


GREGOR HENDEL¹

Adaptive Large Neighborhood Search for Mixed Integer Programming

¹  0000-0001-7132-5142

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Adaptive Large Neighborhood Search for Mixed Integer Programming

Gregor Hendel*

December 18, 2018

Abstract

Large Neighborhood Search (LNS) heuristics are among the most powerful but also most expensive heuristics for mixed integer programs (MIP). Ideally, a solver learns adaptively which LNS heuristics work best for the MIP problem at hand in order to concentrate its limited computational budget.

To this end, this work introduces Adaptive Large Neighborhood Search (ALNS) for MIP, a primal heuristic that acts a framework for eight popular LNS heuristics such as Local Branching and Relaxation Induced Neighborhood Search (RINS). We distinguish the available LNS heuristics by their individual search domains, which we call *neighborhoods*. The decision which neighborhood should be executed is guided by selection strategies for the *multi armed bandit problem*, a related optimization problem during which suitable actions have to be chosen to maximize a reward function. In this paper, we propose an LNS-specific reward function to learn to distinguish between the available neighborhoods based on successful calls and failures. A second, algorithmic enhancement is a generic variable fixing prioritization, which ALNS employs to adjust the subproblem complexity as needed. This is particularly useful for some neighborhoods which do not fix variables by themselves. The proposed primal heuristic has been implemented within the MIP solver SCIP. An extensive computational study is conducted to compare different LNS strategies within our ALNS framework on a large set of publicly available MIP instances from the MIPLIB and Coral benchmark sets. The results of this simulation are used to calibrate the parameters of the bandit selection strategies. A second computational experiment shows the computational benefits of the proposed ALNS framework within the MIP solver SCIP.

1 Introduction

Mixed integer programming (MIP) is a powerful modeling paradigm with numerous relevant industrial applications in scheduling, production planning, traffic optimization [8] and countless more. For solving their models, many practitioners rely on state-of-the-art commercial or noncommercial general purpose MIP solvers such as CPLEX [12], XPress [35], SCIP [1, 22], or CBC [10], all of which employ a variant of the branch-and-bound algorithm [13, 29]. As the branch-and-bound algorithm by itself may be quite slow to provide good solutions, a rich set of *primal heuristic algorithms* has been proposed to improve the primal convergence [5] of the solvers. Primal heuristics can be further classified [2] into rounding algorithms, diving and objective

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, hendel@zib.de

diving heuristics and feasibility-pump [2, 18] procedures, and finally Large Neighborhood Search (LNS) heuristics such as *Relaxation Induced Neighborhood Search* (RINS) [14]. LNS heuristics typically restrict the search space of an input MIP instance to a particular neighborhood of interest. The resulting *auxiliary problem* (cf. Definition 1) is again a MIP, which is then partially solved by a branch-and-bound algorithm under reasonable working limits, and eventual solutions are kept for the main search process. Many different LNS techniques have been proposed in recent years [15, 14, 33, 21, 6, 17]. Their computational effort makes it impractical to apply all of them frequently within the solver. Since a priori, it is unclear which approach might work best for a concrete problem instance, the solver ideally learns during the solving process which LNS heuristics should be applied, and more importantly, which ones can be deactivated.

Following this line of thought, we propose Adaptive Large Neighborhood Search (ALNS) for MIP. We address in particular the question how to select from the set of available neighborhoods, which are introduced in Section 2. In Section 3, we propose a suitable reward function for LNS heuristics to learn to discriminate between the neighborhoods during the search. We also propose a generic variable fixing scheme that can be used to extend the set of fixed variables within a selected neighborhood to reach a target fixing rate. This has a particular impact on LNS heuristics that do not fix variables by themselves and may hence be too expensive on larger problems, such as, e.g., Local Branching [15] (cf. Section 2.2).

The framework is obliged to trade off between exploration and exploitation, because only one neighborhood is selected and evaluated at a specific call. Such a selection scenario is also referred to as *multi armed bandit problem*, in which a player tries to maximize their reward by playing one available action at a time and observing the particular reward of this action only. We review three selection algorithms for the multi-armed bandit problem in Section 4. Two numerical experiments are presented in Section 5, a first one to tune the selection process of the ALNS heuristic, and a second experiment to show that ALNS improves the MIP performance of SCIP on a large set of publicly available benchmark instances from the collections of MIPLIB [7, 3, 27] and Cor@1 [11].

1.1 Related Work

The notion of an Adaptive Large Neighborhood Search has already been coined in the literature, particularly in the context of Constraint Programming, where ALNS is usually tailored to a particular application. The authors [28] were the first to describe an adaptive LNS technique for single-mode scheduling problems, which selects from a finite set of so-called search operators, which are a CP analogue to the neighborhoods for general MIP (see Section 2). Building upon their method, ALNS has also been applied for different types of Vehicle Routing Problems, see [32] for an overview. Throughout the remainder of this article, we will shortly write "ALNS" to denote our proposed "ALNS for MIP".

A different, MIP specific approach to learn how to run heuristics has been recently proposed in [26]. Their work uses logistic regression to predict the probability of success for different diving heuristics. The prediction is based on state information about the current node and the overall search. Their approach is fundamentally different from our proposed method in that it learns one regression for each individual diving heuristic, but does not attempt to prioritize between them.

The first use of bandit related ideas inside MIP solvers [34] concerns the integration of a node selection rule into CPLEX. This node selection approach balances exploration and exploitation of the solution process inspired by a successful method for game search trees, which is related to the Upper Confidence Bounds selection algorithm [9], which is explained in Section 4.1.

2 Large Neighborhood Search Heuristics for MIP

We propose Adaptive Large Neighborhood Search in the context of mixed-integer programs (MIPs). A MIP P is an optimization problem of the form

$$\begin{aligned}
 \min \quad & c^T x \\
 \text{s.t.} \quad & Ax \geq b \\
 & l \leq x \leq u \\
 & x \in \{0, 1\}^{n_b} \times \mathbb{Z}^{n_i - n_b} \times \mathbb{Q}^{n - n_i}
 \end{aligned} \tag{MIP}$$

in n variables and m linear constraints, which are defined by a matrix $A \in \mathbb{Q}^{m,n}$ and a *right hand side* $b \in \mathbb{Q}^m$. Every variable x_j , $j \in \{1, \dots, n\}$ has an *objective coefficient* $c_j \in \mathbb{Q}$ and a *lower and upper bound* denoted by $l_j \in \mathbb{Q} \cup \{-\infty\}$ and $u_j \in \mathbb{Q} \cup \{+\infty\}$, respectively. Finally, without loss of generality, the first n_i variables are further constrained that they must take integer solution values. These are called the *integer variables* of P . An important subset of the integer variables are the $n_b \leq n_i$ *binary variables* with a $\{0, 1\}$ -domain. The shorthand notations for binary variables and nonbinary integer variables, which are called *general integer variables*, are $N_b = \{1, \dots, n_b\}$ and $N_i = \{n_b + 1, \dots, n_i\}$, respectively. Binary variables are often used to model highly relevant yes/no-decisions in an optimization scenario such as, e.g., if a facility should be built at a certain location. Therefore, binary variables often receive a prioritized treatment by the neighborhoods in Sections 2.1 and 2.2. Every point of \mathbb{Q}^n satisfying all of the above constraints is called a *solution* of P , and the set of all solutions of P is denoted by \mathcal{S}_P .

Dropping all integrality restrictions yields the *LP relaxation* of P . It is well known that (MIP) in the presented general form is \mathcal{NP} -hard to solve, which is why all modern MIP solvers employ some form of *branch-and-bound* algorithm [29, 13]. In essence a clever enumeration, the branch-and-bound algorithm repeatedly partitions the search space of an input MIP P , mainly guided by integer variables with noninteger (fractional) values in the solution x^{lp} to its LP relaxation. Since the LP relaxation has fewer constraints than P and hence a broader feasible region, its optimal objective value $c^T x^{\text{lp}}$ is a lower bound to the *optimal value* c^* of P . If, in addition, the LP solution satisfies the integrality requirements $x_j^{\text{lp}} \in \mathbb{Z} \forall j \in \{1, \dots, n_i\}$, then $c^T x^{\text{lp}} = c^*$ and x^{lp} is an optimal solution for P . The minimum lower bound of all unprocessed subproblems is called the *dual bound* and denoted by c^{dual} .

In practice, however, the LP relaxation mostly provides feasible solutions only at deeper levels, ie. later stages of the search tree. Many different primal heuristic algorithms have been proposed to overcome this weakness, which are highly diverse in the computational effort they require. Starting from simple and fast heuristics [24] that attempt to construct feasible solutions by rounding the LP solution, a higher computational effort is usually required by diving or feasibility-pump [18] like procedures, which solve modified LP relaxations. At the most expensive end of the scale lies the

class of *Large Neighborhood Search* (LNS) heuristics that solve an *auxiliary problem* with branch-and-bound techniques.

Definition 1 (Auxiliary problem) *Let P be a MIP with n variables. For a polyhedron $\mathcal{N} \subseteq \mathbb{Q}^n$ and objective coefficients $c_{aux} \in \mathbb{Q}^n$, a MIP P^{aux} defined as*

$$\min \{c_{aux}^T x \mid x \in \mathcal{S}_P \cap \mathcal{N}\} \quad (1)$$

is called an auxiliary problem of P . The polyhedron \mathcal{N} is called neighborhood.

In other words, P^{aux} has the same number of variables (columns) as the original MIP P . Its solution set is a subset of \mathcal{S}_P . Hence, every solution to P^{aux} is a solution for P . Definition 1 requires \mathcal{N} to be a polyhedron, ie., it should be expressed by a finite set of inequalities. The definition includes $\mathcal{N} = \mathbb{Q}^n$. Each neighborhood has its associated auxiliary objective function, which can be different from the main objective function of P .

There are only a few different types of neighborhoods typically used. All LNS heuristics have in common that they solve auxiliary problems around a set of reference points to either provide a first solution or, as in most cases, an improvement to the current incumbent solution. One of the most common classes of neighborhoods is derived by considering a set of reference points and fixing integer variables whose values are integer and agree on all those points.

Definition 2 (Fixing neighborhood) *Let P be a MIP with n variables and $n_i \leq n$ integer variables. Let $\mathcal{M} \subseteq \{1, \dots, n_i\}$ and $x^* \in \mathbb{Q}^n$. A fixing neighborhood*

$$\mathcal{N}^{fix}(\mathcal{M}, x^*) := \{x \in \mathbb{Q}^n \mid x_j = x_j^* \forall j \in \mathcal{M}\}$$

fixes the subset \mathcal{M} to their values in x^ .*

Definition 3 (Matching set) *For $k \geq 1$, let $X = \{x^1, \dots, x^k\} \subset \mathbb{Q}^n$ with $x^i \neq x^{i'} \forall i \neq i' \in \{1, \dots, k\}$. The matching set*

$$\mathcal{M}^=(X) := \{j \in \{1, \dots, n_i\} \mid x_j^i = x_j^1 \forall i \in \{1, \dots, k\}\}$$

describes all integer variable indices whose values agree on X . We call X the set of reference points.

Combining Definitions 2 and 3 is very popular for constructing auxiliary problems. Starting from a set of reference points $X = \{x^1, \dots, x^k\}$, a fixing neighborhood is obtained with the help of the matching set of X . Since all points in X agree on their matching set $\mathcal{M}^=(X)$, the same fixing neighborhood is obtained regardless of the anchor point

$$\mathcal{N}^{fix}(\mathcal{M}^=(X), x^1) = \mathcal{N}^{fix}(\mathcal{M}^=(X), x^i) \quad \forall i \in \{1, \dots, k\}.$$

It should be noted that whenever a set of reference points X contains at least one solution $x \in \mathcal{S}_P$, the auxiliary MIP defined by the fixing neighborhood of the matching set is feasible because $x \in \mathcal{S}_P \cap \mathcal{N}^{fix}(\mathcal{M}^=(X), x)$.

