Chapter 1

GRASP FOR LINEAR INTEGER PROGRAMMING

Teresa Neto
tneto@mat.estv.ipv.pt

João Pedro Pedroso
jpp@ncc.up.pt
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 823
4150-180 Porto, Portugal

Abstract In this paper we introduce a GRASP for the solution of general linear integer problems. The strategy is based on the separation of the set of variables into the integer subset and the continuous subset. The integer variables are fixed by GRASP and replaced in the original linear problem. If the original problem had continuous variables, it becomes a pure continuous problem, which can be solved by a linear program solver to determine the objective value corresponding to the fixed variables. If the original problem was a pure integer problem, simple algebraic manipulations can be used to determine the objective value that corresponds to the fixed variables. When we assign values to integer variables that lead to an impossible linear problem, the evaluation of the corresponding solution is given by the sum of infeasibilities, together with an infeasibility flag.

We report results obtained for some standard benchmark problems, and compare them to those obtained by branch-and-bound and to those obtained by an evolutionary solver.

Keywords: GRASP, Linear Integer Programming

1. Introduction

A wide variety of practical problems can be solved using integer linear programming. Typical problems of this type include lot sizing, scheduling, facility location, vehicle routing, and more; see for example (Nemhauser and Wolsey 1988, Wolsey 1998).

In this paper we introduce a GRASP (greedy randomized adaptive search procedure) for the solution of general linear integer programs. The strategy is based on the separation of the set of variables into the integer subset and the continuous subset (if some). The procedure starts by solving the linear programming (LP) relaxation of the problem. Values for the integer variables are then chosen, through a semi-greedy construction heuristic based on rounding around the LP relaxation, and fixed. The continuous variables (if some) can then be determined in function of them, by solving a linear program where all the integer variables have been fixed by GRASP. Afterwards, local search improvements are made on this solution; these improvements still correspond to changes made exclusively on integer variables, after which the continuous variables are recomputed through the solution of an LP. When the linear program leads to a feasible solution, the evaluation of the choice of the variables is determined directly by the objective function. If the choice of the variables induces an infeasible problem, its evaluation is measured by the sum of infeasibilities.

2. Background

In this paper we focus on the problem of optimizing a linear function subject to a set of linear constraints, in the presence of integer and, possibly, continuous variables. The more general case, where there are integer and continuous variables, is usually called *mixed integer* (MIP).

The general formulation of a mixed integer linear program is

$$\max_{x,y}\{cx + hy : Ax + Gy \leq b, x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p\}, \tag{1.1}$$

where $\mathbb{Z}_+^n$ is the set of nonnegative, n-dimensional integral vectors, and $\mathbb{R}_+^p$ is the set of nonnegative, p-dimensional real vectors. A and G are $m \times n$ and $m \times p$ matrices, respectively, where m is the number of constraints. There are n integer variables (x), and p continuous variables (y).

If the subset of continuous variables is empty, the problem is called *pure integer* (IP); its formulation is

$$\max_{x}\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\}. \tag{1.2}$$

In general, there are additional bound restrictions on the integer variables, stating that $l_i \leq x_i \leq u_i$, for $i = 1, \ldots, n$.

The main idea for the conception of the algorithm described in this paper is provided in (Pedroso 1998; 2002). It consists of fixing the integer variables of a linear integer program by a meta-heuristic—in this case GRASP. For MIP, by replacing these variables on the original formulation, we obtain a pure, continuous LP, whose solution provides an evaluation of the fixed variables. On the case of pure IP, we can compute directly the corresponding objective. We can also directly check feasibility, and compute the constraints' violation.

Notice that this algorithm, as opposed to branch-and-bound, does not work with the solution of continuous relaxations of the initial problem. The solution of LPs is only required for determining the value of the continuous variables, and of the objective that corresponds to a particular instantiation of the integer variables.

Instances of integer linear problems correspond to specifications of the data: the matrix A and the vectors b and c in Equations 1.1 and 1.2 for IPs, and also the matrix G and the vector h for MIPs. The most commonly used representation of instances of these problems is through *MPS* files, which is the format used on this GRASP implementation. We have tested GRASP with a subset of the benchmark problems that are available in this format in the *MIPLIB* (Bixby et al. 1998).

3. GRASP

GRASP (Feo and Resende 1989; 1995, Pitsoulis and Resende 2002, Resende and Ribeiro 2001) is a meta-heuristic based on a multi-start procedure where each iteration has two phases: construction and local search. In the construction phase, a solution is built, one element at a time. At each step of this phase, the candidate elements that can be incorporated to the partial solution are ranked according to a greedy function. The evaluation of these elements leads to the creation of a restricted candidate list (RCL), where a selection of good variables, according to the corresponding value of the greedy function, are inserted. At each step, one element of the RCL is randomly selected and incorporated into the partial solution (this is the probabilistic component of the heuristic). The candidate list and the advantages associated with every element are updated (adaptive component of the heuristic). The number of elements of the RCL is very important for GRASP: if the RCL is restricted to a single element, then only one, purely greedy solution will be produced; if the size of the RCL is not restricted, GRASP produces random solutions. The mean and the variance of the objective value of the solutions built are directly affected by the cardinality of the RCL: if the RCL has more elements, then more different solutions will be produced, implying a larger variance.

The solutions generated in the construction phase generally are not local optima, with respect to some neighborhood. Hence, they can often be improved by means of a local search procedure. The local search phase starts with the constructed solution and applies iterative improvements until a locally optimal solution is found.

The construction and improvement phases are repeated a specified number of times. The best solution over all these GRASP iterations is returned as the result.

In Algorithm 1 we present the structure of a general GRASP algorithm.

Algorithm 1: A general GRASP algorithm.
GRASP()
(1) **while** stopping criterion is not satisfied
(2) $x :=$ SemiGreedy()
(3) $x :=$ LocalSearch(x)
(4) **if** x^* is not initialized **or** x is better than x^*
(5) $x^* := x$
(6) **return** x^*

GRASP has been applied successfully to numerous combinatorial optimization problems in different areas, including routing (Kontoravdis and Bard 1995, Carreto and Baker 2001), scheduling (Feo et al. 1991, Binato et al. 2001), logic (Resende and Feo 1996, Resende et al. 1997), assignment (Li et al. 1994, Robertson 2001). An annotated GRASP bibliography is available in (Festa and Resende 2001).

