Chapter 1

TWO GENERIC SCHEMES FOR EFFICIENT AND ROBUST COOPERATIVE ALGORITHMS

Emilie Danna
ILOG and LIA, Université d'Avignon, CRNS-FRE 2487
edanna@ilog.fr

Claude Le Pape
ILOG
clepape@ilog.fr

Introduction

In the recent past, many examples of hybrid applications have been described and the adopted hybrid algorithms shown to provide better results than pure algorithms based on only one technology. Yet the number of cases in which cooperative strategies have been applied remains very small compared to the number of cases in which they could potentially prove useful.

There might be several explanations for this fact. First, most work on cooperative optimization strategies has been focused either on the resolution of a particular problem at hand or on the design of generic software integration principles. A finer understanding of what makes a cooperative strategy work on a particular application and of the variants of the problem that could be solved well with the same cooperative strategy would be needed to allow an efficient generalization to similar yet different applications. Second, cooperative strategies are hard to develop:

- Appropriate sub-models or alternative models have to be designed and implemented, which often takes more time than the design and implementation of a unique model. One could actually argue that most experimental results from the literature showing the superiority of a cooperative strategy are biased by the fact that the cooperative strategy took much longer to develop than the non-cooperative strategies against which it was tested.

- The design of the cooperative strategy requires some understanding of what each component does, of the strengths and weaknesses of different technologies for the resolution of different types of problems. Very few people if any have a deep understanding of the four main classes of optimization techniques that can be used: (1) efficient (generally, polynomial or pseudo-polynomial) operations research algorithms, (2) constraint programming, (3) linear and mixed integer programming, and (4) local search.
- Cooperative strategies are hard to tune, probably just because the number of parameters that can be tuned is much higher than in the case of a "pure" strategy. Good tools and introductions to these tools are necessary to allow more developers of optimization applications to design, implement, and tune cooperative problem-solving strategies.

This chapter relies on examples from the literature to convey some understanding, or at least some intuition, of what types of cooperative strategies may be worth developing for a given optimization application. Given an optimization application, two complementary issues must be considered. First, the application must be "efficient", *i.e.*, provide a "good" (ideally, optimal) solution within reasonable time and resource limits (CPU, memory). Second, the application must be "robust" with respect to three types of variations in the problem instances: variations in problem size, variations in numerical characteristics, and addition of side constraints. It shall in most cases be easy to include additional constraints without re-designing the overall problem-solving strategy. The goal of hybridization is to improve either efficiency or robustness (or both) without sacrificing the other.

The four main classes of optimization techniques listed above present different characteristics with respect to the efficiency versus robustness compromise:

- Polynomial operations research algorithms exploit the problem structure of well-defined problems to solve them to optimality in time bounded by a polynomial function of the problem size. When the degree of the polynomial function is low, the algorithm is efficient and in general robust to changes in problem size and numerical characteristics. But in most cases the algorithmic efficiency is based on a set of compatible dominance properties $\{p_1, p_2, \ldots, p_n\}$, which state that at least one of the optimal problem solutions satisfies p_1, p_2, ..., and p_n. The use of such dominance rules strongly reduces the set of alternative solutions to be considered. However, dominance properties are in general not stable against the addition of side constraints to the original problem, *i.e.*, when the problem definition is extended, the dominance rules are no longer correct and the algorithm is likely to miss the optimal solutions or even all admissible solutions to the extended problem.

- Constraint programming (*cf.* Chapter **??**) is a problem solving paradigm which establishes a distinction between, on the one hand, a precise definition of the constraints that define the problem to be solved and, on the other hand, the algorithms and heuristics enabling the selection and cancellation of decisions to solve the problem. In constraint programming, a purely deductive process referred to as "constraint propagation" is used to propagate the consequences of problem constraints and decisions. This process is applied each time a new decision is made, and is clearly separated from the decision-making algorithm (usually some form of tree search) *per se.* Most importantly, the overall constraint propagation process results from the combination of several local and incremental processes, each of which is associated with a particular constraint or a particular constraint class. This means that in constraint programming, any added side constraint can propagate and play a role in the resolution of the problem. Constraint programming is particularly efficient when constraint propagation is fast and effective, *i.e.*, if the critical constraints propagate well and, in an optimization context, if a tight bound on the optimization criterion results (by propagation) in constraints that effectively guide the search toward a good solution. Compared to other techniques, constraint programming tends to be more robust against the addition of side constraints, but it is not that robust against changes in problem size and numerical characteristics.

- Mixed integer programming (*cf.* Chapter **??**) also establishes a distinction between the definition of the problem and its resolution. But it gets much of the guidance toward good solutions from the continuous relaxation of the problem. Mixed integer programming is especially efficient whenever the continuous relaxation is a good approximation of the convex envelope of the solutions (at least around the optimal solution) or when the relaxation can be iteratively tightened (by adding cuts) to improve this approximation.

Much work has been done to make commercial implementations rather robust with respect to changes in problem size and numerical characteristics, but there are some problems for which the size of the mixed integer programming formulation grows much faster than the size of the problem. Linear side constraints are easy to accomodate, but the addition of non-linear or disjunctive side constraints generally enforces the addition of new integer variables in the linear model. In both cases, the continuous relaxation may become a bad approximation of the mixed integer problem.