The task of finding an improving solution can be easily incorporated into Definition 1. Assume that an incumbent solution $x^{inc} \in \mathcal{S}_P$ and a dual bound c^{dual} are

already available, and let a given neighborhood $\mathcal{N} \subseteq \mathbb{Q}^n$ contain some nonimproving solutions. For $\delta \in (0, 1)$, every solution $x \in \mathcal{S}_P$ that satisfies

$$c^T x \leq (1 - \delta) \cdot c^T x^{\text{inc}} + \delta \cdot \underbrace{c^{\text{dual}}}_{< c^T x^{\text{inc}}} < c^T x^{\text{inc}}$$

is clearly an improving solution. The set of solutions that are better than x^{inc} by at least δ is contained in the *improvement neighborhood*

$$\mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}}) := \left\{ x \in \mathbb{Q}^n \mid c^T x \leq (1 - \delta) \cdot c^T x^{\text{inc}} + \delta \cdot c^{\text{dual}} \right\}.$$

Therefore, refining the original neighborhood as

$$\mathcal{N}' = \mathcal{N} \cap \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}})$$

filters out all nonimproving solutions, regardless of the choice of c_{aux} . The choice of δ is an important control parameter to weigh between the difficulty (and feasibility) of the sub-MIP and the desired amount of improvement. The above neighborhood notions suffice to describe several popular LNS heuristics.

2.1 Fixing Neighborhood LNS Heuristics

RINS [14] *Relaxation Induced Neighborhood Search* (RINS) is one of the first LNS approaches. The idea of RINS is to fix integer variables whose solution values agree in the solution x^{lp} of the LP relaxation at the current, local node, and the current incumbent solution x^{inc} . After the choice of a suitable improvement δ , the neighborhood of the auxiliary MIP of RINS is

$$\mathcal{N}_{\text{RINS}} := \mathcal{N}^{\text{fix}}(\mathcal{M} = (\{x^{\text{lp}}, x^{\text{inc}}\}), x^{\text{inc}}) \cap \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}}).$$

Crossover [33] Another improvement heuristic is the *Crossover* heuristic, which is inspired by the recombination of solutions within genetic algorithms. Crossover selects $k \geq 2$ already known, feasible solutions $X = \{x^1, \dots, x^k\} \subseteq \mathcal{S}_P$ as reference points. X does not necessarily contain x^{inc} . The *crossover neighborhood* fixes

$$\mathcal{N}_{\text{Cross}} := \mathcal{N}^{\text{fix}}(\mathcal{M} = (X), x^1) \cap \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}}).$$

The authors suggest to use $k = 2$ solutions that are randomly selected from all available solutions, using a bias towards solutions with better objective.

Mutation [33] Furthermore, the authors suggest a second LNS heuristic called *Mutation* that fixes a random subset of integer variables of the incumbent solution. For a randomly chosen subset $\mathcal{M}^{\text{rand}} \subseteq N_b \cup N_i$, the *mutation neighborhood* is defined as

$$\mathcal{N}_{\text{Muta}} := \mathcal{N}^{\text{fix}}(\mathcal{M}^{\text{rand}}, x^{\text{inc}}) \cap \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}}).$$

In order to control the difficulty of the sub-MIP, the obvious input to the mutation neighborhood is a number or percentage of integer variables that should be fixed. Mutation is the only neighborhood for which this number can be controlled directly: all previous neighborhoods depend on the cardinality of their matching set.

RENS [6] Starting from an LP relaxation solution x^{lp} , the *Relaxation Enforced Neighborhood Search* (RENS) neighborhood focusses on the feasible roundings of x^{lp} and can be written as

$$\mathcal{N}_{\text{RENS}} := \left\{ x \in \mathbb{Q}^n \mid \lfloor x_j^{\text{lp}} \rfloor \leq x_j \leq \lceil x_j^{\text{lp}} \rceil, j \in \{1, \dots, n_i\} \right\}.$$

Similarly to the RINS heuristic, the aim of RENS is to construct feasible solutions that are close to the LP relaxation solution and therefore have a near-optimal solution value.

2.2 LNS Heuristics Using Constraints and Auxiliary Objective Functions

All approaches presented so far fix a set of integer variables using one or several reference points. *Local Branching* [15] is the first LNS heuristic that uses a different neighborhood.

Local Branching [15] Instead of fixing a set of variables and solving for improving solution values on the remaining variables, the neighborhood of Local Branching is restricted to a ball around the current incumbent solution. More formally, Let P be a MIP with $n_b \geq 1$ binary variables. Based on the L_1 -norm or Manhattan metric for $x \in \mathbb{Q}^n$, the *binary norm*¹ of x is defined as

$$\|x\|_b := \sum_{j=1}^{n_b} |x_j|.$$

Let $x^{\text{inc}} \in \mathcal{S}_P$ be an incumbent solution for P , and let $d_{\text{max}} > 0$ denote a distance cutoff parameter. The *local branching neighborhood* is the restriction

$$\mathcal{N}_{\text{LBranch}} := \left\{ x \in \mathbb{Q}^n \mid \|x - x^{\text{inc}}\|_b \leq d_{\text{max}} \right\} \cap \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}})$$

The reason for preferring the binary norm over the regular norm or the norm taking all integer variables is practicality. The binary norm can be expressed as a linear constraint without introducing auxiliary variables.

Proximity Search [17] A dual version of Local Branching has been introduced as *Proximity Search*. Using the binary norm, Proximity seeks to minimize the binary norm $\|x - x^{\text{inc}}\|_b$ through an auxiliary objective coefficient vector $c_{\text{Proxi}} = c_{\text{Proxi}}(x^{\text{inc}})$ defined as

$$(c_{\text{Proxi}})_j := \begin{cases} 0 & \text{if } j > n_b \\ 1 & \text{if } x_j^{\text{inc}} = 0 \\ -1 & \text{if } x_j^{\text{inc}} = 1 \end{cases}$$

over the entire set of improving solutions:

$$\mathcal{N}_{\text{Proxi}} := \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}}).$$

¹Strictly speaking, $\|\cdot\|_b$ is a seminorm because nonzero vectors can have binary norm of 0.

Zero Objective The second LNS heuristic that uses an auxiliary objective function different from the original objective function is *Zero Objective*. If an incumbent solution x^{inc} is available, Zero Objective searches the entire set of improving solutions $\mathcal{N}_{\text{Zeroobj}} := \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}})$ but uses $c_{\text{Zeroobj}} := 0$ as auxiliary objective function. If no incumbent is available, $\mathcal{N}_{\text{Zeroobj}} = \mathbb{Q}^n$. Zero Objective thereby reduces the search for an (improving) solution to a feasibility problem.

DINS [21] Distance Induced Neighborhood Search (DINS) combines elements of the Crossover, Local Branching, and RINS heuristics. Similarly to RINS, the intuition is that improving solutions are located between the current incumbent solution x^{inc} and the solution to the node LP relaxation x^{lp} . With the intention of reducing the *integer distance*

$$\left\| x^{\text{inc}} - x^{\text{lp}} \right\|_i := \sum_{j=1}^{n_i} |x_j^{\text{inc}} - x_j^{\text{lp}}|,$$

let $J := \{j \mid |x_j^{\text{inc}} - x_j^{\text{lp}}| \geq 0.5\} \subseteq N_i$ denote the index set of general integer variables with a difference of at least 0.5 between the two reference points. The J -neighborhood of DINS is

$$\mathcal{N}_J := \{x \in \mathbb{Q}^n \mid |x_j - x_j^{\text{lp}}| \leq |x_j^{\text{inc}} - x_j^{\text{lp}}|, j \in J\}.$$

This neighborhood restricts lower and upper bounds of the general integer variables. Let $\mathcal{T} \subseteq \mathcal{S}_P$ denote a subset of currently available solutions, containing x^{inc} , to the MIP at hand. The DINS neighborhood can be written as a combination of a total of five neighborhoods

$$\begin{aligned} \mathcal{N}_{\text{DINS}} := & \mathcal{N}_J \\ & \cap \mathcal{N}^{\text{fix}}(N_i \setminus J, x^{\text{inc}}) \\ & \cap \mathcal{N}^{\text{fix}}(N_b \cap \mathcal{M} = (\{x^{\text{lp}}, x^{\text{root}}\} \cup \mathcal{T}), x^{\text{inc}}) \\ & \cap \mathcal{N}_{\text{LBranch}} \\ & \cap \mathcal{N}^{\text{obj}}(\delta, x^{\text{inc}}). \end{aligned}$$

The set of general integer variables outside of J is fixed to the values in the incumbent solution. Binary variables that have not changed between the root LP relaxation solution x^{root} and x^{lp} are fixed if they have taken the same value in all solutions in \mathcal{T} . Finally, the search is further restricted to a certain binary distance around the current incumbent solution through an additional local branching neighborhood. Among the possible choices for \mathcal{T} , the implementation of DINS for this work uses a set of up to five available solutions with best objective.

There is further work on LNS approaches that are not covered here. Note that the only heuristics that do not use an incumbent solution are RENS and Zero Objective. In [16], an extension of Local Branching has been proposed that starts from an infeasible reference point. Such points are quickly produced by rounding or with a few iterations of the Feasibility Pump [18]. In addition to the local branching constraint, the auxiliary problem of [16] is extended by additional variables to model and penalize the violation of constraints, inspired by the phase 1 of the Simplex algorithm. A recent approach called Alternating Criteria Search [31] also starts from infeasible reference points, and alternates between auxiliary problems with artificial feasibility objective and the original objective function of the input MIP in a parallel setting. The necessary

diversification is obtained by fixing subsets of integer variables indexed by a random consecutive index set. Such a fixing scheme is a variant of Mutation [33] discussed in Section 2.1. The heuristics presented in [19] formulate and solve auxiliary problems only for the set N_i of general integer variables as a final post processing step after fixing all binary variables based on available, global problem structures such as cliques and implications between binary and integer variables.

3 Adaptive Large Neighborhood Search for MIP

The proposed Adaptive Large Neighborhood Search heuristic has as input the set of 8 available neighborhoods from Section 2 denoted by \mathcal{H} (the set of actions). Table 1 gives a quick overview of the neighborhoods used, as well as their individual preconditions. In each call to the heuristic, ALNS basically performs the following steps.

1. Select a promising neighborhood $h_t \in \mathcal{H}$ and its associated auxiliary objective function $c_{\text{aux},t}$ via a *bandit selection strategy*.
2. Setup and solve the auxiliary problem given by $(h_t, c_{\text{aux},t})$.
3. Reward the neighborhood and update the bandit selection strategy for the next selection.

Different bandit selection strategies and their individual update procedures are subject of Section 4. The solution process of the auxiliary problem uses a strict limit on the number of branch-and-bound nodes to terminate the subproblem quickly and keep the overall computational effort small. It may still be very expensive to solve auxiliary problems if the neighborhood is large, especially since some neighborhoods do not fix integer variables directly. In Section 3.1, a generic approach is explained for fixing additional variables to reach any desired target fixing rate and hence reduce the subproblem complexity. Details on the dynamic adjustment of the target fixing rate and node limit are given in Section 3.2. Finally, Section 3.3 introduces the scoring mechanism to reward the selected neighborhood. Note that this work refers to the proposed scoring mechanism as "reward", but does not mention "penalties". Penalties are readily obtained by assigning small rewards.

