Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TIMO BERTHOLD*, STEFAN HEINZ*,
MARC E. PFETSCH[1], STEFAN VIGERSKE[2,*]

Large Neighborhood Search beyond MIP

[1] Technische Universität Braunschweig, Institut für Mathematische Optimierung, Pockelsstraße 14, 38106 Braunschweig, Germany, m.pfetsch@tu-bs.de
[2] Humboldt-Universität zu Berlin, Institut für Mathematik, Unter den Linden 6, 10099 Berlin, Germany, stefan@math.hu-berlin.de

* Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

ZIB-Report 11-21 (May 2011)

Large Neighborhood Search beyond MIP

Timo Berthold[1], Stefan Heinz[1], Marc E. Pfetsch[2], Stefan Vigerske[3]

[1] Zuse Institute Berlin
Takustr. 7, 14195 Berlin, Germany
{berthold,heinz}@zib.de

[2] Technische Universität Braunschweig, Institut für Mathematische Optimierung
Pockelsstraße 14, 38106 Braunschweig, Germany
m.pfetsch@tu-bs.de

[3] Humboldt-Universität zu Berlin, Institut für Mathematik
Unter den Linden 6, 10099 Berlin, Germany
stefan@math.hu-berlin.de

Abstract

Large neighborhood search (LNS) heuristics are an important component of modern branch-and-cut algorithms for solving mixed-integer linear programs (MIPs). Most of these LNS heuristics use the LP relaxation as the basis for their search, which is a reasonable choice in case of MIPs. However, for more general problem classes, the LP relaxation alone may not contain enough information about the original problem to find feasible solutions with these heuristics, e.g., if the problem is nonlinear or not all constraints are present in the current relaxation.

In this paper, we discuss a generic way to extend LNS heuristics that have been developed for MIP to constraint integer programming (CIP), which is a generalization of MIP in the direction of constraint programming (CP). We present computational results of LNS heuristics for three problem classes: mixed-integer quadratically constrained programs, nonlinear pseudo-Boolean optimization instances, and resource-constrained project scheduling problems. Therefore, we have implemented extended versions of the following LNS heuristics in the constraint integer programming framework SCIP: LOCAL BRANCHING, RINS, RENS, CROSSOVER, and DINS. Our results indicate that a generic generalization of LNS heuristics to CIP considerably improves the success rate of these heuristics.

1 Introduction

Large neighborhood search (LNS) is a variant of the local search paradigm that has been widely used in Constraint Programming, Operations Research, and Combinatorial Optimization [1, 22, 23]. LNS has proved to be an extremely successful metaheuristic for a wide range of applications in recent years, see, for example, Pisinger and Røpke [24]. The main idea is to restrict the search for "good" solutions to a neighborhood of specific points – usually close to optimal/feasible solutions. The hope is that such a restriction makes the subproblem much easier to solve, while still providing solutions of high quality.

In *mixed-integer linear programming* (MIP), LNS has recently been realized in a series of primal heuristics [5, 6, 13–15, 26]. The so-called LOCAL BRANCHING heuristic has been further extended to constraint programs [16] and mixed-integer nonlinear programs [19].

The goal of this paper is to show that the above mentioned LNS heuristics, which have specifically been developed for MIP, can be extended in a straightforward manner to the broad class of so-called constraint integer programs. We show – via computational experiments for three general problem classes – that this leads to very powerful heuristics. As prototype applications, we consider *mixed-integer quadratically constrained programs* (MIQCPs), *pseudo-Boolean optimization* (PBO), and *resource-constrained project scheduling problems* (RCPSPs). Each of these problems forms a subclass of *constraint integer programs (CIPs)* [3]. CIP adopts modeling and solving techniques from *constraint programming* (CP), MIP, and *satisfiability testing* (SAT). The aim is to restrict the generality of CP modeling as little as needed, while still retaining the full performance of MIP solving techniques.

The contribution of this paper is to investigate the performance of *generic* implementations of LNS heuristics in a complete CIP solver; note that this is in general not competitive with handcrafted problem

specific heuristics. It turns out, that this allows for considerable improvements in finding "good" solutions in the beginning of the solving process for instances in all three considered problem classes. Since the restriction to a neighborhood of some solution again generates a CIP, LNS heuristics are conceptually well suited as generic heuristics for CIPs. Thus, once a CIP solver is able to handle a certain problem class, many LNS heuristics that work on this problem class come at almost no additional implementation cost. We have realized these generic LNS heuristics and the concrete problem classes in the CIP solver SCIP [3, 27]. Its plugin oriented design allows for an easy implementation.