- Various forms of local search (*cf.* Chapter **??**), operating either on individual solutions (*e.g.*, simulated annealing, tabu search) or on populations of solutions (*e.g.*, genetic algorithms, path relinking), provide excellent results on some problems. Roughly speaking, local search tends to be efficient when good problem solutions share characteristics (*e.g.*, the ordering of two tasks) that can be compactly represented and that are likely to be kept when local search operators proceed from a solution or a set of solutions to the next. The robustness of a local search algorithm depends on the representation, operators, and overall strategy that are used. For example, some local search operators may become too expensive to apply systematically when problem size grows, or even useless when a side constraint is added (*e.g.*, a constraint relating the ordering of two tasks to the ordering of two other tasks, in a complex manner). If the effect of adding a side constraint is that the density of feasible solutions is reduced or that simple problem characteristics are less often shared by good solutions, the local search algorithm needs to be revised.

Looking at examples from the literature, two generic cooperative schemes emerge. The **decomposition** scheme consists in applying one of the above techniques (the "slave") to a sub-problem of the overall problem, in order to gain information. This information is then used by another of the above techniques (the "master"), to solve the overall problem. The sub-problem given to the slave may be a reduced problem problem with less variables, an approximation, a relaxation, or a tightened version of the overall problem. In this case, some of the results given by the slave may be used by the master to focus the overall search, without any risk of missing the optimal solution. In other cases, the results given by the slave may be used as a guide toward good solutions, but do not strictly allow to reduce the search space.

In the **multiple search** scheme, two or more optimization techniques are used in turn or in parallel to provide solutions to the full problem. In the simplest case, each optimization component operates independently from the others. In most cases, however, the solutions found by one component and, more generally, information generated by one component as part of its own optimization process, can be used by other components to better focus subsequent search. Gains in efficiency and robustness stem from introducing diversification in the methods used to explore the search space, as well as some intensification on some promising parts of the search space.

The remainder of this chapter provides examples from the literature which illustrate the use of these two schemes. It is organized in six sections. Each section is dedicated to examples of cooperation involving two of the four classes of techniques discussed above. It is not intended to be exhaustive, but attempts to expose the most significant ways we are aware of to apply the above schemes.

1. Operations Research Algorithms and Constraint Programming

Obviously, the use of a cooperative scheme is not likely to be worthwhile when an efficient operations research algorithm can solve the complete problem under consideration. Hence, operations research algorithms either work alone or appear as slaves in the **decomposition** scheme, working on sub-problems or on simplified versions of the problem to provide guidance to another algorithm.

In the case of constraint programming, two different roles can be taken by the operations research algorithm: either it is used to reinforce constraint propagation, *e.g.*, deduce better bounds for variables of the problem, including the variable representing the cost of a solution, or it is used to heuristically guide the constraint-based tree search to promising regions. The main overall advantage of the combination is that the specific operations research algorithm brings efficiency to the constraint programming framework, while constraint programming is far more robust than the specific algorithm with respect to the addition of side constraints.

Chapter **??** provides many examples of so-called "global" constraints for which polynomial algorithms can be used to reinforce propagation. The improvement in efficiency, compared to more "naive" constraint propagation methods can be significant. The most striking examples come from the scheduling domain. For example, the "edge-finding" algorithm, first presented in [47, 15, 16], was adapted by various researchers to the constraint programming framework (*e.g.*, [43, 17, 5]), and generalized by these researchers to more general cases (*e.g.*, [44, 19, 7]). Results obtained on an industrial project scheduling problem [5] show how the use of edge-finding enabled the resolution in at most five minutes of problems which were not solved in one hour, using standard constraint propagation techniques.

Another typical example is the use of flow or assignment algorithms within global constraints. For example, Régin [51] describes an algorithm, based on matching theory, to achieve the global consistency of the "all-different" constraint. This constraint is defined on a set of variables and constrains these variables to assume pairwise distinct values. Régin's algorithm maintains arc-consistency on the n-ary "all-different" constraint, which is more powerful than achieving arc-consistency for the $n * (n-1)/2$ corresponding binary "different" constraints. Both Caseau and Laburthe [20] and Focacci, Milano and Lodi [31] have proposed extensions of such a global constraint to the case in which the cost to minimize is defined by summing individual costs assigned to each selected assignment.

In the scheduling domain, the case in which the cost function to minimize is a sum (like the weighted number of late jobs or the sum of setup times between activities) is also of particular interest. Propagating an upper bound on the cost, each time a new improving solution is found, is not a problem when the optimization criterion is a "maximum" such as the makespan (maximal completion time) or the maximal tardiness of a given set of activities. Indeed, an upper bound on the optimization criterion is directly propagated on the completion time of the activities under consideration, *i.e.*, the latest end times of these activities are tightened efficiently. On the contrary, when the cost function is a sum, an upper bound on the cost does not directly provide good upper bounds on the different terms of the sum because it is not clear which terms actually have to be reduced. In such situations, better results can be obtained by combining constraint programming with deductive algorithms targeted towards specific cost functions. See, for example, [8] for the weighted number of late jobs and [30] for the sum of setup times.

A good example in which an operations research algorithm is used to guide the search is presented in [40]. The goal is to route commodities over a network and dimension the links of the network so as to minimize the dimensioning costs. A shortest path algorithm is used as a heuristic. It determines the route with the smallest marginal cost for a given commodity, given the commodities that have already been routed. Search proceeds by imposing this route and backtracking on it if necessary. An analysis of the explored search trees showed that during the search almost all the improving solutions questioned one of the first routing decisions that had been taken in order to build the previous solution, *i.e.*, that in the middle and close to the leaves of the search tree, this heuristic almost always led to the best solution, given the choices made close to the root of the search tree. Experimental results with limited CPU time showed that using graph-oriented algorithms, both for heuristic choices and constraint propagation, enabled to save 14% of the costs on average over seven small instances provided by France Telecom. Without these graph-oriented algorithms, constraint programming would not be a viable option for the resolution of these problems.

2. Operations Research Algorithms and Mixed Integer Programming

As in the previous section, operations research algorithms are used as slaves in a **decomposition** scheme, working on sub-problems or on simplified versions of the problem to provide guidance to a MIP solver which acts as a master.

