J Sched (2016) 19:33–42
DOI 10.1007/s10951-015-0459-4

MP or not MP: that is the question

Federico Della Croce[1,2]

Published online: 3 November 2015

Abstract It is well known that in the twentieth century, mathematical programming (MP) modeling and particularly linear programming (LP) modeling, even though strongly applied to combinatorial optimization, were not too successful when directed to scheduling problems. The purpose of this paper is to show that the field of successful applications of LP/MP modeling is still growing and includes also scheduling topics. We first focus on single machine scheduling. We consider a single machine scheduling model where a quadratic programming (QP) formulation handled by means of a QP solver is shown to be competitive with the state of the art approaches. Also, we discuss a single machine bicriterion scheduling problem and show that a standard LP formulation based on positional completion times performs reasonably well when handled by means of a LP solver. Then, we show how LP can be used to tighten bounds for approximation results in sequencing problems. Finally, we show how to enhance the complexity bounds of branch-and-reduce exact exponential algorithms by means of the so-called measure-and-conquer paradigm requiring always the solution of a specific MP model.

Keywords Modeling · Machine scheduling · Approximation · Exact exponential algorithms

1 Introduction

A combinatorial optimization problem can be typically formulated as a mathematical programming (MP) model by first translating each elementary decision into a variable and by then specifying the objective function and the related constraints. To obtain a linear programming (LP) model, the objective is also required to be a linear function of the variables, and the constraints are required to be linear (in)equalities involving the variables. We refer to integer linear programming (ILP) if the variable is also supposed to be an integer. While modeling can sometimes be considered to be an art, which cannot be reduced to a standard set of procedures, there exist, however, references [see for instance Plastria (2002) and Williams (1990)] that discuss some sort of systematic approach in the LP modeling of combinatorial optimization problems. The purpose of this work is actually to indicate that LP/MP modeling is still a strong tool in combinatorial optimization and is more and more applicable to topics not yet considered in the twentieth century. To this extent, we first show in Sect. 2 that pure sequencing problems can be solved by means of an MP or LP solver with effective performances whenever a proper formulation holds. Also, we show in Sect. 3 that approximation results can be derived by means of the iterated solution of several LP models. Finally, we deal in Sect. 4 with exact exponential branch-and-reduce algorithms for combinatorial problems and show that the so-called *measure-and-conquer* paradigm requires the solution of a MP model to improve the worst-case complexity bound of the exact algorithm.

2 Single machine scheduling by MP/LP modeling

LP and MP modeling for machine scheduling were already discussed (Blazewicz et al. 1991) where various formulations were provided for several well known problems, but at that time, the performances of the LP and MP solvers

✉ Federico Della Croce
federico.dellacroce@polito.it

[1] D.A.I. Politecnico di Torino, Torino, Italy

[2] CNR, IEIIT, Torino, Italy

did not allow to reach competitive results by implementing those formulations. Several years later, LP-based formulations in single machine scheduling were already successful when dealing with the static single machine problem with the objective of minimizing the weighted sum of tardy jobs ($1||\sum w_j U_j$). In MHallah and Bulfin (2003), by indexing the jobs in earliest due date (EDD) order $d_1 \leq d_2 \leq \cdots \leq d_n$, the following mathematical model using 0/1 variables x_j for each job j ($x_j = 1$ if j is early, else $x_j = 0$) was proposed:

$$\max \sum_{j=1}^{n} w_j x_{[j]} \tag{1}$$

$$\sum_{i=1}^{j} p_i x_i \leq d_j \quad \forall j = 1, \ldots, n \tag{2}$$

$$x_j \in \{0, 1\}, \quad \forall j = 1, \ldots, n. \tag{3}$$

The objective summed up the weights of all early jobs. The constraints used the fact that if any sequence has all jobs on time, the EDD sequence does. Indeed, the left-hand side of constraint j is the completion time of job j if it is on time, since all on-time jobs with smaller due dates will be scheduled before it. Hence, if job j is to be on time, its completion time must be no greater than its due date. In Baptiste et al. (2010), a dedicated exact algorithm generalizing to the case with deadlines ($1|\overline{d_j}|\sum w_j U_j$), the above ILP formulation was able to solve problems with up to 30, 000 jobs in size. However, the above problems were rather assignment problems, as the only decision required was to define for each job j whether j was early or tardy, the sequence being univocally determined once the assignment was defined. In Baker and Keller (2010) various LP formulations were compared on the well-known single machine total tardiness problem where also sequencing decisions must be taken into account. The formulation based on positional completion times [see also Lasserre and Queyranne (1992)] reached best results and was able to solve to optimality problems with up to 50 jobs. In Della Croce et al. (2014), the $1|r_j|\sum C_j$ problem was considered. For that problem, a formulation based on positional completion times turned out to be the best one with respect to other classical formulations. The application of CPLEX 12.3 to that formulation enhanced by valid cuts based on the findings of Della Croce and T'kindt (2003) managed to reach good results solving to optimality in limited time instances with up to 100 jobs in size. Further, the same formulation embedded into a matheuristic approach showed to be superior to state-of-the-art heuristic approaches. Here, we consider two single machine scheduling problems and provide a QP model and an LP model, respectively. Both models showed up very good performances when handled by means of the QP and LP libraries of CPLEX 12.5 solver.

2.1 Minimizing the total weighted completion time with rejection on a single machine

In this first single machine problem, a set of n jobs must be processed sequentially and non-preemptively on a single machine, which is always available. Each job j can be either processed or rejected, but in the latter case, a rejection cost re_j incurs. The objective is to minimize the summation of the rejection penalties of the removed jobs added to the total weighted completion time of the scheduled jobs. The problem can be denoted as $1|REJ|\sum w_j C_j + \sum re_j$. Engels et al. (2003) showed that the $1|REJ|\sum w_j C_j + \sum re_j$ problem is ordinary NP-hard and proposed two exact pseudo-polynomial dynamic programming (DP) algorithms running in $O(n\sum_{i=1}^{n} p_i)$ time and in $O(n\sum_{i=1}^{n} w_i)$ time, respectively. In Moghaddam et al. (2012), the bicriterion problem $1|rej|\sum w_j C_j, \sum re_j$ was considered. There, both the total weighted completion time of the selected jobs and the total rejection cost must be minimized, and the objective is to search for the Paretian solutions. Several MP/LP models were introduced, but only an LP formulation was tested that required already a significant amount of CPU time on problems with just 15 jobs.