This paper is structured as follows. We first review LNS heuristics in MIP and describe the considered LNS heuristics and problem classes. We then briefly discuss their realization in SCIP, which is a competitive solver for MIQCP [7], PBO [9], and RCPSP [8]. Finally, we present computational results that illustrate the effect of the considered LNS heuristics.

2 Large Neighborhood Search for MIP

A *mixed-integer linear program* is an $\mathcal{NP}$-hard optimization problem, which can be written in the following form

$$\begin{aligned} \min \quad & \boldsymbol{d}^T \boldsymbol{x} \\ \text{s.t.} \quad & A\boldsymbol{x} \leq \boldsymbol{b}, \\ & x_j \in \mathbb{Z}, \qquad\qquad j \in J, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{d} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$, and $J \subseteq \{1, \ldots, n\}$ denotes the subset of integral variables.

Many MIP primal heuristics published in recent years [5, 6, 13–15, 26] are based on large neighborhood search. These heuristics investigate a neighborhood of a small set of starting points such as the best known integral solution (*incumbent*) or the optimal solution of the *linear programming* (LP) relaxation in which the integrality restriction has been dropped. The heuristics create a sub-MIP of the original MIP, typically by fixing some variables to values that are taken from the given points. For problems with binary variables only, another possibility is to add linear constraints, which restrict the number of variables that are different from the given point. By the use of auxiliary variables, this can be extended to problems with general integer variables while maintaining linearity.

Obviously, a good definition of the neighborhood is the crucial point: The neighborhood should contain high quality solutions, these solutions should be easy to find, and the neighborhood should be easy to process. Naturally, these three goals are conflicting in practice. In the remainder of this section, we will give a brief introduction to LNS heuristics for MIPs that have been proposed in the literature of the last ten years.

LOCAL BRANCHING [14] measures the distance to the starting point in Manhattan norm on the integer variables and only considers solutions which are inside a k-neighborhood of the reference solution, where k is typically between 10 and 20. This is done by adding a linear constraint that sums up the distance to the incumbent over all variables. In case of general integer variables, auxiliary variables might have to be used to model the absolute value. For details, see [14].

A "good" MIP solution fulfills three conditions: it is integral, feasible for the linear constraints, and it has a small objective function value (in case of minimization problem). *The relaxation induced neighborhood search* (RINS) [13] uses two starting points: The incumbent MIP solution which fulfills the first two requirements and the optimum of the LP relaxation which fulfills the latter two. RINS defines the neighborhood by fixing all integer variables which take the same value in both solutions.

In contrast to RINS, the *relaxation enforced neighborhood search* (RENS) [6] does not require an incumbent solution. Thus, it can be used as a start heuristic. RENS fixes all integer variables that take an integral value in the optimal solution of the LP relaxation. For the remaining integer variables, the bounds get tightened to the two nearest integral values.

CROSSOVER is an improvement heuristic that is inspired by genetic algorithms [5, 26] and requires more than one feasible solution. For a set of feasible solutions, e.g., the three best found so far, it fixes variables that take identical values in all of them.

RINS, RENS, and CROSSOVER solely fix variables; no further constraints are added to the subproblem.

DINS [15] combines the ideas of RINS and LOCAL BRANCHING. It defines the neighborhood by introducing a distance function between the incumbent solution and the optimum of the LP relaxation. When applied during a branch-and-bound search, it further takes into account how variables change their values at different nodes of the tree.

All those primal heuristics have been developed for mixed-integer *linear* programs. Except for LOCAL BRANCHING [16, 19], the authors are not aware of published extensions to more general problem classes. The mentioned heuristics depend on the existence of a relaxation and/or incumbent solution.

One possibility to extend them to more general problem classes is to apply them to the MIP which results from taking a linear relaxation of the nonlinear problem plus the integrality constraints. In this paper, we show that a more promising way is to create a subproblem by taking a copy of the original problem plus additional constraints specific to the particular LNS heuristic.

3 Constraint Integer Programs

A constraint integer program (CIP) is an optimization problem with a finite number of variables, where a linear function is minimized with respect to a finite set of constraints and with integrality restrictions imposed on all or a subset of the variables. Each constraint is specified by a mapping that indicates whether a given assignment to the variables satisfies this constraint or not. Further, it is required that the subproblem remaining after fixing all integer variables can be solved efficiently. Typically, this subproblem is a linear program [3], but also nonlinear problems are possible, see, e.g., [10]. Note that general objective functions can be modeled by introducing an auxiliary variable that is linked to the actual objective function via an additional constraint.

