Tabu Search for Mixed Integer Programming

João Pedro Pedroso

Technical Report Series: DCC-2004-02

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

---

Universidade do Porto

Rua do Campo Alegre 823, 4150-180 Porto, Portugal

Tel: +351-226.078.830 – Fax: +351-226.003.654

http://www.dcc.fc.up.pt/Pubs/treports.html

Tabu Search for Mixed Integer Programming

João Pedro Pedroso
DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 823, 4150-180 Porto, Portugal
Email: jpp@ncc.up.pt

March 2004

Abstract

This paper introduces tabu search for the solution of general linear integer problems. Search is done on integer variables; if there are continuous variables, their corresponding value is determined through the solution of a linear program, which is also used to evaluate the integer solution.

The complete tabu search procedure includes an intensification and diversification procedure, whose effects are analysed on a set of benchmark problems.

Key-words: Tabu search, Linear Integer Programming, Mixed Integer Programming

1 Introduction

In this work we focus on a tabu search for the problem of optimizing a linear function subject to a set of linear constraints, in the presence of integer and, possibly, continuous variables. If the subset of continuous variables is empty, the problem is called *pure integer* (IP). In the more general case, where there are also continuous variables, the problem is usually called *mixed integer* (MIP).

The mathematical programming formulation of a mixed integer linear program is

$$z = \min_{x,y}\{cx + hy : Ax + Gy \geq b, x \in \mathbb{Z}^n_+, y \in \mathbb{R}^p_+\} \tag{1}$$

where $\mathbb{Z}^n_+$ is the set of nonnegative, integral n-dimensional vectors and $\mathbb{R}^p_+$ is the set of nonnegative, real p-dimensional vectors. A and G are $m \times n$ and $m \times p$ matrices, respectively, where m is the number of constraints. The integer variables are x, and the continuous variables are y. We assume that there are additional bound restrictions on the integer variables: $l_i \leq x_i \leq u_i$, for $i = 1, \ldots, n$.

The strategy proposed consists of fixing the integer variables x by tabu search, and obtaining the corresponding objective z and continuous variables y by solving a linear programming (LP) problem (this idea has been introduced in [12, 13, 11]). We are therefore using tabu search to fix only integer variables, as the continuous ones can be unambiguously determined in function of them.

In the process of evaluation of a solution, we first formulate an LP by fixing all the variables of the MIP at the values determined by tabu search:

$$z = \min_{y}\{cx + hy : Ax + Gy \geq b, x = \bar{x}, y \in \mathbb{R}^p_+\} \tag{2}$$

We are now able to solve this (purely continuous) linear problem using the simplex algorithm. If it is feasible, the evaluation given to $\bar{x}$ is the objective value z; as the solution is feasible, we set the sum of violations, ζ, to zero. If this problem is infeasible, we set ζ equal the total constraint violations (obtained at the end of phase I of the simplex algorithm). Notice that for IPs, after fixing the integer variables the remaining problem has no free variables; some LP solvers might not provide the correct values of z or ζ for the fixed variables.

We say that a solution structure i *is better* than another structure j if $\zeta^i < \zeta^j$, or $\zeta^i = \zeta^j$ and $z^i < z^j$ (for minimization problems).

The initial solution is obtained by rounding the integer variables around their optimal values on LP relaxations. Tabu search starts operating on this solution by making changes exclusively on the integer variables, after which the continuous variables are recomputed through LP solutions.

Modifications of the solution are made using a simple neighborhood structure: incrementing or decrementing one unit to the value of an integer variable of the MIP. This neighborhood for solution x is the set of solutions which differ from x on one element x_i, whose value is one unit above or below x_i. Hence x' is a neighbor solution of x if $x'_i = x_i + 1$, or $x'_i = x_i - 1$, for one index i, and $x'_j = x_j$ for all indices $j \neq i$.

Moves are tabu if they involve a variable which has been changed recently. An aspiration criterion allows tabu moves to be accepted if they lead to the best solution found so far. This is the basic tabu search, based only on short term memory, as described in [6].

As suggested in [7], we complement this simple tabu search with intensification and diversification. Intensification allows non-tabu variables to be fixed by branch-and-bound (B&B). Diversification creates new solutions based on LP relaxations, but keeping a part of the current solution unchanged.

