Math. Program., Ser. A 102: 71–90 (2005)

Digital Object Identifier (DOI) 10.1007/s10107-004-0518-7

Emilie Danna · Edward Rothberg · Claude Le Pape

Exploring relaxation induced neighborhoods to improve MIP solutions

Received: April 24, 2003 / Accepted: March 16, 2004
Published online: 21 May 2004 –

Abstract. Given a feasible solution to a Mixed Integer Programming (MIP) model, a natural question is whether that solution can be improved using local search techniques. Local search has been applied very successfully in a variety of other combinatorial optimization domains. Unfortunately, local search relies extensively on the notion of a solution neighborhood, and this neighborhood is almost always tailored to the structure of the particular problem being solved. A MIP model typically conveys little information about the underlying problem structure. This paper considers two new approaches to exploring interesting, domain-independent neighborhoods in MIP. The more effective of the two, which we call Relaxation Induced Neighborhood Search (RINS), constructs a promising neighborhood using information contained in the continuous relaxation of the MIP model. Neighborhood exploration is then formulated as a MIP model itself and solved recursively. The second, which we call guided dives, is a simple modification of the MIP tree traversal order. Loosely speaking, it guides the search towards nodes that are close neighbors of the best known feasible solution. Extensive computational experiments on very difficult MIP models show that both approaches outperform default CPLEX MIP and a previously described approach for exploring MIP neighborhoods (local branching) with respect to several different metrics. The metrics we consider are quality of the best integer solution produced within a time limit, ability to improve a given integer solution (of both good and poor quality), and time required to diversify the search in order to find a new solution.

Key words. Mixed integer programming – Heuristics – Local search – Hybrid algorithms

1. Introduction

1.1. Mixed integer programming

Mixed Integer Programming (MIP) is one of the most important techniques for solving complex optimization problems. A MIP problem is defined by a set of variables (x), a set of linear constraints on these variables ($Ax = b$), a set of integrality constraints specifying that some of the variables must assume integer values, and a linear objective function of the variables to optimize ($min\ c'x$; we assume minimization throughout this paper). MIP provides a powerful framework for modeling and solving a wide variety of optimization problems.

E. Danna: ILOG S.A. 9, rue de Verdun, F-94253 Gentilly Cédex e-mail: edanna@ilog.fr and Laboratoire d'Informatique d'Avignon, CRNS - FRE 2487, 339, chemin des Meinajariès. Agroparc, BP 1228, 84911 Avignon Cédex 9, France

E. Rothberg: ILOG, Inc., 1080 Linda Vista Avenue, Mountain View, CA 94043, USA
e-mail: rothberg@ilog.com

C.L. Pape: ILOG S.A., 9, rue de Verdun, 94253 Gentilly Cédex, e-mail: clepape@ilog.fr

Mathematics Subject Classification (2000): 20E28, 20G40, 20C20

MIP problems are typically solved using a *branch-and-bound* or *branch-and-cut* algorithm. This approach explores a tree of continuous relaxations of the original MIP model, where at each node in the tree the space of possible solutions is split into two disjoint sub-spaces by imposing complementary bound constraints on an integer variable. This technique is particularly effective when the continuous relaxation of the problem is a good approximation of the convex hull of the feasible solutions (at least around the optimal solution), or alternatively when the relaxation can be tightened by adding cutting planes to have this property. Much work has been done to make commercial implementations robust with respect to problem size and numerical characteristics. Still, some MIP models remain very hard to solve to optimality. In such cases, and unless another exact technique such as constraint programming applies well to the problem, the user must settle for a feasible solution of good quality. Alas, even a good feasible solution might be difficult to obtain in the desired response time for some models.

A common reaction of an Operations Research practitioner confronted with such a problem is to consider the use of other techniques, such as local search. Indeed, various forms of local search, operating either on individual solutions (*e.g.*, simulated annealing [33], tabu search [18]) or on populations of solutions (*e.g.*, genetic algorithms [28], scatter search, and path relinking [19]), are known to provide excellent feasible solutions quite quickly for many problems. This suggests that applying local search concepts (neighborhood, intensification, diversification) might be useful when solving extremely difficult mixed integer programming models.

1.2. Integrating local search

To integrate local search and MIP, three questions must be answered. The most important and also most difficult one is how to define the *neighborhood* of a given solution. A neighborhood is a set of solutions that are close, by some metric, to the given solution. Neighborhoods are almost always built in terms of the high-level structure of the specific problem at hand. For example, a graph partitioning neighborhood might be built by considering swaps of pairs of nodes between the existing partitions. In a generic MIP, this structure would have to be inferred from the constraint matrix, which would present a major challenge. While the difficulty is no doubt clear, the appeal of being able to build a domain-independent "unstructured" neighborhood is also clear: the resulting method could then be applied to any MIP with no other input than the model itself.

An important related issue in local search is how large the neighborhood should be. One strategy that has proven to be quite effective for difficult problems is Large Neighborhood Search (LNS), where large neighborhoods are defined by adding new constraints to the model to fix some explicit or implicit variables to their current values. The remaining problem is then solved on the other variables. LNS has been successfully applied in combination with operations research algorithms [2, 4], constraint programming [7, 10, 12, 31] and recently mixed integer programming [30] — but always relying on the high-level structure of the problem to define the neighborhood (i.e., to choose the variables to fix).

The second important question that must be answered is how to search the neighborhood; that is, how to perform *intensification*. Once a neighborhood has been defined,

depending on its size, it can be explored by complete enumeration, heuristically, or, following the paradigm of large neighborhood search, it can explored with the same complete algorithm (possibly truncated) that is used to solve the global problem. For mixed integer programming, the third option amounts to defining a neighborhood that can be represented as a more constrained MIP model, and exploring this *sub-MIP* with a generic MIP solver.

The final question to be answered is how to perform *diversification*. To be effective, a local search algorithm needs to regularly change the reduced search space on which it focuses. An obvious diversification approach in a MIP context is to consider new incumbents generated in the branch-and-cut process. However, the motivation for this investigation is that there are some models where the MIP solver has difficulty finding new solutions, so this approach may not suffice.

A recently proposed approach called *local branching* [17] provides one set of answers to these three questions. This paper introduces two new approaches. We call the two approaches *Relaxation Induced Neighborhood Search (RINS)* and *guided dives*. As will be discussed shortly, each takes a somewhat different approach to answering these questions.

1.3. Outline of the paper

The organization of the remainder of the paper is as follows. Section 2 presents the methods considered in this paper. We present RINS and guided dives, and also discuss local branching [17]. Section 3 presents the benchmark models used to compare RINS, guided dives, and the alternatives. Section 4 compares the methods in three different contexts. First, we look at how solution quality evolves over time when each of the methods is started from scratch. Next, we look at the ability of each method to improve good quality and poor quality initial solutions. Finally, we look at the ability of each method to diversify the search. That is, we consider the ability of each approach to escape from an unproductive neighborhood and generate a new incumbent solution. The results show that RINS and guided dives are quite effective in all three contexts. Finally, Section 5 presents our conclusions.