4. GRASP implementation

In this section we specialize GRASP for the solution of general integer linear problems. We describe the fundamental aspects taken into account in the GRASP implementation, which are presented in Algorithm 2. The parameters of this procedure are the number of iterations, N (used as a stopping criterion), the largest type of neighborhood, k_{max} (see section 4.4), the seed for initializing the random number generator, and the name of the *MPS* file containing the instance's data.

Algorithm 2: A GRASP for integer programming.
GRASP(N, k_{max}, *seed*, *MPSfile*)
(1) read data A, G, b, c, and h from *MPSfile*
(2) initialize random number generator with *seed*
(3) **for** $k = 1$ **to** N
(4) $\bar{x} :=$ SemiGreedy($\bar{x}^{LP}$)
(5) $\bar{x} :=$ LocalSearch($\bar{x}, k_{max}$)
(6) **if** $\bar{x}^*$ is not initialized **or** $\bar{x}$ is better than $\bar{x}^*$
(7) $\bar{x}^* := \bar{x}$
(8) **return** $\bar{x}^*$

4.1 Representation of the solutions

The part of the solution that is determined by GRASP is the subset of integer variables x in Equations 1.1 or 1.2. The data structure representing a solution used by GRASP is therefore an n-dimensional vector of integers, $\bar{x} = (\bar{x}_1 \dots \bar{x}_n)$.

4.2 Evaluation of solutions

The solutions on which the algorithm works may be feasible or not. For the algorithm to function appropriately it has to be able to deal with both feasible and infeasible solutions in the same framework. We describe next the strategies used for tackling this issue on MIPs and IPs.

4.2.1 Mixed integer programs (MIP). 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 GRASP:

$$z = \max_{y}\{c\bar{x} + hy : Gy \leq b - A\bar{x}, y \in \mathbb{R}^p_+\}. \tag{1.3}$$

We are now able to solve this (purely continuous) linear problem using a standard algorithm, like the simplex.

Feasible solutions. If problem 1.3 is feasible, the evaluation given to the corresponding solution is the objective value z, and the solution is labelled feasible.

Infeasible solutions. If problem 1.3 is infeasible, we formulate another LP for the minimization of the infeasibilities. This is accomplished by setting up artificial variables and minimizing their sum (a procedure that is identical to the phase I of the simplex algorithm):

$$\zeta = \min_{s}\{\sum_{k=1}^{m} s_k : Gy \leq b - A\bar{x} + s, y \in \mathbb{R}^p_+ \ , \ s \in \mathbb{R}^m_+\}, \tag{1.4}$$

where m is the number of constraints.

The evaluation attributed to such a solution $\bar{x}$ is the value ζ of the optimal objective of the LP of Equation 1.4, and the solution is labelled infeasible.

4.2.2 Pure integer programs (IP). Fixing all the integer variables in Equation 1.2 leads to no free variables. Feasibility and the objective value can be inspected directly.

Feasible solutions. If the solution $\bar{x}$ fixed by GRASP does not violate any constraint, the evaluation attributed to the corresponding solution is the objective value $z = c\bar{x}$, and the solution is labelled feasible.

Infeasible solutions. If $\bar{x}$ is infeasible, its evaluation is given by the sum of constraint violations. For problems stated in the canonic form of Equation 1.2, this is done by determining:

$$\zeta = \sum_{k=1}^{m} s_k, \ \text{ with } \ s_k = \max\{A_k\bar{x} - b_k, 0\} \ \text{ for } \ k = 1, \ldots m. \tag{1.5}$$

The evaluation given to the solution is the value ζ and the solution is labelled infeasible.

4.2.3 Evaluation data structure. For a feasible solution of an integer linear program, the evaluation is denoted by $\bar{z}$, a data structure consisting of the objective value z and a flag stating that the solution is feasible. For an infeasible solution, $\bar{z}$ consists of the value ζ and an infeasibility flag.

4.2.4 Comparison of solutions. We have to provide a way of comparing solutions, either they are feasible or not. What we propose is to classify solutions in such a way that:

- feasible solutions are always better than infeasible ones;
- feasible solutions are ranked among them according to the objective of the integer linear problem;
- infeasible solutions are ranked among them according to the sum of infeasibilities (i.e., according to a measure of their distance from the feasible set).

We say that a solution structure i *is better* than another structure j if and only if:

- $\zeta^i < \zeta^j$ (i is closer to the feasible region than j);
- $\zeta^i = \zeta^j$, and (for maximization) $z^i > z^j$ (i has a better objective).

4.3 Construction phase

We propose two construction methods, differing in the greedy function and in the number of elements of the restricted candidate list (RCL). Both are based on the solution of the LP relaxation, which we denote by $x^{LP} = (x_1^{LP}, \ldots, x_n^{LP})$. We use the term RCL with a meaning slightly different of the currently used in the GRASP literature. For each index $k \in \{1, \ldots, n\}$ of the variables, we set up a list of the values that we can potentially assign to it. In a purely greedy construction, we always assign the integer closest to the value of the LP relaxation, x_k^{LP}. Hence, the RCL for a variable x_k would have a single element, the closest integer to x_k^{LP}. The RCL for semi-greedy construction has more elements, as explained below.

In Algorithm 3, we present the construction algorithm. Since there are n variables in the solution, each construction phase consists of n steps. The two constructions differ on steps 2 and 3.

4.3.1 Probabilistic rounding construction. This semi-greedy construction is inspired in an algorithm provided in (Lengauer 1990). It consists of rounding each variable i to the integer closest to its

Algorithm 3: Semi-greedy solution construction.
SemiGreedy($\bar{x}^{LP}$)
(1) **for** $k = 1$ **to** n
(2) RCL:= {values allowed to $\bar{x}_k$ }
(3) select an $r \in RCL$ with some probability
(4) $\bar{x}_k := r$
(5) **return** $\bar{x}$

value on the LP relaxation, x_i^{LP}, in a randomized way, according to some rules. The probabilities for rounding up or down each of the variables are given by the distance from the fractional solution x_k^{LP} to its closest integer.

