Matheuristics: embedding MILP solvers into heuristic algorithms for combinatorial optimization problems

Federico Della Croce[*] *Andrea Grosso*[†] *Fabio Salassa*[‡]

Abstract

Combinatorial Optimization Problems (COPs) have always been studied because of their relevant practical importance. Solution techniques for solving COPs have traditionally been split into exact (mostly based on the optimal solution of the integer programming formulation of the real problem) and heuristic algorithms. Recently the hybridization of these two approaches has become very popular. So called Matheuristics, which rely on the idea of exploiting the best of the two, have raised much attention. In this chapter, we discuss a general matheuristic framework embedding general MILP solvers into heuristic approaches to solve COPs. Different applications are presented related to different kind of problems. For all applications, the use of MILP solvers within the heuristic framework is described.

1 Introduction

Many problems arising in different areas such as in production or distribution of goods and services are combinatorial optimization problems (COPs). There are many examples including shop sequencing and scheduling within production management, timetabling within human capital management, lot sizing for distribution problems and portfolio optimization in the financial management field. These problems are interesting because of their relevant practical importance but are also well known to be difficult to solve. Most of them are, in fact, $NP-hard$ problems [14]. This difficulty and, at the same time, the fact that they are real and important problems, have led to a large number of solution techniques for COPs.

Thus when facing a COP there are a few basic alternatives among which we can choose for solving it. If we can exploit some structure of the problem and derive a polynomial algorithm, the problem is easily solvable. If we are

[*]federico.dellacroce@polito.it, D.A.I. Politecnico di Torino, Italy
[†]grosso@di.unito.it, D.I. Università degli Studi di Torino, Italy
[‡]fabio.salassa@polito.it, D.A.I. Politecnico di Torino, Italy

facing a NP-hard problem, we have different alternatives: (i) we can seek the optimum or (ii) we can approximate the optimum.

Seeking the optimum for an NP-hard problem usually implies using *implicit enumeration* techniques. Well– known methods are *branch-and-bound*, *branch-and-cut*, *branch-and-price* and *dynamic programming*, see, e.g., [22]. They aim to explore the entire solution space, avoiding complete enumeration, by successive splitting of the solution space in order to identify non-optimal families of solutions.

In order to avoid complete (in the worst case) enumeration of the solution space we can use a polynomial-time algorithm that cannot give a provably optimal solution, but a reasonably good, *approximate* solution (that is, not too far from the optimum). Examples of well-known general-purpose *meta-heuristics* algorithms are *simulated annealing*, *tabu search*, *greedy algorithms*, *variable neighborhood search*, *iterated local search* and *evolutionary algorithms*, see [4] for an overview.

Combination (hybridization) of techniques is a popular and well established practice in the metaheuristic field. The popularity, success and importance of that specific line of research is well documented by the large number of successful reported applications (see, for instance, [19],[9] and[23]).

Recently, the hybridization of exact methods and metaheuristics has led to so-called **Matheuristics**. Matheuristics exploit the strength of both metaheuristic algorithms and exact methods within a "hybrid" approach. A distinguishing feature is the exploitation of nontrivial mathematical programming tools as part of the solution process. However, there is neither a unique classification nor a consolidated working framework in the field.

The aim of this chapter is to provide insight into the recent development of Matheuristics when embedding general MILP solvers into heuristic approaches to solve COPs. First, we introduce *Local Branching* which can be viewed as a local search approach exploring a neighborhood of a current solution delimited by a pre-defined Hamming distance. Second, we introduce *Variable Partitioning Local Search* which can be seen as another local search approach based on a partial re-optimization of a variables partitioning. Both approaches can be considered as generalized $k-opt$ neigborhood search procedures. Third, we propose matheuristic solution methods strongly relying on the continuous relaxation of the MILP formulations of the problems considered. We point out that, typically, such methods perform well whenever the continuous relaxation solution of the corresponding problem can give significant insight into the related integer solution. For each of the three classes of solution procedures, illustrative successful applications are described. Finally, we provide an application where a constructive (greedy) approach embedding iterative solutions of MILP models is proposed. The purpose of this last section is to show that, for large size problems where general local search approaches may be too time consuming, even greedy approaches can benefit from the matheuristic framework.

2 Local Branching

The *Local Branching* [10] approach is introduced with the aim of integrating local search and metaheuristic ideas within Mixed Integer Programming. In Local Branching, general MIPs with 0-1 x_j variables are considered. The idea is to iteratively solve a local subproblem corresponding to a classical k-OPT neighborhood, using the MIP-solver that searches for the best solution among those having a Hamming distance not greater than k. In Local Branching, the master process is an exact method, such as Branch and Bound, while the slave algorithm is an exact neighborhood search performed by a MIP solver exploiting neighborhoods generated with the Hamming distance. This is achieved by introducing a local branching constraint based on an incumbent solution $\bar{x}$ where $x_j = \bar{x_j} \ \ \forall j$, which partitions the search space into the k-OPT neighborhood and the rest of the solutions space. Then, the k-OPT neighborhood constraint becomes