2.1.1 Problem formulation

We recall that when no jobs are rejected, that is, for problem $1||\sum w_j C_j$, the optimal sequence is the weighted shortest processing time (WSPT) sequence in which the jobs are sequenced in non-decreasing order of their p_j/w_j values. Correspondingly, for the $1|REJ|\sum w_j C_j + \sum re_j$ problem, all selected jobs will be sequenced according to the WSPT order. Assume that the jobs are indexed according to the WSPT order with ties broken according to SPT. Let denote by C_j the completion time of job j and by x_j a 0/1 variable where $x_j = 1$ if j is selected, else $x_j = 0$. Then, for each selected job j, we have $C_j = \sum_{i=1}^{j} p_i x_i$. Thus, if a job is selected, its contribution to the objective function is given by $w_j C_j x_j = \sum_{i=1}^{j} p_i w_i x_i x_j$. In addition, if a job is rejected, its contribution to the objective function is given by $re_j(1 - x_j)$. Correspondingly, as indicated in Moghaddam et al. (2012), the following MP formulations holds:

$$\min \sum_{j=1}^{n} \sum_{i=1}^{j} p_i w_i x_i x_j + \sum_{j=1}^{n} re_j \left(1 - x_j\right) \tag{4}$$

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, n. \tag{5}$$

2.1.2 Computational experiments

The proposed MP formulation (4)–(5) was provided, but, strangely enough, not tested in Moghaddam et al. (2012)

Table 1 Comparing CPLEX 12.5 solving the proposed MP formulation to the DP approach of Engels et al. (2003) on instances with up to 2000 jobs

n	*REJ* distrib.	CPLEX 12.5—CPU time	CPLEX 12.5—# nodes	DP—CPU time
100	D1	0.07	0	0.11
200	D1	0.08	0.5	0.44
500	D1	0.20	0	2.72
1000	D1	1.21	0	11.11
2000	D1	8.21	0.4	44.16
100	D2	0.19	72.7	0.11
200	D2	0.27	73.7	0.45
500	D2	1.20	122.6	2.83
1000	D2	11.20	672.8	11.20
2000	D2	56.07	581.3	44.82
100	D3	0.06	0	0.21
200	D3	0.08	0.2	0.89
500	D3	0.23	0.6	5.30
1000	D3	1.22	0.5	21.53
2000	D3	7.99	2.4	85.81
100	D4	0.17	99.6	0.23
200	D4	0.51	120.9	0.88
500	D4	1.59	317.6	5.48
1000	D4	8.57	355.6	21.57
2000	D4	54.64	609.1	86.70

where a standard linearization was considered requiring already important computation times with 15 jobs. Here, the proposed MP formulation was solved by means of the quadratic programming library of CPLEX 12.5 on an Intel Dual Core with 1.7 GHz CPU and 4 GB RAM. We generated the instances following the distribution plan proposed in Moghaddam et al. (2012). Processing times were randomly generated using the discrete uniform distribution in the range $[10, 80]$ and weights were generated from the discrete uniform distribution in the range $[1, 30]$. Rejection costs were obtained as follows: for each job j, we have $re_j = \exp(5 + \sqrt{\alpha} + \beta)$, where α is randomly generated using the discrete uniform distribution in the range $[1, 80]$ and β is a random number within $[0, 1]$. Since with the considered distribution of the rejection costs (indicated as D1 in Table 1) in all cases CPLEX generated nearly no search tree nodes, we searched for harder instances and determined another distribution (indicated as D2). In distribution D2, for each job j, processing times and weights are as in distribution D1, while for the rejection costs we have $re_j = w_j \frac{\sum_{i=1}^{n} p_i}{3}(1+\gamma)$, where γ is a random number within $[-0.05, 0.05]$.

Since the weights are generally inferior to the processing times, we implemented also the dynamic programming (indicated as DP in Table 1) of Engels et al. (2003) running with complexity $O(n \sum_{i=1}^{n} w_i)$ time, which is obviously faster than the one with complexity $O(n \sum_{i=1}^{n} p_i)$ time. Since the DP algorithm of Engels et al. (2003) is a pseudo-polynomial, we also considered two other distributions (indicated as D3 and D4, respectively). D3 uses the same range of processing times and rejection costs of D1 while weights were generated from the discrete uniform distribution in the range $[1, 60]$ in order to verify the CPU time increase as the weights increase. Similarly, D4 uses the same range of processing times and rejection costs of D2 while weights were generated from the discrete uniform distribution in the range $[1, 60]$. The values $100, 200, 500, 1000, and 2000$ were considered for the job size n, and for each n, 10 instances were generated. In total, then 200 instances were generated and solved.

In Table 1, we report the related computational results comparing the average CPU times required by CPLEX 12.5 (the average number of nodes is also reported) applied to the above MP formulation to the average ones obtained by our implementation of the DP algorithm of Engels et al. (2003).

The results indicate that the MP-based approach is able to solve instances with up 2000 jobs in less than 60 seconds on the average and is globally superior to the DP algorithm even for limited size of the weights. Also, comparing distributions D1–D3 and D2–D4, we notice that the performances of the MP-based approach are substantially unaffected by the weights distributions, while the DP approach computing time increases linearly with the weight distribution increase.