For all the indices $k \in \{1, \ldots, n\}$, the variable $\bar{x}_k$ is equal to the corresponding LP relaxation value rounded down with probability

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

or rounded up with probability $1 - P(\bar{x}_k = \lfloor x_k^{LP} \rfloor)$.

The RCL is built with the two values that each variable $\bar{x}_k$ can take.

4.3.2 Bi-triangular construction. The goal of this strategy is to increase the diversification of the solutions. In the previous construction, we only have two possibilities when we round the value of the LP relaxation of a variable. We here extend this for having the possibility of assigning each variable to any integer between its lower-bound, l_k, and its upper-bound, u_k. The RCL is built with these values, and, again, we give a probability of assignment to each of them based on the solution of the LP relaxation.

The probability density function that we considered is composed by two triangles (*bi-triangular distribution*) and is defined by three parameters: the minimum $a = l_k - 0.5$, the maximum $b = u_k + 0.5$, and the mean $c = x_k^{LP}$. The values a and b were considered in order to have a non-zero probability of rounding to l_k and u_k. The bi-triangular density function is represented in Figure 1.1.

The area of the left triangle is proportional to the distance between c and b and the area of the right triangle is proportional to the distance between a and c. The combined areas must be one, since it is a density function. The probability density function is given by:

$$f(x) = \begin{cases} \frac{2(b-c)(x-a)}{(b-a)(c-a)^2} & \text{if } a \leq x \leq c, \\ \frac{2(c-a)(b-x)}{(b-a)(b-c)^2} & \text{if } c < x \leq b, \\ 0 & \text{otherwise,} \end{cases}$$

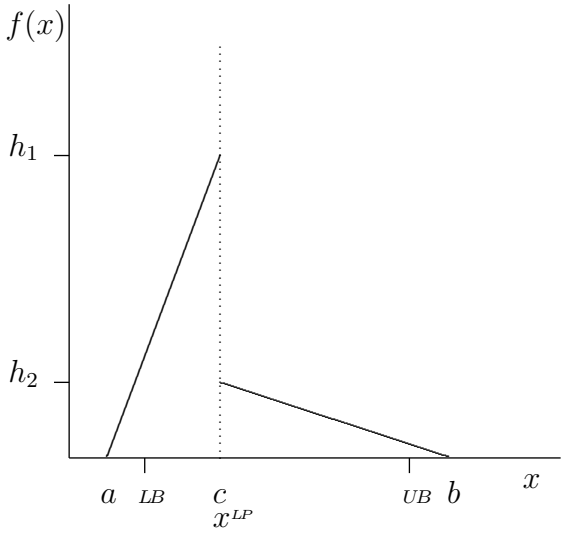

Figure 1.1. Bi-triangular density function.

and the distribution function is:

$$F(x) = \begin{cases} 0 & \text{if } x < a, \\ \frac{(b-c)(x-a)^2}{(b-a)(c-a)^2} & \text{if } a \le x \le c, \\ 1 - \frac{(c-a)(b-x)^2}{(b-a)(b-c)^2} & \text{if } c < x \le b, \\ 1 & \text{if } x > b. \end{cases}$$

The mean for this distribution is c, which corresponds to the value of the LP relaxation.

With this construction, the value for each variable x_k is obtained by drawing a random number with this distribution with $a = l_k - 0.5$, $b = u_k + 0.5$, and $c = x_k^{LP}$, and then rounding it to the closest integer.

4.4 Local search

Local search tries to improve the quality of a solution by hill climbing on its neighborhood, according to one of the improvement methods that are described next. For this purpose we propose neighborhoods that consist of incrementing or decrementing variables, one at a time or simultaneously. The main idea behind the definition of these neighborhoods is the extension to the case of integer values of the idea presented in (Resende and Feo 1996) for the case of binary variables. The local search procedure iterates up to the point where the improvement procedure does not lead to a better solution.

4.4.1 Increment neighborhood. The *increment* neighborhood of a solution x, $N^1(x)$, is composed of solutions which differ from

x on one element x_j, whose value is one unit above or below x_j. Hence y is a neighbor solution of x if for one index i, $y_i = x_i + 1$, or $y_i = x_i - 1$, with $y_j = x_j$ for all indices $j \neq i$:

$$N^1(x) = \{\ y \in \mathbb{Z} : y \text{ can be obtained from } x \text{ by adding or subtracting one unit to an element of } x\}.$$

The idea used on this neighborhood can be extended to the case where we change more than one variable at the same time. For example, the *2-increment* neighborhood of a solution x, $N^2(x)$, is composed of solutions which differ from x on two elements x_i and x_j, whose values are one unit above or below the original ones. Hence y is a neighbor solution of x if for two indices i, j we have $y_i = x_i + 1$ or $y_i = x_i - 1$, and $y_j = x_j + 1$ or $y_j = x_j - 1$, with $y_l = x_l$ for all indices $l \neq i, l \neq j$.

More generally, we can define the *k-increment* neighborhood (for problems with $n \geq k$ integer variables) as:

$$N^k(x) = \{\ y \in \mathbb{Z} : y \text{ can be obtained from } x \text{ by adding or subtracting one unit to } k \text{ of its elements}\}.$$

When k increases, the number of neighbors of a solution increases exponentially. In order to reduce the size of the set of neighbors that is more frequently explored, we devised the following strategy. Let O be the set of indices of variables which have coefficients that are different of zero in the objective function:

$$O = \{i : c_i \neq 0, \text{ for } i = 1, \ldots, n\}.$$

Let $V(x)$ be the set of indices of variables (if some) which have coefficients different of zero on constraints violated by x:

$$V(x) = \{j : a_{ij} \neq 0, \text{ for } i \in \{\text{constraints violated by } x\}\}.$$

For a solution x, the subset of neighbors with at least one index in the sets O (for feasible x) or $V(x)$ (for infeasible x) compose the neighborhood $N^{k^*}(x) \subseteq N^k(x)$ which is explored first. The subset $N^{k'}(x) = N^k(x) \setminus N^{k^*}(x)$ is explored when a local optimum of N^{k^*} has been found. Neighborhoods N^k are explored in increasing order on k.