$$\sum_{j:\bar{x_j}=0} x_j + \sum_{j:\bar{x_j}=1} (1 - x_j) \leq k,$$

while the rest of the solution space is obtained by adding the constraint

$$\sum_{j:\bar{x_j}=0} x_j + \sum_{j:\bar{x_j}=1} (1 - x_j) \geq k + 1.$$

The subproblem related to the k-OPT neighborhood constraint is solved, and if an improved solution is found, a new subproblem is generated and solved; this is repeated as long as an improved solution is found. If the process stops, the rest of the problem is solved in a standard way. This basic mechanism is extended by introducing time limits, automatically modifying the neighborhood size k and adding diversification strategies in order to improve its performance. Notice that Local Branching can be applied in its current form to any combinatorial optimization problems that can be modeled as a MIP, because it has no problem dependent structures included in it. Also, it is important to note that the method is exact in nature. The whole search tree can in fact be explored with this approach but introducing time limits can be effectively used as a heuristic procedure. Below, a successful application of local branching to network design is presented.

2.1 A local branching procedure for network design

Fischetti, Polo and Scantamburlo [11] present a sophisticated, quite general variant of local branching to MILPs with so-called 2-level variables, although the application is mostly focused on a specific network design problem. The overall heuristic seems to be somehow inspired by the feasibility pump (see Section 4.1), and is structured in three stages called **D**iversification, **R**efining

and **T**ight refining (DRT). The authors consider MILPs whose set of binary variables $\mathcal{B}$ can be partitioned into two subsets

$$\mathcal{B}_1 \cup \mathcal{B}_2 = \mathcal{B} \tag{1}$$

of first-level and second-level variables, respectively, with the characteristic that "fixing the value of first-level variables produces an easier (but not trivial) problem" [11].

Given a MILP where

$$\begin{aligned} &\text{minimize } z = c^T x \\ &\text{subject to } Ax \leq b \\ &\qquad x_j \in \{0,1\} \qquad x_j \in \mathcal{B} \\ &\qquad x_j \in \mathbb{Z}_+ \qquad\quad x_j \in \mathcal{I}, \end{aligned}$$

the set of binary variables can be partitioned accordingly with (1) and the DRT phases are illustrated as follows. A current solution $\bar{x}$ is assumed to be known. Similarly to local branching, the MILP augmented with additional constraints is solved several times via a MILP solver, used as a black-box. The Refining phase solves the MILP amended with the additional constraint $\sum_{x_j \in \mathcal{B}_1} |x_j - \bar{x}_j| \leq k$, for small values of the parameter k — the authors state $k = 0$ or 2 as typical values. This local branching constraint is linearized to

$$\sum_{x_j \in \mathcal{B}_1 :\, \bar{x}_j = 0} x_j + \sum_{x_j \in \mathcal{B}_1 :\, \bar{x}_j = 1} (1 - x_j) \leq k.$$

This stage aims to use the MILP solver to explore a neighborhood of $\bar{x}$ whose first-stage variables are, according to [11], "almost fixed". The solver's branch and cut is truncated by a time limit. If the time limit expires before optimality is proven, a Tight refining stage is triggered, where one more local branching constraint is added to the MILP:

$$\sum_{x_j \in \mathcal{B}_2, \bar{x}_j = 0} x_j + \sum_{x_j \in \mathcal{B}_2 \bar{x}_j = 1} (1 - x_j) = k'.$$

The solver is then run with different values of k'.

The Diversification stage is enacted by removing the above local branching constraints while enforcing another constraint on first-level variables:

$$k_1 \leq \sum_{x_j \in \mathcal{B}_1} |x_j - \bar{x}_j| \leq k_2,$$

for suitable parameters k_1, k_2, allowing the solution to move away. Also, this constraint is linearized as above. Furthermore, a *tabu* constraint $\sum_{x_j \in \mathcal{B}_1} |x_j - \bar{x}_j| \geq 1$ is added to the MILP in order to prevent cycling.

Fischetti et al. [11] propose criteria for automatically detecting first-level and second-level variables, hence laying the foundations for a general purpose heuristic. The DRT method is shown to perform well on a set of facility location problems arising in telecommunication network design problems, outperforming ad-hoc Variable-Neighborhood Search and Tabu Search.

3 Variables Partitioning Local Search (VPLS)

The VPLS framework can be seen as a local search approach for MIPs, expecially suited for $0-1$ variables, using a generalization of the k-exchange neighborhood. Consider a general MIP

$$\begin{aligned} \text{minimize } & z = C^T X \\ \text{subject to } & AX \leq B \\ & X \in \{0,1\} \end{aligned}$$

where $X^T = (x_1, x_2, .., x_n)$ is a vector of n variables of the problem and $\bar{X}^T = (\bar{x_1}, \bar{x_2}, .., \bar{x_n})$ is a feasible solution to the MIP. If this is the case, it is always possible to define a subset S of a defined size of variables indices $\{1, 2, .., n\}$. The neighborhood $N(\bar{X})$ consists of all solutions of the MIP where the j^{th} variable is equal to the value of the j^{th} variable in $\bar{X}$ for all $j \notin S$.

$$N(\bar{X}) = \{X \mid x_j = \bar{x}_j,\ \forall j \notin S\}$$

The resulting neighborhoods $N(\bar{X})$ can then be searched for an improving solution using a MIP-solver both optimally or approximately. The idea proposed is sketched in Figure 1 and shows a permutation problem where variables belonging to the current solution are partitioned into two sets. A first set $(\bar{X}/S)$ is then reoptimized by means of a MILP solver generating a permutated assignment while variables in the second set (S) keep the same assignment as in the current solution. In Figure 1, the partition to be reoptimized involves variables with indices 1 to 4, but successive partitioning generates all remaining subsets made of four variables. For the example considered, they induce a neighborhood consisting of seven possibly different solutions, obtained applying the MIP solver seven times (each run related to the seven different subsets of four consecutive variables).

