Meta-heuristics for combinatorial optimisation

João Pedro Pedroso
Centro de Investigação Operacional
Faculdade de Ciências da Universidade de Lisboa
and
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
R. Campo Alegre 823, 4150-180 Porto, Portugal
`jpp@ncc.up.pt`

31 October 2001

1 Introduction

The use of meta-heuristics for solving combinatorial optimisation has now a long history, and there are virtually no well-known, hard optimisation problems for which a meta-heuristic has not been applied. Often, meta-heuristics obtain the best known solutions for hard, large-size real problems, for which exact methods are too time consuming to be applied in in practice.

The pioneering, and landmark work on general considerations about meta-heuristics for combinatorial optimisation which, besides its well known aspects, pointed out many directions which were not exploited at least until very recently, is (Glover, 1989; Glover, 1990). A recent survey of many directions that meta-heuristics are taking, and many problems to which they are being applied is available in (Aarts and Lenstra, 1997). There are many successful applications reported in the literature; still, to the best of our knowledge there are no mathematical programming languages with an interface to solvers based on meta-heuristics. Mathematical programming being the traditional, powerful way of modelling combinatorial problems, we believe that it would be an important achievement to have a solver based on meta-heuristics callable from mathematical programming systems.

In this paper we will focus on the different aspects to take into account when designing a meta-heuristic which can be applied to any combinatorial optimisation problem modelled in mathematical programming.

2 Background

A general formulation of a combinatorial optimisation problem may have continuous and discrete variables, and non-linear objective and constraints. It is called a non-linear mixed integer problem (NLMIP), and can be formulated as follows:

$$\max_{x,y}\{f(x,y) : g(x,y) \leq 0, h(x,y) = 0, x \in [l,u]^n \subset \mathbb{Z}^n_+, y \in \mathbb{R}^p_+\} \tag{1}$$

In this formulation, $x = (x_1, \ldots, x_n)$ are the integer variables, $y = (y_1, \ldots, y_p)$ the continuous variables, $f(x,y)$ is the objective function, $g(x,y) = (g_1(x,y), \ldots, g_m(x,y))$ are inequality constraints and $h(x,y) = (h_1(x,y), \ldots, h_o(x,y))$ are equality constraints. We will assume that lower and upper bounds are known for each discrete variable, $l_i \leq x_i \leq u_i$. $\mathbb{Z}^n_+$ is the set of nonnegative integral n-dimensional vectors and $\mathbb{R}^p_+$ is the set of nonnegative p-dimensional vectors.

This general problem has some important special cases, which are the object of the remaining of this section.

2.1 Pure integer non-linear problems

When all the variables are integer, the problem is pure integer non-linear (NLIP):

$$\max_{x}\{f(x) : g(x) \leq 0, h(x) = 0, x \in [l,u]^n \subset \mathbb{Z}^n_+\} \tag{2}$$

2.2 Pure integer linear problems

If both the objective and the constraints are linear, and all the variables are integer, we have a pure integer linear program (IP), whose formulation is:

$$\max_{x}\{cx : Ax \leq b, x \in [l,u]^n \subset \mathbb{Z}^n_+\} \tag{3}$$

A is an $m \times n$ matrix, and m is the total number of constraints.

2.3 Mixed integer linear problems

If the objective and the constraints are all linear, and there are integer and continuous variables, the problem is called *mixed integer* (MIP). The formulation of a mixed integer linear program is

$$\max_{x,y}\{cx + dy : Ax + Gy \leq b, x \in \mathbb{Z}^n_+, y \in \mathbb{R}^p_+\} \tag{4}$$

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.

Extensive information on problems of this type, which also includes some heuristics for specific problems, is presented in (Nemhauser and Wolsey, 1988).

2.4 Mixed integer problems with linear continuous variables

In this case we have non-linearities arising only on the discrete variables. Its formulation is:

$$\max_{x,y}\{f(x) + dy : g(x) + Gy \leq 0, h(x) + Hy = 0, x \in [l,u]^n \subset \mathbb{Z}^n_+, y \in \mathbb{R}^p_+\} \tag{5}$$

G and H are $m \times p$ and $o \times p$ matrices, respectively, and d is a vector of size o.

3 Solving strategies

3.1 Representation of the solutions

The strategy that we point out for solving the problem (1) consists of fixing the integer variables using a meta-heuristic.