This strategy for exploring the neighborhoods is called *variable neighborhood search* (Hansen and Mladenovic 2001). It consists of searching first restricted neighborhoods which are more likely to contain improving solutions; when there are no better solutions in a restricted neighborhood, this is enlarged, until having explored the whole, unrestricted neighborhood.

4.4.2 Improvements. There are two methods for updating the best solution when searching a particular neighborhood.

The first one, called *breadth-first*, consists of searching the best solution y in the entire neighborhood of a solution x. If it is better than

the current solution x, then x is replaced by y; otherwise, x is a local optimum. This method will hence search the whole neighborhood of the current solution, even if improving solutions can be found on the early exploration of the neighborhood.

The second method, called *depth-first*, consists of replacing x whenever the neighbor generated, y, is better than x. In this case, the subsequent neighbor z is generated from y.

Empirically, the computational time required to obtain a local optimum is longer with the first method, and premature convergence is more likely to occur. Therefore, in this implementation the search is made depth-first.

4.4.3 Local search algorithm. The local search method is presented in Algorithm 4. The improvement of a solution is made in the routine IMPROVE(x, k_{max}). This procedure, presented in Algorithm 5, searches first the neighborhood $N^{1^*}(x)$ and returns the first neighbor better than x found. If no such a neighbor is found, it switches to $N^{1'}(x)$. When no improving solution is found also in this neighborhood, the method explores $N^2(x)$ (first checking in $N^{2^*}(x)$), and so on, until having explored $N^{k_{max}}$.

Algorithm 4: Local search main cycle.
LOCALSEARCH(x, k_{max})
(1) $s :=$ IMPROVE(x, k_{max})
(2) **while** $s \neq x$
(3) $x := s$
(4) $s :=$ IMPROVE(x, k_{max})
(5) **return** s

Algorithm 5: Improvements without hunt search.
IMPROVE(x, k_{max})
(1) $k := 1$
(2) **while** $k \leq k_{max}$
(3) $S := N^{k^*}(x)$
(4) **while** $S \neq \{\}$
(5) $s :=$ RandomChoice(S)
(6) **if** s is better than x
(7) **return** s
(8) $S := S \backslash \{s\}$
(9) $S := N^{k'}(x)$
(10) **while** $S \neq \{\}$
(11) *do the same as in steps (5) to (8)*
(12) $k := k + 1$
(13) **return** x

Algorithms 6 and 7 present a strategy called *hunt search*. It was originally conceived for locating values in an ordered table, and is used here for quickly exploring large ranges, when lower and upper bounds of some variable are far apart. The step added to a variable, Δ, is initially $+1$ or -1, and is doubled until no improvements are obtained, or until reaching a bound of the variable.

Algorithm 6: Improvements with hunt search.

IMPROVE(x, k_{max})
(1) $k := 1$
(2) **while** $k \leq k_{max}$
(3) $S := N^{k^*}(x)$
(4) **while** $S \neq \{\}$
(5) $s :=$ RandomChoice(S)
(6) **if** s is better than x
(7) $i :=$ any index such that $s_i \neq x_i$
(8) $\Delta := s_i - x_i$
(9) $l_i :=$ lower bound of the variable s_i
(10) $u_i :=$ upper bound of the variable s_i
(11) $s :=$ HUNTSEARCH(s, i, Δ, l_i, u_i)
(12) **return** s
(13) $S := S \backslash \{s\}$
(14) $S := N^{k'}(x)$
(15) **while** $S \neq \{\}$
(16) *do the same as in steps (5) to (13)*
(17) $k := k + 1$
(18) **return** x

Algorithm 7: Hunt search on a given index.

HUNTSEARCH(x, i, Δ, l_i, u_i)
(1) **while true**
(2) $s := x$
(3) $\Delta := \Delta \times 2$
(4) $s_i := s_i + \Delta$
(5) **if** $s_i + \Delta \geq u_i$
(6) $s_i := u_i$
(7) **else if** $s_i + \Delta \leq l_i$
(8) $s_i := l_i$
(9) **if** s is better than x
(10) $x = s$
(11) **else**
(12) **return** x

5. Benchmark problems

Languages for mathematical programming are the tools more commonly used for specifying a model, and generally allow transforming the mathematical model into an *MPS* file. As the heuristic that we describe can be used for any MIP model that can be specified as a mathematical program, we have decided to provide the input to the heuristic through *MPS* files. GRASP starts by reading an *MPS* file, and stores the information contained there into an internal representation. The number of variables and constraints, their type and bounds, and all the matrix information is, hence, determined at runtime.

We report results obtained for some standard benchmark problems. The instances of MIP and IP problems used as benchmarks are defined in the *MIPLIB* (Bixby et al. 1998) and are presented in Table 1.1. They were chosen to provide an assortment of MIP structures, with instances coming from different applications.

Notice that the MIPLIB problems are minimizations[1].

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

6. Computational results

We compare the results obtained with GRASP to those obtained by branch-and-bound (B&B)—the classical algorithm for solving general linear integer programs—, and to those obtained by another metaheuristic based on evolutionary computation (Pedroso 1998). B&B starts with a continuous relaxation of the integer linear program and finds the

optimal solution by a systematic division of the domain of the relaxed problem. The evolutionary solver is based on ideas similar to those employed here for solution representation and improvement, but uses population-based methods.

The software implementing the branch-and-bound algorithm used in this experiment is called *lp_solve* (Berkelaar and Dirks). It also comprises a solver for linear programs based on the simplex method, which was used both on this GRASP implementation and on (Pedroso 1998) for solving Equations 1.3 and 1.4^2. Notice that the LPs solved by GRASP at the time of solution evaluation (if some) are often much simpler than those solved by B&B; as all the integer variables are fixed, the size of the LPs may be much smaller. Hence, numerical problems that the LP solver may show up in B&B, do not arise for LPs formulated by GRASP. Therefore, the comparison made in terms of objective evaluations/LP solutions required favors B&B.