In this section we introduce the three problem classes which we use for our experiments. These are mixed-integer quadratically constrained programs, pseudo-Boolean optimization, and resource-constrained project scheduling problems.

3.1 Mixed-Integer Quadratically Constrained Programs

A *mixed-integer quadratically constrained program* (MIQCP) is a special case of a CIP in which all constraints are given by either linear or quadratic functions. MIQCPs arise in many areas, for example in mine production scheduling [11]. Formally, an MIQCP can be written in the following form:

$$\begin{array}{lll} \min & \boldsymbol{d}^T \boldsymbol{x} & \\ \text{s.t.} & \boldsymbol{x}^T A_i \, \boldsymbol{x} + \boldsymbol{b}_i^T \, \boldsymbol{x} + c_i \leq 0, & i = 1, \ldots, m \\ & x_j \in \mathbb{Z}, & j \in J, \end{array}$$

where $A_i \in \mathbb{R}^{n \times n}$, $\boldsymbol{b}_i \in \mathbb{R}^n$, and $c_i \in \mathbb{R}$ for $i = 1, \ldots, m$, and $J \subseteq \{1, \ldots, n\}$ denotes the subset of integral variables. If $A_i = 0$, then constraint i is *linear*, otherwise *quadratic*. If A_i is positive semidefinite, then the constraint i is called *convex*, otherwise *nonconvex*. Note that quadratic objective functions can be handled by including an appropriate constraint and an artificial variable.

In the presence of nonconvex constraints, SCIP applies a spatial branch-and-bound approach [10], i.e., branching on continuous variables. As a consequence, the sizes of nonconvex problems that can be solved efficiently are by several orders of magnitude smaller than those of convex problems. SCIP is a competitive solver for MIQCP, see [10, 17].

Test Set. For our experiments we used the test set introduced in [7], from which we removed instances that SCIP reformulates as MIPs during presolve. This leads to a test set of 64 instances.

3.2 Pseudo-Boolean Optimization

Pseudo-Boolean optimization (PBO) extends the satisfiability testing (SAT) problem by allowing integer coefficients in constraints, multiplication of variables, and an objective function. As in SAT, variables take 0/1 (false/true) values. Applications of PBO problems arise, for example, in cryptography [21].

For a Boolean variable $x \in \{0,1\}$, a *literal* ℓ is either the original variable x or its negation $\overline{x} := 1-x$. A *(nonlinear) pseudo-Boolean problem* with Boolean variables $x_1, \ldots, x_n$ is an optimization problem of the following form:

$$\begin{aligned} \min \quad & \sum_{j=1}^{t_0} c_j \cdot \prod_{\ell \in I_{0j}} \ell && && (1)\\ \text{s.t.} \quad & \sum_{j=1}^{t_i} a_{ij} \cdot \prod_{\ell \in I_{ij}} \ell \ \geq b_i && \text{for } i = 1, \ldots, m \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned}$$

Here, m is the number of constraints, I_{ij} is a subset of literals for $i = 0, \ldots, m$ and $j = 1, \ldots, t_i$, where t_i is the number of summands in constraint i. All coefficients a_{ij}, b_i, c_j are required to be integral. Let

$$\mathcal{I} := \{(i,j) \ : \ i \in \{0, \ldots, m\}, \ j \in \{1, \ldots, t_i\}\}.$$

The above formulation is very general: one can easily incorporate maximization, "$\leq$" constraints, equations, and pure satisfiability problems. If $|I_{ij}| \leq 1$ for all $(i,j) \in \mathcal{I}$, the objective function and the constraints are linear expressions in the variables. We call such instances *linear pseudo-Boolean problems*. If the objective function equals zero (or any other constant), we have *satisfiability* problems, otherwise *optimization* problems. In this paper, we only consider nonlinear PBO problems with objective function. For more details on the handling of nonlinear PBO problems in SCIP, see [9].

Test Set. For our experiments, we use the OPT-SMALLINT-NLC instances of the pseudo-Boolean competition 2010, see [12]. This set contains 409 instances that contain at least one nonlinear constraint and an objective function. In this category, SCIP was the best solver of all submitted solvers in the years 2009 and 2010.