When the problem is pure integer, the objective function (or an infeasibility measure) can be evaluated immediately. Let us call the variables fixed $\bar{x}$, and let $s_k = g_k(\bar{x}, y)$ if $g_k(\bar{x}, y) > 0$ and 0 otherwise, for $k = 1, \ldots, m$. If all $g_k(x) = 0$ and $h_l(x) = 0$, $\bar{x}$ is feasible, and the corresponding objective is $f(\bar{x})$. Otherwise, the choice of $\bar{x}$ is infeasible, and a measure of the infeasibility is given by

$$\zeta = \sum_{k=1}^{m} s_k + \sum_{l=1}^{o} h(\bar{x}) \tag{6}$$

An application of a meta-heuristic for the specific linear, pure IP problem has been presented in (Pedroso, 2001a).

When the problem has continuous variables, we will solve the continuous (linear or not) problem that results from fixing the integer ones, and using the objective corresponding to this problem for evaluating the solution. The continuous variables (if some) are therefore determined as a function of the integer ones, after these have been fixed. When the problem is pure integer, there is no need to solve the second level, continuous problem.

The solution of an NLMIP with the set of continuous variables not empty is, therefore, done in two stages; on the first stage we fix the integer variables, and on the second stage we determine the value of the continuous variables that correspond to the choice made at the first stage.

3.2 Solution of the second stage problem

3.2.1 The general case

The second stage problem is itself composed of two goals: the first goal is to minimise infeasibilities, and if a feasible solution is found, we then try to optimise the (original) objective. The minimisation of infeasibilities in the second stage problem corresponds to

$$\zeta = \min_{s,y}\{ \quad \sum_{k=1}^{m} s_k + \sum_{l=1}^{o} |h(\bar{x}, y)| : \tag{7}$$

$$\begin{aligned}&g_k(\bar{x}, y) \leq s_k, \quad k = 1 \ldots, m\\ &y \in \mathbb{R}^p_+ \ , \ s \in \mathbb{R}^m_+\}\end{aligned}$$

This problem can by itself a very difficult one: it can be a non-convex, non-smooth problem, and there is no easy answer for the selection of a strategy for solving it. As a first approach, one may think of the simplex method for non-linear programming (Nelder and Mead, 1965), which is very elegant and easy to implement. It is, in its original form, a local search method, but there are recent extensions that enable its use in global optimisation (Glover et al., 2000) and to constrained global optimisation (Pedroso, 2001b). It does not use information, and does not make assumptions, on the derivatives (which could potentially pose some problems in a general NLMIP), and requires very few parameters for operation, making it a good candidate for the solution of the second stage problem if it is non-linear.

3.2.2 Mixed integer problems with linear continuous variables

In this case, which corresponds to equations (4) and (5), by fixing all the integer variables of the MIP at the values chosen in the first stage we obtain a linear problem:

$$z = \max_y \{f(\bar{x}) + hy : Gy \leq f(\bar{x}), Hy = h(\bar{x}), y \in \mathbb{R}^p_+\} \tag{8}$$

This (purely continuous) linear problem can be solved using any standard algorithm, like the simplex. For detailed information about this algorithm, and the considerations to take into account for an efficient implementation for this particular case, the reader is referred to (Chvatal, 1980).

If problem 8 is impossible, we formulate another linear program for the minimisation of the infeasibilities. This is accomplished by setting up artificial variables and minimising their sum (a procedure that is identical to the phase I of the simplex algorithm):

$$\begin{aligned}\zeta = \min_{s,t,y}\{ \quad &\sum_{k=1}^{m} s_k + \sum_{l=1}^{o} t_l : && (9)\\ &f(\bar{x}) + Gy \leq s &&\\ &h(\bar{x}) + Hy \leq t && (10)\\ &-h(\bar{x}) - Hy \leq t && (11)\\ &y \in \mathbb{R}^p_+ \ , \ s \in \mathbb{R}^m_+ \ , \ t \in \mathbb{R}^o_+\} && (12)\end{aligned}$$

An application of a meta-heuristic for the specific linear, MIP problem has been presented in (Pedroso, 1998).

3.3 Evaluation and comparison of solutions

The evaluation attributed to a solution is composed of a binary label indicating if it is feasible or not, and a real value which is the original objective (z) if the solution is feasible, or the sum of infeasibilities (ζ) if it is not.