This section begins presenting the results obtained using B&B. Statistical measures used in order to assess the empirical efficiency of GRASP are defined next. Follow the results obtained using GRASP, and a comparison of results of GRASP, B&B, and an evolutionary algorithm.

6.1 Results obtained using branch-and-bound

The results obtained using B&B on the series of benchmark problems selected are provided in the Table 1.2. The maximum number of LPs solved in B&B was limited to 100 million; in cases where this was exceeded, the best solution found within that limit is reported.

Problem name	Best solution found	Number of LPs solved	CPU time	Remarks
bell3a	878430.32	438587	170.55	Optimal solution
bell5	8966406.49	420499	81.61	Optimal solution
egout	562.273	55057	6.06	Incorrect solution
enigma	0	8947	1.8	Optimal solution
flugpl	1201500	1588	0.06	Optimal solution
gt2	-	-	-	Failed (unknown error)
lseu	1120	236207	23.97	Optimal solution
mod008	307	2848585	844.09	Optimal solution
modglob	26308600	> 1.00E+08	47678	Stopped
noswot	-25	3753	3.72	Incorrect solution
p0033	3089	7393	0.25	Optimal solution
pk1	11	3710343	2467.23	Optimal solution
pp08a	9770	> 1.00E+08	29924	Stopped
pp08acut	8110	> 1.00E+08	161880	Stopped
rgn	82.1999	4963	1.2	Optimal solution
stein27	18	11985	2.62	Optimal solution
stein45	30	236453	218.57	Optimal solution
vpm1	22	18354	24.95	Incorrect solution

Table 1.2. Results obtained using branch-and-bound: best solution found, number of LPs solved and CPU time.

6.2 Statistical measures

In order to assess the empirical efficiency of GRASP, we provide measures of the expectation of the number of LP solutions and CPU time required for finding a feasible solution, the best solution found, and the optimal solution, for MIP problems. These measures are similar for IP problems, but instead of being developed in terms of the number of LP solutions, they are made in terms of the number of calls to the objective function (Equations 1.2 or 1.5).

The number of GRASP independent iterations (or *runs*) for each benchmark problem in a given experiment is denoted by N.

6.2.1 Measures in terms of the number of LP solutions. Let r^f, r^b and r^o be the number of runs in which a feasible, the best, and the optimal solution were found, respectively. Let n_k^f be the number of objective evaluations required for obtaining a feasible solution in iteration k, or the total number of evaluations in that run if no feasible solution was found. Identical measures for reaching optimality and the best solution found by GRASP are denoted by n_k^o and n_k^b, respectively. Then, the expected number of evaluations for reaching feasibility, based on these N iterations, is:

$$E[n^f] = \sum_{k=1}^{N} \frac{n_k^f}{r^f}.$$

Equivalently, the expected number of evaluations for reaching the best GRASP solution is

$$E[n^b] = \sum_{k=1}^{N} \frac{n_k^b}{r^b},$$

and the expected number of evaluations for reaching optimality is

$$E[n^o] = \sum_{k=1}^{N} \frac{n_k^o}{r^o}.$$

In case $r^o = 0$, the sum of the evaluations of the total experiment (N iterations) provides a lower bound on the expectations for optimality. The same for feasibility, when $r^f = 0$.

6.2.2 Measures in terms of CPU time. 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 GRASP, respectively. Then, the expected CPU time required for reaching feasibility, based on these N iterations, is:

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

while

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

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

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

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.

6.3 Results obtained using GRASP

In this section we provide a series of results comparing the several strategies that were implemented: the two construction methods, the k-increment neighborhood (N^k) for $k = 1$ and $k = 2$, explored with and without hunt search.

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 1.4 GHz, with 512 Gb of RAM. GRASP was implemented on the C++ programming language.

6.3.1 Hunt search. We started making an experiment for assessing the validity of hunt search (HS), with 1000 GRASP iterations with the 1-increment neighborhood (N^1), and 100 iterations with the 2-increment neighborhood (N^2) (the total time spent in each of the two cases is roughly the same). We used both probabilistic rounding construction (PRC) and bi-triangular construction (BTC).

We report results obtained for instances with non-binary integer variables—*bell3a*, *bell5*, *flugpl*, *gt2* and *noswot* (hunt search does not apply when all the integer variables are binary). Table 1.3 reports the percent distance from the best solution found to the optimal solution[3], for N^1 and N^2, respectively ("-" means that the best solution found is not feasible). Table 1.4 reports the expected number of LP solutions/evaluations for obtaining feasibility. The same results for obtaining optimality are reported in Table 1.5.

Comparing these two strategies (GRASP with and without hunt search), we conclude that, in general, hunt search slightly improves the results. This improvement is more significant for the bi-triangular construction: as the constructed solutions are more likely to be far away from local optima, hunt search has more potential for operating.

In the experiments that follow, GRASP was implemented with hunt search.

	Probabilistic rounding		Bi-triangular		
Problem name	without HS	with HS	without HS	with HS	Neighborhood
bell3a	0.4676	0.2762	1.171	1.325	
bell5	1.365	-	-	-	
flugpl	-	-	-	-	N^1
gt2	-	198.8	222.9	151.8	
noswot	4.651	4.651	4.651	4.651	
bell3a	0	0	0	0	
bell5	0.4012	-	-	-	
flugpl	-	-	-	-	N^2
gt2	141.9	94.06	154.4	102.5	
noswot	4.651	4.651	4.651	4.651	

Table 1.3. Percent distance above optimum observed with and without hunt search, for N^1 (on 1000 iterations) and N^2 (on 100 iterations), and for both construction methods: probabilistic rounding and bi-triangular.

	Probabilistic rounding		Bi-triangular		
Problem name	without HS	with HS	without HS	with HS	Neighborhood
bell3a	60.11	27.45	178.01	43.92	
bell5	730448	>174782	>3957140	>187457	
flugpl	>44720	>47649	>112475	>78854	N^1
gt2	>521465	14450	189306	9538	
noswot	68.29	79.26	40.11	37.14	
bell3a	60.54	27.77	181.63	39.47	
bell5	331254	>226731	>624409	>213884	
flugpl	>24328	>24857	>35634	>33071	N^2
gt2	57324	8990	39384	6948	
noswot	190.46	163.01	90.56	36.32	

Table 1.4. Expected number of objective evaluations for finding a feasible solution with and without hunt search, for N^1 and N^2, and for both construction methods: probabilistic rounding and bi-triangular.