3.3 Resource-Constrained Project Scheduling Problem

The *resource-constrained project scheduling problem* (RCPSP) consists of a set $\mathcal{J}$ of non-preemptable jobs that have to be scheduled over time and a set $\mathcal{R}$ of renewable resources. Each resource $k \in \mathcal{R}$ has bounded capacity $R_k \in \mathbb{N}$. Every job j has a processing time $p_j \in \mathbb{N}$ and resource demands $r_{jk} \in \mathbb{N}$ for each resource $k \in \mathcal{R}$. Moreover, the schedule has to satisfy precedence constraints which are given via a precedence graph $D = (V, A)$ with $V \subseteq \mathcal{J}$. An arc $(i,j) \in A$ represents the fact that job i must be finished before job j starts. The task is to schedule all jobs with respect to resource and precedence constraints, such that the *makespan*, i.e., the latest completion time of all jobs, is minimized.

The RCPSP can be modeled easily as a constraint program using the global `cumulative` constraint [4], which enforces that at each point in time, the accumulated demand of the currently running jobs does not exceed the given capacities. Given a vector $\boldsymbol{S}$ of start time variables S_j for each job j, the RCPSP can be modeled as follows:

$$\begin{aligned} \min \quad & \max_{j \in \mathcal{J}} S_j + p_j && && \\ \text{s.t.} \quad & S_i + p_i \leq S_j && \forall\, (i,j) \in A && (2)\\ & \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{\cdot k}, R_k) && \forall\, k \in \mathcal{R} \\ & S_j \in \mathbb{N} && \forall\, j \in \mathcal{J}. \end{aligned}$$

To formulate this problem as a CIP, we introduce an artificial variable X for the makespan and add constraints $S_j + p_j \leq X$ for all $j \in \mathcal{J}$. This CIP model was previously considered in [8].

Test Set. For our experiments we used the RCPSP instances in the PSPLIB [25]. This library contains four categories which differ by the amount of jobs which have to be scheduled: 30, 60, 90, or 120 jobs. The first three categories contain 480 instances each, the last 600 instances. This gives a total of 2040 instances.

4 Implementation in SCIP

The framework SCIP solves constraint integer programs to proven global optimality by a branch-and-cut algorithm. This means that the problem is recursively split into smaller subproblems, thereby creating a branching tree and implicitly enumerating all potential solutions. At each node, an LP relaxation is solved which may be strengthened by adding further valid constraints (cutting planes). Nodes in which the lower bound of the LP relaxation shows that it cannot contain feasible solutions better than the best primal solution can be pruned from the search tree. Primal heuristics are used as supplementary methods to improve the upper bound.

In branch-and-bound algorithms that use an LP relaxation, the knowledge of a high quality primal solution early during the search often is a crucial part of the optimization procedure. A lot of procedures such as cutting plane separation or pre-solving are applied exclusively or at least more extensively at the root node than at other nodes. Knowing a primal solution during root node processing guides the remaining search and help to deduce further reductions of the variable domains, thereby avoiding redundant search. This is achieved via reduced cost and pseudo objective propagation, see, e.g., [20]. These reductions can lead to stronger cutting planes; thus, a good primal solution may help to improve the dual bound. Note that in SCIP, heuristics are applied in a sequential order, processing the gained information between the calls. Consequently, the performance of one heuristic may influence another heuristic.

SCIP follows a plugin-based design. The core provides all necessary infrastructure to implement branch-and-bound based algorithms. It handles the branching tree along with all subproblem data, automatically updates the LP relaxations and domain storage, manages parameter settings, and incorporates its own memory management. Additionally, a cut and a solution pool, pricing and separation storage management, and a SAT-like conflict analysis mechanism [2] are available.

Besides the infrastructure, all main algorithms are implemented as external plugins. Plugins are objects that interact with the framework through callback functions specified by the user. The default plugins that are included in the standard distribution of SCIP implement a full-scale stand-alone MIP solver. In particular, SCIP features implementations of the five LNS heuristics for MIP that have been described in Section 2. Further, SCIP includes plugins that allow to solve the problem classes from Section 3, see [8–10].

Originally, the LNS heuristics implemented in SCIP created the LNS-subproblem by taking a copy of the LP relaxation, adding integrality constraints, and fixing variables (or adding a local branching constraint). Recently, we have implemented copying procedures in SCIP and redesigned all LNS heuristics to copy the original CIP into a new SCIP instance rather than the relaxation and applying the neighborhood search on this copy.