2. Methods

We now describe the methods considered in this paper in more detail. As noted, all are based on the notion that a small neighborhood of the current incumbent is likely to contain better feasible solutions.

2.1. Relaxation induced neighborhood search

When exploring a branch-and-cut tree, two solutions are typically at our disposal. The incumbent is feasible with respect to the integrality constraints but it is not optimal until the last and optimal integer solution has been found. Conversely, the solution of the continuous relaxation at the current node is most often not integral, but its objective value

is always better than that of the incumbent. Thus, the incumbent and the continuous relaxation each achieve one and fail to achieve one of the following conflicting goals: integrality and optimization of the objective value. While some variables clearly take different values in the incumbent and the relaxation, it is important to note that many take the same values. RINS is based on the intuition that the instantation of these variables forms a partial solution that is likely to be extended towards a complete solution that achieves both integrality and a good objective value. Therefore, it focuses attention on those variables that differ in the continuous relaxation and in the incumbent, which are intuitively the ones that merit further attention.

Our RINS algorithm is thus simple. At a node of the global branch-and-cut tree, the following steps are performed:

1. Fix the variables that have the same values in the incumbent and in the current continuous relaxation;
2. Set an objective cutoff based on the objective value of the current incumbent;
3. Solve a sub-MIP on the remaining variables.

Note that the global MIP formulation typically improves during the branch-and-cut process, due to the addition of cutting planes and the discovery of new global bounds on variables. Our sub-MIP takes advantage of this additional information. However, we do not restrict our search to the current MIP sub-tree, so bounds added due to branching are not respected.

Our RINS sub-MIP is also potentially large and difficult to solve, so its exploration must often be truncated. We do so by setting a node limit nl. Any integer solution found in the sub-MIP is by construction also a solution of the global MIP. Hence, when the sub-MIP exploration terminates (because infeasibility or optimality of the sub-MIP has been proved or nl nodes have been explored), the incumbent of the global MIP is updated by the best integer solution found in the sub-MIP (if any), and exploration of the global MIP is resumed. Note that the only effect of exploring a RINS sub-MIP on the global MIP search is a potential new incumbent solution.

Since the continuous relaxation changes from one node in the branch-and-cut tree to the next, RINS obtains automatic neighborhood diversification. While RINS could be invoked at every node in the tree, the neighborhoods induced by the relaxations of consecutive nodes are typically quite similar, so we have found that it is preferable to apply it only every f nodes for some $f \gg 1$. We should mention that it is possible, though unlikely, that even with a large choice of f, multiple MIP nodes may produce the same RINS sub-MIPs, or similarly that a RINS sub-MIP may be identical to a sub-tree of the global MIP tree. We take no steps to avoid this duplicated effort.

The strength of RINS is that it explores a neighborhood both of the incumbent and of the continuous relaxation. Without outside information such as another lower or upper bound provided by a different algorithm, the MIP solver does not know which of the two is closer to the optimal integer solution. In RINS, the incumbent and the continuous relaxation play perfectly symmetrical roles, so if one is of poor quality, the other automatically helps to define a fruitful neighborhood and vice versa. Therefore, on the one hand, RINS may greatly improve incumbents of poor quality because it is guided by the continuous relaxation. On the other hand, RINS is likely to improve robustness when faced with a loose relaxation, since it is also guided by the current incumbent.

Setting RINS in the context of existing local search approaches, RINS can be viewed as a large neighborhood search method that uses the continuous relaxation to fix variables, thereby defining its neighborhoods. To our knowledge, it is the first LNS method — besides random choice, which is not particularly effective — that does not rely on the known high-level structure of the problem.

RINS is also related to an approach called path relinking [19] because, in a certain way, it relinks two solutions that were found in different parts of the global branch-and-cut tree by exploring the solution space *between* them. RINS differs from path relinking in the two following ways. First, instead of following one or at most a few paths between two solutions, it explores the sub-space defined by the intersection of the two solutions. This exploration is achieved by a powerful and complete method that is truncated to control its execution time. Secondly, and more importantly, RINS combines solutions of two related but different problems: the original MIP and its continuous relaxation. To follow the terminology of path relinking, one of the key properties of RINS is that its pool of elite solutions contains infeasible solutions. We believe that extending the pool in this way addresses an important inherent difficulty in the problem, the gap between the relaxation and any known feasible solutions.

2.2. Local branching

Local branching (LB) is a recently proposed strategy [17] for exploring an explicit neighborhood of a MIP solution. Our RINS strategy borrows several ideas from local branching, including the idea of describing a neighborhood as a sub-MIP model and exploring it using the MIP solver.

Local branching is perhaps most easily described using a slightly generalized notion of a Hamming distance. Given two vectors x and x^*, where some subset B of the entries in the vectors are constrained to take values 0 or 1, the Hamming distance between x and x^* is:

$$H(x, x^*) = \sum_{j \in B} |x_j - x_j^*|.$$

Local branching constructs a sub-MIP that considers only a small neighborhood of the current incumbent x^* by introducing the additional (linearized) constraint $H(x, x^*) \leq r$ for some *neighborhood radius* parameter r. If this neighborhood is indeed rich with better solutions, then those solutions are likely to be found while exploring the sub-MIP search tree. As in RINS, the local branching sub-MIP includes all cutting planes and tighter global variable bounds found during the exploration of the global branch-and-cut tree. It also ignores variable bounds imposed by branching.

Note that the non-binary values in the vectors x or x^* make no contribution to the metric H used by local branching. Hence the local branching constraint can be easily linearized as $\sum_{j \in B \cap \{x_j^*=1\}} (1-x_j) + \sum_{j \in B \cap \{x_j^*=0\}} x_j \leq r$. The metric H could be extended in an obvious way to handle general integer variables, but linearizing the resulting local branching constraint would require the introduction of additional variables. The local branching constraints used in [17] and in our computational testing involve only the binary variables.

A few implementation details are important for obtaining an effective strategy. For example, the local branching sub-MIP can still be quite difficult to solve to optimality, so a truncation scheme is required. As in RINS, our implementation uses a node limit *nl* for every local branching sub-MIP. Unlike RINS, however, we follow Fischetti and Lodi's approach and take further action if a sub-MIP hits the node limit but has not found a new solution. Specifically, we divide the radius by 2 and solve a new sub-MIP. This continues until the radius is less than 5, or until the sub-MIP produces a new solution. After this successful sub-MIP has reached the node limit, the radius is reset to its original value and the process is begun anew on the new solution. Thus, an extended sequence of local branching solutions can be produced. Exploration of the global MIP is resumed when a sub-MIP with $r \leq 5$ fails to produce an improved solution within *nl* nodes. We use different termination criteria for RINS and local branching because it is important for local branching to fully exploit each opportunity to improve an incumbent, as local branching can only be called when a new incumbent is found in the global MIP tree.