3.1 Fixing and Unfixing Variables

Every neighborhood in Section 2 fixes a subset of integer variables to the values of a solution $x^{\text{sol}} \in \mathcal{S}_P$. The *fixed set* of a neighborhood \mathcal{N} is defined as

$$\mathcal{M}^{\text{fix}} := \left\{ j \in \{1, \dots, n_i\} \mid \mathcal{N} \subseteq \left\{ x \in \mathbb{Q}^n \mid x_j = x_j^{\text{sol}} \right\} \right\}.$$

The size of the fixed set is denoted by $n^{\text{fix}} = |\mathcal{M}^{\text{fix}}|$. Every neighborhood (action) $h \in \mathcal{H}$ operated by ALNS has a *target fixing rate* $\phi_{h,t} \in [0, 1)$ that changes over time as will be explained in Section 3.2. It may happen that a selected neighborhood h_t does not reach its specified target fixing rate, ie.

$$n^{\text{fix}} < \phi_{h_t,t} \cdot n_i,$$

or that it fixes much more integer variables and hence restricts the search space more than necessary. ALNS treats both cases very similarly by using a generic variable fixing prioritization to sort the set of possible (un)fixings.

Table 1: Overview of the neighborhoods that are used in ALNS. See Section 2 for information and references.

Neighborhood	Description	Preconditions
RINS	Fixes matching values in incumbent and LP relaxation solution	Feasible LP relaxation at current node, incumbent solution
Crossover	Fixes matching values in 2 or more randomly chosen, available solutions	Sufficiently many solutions
Mutation	Fixes random subset of variables to values in incumbent solution	Incumbent solution
RENS	Restricts the auxiliary problem to the feasible roundings around the current LP relaxation solution	Feasible LP relaxation at current node
Local Branching	Limits the maximal binary distance from the incumbent solution	Incumbent solution, MIP with binary variables
Proximity Search	Finds an improving solution that minimizes the binary distance from the incumbent	Incumbent solution, MIP with binary variables
Zero Objective	Reduces search for an (improving) solution to a feasibility problem	MIP with nonzero objective function
DINS	Sophisticated combination of RINS, Crossover, and Local Branching	Incumbent solution

In the first case, $\phi_{h_i,t} \cdot n_i - n^{\text{fix}}$ additional integer variables from $\overline{\mathcal{M}^{\text{fix}}} = \{1, \dots, n_i\} \setminus \mathcal{M}^{\text{fix}}$ have to be selected. For two variables $x_j, x_{j'} \in \overline{\mathcal{M}^{\text{fix}}}$ with reference solution values $x_j^{\text{sol}}, x_{j'}^{\text{sol}}$, the fixing $x_j = x_j^{\text{sol}}$ is preferred over $x_{j'} = x_{j'}^{\text{sol}}$, if, in decreasing order of priority,

1. x_j has a smaller distance than $x_{j'}$ from \mathcal{M}^{fix} in the *variable constraint graph* (see below).
2. The reduced costs for fixing $x_j = x_j^{\text{sol}}$,

$$c_j^{\text{red}} \cdot (x_j^{\text{sol}} - x_j^{\text{root}_j}) < c_{j'}^{\text{red}} \cdot (x_{j'}^{\text{sol}} - x_{j'}^{\text{root}_{j'}})$$

are smaller than those for fixing $x_{j'} = x_{j'}^{\text{sol}}$.

3. The *pseudo costs* (see below) for fixing $x_j = x_j^{\text{sol}}$ are smaller than for $x_{j'} = x_{j'}^{\text{sol}}$,

$$\Psi_j (x_j^{\text{sol}} - x_j^{\text{root}_j}) < \Psi_{j'} (x_{j'}^{\text{sol}} - x_{j'}^{\text{root}_{j'}}).$$

4. Randomly.

The rationale behind this variable prioritization is to first keep the auxiliary problem connected by considering the *variable constraint graph*. For a given MIP P , the variable constraint graph G_P has the variables and constraints $\{v_j \mid j \in \{1, \dots, n\}\} \cup \{w_j \mid j \in \{1, \dots, m\}\}$ as nodes. The edges $E(G_P)$ correspond to the nonzero entries of the matrix A :

$$E(G_P) = \{(v_j, w_i) \mid a_{ij} \neq 0\}.$$

Distances in G_P are breadth first distances. If the original problem has a block structure, the variable prioritization concentrates additional fixings on those blocks with a nonempty intersection in \mathcal{M}^{fix} . Related ideas are used, e.g., in presolving for detecting independent subproblems [20], or within the *Graph-Induced Neighborhood Search* (GINS) released with SCIP 4.0 [30].

The cost based scores in steps 2 and 3 both penalize a deviation of the potential fixing from an LP solution at the root node of the branch-and-bound search. The first associated penalty uses reduced costs. Reduced costs are recorded after every solved LP at the root node of the solving process. If they are higher than any previously recorded reduced costs for x_j , they are stored together with the corresponding LP solution value $x_j^{\text{root}_j}$. Therefore, the LP solution values used to compare the potential fixings of x_j and $x_{j'}$ may come from different LP solutions.

The pseudo-cost score treats the fixing as a branching and estimates the potential gain in the dual bound. Both variables are compared on the same, namely the final root LP solution on which the solution process started branching. *Pseudo costs* [4] are a common aggregate of branching information on the variables. Given the average increase Ψ_j^+ (Ψ_j^-) per unit fractionality of the dual bound after branching upwards (downwards) on a variable x_j for $j \in \{1, \dots, n_i\}$, the pseudo costs of j are a function

$$\Psi_j : \mathbb{Q} \rightarrow \mathbb{Q}_{\geq 0} \\ z \mapsto \begin{cases} \Psi_j^+ \cdot z, & \text{if } z \geq 0 \\ -\Psi_j^- \cdot z, & \text{if } z < 0. \end{cases} \quad (2)$$

As an example, assume that a binary variable x_j has an LP solution value $x_j^{\text{lp}} = 0.4$ and an incumbent solution value of 1. Assume that the average dual bound increase has been $\Psi_j^- = 10$ for branching down on x_j and $\Psi_j^+ = 5$ for branching up. The pseudo costs for the fixing $x_j = x_j^{\text{sol}}$ can be calculated as $\Psi_j(x_j^{\text{sol}} - x_j^{\text{lp}}) = \Psi_j^+ \cdot 0.6 = 3$. If the solution value had been 0 instead of one, the corresponding pseudo cost score is $\Psi_j(0 - x_j^{\text{lp}}) = -\Psi_j^- \cdot (-0.4) = 4$ for branching down on x_j .

Only slight details are changed if the neighborhood was too restrictive, such that $n^{\text{fix}} - \phi_{h,t} \cdot n_i$ variables from \mathcal{M}^{fix} should be selected and unfixed (relaxed). Distances are now computed in the variable constraint graph starting from $\overline{\mathcal{M}^{\text{fix}}}$, and variables with a small distance are preferably relaxed to keep the auxiliary problem connected. Similarly to before, variables are relaxed preferably if they have a large reduced cost score or, in case a tie occurs, a large pseudo cost score. Finally, if none of the scores discriminate between two variables, the preference is given by a random score assigned to each variable. Generic (un-)fixing within ALNS is only applied if the target fixing rate $\phi_{h,t}$ is missed by a tolerance of 10 %, i.e. only if $n^{\text{fix}} \notin [(\phi_{h,t} - 0.1) \cdot n_i, (\phi_{h,t} + 0.1) \cdot n_i]$.

3.2 Dynamic Limits

Good limits on the computational budget of an LNS heuristic are essential to make it useful inside a MIP solver. To this end, a tradeoff must be made between the intensity of the search inside the auxiliary problem and the runtime. For this work, the complexity of the subproblem and the budget are adapted dynamically between the individual calls to ALNS.

All of the following dynamic decisions consider the auxiliary problem $P^{\text{aux}}(t)$ at the t -th round of ALNS ($t = 0, 1, \dots$) and its *solution status* $s(P^{\text{aux}}(t))$ which can be one of

- *inf*, if $P^{\text{aux}}(t)$ was infeasible,
- *opt*, if $P^{\text{aux}}(t)$ was solved to optimality
- *sol*, if $P^{\text{aux}}(t)$ provided an improving solution for P
- *nosol*, if no improving solution was found searching $P^{\text{aux}}(t)$.

Target fixing rate $\phi_{h,t}$ The first dynamic adjustment of the auxiliary problem complexity over time is described in [33], together with the introduction of the Crossover and Mutation LNS heuristics (cf. Section 2), which have been implemented as neighborhoods available to ALNS. The authors of [33] control the complexity of an auxiliary problem $P^{\text{aux}}(t)$ by specifying the amount of integer variables that should be fixed by a neighborhood. The intuition is that the complexity of $P^{\text{aux}}(t)$ decreases with increasing fixing rate. In the notation of the present work, the amount of fixed integer variables is specified by means of the target fixing rate $\phi_{h,t} \in [0, 1)$ of a neighborhood $h \in \mathcal{H}$ during round t of ALNS.

For $h \in \mathcal{H}$, let $T_h(t)$ denote the number of times that h has been selected, including round t . The fixing rate is modified according to the status in round t as

$$\phi_{h,t+1} = \begin{cases} \phi_{h,t}, & \text{if } h \neq h_t \text{ or } s(P^{\text{aux}}(t)) = \text{sol}, \\ \max\{0.1, \phi_{h,t} - 0.75^{T_h(t)} \cdot 0.2\}, & \text{if } s(P^{\text{aux}}(t)) \in \{\text{inf}, \text{opt}\} \\ \min\{0.9, \phi_{h,t} + 0.75^{T_h(t)} \cdot 0.2\} & \text{if } s(P^{\text{aux}}(t)) = \text{nosol} \end{cases}$$

If $P^{\text{aux}}(t)$ was too easy for the solver, ie. it could be solved to optimality or infeasibility within a given node budget, the fixing rate for the next iteration is decreased. If no new solution was found, the target fixing rate is increased. If a solution was found within the node limit, but the search could not be completed, the fixing rate is kept. The additive change of the fixing rate is 0.2 initially, which is multiplied with 0.75 after every update step, exactly as in [33]. The use of max and min ensures that the target fixing rate stays within 10 % and 90 %. In our implementation, those two values are parametrized and can be individually set for every neighborhood. Every neighborhood h is initialized with a conservative target fixing rate of $\phi_{h,1} = 0.9$ (respectively the user-defined maximum value), see Section 5 for details.

Node limit v_t^{lim} The main budget limitation of ALNS is a limit on the number of branch-and-bound nodes. The node limit v_{t+1}^{lim} for the next round of ALNS is adjusted based on the results of the auxiliary problem of round t as follows.

$$v_{t+1}^{\text{lim}} = \begin{cases} v_t^{\text{lim}}, & \text{if } s(P^{\text{aux}}(t)) \in \{\text{opt}, \text{inf}, \text{sol}\} \\ \min\{\lfloor v_t^{\text{lim}} \cdot 1.05 \rfloor + 1, 5000\}, & \text{if } s(P^{\text{aux}}(t)) = \text{nosol} \end{cases}$$

Similarly to other LNS heuristics in SCIP, ALNS uses an affine linear function of the branch-and-bound nodes v^{bb} in the main search to limit the search effort inside auxiliary problems. Concretely, the next round t of ALNS is called as soon as the main search has made sufficient search progress such that

$$\kappa_0 + \frac{u(t-1)+1}{(t-1)+1} \cdot \kappa_1 \cdot v^{\text{bb}} - \sum_{i=1}^{t-1} (100 + v_i) \geq v_t^{\text{lim}}. \quad (3)$$

Here, κ_0 and κ_1 are positive coefficients, and $u(t-1)$ denotes the total number of improving solutions found by the end of round $t-1$. In 3, the linear growth of the budget is increased or decreased based on the total number of improving solutions that ALNS provided. With this strategy, ALNS slowly fades out if it does not find improving solutions. The last term expresses the total node budget used so far with an additional 100 nodes to account for the setup costs of $P^{\text{aux}}(i)$.

