Hybrid Enumeration Strategies for Mixed Integer Programming

João Pedro Pedroso

Technical Report Series: DCC-2004-8

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

Hybrid Enumeration Strategies 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

October 2004

Abstract

In this paper we present several algorithms which combine a partial enumeration with meta-heuristics for the solution of general mixed-integer programming problems. The enumeration is based on the primal values assignable to the integer variables of the problem. We develop some algorithms for this integration, and test them using a set of well-known benchmark problems.

Key-words: Enumeration, Linear Integer Programming, Mixed Integer Programming

1 Introduction

This paper presents an enumeration strategy for general linear integer problems, based on cycling through the variables' primal, integer domains. Linear programming relaxations are used for choosing the values to assign at each step, as well as for pruning nodes where there is no hope of finding improving solutions.

The enumeration is done on the integer variables; if there are continuous variables, their corresponding value is determined through the solution of a linear program (LP) where all the integer variables are fixed.

For large or hard problems, the complete enumeration is not affordable on realistic setups. We propose several possibilities in order to reduce the search tree, including enumeration on a subset of variables, and enumeration on a subset of the possible outcomes for each variable.

In some situations, especially when variables have large domains, the standard branch-and-bound is likely to provide better results than enumeration. With this in view, we propose the use of branch-and-bound in the last stage of determining a solution, for obtaining the optimal value of some free variables, conditioned to the values of other variables that are kept fixed.

We finally specify a hybrid strategy, where some variables are completely enumerated, and some other are partially enumerated. As a solution improvement after construction, some variables are freed and determined by branch-and-bound. At the end of each iteration, the best found solution is partially destroyed, and the next construction attempt starts with some variables instantiated from that solution. This strategy is rather difficult to classify under the currently categorized meta-heuristics, of which a recent, comprehensive survey has been published in [5]. It might be seen as a special case of GRASP [11], were the semi-greedy construction is replaced by partial enumeration.

Section 6 provides computational results for a set of well-known benchmark problems, available in in the *MPS* format, in the MIPLIB [1, 8]. Results obtained by the enumeration-based strategies are compared to those of branch-and-bound and tabu search. We have used the publicly available *GLPK* software package for providing both branch-and-bound and linear programming solvers.

There have been many approaches to the approximate solution of mixed-integer problems (MIP); remarkable works include [4, 12, 3, 2]. In this contribution, our main aim is to try to explore diversification ideas, through the use forced enumeration on some variables. MIP-based enumeration ideas have to some extent been applied in constraint programming, as referred in [12].

2 Background algorithms

The problem of optimizing a linear function subject to a set of linear constraints in the presence of integer variables is a *pure integer* (IP) programming problem; in the more general case where there are continuous variables, the problem is 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. In this paper we assume that there are additional bound restrictions on the integer variables: $l_i \leq x_i \leq u_i$, for $i = 1, \ldots, n$.

A primal-based enumeration method for determining the solution of this problem consists of progressively fixing the integer variables, each one at every different value of its domain, up to the point where there are no more free (integer) variables [6]. At this point we have a complete solution for the integer variables—call it $\bar{x}$. We can hence solve a linear program to determine the value of the objective that corresponds to these fixed variables, as well as the corresponding optimal value of the continuous variables, if there are some (Equation 2). Notice that if the initial problem is pure integer, at this point there will be no free variables; some linear programming solvers might not provide the correct values of z or ζ for the fixed variables.

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

If this problem is infeasible, we calculate the total constraint violations at the end of phase I of the simplex algorithm, which we denote by ζ. For minimization problems, we say that a solution x^1 is better than a solution x^2 if $\zeta^1 < \zeta^2$, or $\zeta^1 = \zeta^2$ and $z^1 < z^2$.

2.1 Blind enumeration

The enumeration of the integer variables can be described through a recursive function, as shown in Algorithm 1. In this algorithm the set of currently fixed variables is denoted by $\mathcal{F}$, and the set of variables yet to be fixed by $\mathcal{V}$; thus, its first call would be ENUMERATE$(\bar{x}, x^*, \{\}, \{1, \ldots, n\})$.

This algorithm requires a computational time that grows exponentially with the number of integer variables, and therefore is not usable but for toy problems.

2.2 Enumeration with pruning

An improvement of Algorithm1 that in practice can provide a considerable speedup consists of solving an linear programming relaxation in each node, with the currently

Algorithm 1: Integer variables' enumeration.