Another important implementation detail is the issue of when to apply LB. We call local branching at any node where a new incumbent is found, building the local branching neighborhood around the best feasible solution found at that node. MIP heuristics as implemented in CPLEX often find multiple solutions at the same node (particularly at the root node), but we found it to be unproductive to explore any but the best.

One major difference between our implementation and that of Fischetti and Lodi is in how the global MIP search is structured. We treat local branching solely as a heuristic for finding improved integer solutions. Fischetti and Lodi also treat it as a branching meta-strategy. Specifically, a local branching sub-MIP that includes the constraint $H(x, x^*) \leq r$ has a complementary sub-MIP that includes the reverse constraint $H(x, x^*) \geq r + 1$. In cases where the first sub-MIP is explored completely (producing an optimal solution or a proof that no solution exists), then attention from that point on can be restricted to the reverse sub-MIP. The main advantage of their approach is that it avoids reexploration of the given neighborhood of x^*. An important disadvantage is that the cost of each node increases as these dense reverse neighborhood constraints accumulate. In our experience, the reduced node processing throughput caused by these dense constraints outweighs the benefit of avoiding redundant node exploration.

The second major difference in our implementation is the diversification strategy. Our sole means of diversification for local branching is the use of solutions found during the global MIP tree exploration. Fischetti and Lodi don't actually make use of solutions found in the standard MIP tree search, but instead generate known sub-optimal feasible solutions and explore local branching neighborhoods associated with these. Our experimental results suggest that using the global MIP search for diversification is the more effective approach. We show later in this paper that RINS and guided dives are yet more effective.

An obvious question at this point is whether we have preserved the desirable properties of local branching after our modifications. We compared the performance of our implementation against that of Fischetti and Lodi, both built on top of the same version of CPLEX (8.1), and found that ours produced consistently better results. Mean optimality gaps for our implementation on our test set were roughly 20%, while theirs were roughly 30%. Our implementation also outperformed the version Fischetti and Lodi used in their paper [17], which was built on top of an earlier version of CPLEX (7.0).

Another obvious question is why we found it necessary to modify local branching at all. RINS and guided dives (see next section) use a standard MIP tree, so it seemed natural to compare them with an implementation of local branching that also uses a standard MIP tree. In particular, placing RINS and local branching within the identical MIP search tree allows us to focus our attention exclusively on the question of whether hard fixing (RINS) or soft fixing (local branching) produces more fruitful sub-MIPs.

All computational results in this paper come from our implementation of local branching. Indeed, unless otherwise noted, the term "local branching" refers to our implementation.

2.3. Guided dives

In contrast to the other approaches considered in this paper, our next strategy does not consider neighborhoods explicitly, nor does it build sub-MIP models to be solved recursively. Instead, it is a minor modification of the default MIP tree traversal strategy. Consider two important decisions that are made at each node in a MIP search tree when diving to a leaf node. The first is the choice of a variable on which to branch, and the second is the choice of which of the resulting child nodes to explore first. In our *guided dives* strategy, we base the second choice on the value of the branching variable in the current incumbent solution. We choose the child in which the binary branching variable is fixed to the value that it takes in the incumbent. (For a general integer branching variable, we choose the child in which the branching variable is still allowed to take the value it takes in the incumbent.)

This guided dives strategy is trivial to implement, requiring only a query of the value of the branching variable in the current incumbent at each node. Despite its simplicity, the results in the next section show that this approach is actually quite effective at improving the solutions found by CPLEX for the models in our test set.

2.4. Related work

Before proceeding to computational results, let us first consider how the methods just described relate to existing work. Perhaps the closest existing approach to RINS (and local branching) is the TRIP system for crew pairing optimization [3]. This system improved an existing set partitioning solution by repeatedly choosing a random set of binary variables whose values in the current feasible solution are one, temporarily fixing those variables to one, and then formulating and solving a sub-MIP on the remaining variables (typically using column generation to augment the set of remaining variables before solving the MIP). In a set partitioning framework, fixing a variable to one has the effect of fixing a large number of other variables to zero (all those variables whose columns share a non-zero with the column associated with the fixed variable), so the resulting sub-MIP is typically much easier than the original problem. The TRIP approach is conceivably quite general, but as far as we know, it has only been considered in the context of set partitioning.

To the best of our knowledge, the idea of using a known feasible solution to derive improved solutions has not been used often in MIP or constraint programming. Our

introduction already mentioned a few large neighborhood search approaches that used problem structure to create and explore sub-models that correspond to neighborhoods of the best known feasible solution. An additional relevant approach is a problem-specific, branch-and-price heuristic for the vehicle routing problem with time windows that performs local search around the incumbent [14]. Additionally, existing contraint programming approaches for scheduling problems have used the incumbent to guide the search [25] or as a preferred choice in the context of rescheduling to compute a new solution with a minimal number of changes [23, 32].

Existing MIP heuristics most often try to transform a continuous relaxation solution into an integer-feasible solution. In other words, they perform local search around the continuous relaxation solution. General MIP heuristics include [5, 16, 20–22]. Some other heuristics [1, 6, 26, 29] try to achieve integrality by performing simplex pivots. A different class of heuristics is known as hard variable fixing or diving heuristics [9]. They start from the LP solution, fix some variables to integer values, infer 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. Note that diving heuristics typically fix all integer variables that are integral in the LP relaxation to their relaxation values, whereas RINS only fixes those whose values agree with the corresponding incumbent values. This often allows RINS to find solutions that a diving heuristic could not find.

In constraint programming, the heuristics closest to our work are repair heuristics that aim to transform an infeasible assignement of values to variables into a feasible solution. This initial assignement may be complete, as for example in the min-conflicts heuristic [27], or incomplete, as for example in the decision-repair heuristic [24]. Another related constraint programming heuristic is limited backtracking [11].

3. Benchmark models

We have gathered a collection of 37 models for which finding good feasible solutions is difficult. All computational testing described in the next section are performed using these models. These models have been drawn from four sources:

- Five job-shop scheduling instances with earliness and tardiness costs. These `ljb` problems have been widely used in the genetic algorithms (GA) literature, as reported for example in [35]. We use a simple disjunctive MIP formulation [4] to create our test models. These models are available upon request from the authors.
- Eleven network design and multicommodity routing instances, described in [12]. On these `rococo` models, pure MIP approaches perform badly compared to other solution methods. The best known solutions are either provided by branch-and-price or by a combination of constraint programming and structured large neighborhood search (CP+LNS) [12]. These models are part of a larger public benchmark based on industrial data and are available upon request from the authors.
- Twenty of the twenty-four models studied in the local branching paper [17]. Original sources for these models are given in [17]. Two models from this set, `rail507` and `rail2586c`, were removed because they were easily solved to optimality with the latest version of CPLEX, and thus did not fit the objective of the benchmark set.

