

Bringing order into the neighborhoods: relaxation guided variable neighborhood search

Jakob Puchinger, Günther R. Raidl

▶ **To cite this version:**

Jakob Puchinger, Günther R. Raidl. Bringing order into the neighborhoods: relaxation guided variable neighborhood search. Journal of Heuristics, 2008, 14 (5), pp.457-472. 10.1007/s10732-007-9048-9 . hal-01224918

HAL Id: hal-01224918
https://inria.hal.science/hal-01224918

Submitted on 9 Nov 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Bringing Order into the Neighborhoods: Relaxation Guided Variable Neighborhood Search*

Jakob Puchinger and Günther R. Raidl
Institute of Computer Graphics and Algorithms (E186)
Vienna University of Technology,
Favoritenstraße, 9–11, 1040 Vienna, Austria
Phone: +43-1-58801-18616
Email:{puchinger|raidl}@ads.tuwien.ac.at

Abstract

In this article we investigate a new variant of Variable Neighborhood Search (VNS): Relaxation Guided Variable Neighborhood Search. It is based on the general VNS scheme and a new Variable Neighborhood Descent (VND) algorithm. The ordering of the neighborhood structures in this VND is determined dynamically by solving relaxations of them. The objective values of these relaxations are used as indicators for the potential gains of searching the corresponding neighborhoods. We tested this new approach on the well-studied multidimensional knapsack problem. Computational experiments show that our approach is beneficial to the search, improving the obtained results. The concept is, in principle, more generally applicable and seems to be promising for many other combinatorial optimization problems approached by VNS.

Keywords: Variable Neighborhood Search, Integer Linear Programming, Multidimensional Knapsack Problem

*This work is supported by the European RTN ADONET under grant 504438.

1 Introduction

The general Variable Neighborhood Search (VNS) heuristic including Variable Neighborhood Descent (VND) (Hansen and Mladenović, 1999, 2003) is a relatively young metaheuristic concept that has successfully been applied to several combinatorial optimization problems. It relies on the principle of searching different neighborhood structures in a systematic way. In Hansen and Mladenović (2003) the following significant question is raised: *What is the best order in applying the neighborhoods?*

Usually, neighborhoods are sorted according to increasing complexity of searching for the best moves, which often, but not always, corresponds to the size of neighborhoods. Sometimes, however, such an order might be hard to estimate or even be misleading. In particular, a prespecified, fixed order might in general not be ideal during all phases of the optimization process. We will introduce a new dynamic approach, in which the neighborhoods are always processed in an order that depends on estimated improvement-potentials.

We call this enhanced variant of general VNS *Relaxation Guided Variable Neighborhood Search*. It follows the standard VNS scheme but uses a new VND algorithm. As a specific technique for determining improvement-potentials for the neighborhoods considered in VND, we employ (Integer) Linear Programming (ILP/LP) techniques. Note that combining metaheuristics and ILP techniques is a highly promising research area for its own (Dumitrescu and Stuetzle, 2003; Puchinger and Raidl, 2005a), our approach can also be seen as a contribution to this direction.

The VND is guided by always sorting the neighborhoods according to estimations of the improvement-potentials depending on the current solution. For each neighborhood this potential is determined by quickly solving a relaxation. Searching the neighborhoods in this order is expected to increase solution quality and/or to speed up VNS.

In order to evaluate this new approach, we use the Multidimensional Knapsack Problem (MKP). It is a well-studied, strongly NP-hard combinatorial optimization problem occurring in many different application areas. In the last decades a multitude of exact and metaheuristic algorithms were developed for the MKP, some of them being very complex. We examine the benefits of relaxation guided VNS in comparison to standard version of VNS on this problem. Although it is not our primary goal to compete with the leading algorithms for large and complex MKP instances, we get relatively good results that are in general not far from the best known solutions.

In the next section we present the general scheme of Relaxation Guided VNS. Section 3 introduces the MKP in detail. We then describe the neighborhood structures used for the MKP, together with computational experiments comparing standard VNS and relaxation guided VNS in Section 4. Some extensions of this approach using more neighborhood structures and further experiments are presented in Section 5. We close with some concluding remarks and an outlook on future work.

2 Relaxation Guided VNS

Relaxation Guided VNS (RGVNS) follows the the general VNS scheme (Hansen and Mladenović, 1999, 2003) and incorporates an improved VND, which we call Relaxation Guided Variable Neighborhood Descent (RGVND). Let us assume that the neighborhood structures $N_1, \ldots, N_{k_{\max}}$ are to be used within VND. An important question, which often has a substantial impact on the algorithm's performance, is the order in which the neighborhoods are to be considered. Occasionally, this ordering is straight-forward, since the neighborhoods are nested, such that neighborhood N_i always fully contains $N - i - 1,\ \forall i = 2, \ldots, k_{\max}$. An example are classical k-opt neighborhoods for the traveling salesman problem or more general k-exchange neighborhoods. However, a particular strength of VND/VNS often lies in the combination of different *types* of neighborhoods which do not form such a monotone sequence of extensions.