A typical example is column generation or branch-and-price [9]. Chapter **??** explains in details how column generation work, but let us describe the idea on a specific problem: the vehicle routing problem with time windows. The problem consists in assigning to each vehicle of a given pool an ordered list of customers to visit, within allowed time windows. Each "column" corresponds to one feasible route, starting from and ending at the vehicle depot and visiting several customers in their allowed time windows. The master problem is a set covering problem, that selects a number of such columns to cover all customers and to minimize the overall traveling cost. The number of possible columns is exponential so they cannot all be present at the beginning like in mixed integer programming, but they are generated on the fly. At the beginning, a set of naive columns compose a gross approximation of the convex hull of the integer solutions. At each iteration, this approximation is refined by solving a sub-problem to introduce new columns with negative reduced cost (*cf.* Chapter **??** for an introduction to reduced costs): according to the continuous relaxation of the set covering problem, these columns represent a better way to visit customers than the already existing columns. The sub-problem is a shortest path with time windows and it can be solved with any kind of optimization method. It is usually solved with a dynamic programming algorithm [26] that has pseudo-polynomial complexity but is not too expensive in practice. This algorithm is based on labels: each partial path from the depot to node j is associated to a cost and a label, that is a multidimensional vector corresponding to the quantity of each resource used by this path. Partial paths are progressively extended and a dominance relation allows to remove dominated partial paths following Bellman's optimality principle: all non-dominated paths are the extension of a non-dominated path. In other words, if at node j a partial path arrives sooner and is shorter than another path, then this latter path can be removed because all of its extensions will be dominated by the extensions of the former partial path. Additional constraints can easily be taken into account if they are accumulation-type constraints, that is if the state of the constrained resource at node j only depends on the state of the resource at the predecessor of node j in the partial path considered: one only needs to add to the labels one dimension for each additional constrained resource. For example, a capacity constraint limiting the amount of goods that can be carried by each vehicule can be taken into account in that way. This label-based algorithm can also be adapted to a more complex additional constraint: generate only elementary routes, as reported in [28] by adding to each label n 0-1 dimensions (if n is the number of customers), and independently in [23] by modifying the dominance relation.

Another example is Lagrangean relaxation. It consists in relaxing some constraints by penalizing the violation of these constraints in the cost function. The Lagrangean relaxation often yields lower bounds of good quality (when minimizing). This technique is especially interesting if the constraints matrix is structured in k independent sub-problems, be it for a group of coupling constraints that link together all variables. If the group of coupling constraints is dualized, the k sub-problems can be solved independently, and they are often easy to solve because they are smaller in size and "purer" in structure than the initial problem. Therefore dedicated operations research algorithms can be used to solve them. This is a typical example of the **decomposition** scheme. For example, Gendron and Crainic [33] dualize the flow conservation constraints in a network design problem. This leads to continuous knapsack sub-problems which are easily solved by sorting the knapsack cost coefficients.

3. Constraint Programming and Mixed Integer Programming

Combining constraint programming and linear programming or mixed integer programming is the most studied combination in hybrid optimization.

The **multiple search** scheme is not often used, probably because constraint programming really works better than mixed integer programming on some applications, while mixed integer programming really works better than constraint programming on the others, thereby leading to the selection of one of these paradigms to be the master. Yet some exceptions might occur. For example, if we consider the above-mentioned network dimensioning problem [40], it appeared in the course of the study that on some instances constraint programming provided in 5 minutes solutions much better than those found by mixed integer programming in 10 minutes, and vice-versa. In such a case, running the two algorithms, each for 5 minutes, would be beneficial. Needless to say, more complex multiple search strategies can also be considered. For example, constraint programming can be used to quickly construct good solutions and mixed integer programming to get lower bounds and optimal solutions — using the upper bounds from constraint programming to prune the search tree. Branches that are cut off by one of the two algorithms can also be transformed into additional constraints for the other algorithm, preventing a given set of decisions to be made on the same branch, and thereby preventing the exploration of a useless branch.

The **decomposition** scheme with constraint programming as the master is the most often used, because it allows the hybrid algorithm to benefit both from the efficiency of the linear solver and from the robustness of constraint programming to the addition of side constraints. The most classical **decomposition** consists in passing a linear sub-problem, *i.e.*, a set of linear constraints and possibly a linear objective to minimize, to a linear programming solver. The linear solver determines whether the given constraints can be simultaneously satisfied and provides a lower bound on the objective. It can also adjust variable domains when inequalities are implied. CLP(R) [36], CHIP [35], Prolog III [24], are early examples of constraint programming languages embedding a linear solver. Such an integration can be viewed as a particular case of global constraint propagation: the linear solver operates on a global constraint expressed as a conjunction of linear constraints, *i.e.*, on the continuous relaxation of a sub-problem. Note that beside domain adjustments, the linear solver also provides the optimal continuous solution of the sub-problem. When the sub-problem is “close” to the complete problem, this solution can also be used by the constraint programming master as a heuristic guide towards a good solution of the complete problem. A particular example in the field of dynamic scheduling is given in [55]. In this example, the linear solver includes only temporal constraints (some of which have been added to the initial problem in order to ensure the satisfaction of resource sharing constraints) and the definition of the optimization criterion as the total deviation of start times of activities from the start times of the same activities in a reference schedule. An interesting characteristic of this model is that the optimal continuous solution of the linear sub-problem is guaranteed to be integral; hence, either this solution satisfies all the resource constraints and it is optimal, or it violates some resource constraint which can be used to branch on the order of two conflicting activities.