We have tested the tabu search with a subset of benchmark problems that are available, in the *MPS* format, in the *MIPLIB*[1, 10]. Results obtained by tabu search are compared to the solution of B&B. We have used a publicly available implementation of this algorithm, the *GLPK* software package. This was used also as the LP simplex solver on tabu search, as well as the B&B solver on the intensification phase, thus allowing a direct comparison to *GLPK* in terms of CPU time required for the solution.

Various libraries provide canned subroutines for tabu search, and some of these can be adapted to integer programming applications. However, such libraries operate primarily as sets of building blocks that are not organized to take particular advantage of considerations relevant to the general MIP setting. A partial exception is the tabu search component of the COIN-OR open source library (see [2]). However, this component is acknowledged to be rudimentary and does not attempt to encompass many important elements of tabu search.

2 A simple tabu search

In this section we introduce a simple tabu search, based only on short term memory (Algorithm 1). This procedure has one parameter: the number of iterations, N, which is used as the stopping criterion. The other arguments are a seed for initializing the random number generator, and the name of the *MPS* benchmark file.

The part of the MIP solution that is determined by tabu search is the subset of integer variables x in Equation 1. The data structure representing a solution is therefore an n-dimensional vector of integers, $\bar{x} = (\bar{x}_1 \ldots \bar{x}_n)$.

Algorithm 1: Simple tabu search.
SimpleTabu(N, *seed*, *MPSfile*)
(1) read global data A, G, b, c, and h from *MPSfile*
(2) initialize random number generator with *seed*
(3) $\bar{x} :=$ Construct()
(4) $\bar{x}^* := \bar{x}$
(5) $t := (-n, \ldots, -n)$
(6) **for** $i = 1$ **to** N
(7) $\quad \bar{x} :=$ TabuMove$(\bar{x}, \bar{x}^*, i, t)$
(8) $\quad$ **if** $\bar{x}$ is better than $\bar{x}^*$
(9) $\quad\quad \bar{x}^* := \bar{x}$
(10) **return** $\bar{x}^*$

2.1 Construction

Construction is based on the solution of the LP relaxation with a set of variables $\mathcal{F}$ fixed, as stated in equation 3 (empty $\mathcal{F}$ leads to the LP relaxation of the initial problem).

$$\min_{x,y}\{cx + hy : Ax + Gy \geq b, x \in \mathbb{R}^n_+,\ y \in \mathbb{R}^p_+,\ x_i = \bar{x}_i\ \forall\, i \in \mathcal{F}\} \tag{3}$$

The solution of this problem is denoted by $x^{LP} = (x_1^{LP}, \ldots, x_n^{LP})$; these values are rounded up or down with some probabilities and fixed, as shown in Algorithm 2, were we denote by r a continuous random variable with uniform distribution within $[0, 1]$.

Algorithm 2: Semi-greedy solution construction.
Construct()
(1) $\mathcal{F} := \{\}$
(2) $\mathcal{C} := \{1, \ldots, n\}$
(3) **for** $j = 1$ **to** n
(4) $\quad$ solve Equation 3
(5) $\quad$ randomly select index i from $\mathcal{C}$
(6) $\quad$ **if** $r < x_i^{LP} - \lfloor x_i^{LP} \rfloor$
(7) $\quad\quad \bar{x}_i := \lceil x_i^{LP} \rceil$
(8) $\quad$ **else**
(9) $\quad\quad \bar{x}_i := \lfloor x_i^{LP} \rfloor$
(10) $\quad \mathcal{F} := \mathcal{F} \cup \{i\}$
(11) $\quad \mathcal{C} := \mathcal{C} \backslash \{i\}$
(12) **return** $\bar{x}$

This semi-greedy construction is inspired in an algorithm provided in [8]. It consists of rounding each variable i to an integer next to its value on the LP relaxation, x_i^{LP}. For all the indices $i \in \{1, \ldots, n\}$, the variable $\bar{x}_i$ is equal to the value x_i^{LP} rounded down with probability

$$P(\bar{x}_i = \lfloor x_i^{LP} \rfloor) = \lceil x_i^{LP} \rceil - x_i^{LP},$$

or rounded up with probability $1 - P(\bar{x}_i = \lfloor x_i^{LP} \rfloor)$ (lines 6 to 9 of the Algorithm 2).

2.2 Candidate selection

At each tabu search iteration, the neighborhood of the current solution is searched and a neighbor solution is selected, as presented in Algorithm 3. The arguments of this

algorithm are the current solution $\bar{x}$, the best solution found $\bar{x}^*$, the current iteration i, and the tabu vector t. Tabu information is kept in vector t; t_c holds the iteration at which variable c has been updated.

