03*	**3878**	*0,02*
50	8	**3858**	*0,03*	**2106**	*0,02*	**1638**	*0,03*	**1638**	*0,02*
50	9	**3404**	*0,02*	**2074**	*0,02*	**1965**	*0,03*	**1965**	*0,02*
50	10	**4926**	*0,02*	**2920**	*0,03*	**2091**	*0,03*	**1995**	*0,02*
Av CPU			*0,02*		*0,02*		*0,03*		*0,02*
Max CPU			*0,03*		*0,03*		*0,04*		*0,03*

n	k	h=0.2		h=0.4		h=0.6		h=0.8	
50	1	**40483**	*0,12*	**23684**	*0,12*	**17969**	*0,13*	**17934**	*0,09*
50	2	**30455**	*0,15*	**17827**	*0,14*	**14050**	*0,19*	**14040**	*0,13*
50	3	**34425**	*0,11*	**20500**	*0,19*	**16497**	*0,11*	**16497**	*0,1*
50	4	**27601**	*0,11*	**16592**	*0,11*	**14080**	*0,1*	**14080**	*0,11*
50	5	**32137**	*0,13*	**17928**	*0,1*	**14605**	*0,11*	**14605**	*0,11*
50	6	**34782**	*0,13*	**20292**	*0,13*	**14233**	*0,15*	**14066**	*0,15*
50	7	**42879**	*0,1*	**22922**	*0,28*	**17616**	*0,12*	**17616**	*0,11*
50	8	**43630**	*0,14*	**24793**	*0,18*	**21329**	*0,21*	**21329**	*0,15*
50	9	**34010**	*0,08*	**19893**	*0,17*	**14177**	*0,16*	**13942**	*0,12*
50	10	**32958**	*0,12*	**19167**	*0,12*	**14366**	*0,14*	**14363**	*0,17*
Av CPU			*0,12*		*0,15*		*0,14*		*0,12*
Max CPU			*0,15*		*0,28*		*0,21*		*0,17*

100 jobs

n	k	h=0.2		h=0.4		h=0.6		h=0.8	
100	1	**145143**	*0,77*	**85728**	*0,97*	**72017**	*1,04*	**72017**	*1,11*
100	2	**124563**	*0,65*	**72816**	*0,65*	**59230**	*0,74*	**59230**	*0,75*
100	3	**129451**	*0,55*	**79486**	*0,9*	**68537**	*0,8*	**68537**	*0,85*
100	4	**129217**	*0,79*	**79274**	*0,94*	**68759**	*1,52*	**68759**	*1,53*
100	5	**124012**	*0,7*	**71130**	*0,88*	**55286**	*1,37*	**55103**	*0,74*
100	6	**138769**	*0,94*	**77618**	*0,99*	**62398**	*1,92*	**62398**	*1,69*
100	7	**134650**	*0,54*	**78078**	*0,66*	**62197**	*0,81*	**62197**	*0,77*
100	8	**160147**	*0,61*	**94365**	*0,74*	**80708**	*0,64*	**80708**	*0,66*
100	9	**116188**	*0,58*	**69335**	*0,95*	**58727**	*1,16*	**58727**	*1,28*
100	10	**118595**	*0,52*	**71749**	*0,78*	**61361**	*0,76*	**61361**	*0,79*
Av CPU			***0,67***		***0,85***		***1,08***		***1,02***
Max CPU			*0,94*		*0,99*		*1,92*		*1,69*