2.2 Minimizing maximum earliness and number of tardy jobs on a single machine

In this second single machine problem, a set N of n jobs must be processed on a single machine, which is always available. Each job j has a processing time p_j and a due date d_j and is available at time zero. The machine can process one job at a time. The objective is to minimize both the maximum earliness and the number of tardy jobs. All the data are integers. In the problem, preemption is not permitted, and the goal is to find all the efficient solutions. The problem is denoted $1||E_{\max}, n_T$ in the extended three-field classification of Tkindt and Billaut (2002). As far as the literature related to the $1||E_{\max}, n_T$ problem is concerned, in Azizoglu et al. (2003), polynomial time algorithms for the maximum earliness problem subject to no tardy jobs and the maximum earliness problem for a given set of tardy jobs were proposed. Also, the authors discussed the use of the latter algorithm in generating all efficient schedules. Notice that the single machine problem with as primary criterion the number of tardy jobs and secondary criterion the maximum earliness

was shown to be strongly NP-hard in Lee and Vairaktarakis (1993). Correspondingly, the $1||E_{\max}, n_T$ is also strongly NP-hard. In Molaee et al. (2010) the $1|nmit|E_{\max}, n_T$ problem was considered, that is, the same problem tackled here, but with the additional constraint that no machine idle time was allowed. For that problem, an exact procedure based on a branch-and-bound approach was proposed and tested on instances with up to 35 jobs. The proposed approach managed to solve to optimality within a CPU time limit of 4000 seconds on a PIV PC with 3.4 GHz CPU and 1 GB RAM all instances with up to 25 jobs but already exceeded the CPU time limit on a couple of instances with 30 jobs. To the author's knowledge, for the $1||E_{\max}, n_T$ no exact approach has been fully computationally tested.

2.2.1 Problem formulation

Let γ be the solution value of the $1||n_T$ problem, that is, the single machine problem with the objective of minimizing the number of tardy jobs. Then, any feasible solution of problem $1||E_{\max}, n_T$ has at most $n - \gamma$ early jobs. In order to derive efficient solutions of problem $1||E_{\max}, n_T$, a standard approach consists in applying the classical ϵ-constraint approach. This corresponds here to consider simply the minimization of the maximum earliness subject to a predefined given number k of early jobs for any value of $k = 1, \ldots, n - \gamma$, and then delete dominated solutions whenever for consecutive values of k the maximum earliness does not change. Let us denote this latter problem as $1|n_T = n - k|E_{\max}$. Notice that for this problem, all late jobs are simply discarded (as idle time is allowed), and there is at least one optimal solution where the last early job completes at its due date (or else it would be possible to postpone it to its due date without increasing the cost function value). Correspondingly, we are interested in the completion times of the first k jobs only, as all the other completion times relate to the discarded jobs. For that reason, we consider the following ILP formulation based on positional completion times variables.

Let $C_{[j]}$ be the completion time of the jth job processed. Let $E_{[j]}$ be the earliness of the jth job processed. Let x_{ij} be a 0/1 variable, where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, k\}$. A variable x_{ij} is equal to 1 if job i is early in position j and 0 otherwise. Let u_i be a 0/1 variable, where $i \in \{1, \ldots, n\}$. A variable u_i is equal to 1 if job i is tardy and 0 otherwise. Finally, let $E_{\max}$ be the maximum earliness to be minimized. The ILP model is as follows:

$$\min E_{\max} \tag{6}$$

$$\sum_{i=1}^{n} u_i = n - k \tag{7}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1, \ldots, k \tag{8}$$

$$\sum_{j=1}^{k} x_{ij} + u_i = 1 \qquad \forall i = 1, \ldots, n \tag{9}$$

$$C_{[1]} \geq \sum_{i=1}^{n} p_i x_{i1} \tag{10}$$

$$C_{[j]} \geq C_{[j-1]} + \sum_{i=1}^{n} p_i x_{ij} \qquad \forall j = 2, \ldots, k \tag{11}$$

$$C_{[k]} = \sum_{i=1}^{n} d_i x_{ik} \tag{12}$$

$$C_{[j]} \leq \sum_{i=1}^{n} d_i x_{ij} \qquad \forall j = 1, \ldots, k-1 \tag{13}$$

$$E_{[j]} \geq \sum_{i=1}^{n} d_i x_{ij} - C_{[j]} \qquad \forall j = 1, \ldots, k \tag{14}$$

$$E_{\max} \geq E_{[j]} \qquad \forall j = 1, \ldots, k \tag{15}$$

$$x_{ij} \in \{0, 1\}, \quad x_{ij} \in \{0, 1\}, \quad C_{[j]} \geq 0, \tag{16}$$

where constraints (7) state that there are exactly k early jobs; constraints (8) state that a job is chosen for each early position; constraints (9) state that each job either occupies an early position or is tardy; constraint (10) sets the completion time of the first job to be non inferior to its processing time; constraints (11) forbid for each job to start before the job in the previous position completes; constraints (12)–(13) impose that the first k jobs must be early and that the kth must complete at its due date; constraints (14) link the earliness of the jth job to its completion time; finally, constraints (15) indicate that $E_{\max}$ is the maximum earliness among the earlinesses of the early jobs.

2.2.2 Computational experiments

In order to test the effectiveness of the proposed LP formulation (6)–(16), we solved it by means of CPLEX 12.5 on an Intel Dual Core with 1.7 GHz CPU and 4 GB RAM. We generated the instances according to the same distribution plan proposed in Molaee et al. (2010). Processing times were randomly generated using the discrete uniform distribution in the range [1, 10] and due dates were generated from the discrete uniform distribution in the range $[0, \rho \sum_{i=1}^{n} p_i]$. Four values of ρ were considered, namely $\rho = \{0.4, 0.6, 0.8, 1\}$.

The values 10, 20, 30, *and*35 were considered for the job size n, and for each combination of ρ and n, 20 instances were generated. In total, then 320 instances were generated and solved. For each instance, $k = n - \gamma$ ILP models were

Table 2 Performances of the ILP based approach on instances with up to 35 jobs

n	ρ	N. found opt.	Avg. CPU time	Max. CPU time
10	0.4	20	0.60	0.84
10	0.6	20	0.68	1.06
10	0.8	20	1.02	1.53
10	1	20	1.12	1.87
20	0.4	20	2.32	3.63
20	0.6	20	3.65	7.27
20	0.8	20	5.85	9.19
20	1	20	6.01	13.84
30	0.4	20	7.32	16.83
30	0.6	20	16.50	84.41
30	0.8	20	69.72	325.65
30	1	20	178.22	876.36
35	0.4	20	18.29	67.95
35	0.6	20	69.15	151.30
35	0.8	19	329.04	1800
35	1	16	801.29	1800

solved by means of CPLEX 12.5 and a CPU time limit of 900 s for each ILP and 1800 for the generation of all the non-dominated solutions.

In Table 2, we report the related computational results providing for each combination of ρ and n the number of instances solved to optimality within the time limit and average and maximum CPU time required.