Algorithm 3: Search of a candidate at each tabu search iteration.

```
TabuMove(x̄, x̄*, k, t)
(1)    if k − t_i > n   ∀ i
(2)        c := R[1, n]
(3)        x̄_c := R[l_c, u_c]
(4)        t_c := k
(5)        return x̄
(6)    v := x̄
(7)    for i = 1 to n
(8)        s := x̄
(9)        d := R[1, n]
(10)       foreach δ ∈ {−1, +1}
(11)           s_i := x̄_i + δ
(12)           if s_i ∈ [l_i, u_i] and s better than v
(13)               if k − t_i > d or s better than x̄*
(14)                   v := s
(15)                   c := i
(16)   x̄ := v
(17)   t_c := k
(18)   return x̄
```

Lines 1 to 5 prevent the case where the search is blocked, all moves being potentially tabu. In this case a random move is taken: an index is selected randomly, and a value for that variable is drawn within its bounds, with uniform distribution. (We denote by $R[1, n]$ a random integer with uniform distribution within $1, \ldots, n$.)

The neighborhood is searched by adding a value $\delta = \mp 1$ to each of the variables $1, \ldots, n$, as long as they are kept within their bounds. We have tested two search possibilities: breadth first and depth first. With breadth first all the neighbor solutions are checked, and the best is returned (lines 7 to 15). With depth first, as soon as a non-tabu solution better than the current solution is found, it is returned. Results obtained for these two strategies are rather similar, but there seems to be a slight advantage to breadth-first, which is the strategy that adopted is this paper. More sophisticated ways of managing choice rules by means of candidate list strategies are an important topic in tabu search (see, e.g., [4]), and may offer improvements, but we elected to keep this aspect of the method at a simple level.

The tabu tenure (the number of iterations during which an changed variable remains tabu) is generally a parameter of tabu search. In order keep simplify the parameterization, we decided to consider it a random value between 1 and the number of integer variables, n. Such value, d, is drawn independently for each variable (line 9); this might additionally lead to different search paths when escaping the same local optimum, in case this situation arises.

2.3 Results

The set of benchmark problems and the statistical measures used to report solutions are presented in appendix C.

Results obtained by this simple tabu search, presented in table 1 are encouraging, as good solutions were found to problems which could not be easily solved by B&B (please see next section for B&B results). Still, for many problems the optimal solution was not found. This simple tabu search is many times trapped in regions from which it cannot easily escape, wasting large amounts of time. As we will see in the next section, this can be dramatically improved with intensification and diversification procedures.

problem name	best z	best ζ	%above optimum	%feas runs	Feasibility ($E[t^f]$ (s))	%best runs	Best sol. ($E[t^f]$ (s))	%opt runs	Optimality ($E[t^f]$ (s))
bell3a	878430.32	0	0	100	0.23	75	50.32	75	50.32
bell5	9011612.98	0	0.50	30	212.01	5	1688.15	0	$\gg$1764.32
egout	568.1007	0	0	100	0.47	100	7.56	100	7.56
enigma	0	0.0278	n/a	0	$\gg$1638.69	5	1559.63	0	$\gg$1638.69
flugpl	1201500	0	0	30	19.54	15	42.85	15	42.85
gt2	23518	0	11.11	100	0.67	5	4506.53	0	$\gg$4659.14
lseu	1218	0	8.75	45	95.75	5	1441.24	0	$\gg$1512.53
mod008	307	0	0	100	0.17	5	10751.22	5	10751.22
modglob	20757757.11	0	0.08	100	0.18	5	26546.63	0	$\gg$27873.20
noswot	-41	0	4.65	95	13.06	20	928.09	0	$\gg$4667.98
p0033	3347	0	8.35	20	55.55	15	78.70	0	$\gg$272.96
pk1	17	0	54.54	100	0.03	5	2009.57	0	$\gg$2046.81
pp08a	7350	0	0	100	0.11	25	1439.24	25	1439.24
pp08aCUT	7350	0	0	100	0.15	20	3090.42	20	3090.42
rgn	82.1999	0	0	100	0.03	100	6.13	100	6.13
stein27	18	0	0	100	0.01	100	0.05	100	0.05
stein45	30	0	0	100	0.06	80	66.39	80	66.39
vpm1	20	0	0	100	0.52	5	12149.37	5	12149.37

Table 1: Simple tabu search: best solution found, percent distance above optimum; expected CPU time required for reaching feasibility, the best solution, and optimality. (Results based on 20 observations of the algorithm running for 5000 iterations.)