ENUMERATE($\bar{x}$, x^*, $\mathcal{F}$, $\mathcal{V}$)

(1) **if** $\mathcal{V} = \{\}$
(2) determine z and ζ through Equation 2
(3) **if** $\bar{x} = x^{LP}$ is better than x^*
(4) $x^* := \bar{x}$
(5) **return**
(6) select index i from $\mathcal{V}$
(7) $\mathcal{F} := \mathcal{F} \cup \{i\}$
(8) $\mathcal{V} := \mathcal{V} \backslash \{i\}$
(9) **foreach** $j \in \{l_i, u_i\}$
(10) fix $\bar{x}_i := j$
(11) ENUMERATE($\bar{x}$, x^*, $\mathcal{F}$, $\mathcal{V}$)
(12) release $\bar{x}_i$
(13) $\mathcal{F} := \mathcal{F} \backslash \{i\}$
(14) $\mathcal{V} := \mathcal{V} \cup \{i\}$

enumerated variables fixed and the other relaxed, before fixing any variable, as specified in Equation 3.

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

This LP relaxation provides several useful informations. First, if it is infeasible we can prune the current node, as there will be not feasible solution on its branches. We can also prune it if the value of the objective is equal, or worse than that of a known feasible solution. Finally, if we denote this relaxation's solution by $x^{LP} = (x_1^{LP}, \ldots, x_n^{LP})$, the value of a relaxed variable x_i^{LP} determined by this problem can be rounded; the rounded value and its integer neighbors provide good initial guesses for the enumeration, as a tentative for quickly obtaining good solutions.

In equation 3, variables in the set $\mathcal{F}$ are fixed at the value determined by the enumeration, and the other integer variables are relaxed.

An enumeration based on these considerations is presented in Algorithm 2. In this algorithm, for the case of binary variables, the set $\mathcal{J}$ (line 12) contains only two values; but for general integer variables, it can be rather large. In order to tackle this issue, we propose to solve the relaxation problem with modified objectives $\max x_i$ and $\min x_i$, in order to try to reduce respectively u_i and l_i for this particular node. We only do this when, for the original bounds, $u_i - l_i \geq 3$.

This enumeration provides a strategy that is somehow comparable to the standard branch-and-bound. It can determine the optimal solution of any MIP, at the cost of a computational time that on the worst case is exponential on the number of variables. Numerical results for a set of benchmark problems are presented in section 6.1.

3 Enumeration-based probabilistic construction

We introduce now a construction procedure based on the enumeration strategy presented in the previous section, but which also integrates a probabilistic factor limiting the size of the enumeration, in order to try to quickly generate good solutions. This strategy is based on doing an (possibly partial) enumeration for some variables, while other variables are simply rounded to one of the integers closest to their value in the current LP relaxation.

Algorithm 2: Integer variables' enumeration with node pruning.
ENUMERATE($\bar{x}$, x^*, $\mathcal{F}$, $\mathcal{V}$)
(1) determine x^{LP}, z and ζ through Equation 3
(2) **if** $\mathcal{V} = \{\}$
(3) **if** $\bar{x} = x^{LP}$ is better than x^*
(4) $x^* := \bar{x}$
(5) **return**
(6) **if** $\zeta > 0$, or $\zeta = 0$ and $z \geq z^*$
(7) **return**
(8) select index i from $\mathcal{V}$
(9) $\mathcal{F} := \mathcal{F} \cup \{i\}$
(10) $\mathcal{V} := \mathcal{V} \backslash \{i\}$
(11) $k =$ closest integer to x_i^{LP}
(12) $\mathcal{J} := \{k\}$
(13) **foreach** $j \in \{k+1, k-1, \ldots\} \ : \ l_i \leq j \leq u_i$
(14) $\mathcal{J} := \mathcal{J} \cup \{j\}$
(15) **foreach** $j \in \mathcal{J}$
(16) fix $\bar{x}_i := j$
(17) ENUMERATE($\bar{x}$, x^*, $\mathcal{F}$, $\mathcal{V}$)
(18) release $\bar{x}_i$
(19) $\mathcal{F} := \mathcal{F} \backslash \{i\}$
(20) $\mathcal{V} := \mathcal{V} \cup \{i\}$

The construction strategy is presented in Algorithm 3, which takes a parameter π, specifying the probability of a variable being selected for enumeration (as opposed to rounding). This parameter also specifies the probability of using each value of the variable's domain on the enumeration. If a variable i is selected for enumeration, the integer closest to x_i^{LP} (its value at the current LP relaxation) is selected for enumeration with probability one; all the other values in $[l_i, u_i]$ are selected with probability π.