The results indicate that the approach is viable for medium-size problems as all instances, but 5 were solved to optimality within the time limit. Admittedly, since this is the first exact approach (to the author's knowledge) for the $1||E_{\max}, n_T$ problem, no conclusive statement can be expressed on the effectiveness of the approach. However, keeping in mind that also for the close $1|nmit|E_{\max}, n_T$ problem the approach of Molaee et al. (2010) could handle instances with no more than 35 jobs, we can at least conclude that an exact approach based on positional completion times ILP formulations is often a viable option (when applicable) for single machine scheduling. Notice also that a matheuristic approach on the lines of the one presented in Della Croce et al. (2014) can be immediately devised and is expected to reach high quality results on medium/large-size instances.

3 Computing bounds for approximation results in scheduling

A wide area of research relates to the search for approximation algorithms for $NP\text{-}hard$ scheduling problems. Then, whenever polynomial time approximation schemes are not reachable, it is common to search for polynomial time algorithms capable of reaching constant approximation ratios. If a minimization problem is considered where $f(OPT)$ is the optimal solution value, then we look for a polynomial algorithm A capable of reaching a heuristic solution H such that the ratio $\rho = \frac{f(H)}{f(OPT)}$ is not superior to some given constant k. To prove this, we need to show that, for any instance, algorithm A reaches a solution such that $\rho > k$ can never occur. Then we say that such ratio is *tight* if we are able to find an instance of the problem such that algorithm A performs with ratio $\rho = k$, or it is *asymptotically tight* if we are able to find a family of instances of the problem such that algorithm A performs with ratio ρ that tends asymptotically to k. Or else, we search for an instance that is close enough to the upper bound k previously computed. The search for such an instance is indeed doable by means of LP modeling techniques. In this section, we show, as an example, that LP modeling can be successfully applied to find a lower bound to the approximation ratio in a two-machine flowshop scheduling problem under the re-optimization paradigm.

3.1 Approximating the 2-machine total completion time flowshop problem in the re-optimization setting under job insertion

Consider the solution of machine scheduling problems in the re-optimization setting, which can be described as follows: considering an instance I of a given problem Π for which an optimal solution OPT is provided, and an instance I', which results from a local perturbation of I, can the information provided by OPT be used to solve I' in a more efficient way (i.e., with a lower complexity and/or with a better approximation ratio) than if this information was not available? This is also related to the so-called *reactive scheduling* [see Smith (1994)], that is, what refers to the schedule modifications that may have to be made during project execution starting from a baseline schedule. In some cases, the reactive scheduling effort may rely on very simple techniques aimed at a quick schedule consistency restoration. For this purpose, by considering an optimal schedule as baseline schedule and some jobs additions or deletions as schedule disruptions, we can see that scheduling re-optimization may be considered to be strictly linked to reactive scheduling, particularly with simple re-optimization strategies so that the baseline schedule is only mildly modified.

Here, we study as a simple example the 2-machine total completion time flowshop problem [$F2 \mid\mid \sum C_j$ according to the three-field notation of Lawler et al. (1993)] that can be stated as follows. A set of n jobs is available at time 0 to be processed on 2 machines where each job is processed on machine M_1 first, and then on machine M_2. We denote by $p_{k,j}$ the processing time of job j on machine k ($j = 1, \ldots, n$, $k = 1, 2$). Also, we denote by $C_{k,j}$ the completion time of

job j on machine M_k. Preemption on either machine is not allowed. The objective is the minimization of the sum of completion times on the second machine. It is well known for this problem that there exists at least one optimal solution which is a permutation schedule, that is the jobs share the same sequence on all machines. This problem is known to be NP-hard in the strong sense. We denote by $F2 \mid\mid \sum C_j+$ the re-optimization of version of $F2 \mid\mid \sum C_j$ where a single job x with processing times p_{1x} and p_{2x} is added to an instance already solved to optimality. We analyze the approximation ratios of simple re-optimization strategies that keep unchanged the baseline schedule, namely that merely insert the new job into the initial optimum. We denote by $A_{i,j}$ $(1 \leq i \leq j \leq n+1)$ the re-optimization strategy that consists of computing $j-i+1$ candidate solutions, by inserting the new job in the initial optimum in the k^{th} position for all possible ks between i and j, while leaving the rest of the scheduling unchanged. We provide here the findings of Boria and Della (2014) on this problem. Without loss of generality, the initial optimum is given by schedule $[1, \ldots, n]$ and has value $f(OPT) = \sum_{j=1}^{n} C_{2j}$. On the other hand, the unknown optimal sequence on the modified instance is denoted by OPT' and its value is denoted by $f(OPT')$.

The following propositions [proposed in Boria and Della (2014)] hold.

Proposition 1 *$F2 \mid\mid \sum C_j+$ is approximable within ratio $\frac{3}{2}$ by algorithm $A_{n,n+1}$, and this bound is tight.*

Proposition 2 *The algorithm $A_{1,n+1}$ cannot ensure approximation ratio better than $\frac{7}{5}$ for the $F2 \mid\mid \sum C_j+$ problem.*

In order to show that the tightness of the ratio $\frac{3}{2}$ for algorithm $A_{n,n+1}$ in Proposition 1 and to prove Proposition 2, we need to derive the related instances. In Boria and Della (2014), the related instances were simply provided, here we show how this issue is tackled by means of LP modeling. The rationale of the approach is as follows. Let us denote by $S1_h$ for $h = 1, \ldots, k!$ all permutations of k jobs and let assume without loss of generality that $S1_1 = [1, \ldots, k]$ is optimal, that is, $f(S1_1) = OPT$. On the other hand, let us denote by $S2_h$ for $h = 1, \ldots, (k+1)!$ all permutations of $k+1$ jobs (the first k jobs of the original sequence plus job x).

The proposed approach uses as constraints the requirement that the initial optimal schedule is $S1_1$ [namely $f(S1_h) \geq f(S1_1)$ for $h = 2, \ldots, k$] and the requirement that all schedules reachable by algorithms $A_{n,n+1}$ (for the instance of Proposition 1) or $A_{1,n+1}$ (for the instance of Proposition 2) cannot have a solution value inferior to some given predefined value τ. Finally, the goal is to find a sequence $S2_h$ for $h = 1, \ldots, (k+1)!$ such that $f(S2_h)$ is minimum. The variables are then the jobs processing times [corresponding to $2(k+1)$ variables] and the jobs completion times induced by each sequence [corresponding to $2k \cdot k!$ variables for sequences $S1_h$ and to $2(k+1) \cdot (k+1)!$ variables for sequences $S2_h$]. For any given sequence $S2_\gamma$, this is accomplished by solving a related ILP model.