One model, van, was removed because the main difficulty in this model is poor numerical behavior, rather than an inherent combinatorial difficulty in finding good feasible solutions. Finally, model NSR8K was removed because no algorithm we tried, including Fischetti and Lodi's implementation of local branching, found a feasible solution, even after extended runs (5 hours) on our test platform (Fischetti and Lodi's original results were in fact obtained with CPLEX presolve turned off). These models are available on the Web[1].

– One model, swath, from MIPLIB 3.0 [8].

Table 1 gives various statistics for each instance, including the total number of variables (n), the number of binary variables (b), the number of general integer variables (i), the number of constraints (m), the objective value for the best integer solution known to us (bestUB), the corresponding relative gap with respect to the best known lower bound known to us, computed as $\frac{bestUB-bestLB}{bestUB}$, and the method that produced the best known solution (bestUBAlg).

To get a better sense of the scope for improvement on these models, we also include a few lower and upper bounds that were not obtained in the experiments described in the next section of this paper. For example, we ran some of our codes for very long time periods when we suspected that doing so might produce significantly better bounds. For the network design and job-shop models, better solutions were also often found by customized methods. We note those instances where the best solutions did not come from the experiments of the next section, including relevant references when appropriate. LB+RINS 1 and 2 refer to two hybrid algorithms combining RINS and local branching that are described in Section 4.1. Lower bounds used to compute the gaps reported in Table 1 are the best bounds obtained during the experiments described in Section 4 or during the few long runs previously mentioned. The best lower bounds for production models tr12-30, A1C1S1, A2C1S1, B1C1S1, and B2C1S1 come from M. Van Vyve's PhD thesis [34].

Let us reiterate that the methods considered in this paper are geared towards very difficult MIP models. They degrade performance, often significantly, on models where good feasible solutions are not difficult to find. Most MIP models are not of sufficient difficulty to showcase the abilities of these methods to find improved MIP solutions.

4. Computational results

4.1. Methods

We now present experimental results for the methods we have described, run on the benchmark set just presented. All experiments were done using CPLEX 8.1 on a 2Ghz Pentium IV system running Linux. Recall that the approaches considered are:

– Default CPLEX (see [9] for a description of the default CPLEX branch-and-cut strategies)
– Relaxation Induced Neighborhood Search (RINS), as described in Section 2.1

[1] http://www.or.deis.unibo.it/research_pages/ORinstances/MIPs.html

Table 1. The benchmark

Instance	n	b	i	m	bestUB	gap	bestUBAlg
A1C1S1	3648	192	0	3312	11557.09	4.00%	LB+RINS 1
A2C1S1	3648	192	0	3312	10889.14	2.35%	LB+RINS 2
B1C1S1	3872	288	0	3904	24544.25	10.96%	LB+RINS 1 (long)
B2C1S1	3872	288	0	3904	25740.15	12.60%	LB+RINS 1 (long)
arki001	1388	415	123	1048	7580813.0459	0.00%	LB
biella1	7328	6110	0	1203	3065084.57	0.03%	RINS, LB+RINS 2
glass4	322	302	0	396	1460013800.0	40.90%	LB+RINS 1
net12	14115	1603	0	14021	214	25.34%	RINS, guided dives, LB+RINS 1
nsrand_ipx	6621	6620	0	735	51360	1.39%	RINS, guided dives, LB+RINS 1, LB+RINS 2
rail2586c	13226	13215	0	2589	953	1.78%	LB
rail4284c	21714	21705	0	4287	1071	1.58%	LB
rail4872c	24656	24645	0	4875	1550	2.50%	LB+RINS 1
roll3000	1166	246	492	2295	12890	3.05%	LB+RINS 1
seymour	1372	1372	0	4944	423	2.62%	RINS
sp97ar	14101	14101	0	1761	662671913.92	1.16%	LB+RINS 2
sp97ic	12497	12497	0	1033	429562635.68	1.19%	LB+RINS 1
sp98ar	15085	15085	0	1435	529814784.7	0.35%	LB
sp98ic	10894	10894	0	825	449144758.40	0%	LB+RINS 1 (long)
tr12-30	1080	360	0	750	130596	0%	Default CPLEX and other strategies
UMTS	2947	2802	72	4465	30122200	0.21%	LB+RINS 1 (long)
swath	6805	6724	0	884	471.03	15.99%	LB-F&L 8.1
rococoB10-011000	4456	4320	136	1667	19449	3.04%	CP+LNS — inferred [12]
rococoB10-011001	4456	4320	136	1677	21265	9.68%	CP+LNS — inferred [12]
rococoB11-010000	12376	12210	166	3792	32246	11.99%	CP+LNS — inferred [12]
rococoB11-110001	12431	12265	166	8148	42444	12.17%	CP+LNS [12]
rococoB12-111111	9109	8778	331	8978	39831	19.50%	LB+RINS 1 (long)
rococoC10-001000	3117	2993	124	1293	11460	0%	CPLEX with `MIPEmphasis = 3` (long)
rococoC10-100001	5864	5740	124	7596	16664	11.01%	CP+LNS, default CPLEX — inferred [12]
rococoC11-010100	12321	12155	166	4010	20889	18.57%	CP+LNS — inferred [12]
rococoC11-011100	6491	6325	166	2367	20889	6.66%	CP+LNS [12]
rococoC12-100000	17299	17112	187	21550	35512	10.94%	CP+LNS [12]
rococoC12-111100	8619	8432	187	10842	35909	3.09%	CP+LNS — inferred [12]
ljb2	771	681	0	1482	0.507679	22.47%	RINS, LB+RINS 1, LB+RINS 2
ljb7	4163	3920	0	8133	0.133655	57.08%	RINS
ljb9	4721	4460	0	9231	0.739	76.67%	GA [35]
ljb10	5496	5196	0	10742	0.512	54.58%	GA [35]
ljb12	4913	4633	0	9596	0.399	79.73%	GA [35]

- Our implementation of local branching (LB), as described in Section 2.2
- Default CPLEX with guided dives, as described in Section 2.3

As mentioned in Sections 2.1 and 2.2, RINS and local branching each have two parameters that need to be set. For RINS, we need to choose the frequency f at which to call RINS and the sub-MIP node limit nl. For local branching, we need to choose the neighborhood radius r and the sub-MIP node limit nl. We experimented with different settings, and it turned out that results varied very little for RINS and slightly more for local branching. For the experiments reported in the following, we chose the parameters that appeared to perform best: $f = 100$ and $nl = 1000$ for RINS; $r = 10$ and $nl = 1000$ for local branching.