3.3 A Reward Function for Neighborhoods

All of the bandit selection strategies presented in Section 4 require the definition of a suitable reward function. Intuitively, the reward should always be higher for neighborhoods that improve over the current incumbent solution, but also depend on the achieved objective quality. Furthermore, a neighborhood that failed fast should be rewarded higher than an unsuccessful neighborhood whose execution required more of the budget. In order for some of the selection strategies from Section 4 to work correctly, we require that a reward should be in the interval $[0, 1]$. A reward of 0 is the worst possible score, i.e., the maximum penalty.

Let $h_t \in \mathcal{H}$ denote the selected neighborhood in round $t > 0$, and let $c^{\text{old}} := c^T x^{\text{inc}}$ denote the objective before the execution of h_t , if an incumbent solution $x^{\text{inc}} \in \mathcal{S}_P$ is available, or $c^{\text{old}} := \infty$, otherwise. Similarly, c^{new} is the objective of the best known solution after running h_t . Further, let v_t^{lim} and v_t denote the node limit and amount of nodes used by h_t , respectively, and let $n^{\text{fix}}(t)$ denote the number of integer variables fixed by h_t .

Two reward functions are combined to reward both the presence of a new incumbent solution and the objective improvement. The former is expressed by the *solution reward*

$$r^{\text{sol}}(h_t, t) := \begin{cases} 1, & \text{if } s(P^{\text{aux}}(t)) \in \{\text{opt}, \text{sol}\}, \\ 0, & \text{else.} \end{cases}$$

The improvement in solution quality is measured by the *closed gap reward*

$$r^{\text{gap}}(h_t, t) := \frac{c^{\text{old}} - c^{\text{new}}}{c^{\text{old}} - c^{\text{dual}}},$$

which evaluates to 0 if no improving solution could be found, and is 1 if the new solution has an objective that is equal to the dual bound, i.e. that is optimal for P . As a convention, the closed gap reward is 1 if the neighborhood could contribute the first solution to the problem. Since most neighborhoods require a known solution as input (cf. 1), this is only possible with RENS and Zero Objective.

Since the time measurement in SCIP is not deterministic, we approximate the *effort* $\xi(t)$ spent on the search of the auxiliary problem by using the number of nodes as

$$\xi(t) = \left(1 - \frac{n^{\text{fix}}(t)}{n_i}\right) \frac{v_t}{v_t^{\text{lim}}}.$$

In order to compensate for different target fixing rates between different heuristics, $\xi(t)$ uses a scaling by the remaining number of free integer variables. The effort lies in the interval $[0, 1]$. It is $\xi(t) = 1$ if and only if the node limit was exhausted ($v_t = v_t^{\text{lim}}$) and no integer variables were fixed by the neighborhood. We propose to use the effort in two ways. If a neighborhood fails to produce a better solution, the last of the three individual reward functions is the *failure reward*

$$r^{\text{fail}}(h_t, t) := \begin{cases} 1, & \text{if } s(P^{\text{aux}}(t)) \in \{\text{opt}, \text{sol}\}, \\ 1 - \xi(t), & \text{else,} \end{cases}$$

which uses the effort directly and becomes smaller depending on the effort spent in an auxiliary problem, if no improving solution was found. With two additional convex combination parameters $\eta_1, \eta_2 \in [0, 1]$, the reward function of ALNS combines all three rewards as

$$r^{\text{alns}}(h_t, t) := \eta_1 r^{\text{fail}}(h_t, t) + (1 - \eta_1) \cdot \frac{\eta_2 r^{\text{sol}}(h_t, t) + (1 - \eta_2) r^{\text{gap}}(h_t, t)}{1 + \xi(t)} \quad (4)$$

The first control parameter η_1 separates the reward between runs that were successful and runs that failed to improve the incumbent solution. The second parameter η_2 adjusts between the solution and the closed gap rewards. The result, which is again a reward in the interval $[0, 1]$, is scaled by the effort involved to reward fast neighborhoods higher. Figure 1 depicts the individual elements of the ALNS reward function visually. For the computational experiments, $\eta_1 = 0.5$ and $\eta_2 = 0.8$ were used.

4 Selection Strategies for Multi Armed Bandit Problems

The goal of the present work is a framework that selects among the LNS heuristics presented in Section 2 and that tries to maximize their utility under a shared computing

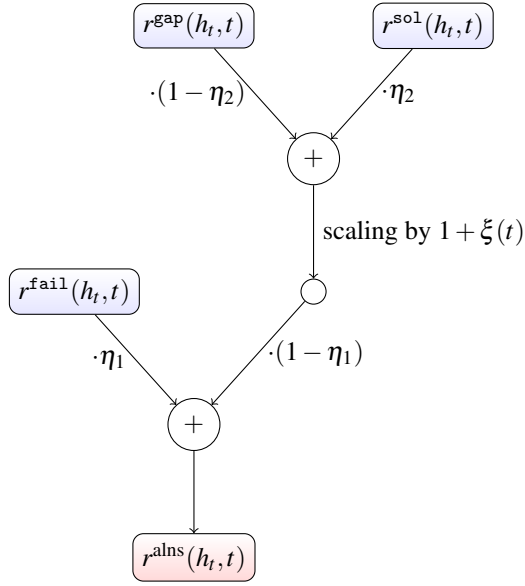


Figure 1: Diagram of the proposed reward function.

budget. Such a sequential decision process from a finite set of actions (heuristics) with unknown outcome appears in the literature as *Multi Armed Bandit Problem* [9].

The basic multi armed bandit problem can be described as a game, which is played over multiple rounds. In every round $t = 1, 2, \dots$, the player chooses one action $h_t \in \mathcal{H}$ from a finite set of available actions. In return for playing h_t , the player observes a *reward* $r(h_t, t) \in [0, 1]$ for the selected action. The aim for the player is to maximize their total revenue $\sum_t r(h_t, t)$. Since only the reward of the selected action can be observed at a time, every suitable algorithmic strategy must find a good balance between exploration across all actions and exploitation of the best action seen so far.

Depending on the nature of the reward distribution, we distinguish between two main scenarios of multi armed bandit problems. In the *stochastic scenario*, the observable rewards $r(h, t)$ for every action $h \in \mathcal{H}$ are independent, identically distributed (i.i.d.) random draws over time from a probability distribution with unknown *expected reward* $\mu_h \in [0, 1]$. In the stochastic scenario, a good strategy should play an action h^* with maximum expected reward $\mu_{h^*} \geq \mu_{h'} \forall h' \in \mathcal{H}$ as often as possible.

In the *adversarial scenario*, the player faces an opponent that chooses the rewards with the goal to maximize the player's *regret*—the discrepancy between the player's revenue and the best possible revenue. The opponent may take into account all choices previously made by the player, but does not know the selected action at time t . After the player and the opponent have each made their decisions, the player receives the reward $r(h_t, t)$ for the selected action only, while the opponent is informed about the player's choice h_t . It is noteworthy that in the adversarial scenario, the opponent has an incentive to play rewards different from 0 in every round of the game because the player's regret is minimal in every round t where all actions have a reward of 0. In the adversarial scenario, a good strategy must be necessarily randomized in some way because every deterministic algorithm is easily fooled by the opponent, who can minimize the player's total revenue by assigning a reward of 0 to the player's deterministic next action, and 1 to all other actions.

Intuitively, the adversarial scenario seems much harder to approach than the stochastic scenario because the latter is indifferent to choices made by the player, and estimates of the expected rewards can be built over time. It turns out that it is possible, even for the adversarial scenario, to create strategies that yield an asymptotically optimal revenue in their respective scenario. The reader is referred to the survey [9] for more information about and variants of the following algorithms.

4.1 Strategies for the Stochastic Scenario

The assumption of the stochastic scenario is that the observable reward of each action $h \in \mathcal{H}$ is the realization of a random variable R_h around its expected value

$\mu_h = \mathbb{E}(R_h)$. Let $T_h(t) := \sum_{i=1}^t \mathbb{1}_{h_i=h}$ denote the number of times that action h has been selected until round t . The *sample average*

$$\bar{r}_h(t) := \frac{1}{T_h(t)} \sum_{i=1}^t \mathbb{1}_{h_i=h} r(h, i)$$

is an unbiased estimator of the unknown quantity μ_h .

Algorithm 1: ε -greedy

Input: Set of actions \mathcal{H} , parameter $\varepsilon \geq 0$

```

1  $t \leftarrow 0$ 
2 while not stopped do
3    $t \leftarrow t + 1$ 
4    $\varepsilon_t \leftarrow \varepsilon \cdot \sqrt{\frac{|\mathcal{H}|}{t}}$ 
5   Draw  $v_t \sim \mathbb{U}([0, 1])$ 
6   if  $v_t \leq \varepsilon_t$  /* Selection of next action */
7     then
8     | Draw  $h_t \sim \mathbb{U}(\mathcal{H})$ 
9   else
10  |  $h_t \leftarrow \operatorname{argmax}_{h \in \mathcal{H}} \bar{r}_h(t)$ 
11  Update  $\bar{r}_{h_t}(t+1)$  by the observed reward  $r(h_t, t)$ 
```

Algorithm 1 is a very simple, randomized selection strategy for the multi armed bandit problem. It uses the short notation $\mathbb{U}(X)$ to denote the *uniform distribution* over a set X . The initial lack of reward information is compensated by a randomized selection of the first few actions. The amount of random selections decreases at the speed of $\frac{1}{\sqrt{t}}$ and can be controlled by the input parameter ε . With increasing t , it therefore becomes less and less likely to choose an action at random, whereas the probability of greedily exploiting the best action increases.

A different, deterministic approach to the stochastic scenario is based on the principle of optimism at the face of uncertainty. The approach is called *Upper Confidence Bound (UCB)* algorithm. Under the assumption that all rewards are in the interval $[0, 1]$, it holds with a probability of at least $1 - \delta$ that

$$\bar{r}_h(t) + \sqrt{\frac{2}{T_h(t)} \ln \frac{1}{\delta}} > \mu_h. \quad (5)$$

The quantity on the left side of Equation 5 is an upper confidence bound for μ_h at confidence level δ . With the goal to ultimately find the action h^* with maximum expected reward μ_{h^*} , the UCB algorithm selects the action that maximizes the upper confidence bound instead, while increasing the confidence level as a function of time, e.g., $1/\delta = 1 + t$. The rationale behind this is that also inferior actions become more attractive to the algorithm after they have not been selected for a while. The width of the confidence interval is further controlled by a parameter $\alpha \geq 0$. Let H be an ordered tuple of the elements in \mathcal{H} . The selection strategy α -UCB selects

$$h_t \in \begin{cases} \operatorname{argmax}_{h \in \mathcal{H}} \left\{ \bar{r}_h(t-1) + \sqrt{\frac{\alpha \ln(1+t)}{T_h(t-1)}} \right\} & \text{if } t > |\mathcal{H}|, \\ \{H_t\} & \text{if } t \leq |\mathcal{H}|. \end{cases} \quad (6)$$

The case distinction in Equation 6 is necessary to obtain a meaningful initialization of all sample means and because $T_h(|\mathcal{H}| + 1) \geq 1$ for all $h \in \mathcal{H}$ is required for Equations 5 and 6 to be well defined. In the first case of Equation 6, eventual occurring ties are broken uniformly at random. The special case of $\alpha = 0$ yields a completely greedy exploration strategy that does not take into account the upper confidence bound.