Let $p(i, j)$ be the processing time of job j on machine i. Let $p(h, i, [j])$ be the processing time of the $j-th$ job on machine i according to sequence h. Let $C(h, i, [j])$ be the completion time of the $j-th$ job on machine i according to sequence h. Let $f(S2_\gamma)$ denote the sum of completion times on the second machine of sequence $S2_\gamma$. All these are integer variables. The LP model is as follows:

$$\min f(S2_\gamma) = \sum_{j=1}^{k+1} C\big(S2_\gamma, 2, [j]\big) \tag{17}$$

$$\sum_{j=1}^{k} C(S1_h, 2, [j]) \geq \sum_{j=1}^{k} C\big(S1_1, 2, [j]\big) \quad \forall h = 2 \ldots k! \tag{18}$$

$$\sum_{j=1}^{k} C\big(S2_h, 2, [j]\big) \geq \tau \quad \forall h \in Q \tag{19}$$

$$C\big(S1_h, 1, [1]\big) = p\big(S1_h, 1, [1]\big) \quad \forall h = 1 \ldots k! \tag{20}$$

$$C\big(S1_h, 1, [j]\big) = C\big(S1_h, 1, [j-1]\big) + p\big(S1_h, 1, [j]\big) \quad \forall h = 1 \ldots k! \, \forall j = 2 \ldots k \tag{21}$$

$$C\big(S1_h, 2, [1]\big) = C\big(S1_h, 1, [1]\big) + p\big(S1_h, 2, [1]\big) \quad \forall h = 1 \ldots k! \tag{22}$$

$$C\big(S1_h, 2, [j]\big) = \max\Big\{C\big(S1_h, 1, [j]\big), C\big(S1_h, 2, [j-1]\big)\Big\} + p\big(S1_h, 1, [j]\big) \tag{23}$$

$$\forall h = 1 \ldots k! \quad \forall j = 2 \ldots k \tag{24}$$

$$C\big(S2_h, 1, [1]\big) = p\big(S2_h, 1, [1]\big) \quad \forall h = 1 \ldots k+1! \tag{25}$$

$$C\big(S2_h, 1, [j]\big) = C\big(S2_h, 1, [j-1]\big) + p\big(S2_h, 1, [j]\big) \quad \forall h = 1 \ldots k+1! \, \forall j = 2 \ldots k \tag{26}$$

$$C\big(S2_h, 2, [1]\big) = C\big(S2_h, 1, [1]\big) + p\big(S2_h, 2, [1]\big) \quad \forall h = 1 \ldots k+1! \tag{27}$$

$$C\big(S2_h, 2, [j]\big) = \max\Big\{C\big(S2_h, 1, [j]\big), C\big(S2_h, 2, [j-1]\big)\Big\} + p\big(S1_h, 1, [j]\big) \tag{28}$$

$$\forall h = 1 \ldots k+1! \quad \forall j = 2 \ldots k \tag{29}$$

matching constraints binding $p(h, i, [j])$'s with $p(i, j)$'s $\forall h, \forall i, \forall j$. (30)

Here, constraints (18) indicate that $S1_1$ is the optimal sequence for the original problem. Also, in (19), Q is the subset of schedules whose cost function value must be $\geq \tau$ as indicated in Propositions 1 and 2. Then, (20–29) are the standard sequencing constraints between adjacent jobs on any given machine or between adjacent operations of the

same job. Notice that here for conciseness we kept in constraints (23, 24) and (28, 29) a nonlinear max notation that can be easily transformed into linear constraints by means of *big-M* coefficients and the introduction of 0/1 variables. Finally, constraints (30) link together the $p(h, i, [j])$ variables with the $p(i, j)$ ones (for instance, if in $S1_1$ job 2 is in first position, then $p(S1_1, i, [1]) = p(i, 2)\ \forall i = 1, 2$).

Then, by iterating the solution of the LP model on all $\gamma = 1, \ldots, (k+1)!$ and taking the minimum value, we get the best possible instance (with $k+1$ jobs where job $k+1$ corresponds to job x). Notice, however, that the size k must be strongly limited.

By means of the above approach, it was possible to derive for Proposition 1 the following original instance with $k = 2$. Here, we have job 1 that has $p_{1,1} = 0$ and $p_{2,1} = 1$, and job 2 that has $p_{1,2} = 1$ and $p_{2,2} = 0$. For this instance, the optimal scheduling is [1, 2] and has value 2. Then, the new job x added has $p_{1,x} = 0$ and $p_{2,x} = 0$. In this case, it is easy to see that the optimal sequence for the new instance with job x is $[x-1-2]$ with value 2, while algorithm $A_{n,n+1}$ reaches in the best case value 3.

On the other hand, it was possible to derive for Proposition 2 the following original instance still with $k = 2$. Here, we have job 1 that has $p_{1,1} = 0$ and $p_{2,1} = 2$, and job 2 that has $p_{1,2} = 1$ and $p_{2,2} = \delta$. Again, for this instance, the optimal scheduling is [1, 2] and has value $4+\delta$. Then, the new job x added has $p_{1,x} = 0$ and $p_{2,x} = 1$. In this case, it is easy to see that the optimal sequence for the new instance with job x is $[x-2-1]$ with value $5+2\delta$, while algorithm $A_{1,n+1}$ reaches in the best case value $7+\delta$.