Note that it is possible to build many hybrid approaches by combining RINS, local branching, and guided dives. To give a simple example, guided dives could be applied to the RINS sub-MIP. It is a research subject of its own to explore thoroughly all possible combinations of these methods. To get some sense of the scope for improvement, though, we tried two straightforward hybrids that attempt to address some of the potential weaknesses of the individual algorithms. The first, which we call LB+RINS 1, enables both LB and RINS in the same MIP tree. Local branching is thus used to improve solutions found by RINS (as well as other solutions found in the branch-and-bound process). More precisely, RINS is called every f nodes. When the RINS sub-MIP exploration terminates, local branching is called on the last solution found by RINS (if an improving

solution was found). Local branching is also directly called when the global MIP solver finds a new incumbent. In both cases, when local branching terminates, optimization of the global MIP is resumed with a possibly updated incumbent. Hence the next iteration of RINS may use an incumbent discovered by local branching. The main goal is to increase both the number of local branching neighborhoods explored and their quality. Our second hybrid, LB+RINS 2, adds a local branching neighborhood constraint ($r = 20$) to every RINS sub-MIP. The goal is to limit the difficulty of the sub-MIP in cases where the relaxation and the incumbent differ significantly. For both hybrids, RINS and local branching termination criteria are the same as in pure RINS and pure local branching. Results for both of these hybrids will be presented along with those of the individual methods.

4.2. Methodology

Before discussing our experimental results, we first must make a few comments about how the results will be presented. Regarding solution quality, we have chosen to always capture quality as ratios of the objective value for the solution obtained by a method divided by the objective value for the best known solution for that model (from Table 1). This ratio is of course never less than 1.0.

Given the sheer volume of data our tests generated (6 different methods on 37 different models, with each generating multiple intermediate solutions), we found it necessary to reduce the data into summary results. Our goal with this summary information is to give a rough sense of which method has produced the best aggregate results in a given amount of time. We try to capture this notion using the geometric mean over all models in the relevant model set of the best solution obtained divided by the best solution known.

One obvious question is how long to allow the methods to run. A time limit that is too short may not give a method the opportunity to finish. This is particularly true for local branching, where a sequence of improving solutions may produce numerous sub-MIPs. On the other hand, a time limit that is too long may obscure significant differences between the algorithms, especially if one finds the optimal solution early in the time interval. We found that a one hour time limit struck a reasonable balance between these two considerations for most of the models in our test. That is, most methods had ceased to make significant progress after an hour, yet the methods were often finding marginally better solutions late in this time period. This will be apparent in the graphs shown shortly.

To avoid obscuring important differences between the behaviors of the different algorithms, we chose to group our models into three different sets. The ‘small spread’ models were those where the gap between the worst solution obtained by any of the 6 methods considered here and the best was less than 10%. The ‘medium spread’ models were those for which the gap was between 10% and 100%, and finally the ‘large spread’ models were those where the gap was larger than 100%. We chose to run the ‘large spread’ models for two hours instead of one, on the assumption that large differences between methods suggest that the results were likely to continue to evolve given more time.

4.3. Optimizing from scratch

We now present results for applying the various methods to each model from scratch. Recall that we will look at the ability of the various methods to improve a given solution or escape from a local minimum later in this section.

Table 2 shows solution quality obtained by the various methods at the end of the time limit, expressed as the ratio of the solution value found by the method divided by the best known solution value. The best ratio obtained by any method for a model is emphasized in bold face (with ties producing multiple bold ratios).

The numbers in this table were used to classify our models into the three previously described sets. One may note that the 'medium spread' set consists essentially of `rococo` problems, and that the 'large spread' set gathers all but one of the job-shop problems with earliness and tardiness costs. It is perhaps interesting to note that model

Table 2. Ratio solution reached/best known solution

Instance	Default CPLEX	RINS	Local branching	Guided dives	LB+RINS 1	LB+RINS 2
'Small spread' problems - one hour						
A1C1S1	1.039	1.006	1.011	1.016	**1.000**	1.002
A2C1S1	1.038	1.007	**1.000**	1.035	1.006	**1.000**
arki001	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
B2C1S1	1.070	1.041	1.079	1.047	**1.010**	1.024
biella1	1.004	**1.000**	1.001	**1.000**	1.001	**1.000**
nsrand_ipx	1.006	**1.000**	1.003	**1.000**	**1.000**	**1.000**
rail2586c	1.015	1.016	1.014	1.018	**1.007**	1.019
rail4284c	1.007	1.009	1.007	1.005	**1.002**	1.009
rail4872c	1.010	1.008	1.008	1.014	**1.000**	1.006
rococoB10-011000	1.041	1.022	1.034	1.022	**1.008**	1.044
rococoB10-011001	1.041	1.033	**1.025**	1.083	1.101	1.079
rococoB11-010000	**1.042**	1.061	1.118	**1.042**	1.119	1.093
rococoC10-001000	1.001	1.001	1.001	1.001	**1.000**	**1.000**
roll3000	1.014	1.005	1.003	1.016	**1.000**	1.001
seymour	1.012	1.002	1.009	1.005	1.002	1.002
sp97ar	1.012	1.001	1.012	1.006	1.005	**1.000**
sp97ic	1.026	1.002	1.012	1.013	**1.000**	1.006
sp98ar	1.007	**1.002**	**1.002**	1.003	**1.002**	1.003
sp98ic	1.015	1.003	1.004	1.004	**1.002**	1.005
tr12-30	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
UMTS	1.001	**1.000**	1.001	1.001	**1.000**	1.001
'Medium spread' problems - one hour						
B1C1S1	1.118	**1.011**	1.020	1.096	1.018	1.013
glass4	1.123	1.096	1.130	1.113	**1.000**	1.119
ljb2	1.225	**1.000**	1.116	1.038	**1.000**	**1.000**
net12	1.192	**1.000**	1.192	**1.000**	**1.000**	**1.000**
rococoB11-110001	1.205	**1.121**	1.254	1.224	1.216	1.148
rococoB12-111111	1.141	**1.029**	1.057	1.099	1.084	1.062
rococoC10-100001	1.180	**1.095**	1.214	1.132	1.137	1.294
rococoC11-010100	1.173	1.081	1.081	1.123	**1.053**	1.060
rococoC11-011100	1.312	**1.055**	1.330	1.118	**1.055**	1.452
rococoC12-100000	1.270	**1.096**	1.110	1.268	1.117	1.278
rococoC12-111100	1.148	**1.025**	1.125	1.112	1.070	1.081
swath	1.222	1.048	1.154	1.093	**1.031**	1.048
'Large spread' problems - two hours						
ljb7	2.375	1.061	1.580	1.582	**1.028**	1.255
ljb9	1.858	**1.581**	1.995	1.718	1.646	1.809
ljb10	1.601	**1.212**	1.693	1.295	1.226	1.500
ljb12	2.568	**1.512**	3.083	2.012	2.097	2.418