Refinements of this **decomposition** scheme are presented in [41]. For example, when strong bounds on the objective are known, reduced costs resulting from linear solving can also be used to adjust variable domains (*cf.* Chapter **??**). Milano and van Hoeve [42] present a different use of reduced costs to guide constraint programming. First a linear relaxation of the original problem is solved. The resulting reduced costs are used to rank values inside each variable domain, which are accordingly divided into a *good* set and a *bad* set of values. The branching strategy at each node is to choose one branching variable and reduce its domain to its *good* set in one child node and to its *bad* set in the other child node. The global constraint programming search tree is then explored with limited discrepancy search [34], first exploring the most promising problems, *i.e.*, those where most variables range on their *good* set. Results on pure TSP problems and some variants show that the proposed ranking based on reduced costs is extremely accurate: in almost all cases, even if the relaxation is loose, the optimal solution is found in the sub-problem where all variables range on their *good* set. This sub-problem is explored first, which decreases the computational time needed to reach the optimal solution. However, the tightness of the lower bound is instrumental for completing rapidly the proof of optimality.

In some cases, the linear model can also be tightened as the constraint programming engine progresses toward a solution. Refalo [50] discusses the case of piecewise linear functions. Given is a constraint $y = f(x)$, where f is a piecewise linear function. The convex hull of this constraint is maintained at all nodes in the search tree: cuts are added to the linear formulation when the domain of x is reduced. For small transportation problems with costs y depending on shipped quantities x, such a refinement enabled to save between one and two orders of magnitude in CPU time.

Transformations of non-linear constraints, in particular disjunctive constraints, into linear constraints can also be considered [53, 49]. In general, this leads to the introduction of additional binary variables in the linear model. Hoist scheduling problem results provided in [52] show that the resulting cooperative algorithm is more robust than pure constraint programming or mixed integer programming.

Other forms of **decomposition** also appear in the literature.

The first example is branch-and-price, where the sub-problem is solved with constraint programming (*cf.* Chapter **??** for an in-depth discussion of constraint programming based branch-and-price). This allows to combine the power of branch-and-price to optimize very big problems and the flexibility of constraint programming to take into account side constraints. [54] applies this cooperation technique to the vehicle routing problem with time windows. [63] applies it to crew scheduling and crew rostering problems for a brazilian urban transit bus company: the hybrid method is shown on multiple and various real world instances to perform better than both integer programming and constraint programming approaches used alone.

The second example is Lagrangean relaxation (*cf.* Section 2 for an introduction). A typical example of this **decomposition** scheme is [12]. The Traveling Tournament Problem is a variant of the sports league tournament: one has to organize a set of matches between all teams of a sports league in a given number of weeks, such that each team plays each other team twice, once at home and once away, the sequence of meetings for each team abiding some constraints (for example no more than 3 consecutive home games are allowed). The objective is to minimize the sum of the distance travelled by all teams. The coupling constraints in this problem are that team A must be at home when team B visits it and that no more than one visitor stays at each team's home simultaneously. Once these coupling constraints have been dualized, each sub-problem dealing with the sequence of games of one team can be solved independently with a dedicated TSP constraint-based solver. The computation of this Lagrangean lower bound is embedded in a global constraint that is used in the main constraint programming model driving the tree search.

Sellmann and Fahle [56] give a general framework to solve linear problems such that the constraints can be divided in two families A and B, for which an efficient propagation algorithm exists: respectively $Prop(A)$ and $Prop(B)$. However, combining the two efficient propagation algorithms in a straightforward manner may not be efficient because it is likely that tight bounds on the objective are not obtained by taking only a subset of the restrictions into account. Therefore, Sellman and Fahle propose the following **decomposition** scheme: dualize the A constraints family and use $Prop(B)$ for propagation in each corresponding Lagrangean sub-problem, then apply $Prop(A)$ on the Lagrangean sub-problem obtained when dualizing the B constraints family. This scheme can be generalized to more than two families of constraints. It is applied to the Automatic Recording Problem that contains a knapsack constraint and a maximum weighted stable set constraint on an interval graph. The numerical results show the coupling method significantly decreases computation time and the number of choice points. In both examples [12, 56], the tight bounds on the objective allow to remove some values from the variables domains. Hence Lagrangean relaxation enables constraint programming to be applied more successfully to optimization problems.

The third example is Benders decomposition [10] which is often understood as the “dual” of column generation: instead of generating columns iteratively, the set of columns is given but the rows are generated iteratively. Whereas Lagrangean relaxation is particularly helpful in presence of coupling constraints, Benders decomposition is particularly helpful in presence of coupling variables. The principle is to solve a relaxed master problem to fix the values of the coupling variables. The rest of the variables are then assigned in k sub-problems (that can now be solved independently because the coupling variables are fixed), and each solution to the sub-problems generates a Benders cut that is added to the relaxed master problem. Both the master problem and the sub-problems are usually solved with linear programming, but each can be solved with constraint programming. For example, [58] presents Branch-and-Check, a framework that encompasses Benders decomposition and allows for the sub-problems to be modeled and solved with constraint programming. [11] presents a Benders decomposition applied to a workforce scheduling problem where the sub-problem is solved as usual with linear programming, but where the master problem is solved with constraint programming based on a flow global constraint. This cooperation allows to handle a wide variety of additional constraints thanks to the flexibility of constraint programming. [27] presents an implementation of hybrid Benders decomposition in ECLiPSe, where the master problem is also solved by constraint programming. With this implementation, the programmer need only specify which variables belong to which sub-problems, the dual form of any sub-problem is automatically derived, and the Benders decomposition is extracted automatically. Such implementations are very helpful to promote hybrid algorithms because they permit to build prototypes more quickly and they allow specialists of only one component of the cooperative solver to design hybrid algorithms.