This approach can be generalized to other sequencing problems whenever the sequences reached by the proposed algorithm can be easily detected. Consider for instance the Traveling Salesman problem (TSP) (Lawler et al. 1985). It is well known that when the triangular inequality holds (the so-called Metric TSP), a tight $\frac{3}{2}$ approximation ratio is reached by means of the algorithm proposed in Christofides (1976). Restricting the edges to have distances (1, 2), we have the so-called $\text{TSP}_{1,2}$. For $\text{TSP}_{1,2}$, it is well known that a $\frac{7}{6}$ approximation ratio is reached by means of the algorithm proposed in Papadimitriou and Yannakakis (1993). On the other hand, among the various heuristics proposed for this problem, the so-called dynasearch approach presented in Potts and Velde (1995) was shown to be a neighborhood search approach capable of finding in polynomial time the best solution of an exponential size neighborhood. It is then of interest to see whether for any given starting solution, the approximation ratio of dynasearch may or not reach an approximation ratio inferior to $\frac{3}{2}$ for Metric TSP and inferior to $\frac{7}{6}$ for $\text{TSP}_{1,2}$. It is possible to apply the above mentioned approach both to metric TSP and $\text{TSP}_{1,2}$ by defining in this case as variables the edges lengths. An initial cycle $C_1 = [1-2-\cdots-n]$ can be assigned without loss of generality with some predefined value $k = f(C_1)$. Let $N_{\text{dyn}}(C_1)$ be the dynasearch neighborhood of C_1. Then, by constraining cycle C_1 to be a local minimum for $N_{\text{dyn}}(C_1)$, that is, by imposing $f(C_1) \leq f(C_j)\ \forall C_j \in N_{\text{dyn}}(C_1)$, it is possible to determine the approximation ratio of dynasearch by iteratively minimizing $f(C_k)\ \forall C_k \notin N_{\text{dyn}}(C_1)$. When n is small enough, this can be easily accomplished. Indeed, two simple examples with six cities depicted in Table 3 for metric TSP and in Table 4 for $\text{TSP}_{1,2}$ were derived with this approach for a dynasearch neighborhood based on the combination of insertion, swap, and twist neighborhoods [see Ergun and Orlin (2006) for a specific description of the related operators]. Correspondingly, a $\frac{3}{2}$ approximation ratio was obtained for the metric TSP and a $\frac{7}{6}$ approximation ratio was obtained for $\text{TSP}_{1,2}$. For the metric TSP, the local minimum is $C_1 = [1-2-3-4-5-6]$ with value $f(C_1) = 6$ and the optimal solution is $C_j = [1-4-5-3-2-6]$ with value $f(C_j) = 4$ where $C_j \notin N_{\text{dyn}}(C_1)$. Similarly, for $\text{TSP}_{1,2}$, the local minimum is always $C_1 = [1-2-3-4-5-6]$ with value $f(C_1) = 7$, while the optimal solution is $C_k = [1-2-4-6-5-3]$ with value $f(C_k) = 6$, where $C_k \notin N_{\text{dyn}}(C_1)$. Hence, the following proposition holds stating that is unlikely to improve the available polynomial time approximation ratios for metric TSP and $\text{TSP}_{1,2}$ by means of dynasearch.

Proposition 3 *The approximation ratio of the dynasearch neighborhood is not inferior to $\frac{3}{2}$ for the Metric TSP and not inferior to $\frac{7}{6}$ for $TSP_{1,2}$.*

Table 3 A 6-town TSP distance where dynasearch has approximation ratio $\frac{3}{2}$ for Metric TSP

	1	2	3	4	5	6
1	–	1	2	1	2	1
2	1	–	1	2	1	0
3	2	1	–	1	0	1
4	1	2	1	–	1	2
5	2	1	0	1	–	1
6	1	0	1	2	1	–

Table 4 A 6-town TSP distance where dynasearch has approximation ratio $\frac{7}{6}$ for $\text{TSP}_{1,2}$

	1	2	3	4	5	6
1	–	1	1	2	2	2
2	1	–	1	1	2	2
3	1	1	–	1	1	2
4	2	1	1	–	1	1
5	2	2	1	1	–	1
6	2	2	2	1	1	–

4 Mathematical programming modeling for the measure-and-conquer paradigm in exact branch-and-reduce algorithms

The design of exact methods for NP-hard combinatorial optimization problems has always been a challenging issue. Among the existing exact methods, search tree algorithms have been widely applied. Here, we study their application in the context of worst-case analysis of exact algorithms where classical search tree algorithms are more commonly defined as branch-and-reduce algorithms as typically a branch in the search tree induces the generation of two or more subproblems each with a reduced number of variables with respect to the original problem. Consider now a combinatorial optimization problem that can be represented by means of n binary variables. Let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on integers. In what follows, using notations in Woeginger (2003), for an integer n, we express running-time bounds of the form $p(n) \cdot T(n)$ as $O^*(T(n))$, the asterisk meaning that we ignore polynomial factors. We denote by $T(n)$ the worst-case time required to exactly solve the considered combinatorial optimization problem with n binary variables. We recall [see, for instance, Eppstein (2001)] that if it is possible to bound above $T(n)$ by a recurrence expression of the type $T(n) \leq \sum T(n-r_i)+O(p(n))$, we have $T(n) = O^*(\alpha(r_1, r_2, \ldots)^n)$, where $\alpha(r_1, r_2, \ldots)$ is the unique positive real root (zero) of the function $f(x) = 1 - \sum x^{-r_i}$ where all $r_i > 0$. As an example, consider an ILP model with only binary variables and suppose that a branching scheme can be constructed such that at each branch at least either 1 or 3 variables can be fixed. Then, we have $T(n) \leq \sum T(n-1) + T(n-3) + O(p(n))$ and we get $T(n) = O^*(\alpha^n)$ with $\alpha =$ largest real root of $1 - \frac{1}{\alpha} - \frac{1}{\alpha^3} = 0$, that is $\alpha^3 = \alpha^2 + 1$, namely $\alpha \approx 1.4656$, and hence $T(n) = O^*(1.4656^n)$. This value can actually be expressed by the following mathematical programming model

$$\min \alpha \tag{31}$$
$$\alpha^3 - \alpha^2 \geq 1 \tag{32}$$
$$\alpha \geq 0 \tag{33}$$