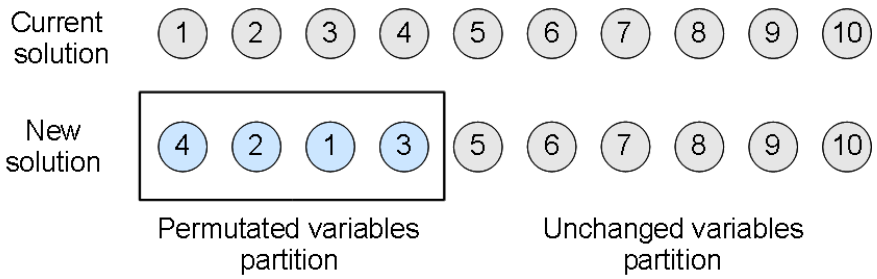

Fig. 1: VPLS framework example

3.1 A VPLS procedure for the Two-Machine total completion time flow shop problem

In [6], the 2-machine flow shop scheduling problem with the objective of minimizing the sum of completion times ($F2||\sum C_j$ in the three-fields notation) is considered. In this problem, a set of jobs $N = \{1, 2, ..., n\}$ is to be scheduled on two machines. Every job consists of 2 operations where the first (the second) one is required to run continuously for p_{1i} (p_{2i}) units of time on the first (second) machine. For each job, the second operation cannot begin if the first one has not been completed. The problem is known to be NP-hard. Also, at least an optimal solution is known to be a permutation schedule, where the (operations of the) jobs share the same sequence on both machines. Thus, equivalently, the permutation flow shop problem $F2|\text{perm}|\sum C_j$ is considered. For this problem, a MIP model can be derived by using so-called positional $0-1$ variables x_{ij}. Let C_{ki} be variables representing the completion times of i-th job processed by machine $k = 1, 2$ and x_{ij} 0-1 decision variables, where $i, j \in \{1, \ldots, n\}$. A variable x_{ij} is equal to 1 if job i is in position j of the sequence, zero otherwise.

The problem can be formulated as follows.

$$\min \sum_{j=1}^{n} C_{2j} \tag{2}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1, \ldots, n \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{4}$$

$$C_{11} = \sum_{i=1}^{n} p_{1i} x_{i1} \tag{5}$$

$$C_{21} = C_{11} + \sum_{i=1}^{n} p_{2i} x_{i1} \tag{6}$$

$$C_{1j} = C_{1,j-1} + \sum_{i=1}^{n} p_{1i} x_{ij} \qquad \forall j = 2, \ldots, n \tag{7}$$

$$C_{2j} \geq C_{1j} + \sum_{i=1}^{n} p_{2i} x_{ij} \qquad \forall j = 2, \ldots, n \tag{8}$$

$$C_{2j} \geq C_{2,j-1} + \sum_{i=1}^{n} p_{2i} x_{ij} \qquad \forall j = 2, \ldots, n \tag{9}$$

$$x_{ij} \in \{0, 1\} \tag{10}$$

where constraints (3)–(4) state that a job is chosen for each position in the sequence and each job is processed exactly once. Constraints (5)–(7) set the completion time of the first job on both machines. For each job, constraints (8)–(9) forbid the start of the 2nd operation on machine *two* before its preceding operation on machine *one* is completed.

With this formulation, a neighborhood can easily be derived and expressed by means of the positional variables. Consider a working sequence $\bar{S}$ corresponding to a valid configuration $\bar{x} = (\bar{x}_{ij} \colon i, j = 1, \ldots, n)$ of the x_{ij} variables. A neighborhood $\mathcal{N}(\bar{S}, r, h)$ can be defined by choosing a position r in the sequence and a size parameter h; let $\bar{S}(r; h) = \{[r], [r+1], \ldots, [r+h-1]\}$ be the index set of the jobs located in the consecutive positions $r, \ldots, r+h-1$ of sequence $\bar{S}$. This can be described with the constraint stating:

$$x_{ij} = \bar{x}_{ij} \qquad i \notin \bar{S}(r; h),\ j \notin \{r, \ldots, r+h-1\}. \tag{S1}$$

The choice of the best solution in the neighborhood $\mathcal{N}(\bar{S}, r, h)$ is accomplished by keeping the sequence unchanged outside the jobs window and optimizing the sequence within the window. The resulting minimization program is solved by means of the MIP solver. The additional constraints (S1) state that in the new solution all jobs that do not belong to the window are fixed in the position they have in the current solution, while the window gets reoptimized. If no improved solution is found a new job-window is selected to be optimized until all possible $O(n)$ windows have been selected. The search is stopped because of local optimality (no window reoptimization offers any improved solution) or because a predefined time limit expires.