4. Operations Research Algorithms and Local Search

As in the case of constraint programming and mixed integer programming, efficient operations research algorithms can be used as slaves of local search masters. In a local search context, a specific algorithm can help in two ways: either to explore a given neighborhood or to select the neighborhood to be explored.

For example, Adams, Balas, and Zawack [2] applied a procedure referred to as “shifting bottleneck” (later generalized as “shuffle” [3] and “large neighborhood search” [57]) to the job-shop scheduling problem. The basic idea behind large neighborhood search is straightforward. Given a solution, a neighborhood is constructed by relaxing some, but not all, of the decisions in the solution. For example, one can keep the values of some of the variables and relax the others, *i.e.*, allow these variables to take new values, provided that all the constraints of the problem are satisfied. A new solution is then searched for in this neighborhood, using whatever optimization technique is the most appropriate. The decisions to be relaxed at each step can be chosen randomly or according to some given patterns. In the particular case of the shifting bottleneck procedure, a resource (the current “bottleneck”) is chosen. All the ordering decisions determining the sequencing of the activities on the resources are kept, except those that concern the bottleneck. The resulting neighborhood is explored with the help of a specific sequencing algorithm described in [14], which is not polynomial in the worst case, yet appears to be efficient in almost all cases. The shifting bottleneck procedure is no longer state-of-the-art on the job-shop scheduling problem, but constraint programming forms of large neighborhood search are still used extensively in this domain, as reported for example in [18, 45, 21].

In the best case, the use of an operations research algorithm can also lead to the adoption of a simpler representation in the local search space, dealing only with some of the problem variables, while the operations research algorithm is able to optimally fill in the details and compute the exact cost of a solution. For example, many applications of local search to scheduling problems assume that, given a set of sequencing decisions, a longest path algorithm will be used to compute the exact start and end times of each activity. The local search algorithm explores the space of possible sequences, while the longest path algorithm provides the optimal start and end times for a given sequence, thereby reducing an apparently large neighborhood to its locally optimal point.

Operations research algorithms can also be used to select composite moves. For example, for a vehicle routing problem, large neighborhood search consists in iterating composite moves: at each step, several customers are moved from their positions in their current routes to other positions in the same route or to different routes. Xu and Kelly [62] model ejections and insertions of customers from routes as a network flow problem. The objective function considers both an approximation of the distance changes implied by the moves and a penalty mechanism to heuristically ensure the feasibility of moves, for example with respect to the capacity constraints. The constraints represent the flow balance constraints: if a customer is removed from its current route, it must be assigned to another route. This network flow model can be solved with a linear programming solver or even better with network algorithms in order to select a feasible and cost-improving composite move at each step of the large neighborhood search.

5. Mixed Integer Programming and Local Search

For mixed integer programs, two aspects of the problem are important: tightening the continuous relaxation by adding cuts in order to increase the lower bound (when minimizing), finding good integer solutions in order to decrease the upper bound. The traditional way of obtaining integer solutions is to explore a tree: at each node, a variable that is fractional in the current relaxation is chosen and several child nodes are created by adding to each a new constraint to partition the search space while cutting off the current relaxation. For example, if integer variable x has fractional value f in the relaxation, two child nodes are created: one which the new constraint $x \leq \lfloor f \rfloor$, the other with the new constraint $x \geq \lceil f \rceil$. Thus, as the depth in the tree increases, the number of variables with fractional values is expected to decrease, and it is hoped that at some node, the relaxation will have an integer solution. However, if the relaxation is a very poor approximation of the convex hull of the integer solutions, and if the branching decisions add variable bounds that fail to improve rapidly this approximation, and if, as a consequence, the relaxation is not a good guide to decide on which part of the tree to concentrate, it is very difficult to find nodes where the relaxation solution is spontaneously integer. Using additional algorithms such as local search to find integer solutions might then be beneficial. This is why heuristics are often combined with branch-and-bound to find quickly integer solutions. There are few domain independent heuristics and there mainly exist domain specific heuristics that take advantage of the high-level problem structure. Two kinds of heuristics exist:

- Rounding heuristics start from an integer-infeasible solution, typically the solution of the continuous relaxation, and try to infer from it an integer-feasible solution. In other words, they perform local search around the continuous relaxation. The most studied heuristic of this kind is the Pivot & Complement heuristic [4]; other heuristics such as those presented in [38, 1] use Pivot & Complement as a subroutine inside a metaheuristic. Another common strategy is what is described in [13, 29] as hard variable fixing or diving: fix some variables to integer values, infer some new bounds on the remaining variables due to these fixings, solve the LP relaxation (or not, for a less expensive variant of the heuristic), and repeat until all variables have been fixed. Randomized rounding [48] is another rounding heuristic: consider a pure 0-1 integer program; if v_i is the value for variable x_i in the continuous relaxation, then x_i is rounded up to 1 with probability v_i and rounded down to 0 with probability $1 - v_i$. Thus, for each constraint, the expected value of the right hand side is equal to its value for the relaxation, which is compatible with the left hand side, and the expected value of the rounded solution is the value of the continuous relaxation. Of course, it is only on some models (for example, set covering) that randomized rounding leads to feasible solutions, where all constraints are valid simultaneously and not only in expectation. On some models, the rounded solution can be theoretically proved to be feasible and to have a cost within a certain range of the optimal solution. When the rounded solution is not feasible, repair heuristics can be used: Walser [61] transforms the MIP problem into an over-constrained problem and uses an algorithm similar to Walksat to reduce the number of violated constraints. An example of domain-specific rounding heuristic appears in [22]: the relaxation is used to order jobs in a scheduling problem, which produces approximation algorithms with performance guarantees.