In the context of branch-and-reduce algorithms, the measure-and-conquer paradigm has been proposed in Fomin et al. (2009), which is based upon a more elaborated design of a non-straightforward measure of the subproblems sizes. Such measure is used to lower bound the progress made by the algorithm at each branching step and may exploit behaviors of the algorithm that a standard measure might not be able to get leading typically to an improved worst-case time analysis. Here, we analyze a standard branch-and-reduce algorithm for the maximum independent set (MIS) problem and how it can be strongly improved by means of the measure-and-conquer paradigm that embeds to this extent the solution of a nonlinear mathematical programming model. In the MIS problem, given a graph $G(V, E)$, we search for a maximum cardinality subset $I \subset V$ of vertices that does not induce any edges. It is well known that the MIS problem is polynomially solvable if all vertices have degree ≤ 2 ($\forall i \in V \quad d_i \leq 2$). Alternatively, G contains a vertex j of degree $d_j \geq 3$. Consider a standard branch-and-reduce exact algorithm where a classical binary branching scheme on some vertex j where j is either selected or discarded. The only peculiarity is that we repeat branching on the largest degree vertex j, that is $d_j = d_{\max}$ until $d_j \leq 2, \forall j$ (once $d_j \leq 2, \forall j$, the remaining problem is solved to optimality in polynomial time). Then at each branch, we have

1. $j \notin I$;
2. $j \in I \longrightarrow I$ cannot contain any neighbor of v.

In the first case, one vertex is fixed, while in the second case, $d_j + 1$ vertices are fixed (j plus all neighbors of j). As $d_j \geq 3$, the worst case occurs for $d_j = 3$. Hence, the induced recursion is $T(n) \leq T(n-1)+T(n-4)+O(p(n))$. Correspondingly, by solving the related mathematical programming model

$$\min \alpha \tag{34}$$
$$\alpha^4 - \alpha^3 \geq 1 \tag{35}$$
$$\alpha \geq 0, \tag{36}$$

we get $T(n) = O^*(1, 3803^n)$.

In the above branch-and-reduce algorithm, for each level of the search tree, we define as *fixed* those vertices that have already been selected or discarded, while we define as *free* the other vertices.

Now, consider using the measure-and-conquer paradigm. We do not count in the measure the fixed vertices, and we count with weight $w_h = w_{[d_h]}$ the free vertices h where the vertex j to be selected for branching is the one with largest degree that is with largest weight. We take also into account the following known reduction rule: if we encounter a subproblem with a vertex h having degree 1, then h can be included without loss of optimality in I, and its neighbor can be discarded. Then, we impose

$$0 = w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \cdots \leq w_{[6]} = w_{[n]} = 1 \tag{37}$$

that is, the weights of the free vertices are strictly increasing in their degrees and that all free vertices with degree ≥ 6 have weight $= 1$ (while $w_{[1]} = 0$ as vertices with just one neighbor can be immediately fixed. We also get recurrences on the time $T(p)$ required to solve instances of size p, where

the size of an instance is the sum of the weights of the vertices. Since initially $p \leq n$, the overall running time is expressed as a function of n. This is valid since when $p = 0$, all vertices have been fixed. Now consider the branching. If we discard j, vertex j is fixed and d_j vertices (the neighbors of j) decrease their degree and correspondingly their weight by one unit. On the other hand, if we select j, d_j other vertices (the neighbors of j) are discarded, that is $d_j + 1$ vertices are fixed. Hence, the recurrence becomes

$$T(p) \leq T\left(p - w_{[d_{\max}]} - \sum_{h \in N(j)} \left(w_{[d_h]} - w_{[d_h-1]}\right)\right) + T\left(p - w_{[d_{\max}]} - \sum_{h \in N(j)} w_{[d_h]}\right) + p(n). \quad (38)$$

To compute the worst-case complexity under the measure-and-conquer paradigm, we have to compute the best value of the weights $w_{[2]}, \ldots, w_{[n]}$ such that recurrence (38) is respected for any possible value of $d_{\max}$ with $3 \leq d_{\max} \leq n$ as the value of $d_{\max}$ may change in the various branches of the search tree.

To avoid a combinatorial explosion of the different subcases of (38) with respect to the possible degrees of the neighbors of the vertex j selected for branching, we impose that

$$w_{[j]} - w_{[j-1]} \leq w_{[j-1]} - w_{[j-2]}, \quad j = 3, \ldots, n. \quad (39)$$

where we recall from (37) that $w_{[1]} = 0$ and $w_{[6]} = \ldots = w_{[n]} = 1$. Also, we slightly modify the recurrence. Indeed, from (37) we have that

$$\sum_{h \in N(j)} w_{[d_h]} \geq \sum_{h \in N(j)} w_{[2]} \quad (40)$$

always holds as the smallest weight greater than zero is w_2. Finally, from (37), (39), and (40), we have that the following recurrence always holds

$$T(p) \leq T\left(p - w_{[d_{\max}]} - d_{\max}\left(w_{[d_{\max}]} - w_{[d_{\max}-1]}\right)\right) + T\left(p - w_{[d_{\max}]} - \sum_{h \in N(j)} w_{[2]}\right) \quad (41)$$

where there is no need to consider the recurrence for $d_{\max} \geq 8$ as $w_{[k]} - w_{[k-1]} = 0$ for $k \geq 7$. All together, we need to guarantee that recurrences (41) for $7 \geq d_{\max} \geq 3$, and constraints (37) and (39) are satisfied simultaneously.

This corresponds to a nonlinear programming model of the form:

$$\min \alpha \quad (42)$$

$$\alpha^{w_{[j]}+j*w_{[2]}} - \alpha^{(j*w_{[2]}-j*(w_{[j]}-w_{[j-1]}))} \geq 1, \quad j = 3, \ldots, 7 \quad (43)$$

$$w_{[j]} \leq w_{[j+1]}, \quad j = 2, \ldots, 5 \quad (44)$$

$$w_{[6]} = w_{[7]} = 1, \quad (45)$$

$$w_{[j]} - w_{[j-1]} \leq w_{[j-1]} - w_{[j-2]}, \quad j = 3, \ldots, 6 \quad (46)$$

$$\alpha \geq 0, \quad (47)$$

where constraints (43) correspond to recurrences (41), constraints (44) and (45) correspond to constraints (37), and constraints (46) correspond to constraints (39), respectively. The corresponding optimal solution is $\alpha = 1.3496$. Thus, we have $T(n) = O^*(1.3496^n)$. Notice that if the recurrence (38) is kept for $d_{\max} = 3, 4$ with all possible related (4 for $d_{\max} = 3$ and 15 for $d_{\max} = 4$) subcases, while recurrence (41) is used for $d_{\max} \geq 5$, then we obtain $T(n) = O^*(1.3186^n)$. Notice also that the currently best available exact algorithm (that uses the measure-and-conquer paradigm) for the maximum independent set problem [see Bourgeois et al. (2012) reaches $T(n) = O^*(1.2114^n)$].