3.2 A VPLS approach for Nurse Rostering

In [8], a practical nurse rostering problem is considered, which arises at a ward of a private hospital in Italy. The problem consists in optimally assigning a working shift or a day off to each nurse, on each day of a month,

according to several contractual and operational requirements. This problem can be formulated as a MIP model, where, with n nurses and m days in a month, it is sufficient to introduce a set of $0-1$ variables denoted $x_{i,j,k}$ ($i = 1,..n$, $j = 1,..m$, $k = 1,..l$) indicating if nurse i is assigned to shift k on day j (for a complete model description, see [24]). If a solution $\bar{X}$ ($\bar{X}$: $x_{i,j,k} = \bar{x}_{i,j,k}$, $\forall i = 1,..n$, $j = 1,..m$, $k = 1,..l$) of the model is considered, by adding different constraints to the original ILP model, different neighborhood structures can be represented. A first neighborhood, similar to the Local Branching one, indicates that the number of variables $x_{i,j,k}$ that can change their value from the current solution is less than or equal to a parameter value h_1, namely, the neighborhood size is controlled by parameter h_1. Indeed, h_1 is the maximum Hamming distance (that is the h_1-OPT neighborhood) between the current solution and all feasible solutions of the whole neighborhood. In other words, h_1 is the maximum number of changed shifts in the new solution with respect to the incumbent solution. For this neighborhood the constraint to be added to the model is:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l}\bar{x}_{i,j,k}(1-x_{i,j,k}) + (1-\bar{x}_{i,j,k})x_{i,j,k} \leq h_1.$$

A second neighborhood can consider all subsets of size h_2 (e.g. a week or 10 days) of consecutive days in the solution from day 1 to day $m - h_2 + 1$. Similarly to the previous one, a neighborhood considering all subsets of size h_3 of consecutive nurses in the schedule from nurse 1 to nurse $m - h_3 + 1$ is easy to generate. Also, by building subsets with consecutive days or nurses, complete neighborhoods of a solution are realized. For the last two neighborhoods the constraint to be added to the model is:

$$x_{ijk} = \bar{x}_{ijk} \qquad i,j,k \notin (\bar{X}, h_2 \text{ or } h_3)$$

that is, the variables not belonging to the selected region (made by days or nurses) of the solution space must keep the same value they have in the current solution while the remaining variables are free to change values. Each resulting optimization problem, generated by the three neighborhoods, can be solved by means of the MIP solver. If an improvement is found, the incumbent solution is updated and the process iterates. The mix of these constraints generates a sequence of diversified neighborhoods, indeed falling into a Variable Neighborhood Search field [21] [17] [18].

4 Continous relaxation based (CRB) matheuristics

Methods in this class are characterized by a clever use of information extracted from the LP relaxation of the (MILP model of the) problem. This can be done in a completely general framework or by exploiting specific

problem characteristics. We first consider the so-called *feasibility pump* approach for general MILPs which is now embedded into several commercial and open source packages. Then, we focus on applications where continuous solutions of related problems are of fundamental importance to design solution approaches. In the first application, the relaxation is almost integral inducing an iterative rounding approach that requires the integer solution of very small size problems. In the second application, a partial enumeration, strongly based on the reduced costs of the non basic variables of the continuous relaxation solution, is performed to deal with the problem considered.

4.1 Feasibility pump for general MILPs

An interested reader might hesitate to consider this approach as part of the *matheuristic* framework: after all, rounding techniques for generating feasible solution for MILPs have been around for a long time (see for example [2, 15, 16] etc.). Nevertheless, we consider the feasibility pump proposed by Fischetti et al. [12] as a CRB matheuristic technique because the procedure closely resembles a local search, and it explicitly relies on a MILP solver that is iteratively used as a black box.

Consider a given MILP problem with variables $x = (x_1, \ldots, x_n)$, including a subset of (general) integer variables $\mathcal{I}$ and a subset of binary variables $\mathcal{B}$

$$\begin{aligned} \min \big\{ z = & c^T x \colon x \in P, \\ & l_j \le x_j \le x_j,\ x_j \in \mathbb{Z}_+ \quad \text{for } j \in \mathcal{I}, \\ & x_j \in \{0,1\} \quad \text{for } x_j \in \mathcal{B}. \big\} \end{aligned} \tag{11}$$

The LP-feasible set P is a bounded polyhedron described by some system of linear constraints $Ax \le b$.