	Probabilistic rounding		Bi-triangular		
Problem name	without HS	with HS	without HS	with HS	Neighborhood
bell3a	>205578	>351353	>1964395	>612769	
bell5	>730591	>174782	>3957140	>187457	
flugpl	>44720	>47649	>112475	>78854	N^1
gt2	>521465	>506547	>568958	>539824	
noswot	>294254	>439764	>643568	>342003	
bell3a	6513	10630	17147	33637	
bell5	>352317	>226731	>624409	>213884	
flugpl	>24328	>24857	>35634	>33071	N^2
gt2	>4752588	>5594115	>4824420	>5563503	
noswot	>765014	>876859	>871802	>778050	

Table 1.5. Expected number of objective evaluations for finding an optimal solution with and without hunt search, for N^1 and N^2, and for both construction methods: probabilistic rounding and bi-triangular.

6.3.2 Construction methods. The next experiment was conceived in order to assess the influence of the construction method in the GRASP performance, and to choose one of the methods for later comparison with other methods. The results presented are based on a sample obtained with 1000 GRASP iterations for N^1, and 100 iterations for N^2.

In Tables 1.6 and 1.7 we compare probabilistic rounding to bi-triangular construction. The comparison is made in terms of the percent distance above optimum, and on the expected number of LP solutions for reaching feasibility and optimality.

	Neighborhood N^1		Neighborhood N^2	
Problem name	PRC	BTC	PRC	BTC
bell3a	0.2762	1.325	0	0
bell5	-	-	-	-
egout	10.12	11.22	0	1.387
enigma	-	-	-	-
flugpl	-	-	-	-
gt2	198.8	151.8	94.06	102.5
lseu	2.589	4.821	0	1.429
mod008	0	0.3257	0	0
modglob	0.09060	0.08530	0	0
noswot	4.651	4.651	4.651	4.651
p0033	-	-	0.1942	0
pk1	100	100	63.64	45.45
pp08a	1.769	0.1361	0.1361	0.6803
pp08acut	0	0.5442	0.1361	0
rgn	0	0	0	0
stein27	0	0	0	0
stein45	3.33	3.33	3.33	3.33
vpm1	0	0	0	0

Table 1.6. Comparison between probabilistic rounding and bi-triangular construction: percent distance above optimum observed for neighborhoods N^1 (on 1000 iterations) and N^2 (on 100 iterations).

The results show that probabilistic rounding is in general preferable to the bi-triangular construction, if we take into account the computational burden. Hence, probabilistic rounding is the construction method used for comparing GRASP to other approaches.

6.3.3 Neighborhoods. We now present results of an experiment conceived for assessing the influence of the neighborhoods used, and to choose one of N^1 and N^2 for comparing GRASP to other methods. As the distinction between these results is less clear than the preceding, they are now based on a larger sample of 10000 GRASP iterations for N^1, and 1000 iterations for N^2. The results are reported in the Tables 1.8 and 1.9.

The results show a superiority of the N^2 neighborhood for most of the instances, both in terms of solution quality and expected evaluations, or

	Feasibility ($E[n^f]$)		Optimality ($E[n^o]$)		
Problem name	PRC	BTC	PRC	BTC	Neighborhood
bell3a	27.45	43.92	>351353	>612769	
bell5	>174782	>187457	≫174782	≫187457	
egout	142.35	148.12	>240030	>244815	
enigma	>296625	>297595	≫296625	≫297595	
flugpl	>47649	>78854	≫47649	≫78854	
gt2	14450	9538	>506547	>539824	
lseu	4587.68	3410.4	>208294	>213632	
mod008	3.33	3.24	692264	>681598	
modglob	1	1	>442000	>444842	N^1
noswot	79.26	37.14	>439764	>342003	
p0033	>76324	>73073	≫76324	≫73073	
pk1	1	1	>115125	>112925	
pp08a	30.01	37.02	>275048	>287272	
pp08acut	21.87	27.57	260959	>270440	
rgn	12.02	4.94	2774	3337	
stein27	12.89	18.85	182.34	201.43	
stein45	24.61	35.39	>121756	>134237	
vpm1	9.61	9.29	1292.02	931.99	
bell3a	27.77	39.47	10630	33637	
bell5	>226731	>213884	≫226731	≫213884	
egout	143.17	150.76	490209	>499191	
enigma	>729342	>698885	≫729342	≫698885	
flugpl	>24857	>33071	≫24857	≫33071	
gt2	8989	6948	>5594115	>5563503	
lseu	669.22	681	439230	>850069	
mod008	3.6	2.91	922149	1008139	
modglob	1	1	12724	12564	N^2
noswot	163.01	36.32	>876859	>778050	
p0033	3439.42	3250.79	>105066	100819	
pk1	1	1	>346686	>334838	
pp08a	30.39	37.02	>424686	>431121	
pp08acut	20.23	29.25	>394332	219656	
rgn	12.95	3.82	2275	2415	
stein27	13.21	19.22	1109	1027	
stein45	24.86	34.79	>111285	>112306	
vpm1	9.51	8.18	57224	53631	

Table 1.7. Comparison between probabilistic rounding and bi-triangular construction: expected number of objective evaluations for obtaining feasibility and optimality, for neighborhoods N^1 and N^2.