3 Intensification and diversification

We now introduce intensification and diversification procedures to complement the simple tabu search presented in the previous section. These are essential to the performance of the algorithm; in many situations they can save a large amount of computational time, both by speeding up the search in case of being far from a local optima, or by moving the tabu search to different regions when it is trapped somewhere it cannot easily escape.

3.1 Intensification

The complete intensification strategy is presented in Algorithm 4. It consists of fixing all the variables which are tabu (those belonging to set $\mathcal{F}$, determined in line 1 of Algorithm 4), releasing all the non-tabu variables, and solving the remaining MIP on these:

$$\min_{x,y}\{cx + hy : Ax + Gy \geq b, x \in \mathbb{Z}^n_+,\ y \in \mathbb{R}^p_+,\ x_k = \bar{x}_k\ \forall\, k \in \mathcal{F}\} \tag{4}$$

The rationale behind this procedure is the following: most of the difficult MIP became easy to solve (or at least much easier than the original problem) if a set of important variables are fixed (a fuller development of this type of strategy can be found in [5], and a related discussion appears in [3]). What we are doing here is to say that important variables at a given point of the search are those which are tabu (i.e., those which have been updated recently).

For some strongly constrained instances, the problem 4 might have no feasible LP relaxation. In this case, we randomly remove variables from the set $\mathcal{F}$, until the LP relaxation becomes feasible (lines 2 to 4). On the other end, the MIP problem of

Algorithm 4: Intensification.
INTENSIFY(x, t, i)
(1) $\mathcal{F} := \{k : i - t_k > n\}$
(2) **while** Equation 3 is not feasible
(3) randomly select index k from $\mathcal{F}$
(4) $\mathcal{F} := \mathcal{F} \backslash \{k\}$
(5) solve Equation 4 (allow search for n seconds, max.)
(6) **if** no integer solution was found
(7) **return** x
(8) let x' be the solution of Equation 4
(9) **return** x'

Equation 4 might still be very difficult to solve; in order to avoid wasting too much time on its solution, we limit the time spent on it. This limit could be parameterized; but, in order to keep the discussion of results free of parameters, we decided to allow an amount of time equal to the number of integer variables, n, in seconds. (Actually, this is a rather poor choice: in our machine, in most of the cases either a good solution is found in about one second, or it is not found in n seconds; but let us keep it as proposed for the sake of simplicity. Notice also that a way of improving this parameterization would be to use a dynamic strategy, allowing more time to intensification when it appears to be rewarding, as is current practice in tabu search.)

3.2 Diversification

The diversification procedure is similar to construction, but it keeps a part of the current solution structure unchanged. This procedure, presented in Algorithm 5 starts by drawing a random integer l, between 1 and the number of integer variables n. It will then randomly select l variables to remove (lines 4 to 7), and fix them (in a random order), by means of the rounding technique described in section 2.1 (lines 8 to 16).

With this procedure, on average 50% of the current solution structure will be kept after diversification; additionally, the reconstructed part will still have the high quality provided by the rounding procedure.

3.3 The complete tabu search

Diversification and intensification have to be carefully combined in order to do a good team work inside tabu search. The main procedure, presented in Algorithm 6, starts with a construction (as in the case of simple tabu). This initiates the first "diversification stream". A simple, short term memory tabu search starts with this solution (lines 15, 16), and pursues until, at a certain point of the search on this stream, there will be an intensification (lines 9, 10). After doing at least one intensification, and having no improvements for a certain number of tabu search iterations, there will be a diversification (lines 11 to 14); this starts the next diversification stream.

In order to do this search in a "parameter-free" fashion, we propose the following: do an intensification after n (the number of integer variables) iterations with no improvement on the best solution found on the current diversification stream. An intensification starts with the best solution found on the current diversification stream, not with the current solution. After $n + 1$ non-improving tabu search iterations (hence after doing at least one intensification), do a diversification. On Algorithm 6, the number of iterations with no improvement on the current stream's best solution is computed as variable q, in lines 14 and 17 to 23.