We denote a continuous random variable with uniform distribution within $[0, 1]$ by $\mathscr{R}$.

4 Solution improvements

After a solution is constructed, we do a local search on the each integer variable individually, keeping all the other variables fixed.

Finally, we attempt an intensification on this solution, by releasing some of the integer variables on running a partial branch-and-bound to determine their optimal value subject to the values of the other, fixed variables.

On this setting, both local search and intensification make sense only if the constructed solution if feasible. Therefore, if the construction fails, a new construction starts without these two steps.

4.1 Local search

We present here a simple local search method, based on depth-first hill-climbing on a neighborhood $N(\bar{x})$ consisting of the solutions which have all variables identical to $\bar{x}$ except for one index. The best of such neighbors for an index j can be determined solving Equation 4, where $\mathcal{I} = \{1, \ldots, n\}$.

$$\min_{x,y}\{cx + hy : Ax + Gy \geq b, y \in \mathbb{R}_+^p, \ x_j \in \mathbb{Z}, \ x_i = \bar{x}_i \ \forall\, i \in \mathcal{I} \backslash \{j\}\} \tag{4}$$

Algorithm 3: Construction procedure.

$\textsc{Construct}(\bar{x}, x^*, \mathcal{F}, \mathcal{V}, \pi)$

(1) determine x^{LP}, z and ζ through Equation 3
(2) **if** $\mathcal{V} = \{\}$
(3) **if** x^* not initialized **or** $\bar{x} = x^{LP}$ is better than x^*
(4) $x^* := \bar{x}$
(5) **return**
(6) **if** $\zeta > 0$, or $\zeta = 0$ and $z \geq z^*$
(7) **return**
(8) randomly select index i from $\mathcal{V}$
(9) $\mathcal{F} := \mathcal{F} \cup \{i\}$
(10) $\mathcal{V} := \mathcal{V} \backslash \{i\}$
(11) **if** $\mathscr{R} > \pi$
(12) **if** $\mathscr{R} < x_i^{LP} - \lfloor x_i^{LP} \rfloor$
(13) $\bar{x}_i := \lceil x_i^{LP} \rceil$
(14) **else**
(15) $\bar{x}_i := \lfloor x_i^{LP} \rfloor$
(16) $\textsc{Construct}(\bar{x}, x^*, \mathcal{F}, \mathcal{V}, \pi)$
(17) **else**
(18) $k =$ closest integer to x_i^{LP}
(19) $\mathcal{J} := \{k\}$
(20) **foreach** $j \in \{k+1, k-1, \ldots\} \ : \ l_i \leq j \leq u_i$
(21) **if** $\mathscr{R} > \pi$
(22) $\mathcal{J} := \mathcal{J} \cup \{j\}$
(23) **foreach** $j \in \mathcal{J}$
(24) $\bar{x}_i := j$
(25) $\textsc{Construct}(\bar{x}, x^*, \mathcal{F}, \mathcal{V}, \pi)$
(26) release $\bar{x}_i$
(27) $\mathcal{F} := \mathcal{F} \backslash \{i\}$
(28) $\mathcal{V} := \mathcal{V} \cup \{i\}$

The actual implementation of the search of neighborhood $N(\bar{x})$ is done by fixing all the variables x at $\bar{x}$, except for an index j, and determining the optimal value of the variable x_j by a (single integer variable) branch-and-bound.

Local search is done depth-first, i.e., a neighbor that improves the quality of the current solution is immediately accepted, as described in Algorithm 4. In the local search routine we will do as many movements as required for the current solution to have no neighbor better that itself, as shown in Algorithm 5.

Given the quality that is expected from the construction algorithm, the improvements due to local search are expected to be relatively few; anyway, as the computational burden of this search is comparatively small, it is worthy to try a local search at the end of every construction.