The original approach works well for MIP, since the MIP is fully specified through its LP relaxation and the integrality constraints. This, however, is not true for more general classes of CIP, i.e., an LP relaxation may not incorporate all information to specify the corresponding CIP, see, e.g., [28] for LP relaxations of MINLPs. In this case, a feasible solution of the LNS problem is no longer guaranteed to be feasible for the original problem. As a consequence, the chances that a LNS heuristic finds a feasible solution when working only on the relaxation of the problem are usually small. Thus, copying the whole problem, restricting the search space to a neighborhood of some point, and solving the resulting, hopefully easier, CIP seems a more promising approach for problem classes that are more general than MIP.

Note that LNS heuristics do not make any particular assumptions on the problem class. Thus, the approach of using a copy of the whole CIP enables the easy application of LNS heuristics to any problem class for which the corresponding SCIP plugins implement the required copy methods. SCIP can be

seen as an interface here: the formulation of a problem as constraint integer program allows the access to all metaheuristic methods described in Section 2. Unlike for most classic metaheuristic approaches, no additional problem specific adaption of the heuristic is necessary. The implementation of the constraint handlers (e.g., for quadratic, cumulative, and logic constraints) that enable SCIP to solve the corresponding types of problems to optimality suffices to get an implementation of the mentioned LNS heuristics for free. The additional power comes from the fact that the copied instance allows for stronger constraint propagation than that of linear constraints and separation of problem specific cutting planes.

5 Computational Experiments

The aim of our computational experiments is to investigate the potential of LNS heuristics, applied inside a branch-and-bound process, for more general problem classes than MIP. Since we are interested in finding good primal solutions early in the solution processes, we ran all experiments with a node limit of one, i.e., we only solved the root node of the branch-and-bound tree. We also impose a time limit of 24 hours. All computations presented in the following used SCIP version 2.0.1.3. As the underlying linear programming solver we choose CPLEX 12.2.0.0, continuous quadratic subproblems in the MIQCP experiments were solved by IPOPT 3.9 using CppAD version `trunk` (20110118) for computing function derivatives in case quadratic constraints are reformulated to second order cone constraints. The results were obtained on a Linux cluster of 64bit Intel Core i3-550 CPUs at 3.2 GHz with 4 MB cache and 8 GB main memory.

In the first experiment, we used the *default* settings of SCIP, in which the only LNS heuristic used at the root node is RENS, based on the LP relaxation. In [6], it has been shown that RENS often succeeds in finding feasible solution for MIPs during root node processing. In a second experiment, we keep the LP as basis for the LNS heuristics and additionally run all other LNS heuristics at the root node; we call the corresponding settings *LP based.* In our third experiment, we again use all LNS heuristics and enabled copying the original CIP when creating the subproblem in a LNS heuristic; these settings are called *CIP based.*

Most of the heuristics mentioned above are improvement heuristics. This means they need a feasible solution as starting point. In case of the resource-constrained project scheduling problems we are faced with the problem that in most cases none of the default SCIP root node heuristics produces a feasible solution. Therefore, we used a standard problem specific start heuristic which is based on a fast list scheduling algorithm [18] and is run before the root node is processed. This heuristic has been used in all three runs right in the beginning of root node processing. Hence, the initial situation is the same in all cases.

Tables 1 and 2 state the results for mixed-integer quadratically constrained programs (MIQCP), pseudo-Boolean optimization (PBO), and resource-constrained project scheduling problems (RCPSP). Columns “setting” indicates the used settings. Column “opt” gives the number of instances which were solved to global optimality after the root node. The following two columns display the number of instances where at least one feasible solution was found by one out of the five LNS heuristic discussed in this paper (“sol”) and how often a “best” one. The next three columns compare the special setting results against the default settings. Thereby, the columns “better” and “worse” compare the primal solution (w.r.t. the objective function value) and state the number of instances where a better and a worse primal solution was found. The column “dual” indicates the number of instances the dual bound was better (w.r.t. the default setting). Finally, we state for each setting the shifted geometric mean of the running “time” in seconds. Figure 1 and 2 show for each setting the distribution of the best solutions found among the five LNS heuristics w.r.t. the overall test set size.