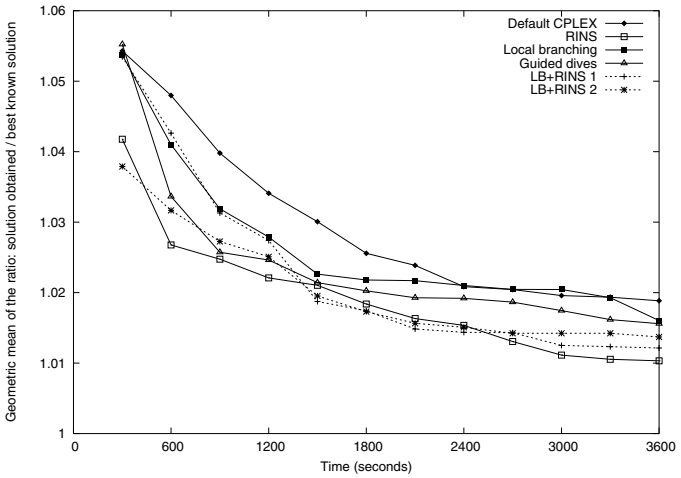

Fig. 1. 'Small spread' problems

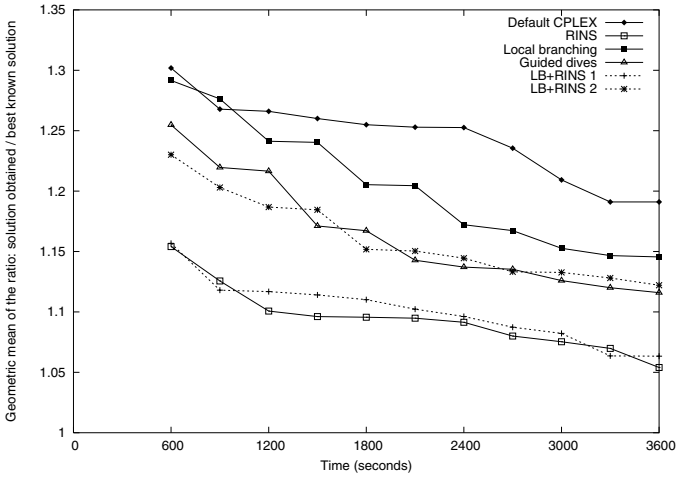

Fig. 2. 'Medium spread' problems

size is not the main determinant of spread in our set. In fact, the models in the 'large spread' set are among the smallest we consider.

Figures 1 through 3 give information on how solution quality evolves over time for the various methods.

We can draw a few basic conclusions from this experimental data. First, RINS is the most effective of the methods. It obtains good solutions earlier than the alternatives, and maintains a lead even as more time is allowed. Though less effective than RINS, guided dives still performs quite well, providing significantly better results than default CPLEX. Finally, local branching gives somewhat better results than default CPLEX overall, but runs into serious trouble on the problems of the 'large spread' set.

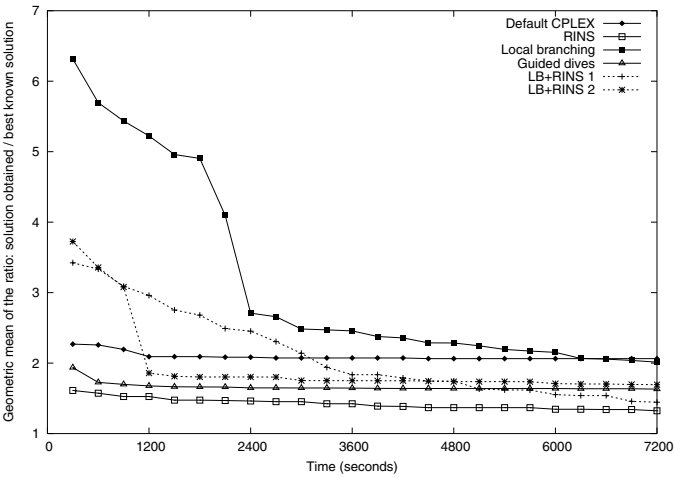

Fig. 3. 'Large spread' problems

If we look at the two hybrid methods LB+RINS 1 and LB+RINS 2, we find that both perform better than pure local branching, but worse than pure RINS. In other words, we see no apparent benefit to combining the techniques in this way. However, as we said earlier, the methods we consider can be combined in many different ways to produce hybrid approaches, so it seems likely to us that some other combination could be productive.

If we take a closer look at local branching performance on the 'large spread' models, we find that this behavior can perhaps be explained by looking at the first solutions discovered. Recall that RINS, local branching and guided dives are all inactive until the MIP solver has found its first integer solution, and thus they all begin with the same initial solution. It turns out that the first solutions found for these models are generally of poor quality. Thus, the local branching neighborhoods around them are unlikely to contain good solutions. This will produce new neighbor solutions of similarly poor quality, whose local branching neighborhoods will then be explored. This leads to a long series of not very fruitful sub-MIP explorations. RINS is not as sensitive to the quality of the first integer solutions found by the MIP solver for two reasons. The first is that the incumbent and the continuous relaxation serve symmetric roles in RINS. A poor quality incumbent does not necessarily produce a poor quality neighborhood. Secondly, the RINS sub-MIP contains fewer variables than the LB sub-MIP, and does not include the additional dense local branching constraint, so a given number of nodes is typically processed much more quickly.

It is clear from Figure 3 that branching within the standard MIP tree, as is done in default CPLEX and in guided dives, improves the initial poor quality solutions quickly (note that the left-most point on Figure 3 shows the solution after 5 minutes, not the first solution). It is therefore natural to wonder whether local branching would produce better results if it simply waited until standard branching found a good quality solution. The problem with such an approach is in how to recognize a good quality solution, especially for very difficult models, where the relaxation objective may be quite far from the optimal objective and thus may not provide useful guidance.

Table 3. RINS sub-MIPs

Instance	Size of sub-MIPs	Time spent in sub-MIPs	Number of sub-MIPs
A1C1S1	0.882	0.910	35
A2C1S1	0.891	0.906	35
arki001	0.328	0.660	1260
B2C1S1	0.873	0.827	14
biella1	0.077	0.004	192
nsrand_ipx	0.026	0.306	858
rail2586c	0.126	0.429	9
rail4284c	0.096	0.263	4
rail4872c	0.104	0.312	3
rococoB10-011000	0.109	0.104	331
rococoB10-011001	0.111	0.262	252
rococoB11-010000	0.062	0.123	48
rococoC10-001000	0.095	0.068	308
roll3000	0.423	0.126	1111
seymour	0.398	0.606	65
sp97ar	0.012	0.182	343
sp97ic	0.005	0.107	1380
sp98ar	0.009	0.144	601
sp98ic	0.006	0.151	887
tr12-30	0.590	0.353	2788
UMTS	0.134	0.506	741
B1C1S1	0.880	0.884	23
glass4	0.298	0.495	23856
ljb2	0.199	0.806	1700
net12	0.915	0.001	10
rococoB11-110001	0.057	0.142	42
rococoB12-111111	0.054	0.154	80
rococoC10-100001	0.095	0.068	65
rococoC11-010100	0.139	0.234	28
rococoC11-011100	0.109	0.178	56
rococoC12-100000	0.041	0.037	36
rococoC12-111100	0.043	0.043	144
swath	0.013	0.050	5056
ljb7	0.098	0.739	780
ljb9	0.110	0.794	547
ljb10	0.138	0.825	428
ljb12	0.142	0.839	359
Geometric mean over			
... all instances	0.109	0.202	-
... 'small spread' set	0.092	0.211	-
... 'medium spread' set	0.141	0.113	-
... 'large spread' set	0.121	0.798	-

A natural question at this point is how large the RINS sub-MIPs are, and how difficult they are to solve. Table 3 shows the average number of variables of a RINS sub-MIP (expressed as a fraction of the global MIP size), the time spent in RINS sub-MIPs (expressed as a fraction of the overall time), and the total number of sub-MIPs explored within the time limit. While we had some concern that relaxations and incumbents might sometimes differ greatly, thus producing large, time-consuming sub-MIPs, we in fact found limited correlation between the size of the sub-MIP and the fraction of time spent within the RINS heuristic. As is true of MIPs in general, the difficulty of a sub-MIP depends on more than just its size.