The basic feasibility pump procedure defines a (linear) distance function $\Delta(\bar{x}, \tilde{x})$ between two solutions $\bar{x}$, $\tilde{x}$. In [12] a L_1 norm is used:

$$\Delta(\bar{x}, \tilde{x}) := \sum_{x_j \in \mathcal{I}} |\bar{x}_j - \tilde{x}_j|.$$

The procedure first computes the optimal solution x^* for the LP-relaxation, then executes two steps iteratively:

$$\text{Rounding:} \quad \text{set } \tilde{x}_j := \lfloor x^*_j + 0.5 \rfloor \text{ for } x_j \in \mathcal{B} \cup \mathcal{I}; \tag{I}$$

$$\text{Solve and update:} \quad \text{set } x^* = \min \Big\{ \Delta(x, \tilde{x}) := \sum_{x_j \in \mathcal{I}} |x_j - \tilde{x}_j| \colon x \in P \Big\}. \tag{II}$$

Step (I) is a nearest-integer rounding. Note that the integer solution $\tilde{x}$ in general does not belong to P, i.e., it is not feasible. Step (II) solves a linear program. The auxiliary objective function $\Delta(x, \tilde{x})$ can be linearized as

$$\Delta(x, \tilde{x}) = \Delta^{\mathcal{B}}(x, \tilde{x}) + \Delta^{\mathcal{I}}(x, \tilde{x})$$

with

$$\Delta^{\mathcal{B}}(x,\tilde{x}) = \sum_{x_j \in \mathcal{B}:\ \tilde{x}_j=0} x_j + \sum_{x_j \in \mathcal{B}:\ \tilde{x}_j=1} (1-x_j)$$

and $\Delta^{\mathcal{I}}(x,\tilde{x})$ involving auxiliary variables d_i, $x_i \in \mathcal{I}$:

$$\Delta^{\mathcal{I}}(x,\tilde{x}) = \sum_{x_j \in \mathcal{I}:\ \tilde{x}_j=l_j} (x_j - l_j) + \sum_{x_j \in \mathcal{I}:\ \tilde{x}_j=u_j} (u_j - x_j) + \sum_{x_j \in \mathcal{I}:\ l_j<\tilde{x}_j<u_j} d_j$$
$$\text{subject to} \quad d_j \geq x_j - \tilde{x}_j,\ d_j \geq \tilde{x}_j - x_j \qquad \forall x_j \in \mathcal{I}, l_j < \tilde{x}_j < u_j$$

The feasibility pump procedure tries to generate a relaxed optimum which lies as close as possible to $\tilde{x}$. If a LP-solution x^* is found with $\Delta(x^*,\tilde{x}) = 0$, then $x^* = \tilde{x}$ and this solution is feasible for (11). Steps (I)–(II) are iterated until $\tilde{x}$ becomes feasible or some time limit is reached. Some care needs to be paid to cycle avoidance: hence if $\tilde{x} = x^*$ just after the update step but such $\tilde{x}$ is still unfeasible, the latter acquires a random perturbation on a certain fraction of its components.

The basic scheme has the drawback that it only focuses on feasibility, without taking into account the quality of the solution generated, similarly to the phase-1 optimization in the simplex algorithm for linear programming. Also, the authors report that this procedure is very effective on binary MILPs (i.e., when $\mathcal{I} = \emptyset$) while the cases with general-integer variables ($\mathcal{I} \neq \emptyset$) are harder. In [12], the solution quality is taken into account by adding to the linear program of Step (II) a bounding constraint $c^T x \leq \text{UB}$. The value for UB is updated by $\text{UB} = \alpha c^T x^* + (1-\alpha) c^T x^H$, where x^H is the best known feasible solution delivered by previous stages of the feasibility pump. This actually turns the procedure into a local search, and the stopping criterion is no longer based on feasibility but on the maximum number of iterations allowed. Bertacco, Fischetti and Lodi [3] propose a refined procedure, structured into three stages. In the first stage only integrality of binary variables is taken into account (i.e., $\Delta(x,\tilde{x}) = \Delta^{\mathcal{B}}(x,\tilde{x})$) neglecting the integrality of variables in $\mathcal{I}$. This stage is terminated when integrality is achieved for all variables in $\mathcal{B}$, or a given number of iteration has been executed without (significantly) decreasing $\Delta^{\mathcal{B}}(x,\tilde{x})$. In the second stage, all integer variables $\mathcal{B} \cup \mathcal{I}$ are handled, using as starting $\tilde{x}$ the best point generated by the previous stage. The stage is terminated when integrality of all variables in $\mathcal{B} \cup \mathcal{I}$ is achieved or a number of iterations have been executed without improving $\Delta(x,\tilde{x})$. Finally, a third stage is executed if even the second stage is unsuccessful, by optimizing via truncated branch

and bound the MILP

$$\begin{aligned}\min \big\{ z = & \Delta(x, \tilde{x}) \colon x \in P, \\ & l_j \leq x_j \leq x_j,\ x_j \in \mathbb{Z}_+ \quad \text{for } x_j \in \mathcal{I}, \\ & x_j \in \{0,1\} \quad \text{for } x_j \in \mathcal{B} \big\}.\end{aligned}$$

Achterberg and Berthold [1] propose a further improvement to the refined procedure where the distance objective function $\Delta(x, \tilde{x})$ is replaced by

$$\hat{\Delta}^S(x, \tilde{x}; \alpha) = (1 - \alpha)\Delta^S(x, \tilde{x}) + \alpha \frac{\sqrt{|S|}}{\|c\|} c^T x$$

where $S = \mathcal{B}$ or $\mathcal{B} \cup \mathcal{I}$, in accordance with each of the three stages. The parameter $\alpha \in (0, 1)$ is used for mixing the original objective function with the integrality measure, in order to improve the quality of the generated feasible solution. This parameter is geometrically decreased at each iteration, multiplying it by a prefixed factor $\varphi \in (0, 1)$. The approach of [1] shows strongly improved performances with respect to the procedure developed by Bertacco et al [3] on a large number of MILP examples from the MIPLIB 2003 collection, although no dominance can be proved.