We need a way of comparing solutions, independently of them being feasible or not. What we propose is to classify solutions in such a way that feasible solutions are always better than infeasible ones, and are ranked among them according to the objective z; infeasible solutions are ranked among them according to the sum of infeasibilities ζ.

We say that $\bar{z}^i \succ \bar{z}^j$ (meaning that the structure i *is better* than the structure j) iff:

- $\zeta^i < \zeta^j$ (i is closer to the feasible region than j).
- $\zeta^i = \zeta^j = 0$, and $z^i > z^j$ (i has a better objective);

4 Tools for implementing meta-heuristics

4.1 Construction

Construction concerns the choice of the initial values of the integer variables of 1. As seen before, the continuous variables (if some) are computed on the basis of this choice.

4.1.1 Random construction

This construction has the advantage of being a powerful way of obtaining diversity; but in general will lead to solutions of very poor quality.

It consists of assigning each of the variables x_i a random integer value $r_i \in [l_i, u_i]$ drawn with uniform distribution. The remaining, continuous variables (if some), are computed on a second stage, as explained in 3.2.

An interesting property of this construction is that it by itself has asymptotic convergence to the global optimum, if it is repeated an infinite number of times.

4.1.2 Construction based on the continuous relaxation

The main idea for these construction methods has been presented for linear integer programs in (Lengauer, 1990) on the static form. There, solutions of the problem were attempted by solving their continuous relaxation, and rounding each variable to one of the integers closest to the solution of the relaxation, with probabilities proportional to the closeness of the continuous value to an integer.

On (Nemhauser and Wolsey, 1988) a (non-probabilistic) greedy construction, on the dynamic form, is presented for linear mixed integer problems. At each step, a variable that has a fractional value on the solution of the linear relaxation is fixed to its closest integer, and the linear relaxation is resolved, until all variables are integer.

Static construction On this construction, the continuous relaxation of the problem 1 is solved, and at each step a variable is fixed to an integer value close to the (in general continuous) value obtained with the relaxation. Depending on the distribution used for drawing the random integer, the best solutions obtained with this construction may have or not asymptotic convergence to the optimal value. Pseudo programming code for this construction is presented in algorithm 1.

Algorithm 1: Semi-greedy solution construction: static version.

```
StaticSemiGreedy()
(1)   solve the continuous relaxation
(2)   for $k = 1$ to $n$
(3)       draw random integer $r$ with some distribution
(4)       $\bar{x}_k := r$
(5)   return $\bar{x}$
```

Dynamic construction In this case, the continuous relaxation of the problem (1) is solved as before, and at each step a (randomly selected) variable is fixed to an integer value close to its value at the relaxation's solution. The difference with respect to the static construction is that, in the dynamic version, a new relaxation is solved after each variable is fixed. Pseudo programming code for this construction is presented in algorithm 2.

4.1.3 Distributions for random number generation

Probabilistic rounding construction. This construction consists of rounding each variable to one of the integers closest to its value at the continuous relaxation. The probabilities for rounding

Algorithm 2: Semi-greedy solution construction: dynamic version.

```
DynamicSemiGreedy()
(1)    $C := \{1, \dots, n\}$
(2)    while $C \neq \{\}$
(3)        solve relaxation with all variables $x_i$, $i \in C$ relaxed
(4)        randomly select index $k$ from $C$
(5)        draw random integer $r$ with some distribution
(6)        $\bar{x}_k := r$
(7)        $C := C \backslash \{k\}$
(8)    return $\bar{x}$
```

up or down each of the variables are given by the distance from the fractional solution, x_k^R, to its closest integers. For an index $k \in \{1, \dots, n\}$, the random value r is drawn with the following probabilities:

$$P(r = \lfloor x_k^R \rfloor) = 1 - (x_k^R - \lfloor x_k^R \rfloor)$$
$$P(r = \lceil x_k^R \rceil) = (x_k^R - \lfloor x_k^R \rfloor)$$

These distributions might generate solutions which are too similar to the solution of the relaxation; in particular, is a variable is integer in the relaxation, it will only be assigned that integer value in all the constructed solutions. In general, construction with this distribution will not have asymptotic convergence to the optimum, for an infinite number of tentatives.