Another natural question is whether a failure to find a particular, good solution was more often due to limitations in the neighborhood (i.e., the RINS neighborhoods did not contain that solution), or limitations of sub-MIP exploration (i.e., the sub-MIP was too difficult to explore in any depth). We tried a simple experiment, where we considered a feasible solution that RINS had difficulty finding and tested whether any RINS neighborhood built during the MIP search contained that solution. We found that they rarely did. While RINS neighborhoods may contain improving solutions that weren't found by our exploration approach, our conclusion is that significant improvements in RINS are more likely to come from improved neighborhood definition rather than improved exploration of neighborhoods.

4.4. Improving a given solution

To further understand the strengths and weaknesses of the various strategies considered in this paper, with the hope of using any resulting insights to build better combinations of these strategies, we designed an additional set of experiments. In these experiments, we identified three specific factors that might be useful for achieving good overall results. The first was the ability of an algorithm to quickly improve on a poor solution. This was motivated by our observations about the performance of local branching on the 'large spread' model set. The second was the ability of an algorithm to continue improving on a good incumbent. The last, somewhat related factor was the ability of an algorithm to diversify the search so as to generate new neighborhoods in which to perform new intensification. If different algorithms showed different strengths, that would certainly help to guide our efforts to find an effective hybrid strategy.

This section considers the ability of each algorithm to improve on a given solution. More specifically, for each model, two initial feasible solutions were used:

- A solution of poor quality: one of the first integer solutions obtained by default CPLEX.
- A solution of good quality: the solution obtained by default CPLEX after one hour computation.

For each algorithm (default CPLEX included), the initial solution is given as a MIP start at the top of a new branch-and-cut tree. This does not exactly correspond to the situation of improving a given solution encountered during optimizing from scratch since it does not take into account how the MIP tree has been built and pruned before finding this initial solution, but this information is equally disregarded for each algorithm.

For each model, the time limit was set to the amount of time needed by local branching to finish improving the initial solution, rounded up to the nearest 100 seconds. Since one local branching iteration takes typically more time than one RINS sub-MIP, our chosen time limit allows each method to complete at least one iteration. While RINS benefits from the global MIP enumeration if the first RINS sub-MIP exploration terminates before the time limit, experience shows that, on our benchmark, the global MIP solver rarely finds integer solutions by itself. Therefore the incumbent improvement in RINS runs is provided almost exclusively by the exploration of RINS sub-MIPs.

Table 4 shows the mean improvement in solution quality. We computed these numbers by taking the geometric mean of the solution objective value at the end of the time

Table 4. Average percent improvement in solution quality

	Default CPLEX	RINS	Local branching	Guided dives
Starting from a poor solution				
... all instances	39.44%	44.98%	35.83%	42.16%
... 'small spread' set	26.96%	28.03%	24.56%	26.97%
... 'medium spread' set	38.91%	48.39%	40.03%	44.08%
... 'large spread' set	77.94%	83.74%	66.38%	81.19%
Starting from a good solution				
... all instances	0.28%	3.95%	1.99%	2.82%
... 'small spread' set	0.00%	0.64%	0.62%	0.25%
... 'medium spread' set	0.18%	2.30%	2.53%	1.08%
... 'large spread' set	2.04%	23.62%	7.36%	19.69%

period over the initial solution value, and subtracting the result from 1.0. A 50% improvement would thus mean that the objective was reduced by half. The data shows that RINS is the best of these strategies for improving a poor solution. This result was expected, since the use of the continuous relaxation often compensates for poor incumbents. The tentative explanation given in the previous section to understand the poor performance of local branching over the `ljb` models is confirmed here: local branching is significantly worse than both RINS and guided dives, and only sometimes slightly better than default CPLEX, at improving poor solutions. More surprisingly, Table 4 shows that local branching is never significantly better than RINS or guided dives at improving a good solution, and sometimes much less effective.

4.5. Diversifying the search

In this section, we look at another aspect of algorithm effectiveness by considering how much time each algorithm requires to produce an improved solution; that is, how effective its diversification scheme is. Recall that our implementation of local branching relies on the global MIP solver (default CPLEX) for diversification, so we present no local branching results here.

Table 5 shows the effectiveness of each algorithm at diversifying the search, expressed as the median time required to produce an improved solution. We started each algorithm from the same good and poor solutions as in the previous section. In contrast to the previous test, runtimes for each method were limited to half an hour for all models. The table shows that RINS is the most robust of the methods, experiencing the fewest failures to find an improved solution within the time limit. It is also the fastest method when starting from a solution of poor or good quality.

While it is interesting to discover that RINS was the most effective method in all of our more focused experiments, these results unfortunately offer little insight into how to combine algorithms to form a more effective hybrid.

5. Conclusion

In this paper we introduced two new generic strategies to quickly find good integer solutions for hard MIP models: a heuristic we call *Relaxation Induced Neighborhood*

Table 5. Diversification efficiency

	Default CPLEX	RINS	Guided dives
Starting from a poor solution			
Number of fails (out of 37): no improved solution found in half an hour	1	0	1
Median time (in seconds) to first improved solution			
... in the 36 cases where all algorithms succeed	36.48	24.98	30.59
Starting from a good solution			
Number of fails (out of 37): no improved solution found in half an hour	32	4	14
Median time (in seconds) to first improved solution			
... in the 5 cases where all algorithms succeed	284.44	9.09	53.19
... in the 23 cases where only RINS and guided dives succeed	-	99.56	119.74

Search (RINS) and a tree traversal strategy we call *guided dives*. Both algorithms are domain-independent; they can be applied to any MIP, with no other input than the model itself.

Comparing RINS to an existing neighboorhood exploration method called local branching [17], we find that RINS improves on it in several ways:

- Local branching defines its neighborhood around the incumbent. In contrast, RINS neighborhoods are based on the incumbent and on the continuous relaxation, which play symmetrical roles. This allows for RINS to improve quickly on poor incumbents and to be robust with respect to loose continuous relaxation.
- Local branching can only be invoked when a new incumbent is found. In contrast, RINS can be invoked at each node of the branch-and-cut tree, since diversification is achieved through changes in the continuous relaxations. This automatic diversification is especially important for models where a standard MIP solver may find new feasible solutions only rarely.
- A RINS sub-MIP solves faster than a local branching sub-MIP. It contains fewer variables, and it does not contain the local branching dense constraint.