4.2 A Strategy for the Adversarial Scenario

Algorithm 2 is the standard algorithmic strategy for adversarial multi armed bandit scenarios. It is briefly called Exp.3, which is an abbreviation of "Exponential Weight Algorithm for Exploration and Exploitation". In each round t , the next action is selected randomly from a probability distribution defined by marginal probabilities $p_{h,t}$ for each $h \in \mathcal{H}$. After receiving the reward $r(h_t, t)$, the weight update is performed in two steps. First, the cumulative reward of the selected action h_t is updated in line 7. The cumulative reward weighs the observed reward with the probability to select h_t , thereby emphasizing actions with a high reward compared to their current selection probability. Second, the probabilities for the next iteration $t + 1$ are computed as a convex combination of two probability distributions based on the choice of γ . In the two extreme cases $\gamma \in \{0, 1\}$, the algorithm either draws from a uniform distribution ($\gamma = 1$) in the next iteration, or from a distribution defined by the normalized exponential function of the cumulative rewards ($\gamma = 0$).

Algorithm 2: Exp.3

Input: Set of actions \mathcal{H} , convex combination parameter $\gamma \in [0, 1]$

- 1 $p_{h,1} \leftarrow \frac{1}{|\mathcal{H}|}, Q_h \leftarrow 0 \forall h \in \mathcal{H}$
- 2 $t \leftarrow 0$
- 3 **while** *not stopped* **do**
- 4 $t \leftarrow t + 1$
- 5 Draw h_t from probability distribution $p_{h,t}$
- 6 Observe reward $r(h_t, t)$
- 7 $Q_{h_t} \leftarrow Q_{h_t} + \frac{r(h_t, t)}{p_{h_t}}$
- 8 **foreach** $h \in \mathcal{H}$ **do**
- 9 $p_{h,t+1} \leftarrow (1 - \gamma) \frac{\exp(Q_h)}{\sum_{h'} \exp(Q_{h'})} + \frac{\gamma}{|\mathcal{H}|}$

Table 2: Overview of involved parameters and values for the simulation and the MIP experiments. All SCIP parameters are preceded by `heuristics/alns/`. The placeholder `*` must be substituted by the name of a neighborhood, e.g., `rens`.

Symbol	SCIP parameter(s)	Section Ref.	Simulation	MIP
δ	<code>minimprov{low,high}</code>	2	0.01	0.01
k	<code>crossover/nsols</code>	2.1	2	2
d_{\max}	not parameterized	2.2	$0.2 \cdot n_b$	$0.2n_b$
$ \mathcal{S} $	<code>dins/npoolsols</code>	2.2	5	5
v_1^{lim}	<code>alns/minnodes</code>	3.2	50	50
κ_0	<code>alns/nodesofs</code>	3.2	500	500
κ_1	<code>alns/nodesquot</code>	3.2	0.1	0.1
η_1	<code>rewardbaseline</code>	3.3	0.5	0.5
η_2	<code>rewardcontrol</code>	3.3	0.8	0.8
$\phi_{h,t}$	<code>*/{min,max}fixingrate</code>	3.1	$\{0.1, 0.3, \dots, 0.9\}$	dynamic in $[0.3, 0.9]$
ε	<code>epsilon</code>	4.1	–	0.4685844
α	<code>alpha</code>	4.1	–	0.0016
γ	<code>gamma</code>	4.2	–	0.07041455
–	<code>banditalgo</code>	4	–	$\{\text{Exp.3,UCB},\varepsilon\text{-greedy}\}$

5 Computational Results

The proposed ALNS framework has been implemented and tested as an additional plugin on top of SCIP 5.0, using CPLEX 12.7.1 as the underlying LP solver. All 8 neighborhoods listed in Table 1 have been incorporated into ALNS. As instance set, we use the union of three MIPLIB collections 3.0, 2003, and 2010, [7, 3, 27] and the Coral [11] instance set, totaling to 666 instances. The computational experiments for the present work are split into two parts. The first part is an offline simulation that uses reward information about all neighborhoods in each call to ALNS. This information is used to compare neighborhoods directly, and to calibrate the parameters of the bandit selection strategies from Section 4. Section 5.3 describes the results that we obtained with the ALNS framework inside of SCIP using the readily calibrated selection strategies. Since a lot of parameters have been introduced in the previous sections, Table 2 summarizes the parameter settings used for the simulation and the performance experiments in this section.

5.1 Neighborhood Comparison

The first part aims at providing a fair comparison between the neighborhoods that have been implemented in the ALNS framework. Instead of choosing a single neighborhood at each call, all neighborhoods are executed one after another, and their individual rewards are recorded. In order to ensure fairness, every found improving solution is only recorded for the reward function. SCIP does not receive a solution as this would impact consecutive neighborhoods. All dynamic decisions are deactivated for this experiment. The target fixing rate is kept fixed at $0.1 - 0.9$ in steps of 0.2 , with a tolerance of ± 0.1 . Recall that the additional generic fixings/unfixings are only applied if the obtained fixing rate lies outside of the tolerance interval. The experiments have been conducted on a Linux cluster using Ubuntu 16.04, with a time limit of 5h for each instance.

Not all neighborhoods can be used for all problems. Local Branching and Proximity can only be executed for instances involving binary variables. Zero Objective requires a nonzero objective function. Furthermore, DINS requires a solution pool

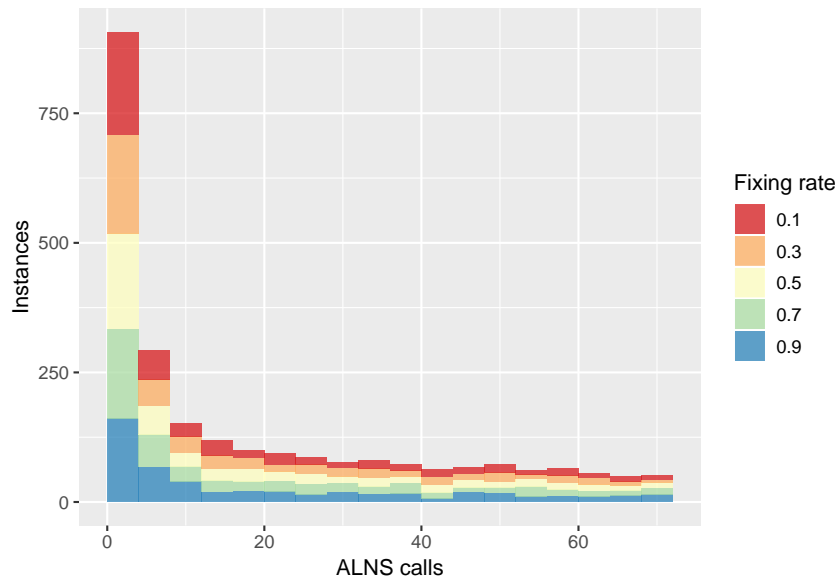


Figure 2: Histogram of call frequencies.

Table 3: Number of successful calls of ALNS

Fixing rate	Calls	Success	Succ. Rate
0.1	9037	841	0.093
0.3	9233	975	0.106
0.5	9789	1005	0.103
0.7	9925	1196	0.121
0.9	10085	1337	0.133

with at least 5 solutions, while RENS and RINS require a feasible LP relaxation at the local node. For this experiment, the ALNS framework is only executed if all those conditions are met.

Among the total set of instances, 494 instances allowed to execute all 8 neighborhoods at least once. A histogram of the number of calls is shown in Figure 2 with a bin width of 4 calls. One observes that for more than half of the instances, the heuristic is called at most eight times. This means that a UCT selection algorithm would have tried every neighborhood at most once in the real setup. On the other hand, the initialization phase has been finished on more than 200 instances. On average, ALNS was executed between 18.3 and 20.4 times per instance, depending on the fixing rate.

As one expects, the running time spent in a sub-MIP decreases with the fixing rate. Therefore, the number of times that ALNS could be executed is different for every fixing rate. Table 3 shows the number of finished calls to the framework for every tested fixing rate. It ranges from 9037 at a fixing rate of 0.1 to 10085 at 0.9. Particularly interesting are the calls where at least one of the tested neighborhoods finds a solution. The number of successful calls of an oracle neighborhood is shown in total in column “Success” and as a rate in column “Succ. Rate”. The success rate of

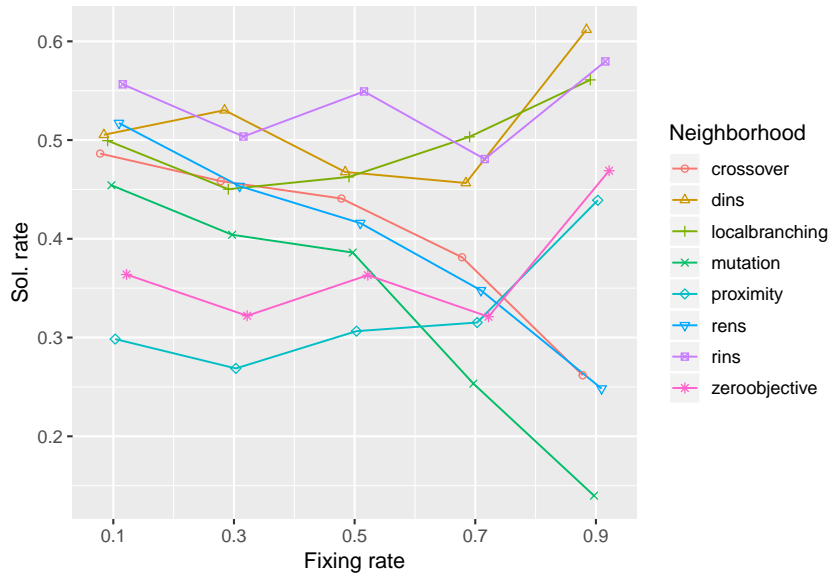


Figure 3: Solution rate by neighborhood and fixing rate.

the oracle neighborhood ranges from 9.3% to 13.3% and increases with an increasing fixing rate.

In all other calls, the selection process is only required to select a neighborhood that fails fast, but cannot contribute to the overall search process with an incumbent solution. Therefore, all results reported in the remainder of this section are reported for the subset of calls where at least one neighborhood finds a solution.

For this set of calls, Figure 3 shows the average number of solutions for every heuristic at every fixing rate. It can be observed that RINS, DINS, and Local Branching are almost consistently the top three neighborhoods across all tested fixing rates. While the solution frequency for those neighborhoods does not show monotonous trends, the solution rates for Crossover, RENS, and Mutation are decreasing with an increasing fixing rate.

The ranking between the neighborhoods is similar for the obtained rewards illustrated in Figure 4. Here, the average reward increases with the fixing rate. This is partly because the reward definition penalizes neighborhoods with a high fixing rate less strictly. A higher average reward results from an increased solution frequency, a better solution quality, and/or less effort to solve the subproblem. The neighborhood rewards at a particular fixing rate can be compared well.