As mentioned in Section 2, SCIP calls heuristics sequentially, updating its status in between. In particular, the incumbent solution used by an LNS improvement heuristic – and hence the constructed subproblem – will be different when another (LNS) heuristic finds a different solution beforehand. In an extreme case, a better solution found by one heuristic might lead to a worse overall performance, because another heuristic fails. This explains, why there are a few cases where the solution quality deteriorates

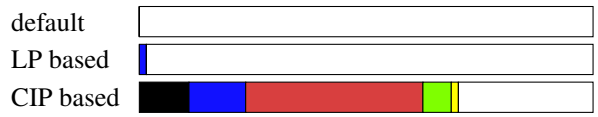
Table 1: The effect of LNS heuristics on mixed-integer quadratically constrained programs (MIQCP) and nonlinear pseudo-Boolean optimization instances (PBO).

(a) 64 mixed-integer quadratically constrained program instances from [7].

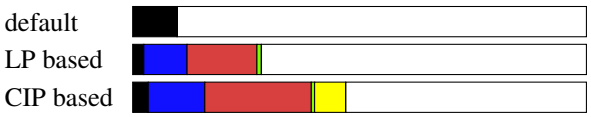
setting	opt	sol	best	better	worse	dual	time
default	2	0	0	–	–	–	4.9
LP based	2	3	1	6	1	0	5.3
CIP based	4	47	45	39	0	3	16.1
best of	4			40		3	

(b) 409 nonlinear pseudo-Boolean optimization instances of the pseudo-Boolean competition 2010 [12].

setting	opt	sol	best	better	worse	dual	time
default	134	43	40	–	–	–	14.4
LP based	139	153	116	102	0	4	15.5
CIP based	150	193	192	174	4	12	15.8
best of	150			179		12	



(a) 64 mixed-integer quadratically constrained program instances from [7].



(b) 409 nonlinear pseudo-Boolean optimization instances of the pseudo-Boolean competition 2010 [12].

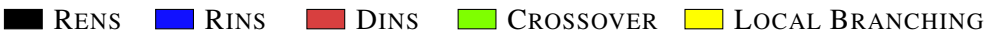


Figure 1: Distribution of best solutions among the five LNS heuristics for the MIQCP and PBO test sets

when using more heuristics.

The shifted geometric mean of values $t_1, \ldots, t_n$ is defined as $\left(\prod(t_i + s)\right)^{1/n} - s$ with shift s. We use a shift of $s = 10$ in order to reduce the effect of very easy instances in the mean values. Further, using a *geometric* mean avoids that hard instances at or close to the time limit have a huge impact on the measures. Thus, the shifted geometric mean has the advantage that it reduces the influence of outliers.

For each experiment we also present a row for the so-called "best of" setting. In this setting one hypothetically selects the best setting out of the three considered ones for each individual instance.

It is notable in the results that the time limit of 24 hours was only hit by one pseudo-Boolean instance (for all settings) and two MIQCP instances (for the CIP based setting). In case of the scheduling test sets we found for all instances a feasible solution independently of the chosen setting. This is due to the use of the problem specific start heuristic. For the nonlinear pseudo-Boolean test set SCIP found a feasible solution for 358 out of 409 instances. Again this is independently of the settings. For the MIQCP test set, however, the default and LP based settings could only show feasibility for 50 instances. The CIP based setting proved feasibility for 6 additional instances.

Applying all LNS heuristics in LP based fashion instead of only the default heuristics lead to almost no improvement for the RCPSP test set, i.e., we improve only on one out of 2040 instances (see Table 2(b)). For the MIQCP test set we achieve a small improvement, that is, for 9% of the instances a better feasible solution is found. Considerable improvements can be observed on the PBO test set. Here, five instances are additionally solved to optimality and for 102 out of 275 instances, that could not be solved to optimality using default settings, a better solution is found. The running time for using all LNS heuristics based on the LP relaxation increases moderately.

The picture changes significantly when the LNS heuristics are applied to the CIP itself. Now, large improvements in the number and quality of feasible solutions can be observed. For 47 (out of 64) MIQCP instances and 194 (out of 409) PBO instances, at least one of the heuristics finds a solution; in 45 and 192 cases, respectively, the best solution is found by an LNS heuristic. For RCPSP, we observe that there are 406 (out of 2040) instances where LNS heuristics produce a feasible solution, in all cases this further is the best solution found during root node processing.

Better solutions are found when compared to default settings for 63% (174 of the 275 instances that have not been solved to optimality by default) of the PBO instances, 62% (3%62) of the MIQCP instances,

Table 2: The effect of LNS heuristics on resource-constrained project scheduling problems (RCPSP) of the PSPLIB [25].

(a) 480 instances with 30 jobs each.

setting	opt	sol	best	better	worse	dual	time
default	278	0	0	–	–	–	0.0
LP based	278	0	0	0	0	0	0.0
CIP based	298	83	83	82	0	7	0.0
best of	298			82		7	

(b) 480 instances with 60 jobs each.