4.2 Intensification

The process of releasing one integer variable, while keeping the other fixed, can be extended: we might release more variables, letting branch-and-bound do a deeper search, which we call intensification. On one extreme, if all the variables are released, there would be a pure branch-and-bound; on the other, if no variable is released, there would

Algorithm 4: Solution improvement.
IMPROVE($\bar{x}$)
(1) $\mathcal{I} := \{1, \ldots, n\}$
(2) **while** $\mathcal{I} \neq \{\}$
(3) randomly select $j \in \mathcal{I}$
(4) solve Equation 4, determining x_j
(5) **if** x is better than $\bar{x}$
(6) **return** x
(7) $\mathcal{I} := \mathcal{I} \backslash \{j\}$
(8) **return** $\bar{x}$

Algorithm 5: Local search routine.
LOCALSEARCH($\bar{x}$)
(1) $s :=$ IMPROVE($\bar{x}$)
(2) **while** $s \neq \bar{x}$
(3) $\bar{x} := s$
(4) $s :=$ IMPROVE($\bar{x}$)
(5) **return** $\bar{x}$

be no intensification. We propose to use a parameter ρ specifying the probability of any individual variable to be selected for being released; with $\rho = 1$ all variables would be selected, with $\rho = 0$ no variable is selected.

The problem corresponding to the intensification procedure is formulated in Equation 5, where $\mathcal{V}$ is the set of released variables, and $\mathcal{F}$ the set of variables kept fixed.

$$\min_{x,y}\{cx + hy : Ax + Gy \geq b, y \in \mathbb{R}^p_+,\ x_j \in \mathbb{Z}\ \forall\, j \in \mathcal{V},\ x_i = \bar{x}_i\ \forall\, i \in \mathcal{F}\} \tag{5}$$

The intensification procedure is presented in Algorithm 6.

Algorithm 6: Intensification.
INTENSIFY($\bar{x}, \rho, i$)
(1) $\mathcal{F} := \{\}$
(2) $\mathcal{V} := \{1, \ldots, n\}$
(3) **foreach** $i \in \{1, \ldots, n\}$
(4) **if** $\mathscr{R} < \pi$
(5) $\mathcal{F} := \mathcal{F} \cup \{i\}$
(6) $\mathcal{V} := \mathcal{V} \backslash \{i\}$
(7) solve Equation 5
(8) **if** no integer solution was found
(9) **return** $\bar{x}$
(10) **return** x, the solution of Equation 5

5 The Complete Meta-heuristic

5.1 Parameterized version

The elements presented in the previous sections are combined to form a complete meta-heuristic, in Algorithm 7. The input parameters are the number of iterations T, and the probability of selection for enumeration π, the probability of selection for intensification ρ, and the name of the file containing the instance to solve.

Algorithm 7: A complete meta-heuristic — parameterized version.
MetaHeuristic(T, *seed*, π, ρ, *MPSfile*)
(1) read global data A, G, b, c, and h from *MPSfile*
(2) initialize random number generator with *seed*
(3) $t = 0$
(4) $\mathcal{F} := \{\}$
(5) $\mathcal{V} := \{1, \ldots, n\}$
(6) **while** $t \leq T$
(7) Construct($\bar{x}, x^*, \mathcal{F}, \mathcal{V}, \pi$)
(8) $\bar{x} :=$ LocalSearch($\bar{x}$)
(9) $\bar{x} :=$ Intensify($\bar{x}, \rho$)
(10) **if** x^* not initialized **or** $\bar{x}$ better than x^*
(11) $x^* := \bar{x}$
(12) $t = t + 1$
(13) **return** x^*

This algorithm has the nice property of permitting the user to select the amount of effort to use in enumeration, intensification, and in rounding construction and local search. In particular, with $\pi = 1$ there will be a complete enumeration; with $\rho = 1$ there will be a complete branch-and-bound; and with $\pi = \rho = 0$ there will construction based on random rounding (with no enumeration), complemented by local search.

However, there is a significant difficulty on the usage of this algorithm, on finding a good, general parameterization. The appropriate parameters for a given problem can produce very poor results if used on a different one. In particular, for easy problems it seems to be more appropriate to run the full enumeration or branch-and-bound, as in these cases in addition to obtaining the optimal solution we also obtain the proof that it is optimal.

We next propose a way of exploring two issues: varying the parameters π and ρ dynamically during the search, and allowing a limited CPU time per iteration.

5.2 Parameter-free version

In order to do the search in a fashion that is as much as possible "parameter-free", we propose to randomly determine the parameters that control the previous algorithm, π and ρ, in each iteration, with uniform probability in the interval $[0, 1]$, as specified in Algorithm 7. This has the inconvenient of potentially producing iterations where too much CPU time is spent, if either of the parameters is close to one. To tackle this, we have to limit the CPU time spent both in enumeration and in intensification, for each iteration. This, again, we do in a randomized way, by setting the limit CPU time of either procedure to be a random fraction of the number of variables of the problem. The choice of the number of variables as a basis for CPU allowance is somewhat arbitrary, somewhat intuitive, but seems to produce reasonable results.