Problem name	Best sol. found	% distance to optimum	$E[n^b]$	$E[n^f]$	$E[n^o]$	Neighborhood
bell3a	880857	0.2763	116312	26.52	>3376050	
bell5	0.31887 (inf.)	-	165.10	>1744874	>1744874	
egout	615.719	8.382	2391640	141.85	>2391720	
enigma	4 (inf.)	-	74585.23	>2988675	>2988675	
flugpl	0.6 (inf.)	-	43411.73	>477819	>477819	
gt2	49556	134.1	5068183	14034	>5068412	
lseu	1149	2.589	2110868	5119	>2110972	
mod008	307	0	2346518	3.42	2346518	
modglob	20755100	0.07036	4436169	1.00	>4436268	N^1
noswot	-41	4.651	3249.46	81.61	>4372469	
p0033	3095	0.1942	764869	764869	>764916	
pk1	19	72.73	381708	1.00	>1145420	
pp08a	7380	0.4082	925905	30.04	>277800	
pp08acut	7350	0	435974	21.46	435974	
rgn	82.1999	0	2784.93	12.33	2784.93	
stein27	18	0	184.03	12.93	184.03	
stein45	30	0	93568	24.31	93568	
vpm1	20	0	1203	9.90	1202.59	
bell3a	878430.32	0	11216	27.14	11216	
bell5	9030230	0.7118	2601738	2256575	>2604076	
egout	568.101	0	977203	141.48	977203	
enigma	4 (inf.)	-	671243	>7439411	>7439411	
flugpl	0.7 (inf.)	-	6861	>248127	>248127	
gt2	36131	70.70	55164302	9194	>55183799	
lseu	1120	0	1245115	693.27	1245115	
mod008	307	0	871031	3.42	871031	
modglob	20740508	0	12381	1.00	12381	N^2
noswot	-41	4.651	24533	140.36	>8714283	
p0033	3089	0	1066553	3322	1066553	
pk1	16	45.45	435039	1.00	>3492631	
pp08a	7350	0	2161662	30.16	2161662	
pp08acut	7350	0	4173685	21.18	4173685	
rgn	82.1999	0	2469	12.09	2469	
stein27	18	0	1130	12.91	1130	
stein45	30	0	1110292	23.97	1110292	
vpm1	20	0	64719	9.97	64719	

Table 1.8. Best solution found, percent distance above optimum, and expected number of LP solutions for reaching the best solution, feasibility and optimality. Results obtained with 10000 iterations for N^1 and 1000 iterations for N^2.

Problem	Neighborhood N^1			Neighborhood N^2		
name	$E[t^b]$	$E[t^f]$	$E[t^o]$	$E[t^b]$	$E[t^f]$	$E[t^o]$
bell3a	299.3	0.08	>8687	34.52	0.08	34.52
bell5	0.3	>3197	>3197	4880	4289	>4884
egout	5547	0.37	>5547	2394	0.38	2394
enigma	7.23	>289.7	>289.7	85	>935.3	>935.3
flugpl	12.82	>141.1	>141.1	2.12	>76.69	>76.69
gt2	392.2	1.09	>392.2	4297	0.5517	>4298
lseu	95.99	0.23	>96	90.72	0.0328	90.72
mod008	552.4	0.001	552.4	298.9	0.0013	298.9
modglob	57039	0.02	>57040	146.7	0.0133	146.7
noswot	3.13	0.10	>4198	28.83	0.1788	>10244
p0033	17.41	17.41	>17.41	25.66	0.074	25.66
pk1	152.9	0.0004	>458.8	181.9	0.0005	>1460
pp08a	5209	0.31	>15627	12571	0.3244	12571
pp08acut	3762	0.4	3762	37617	0.41	37617
rgn	2.93	0.01	2.93	2.77	0.0134	2.77
stein27	0.01	0.0008	0.01	0.08	0.0008	0.077
stein45	19.61	0.01	19.61	248.9	0.0053	248.9
vpm1	6.9	0.1	6.88	415.9	0.1007	415.9

Table 1.9. Comparison of expected CPU time (in seconds) required using neighborhoods N^1 and N^2, in order to obtain the best solution, feasibility and optimality, using probabilistic rounding construction.

CPU time, required to obtain them. Therefore, for comparison with other methods, we decided to use the results obtained with a GRASP implementation using probabilistic rounding construction, and the N^2 neighborhood.

6.3.4 Comparison of GRASP with other methods. For comparing GRASP to other methods, the criteria used are the best solution found and its distance to the optimum, the actual or expected number of LP solutions required, and the actual or expected CPU time used. A comparison between GRASP and B&B is presented in Table 1.10. Table 1.11 reports a comparison between GRASP and an evolutionary solver.

The comparison with B&B indicates that each algorithm works well on rather different instances: GRASP determines a good feasible solution in all the cases where B&B failed, and B&B quickly determines the optimal solution for the two instances where GRASP could not find any feasible solution (*enigma* and *flugpl*). The expected number of LP solutions and the expected CPU time is many times smaller for GRASP than the number of LPs and CPU time required by B&B. For larger problems, like *modglob* and *vpm1* GRASP seems to have some advantage.

The results obtained with the evolutionary solver (ES) were reported in (Pedroso 1998). The comparison between GRASP and this metaheuristic is made in terms of the percent distance to optimum, and the expected number of LP solutions for reaching feasibility and optimal-

Problem	GRASP	B&B	GRASP	B&B	GRASP	B&B
name	% to opt.	% to opt.	$E[n^b]$	# LPs	$E[t^b]$	CPU time
bell3a	0	0	11216	438587	34.52	170.56
bell5	0.7118	0	2601738	420499	4880	81.61
egout	0	1.03	977203	55057	2394	6.06
enigma	-	0	671243	8947	85	1.8
flugpl	-	0	6861	1588	2.12	0.06
gt2	70.70	-	55164302	-	4297	-
lseu	0	0	1245115	236207	90.72	23.97
mod008	0	0	871031	2848585	298.9	844.1
modglob	0	26.85	12381	1.00E+08	146.7	47678
noswot	4.651	40.48	24533	3753	28.83	3.72
p0033	0	0	1066553	7393	25.66	0.25
pk1	45.45	0	435039	3710343	181.9	2467
pp08a	0	32.93	2161662	1.00E+08	12571	29924
pp08acut	0	10.34	4173685	1.00E+08	37617	161880
rgn	0	0	2469	4963	2.77	1.2
stein27	0	0	1130	11985	0.08	2.62
stein45	0	0	1110292	236453	248.9	218.6
vpm1	0	10	64719	18354	415.9	24.95

Table 1.10. Comparison between GRASP and B&B: percent distance above optimum, expected number of LP solutions and CPU time for GRASP to obtain its best solution, and number of LPs and CPU required by B&B.