200 jobs

n	k	h=0.2		h=0.4		h=0.6		h=0.8	
200	1	**497941**	*4,72*	**295437**	*7,14*	**254259**	*7,35*	**254259**	*7,81*
200	2	**540424**	*6,24*	**318897**	*6,88*	**266002**	*6,47*	**266002**	*6,99*
200	3	**487958**	*6,33*	**293627**	*8,72*	**254476**	*10*	**254476**	*10,19*
200	4	**585482**	*5,53*	**352722**	*8,1*	**297109**	*8,38*	**297109**	*7,8*
200	5	**512516**	*6,82*	**304375**	*7,11*	**260278**	*7,81*	**260278**	*7,8*
200	6	**477311**	*6,02*	**279652**	*5,28*	**235702**	*5,24*	**235702**	*5,3*
200	7	**454757**	*7,4*	**275017**	*7,59*	**246307**	*6,99*	**246307**	*6,21*
200	8	**493529**	*5,89*	**278871**	*6,38*	**225215**	*7,58*	**225215**	*6,83*
200	9	**529275**	*4,41*	**310400**	*7,14*	**254637**	*7,6*	**254637**	*6,76*
200	10	**537604**	*5,66*	**322778**	*7,57*	**268353**	*10,87*	**268353**	*9,85*
Av CPU			***5,9***		***7,19***		***7,83***		***7,56***
Max CPU			*7,4*		*8,72*		*10,87*		*10,19*

500 jobs

n	k	h=0.2		h=0.4		h=0.6		h=0.8	
500	1	**2954852**	*63,51*	**1787693**	*79,44*	**1579031**	*87,98*	**1579031**	*92,42*
500	2	**3365830**	*55,28*	**1994771**	*105,25*	**1712195**	*90,59*	**1712195**	*89,38*
500	3	**3102561**	*65,16*	**1864365**	*101,22*	**1641438**	*106,73*	**1641438**	*111,81*
500	4	**3221011**	*62,54*	**1887284**	*77,58*	**1640783**	*80,44*	**1640783**	*81,43*
500	5	**3114756**	*60,39*	**1806978**	*79,72*	**1468231**	*77,39*	**1468231**	*72,57*
500	6	**2792231**	*65,16*	**1610015**	*84,7*	**1411830**	*86,09*	**1411830**	*89,75*
500	7	**3172398**	*62,03*	**1902617**	*94,82*	**1634330**	*87,48*	**1634330**	*88,13*
500	8	**3122267**	*78,68*	**1819185**	*92,31*	**1540377**	*115,38*	**1540377**	*117,01*
500	9	**3364310**	*56,02*	**1973635**	*91,06*	**1680187**	*139,65*	**1680187**	*150,15*
500	10	**3120383**	*75,11*	**1837325**	*97,59*	**1519181**	*117,41*	**1519181**	*118,15*
Av CPU			***64,39***		***90,37***		***98,91***		***101,08***
Max CPU			*78,68*		*105,25*		*139,65*		*150,15*

1000 jobs

n	k	h=0.2		h=0.4		h=0.6		h=0.8	
1000	1	**14054917**	*592,63*	**8110892**	*1027,43*	**6410875**	*799,31*	**6410875**	*763,85*
1000	2	**12295997**	*503,23*	**7271371**	*743,28*	**6110091**	*878,89*	**6110091**	*804,98*
1000	3	**11967282**	*552,48*	**6986816**	*715,31*	**5983303**	*891,96*	**5983303**	*831,04*
1000	4	**11796594**	*662,88*	**7024050**	*716,95*	**6085846**	*799,28*	**6085846**	*795,77*
1000	5	**12449586**	*850,48*	**7364795**	*738,39*	**6341477**	*1321,65*	**6341477**	*1394,38*
1000	6	**11644085**	*475,23*	**6927584**	*883,83*	**6078373**	*1122,39*	**6078373**	*1225,06*
1000	7	**13276996**	*669,94*	**7861297**	*787,78*	**6574297**	*899,3*	**6574297**	*905,29*
1000	8	**12274736**	*640,39*	**7222137**	*780,42*	**6067312**	*819,72*	**6067312**	*831,95*
1000	9	**11757063**	*554,45*	**7058766**	*826,7*	**6185321**	*782,75*	**6185321**	*800,86*
1000	10	**12427441**	*616,1*	**7275933**	*722,73*	**6145737**	*829,94*	**6145737**	*831,99*
Av CPU			***611,78***		***794,28***		***914,52***		***918,52***
Max CPU			*850,48*		*1027,43*		*1321,65*		*1394,38*