We showed with extensive computational experiments on a number of very difficult MIP models that RINS outperforms local branching, guided dives, and default CPLEX for generating good integer solutions within a time limit, improving a given solution of good or poor quality, and diversifying the search. Guided dives is the second best strategy for these three criteria, also providing consistent improvements over default CPLEX.

Both RINS and guided dives are included in CPLEX 9.0. RINS is invoked with parameter `IloCplex::MIPEmphasis=4`, and guided dives is invoked with parameter `IloCplex::DiveType=3`.

Finally, let us set the algorithms considered in this paper in the more general context of hybrid algorithms. We view RINS, guided dives, and local branching as being *hybrid in spirit*. They combine concepts from mixed integer programming (linear relaxation in RINS; valid inequalities in local branching), from general tree search (hard fixing in RINS; soft fixing in local branching; diving to promising nodes in guided dives), and from local search (intensification, diversification and particularly the concept of a solution neighborhood). However, unlike traditional hybrid algorithms [15], they integrate

these concepts in the framework of only one optimization technique (here, branch-and-cut). This property provides several advantages, including reduced software complexity, limited risk of over-specialization to a particular problem class (no problem specific parameter tuning was required for the results in this paper), and automatic gains from improvements in the underlying method. We refer the reader to [13] for a more detailed discussion of algorithms that are hybrid in spirit.

Acknowledgements. We wish to thank the two anonymous referees for their helpful comments.

References

1. Aboudi, R., Jörnsten, K.: Tabu Search for General Zero-One Integer Programs Using the Pivot and Complement Heuristic. ORSA J. Comput. **6** (1), 82–93 (1994)
2. Adams, J., Balas, E., Zawack, D.: The Shifting Bottleneck Procedure for Job-Shop Scheduling. Manage. Sci. **34** (3), 391–401 (1988)
3. Anbil, R., Gelman, E., Patty, B., Tanga, R.: Recent advances in crew-pairing optimization at American Airlines. Interfaces **21**, 62–74 (1991)
4. Applegate, D., Cook, W.: A Computational Study of the Job-Shop Scheduling Problem. ORSA J. Comput. **3** (2), 149–156 (1991)
5. Balas, E., Ceria, S., Dawande, M., Margot, F., Pataki, G.: OCTANE: A New Heuristic for Pure 0-1 Programs. Oper. Res. **49** (2), 207–225 (2001)
6. Balas, E., Martin, C.: Pivot and Complement – A Heuristic for 0-1 Programming. Manage. Sci. **26** (1), 86–96 (1980)
7. Baptiste, P., Le Pape, C., Nuijten, W.: Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling. In: Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research, 1995
8. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0 Optima **58**, 12–15 (1998)
9. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and practice – closing the gap. Kluwer Academic Publishers, 2000, pp. 19–49
10. Caseau, Y., Laburthe, F.: Disjunctive Scheduling with Task Intervals. Technical report, École Normale Supérieure, 1995
11. Caseau, Y., Laburthe, F.: SaLSA Specification language for search algorithms. Technical report, École Normale Supérieure, LIENS-97-11, 1997
12. Chabrier, A., Danna, E., Le Pape, C., Perron, L.: Solving a Network Design Problem. To appear in Annals of Operations Research, Special Issue following CP-AI-OR'2002, 2004
13. Danna, E.: Intégration des techniques de recherche locale à la programmation linéaire en nombres entiers (in French). PhD thesis, Université d'Avignon, 2004
14. Danna, E., Le Pape, C.: Accelerating branch-and-price with local search: A case study on the vehicle routing problem with time windows. Technical Report, ILOG, 03-006, 2003
15. Danna, E., Le Pape, C.: Two generic schemes for efficient and robust cooperative algorithms. Constraint and integer programming, Michela Milano (ed.), Kluwer Academic Publishers, 2003, pp. 33–57
16. Faaland, B.H., Hillier, F.S.: Interior path methods for heuristic integer programming procedures. Oper. Res. **27** (6), 1069–1087 (1979)
17. Fischetti, M., Lodi, A.: Local Branching. Math. Program. Ser. B **98**, 23–47 (2003)
18. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, 1997
19. Glover, F., Laguna, M., Martí, F.: Fundamentals of Scatter Search and Path Relinking. Control and Cybernetics **29** (3), 653–684 (2000)
20. Glover, F., Løkketangen, A., Woodruff, D.L.: Scatter Search to Generate Diverse MIP Solutions. In: M. Laguna, J.L. González-Velarde, (eds.), OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, Kluwer Academic Publishers, 2000, pp. 299–317
21. Hillier, F.S.: Efficient heuristic procedures for integer linear programming with an interior. Oper. Res. **17** (4), 600–637 (1969)
22. Ibaraki, T., Ohashi, T., Mine, H.: A heuristic algorithm for mixed-integer programming problems. Math. Program. Study **2**, 115–136 (1974)

23. Junker, U., Nuijten, W.: Preference-based Search for Minimizing Changes in Rescheduling Problems. In: IJCAI-99 Workshop on scheduling and planning meet real-time: Monitoring in a dynamic and uncertain world, 1999, pp. 39–45
24. Jussien, N., Lhomme, O.: Local search with constraint propagation and conflict-based heuristics. Artificial Intelligence **139**, 21–45 (2002)
25. Le Pape, C., Baptiste, P.: Heuristic Control of a Constraint-Based Algorithm for the Preemptive Job-Shop Scheduling Problem. J. Heuristics **5** (3), 305–325 (1999)
26. Løkketangen, A., Woodruff, D.L.: 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)
27. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling. Artificial Intelligence **58**, 161–205 (1992)
28. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, 1996
29. Nediak, M., Eckstein, J.: Pivot, cut and dive: A Heuristic for 0-1 Mixed Integer Programming. Technical Report, Rutgers Center for Operations Research, RRR 53-2001, 2001
30. Palpant, M., Artigues, C., Michelon, P.: A heuristic for solving the frequency assignment problem. In: XI Latin-Iberian American Congress of Operations Research (CLAIO), 2002
31. Shaw, P.: 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), 1998, pp. 417–431
32. Van Hentenryck, P., Le Provost, T.: Incremental Search in Constraint Logic Programming. New Generation Comput. **9** (3), 257–275 (1991)
33. Van Laarhoven, P.J.M., Aarts, E.H.L.: Simulated Annealing: Theory and Practice. Kluwer Academic Publishers, 1987
34. Van Vyve, M.: A solution approach of production planning problems based on compact formulations for single-item lot-sizing models. PhD thesis, Université catholique de Louvain-la-Neuve, 2003
35. Vásquez, M., Whitley, L.D.: A comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), 2000, pp. 1011–1018