Tabu Search and Strong Cutting Planes for the Flow Shop Scheduling Problem

Marcus Poggi de Aragão
e-mail: poggi@inf.puc-rio.br
Márcio de Moraes Palmeira
Departamento de Engenharia El'etrica
Pontifícia Universidade Católica do Rio de Janeiro, Brazil
e-mail: marcio@ele.puc-rio.br

PUC-Rio.Inf.MCC 52/98 December, 1998

Abstract: *We present a Tabu Search procedure for the Flow Shop Scheduling problem (FSP) where strong cutting planes are used to implement an intensification mechanism. These cutting planes were proposed by Bellmore and Ratliff for the Set Covering problem and are strong in the sense that they also cut integer feasible solutions, but only solutions that are dominated by a known one. We show how to efficiently construct these cuts for the FSP and propose a penalty scheme to consider their violation in the resulting Tabu Search procedure. Finally, we present computational results, on benchmarks instances proposed by Taillard, which indicate that a significant improvement is obtained.*

Keywords: Tabu Search, Scheduling, Mixed-Integer Programming, Cutting Planes

Resumo: Este trabalho tem como objetivo investigar o emprego de planos de corte "fortes" como mecanismo de intensificação em uma busca heurística. Aplica-se esta idéia a uma Busca Tabu para o *Flow Shop Scheduling Problem* (FSP). Planos de corte "fortes" foram propostos por Bellmore e Ratliff para o Problema de Recobrimento e são fortes porque cortam também soluções inteiras viáveis. Entretanto, estas são necessariamente dominadas por uma solução conhecida. Mostra-se aqui como construir eficientemente estes cortes para o FSP, assim como um esquema de penalidades para considerar sua violação dentro do procedimento de busca tabu resultante. Nos resultados computacionais obtidos pode-se verificar que a utilização destes cortes implica em uma melhoria significativa da busca tabu.

Palavras-chave: Busca Tabu, Escalonamento, Programação Inteira Mista, Planos de Corte

1 Introduction

The Bellmore and Ratliff inequalities, proposed in 1971 for the Set Covering Problem (SCP) [1], define cutting planes that are strong in the sense that they also cut integer solutions. These inequalities can be obtained from any nonredundant feasible solution of the SCP, where a solution is redundant if it remains feasible after setting to 0 one of its variables at 1. The generation procedure is based on the argument that once a basis is associated to a 0-1 solution of the SCP, any 0-1 solution with an improved objective value must have at least one of the variables, that are at 0 and with a negative reduced cost in this basis, at 1. This follows from linear programming, since the SCP has no variable with an upper bound.

The motivation for this paper is to investigate Bellmore-Ratliff's strong cutting planes as an *intensification* mechanism for Tabu Search heuristics. The idea is to apply Tabu Search to a combinatorial problem and use the strong cutting planes to force exploring regions of the solution space that we know to be promising by construction. Since considering constraints during all the process may reduce too much the connection between regions of the solution space, we propose to use penalties to consider the violation of these cutting planes. The penalty value is increased and decreased generating different periodic intensification phases in the Tabu Search procedure.

We apply this idea to the Flow Shop Scheduling problem (FSP) where only permutation schedules are allowed. In the FSP there are n jobs that must be processed on machines 1, 2, ..., m, in this order. Also, the job processing order must be the same on all the machines. The processing times of each job j on each machine i, d_{ij}, are given. The objective function of the FSP is to minimize the total processing time, which is frequently referenced as the *makespan*.

2 A MIP Model for the *FSP*

The model we consider is based on the definition of 0-1 variables to indicate that a given job k is the j^{th} to be processed on all the machines. Therefore, the first $2n$ constraints define an assignment. The constraints that follow allow computing the makespan, the objective function, for a given order of processing the jobs on the machines. This MIP formulation uses variables x_{kj} - a 0-1 one variable assumes value 1 when job k is the j^{th} to be processed, t_{ij} - the duration of the j^{th} job processed by machine i and s_{ij} - the starting time of the j^{th} job on machine i. It is presented below.