Algorithm 5: Diversification: partial solution destruction and reconstruction.
DIVERSIFY(x)
(1) $\mathcal{F} := \{1, \ldots, n\}$
(2) $\mathcal{C} := \{\}$
(3) $l = R[1, n]$
(4) **for** $k = 1$ **to** l
(5) randomly select index k from $\mathcal{F}$
(6) $\mathcal{F} := \mathcal{F} \backslash \{k\}$
(7) $\mathcal{C} := \mathcal{C} \cup \{k\}$
(8) **for** $k = 1$ **to** l
(9) solve Equation 3
(10) randomly select index k from $\mathcal{C}$
(11) **if** $r < x_k^{LP} - \lfloor x_k^{LP} \rfloor$
(12) $x_k := \lceil x_k^{LP} \rceil$
(13) **else**
(14) $x_k := \lfloor x_k^{LP} \rfloor$
(15) $\mathcal{F} := \mathcal{F} \cup \{k\}$
(16) $\mathcal{C} := \mathcal{C} \backslash \{k\}$
(17) **return** $\bar{x}$

3.4 Results

We present results obtained by the complete tabu search algorithm, in table 2. Comparing these with the results of simple tabu the importance of intensification and diversification becomes clear; the solution quality is considerably improved, and the CPU time required to solving the problems is much reduced.

The results obtained utilizing the B&B implementation of *GLPK* on the series of benchmark problems selected are provided in the Table 3. The maximum CPU time allowed is 24 hours; in case this limit was exceeded, the best solution found within the limit is reported. *GLPK* uses a heuristic by Driebeck and Tomlin to choose a variable for branching, and the best projection heuristic for backtracking (see [9] for further details).

The analysis of tables 2 and 3 shows that for most of the benchmark instances, tabu search requires substantially less CPU for obtaining the optimal solution than B&B (though the times reported for B&B are for the complete solution of the problem, not only finding the optimal solution). Two of the problems for which B&B could not find an optimal solution in 24 hours of CPU time (gt2 and modglob) were solved by tabu search in a reasonable amount of time.

We also present the results obtained utilizing the commercial solver *Xpress-MP Optimizer, Release 13.02*, again limiting the CPU time to 24 hours maximum. Although there are some exceptions, this solver is generally much faster than our implementation of tabu search, but, as the code is not open, we do not know why. A part of the differences could be explained by the quality of the LP solver. Another part could be due to the use of branch-and-cut. Finally, some differences could be due to preprocessing; in our opinion, this is probably the improvement on tabu search that could bring more important rewards.

4 Conclusion

The literature in meta-heuristics reports many tabu search applications to specific problems. In this paper we present a version to solve general integer linear problems. In

problem name	best z	best ζ	%above optim.	%feas runs	Feasibility ($E[t^f]$(s))	%best runs	Best sol. ($E[t^f]$(s))	%opt runs	Optimality ($E[t^f]$(s))
bell3a	878430.32	0	0	100	0.24	100	4.38	100	4.38
bell5	8966406.49	0	0	100	8.60	100	38.24	100	38.24
egout	568.1007	0	0	100	0.47	100	6.76	100	6.76
enigma	0	0	0	35	187.51	20	376.66	20	376.66
flugpl	1201500	0	0	100	1.52	100	1.55	100	1.55
gt2	21166	0	0	100	0.65	15	2216.95	15	2216.95
lseu	1120	0	0	100	1.47	10	770.45	10	770.45
mod008	307	0	0	100	0.17	40	1119.57	40	1119.57
modglob	20740508.1	0	0	100	0.16	100	1404.29	100	1404.29
noswot	-41	0	4.65	100	8.38	95	239.96	0	$\gg$15653.99
p0033	3089	0	0	100	0.17	90	4.10	90	4.10
pk1	15	0	36.36	100	0.03	10	1111.96	0	$\gg$2357.09
pp08a	7350	0	0	100	0.11	45	4316.38	45	4316.38
pp08aCUT	7350	0	0	100	0.15	70	766.59	70	766.59
rgn	82.1999	0	0	100	0.03	100	0.73	100	0.73
stein27	18	0	0	100	0.02	100	0.05	100	0.05
stein45	30	0	0	100	0.06	80	66.65	80	66.65
vpm1	20	0	0	100	0.48	95	393.73	95	393.73

Table 2: Complete tabu search: best solution found, percent distance above optimum; expected CPU time required for reaching feasibility, the best solution, and optimality. (Results based on 20 observations of the algorithm running for 5000 iterations.)