Recently, Fischetti and Salvagnin [13] have applied domain-reduction and constraint propagation techniques in order to realize a sophisticated rounding step (I). This further improves performances of the feasibility pump in practice, although again no dominance is proven over previous versions.

4.2 A CRB matheuristic procedure for the Closest String Problem

In [7], the Closest String Problem (CSP) is tackled. This problem can be stated as follows. Consider a set $S = \{s_1, ..., s_m\}$ of m strings over an alphabet $\Sigma = \{1, ..., h\}$ where each string $s_i \in S$ has length n. The CSP is to find a minimal integer d (and a corresponding string t of length n over the same alphabet Σ) such that each string s_i has Hamming distance at most d to t. In CSP, any string of length n is feasible provided that each element of the string contains a symbol belonging to the specific alphabet. To derive an ILP model of CSP, let us denote by $s_i(k)$ the value of string s_i in position k. Also, let $x(j, k)$ be a binary variable where $x(j, k) = 1$ if string t has value j in position k, otherwise $x(j, k) = 0$ (that is $x(j, k) = 1 \iff t(k) = j$). Correspondingly, the following ILP model proposed in [20] holds for CSP.

$$\min d \tag{12}$$

$$\sum_{j=1}^{h} x(j,k) = 1 \;\; \forall k \in 1...n \tag{13}$$

$$n - (\sum_{k=1}^{n} x(s_i(k),k)) <= d \;\; \forall i \in 1...m \tag{14}$$

$$x(j,k) \in \{0,1\} \;\; \forall j \in 1...h, \;\; \forall k \in 1...n \tag{15}$$

$$d \geq 0. \tag{16}$$

The cost function (12) minimizes the maximum Hamming distance d. Also, constraints (13) indicate that string t must contain a symbol of the alphabet $\Sigma = \{1, ..., h\}$ in each position $k \in 1...n$ and constraints (14) indicate that for each string s_i, the Hamming distance between t and s_i must not exceed d. Finally, constraints (15) indicate that all $x(j,k)$ variables are binary, while constraint (16) indicates that d is positive. For most instances of CSP, the continuous relaxation of the model is almost integral. Thus, in a heuristic framework, the solution space to be analyzed to generate a complete integer solution is strongly reduced. Henceforth, by exploiting the integral part of the continuous relaxation, given the optimal solution $\overline{X}$ of the continuous relaxation of the model, to get a complete integer feasible solution, it is sufficient to consider the subset of constraints involving fractional variables. A way to do this is to keep all $x(j,k)$ variables such that $\overline{x(j,k)}$ is integer unchanged and to consider all positions k where the continuous relaxation solution is fractional and simply round up the $\overline{x(j,k)}$ variables with largest fractional value. Alternatively, a more efficient approach can be expressed as follows. One at a time, the fractional variables of the continuous solution of model (12–16) are set to 1 and the continuous model is re-run. Hence, if k are the fractional variables, we get $k+1$ different continuous solutions. Then, all variables $x(j,k)$ that are integer and keep the same value in all runs, can be fixed to that value and are excluded from the core problem. Finally, the remaining integer core problem is solved to optimality by means of the ILP solver.

4.3 A CRB matheuristic procedure for the Multi-dimensional Knapsack Problem

A MILP based partial enumeration heuristic has been developed in [5] for the 0-1 multidimensional knapsack problem ($0-1MKP$). The $0-1MKP$ is a well-known NP-hard optimization problem. It can be stated as follows.

$$\max \sum_{j=1}^{n} p_j x_j \tag{17}$$

$$\sum_{j=1}^{n} w_{ij}x_j \leq c_i, \quad i = 1, ..., m \tag{18}$$

$$x_j \in \{0, 1\}, \quad j = 1, ..., n \tag{19}$$

that is, we have n items with profits $p_j > 0$ and m resources with capacities $c_i > 0$, where each item j consumes an amount $w_{ij} \geq 0$ from each resource i and the selected items must not exceed the resource capacities c_i. There are correspondingly n $0-1$ decision variables x_j indicating which items are selected and the goal is to choose a subset of items with maximum total profit.

The proposed partial enumeration approach is based on a generalization of the enumeration scheme of [26]. Consider the optimal solution $\overline{Z}$ of the LP relaxation of $0-1MKP$ and let Y be the set of x_j variables such that $\overline{x_j}$ is integer. In [26], an exact algorithm was proposed for $0-1$ integer programming where a search tree was generated according to the following binary branching scheme: either all variables in Y are set to their $\overline{x_j}$ (integer) value, or for at least one variable $x_j \in Y$, its optimal solution value must be $x_j^* = 1 - \overline{x_j}$. In [5], the enumeration scheme of [26] is generalized, by taking into account the reduced costs of the variables in Y. More in details, only a subset of the variables in Y is set to the related $\overline{x_j}$ (integer) value, namely the subset Δ of non-basic variables with largest $|r_j|$. Then, either for all these x_j variables we have $x_j^* = \overline{x_j}$, or for at least one of these variables we have $x_j^* = 1 - \overline{x_j}$. Correspondingly, the following branch can be performed.