- Local improvement heuristics start from one or several integer solutions and try to find a better integer solution. For example, [23] uses a specific heuristic for the vehicle routing problem with time windows based on large neighborhood search to improve on the integer solutions found by mixed integer programming (in fact a column generation approach). In this case, cooperation is even tighter: the local search heuristic not only provides integer solutions, but it also yields new columns for the column generation master problem, because it is not restricted to combining existing columns to generate integer solutions.

Such cooperation between local search and mixed integer programming clearly corresponds to the **multiple search** cooperative scheme described in the introduction. Local search algorithms succeed in finding quickly integer solutions, which obviously helps the MIP solver. But the MIP solver acts in turn as a good diversification scheme for the local search algorithm. One of the problems of local search algorithms is indeed that they tend to get stuck in local optima, so a good diversification scheme is needed to explore a different part of the solution space. Often, degrading the objective value is allowed during diversification.

In this cooperation scheme however, the MIP solver diversifies the local search and always finds a new solution with better cost than the local optima, because its upper bound is always updated with the cost of the best current integer solution. Besides, this cooperative scheme combines the power of an incomplete search method (local search) for finding solutions, and the power of a complete method (mixed integer programming) for proving the optimality of those solutions.

Two recent heuristics obey to an entirely different paradigm: a **decomposition** scheme where the initial master is the MIP solver, but where local search is called regularly and acts as a secondary master, defining neighborhoods around solutions found by the MIP solver, which are explored by a recursive MIP slave. This paradigm is based on assumptions on the solution space, on which local search is also based and which can be summarized as the following "locality principle": there exist good solutions in the neighborhood of a good solution. Local branching [29] (see also [60] for a similar approach applied to the multidimensional knapsack problem) explores the neighborhood of the current integer solution, defining the recursive MIP by adding a constraint to the global MIP stating that all subsequent solutions in the recurvive MIP must be within Hamming distance k of the current integer solution, that is there will be at most k variables that have different values in the current solution and in each subsequent solution. Relaxation Induced Neighborhood Search (RINS) [25] explores both the neighborhood of the current integer solution and the neighborhood of the continuous relaxation, defining the recursive MIP on the variables that have different values in the current integer solution and in the continuous relaxation, fixing the other variables to their common value. [25] presents an in-depth comparison of local branching and RINS on multiple hard difficult MIP models. It appears computationally that RINS performs consistently better than local branching.

6. Constraint Programming and Local Search

As mixed integer programming, constraint programming can be used in a **decomposition** scheme as a slave to a local search algorithm. The local search master can benefit from constraint programming for at least two reasons:

- Constraint propagation can be used to determine characteristics that are shared by all the solutions or, similarly, by all the solutions that respect a given set of decisions and are better than (or close to) the best solution found so far. The solution space to be explored by local search can then be restricted to the solutions that meet these characteristics — provided this does not lead to a dramatic disconnection of the solution space. As mentioned in [59], this has two advantages: the time needed to explore a neighborhood is reduced, and neighborhoods with no good solutions are not explored. Constraint programming can be used as a preprocessing to local search, as in [59], or as a technique to explore the neighborhood, as in [46]. In [59], a pure tabu search algorithm and a combination of constraint propagation and tabu search are applied to thirty instances of a frequency assignment problem. Results show that the pure tabu search algorithm provided most of the best results, but also some of the worst. The combination was altogether more robust. In [46], constraint programming is used to find the best non-tabu neighbor of a given solution to a traveling salesman problem with time windows. In the reported results, the average ratio between the number of backtracks and the size of the neighborhood is 9% (29% for the worst instance), showing that many neighbors are indeed discarded by constraint programming.

- Using constraint programming techniques as part of a local search algorithm may also enable one to keep as much as possible of the local search efficiency enjoyed on the "pure" problems and gain in flexibility by allowing side constraints to be taken into account. For example, Kilby, Prosser, and Shaw report results obtained on vehicle routing problems with side constraints, showing that the use of constraint programming to construct initial solutions and explore neighborhoods enhances robustness with respect to the side constraints [37].

Globally, such approaches are promising whenever local search operators, possibly large neighborhood search operators, provide a good basis for the exploration of the search space, and either side constraints or effective constraint propagation algorithms can be used to prune the search space.

The **multiple search** scheme can also be used to combine constraint programming and local search, taken as alternative ways to explore the search space. For example, Caseau and Laburthe [18] describe an algorithm for the job-shop scheduling problem which combines constraint programming and local search. The overall algorithm finds an approximate solution to start with, makes local changes and repairs on it to quickly decrease the makespan (*i.e.*, the time at which all activities are completed) and, finally, performs a constraint-based exhaustive search for decreasing makespans. Given a schedule S, a critical path is defined as a sequence of activities A_1, ..., A_n such that for all i in $\{1 \dots n-1\}$, A_i

precedes A_{i+1} and Σ *duration*(A_i) = *makespan*(S). Two types of local moves are considered:

- "Repair" moves swap two activities scheduled on the same machine to shrink or reduce the number of critical paths.
- "Shuffle" moves [3] keep part of the solution and search through the rest of the solution space to complete it. Each shuffle move is implemented as a constraint-based search algorithm with a limited number of backtracks (typically 10, progressively increased to 100 or 1000), under the constraint that the makespan of the solution must be improved (with an improvement step of a given δ, typically 1% of the makespan, progressively decreased to one time unit).

Hence, in this example, both generic schemes are applied: local search is followed by exhaustive constraint programming tree search, and part of the local search relies on constraint programming to limit the exploration of the neighborhood to relevant problem solutions. Excellent computational results have been obtained with this approach [18, 21] as well as with other constraint-based implementations of shuffle moves, as reported in [6, 45].