As expected from the previous analysis, RINS, Local Branching, and DINS reach high average rewards. The highest increase in average reward can be observed for the Proximity and Zero Objective neighborhoods. At a high fixing rate of 0.9, RENS achieves the smallest average reward. The reward for Mutation only increases up to a fixing rate of 50%. Its reward is almost constant for all fixing rates $\geq 50\%$. RENS lacks a reference solution for additional, generic fixings, which is why it can run less frequently than others. A possible explanation for the decreasing scores of Crossover is the random selection of reference solutions. Searching a narrow neighborhood of a reference solution far away from the incumbent may lower its chances to find a better solution. The lower scores of Mutation are remarkable because RINS and Mutation

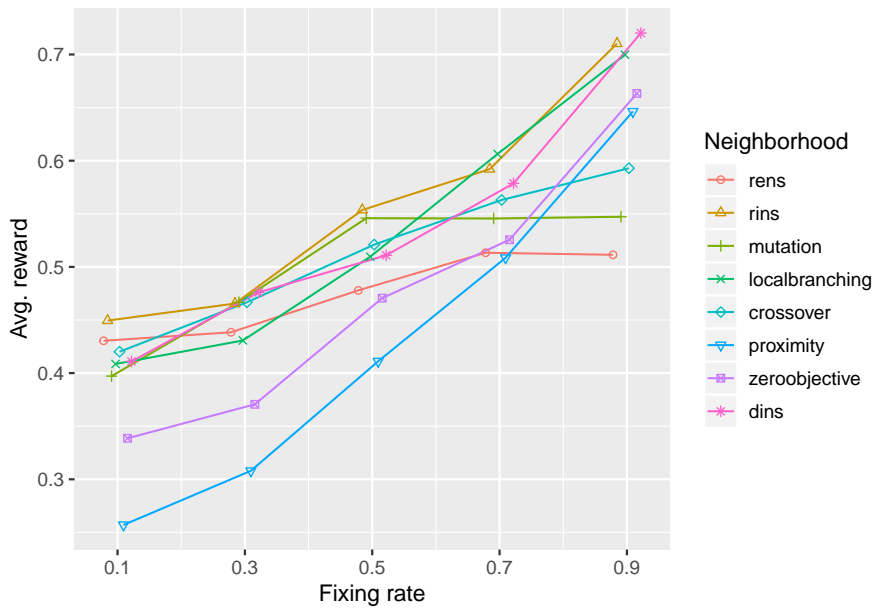


Figure 4: Avg. reward by neighborhood and fixing rate.

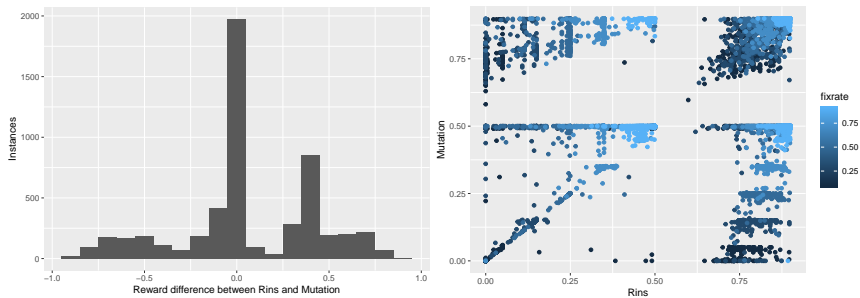


Figure 5: Reward comparison of RINS and Mutation.

use the same reference solution, namely the incumbent. RINS may even need additional generic fixings to reach higher target fixing rates, whereas the Mutation scheme always fixes the targeted percentage of integer variables. The large discrepancy in their solution rates indicates that more informed approaches such as the LP driven neighborhoods RINS or DINS are the most important fixing schemes.

One may ask the question whether a well-performing neighborhood such as RINS entirely dominates the less well-performing neighborhoods such as Mutation. Figure 5 illustrates the measured rewards for RINS and Mutation, respectively. RINS has a clear tendency to score higher, especially at larger fixing rates. However, also the execution of Mutation can be beneficial, as it reaches a higher reward in about 30% of the cases. Analogous comparisons for other pairs of neighborhoods yield similar results. Based on these observations, it is reasonable to enable all 8 neighborhoods by default, and to rely on the selection mechanism.

5.2 Simulation of the Selection Process

The rich data set from the previous section is now used for an offline calibration of the three bandit selection strategies. Each of the three bandit selection methods presented in Section 4 has a single parameter that can be calibrated for the use inside the ALNS selection process. Therefore, all three UCB, Exp.3, and ϵ -greedy are run on the entire collected data (all calls with and without at least one solution) with the aim to maximize the average reward obtained. Average rewards are computed over 100 repetitions of each algorithm. The simulation of the selection routines has been implemented in the programming language R.

Ideally, the selection performs better than a pure random selection for instances that allow for a certain number of calls to initialize the selection process. Note that certain parameter choices of the Exp.3 ($\gamma = 1$) and ϵ -greedy bandits are equivalent to a uniform random selection.

Figure 6 shows the selection quality for the three selection methods in terms of the average reward and solution rate. As in the previous section, the figures in this section refer to the subset of calls with at least one successful neighborhood. This means that the solution rate of an optimal selection algorithm would be a horizontal line with value 1.0 across all fixing rates. The first row depicts the selection quality of Exp.3 at different values of the γ parameter. Some hand-picked values $\{0.05, 0.5, 0.95\}$ are compared to $\gamma = 0.07041455$, the optimal value for γ computed by the R function `optimize`, and a pure random selection named “avg”. At all tested values of the γ -parameter, the selection quality of Exp.3 is better than purely random. Furthermore, the experiment reveals that higher values of γ decrease the selection quality across all tested fixing rates. The choice of $\gamma = 0.95$ shows, as expected, almost the same selection quality as a pure random selection.

The best choice for ϵ computed by R is 0.4685844. The results for $\epsilon = 4$ are indistinguishable from random sampling. The reason is the limited number of calls to ALNS, for which $\epsilon_t \geq 1$ for all $t \leq 71$, such that only random sampling is applied by the ϵ -greedy algorithm.

Finally, the average reward of the UCB selection algorithm has been maximized for the parameter choice of $\alpha = 0.0016$. Its selection quality is depicted in the last row of Figure 6. It can be noted that the average selection quality is higher for UCB and ϵ -greedy than for Exp.3. Especially the solution rate is maximized by the UCB algorithm. The optimal values for the different parameters can be interpreted as follows. The optimal value for the γ -parameter is very close to a purely weight based Exp.3 selection strategy. The optimal value of the α -parameter shows a higher selection quality than the nearby value of $\alpha = 0$, a purely greedy selection. This is seconded by the optimal value of the ϵ -parameter. This shows that learning from past observations clearly helps the selection process at later stages.

Another observation is that the plots of Figure 6 seldomly cross, i.e. the ranking between different parameter choices is the same for different fixing rates. This indicates that the selection algorithm can be safely combined with an adaptive fixing rate.

The learning success of the bandit selection methods is depicted in Figure 7. It shows the average solution rate of Exp.3 as a function of the number of calls to the ALNS framework for all four different choices of the parameter γ . As a comparison serves the average solution rate per call. The picture illustrates that the solution rate for the choice of $\gamma = 0.07041455$ is better than random after a small number of calls. Furthermore, while the average solution rate stays relatively constant around 0.3, the solution rate of Exp.3 has a clear tendency to increase with the number of calls. It

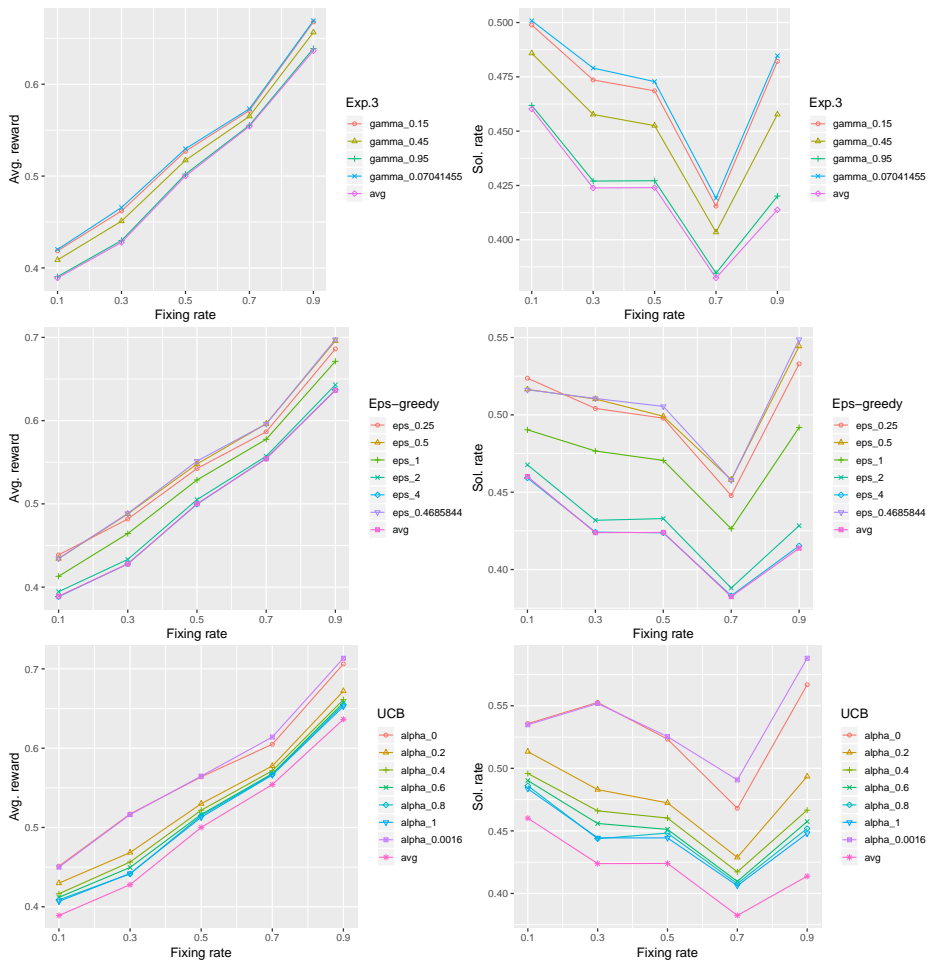


Figure 6: Comparison of selection performance for different parameter choices.

reaches a solution rate of 0.6 after 63 calls.

The two other bandit algorithms achieve this mark much earlier. The second diagram shows the solution rate by call for the ϵ -greedy algorithm for two choices of ϵ . Compared to Exp.3, the selection achieves an even better solution rate. Its margin from the average solution rate is higher. As an example, the (arbitrary) mark of a solution rate of 0.6 is first reached after 24 calls, and reliably surpassed after 40 calls to the selection routine.

The final plot shows that UCB outperforms the two other bandit selection routines. The mark of 0.6 is reached after 17 calls for the first time, and almost consistently surpassed after 30 calls. The price for this selection performance is that the first 8 observations must be spread over the 8 neighborhoods to select from, which is why UCB achieves exactly average performance at this early stage. At a later stage, UCB reaches a solution rate of 1.0 for the four rightmost observations, i.e., UCB can safely identify and select a well performing neighborhood at this stage. Recall that these plots represent average solution rates over 100 repetitions of the experiment.

As a conclusion, all three bandit selection algorithms achieve an above average selection performance, as desired. With an increasing initialization time, the learning effect becomes more pronounced. UCB achieves the best solution rate, followed by ϵ -greedy and Exp.3. Arguably, the good solution rate is an indication that the designed reward function, which the bandits actually receive, captures the ranking between the neighborhoods sufficiently well within the ALNS framework.