problem name	best z	CPU time (s)	remarks
bell3a	878430.32	134.7	
bell5	8966406.49	143.3	
egout	568.1007	3.6	
enigma	0	14.1	
flugpl	1201500	1.3	
gt2	30161*	93822.3	stopped, >24h CPU time
lseu	1120	96.6	
mod008	307	51.0	
modglob	20815372.17*	93839.7	stopped, >24h CPU time
noswot	-41*	137.9	stopped, numerical instability
p0033	3089	1.1	
pk1	11	49713.9	
pp08a	7350	93823.4	stopped, >24h CPU time
pp08aCUT	7350	93822.3	stopped, >24h CPU time
rgn	82.12	4.1	
stein27	18	3.9	
stein45	30	269.3	
vpm1	20	10261.8	

Table 3: Results obtained by branch-and-bound, using *GLPK - version4.4*: solution found and CPU time. (* indicates non-optimal solutions.)

Algorithm 6: A complete tabu search.

```
TabuSearch(N, seed, MPSfile)
(1)    read global data A, G, b, c, and h from MPSfile
(2)    initialize random number generator with seed
(3)    x̄ := Construct()
(4)    x̄* := x̄
(5)    x̄' := x̄
(6)    q = 0
(7)    t := (−n, ..., −n)
(8)    for i = 1 to N
(9)       if q = n
(10)         x̄ := Intensify(x̄, t, i)
(11)      else if q > n
(12)         x̄ := Diversify(x̄)
(13)         x̄' := x̄
(14)         q = 0
(15)      else
(16)         x̄ := TabuMove(x̄, x̄*, i, t)
(17)      if x̄ is better than x̄*
(18)         x̄* := x̄
(19)      if x̄ is better than x̄'
(20)         x̄' := x̄
(21)         q = 0
(22)      else
(23)         q := q + 1
(24)   return x̄*
```

this domain, the most commonly used algorithm is branch-and-bound. This algorithm converges to the optimal solution, but might be unusable in practical situations due to the large amounts of time or memory required for solving some problems.

The tabu search proposed in this paper provides a way for quickly solving to optimality most of the problems analyzed. In terms of time required for reaching the optimal (or a good) solution, comparison with the branch-and-bound algorithm implemented in the *GLPK* favors tabu search. Comparison with the commercial solver *Xpress-MP Optimizer* in general is not favorable to tabu search, although there are some exceptions. Probably, preprocessing is playing an important role in *Xpress-MP*'s performance; actually, we believe that preprocessing is the most promising research direction for improving tabu search performance.

Our tabu search implementation, utilizing the *GLPK* routines, is publicly available [14]; the reader is kindly invited to use it. The implementation with *GLPK* is very simple: tabu search routines are just a few hundreds of C programming lines, which can be easily adapted to more specific situations.

The strategy proposed in this work makes it straightforward to apply tabu search to any model that can be specified in mathematical programming, and thus opens a wide range of applications for tabu search within a single framework. Tabu search can be used to provide an initial solution for starting a branch-and-bound process; it can also be used for improving an integer solution found by a branch-and-bound which had to be interrupted due to computational time limitations.

We emphasize our implementation is not the only one possible. Three types of

problem name	best z	CPU time (s)	remarks
bell3a	878430.32	87	
bell5*	8988042.65*	>24h	stopped, >24h CPU time
egout	568.1007	0	
enigma	0	0	
flugpl	1201500	0	
gt2	21166	0	
lseu	1120	0	
mod008	307	0	
modglob	20740508.1	0	
noswot*	-41*	>24h	stopped, >24h CPU time
p0033	3089	0	
pk1	11	937	
pp08a	7350	31	
pp08aCUT	7350	5	
rgn	82.1999	0	
stein27	18	1	
stein45	30	142	
vpm1	20	0	

Table 4: Results obtained by the commercial *Xpress-MP Optimizer, Release 13.02*: solution found and CPU time reported by the solver. (* indicates non-optimal solutions.)

procedures for applying tabu search to MIP problems are discussed in [7], utilizing strategies somewhat different than those proposed here. To our knowledge, none of these alternatives has been implemented and tested. Additional considerations for applying tabu search to mixed integer programming are discussed in [4], including tabu branching procedures and associated ideas of branching on created variables that provide an opportunity to generate stronger branches than those traditionally employed. It is hoped that the present work, which appears to mark the first effort to investigate tabu search in the general MIP setting, will spur additional explorations of this topic.