In the same spirit, the best algorithm we are aware of for the preemptive job-shop scheduling problem [39] relies on the combination of:

- a strong constraint propagation algorithm (edge-finding);
- a local optimization operator called "Jackson derivation". Given a schedule S, its Jackson derivation $J(S)$ is obtained by (i) defining the due-date of an activity A in a job J as the start time in S of the successor of A along job J and (ii) rescheduling all activities by giving the highest priority to activities with the smallest due-dates. This operator and the symmetric operator based on the end time of the predecessor of each activity A are systematically applied each time a new solution is found.
- limited discrepancy search [34] around the best schedule found so far. Limited discrepancy search is an alternative to depth-first search, which relies on the intuition that heuristics make few mistakes through the search tree. Thus, considering the path from the root node of the tree to the first solution found by a depth-first search algorithm, there should be few "wrong turns" (*i.*e., few nodes which were not immediately selected by the heuristic). The basic idea is to restrict the search to paths that do not diverge more than w times from the choices recommended by the heuristic. Each time this limited search fails to improve on the best current

schedule, w is incremented and the process is iterated, until either a better solution is found or it is proven that there is no better solution. It is easy to prove that when w gets large enough, limited discrepancy search is complete. Yet it can be seen as a form of local search around the recommendation of the heuristic.

On ten well-known problem instances, each with 100 activities, experimental results show that each of the three techniques mentioned above brings improvements in efficiency, the average deviation to optimal solutions after 10 minutes of CPU time falling from 13.72% when none of these techniques is used to 0.23% when they are all employed.

Focacci and Shaw [32] provide an example in which local search is used as a slave of a constraint programming master, as an alternative to dominance rules to prune the constraint programming search tree. For a minimization problem, dominance rules state that if a partial solution s can be extended to a complete solution whose cost is less than the optimal extension of partial solution s', then all extensions of s' can be pruned. This enables to drastically reduce the search space. However, pruning s' could be counter-productive if the sub-tree rooted in s has not been explored yet. Indeed, by exploring the sub-tree rooted in s' immediately, one may obtain quickly an extension of s' better than the best solution known so far and immediately tighten the remainder of the search. As a result, the systematic use of dominance rules may make it difficult to find good solutions in limited CPU time. Therefore, Focacci and Shaw propose to prune only the partial solutions that cannot be extended to solutions better than the best known solution. In this case, s' is handled as a no-good, *i.e.*, a partial solution that cannot be extended to a complete solution, because it would not be legal with respect to the current upper bound on cost. Any algorithm can be used to prove that a partial solution is dominated: an incomplete local search algorithm is used in [32] (a nearest neighbor heuristic and a relocate operator for the traveling salesman problem). When this algorithm discovers that the sub-problem defined by the partial solution under consideration is a relaxation of the sub-problem defined by one of the stored no-goods, the partial solution can be pruned. On numerous symmetric and asymmetric TSP instances, the hybrid method highly reduces the CPU time and the number of fails, compared to a pure constraint programming approach and to the standard application of dominance rules.

Conclusion

In this chapter we gave an overview of hybrid algorithms. We tried to abstract from the examples two generic cooperation schemes : **de-**

composition and **multiple search**. It appears however that there is still a lot of work to be done to create a comprehensive taxonomy of cooperation techniques and to understand on which kind of problems which kind of cooperation has more chances to be efficient and robust.

The reader is referred to the next chapters for a more extensive discussion of some of the cooperation techniques we mentioned: Chapter **??** for a global framework to unify constraint programming and integer programming; Chapter **??** for the integration of operations research algorithms in constraint programming to develop global constraints and filtering algorithms; Chapter **??** for integrating constraint programming and mixed integer programming; Chapter **??** for constraint programming based branch-and-price; Chapter **??** for more details on integrating constraint programming and local search; and Chapter **??** for randomized algorithms and algorithm portfolio design — another example of **multiple search**.

References

[1] Ronny Aboudi and Kurt Jörnsten. Tabu search for general zero-one integer programs using the pivot and complement heuristic. *ORSA Journal on Computing*, 6(1):82–93, 1994.

[2] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job-shop scheduling. *Management Science*, 34(3):391–401, 1988.

[3] David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

[4] E. Balas and C. Martin. Pivot and complement — a heuristic for 0-1 programming. *Management Science*, 26(1):224–234, 1980.

[5] Philippe Baptiste and Claude Le Pape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[6] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

[7] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92:305–333, 1999.

[8] Philippe Baptiste, Claude Le Pape, and Laurent Péridy. Global constraints for partial CSPs: A case study of resource and due-date constraints. In *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, 1998.

[9] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W.P. Savelsbergh, and Pamela H. Vance. Branch-and-price:

Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[10] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[11] Thierry Benoist, Étienne Gaudin, and Benoît Rottembourg. Constraint programming contribution to Benders decomposition: A case study. In *Proceedings of the Eigth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, pages 603–617, 2002.

[12] Thierry Benoist, François Laburthe, and Benoît Rottembourg. Lagrange relaxation and constraint programming collabortaive schemes for traveling tournament problems. In *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'01)*, pages 15–26, 2001.

[13] Robert E. Bixby, Mary Fenelon, Zonghao Gu, Edward Rothberg, and Roland Wunderling. *MIP: Theory and Practice — Closing the Gap*, pages 19–49. Kluwer Academic Publishers, 2000.

[14] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47, 1982.

[15] Jacques Carlier and Éric Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[16] Jacques Carlier and Éric Pinson. A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[17] Yves Caseau and François Laburthe. Improved CLP scheduling with task intervals. In *Proceedings of the Eleventh International Conference on Logic Programming*, 1994.