Algorithm 8: A complete meta-heuristic — parameter-free version.
MetaHeuristic(T, *seed*, *MPSfile*)
(1) read global data A, G, b, c, and h from *MPSfile*
(2) initialize random number generator with *seed*
(3) $t = 0$
(4) $\mathcal{F} := \{\}$
(5) $\mathcal{V} := \{1, \ldots, n\}$
(6) **while** $t \leq T$
(7) $\pi = \mathscr{R}$
(8) Construct($\bar{x}, x^*, \mathcal{F}, \mathcal{V}, \pi$) (allow $\mathscr{R} \times n$ seconds, max.)
(9) $\bar{x} :=$ LocalSearch($\bar{x}$)
(10) $\rho = \mathscr{R}$
(11) $\bar{x} :=$ Intensify($\bar{x}, \rho$) (allow $\mathscr{R} \times n$ seconds, max.)
(12) **if** x^* not initialized **or** $\bar{x}$ better than x^*
(13) $x^* := \bar{x}$
(14) $t = t + 1$
(15) **return** x^*

6 Results

6.1 Complete enumeration

The performance of the complete enumeration is somewhat comparable to that of branch-and-bound, especially for problems with all integer variables binary. We must however note that the results obtained by enumeration are extremely dependent on the sequence of enumeration; a variation of two orders of magnitude on the CPU time is often observed when the variables are enumerated in different sequences. For the results presented here, the variables were enumerated on the order of their appearance in the input file.

Complete primal-based enumeration results, as well as results obtained utilizing the B&B implementation of *GLPK*, on the series of benchmark problems selected, are provided in the Table 1. The maximum CPU time allowed is 24 hours; in case this limit is 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 [7] for further details).

6.2 Meta-heuristic (parameter-free version)

We present results obtained by the parameter-free meta-heuristic in table 2[1]. Comparison of these results with those of the complete enumeration and branch-and-bound, provided in the previous section, shows that for easy problems, branch-and-bound and the complete enumeration are generally preferable. For large problems, the meta-heuristic is likely to be preferable, as it quickly provides solutions for problems on which the exact approaches use a very large amount of CPU time.

The results can also be compared to those obtained by a tabu search implementation, which are provided in appendix D, table 4. This comparison shows that the results are in general favorable to tabu search. We believe that one of the main weaknesses of the enumeration-based meta-heuristic resides in parameterization, which remains a subject

[1]The number of iterations allowed in this experiment was probably too small, as we observed that in most of the cases improving solutions were found in the last iterations. A more thorough experiment is currently under way.

problem	Branch-and-bound			Primal enumeration		
name	CPU (s)	best z	remarks	CPU (s)	best z	remarks
bell3a	124	878430.32		111	878430.32	
bell5	132	8966406.49		2372	8966406.49	
egout	3.3	568.1007		657	568.101	
enigma	13	0		5.3	0	
flugpl	1.2	1201500		0.07	1201500	
gt2	>24h	30161*	stopped	>24h	(not found)	stopped
lseu	89	1120		64	1120	
mod008	47	307		58	307	
modglob	>24h	20815372.17*	stopped	>24h	29059400*	stopped
noswot	127	-41*	failed	>24h	-40*	stopped
p0033	1	3089		1.1	3089	
pk1	45781	11		9413	11	
pp08a	>24h	7350	stopped	>24h	7790*	stopped
pp08aCUT	>24h	7350	stopped	>24h	7920*	stopped
rgn	3.8	82.2		2.9	82.2	
stein27	3.6	18		6.3	18	
stein45	248	30		488	30	
vpm1	9450	20		>24h	21*	stopped

Table 1: Results obtained by the complete enumeration, and by branch-and-bound, using *GLPK - version4.4*. Solvers were interrupted when 24 hours of CPU time have been spent. (* indicates non-optimal solutions.)