$$MIP_{FSP}\left\{\begin{array}{lrcll}
\text{min} & z & = & s_{mn}+t_{mn} & & (0)\\
\text{subject to} & & & & & \\
 & \sum_{j=1}^{n} x_{kj} & = & 1 & k = 1,\ldots,n & (1)\\
 & \sum_{k=1}^{n} x_{kj} & = & 1 & j = 1,\ldots,n & (2)\\
 & t_{ij} & = & \sum_{k=1}^{n} d_{ik}.x_{kj} & i = 1,\ldots,m;\ j = 1,\ldots,n & (3)\\
 & s_{ij} & \geq & s_{(i)(j-1)}+t_{(i)(j-1)} & i = 1,\ldots,m;\ j = 2,\ldots,n & (4)\\
 & s_{ij} & \geq & s_{(i-1)(j)}+t_{(i-1)(j)} & i = 2,\ldots,m;\ j = 1,\ldots,n & (5)\\
 & x_{kj} & \in & \{0,1\} & \text{for all } k,j & (6)\\
 & s_{ij},t_{ij} & \geq & 0 & \text{for all } i,j & (7)
\end{array}\right.$$

where s_{11} is the starting production time which we set to zero. The objective function is the makespan value, i.e. the starting time of the last job on the last machine plus its duration.

A permutation, which is a feasible solution for the FSP, has exactly n of the x_{kj} variables at 1. Therefore, a basic solution for the above formulation would require n extra x_{kj} variables to belong to the basis with 0 value. This would cause two difficulties: finding these n extra x_{kj} variables and solving the linear system to obtain the associated simplex multipliers (dual variables). The former requires selecting extra x_{kj} variables such that the basic solution is the given permutation, which is certainly time consuming. The latter is due to the need to solve this system every time a strong cutting plane is generated. We will see in the next section that the following reformulation allows these tasks to be done efficiently.

We use the fact that all the durations are nonnegative and that we minimize the makespan to relax the equality constraints (1), (2) and (3), to greater or equal constraints. We call the resulting formulation MIP'_{FSP} and its relaxed constraints (1′), (2′) and (3′). It is clear that any optimal solution for MIP'_{FSP} is an optimal solution for MIP_{FSP}, i.e. there is an optimal solution for which equality holds for constraints (1′), (2′) and (3′). Since increasing the left-hand-side(lhs) of (1′) or (2′) implies increasing the lhs of (3′) (the durations are nonnegative) and increasing the lhs of (3′) implies increasing the lhs of (4) or (5), i.e. the value of any starting time, the result follows.

3 The Bellmore-Ratliff Inequalities for the FSP

To construct a basis from permutation of the n jobs, observe that all the continuous variables are nonnegative and unbounded. Next, verify that in any feasible solution, these variables (t_{ij} and s_{ij}) assume positive values (except for s_{11} which is set to zero and removed from the formulation). Consequently, all of them must be basic. The remaining positive valued variables are the x_{kj} which define the permutation and some slacks on constraints (4) and (5). Another observation is that once the permutation is specified, all the basic variables on constraints (3′), (4) and (5) are defined except for eventually some pair of slacks which are at zero on constraints (4) and (5) with the same variable s_{ij}. In these cases, we can arbitrarily choose a slack, from such pair, to be in the basis.

It remains to define which variables are basic on constraints (1′) and (2′). We have n x_{kj} variables at 1 that saturate all of these $2.n$ constraints. This fact allows to arbitrarily choose a slack to be in basis in one of each pair of constraints that each variable x_{kj} at 1 saturates. As a consequence, there are always 2^n possible different basis that can be constructed from any feasible solution to the FSP. For the purposes of this work, we pick one of these basis at random.

Once a basis is known, deciding whether a variable x_{kj} is in the inequality or not amounts to computing its reduced cost. Remark that only x_{kj} variables can have a negative reduced cost. This can be verified by noting that all the variables that are out of the basis are the x_{kj} variables and some slacks. But since increasing the value of any slack can only augment the objective function value, these variables can only have nonnegative reduced costs.

To compute the reduced costs we need to calculate the value of the simplex multipliers associated to basis we constructed. This is done by solving the system $u.B = c_b$, where u is the vector of simplex multipliers, B is the basis and c_b represent the vector of the costs of the basic variables. But this calculation can always be done in a straight forward way. Set to 0 all multipliers corresponding to a basic slack variable, then the remaining multipliers can be computed in sequence since one multiplier value will always be readily available.

The BR-inequality for the FSP is then $\sum_{\overline{c}_{kj}<0} x_{kj} \geq 1$ where $\overline{c}_{kj}$ is the reduced cost of variable x_{kj} which can now be computed.