5 Conclusions

We have considered in this work the application of MP/LP modeling in various fields related to combinatorial optimization with emphasis on scheduling problems. We have seen that a tailored MP model is very efficient in minimizing the total weighted completion time with rejection on a single machine. Similarly, we have seen that the bicriterion single machine problem of searching for the non-dominated solution when minimizing both the maximum earliness and the number of tardy jobs can be satisfactorily solved by means of a positional completion times LP model. Also, we have discussed the application of LP modeling for tightening bounds in approximation results. Notice that such approach applied here to a scheduling problem under a re-optimization framework and to the Traveling Salesman problem can be easily generalized to sequencing problems where the sequences reached by the proposed algorithm can be easily detected. Finally, an example of MP modeling has been provided within the so-called measure-and-conquer paradigm for exact exponential branch-and-reduce paradigm.

We believe that MP/LP modeling remains a fantastic tool for combinatorial optimizers. Nonetheless, several steps are still to come, for instance, the generation of an efficient MP/LP model for the well known minimum makespan job-

shop problem ($J||C_{\max}$) remains a formidable challenge for the OR community!

References

Azizoglu, M., Koksalan, M., & Koksalan, S. K. (2003). Scheduling to minimize maximum earliness and number of tardy jobs where machine idle time is allowed. *Journal of the Operational Research Society*, *54*, 661–664.

Baker, K. R., & Keller, B. (2010). Solving the single-machine sequencing problem using integer programming. *Computers and Industrial Engineering*, *59*, 730–735.

Baptiste, Ph, Della Croce, F., Grosso, A., & Tkindt, V. (2010). Sequencing a single machine with due dates and deadlines: An ILP-based approach to solve very large instances. *Journal of Scheduling*, *13*, 39–47.

Blazewicz, J., Dror, M., & Weglarz, J. (1991). Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, *51*, 283–300.

Boria, N., & Della Croce, F. (2014). Re-optimization in machine scheduling. *Theoretical Computer Science*, *540–541*, 13–26.

Bourgeois, N., Escoffier, B., Paschos, V Th, & van Rooij, J. M. M. (2012). Fast algorithms for max independent set. *Algorithmica*, *62*, 382–415.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the traveling salesman problem, Technical report, GSIA, Carnegie-Mellon University, Pittsburgh.

Della Croce, F., Salassa, F., & T'kindt, V. (2014). A hybrid heuristic approach for single machine scheduling with release times. *Computers and Operations Research*, *45*, 7–11.

Della Croce, F., & T'kindt, V. (2003). Improving the preemptive bound for the one-machine dynamic total completion time scheduling problem. *Operations Research Letters*, *31*, 142–148.

Engels, D. W., Karger, D. R., Kolliopoulos, S. G., Sengupta, S., Uma, R. N., & Wein, J. (2003). Techniques for scheduling with rejection. *Journal of Algorithms*, *49*, 175–191.

Eppstein, D. (2001). Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction, In *Proceedings of Symposium on Discrete Algorithms, SODA01* (pp. 329–337).

Ergun, O., & Orlin, J. B. (2006). Fast neighborhood search for the single machine total weighted tardiness problem. *Operations Research Letters*, *34*, 41–45.

Fomin, F. V., Grandoni, F., & Kratsch, D. (2009). A measure and conquer approach for the analysis of exact algorithms. *Journal of the ACM*, *56*, 1–32.

Lasserre, J. B., & Queyranne, M. (1992). Generic scheduling polyhedral and a new mixed integer formulation for single machine scheduling. In *Proceedings of the IPCO Conference* (pp. 136–149).

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The traveling salesman problem: A guided tour of combinatorial optimization*. New York: Wiley. (Ed.).

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, & P. H. Zipkin (Eds.), *Logistics of Production and Inventory* (pp. 445–522). Amsterdam: North Holland.

Lee, C. Y., & Vairaktarakis, G. L. (1993). Complexity of single machine hierarchical scheduling: A survey. In P. M. Pardalos (Ed.), *Complexity in numerical optimization* (pp. 269–298). Singapore: World Scientific.

MHallah, R., & Bulfin, R. L. (2003). Minimizing the weighted number of tardy jobs on a single machine. *European Journal of Operational Research*, *145*, 45–56.

Moghaddam, A., Amodeo, L., Yalaoui, F., & Karimi, B. (2012). Single machine scheduling with rejection: Minimizing total weighted completion time and rejection cost. *International Journal of Applied Evolutionary Computation*, *3*, 42–61.

Molaee, E., Moslehi, G., & Reisi, M. (2010). Minimizing maximum earliness and number of tardy jobs in the single machine scheduling problem. *Computers and Mathematics with Applications*, *60*, 2909–2919.

Papadimitriou, C. H., & Yannakakis, M. (1993). The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, *18*, 1–11.

Plastria, F. (2002). Formulating logical implications in combinatorial optimisation. *European Journal of Operational Research*, *140*, 338–353.

Potts, C. N. P., & Van de Velde, S. L. (1995). *Dynasearch: Iterative local improvement by dynamic programming: Part i, the traveling salesman problem, faculty of mathematical studies*. Southampton: University of Southampton.

Smith, S. S. (1994). Reactive scheduling systems. In D. E. Brown & W. T. Scherer (Eds.), *Intelligent scheduling systems* (pp. 155–192). Boston: Kluwer Academic Publishers.

T'Kindt, V., & Billaut, J.-C. (2002). *Multicriteria scheduling: Theory, models and algorithms*. Heidelberg: Springer.

Williams, H. P. (1990). *Model building in mathematical programming*. New York: Wiley.

Woeginger, G. J. (2003). Exact algorithms for NP-hard problems: A survey. In M. Juenger, G. Reinelt, & G. Rinaldi (Eds.), *Combinatorial Optimization: Eureka! You shrink!, volume 2570 of Lecture Notes in Computer Science* (pp. 185–207). New York: Springer.