5.3 MIP Performance

This section presents the computational benefit of ALNS inside SCIP in a real setting where only one neighborhood can be called at a time. All 3 bandit selection methods have been tested and compared to SCIP with deactivated ALNS. The individual selection parameters are set to their individual optimal value as described in the previous section. The individual, existing Large Neighborhood Search heuristics RENS, RINS, and Crossover are active independently from ALNS in all four settings, as this represents the default settings of SCIP 5.0 with CPLEX 12.7.1 as LP solver. This experiment has been conducted on a Linux cluster of 32 computing nodes equipped with Intel Xeon CPU E5-2670 v2 at 2.50GHz. The time limit was 2h for every instance. In order to measure time as accurately as possible, every job has been scheduled exclusively.

Table 4 shows aggregated results for two performance measures, the solving time to optimality and the primal integral. It has been prepared using the Interactive Performance Evaluation Tools [23]. Individual outcomes for every instance and setting are found in Table 5 in the appendix. The measures are presented as shifted geometric mean time (using a shift of 1 sec.) and average primal integral. For a better quantitative assessment, relative percentage deviations from the fastest setting are shown. Table 4 summarizes the performance for the entire test bed as well as three interesting subgroups. The first subgroup “Diff” contains all instances on which activating ALNS has an effect on the solving process of SCIP for at least one selection strategy. On the complementary group “Equal”, ALNS does not alter the solving process. A change in the path is detected by a change in the number of LP iterations of the main solution process. Both groups drop all instances that could be solved by none of the four settings. The last subgroup is the subset of MIPLIB 2010 [27] benchmark instances.

The overall fastest setting in this experiment is ALNS equipped with an Exp.3 selection algorithm. On the entire test bed, an approximate 2 % time improvement can be observed. Also with a UCB or ϵ -greedy selection, a time improvement can be

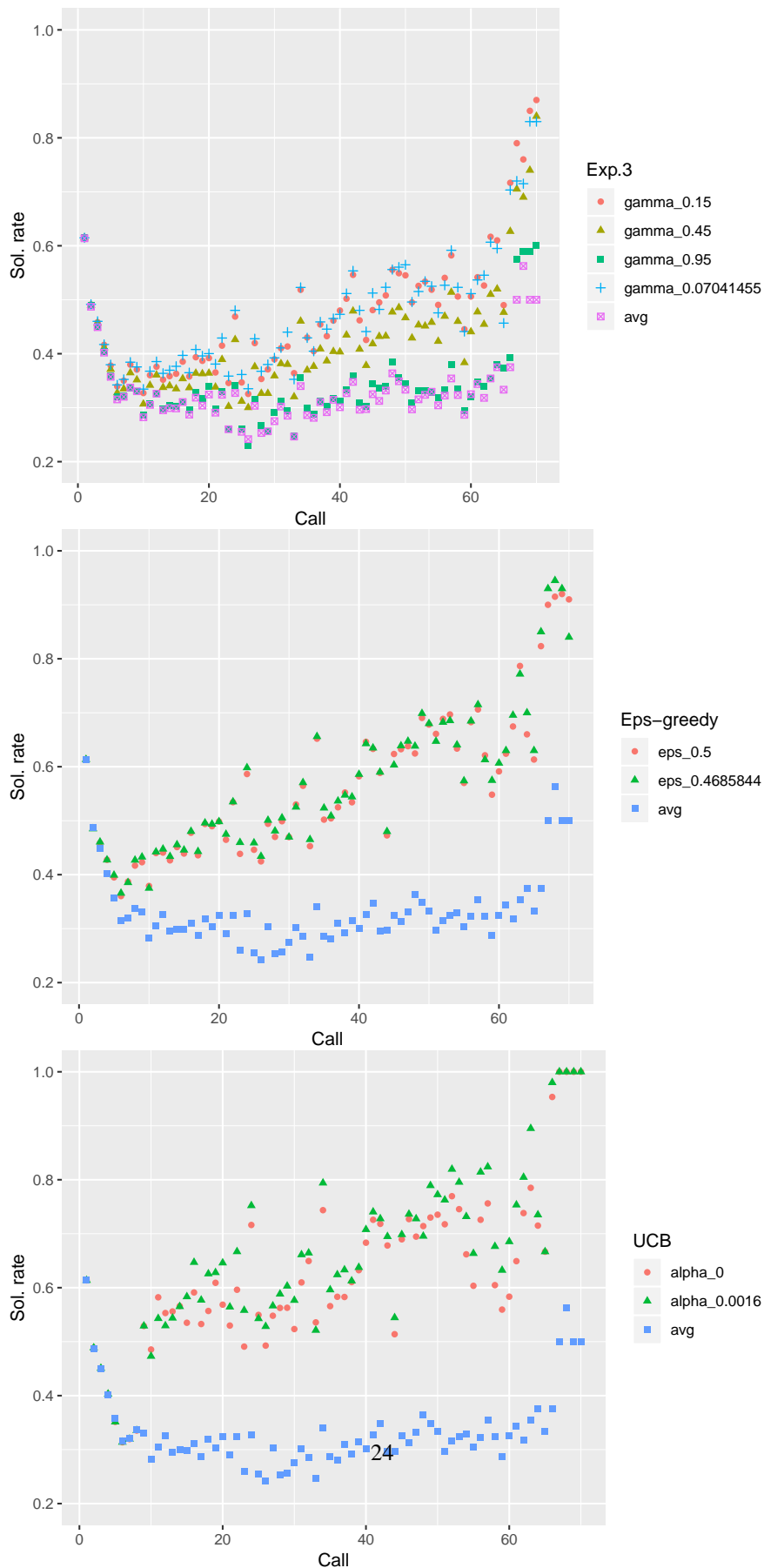


Figure 7: Average solution rate as a function of the individual call.

Table 4: Performance results of ALNS with different selection algorithms

Group	Setting	Inst.	Tlim	Time	Rel. Time %	Integral	Rel. Int. %
All	Exp.3	665	263	358.61	-1.89	77204.62	-2.11
All	Eps-greedy	665	266	360.90	-1.26	78001.49	-1.10
All	UCB	665	264	360.17	-1.46	76516.88	-2.98
All	ALNS off	665	265	365.52	0.00	78868.44	0.00
Diff	Exp.3	110	2	136.40	-11.39	10281.80	-34.92
Diff	Eps-greedy	110	5	137.21	-10.87	12278.62	-22.27
Diff	UCB	110	3	137.55	-10.65	10046.82	-36.40
Diff	ALNS off	110	4	153.94	0.00	15797.51	0.00
Equal	Exp.3	293	0	35.05	0.19	12888.25	0.23
Equal	Eps-greedy	293	0	35.49	1.46	12876.27	0.14
Equal	UCB	293	0	35.29	0.88	12870.50	0.10
Equal	ALNS off	293	0	34.98	0.00	12858.12	0.00
MIPLIB2010	Exp.3	87	6	314.01	-4.59	21610.16	-23.68
MIPLIB2010	Eps-greedy	87	7	321.02	-2.46	23195.39	-18.08
MIPLIB2010	UCB	87	6	323.58	-1.69	21097.42	-25.49
MIPLIB2010	ALNS off	87	7	329.12	0.00	28315.92	0.00

observed, albeit smaller. The time improvement is more pronounced for MIPLIB 2010, where ALNS with Exp.3 is more than 2 % faster than its bandit competitors and 4.6 % faster than SCIP without ALNS. Finally, for the set of affected instances, the ALNS time improvement (using Exp.3) is more than 11 %. A very similar time result has been obtained for the other two selection algorithms.

The primal integral is minimized by ALNS with the UCB selection strategy, closely followed by Exp.3. Overall, UCB achieves an improvement of 3 %, but the improvement is substantial for the subgroups MIPLIB2010 and Diff, where improvements of more than 25 % and 36 % are obtained, respectively. The ϵ -greedy selection also yields improved primal integrals for all sets of instances, especially on the set of affected instances. However, the improvements are less pronounced than for the other bandit algorithms. The subgroup “Equal” shows that the primal integral is almost not affected by ALNS if it does not affect the solution process. This indicates that the overhead caused by ALNS is negligible in case its solutions do not contribute to the overall search progress. Note that it is possible that the search is not affected even though ALNS finds a solution.

Overall, ALNS improves SCIP with all tested selection strategies with respect to both time and primal integral. The positive outcome for Exp.3 is unexpected because Exp.3 scored worse than UCB and ϵ -greedy in the simulation experiment. A possible explanation is the different scenario assumption of Exp.3. The important difference between the simulation and the MIP performance results is that solutions found by ALNS during the simulation experiment have been discarded and only reward information was kept. It is not unlikely that during the simulation, repeated calls to the same neighborhood resulted in the same solution. Designed for the stochastic scenario, UCB and ϵ -greedy aim at identifying a single, best performing neighborhood, which may give them an advantage over Exp.3 only in the simulation. The weighted random-

ized selection of Exp.3 may reduce the initial set of neighborhoods over time to two or three equally strong neighborhoods for an instance, or even continue using all available neighborhoods if they achieve similar rewards.

6 Conclusion

This article introduces Adaptive Large Neighborhood Search for MIP, a framework around eight well-known LNS neighborhoods from the literature. It has been implemented as a primal heuristic in SCIP and is publicly available since SCIP 5.0. The framework combines a selection procedure, which is governed by algorithms for the multi armed bandit problem, and the idea of generic additional variable fixings to adjust the complexity of the subproblems, as needed. We propose a reward function that combines the important aspects of solution quality and subproblem effort into a single number. We have used a simulation experiment to calibrate each bandit algorithm individually. Training the bandit algorithms with this reward function shows a clear trend to improve the solution rate with an increasing number of calls. As a byproduct of the simulation, we could see clear differences between the neighborhoods regarding the number of solutions they produce. Two of the neighborhoods that are most successful in our experiments, DINS and Local Branching, have been previously inactive in SCIP.

We currently see two future perspectives for this work. Adaptive algorithm selection may also be beneficial in other parts of the solver where the choice between similar methods largely affects the overall performance of the solver. In a recent article [25], promising results are presented for diving heuristics, and for dynamic switching between different pricing strategies of the Dual Simplex procedure to maximize the node throughput during the search. On the other hand, further algorithmic improvements regarding the use of auxiliary problems in general can be more easily incorporated into a single framework, such as the transfer of variable histories or conflict clauses, which may be very useful for the remaining search process.

Acknowledgements

The work for this article has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM). The author would like to thank his colleagues Timo Berthold and Thomas Schlechte for their reviews and comments on earlier versions of this manuscript.

References

- [1] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
- [2] T. Achterberg, T. Berthold, and G. Hendel. Rounding and propagation heuristics for mixed integer programming. In D. Klatté, H.-J. Lüthi, and K. Schmedders, editors, *Operations Research Proceedings 2011*, pages 71–76. Springer Berlin Heidelberg, 2012.
- [3] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006.