Problem	% dist. to optimum		$E[n^f]$		$E[n^o]$	
name	GRASP	ES	GRASP	ES	GRASP	ES
bell3a	0	0.3990	27.14	2053	11216	>18246645
bell5	0.712	0.7143	2256575	33738	>2604076	>18024642
egout	0	0	141.48	423	977203	133764
enigma	-	-	>7439411	>11876637	≫7439411	≫11876637
flugpl	-	0	>248127	29048	≫248127	91004
gt2	70.70	5.556	9194	6383	>55183799	>37665907
lseu	0	0	693.27	1985	1245115	10269416
mod008	0	0	3.4	17	871031	2557585
modglob	0	0	1	3	12381	99478
noswot	4.651	4.651	140.36	33627	>8714283	>34335094
p0033	0	0	332	8350	1066553	93571
pk1	45.46	72.72	1	3	>3492631	>6259152
pp08a	0	0	30.16	49	2161662	177969
pp08acut	0	0	21.18	33	4173685	45582
rgn	0	0	12.09	21	2469	8050
stein27	0	0	12.91	41	1130	286
stein45	0	0	23.97	61	1110292	54791
vpm1	0	0	9.97	123	64719	7397

Table 1.11. Comparison between GRASP and an evolutionary solver: percent distance to optimum, and expected number of LP solutions for reaching feasibility and optimality.

ity. The results are reported in Table 1.11. Notice that the termination criteria for GRASP and for the ES are very different, and hence the comparison in terms of distance to optimality is not very meaningful. Still, it shows that the two meta-heuristics have difficulties on roughly the same instances. This is not surprising, as the ES has an improvement routine based on a neighborhood similar to N^1. On the other hand, the expected times required for obtaining feasibility and optimality can be dramatically different, indicating that population-based routines and recombination are a good complement to moves within the neighborhoods N^1 and N^2. Instances where the ES is much slower than GRASP are probably due to the absence of improvements based on N^2 on that solver, or to the lack of diversity generated by construction on GRASP. (Comparisons based on CPU time were not made, as the ES results were obtained on very different machines.)

Comparing GRASP to the state-of-the-art commercial solver *Xpress-MP Optimizer, Release 13.02* indicated a clear advantage of that solver, which in most cases could find an optimal solution one to four orders of magnitude faster. Still, this solver had problems on some instances: *bell5* and *noswot* could not be solved in 24 hours of CPU time. For some other instances (*bell3a, pk1, stein27*), Xpress-MP required more LP solutions than GRASP.

7. Conclusion

In this paper we present a GRASP for the solution of integer linear problems. The algorithm starts by reading an *MPS* file with the instance data. When the problem is a MIP, the integer variables are fixed by GRASP and replaced in the original problem, leading to a pure continuous problem. This problem can be solved by a liner program solver, to evaluate the corresponding fixed variables. When the original problem is an IP, simple algebraic manipulations can be used to evaluate the fixed variables.

The algorithm works with feasible and infeasible solutions. If the solution is feasible, its evaluation is determined directly by the objective function. If the solution is infeasible, the evaluation is given by the sum of constraint violations, which is determined by solving an LP (for MIP problems) or by simple algebraic manipulations (for IP problems).

The results obtained with GRASP for some benchmark problems were compared to those obtained by B&B and to those obtained by an evolutionary solver. The comparison with B&B shows that GRASP has a very interesting behavior, as it determines good feasible solutions in the cases where B&B fails. In the comparison with the evolutionary solver, we could verify that the population-based methods used there could lead many times to substantial reductions on the CPU time required to obtain a given solution. On other cases, substantial CPU time reductions are on the side of GRASP; therefore, no clear conclusion about which

of the meta-heuristics is better could be drawn. GRASP being simpler, it might be the appropriate choice if implementation burden is to be avoided.

Notes

1. The GRASP implementation works for minimization and maximization, by adapting the meaning of *is better* (see section 4.2.4).

2. This software has the advantage of being free; on the other hand, it does not have some important components, like the dual simplex method, which would allow to quickly reoptimize Equation 1.3 from a dual solution after a change in the right hand side.

3. Let f^b be the objective value for the best feasible solution, and f^o for the optimal solution. The percent distance above the optimum is given by $\left|100 \times (f^b - f^o)/f^o\right|$.

Bibliography

Michel R. Berkelaar and Jeroen Dirks. lp_solve - a solver for linear programming problems with a callable subroutine library. Internet repository, version 2.2. ftp://ftp.es.ele.tue.nl/pub/lp_solve.

S. Binato, W. J. Henry, D. Loewenstern, and M. G. C. Resende. A greedy randomized adaptive search procedure for job scheduling. In C. C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*. Kluwer Academic Publishers, 2001.

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.

C. Carreto and B. Baker. A GRASP interactive approach to the vehicle routing problem with backhauls. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, 2001.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computacionally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.

T. A. Feo, K. Venkatraman, and J. F. Bard. A GRASP for a difficult single machine sheduling problem. *Computers & Operations Research*, 18:635–643, 1991.

P. Festa and M. G. C. Resende. GRASP: an annotated bibliography. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, 2001.

Pierre Hansen and Nenad Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

G. Kontoravdis and J. F. Bard. A GRASP for the vehicle routing problem with time windows. *ORSA J. on Computing*, 7:10–23, 1995.

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

Y. Li, P. M. Pardalos, and M. G. C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience in Discrete Mathematics and Optimization, 1988.

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

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

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 168–183. Oxford University Press, 2002.

M. G. C. Resende and T. A. Feo. A GRASP for satisfiability. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring and Satisfiability: The second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.

M. G. C. Resende, L. S. Pitsoulis, and P. M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In *Satisfiability problems*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 393–405. American Mathematical Society, 1997.

M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedure. In Fred Glover and G. Kochenberger, editors, *State of the Art Handbook in Metaheuristics*. Kluwer Academic Publishers, 2001.

A.J. Robertson. A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment

problem. *Computational Optimization and Applications*, 19:145–164, 2001.

Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience in Discrete Mathematics and Optimization, 1998.