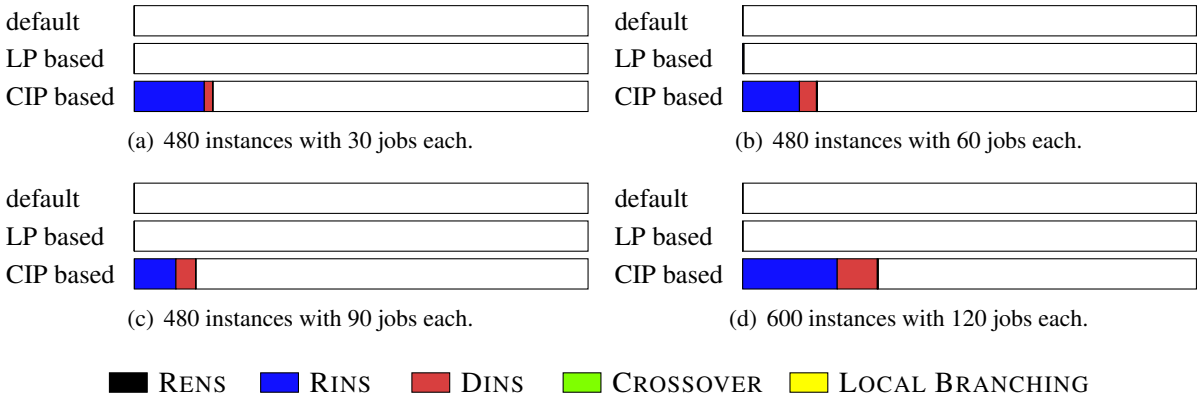
setting	opt	sol	best	better	worse	dual	time
default	294	0	0	–	–	–	0.0
LP based	295	1	1	1	0	0	0.0
CIP based	321	79	79	71	0	6	0.1
best of	321			71		6	

(c) 480 instances with 90 jobs each.

setting	opt	sol	best	better	worse	dual	time
default	319	0	0	–	–	–	0.0
LP based	319	0	0	0	0	0	0.0
CIP based	331	65	65	50	0	2	0.2
best of	331			50		2	

(d) 600 instances with 120 jobs each.

setting	opt	sol	best	better	worse	dual	time
default	120	0	0	–	–	–	0.1
LP based	120	0	0	0	0	0	0.1
CIP based	143	179	179	107	0	7	1.2
best of	143			107		7	



(a) 480 instances with 30 jobs each.

(b) 480 instances with 60 jobs each.

(c) 480 instances with 90 jobs each.

(d) 600 instances with 120 jobs each.

Figure 2: Distribution of best solutions among the five LNS heuristics for the RCPSP of the PSPLIB

and between 33% ($^{50}/_{151}$ for 90 jobs) and 40% ($^{82}/_{202}$ for 30 jobs) of the RCPSP instances. Additionally, more instances are solved to optimality now: two for MIQCP, 16 for PBO, and 82 for RCPSP. The impact to the dual bound after root node processing is small, but mentionable: In each test set there are at least two (and at most twelve) instances for which a better primal bound leads to an improvement of the dual bound.

Further, one observes that the "best of" setting almost always coincides with the CIP based setting, i.e., using the CIP based setting almost never leads to a deterioration in solution quality. There are only five instances of the PBO test set and one of the MIQCP test set for which the LP based version is better than the CIP based. Hence, using a copy of the original CIP rather than a MIP relaxation nearly always seems to be the best choice in terms of number of found solutions and solution quality.

Of course, these improvements may come at the cost of longer running times due to more difficult subproblems in the heuristics. Especially the problem class MIQCP illustrates a main difference between the LP and CIP based settings. While for a MIP one can usually assume that its difficulty is reduced considerably by restricting the integer variables to a neighborhood of some reference point, this is not necessarily need to be the case for every type of CIP. Thus, for MIQCP, the mean running time in the CIP based setting is three times larger than in the LP based setting. On the other hand, for PBO the increase in running time is marginal. For RCPSP, no clear statement is possible; the running times are very small in all cases. Note, however, that the increase in root node processing time is likely to pay

off when solving problems to optimality, since the branch-and-bound search can be initialized with an improved primal bound.