problem	best	best	%above	%feas	Feasibility	%best	Best sol.	%opt	Optimality
name	z	ζ	optim.	runs	($E[t^f]$(s))	runs	($E[t^f]$(s))	runs	($E[t^f]$(s))
bell3a	878430.32	0	0	95	10.86	95	31.11	95	31.11
bell5	8966406.5	0	0	100	28.19	75	283.6	75	283.64
egout	568.101	0	0	100	0.19	95	52.49	95	52.49
enigma	0	0	0	80	572.70	65	728.5	65	728.46
flugpl	1201500	0	0	60	14.78	60	15.37	60	15.37
gt2	21166	0	0	100	18.24	15	5524.	15	5523.96
lseu	1120	0	0	100	0.12	65	174.7	65	174.72
mod008	307	0	0	100	0.59	95	33.33	95	33.33
modglob	20781300.2	0	0.20	100	0.16	5	6211.	0	$\gg$ 6375.75
noswot	-41	0	4.65	95	21.29	45	491.4	0	$\gg$ 7900.36
p0033	3089	0	0	100	0.95	100	1.449	100	1.45
pk1	14	0	27.3	100	0.034	5	2747.	0	$\gg$ 2833.90
pp08a	7460	0	1.50	100	0.085	5	5346.	0	$\gg$ 5627.83
pp08aCUT	7370	0	0.27	100	0.11	5	4975.	0	$\gg$ 5004.99
rgn	82.2	0	0	100	0.033	100	0.086	100	.086
stein27	18	0	0	100	0.022	100	0.036	100	.036
stein45	30	0	0	100	0.065	70	20.78	70	20.79
vpm1	20	0	0	100	0.20	75	456.1	75	456.07

Table 2: Parameter-free meta-heuristic (Algorithm 8): 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 1000 iterations)

to be further studied.

We also present results obtained utilizing the commercial solver in appendix E, table 5. Although there are some exceptions, this solver is generally much faster than meta-heuristics, although, as the code is not open, we cannot determine why. Differences might be due to the use of branch-and-cut, the variable choice in branch-and-bound, the quality of the LP solver, and preprocessing.

7 Conclusion

In the first part of this paper we presented a primal-based, complete enumeration algorithm for the solution of MIPs. The performance of this algorithm is in most of the cases comparable to that of branch-and-bound, though in some cases it is much worse. The choice of a good heuristic for variable selection on the enumeration is likely to provide interesting results, as there is a very high dependence of the performance of the algorithm on the enumeration order. This complete enumeration strategy might be appealing for the solution of non-linear integer and mixed integer problems, as there is no general, exact optimal branch-and-bound strategy for these cases.

We presented several ideas about partial enumeration and its integration in a meta-heuristic, with an application for general MIP. To the best of our knowledge, this type of integration is applied here for the first time; also new, to some extent, is the method of partially destroying a solution at the end of an iteration for providing a start solution to the next one. This partial destruction, presently rather blind, should be further studied, in order to try to actively preserve "good" parts of the solution.

The probabilistic enumeration strategy proposed in this paper provides a way for quickly solving most of the problems analyzed, even though it is generally not as good as tabu search on the selected benchmarks. These are the first attempts on the integration of enumeration in a meta-heuristic, and should therefore be further explored.

The readers interested in more details on the implementation of the strategies presented in this paper are referred to the *C* programming code, available in [9].

A Benchmark Problems

The instances of MIP and IP problems used as benchmarks are defined in the [1] and are presented in Table 3. They were chosen to provide an assortment of MIP structures, with instances coming from different applications.

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. The algorithms of this papar and *GLPK* were all 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

Problem name	Application	Number of variables total	integer	binary	Number of constraints	Optimal solution
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 3: Set of benchmark problems used: application, number of constraints, number of variables and optimal solutions as reported in MIPLIB.

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.

D Tabu search results.

A tabu search approach to the MIP has been presented in [10], on the same computational setup used here. We transcribe those results, for comparison with the enumeration-based meta-heuristic.

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 4: Tabu search results: 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.)

E Results obtained by a commercial solver.

We present here the results obtained by the commercial solver *Xpress-MP Optimizer, Release 13.02*, limiting the CPU time to 24 hours maximum.

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

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

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

[4] Fred Glover. Parametric tabu search for mixed integer programs. Technical report, University of Colorado, Leeds School of Business, 2004.

[5] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search — Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[6] Jr. L. E. Trotter and C. M. Shetty. An algorithm for the bounded variable integer programming problem. *J. ACM*, 21(3):505–513, 1974.

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

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

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

[10] João P. Pedroso. Tabu search for mixed integer programming. In Cesar Rego, editor, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers, 2004. (On press).

[11] L.S. Pitsoulis and M.G.C. Resende. Greedy randomized adaptive search procedures. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 178–183. Oxford University Press, 2002.

[12] Joachim P. Walser. Integer optimization by local search – a domain-independent approach. *Lecture Notes in Artificial Intelligence*, LNAI-1637, 1999.