Left branch: all variables $x_j \in \Delta$ with largest $|r_j|$ are fixed inducing a max smaller for $|\Delta|$ large enough;

Right branch: the inequality $\sum_{j\in\Delta:\overline{x_j}=0} x_j + \sum_{j\in\Delta:\overline{x_j}=1}(1-x_j) \geq 1$ is added.

For $|\Delta|$ large enough (even though $|\Delta| < |Y|$), the left branch induces, at least for the first levels of the related search tree, subproblems that can be solved to optimality within a very short time by means of an ILP solver. Hence, a heuristic partial enumeration approach can be devised where only the left branches of the search tree are tackled through a depth first search. These subproblems are either solved to optimality or stopped due to CPU time limit. Notice that the approach is strongly parallelizable as starting from the root node, only the continuous solution of the parent node is required to compute the reduced costs and correspondingly the inequality added to the child node related to the right branch. At each branch, the number of basic variables increases (due to the added inequality) and after few levels of the tree the ILP subproblem corresponding to the left branch is no longer solvable to optimality and a CPU time limit $T_{\max}$ is added. An overall limit $SB_{\max}$ of branches is then imposed to bound the total CPU time.

5 Greedy based matheuristics

Beyond the approaches proposed, other solution methods can be developed expoiting both heuristic frameworks and mathematical programming. In particular, for large size problems, solution procedures based on concepts related to *constructive heuristics* can be conceived. Below, we provide an example of a greedy procedure with embedded a MILP solver dealing with a well known and challenging logic puzzle problem.

5.1 A greedy based matheuristic procedure for the Eternity II problem

A constructive heuristic based on the MILP formulation of the problem has been developed for the Eternity II (EII) puzzle problem [25]. The Eternity II puzzle is an edge matching puzzle with 256 square tiles that need to be placed in a grid with 16 rows and 16 columns, whereby each tile has four edges with a colored pattern. A solution to this puzzle requires that each tile is placed at a unique position in the grid such that the edge shared between any two adjacent tiles is matched. The puzzle consists of an 16×16 square on which 16^2 tiles need to be placed. The index $t = 1, \ldots, 16^2$ is used to refer to tiles. The indices $r = 1 \ldots 16$, $c = 1, \ldots, 16$ denote the rows, resp. columns of the puzzle board. The index $\alpha = 0, \ldots, 3$ refers to the rotation of the tile. The heuristic is based on subproblem optimization. Subproblems are generated from the MIP model but only take into consideration adjacent regions of fixed size and shape in the puzzle to be assigned. The variables corresponding to this region are then optimally assigned by the MILP solver. Iteratively, non-overlapping, adjacent regions are solved to optimality, until all tiles have been assigned to a unique position in the puzzle. It is also required that the optimal value of each subproblem is zero, which means that all tiles in the subregion should match. Whenever the lower bound of a subregion is detected to be greater than zero by the solver, the subregion optimization is stopped. The previously optimized subregion is then reconsidered within a backtracking algorithm so as to find a different partial solution in which the current region may have a zero lower bound. In order to cut off the previous partial solution, the following constraint is added to the model:

$$\sum_{t=1}^{n^2} \sum_{r=1}^{n} \sum_{c=1}^{n} \sum_{\alpha=0}^{3} \bar{x}_{t,r,c,\alpha} \cdot x_{t,r,c,\alpha} \leq n^2 - 1$$

where $\bar{x}_{t,r,c,\alpha}$ denotes the value of the x variables in the current partial solution of size n. This constraint requires a different assignment of variables than the one in the current solution, thus forbidding the current solution. Whenever no solutions of the previous region lead to a zero lower bound in

the current region, the procedure backtracks further and the algorithm continues. As can be noticed, this procedures can lead to incomplete solutions even if a large time limit is fixed. Hence,the backtracking procedure is run for a fixed time limit, then the solution approach continues, with the same size and shape subproblems, with a pure greedy heuristic until a complete solution is generated. In this case, in order to provide different solutions, a random inner tile is placed within the first region, before the backtracking heuristic starts.

6 Conclusions

The availability of reliable and efficient MILP solvers, combined with fast multicore processors, make it possible to embed the solution of non trivial mathematical programs as subroutines of solution algorithms. This would not have been possible even 15 years ago, when a MILP model was usually a monolithic object to be solved in one run or simply studied in order to gain insight into the problem structure. The result of combining sophisticated search heuristics with powerful MILP solvers gave birth to the so called *Matheuristics* for combinatorial optimization problems.

We have surveyed four different approaches of embedding solvers into heuristic algorithms, namely:

- local search based on the Hamming distance,
- local search based on variables partitioning,
- clever use of the continous relaxation,
- greedy constructive techniques enhanced by mathematical programming.

The literature surveyed provides evidence of effectiveness of matheuristic solution procedures both on complex real-world problems as well as on classical academic models.