6 Conclusion & Outlook

We proposed a generic way of generalizing large neighborhood search heuristics from mixed-integer programming to constraint integer programming, using MIQCP, PBO, and RCPSP as showcases. The generalization is *not* done in a problem-specific, but in a *generic* way, by using a restricted copy of the full problem for the neighborhood search.

Computational results on mixed-integer quadratically constrained programs, pseudo-Boolean optimization, and resource-constrained project scheduling problems have shown that this straight forward approach already yields considerable improvements for CIPs with nonlinear constraints. For each problem type, the generalized LNS heuristics increased the quality of the best feasible solution after root node computation. These results show a successful application of metaheuristics inside a complete solver.

We plan to investigate the interaction of different LNS heuristics. For MIP, this has partially been done in [5]. Questions to answer are how deactivating a certain LNS heuristic or restricting to a single one influences the performance of SCIP. Further, it would be interesting to see the impact of using LNS heuristics inside other LNS heuristics.

References

[1] Emile Aarts and Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., 1997.

[2] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007.

[3] Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[4] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[5] Timo Berthold. Primal Heuristics for Mixed Integer Programs. Diploma thesis, Technische Universität Berlin, 2006.

[6] Timo Berthold. RENS – relaxation enforced neighborhood search. ZIB-Report 07-28, Zuse Institute Berlin, 2007.

[7] Timo Berthold, Stefan Heinz, Ambros M. Gleixner, and Stefan Vigerske. On the computational impact of MIQCP solver components. ZIB-Report 11-01, Zuse Institute Berlin, 2010.

[8] Timo Berthold, Stefan Heinz, Marco E. Lübbecke, Rolf H. Möhring, and Jens Schulz. A constraint integer programming approach for resource-constrained project scheduling. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *LNCS*, pages 313–317, 2010.

[9] Timo Berthold, Stefan Heinz, and Marc E. Pfetsch. Nonlinear pseudo-boolean optimization: relaxation or propagation? In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing (SAT 2009)*, number 5584 in LNCS, pages 441–446, 2009.

[10] Timo Berthold, Stefan Heinz, and Stefan Vigerske. Extending a CIP framework to solve MIQCPs. In Jon Lee and Sven Leyffer, editors, *Mixed-integer nonlinear optimization: Algorithmic advances and applications*, IMA volumes in Mathematics and its Applications. Springer, to appear.

[11] Andreas Bley, Ambros M. Gleixner, Thorsten Koch, and Stefan Vigerske. Comparing MIQCP solvers to a specialised algorithm for mine production scheduling. ZIB-Report 09-32, Zuse Institute Berlin, October 2009.

[12] Pseudo Boolean Competition 2010. http://www.cril.univ-artois.fr/PB10/.

[13] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming A*, 102(1):71–90, 2004.

[14] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming B*, 98(1-3):23–47, 2003.

[15] Shubhashis Ghosh. DINS, a MIP improvement heuristic. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization (IPCO 2007)*, volume 4513 of *LNCS*, pages 310–323, 2007.

[16] Zeynep Kiziltan, Andrea Lodi, Michela Milano, and Fabio Parisini. CP-based local branching. In Christian Bessière, editor, *Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *LNCS*, pages 847–855. Springer, 2007.

[17] Hans Mittelmann. Mixed integer (QC)QP benchmark, May 2011. http://plato.asu.edu/ftp/miqp.html.

[18] Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003.

[19] Giacomo Nannicini, Pietro Belotti, and Leo Liberti. A local branching heuristic for MINLPs. ArXiv, paper 0812.2188, 2009.

[20] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[21] Yossef Oren, Mario Kirschbaum, Thomas Popp, and Avishai Wool. Algebraic side-channel analysis in the presence of errors. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems (CHES 2010)*, volume 6225 of *LNCS*, pages 428–442, 2010.

[22] Laurent Perron, Paul Shaw, and Vincent Furnon. Constraint integer programming: A new approach tointegrate CP and MIP. In Mark Wallace, editor, *Proc. of CP 2004*, volume 3258 of *LNCS*, pages 468–481. Springer, 2004.

[23] Gilles Pesant and Michel Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.

[24] David Pisinger and Stefan Røpke. Large neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter 13, pages 399–420. Springer, 2nd edition, 2010.

[25] PSPLib. Project Scheduling Problem Library. http://129.187.106.231/psplib/.

[26] Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

[27] SCIP. Solving Constraint Integer Programs. http://scip.zib.de.

[28] Edward M.B. Smith and Constantinos C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimization of nonconvex MINLPs. *Computers & Chemical Engineering*, 23:457–478, 1999.