4 The Tabu Search Procedure

Let p be a permutation of $\{1, 2, \ldots, n\}$. Therefore, p_j is the j^{th} job to be processed on all m machines. We define the neighborhood of p, $\mathcal{N}(p)$, as the set of all permutations p' such that they differ from p on exactly two positions. Our intensification mechanism works by the addition of a positive value to the makespan $MK(p)$ corresponding to the current permutation p, i.e. a penalty. Penalties are fixed during *num-rep* iterations and are multiplied by the number $CV(p)$ of violated cuts. In order to allow a smooth navigation through the solution space, we slowly increase the penalty, then abruptly decrease its value to 0. Our strategy for cut generation is to only generate cuts from local optima, or when a tabu is set. We consider as tabu attributes the positions in p. A tabu movement is then a swap that has either or both positions tabu. Let t_j $(j = 1, 2, \ldots, n)$ be the last iteration where position j is tabu and the set $T(p, p')$ contain the positions where permutations p and p' differ. The Tabu Search obtained, which we name **BR-Tabu** after the Bellmore and Ratliff cuts, is presented in figure 1.

5 Computational results and conclusions

The BR-Tabu algorithm was implemented in C. The results shown in this section were obtained on a Sun ULTRA workstation. All the experiments described here use the instances proposed by Taillard[3]. Three different sets of instances were used: 20 jobs/20 machines; 50 jobs/20 machines; and 100 jobs/20 machines.

In order to test the algorithm, we limited to 5 minutes the use of the cpu for the 20 jobs/20 machines instances, and to 30 minutes the use of the cpu for the other instances. For each instance, the following experiments were made: local search only (no tabu is set and no cuts) - *this experiment was made only for the 20 jobs/20 machines instances*; tabu search only (no cuts); local search with BR-cuts (no tabu is set) - *this experiment was not made for the 100 jobs/20 machines instances*; and tabu search with BR-cuts. In addition, we used the following values for the parameters of the BR-Tabu algorithm for the 20 jobs/20 machines instances: tabu tag = 2, tabu random = 6, tolerance = 3, last penalty stage = 3, number of repetitions = 10, and number of improvements before start cutting = 10. For the 50 jobs/20 machines and 100 jobs/20 machines instances, a different set of values were used: tabu tag = 2, tabu random = 5, tolerance = 3, last penalty stage = 3, number of repetitions = 10 and number of improvements before start cutting = 0. These numbers were obtained after testing a few times different combinations of values for the parameters. We believe that by choosing the right values to the parameters we can reach optimality in all the 20 jobs/20 machines. Also, we can certainly improve the solutions listed in tables 2 and 3 for the 50 jobs/20 machines and 100 jobs/20 machines instances. To provide a measure of how good the proposed procedure is, we present in all tables the value of the best known solution for each instance, columns labeled with f^+ for tables 2 and 3. Table one is labeled with f^* since these values are proven optimal.

Table 1 shows the makespan for each of the experiments of the 20 jobs/20

```
procedure BR-Tabu
    Generate a random permutation p
    p^best ← p
    Mk^best ← MK(p)
    t_j ← 0, for j = 1, 2, ..., n
    iteration ← 0
    while true do
    penalty ← an element of {20, 10, 5, 2, 1}
        for stage = 0 to last-stage do
            pen ← penalty . stage
            for tol = 1 to num-tol do
                tolerance ← tol(MOD2)
                for rep = 1 to num-rep do
                    iteration ← iteration + 1
                    for all p' ∈ N(p)
                        if MK(p') < Mk^best then
                            p^best ← p'
                            Mk^best ← MK(p')
                        end-if
                        Let {i, j} = T(p, p')
                        level ← 0
                        if (t_i ≤ iteration) level ← level + 1
                        if (t_j ≤ iteration) level ← level + 1
                        if (MK(p')+pen.CV(p'))and (level ≤ tolerance) then
                            p^+ ← p'
                        end-if
                    end-for-all
                    if (MK(p^+) + pen.CV(p^+)) ≥ (MK(p^+) + pen . CV(p))
                    then
                        if tabu-on then
                            Let {i, j} = T(p, p^+)
                            t_i ← iteration+UNIF(tag,tag + tr)
                            t_j ← iteration+UNIF(tag,tag + tr)
                        end-if
                        if cut-on then
                            Generate one strong cutting plane from p
                        end-if
                    end-if
                end-for
            end-for
        end-for
    end-while
    return p^best
end BR-Tabu
```


Figure 1: BR-Tabu Procedure

machines instances. The optimal makespan is listed in the first column. The remaining 4 columns contain the makespan obtained by applying the algorithm with no tabu set and no cuts; with cuts, but no tabu is set; setting tabu, but with no cuts; and with both tabu and cut, respectively. Its clear from table 1 that the algorithm produces very good results in these instances - we obtained 5 optimal solutions, the remaining solutions were at most 0.2% apart from the optimal.