A Benchmark problems

The instances of MIP and IP problems used as benchmarks are defined in the *MIPLIB* [1] and are presented in Table 5. They were chosen to provide an assortment of MIP structures, with instances coming from different applications.

Problem name	Application	Number of variables			Number of constraints	Optimal solution
		total	integer	binary		
bell3a	fiber optic net. design	133	71	39	123	878430.32
bell5	fiber optic net. design	104	58	30	91	8966406.49
egout	drainage syst. design	141	55	55	98	568.101
enigma	unknown	100	100	100	21	0
flugpl	airline model	18	11	0	18	1201500
gt2	truck routing	188	188	24	29	21166
lseu	unknown	89	89	89	28	1120
mod008	machine load	319	319	319	6	307
modglob	heating syst. design	422	98	98	291	20740508
noswot	unknown	128	100	75	182	-43
p0033	unknown	33	33	33	16	3089
pk1	unknown	86	55	55	45	11
pp08a	unknown	240	64	64	136	7350
pp08acut	unknown	240	64	64	246	7350
rgn	unknown	180	100	100	24	82.1999
stein27	unknown	27	27	27	118	18
stein45	unknown	45	45	45	331	30
vpm1	unknown	378	168	168	234	20

Table 5: Set of benchmark problems used: application, number of constraints, number of variables and optimal solutions as reported in MIPLIB.

B Computational environment

The computer environment used on this experiment is the following: a Linux Debian operating system running on a machine with an AMD Athlon processor at 1.0 GHz, with 512 Gb of RAM. Both tabu search and *GLPK* were implemented on the C programming language.

C Statistics used

In order to assess the empirical efficiency of tabu search, we provide measures of the expectation of the CPU time required for finding a feasible solution, the best solution found, and the optimal solution, for each of the selected MIP problems.

Let t_k^f be the CPU time required for obtaining a feasible solution in iteration k, or the total CPU time in that iteration if no feasible solution was found. Let t_k^o and t_k^b be identical measures for reaching optimality, and the best solution found by tabu search, respectively. The number of independent tabu search runs observed for each benchmark is denoted by K. Then, the expected CPU time required for reaching feasibility, based on these K iterations, is:

$$E[t^f] = \sum_{k=1}^{K} \frac{t_k^f}{r^f},$$

while

$$E[t^b] = \sum_{k=1}^{K} \frac{t_k^b}{r^b}$$

is the expected CPU time for finding the best tabu search solution, and the expected CPU time required for reaching optimality is

$$E[t^o] = \sum_{k=1}^{K} \frac{t_k^o}{r^o}.$$

For $r^f = 0$ and $r^o = 0$, the sums provide respectively a lower bound on the expectations of CPU time required for feasibility and optimality.

References

[1] Robert E. Bixby, Sebastiàn Ceria, Cassandra M. McZeal, and Martin W. P. Savelsbergh. An updated mixed integer programming library. Technical report, Rice University, 1998. TR98-03.

[2] COIN-OR. COmputational INfrastructure for Operations Research. Internet repository, version 1.0, 2004. www.coin-or.org.

[3] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[4] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[5] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[6] Fred Glover. Tabu search—part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[7] Fred Glover. Tabu search—part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[8] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*, chapter 8, pages 427–446. Applicable Theory in Computer Science. John Wiley and Sons, 1990.

[9] Andrew Makhorin. *GLPK – GNU Linear Programming Kit.* Free Software Foundation, www.gnu.org, January 2004. version 4.4.

[10] Internet repository, version 3.0, 1996. www.caam.rice.edu/∼bixby/miplib.

[11] Teresa Neto and João P. Pedroso. Grasp for linear integer programming. In Jorge P. Sousa and Mauricio G. C. Resende, editors, *METAHEURISTICS: Computer Decision-Making*, Combinatorial Optimization Book Series, pages 545–574. Kluwer Academic Publishers, 2003.

[12] João P. Pedroso. An evolutionary solver for linear integer programming. BSIS Technical Report 98-7, Riken Brain Science Institute, Wako-shi, Saitama, Japan, 1998.

[13] João P. Pedroso. An evolutionary solver for pure integer linear programming. *International Transactions in Operational Research*, 9(3):337–352, May 2002.

[14] João P. Pedroso. Tabu search for MIP: an implementation in the C programming language. Internet repository, version 1.0, 2004. www.ncc.up.pt/∼jpp/mipts.