Bi-triangular construction. The goal of this strategy is to increase the diversification, while keeping a good quality of the solutions, on average. We now extended the possibility of rounding each variable x_k to any integer between its lower-bound, l_k, and its upper-bound, u_k, with a distribution that has a mean of x_k^R.

A density function that can be used is the bi-triangular distribution, composed by two triangles, which is defined by three parameters: the minimum a that is equal to $l_k - 0.5$, the maximum b that is equal to $u_k + 0.5$, and the mean c, equal to the relaxation value. The probability density function is given by:

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

We draw a random number x using this distribution, and let r be its closest integer. The bi-triangular density function is represented in figure 1. This distribution spans the whole domain of each of the variables, and hence has asymptotic convergence to the optimum for an infinite number of solutions generated.

4.2 Neighbourhoods

Meta-heuristics based on local search try, at some point, to improve the quality of a solution by hill climbing on its neighbourhood. For this purpose we propose neighbourhoods that consist of incrementing or decrementing variables, independently or simultaneously.

The variables that are considered on these neighbourhoods are the integer one, as the continuous, if some, are determined in function of them in a second stage problem.

4.2.1 Increment neighbourhood

The *increment* neighbourhood of a solution x, $N^1(x)$, is composed of solutions which differ from x in one element x_j, whose value is one unit above or below x_j. Hence y is a neighbour 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 $j \neq i$.

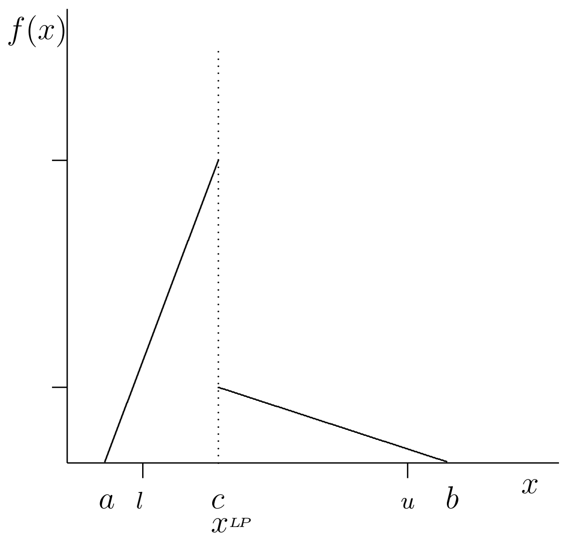

Figure 1: Bi-triangular density function.

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

The idea used on this neighbourhood can be extended to the case where we change more than one variable at the same time. For example, the *2-increment* neighbourhood 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 neighbour 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* neighbourhood 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 neighbours of a solution increases exponentially. We may use a strategy for reducing the size of the set of neighbours that is explored. Let O be the set of indices of variables which appear in the objective function. For a feasible solution x, one may search only the subset of $N^k(x)$, where at least one variable whose index is in O is changed. Identically, for an infeasible solution x, let V be the set of indices of variables which appear in the constraints that were violated. One may search only the subset of $N^k(x)$ where at least one variable has an index in $V(x)$. If either O or $V(x)$ are empty, then the important integer variables only indirectly affect the objective or violated constraints, and this heuristic will not work. In this case, one should search the whole neighbourhood $N^k(x)$.

If one can afford spending a large time in the search, one might consider a superset of the *k-increment* neighbourhood:

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

A version of the N^1 neighbourhood was proposed in (Resende and Feo, 1996) for satisfiability problems, where there are only binary variables. The *1-flip* neighbourhood proposed there consisted of negating the value of the variable in one index, and keeping all the other variables unchanged.

4.2.2 Hunt search

Hunt search was originally conceived for locating values in an ordered table; search started with a unit step, and the step was doubled at each iteration, until having crossed the wished value. When applied to local search, the idea is to check for improvements in the objective (or infeasibility) when one of the n integer variables is independently perturbed, with a geometrically increasing step. The steps are $1, 2, 4, 8, \ldots$; therefore, for an index k values searched are going to be $\bar{x}_k + 1, \bar{x}_k + 3, \bar{x}_k + 7, \ldots$, or its symmetric $\bar{x}_k - 1, \bar{x}_k - 3, \bar{x}_k - 7, \ldots$.

Hunt search stops when there are no more improvements on the objective, or when the bound of the variable has been crossed.