Table 2 shows the results obtained by the algorithm for the 3 experiments with the instances of 50 jobs/20 machines. Also with the makespan calculated, is the number of iterations and time of cpu needed to find the solution. Columns 2, 3 and 4 exhibit data from the experiment without tabu, but with cuts; columns 5, 6 and 7 exhibit data from the experiment without cuts, but with tabu; and finally, columns 8, 9 and 10 has data from the experiment with both tabu and cuts.

Table 3 is very similar to table 2. It shows the results obtained by the algorithm in the 2 experiments with the instances of 100 jobs/20 machines. Columns 2, 3 and 4 have data from the experiment with tabu and with cuts, while columns 5, 6 and 7 have data from the experiment with tabu, but without cuts.

Although the best solutions obtained for the last two set of instances are still about 3% from the best known solution, we may infer that they can be improved by letting the algorithm run for more than just 30 minutes. Note how close to the time limit the best solutions were found. Comparing the solutions obtained by the experiments we can see that the **solutions with cuts and tabus are, in general, better than the solutions without tabus or cuts.** Examining the number of iterations and time of cpu for each of the experiments we find that when we do not allow cutting, the number of iterations per minute is much larger than when we use cuts. Nevertheless, the solution produced with strong cutting planes worth the extra computation time per iteration. For instance, just check table 3 where 1 iteration generating cuts takes about twice the cpu time when cuts are not generated. In this table, the procedure using cuts always find best solutions.

f^*	$f_{tOFF,cOFF}$	$f_{tOFF,cON}$	$f_{tON,cOFF}$	$f_{tON,cON}$
2297	2389	2389	2297*	2203
2099	2179	2105	2101	2111
2326	2418	2338	2328	2334
2223	2350	2350	2223*	2225
2291	2369	2305	2296	2300
2226	2366	2366	2230	2226*
2273	2382	2382	2273*	2284
2200	2329	2329	2207	2202
2237	2378	2378	2242	2242
2178	2292	2292	2178*	2189

Table 1 - 20 jobs, 20 machines, cpu time limited to 5 minutes

f^*	$f_{tOFF,cON}$	Its	cpu_{time}	$f_{tON,cOFF}$	It	cpu_{time}	$f_{tON,cON}$	Its	cpu_{time}
3856	3925	2443	29:58	3916	5507	11:26	3925	1811	17:14
3707	3770	1581	13:48	3805	12518	26:10	3735	2340	28:09
3643	3720	2267	28:31	3750	5249	10:51	3714	1498	12:11
3731	3816	1803	21:15	3799	7821	16:23	3775	2218	27:11
3619	3711	1532	13:48	3780	8222	17:13	3705	824	4:55
3687	3762	1446	12:08	3756	9876	20:31	3748	2241	27:16
3706	3785	1458	13:08	3795	6519	13:42	3768	2242	24:38
3700	3784	1749	20:19	3793	2379	29:59	3793	2379	29:59
3755	3838	2275	27:45	3881	12977	27:09	3817	2283	27:49
3767	3813	1278	10:49	3828	9758	20:23	3806	2030	22:33

Table 2 - 50 jobs, 20 machines, cpu time limited to 30 minutes

f^+	$f_{tON,cON}$	$Iterations$	cpu_{time}	$f_{tON,cOFF}$	$Iterations$	cpu_{time}
6228	6422	807	27:11	6433	1215	23:54
6210	6344	892	29:24	6397	1411	27:21
6271	6425	499	11:36	6437	1010	19:48
6269	6384	956	30:05	6475	1081	20:52
6319	6426	705	20:29	6532	520	9:44
6403	6507	867	25:23	6577	740	14:08
6292	6469	472	11:12	6502	967	18:47
6423	6600	750	26:45	6652	1511	29:55
6275	6438	820	26:52	6482	1483	29:18
6434	6533	910	28:12	6629	937	18:20

Table 3 - 100 jobs, 20 machines, cpu time limited to 30 minutes

References

[1] Bellmore M. and H.D. Ratliff, *Set Covering and Involutory Basis*, Management Science 18(3), 194-206, 1971.

[2] F. Glover, Tabu Search - Part I,II, ORSA Journal on Computing **1** (1989), 190-206, **2** (1990), 4-32.

[3] Taillard E., *Benchmarks for Basic Scheduling Problems* European Journal of Operational Research 64, 278-285, 1993.