[18] Yves Caseau and François Laburthe. Disjunctive scheduling with task intervals. Technical report, École Normale Supérieure, 1995.

[19] Yves Caseau and François Laburthe. Cumulative scheduling with task intervals. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, 1996.

[20] Yves Caseau and François Laburthe. Solving various weighted matching problems with constraints. *Constraints*, 5(2):141–160, 2000.

[21] Yves Caseau, François Laburthe, Claude Le Pape, and Benoît Rottembourg. Combining local and global search in a constraint programming environment. *Knowledge Engineering Review*, 16(1):41–68, 2001.

[22] C.C.B. Cavalcante, C. Caravlho de Souza, M.W.P. Savelsbergh, Y. Wang, and L.A. Wolsey. Scheduling projects with labor constraints. *Discrete Applied Mathematics*, 112:27–52, 2001.

[23] Alain Chabrier, Émilie Danna, and Claude Le Pape. Coopération entre génération de colonnes avec tournées sans cycle et recherche locale appliquée au routage de véhicules. In *Huitièmes Journées Nationales sur la résolution de Problèmes NP-Complets (JNPC'2002)*, 2002.

[24] Alain Colmerauer. An introduction to PROLOG III. *Communications of the ACM*, 33(7):69–90, 1990.

[25] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. Technical report, ILOG, 2003.

[26] Martin Desrochers. An algorithm for the shortest path problem with resource constraints. Technical Report G-88-27, Les cahiers du GERAD, september 1988.

[27] Andrew Eremin and Mark Wallace. Hybrid Benders decomposition algorithms in constraint logic programming. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'2001)*, 2001.

[28] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. submitted to Networks, 2001.

[29] Matteo Fischetti and Andrea Lodi. Local branching. In *Proceedings of the Integer Programming Conference in honor of Egon Balas*, 2002.

[30] Filippo Focacci, Philippe Laborie, and Wim Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000.

[31] Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming*, 1999.

[32] Filippo Focacci and Paul Shaw. Pruning sub-optimal search branches using local search. In *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 181–189, 2002.

[33] Bernard Gendron and Teodor G. Crainic. Relaxations for multicommodity capacitated network design problems. Technical Report CRT-965, Centre de recherche sur les transports, Université de Montréal, 1994.

[34] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[35] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[36] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, 1987.

[37] Philip Kilby, Patrick Prosser, and Paul Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints*, 5(4):389–414, 2000.

[38] Arne Løkketangen and David L. Woodruff. Integrating pivot based search with branch and bound for binary MIP's. *Control and Cybernetics, Special issue on Tabu Search*, 29(3):741–760, 2001.

[39] Claude Le Pape and Philippe Baptiste. Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics*, 5(3):305–325, 1999.

[40] Claude Le Pape, Laurent Perron, Jean-Charles Régin, and Paul Shaw. Robust and parallel solving of a network design problem. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, 2002.

[41] Michela Milano, Greger Ottosson, Philippe Refalo, and Erlendur S. Thorsteinsson. The role of integer programming techniques in constraint programming's global constraints. *INFORMS Journal on Computing*, to appear.

[42] Michela Milano and Willem J. van Hoeve. Reduced cost-based ranking for generating promosing subproblems. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, pages 1–16, 2002.

[43] W. P. M. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach.* PhD thesis, Eindhoven University of Technology, 1994.

[44] W. P. M. Nuijten and E. H. L. Aarts. Constraint satisfaction for multiple capacitated job-shop scheduling. In *Proceedings of the Eleventh European Conference on Artificial Intelligence*, 1994.

[45] Wim Nuijten and Claude Le Pape. Constraint-based job-shop scheduling with ILOG Scheduler. *Journal of Heuristics*, 3(4):271–286, 1998.

[46] Gilles Pesant and Michel Gendreau. A view of local search in constraint programming. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, 1996.

[47] Éric Pinson. *Le problème de job-shop.* PhD thesis, University Paris VI, 1988.

[48] P. Raghavan and C.D. Thompson. Randomized rounding : A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[49] P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, pages 369–383, Singapore, September 2000. Springer-Verlag.

[50] Philippe Refalo. Tight cooperation and its application in piecewise linear optimization. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming*, 1999.

[51] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[52] Robert Rodosek and Mark Wallace. A generic model and hybrid algorithm for hoist scheduling problems. In *Proceedings of the Fourth*

International Conference on Principles and Practice of Constraint Programming, 1998.

[53] Robert Rodosek, Mark Wallace, and Mozafar T. Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86:63–87, 1999.

[54] Louis-Martin Rousseau, Michel Gendreau, and Gilles Pesant. Solving small VRPTWs with constraint programming based column generation. In *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 333–344, 2002.

[55] Hani El Sakkout and Mark Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

[56] Meinolf Sellmann and Torsten Fahle. Constraint programming based lagrangian relaxation for the automatic recording problem. *Annals of Operations Research*, 118:17–33, 2003.

[57] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP'98)*, pages 417–431, 1998.

[58] Erlendur S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'2001)*, 2001.

[59] Michel Vasquez. Arc-consistency and tabu search for the frequency assignment problem with polarization. In *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 359–372, 2002.

[60] Michel Vasquez and Jin-Kao Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJ-CAI'2001)*, pages 328–333, 2001.

[61] Joachim Paul Walser. *Domain-Independent Local Search for Linear Integer Optimization.* PhD thesis, Technischen Fakultät der Universität des Saarlandes, 1998.

[62] Jeifeng Xu and James P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30(4):379–393, 1996.

[63] T.H. Yunes, A.V. Moura, and C.C. de Souza. A hybrid approach for solving large scale crew scheduling problems. In *Practical Aspects of Declarative Languages*, pages 293–307, 2000.