This method can lead to dramatic reductions on the time consumed when one is making improvements in a solution where the upper and lower bounds are far apart, and the initial solution is not close to a local optimum. It should, therefore, be a complement of the implementation of the neighbourhood search.

4.3 Meta-heuristics

Many meta-heuristics which are based on local search can be used for combining a construction and a local search of the neighbourhoods defined above.

A basic tool that should be provided is, of course, simple local search.

Iterated local search, or a variant of GRASP, might be the next step. Using this method with an appropriate construction will combine fast generation of good solutions with asymptotic convergence to the optimum, and might therefore please many of the users.

Tabu search seems to be a good way of pursuing, as it will allow fast generation of good solutions, and possibly a better quality of the solutions, on average, than the preceding methods. One might combine it with a restart procedure when stagnation is observed, especially if theoretical convergence to the global optimum is required.

Simulated annealing and evolutionary algorithms are also possible ways of enlarging the meta-heuristics stack; but on these cases, parameterisation might pose some problems, as the amount of information that is required from the user might be a burden that one cannot afford in a general tool.

5 Interface with mathematical programming tools

For linear problems, the most commonly used format for communicating a problem to a solver is the MPS format. This is a rather old and limited format, but is still the de facto standard. All major mathematical programming systems have the possibility of generating MPS files from the (linear) model and data. Meta-heuristics for these cases were already implemented, with interfaces using this protocol.

For non-linear problems, there is no such wide-spread standard as the MPS, and most likely the interface will have to be implemented in a case-by-case basis. An interface to the AMPL language (Fourer et al., 1993) is currently under way.

For the exact solution of a MIP, the solver might be called with no information other than the model itself. On the other hand, in order to have control over the meta-heuristic, some information concerning the solution strategy has to be passed; at a bare minimum, the stopping criterion should be stated, using default values for all the other settings. Useful information that can be transmitted when solving a problem with a meta-heuristic includes:

Stopping criterion: most commonly, the maximum time/iterations allowed.

Construction: the type of construction to be employed. At least two options should be available: random and greedy construction.

Neighbourhood: the type of neighbourhood to be used in the search.

Method: the meta-heuristic method to employ. Possibly, simple local search should be the default value.

One interesting feature that a mathematical programming system might allow to explore concerns the integration of the meta-heuristic with exact approaches. One might be interested in using a solution provided by the meta-heuristic for eliminating subtrees of a branch-and-bound process, and similarly make the meta-heuristic search from a partial solution found in the branch-and-bound; particularly, if the branch-and-bound has to be interrupted, one might want to make sure that the solution is a local optimum before using it in some practical application.

6 Conclusion

In this paper we have provided the basis for implementing meta-heuristics for a general non-linear, mixed integer program, provided it has a model in mathematical programming.

The main idea underlying the proposed methods is the possibility of enlarging the use meta-heuristics to the community of operations researchers which are used to specifying models in mathematical programming.

Though implementation of these ideas has been done for some specific cases, most of the situations are not is still not handled. Its implementation is required for assessing the quality and efficiency of the methods in practice.

References

Aarts, E. and Lenstra, J. K., editors (1997). *Local Search in Combinatorial Optimization.* John Wiley & Sons.

Chvatal, V. (1980). *Linear Programming.* Freeman.

Fourer, R., Gay, D. M., and Kernighan, B. W. (1993). *AMPL — A Modeling Language for Mathematical Programming.* The Scientific Press.

Glover, F. (1989). Tabu search—part I. *ORSA Journal on Computing*, 1:190–206.

Glover, F. (1990). Tabu search—part II. *ORSA Journal on Computing*, 2:4–32.

Glover, F., Laguna, M., and Martí, R. (2000). Scatter search. Technical report, Graduate School of Business and Administration, University of Colorado.

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

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7:308–313.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization.* Wiley-Interscience in Discrete Mathematics and Optimization. John Wiley & Sons, New York.

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

Pedroso, J. P. (2001a). An evolutionary solver for pure integer linear programming. *International Transactions on Operations Research.* Accepted for publication.

Pedroso, J. P. (2001b). Meta-heuristics using the simplex algorithm for nonlinear programming. In of Nonlinear Theory, R. S. and its Applications, editors, *Proceedings of the 2001 International Symposium on Nonlinear Theory and its Applications*, pages 315–318, Miyagi, Japan.

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