In all applications, an essential point is the mathematical formulation of the problem to be studied. The importance can be found in different aspects: on one hand the mathematical formulation is a fundamental part of the solution approach so as to be able to generate neighborhoods iteratively modifying that formulation as in the Network Design, Scheduling and Timetabling examples and then applying the MILP solver to the newly generated subproblems. Also, the testing of new neighborhoods in most cases simply requires adding a constraint to the related MILP model. On the other hand, the mathematical models are of major relevance because their relaxations can induce properties or characteristics to be exploited when developing a solution method such as in the Feasibility Pump, Closest String Problem and Multi-dimensional Knapsack Problem.

Finally, models can serve as the basic problem formulations to be successively extended for constructive approaches when there are no feasibility issues such as in the Eternity II problem where subsets of variables are optimized by a MILP solver until a complete solution is constructed.

The extensive use of models and MILP solvers is also a basic element in designing a matheuristic approach because the use of solvers as subroutines may allow to substantially decrease the effort in software code development, thus letting most of the effort to be dedicated to the algorithmic design.

References

[1] Achterberg T., Berthold T., "Improving the feasibility pump", Discrete Optimization, 4, pp. 77-86, 2007.

[2] Balas E., Martin C. H., "Pivot-and-complement: a heuristic for 0-1 programming", Management Science, 26, pp. 86-96, 1980.

[3] Bertacco L., Fischetti M., Lodi A., "A feasibility pump heuristic for general mixed-integer problems", 4, pp. 63-76, 2007.

[4] Blum C., Roli A., "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", ACM Computing Surveys, 35, pp. 268-308, 2003.

[5] Della Croce F., Grosso A., "Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem", Computers and Operations Research, 39, pp. 27-31, 2012.

[6] Della Croce F., Grosso A., Salassa F., "A matheuristic approach for the two-machine total completion time flow shop problem", Annals of Operations Research, forthcoming.

[7] Della Croce F., Salassa F., "Improved LP-based algorithms for the closest string problem", Computers and Operations Research, 39, pp.746-749, 2012.

[8] Della Croce F., Salassa F., "A Variable Neighborhood Search Based Matheuristic for Nurse Rostering Problems", PATAT 2010 Proceedings, 8th International Conference on the Practice and Theory of Automated Timetabling, pp. 167-175, Belfast (UK) 10 - 13 August, 2010.

[9] El-Abd M., Kamel M., "A taxonomy of cooperative search algorithms", In Blesa M.J., Blum C., Roli A., Sampels M. (eds.): Hybrid Metaheuristics: Second International Workshop. LNCS, 3636, pp. 32-41, Springer, 2005.

[10] Fischetti M., Lodi A., "Local Branching", Mathematical Programming B, 98, pp. 23-47, 2003.

[11] Fischetti M., Polo C., Scantamburlo M., "A local branching heuristic for mixed-integer programs with 2-level variables with an Application to a Telecommunication Network Design Problem", Networks, 44, pp. 61-72, 2004.

[12] Fischetti M., Glover F., Lodi A., "The feasibility pump", Mathematical Programming, 104, pp. 91-104, 2005.

[13] Fischetti M., Salvagnin D., "Feasibility Pump 2.0", Mathematical Programming Computation, 1, pp. 201-222, 2009.

[14] Garey M.R., Johnson D.S., "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman, New York, NY, 1979.

[15] Glover F., Laguna M., "General purpose heuristics for integer programming — part I", Journal of Heuristics, 2, pp. 343-358, 1997.

[16] Glover F., Laguna M., "General purpose heuristics for integer programming — part II", Journal of Heuristics, 3, pp. 161-179, 1997.

[17] Hansen P., Mladenovic N., "Variable neighborhood search: Principles and applications", European Journal of Operational Research, 130, pp. 449-467, 2001.

[18] Hansen P., Mladenovic N., Urosevic D., "Variable neighborhood search and local branching", Computers and Operations Research, 33, pp. 3034-3045, 2006.

[19] Jourdan L., Basseur M., Talbi E.G., "Hybridizing exact methods and metaheuristics: a taxonomy", European Journal of Operational Research, 199, pp. 620-629, 2009.

[20] Liu X., Liu S., Hao Z., Mauch H., "Exact algorithm and heuristic for the Closest String Problem", Computers and Operations Research, 38, 1513-1520, 2011.

[21] Mladenovic N., Hansen P., "Variable neighborhood search", Computers and Operations Research, 24, pp. 1097-1100, 1997.

[22] Nemhauser G., Wolsey L., "Integer and Combinatorial Optimization", John Wiley & Sons, New York, NY, 1988.

[23] Raidl G.R., "A unified view on hybrid metaheuristics", In: Almeida F., Blesa Aguilera M.J.B., Blum C., Moreno Vega J.M., Perez Perez M.,

Roli A. (eds.): Hybrid metaheuristics. LNCS, 4030, pp. 1-126. Springer, Berlin, 2006.

[24] Salassa F., "Matheuristics for Combinatorial Optimization Problems: Applications to Services and Production Systems", PhD Thesis, Politecnico di Torino, February 2011.

[25] Salassa F., Vancroonenburg W., Wauters T., Della Croce F., Vanden Berghe G., "Matheuristics for the Eternity II puzzle", Technical Report TR-02-02-2012, D.A.I. Politecnico di Torino, 2012.

[26] Soyster A. L., Lev B., Slivka W., "Zero-one programming with many variables and few constraints", European Journal of Operational Research 2, pp. 195-201, 1978.