- [4] M. Bénichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent. Experiments in mixed-integer programming. *Mathematical Programming*, 1:76–94, 1971.
- [5] T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.
- [6] T. Berthold. RENS—the optimal rounding. *Mathematical Programming Computation*, 6(1):33–54, 2014.
- [7] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.
- [8] R. Borndörfer, H. Hoppmann, and M. Karbstein. A configuration model for the line planning problem. In D. Frigioni and S. Stiller, editors, *ATMOS 2013 - 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, volume 33, pages 68 – 79, 2013.
- [9] S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *CoRR*, abs/1204.5721, 2012.
- [10] COIN-OR branch-and-cut MIP solver, 2016. <https://projects.coin-or.org/Cbc>.
- [11] Coral mip benchmark library, 2016. <http://coral.ise.lehigh.edu/data-sets/mixed-integer-instances>.
- [12] IBM ILOG CPLEX Optimizer, 2016. <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/>.
- [13] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.
- [14] E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005.
- [15] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.
- [16] M. Fischetti and A. Lodi. Repairing milp infeasibility through local branching. 35:1436–1445, 05 2008.
- [17] M. Fischetti and M. Monaci. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731, Dec 2014.
- [18] M. Fischetti and D. Salvagnin. Feasibility pump 2.0. *Mathematical Programming Computation*, 1(2):201–222, Oct 2009.
- [19] G. Gamrath, T. Berthold, S. Heinz, and M. Winkler. Structure-based primal heuristics for mixed integer programming. In *Optimization in the Real World*, volume 13, pages 37 – 53. 2015.
- [20] G. Gamrath, T. Koch, A. Martin, M. Miltenberger, and D. Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7(4):367–398, 2015.

- [21] S. Ghosh. DINS, a MIP Improvement Heuristic. In M. Fischetti and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization: 12th International IPCO Conference, Ithaca, NY, USA, June 25-27, 2007. Proceedings*, pages 310–323, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [22] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. The scip optimization suite 6.0. Technical Report 18-26, ZIB, Takustr. 7, 14195 Berlin, 2018.
- [23] G. Hendel. IPET interactive performance evaluation tools. <https://github.com/GregorCH/ipet>.
- [24] G. Hendel. New rounding and propagation heuristics for mixed integer programming. Bachelor thesis, 2011.
- [25] G. Hendel, M. Miltenberger, and J. Witzig. Adaptive algorithmic behavior for solving mixed integer programs using bandit algorithms. Technical Report 18-36, ZIB, Takustr. 7, 14195 Berlin, 2018.
- [26] E. B. Khalil, B. Dilkina, G. Nemhauser, S. Ahmed, and Y. Shao. Learning to run heuristics in tree search. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.
- [27] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelman, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.
- [28] P. Laborie and D. Godard. Self-adapting large neighborhood search: Application to single-mode scheduling problems. In P. Baptiste, G. Kendall, A. Munier, and F. Sourd, editors, *MISTA-07*, 08 2007.
- [29] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.
- [30] S. J. Maher, T. Fischer, T. Gally, G. Gamrath, A. Gleixner, R. L. Gottwald, G. Hendel, T. Koch, M. E. Lübbecke, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, D. Weninger, J. T. Witt, and J. Witzig. The scip optimization suite 4.0. Technical Report 17-12, ZIB, Takustr. 7, 14195 Berlin, 2017.
- [31] L.-M. Munguía, S. Ahmed, D. A. Bader, G. L. Nemhauser, and Y. Shao. Alternating criteria search: a parallel large neighborhood search algorithm for mixed integer programs. *Computational Optimization and Applications*, 69(1):1–24, Jan 2018.
- [32] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435, 2007.
- [33] E. Rothberg. An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

[34] A. Sabharwal, H. Samulowitz, and C. Reddy. Guiding combinatorial optimization with UCT. In N. Beldiceanu, N. Jussien, and E. Pinson, editors, *CPAIOR*, volume 7298 of *Lecture Notes in Computer Science*, pages 356–361. Springer, 2012.

[35] Xpress. FICO Xpress-Optimizer, 2016. <http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx>.

Appendix

Table 5: The MIP performance results from Section 5.3 for every instance. The 3 columns 2010, Diff, and Equal indicate the group membership of an instance to the respective group in Table 4.

ProblemName	2010	Diff	Equal	Time (sec.)				Prim. Int.			
				Exp.3	UCB	ϵ -greedy	ALNS off	Exp.3	UCB	ϵ -greedy	ALNS off
10teams			✓	13	13	13	12	558	588	599	606
22433			✓	3	3	3	3	290	270	270	270
23588			✓	6	6	6	6	212	202	222	202
30n20b8	✓		✓	256	254	255	254	15902	15736	15769	15814
50v-10				7200	7200	7200	7200	2921	2916	2912	2970
Test3			✓	5	5	5	5	195	195	195	195
a1c1s1				7200	7200	7200	7200	3913	2832	3189	2830
acc-tight4			✓	170	170	170	171	17000	17000	17000	17100
acc-tight5	✓		✓	118	116	117	116	11800	11600	11700	11600
acc-tight6			✓	71	71	71	70	7060	7070	7100	7040
aflow30a			✓	27	27	27	27	718	736	753	718
aflow40b	✓		✓	900	900	898	894	3868	3879	3861	3725
air03			✓	2	3	3	3	63	96	83	94
air04	✓		✓	27	26	27	27	262	222	252	252
air05			✓	18	18	19	17	172	206	193	166
alignmq			✓	16	16	16	16	380	370	380	370
app1-2	✓	✓		5326	4415	7200	7200	177568	135961	309051	720001
arki001		✓		7200	6777	7200	7200	392	416	434	431
ash608gpa-3col	✓		✓	61	61	61	61	6086	6074	6064	6097
atlanta-ip				7200	7200	7200	7200	46308	51595	42982	65872
atm20-100				7200	7200	7200	7200	720000	720000	720000	720000
b2c1s1				7200	7200	7200	7200	17792	9487	10847	24912
bab1				7200	7200	7200	7200	1650	1590	4326	3238
bab3				7200	7200	7200	7200	720271	720122	720545	720325
bab5	✓	✓		677	871	1116	2819	2746	2707	2737	2896
bc			✓	1146	1151	1145	1143	15356	15685	15301	15302
bc1			✓	254	254	251	247	6265	6306	6195	6165
beasleyC3	✓	✓		99	76	108	72	532	504	515	556
bell3a			✓	8	8	8	8	0	0	0	0
bell5			✓	1	1	1	1	10	10	10	10
berlin_5_8_0				7200	7200	7200	7200	1223	1232	1408	1232
bg512142				7200	7200	7200	7200	36444	38919	33846	37986
biella1	✓	✓		722	722	234	709	2495	2521	1995	2418
bienst1			✓	35	36	36	35	151	170	179	149
bienst2	✓	✓	✓	342	342	343	343	471	467	500	474
binkar10.1	✓	✓		6	48	50	49	49	33	52	37
blend2	✓	✓		2	2	2	5	19	30	29	217
bley-x11	✓		✓	177	176	178	177	16733	16728	16833	16795
blp-ar98				7200	7200	7200	7200	720000	720000	720002	720001
blp-ic97				7200	7200	7200	7200	3661	4616	4613	4602
bnatt350	✓		✓	1093	1094	1095	1106	109300	109400	109500	110600
bnatt400			✓	1302	1296	1300	1307	101100	100700	101000	101400
buildingenergy				7200	7200	7200	7200	17865	17868	17894	17887
cap6000		✓		5	5	5	5	29	66	59	29
circ10-3				7200	7200	7200	7200	720002	720001	720001	720001
co-100				7200	7200	7200	7200	55008	54391	54275	54226
core2536-691	✓	✓		59	49	58	108	273	249	273	357
core4872-1529				7200	7200	7200	7200	18330	19309	18893	19981
cov1075			✓	135	135	135	135	166	166	167	166
csched007			✓	2802	2748	2749	2740	5341	5243	5239	5204
csched008			✓	446	446	446	446	1454	1455	1464	1424
csched010	✓		✓	2567	2569	2569	2567	4819	4829	4832	4817
d10200				7200	7200	7200	7200	750	751	751	751
d20200				7200	7200	7200	7200	1127	1124	1121	1121
dano3.3		✓		69	54	69	68	535	577	535	535
dano3.4			✓	82	62	82	37	457	437	477	477
dano3.5		✓		158	166	183	159	376	399	376	396
dano3mp				7200	7200	7200	7200	27771	27675	27772	27754
danoint	✓	✓		5846	5841	5847	5957	584	585	759	598
datt256				7200	7200	7200	7200	719973	719952	719990	720043
dc1c				7200	7200	7200	7200	11541	10906	27174	11459
dc11				7200	7200	7200	7200	43512	43428	45520	43059
dcmulti		✓		3	2	3	2	15	14	15	14
dfn-gwin-UUM	✓	✓		76	68	68	88	156	166	166	169
dg012142				7200	7200	7200	7200	175566	111040	161012	187792
disctom			✓	2	2	2	2	200	200	200	160
dolom1				7200	7200	7200	7200	549206	571878	549027	548652
ds				7200	7200	7200	7200	541837	542147	541870	541299
ds-big				7200	7200	7200	7200	510983	503225	454460	498603
dsbmp			✓	1	1	1	1	43	43	43	83

continued on next page ...

ProblemName	2010	Diff	Equal	Time (sec.)				Prim. Int.			
				Exp.3	UCB	ϵ -greedy	ALNS off	Exp.3	UCB	ϵ -greedy	ALNS off
sp98ar				7200	7200	7200	7200	6469	4884	5810	4753
sp98ic	✓		✓	1434	1440	1441	1431	3189	3186	3188	3176
sp98ir	✓	✓		61	51	82	66	227	241	299	291
stein27			✓	1	2	1	1	0	0	0	0
stein45			✓	28	28	28	28	9	11	12	11
stockholm				7200	7200	7200	7200	73504	73507	73495	73496
stp3d				7200	7200	7200	7200	720003	720004	720002	720003
sts405				7200	7200	7200	7200	13499	13491	13487	13484
sts729				7200	7200	7200	7200	6714	6706	6727	6712
swath				7200	7200	7200	7200	4485	4502	4527	4457
t1717				7200	7200	7200	7200	182238	182192	182164	182199
t1722				7200	7200	7200	7200	97985	96676	91628	97322
tanglegram1	✓		✓	385	390	388	386	895	900	905	894
tanglegram2	✓		✓	9	9	9	9	177	169	169	195
timtab1	✓	✓		89	100	95	101	578	859	609	860
timtab2				7200	7200	7200	7200	14860	14888	14883	14852
toll-like				7200	7200	7200	7200	1835	1844	1839	1841
tr12-30			✓	1442	1444	1440	1442	61	60	61	59
triptim1	✓	✓		447	268	447	450	22037	21026	22043	22053
triptim2				7200	7200	7200	7200	173782	174882	177781	174282
triptim3				7200	7200	7200	7200	81753	81753	81753	81853
tw-myciel4				7200	7200	7200	7200	1468	1507	1460	1457
uc-case11				7200	7200	7200	7200	8966	8957	8946	8939
uc-case3				7200	7200	7200	7200	8397	8038	8018	7946
uct-subprob			✓	1065	1064	1066	1062	1479	1494	1481	1478
umts		✓		3966	2698	6103	3945	406	402	619	389
unitcal_7	✓		✓	275	274	274	274	4840	4820	4830	4850
usAbbrv-8-25-70				7200	7200	7200	7200	10781	10676	10683	10608
van				7200	7200	7200	7200	89471	93927	78965	70697
vpm1			✓	1	1	1	1	0	0	0	0
vpm2			✓	2	2	2	2	43	53	66	43
vpphard	✓			7200	7200	7200	7200	689845	690004	690144	689640
vpphard2				7200	7200	7200	7200	634470	634712	642961	634393
wachplan			✓	933	936	938	937	1680	1760	1830	1640
wnq-n100-mw99-14				7200	7200	7200	7200	531843	531683	531586	531843
zib01				30	31	30	30	3020	3097	3028	3025
zib02				7200	7200	7200	7200	0	0	0	0
zib54-UUE	✓	✓		2853	2291	2644	2310	1495	1545	1460	1502