In such cases, rules of thumb are typically used for defining a meaningful order. Examples are: (a) search smaller neighborhoods first; (b) search neighborhoods first for which faster algorithms exist; and (c) search neighborhoods first which are generally considered to be "more promising". In many situations, however, these strategies do not lead to the overall best ordering. Furthermore, the ordering that is best suited in a particular situation might in general depend on the current solution, i.e. there is not a single ordering which is optimal at any time of the heuristic search. With the exception of our preliminary results we published in Puchinger and Raidl (2005b), we are not aware of any previous work where the ordering of the neighborhoods is determined in an automatic way and, especially, is adapted during the search.

The central point of our extended variant of VND is that we automatically steer the order in which the neighborhood structures are processed in dependence of estimated improvement-potentials. These potentials are devised by quickly solving a relaxation of each neighborhood

structure. If the improvement-potentials are able to predict to a certain degree which neighborhoods are more promising and which are less, considering the neighborhoods in this order will yield better results and/or shorter running times.

In the following we will consider maximization problems; minimization problems can be treated analogously.

Assume that we are given a Combinatorial Optimization Problem (COP) defined as

$$z^{\mathrm{COP}} = \max\{f(x) \mid x \in S\}, \tag{1}$$

with S being a finite set of solutions and $f(x) : S \to \mathbb{R}$ an objective function. In analogy to the relaxation of an ILP (Wolsey, 1998), we introduce the following formal definition of a relaxation of a COP.

Definition 1 *A* relaxation *R of COP is any maximization problem defined as*

$$z^{\mathrm{R}} = \max\{f^{\mathrm{R}}(x) \mid x \in S^{\mathrm{R}}\} \tag{2}$$

with the following properties:

(i) $S \subseteq S^{\mathrm{R}}$

(ii) $f(x) \leq f^{\mathrm{R}}(x)$, $\forall x \in S$.

The following evident result (Wolsey, 1998) yields a bound for COP.

Proposition 1 *If R is a relaxation of COP, $z^{\mathrm{R}} \geq z$.*

If such a relaxation is chosen appropriately, it is substantially faster to calculate its optimal solution than the solution to the original COP. An example is the widely used LP relaxation of an ILP formulation for a COP, which can be solved in polynomial time. Other generic relaxation methodologies from the ILP domain, which are commonly applied with much success, are e.g. the surrogate relaxation, in which constraints are combined, and the Lagrangian relaxation, where constraints are transformed into penalty terms added to the objective function.

It is a prerequisite for our RGVND scheme that the used relaxations can be solved to optimality much faster than their corresponding original neighborhood structures on one side,

Algorithm 1: Relaxation Guided VND (RGVND)

Input: A feasible solution x

1 $l \leftarrow 1$

2 π = DetermineOrderOfNeighborhoods(x)

3 **repeat**

4 Find the best neighbor $x^* \in N_{\pi(k)}(x) \mid f(x^*) \geq f(x') \; \forall x' \in N_{\pi(k)}(x)$

5 **if** $f(x^*) > f(x)$ **then**

6 $x \leftarrow x^*$

7 $k \leftarrow 1$

8 π = DetermineOrderOfNeighborhoods(x)

9 **else**

10 $k \leftarrow k + 1$

11 **until** $k = k_{\max}$

12 **return** x

and that the obtained bounds are "good" in the sense that they give some indication for the objective values of the best solution in the neighborhood.

As a second precondition we assume that the used neighborhoods are not nested since this would lead to trivial orderings and render our approach meaningless, i.e. $N_k \not\subseteq N_{k'}$ and $N_{k'} \not\subseteq N_k$ holds for any $N_k, N_{k'}$ with $k \neq k'$.

Algorithm 2: DetermineOrderOfNeighborhoods(x)

1 **for** $k = 1, \ldots, k_{\max}$ **do**

2 Solve $N_k^{\mathrm{R}}(x)$ yielding solution value z_k^{R}

3 Sort $\pi = (1, \ldots, k_{\max})$ according to decreasing z_k^{R}

4 **return** π

In Algorithm 1 the pseudocode of RGVND is given. The significant differences to the standard VND scheme, as described in Hansen and Mladenović (1999, 2003), are the calls of function DetermineOrderOfNeighborhoods(x) in lines 2 and 8. This function determines the order of the neighborhood structures by first solving their relaxations yielding objective values z_k^{R}, and then sorting the neighborhoods according to decreasing z_k^{R}. Ties are broken arbitrarily or according to some static heuristic rules.

3 The Multidimensional Knapsack Problem

The Multidimensional Knapsack Problem (MKP) is a well-studied, strongly NP-hard combinatorial optimization problem occurring in many different applications. It can be defined by the following ILP:

$$\text{(MKP)} \qquad \text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{3}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i = 1, \ldots, m \tag{4}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{5}$$

There are n items with profits $p_j > 0$ and m resources with capacities $c_i > 0$ given. Each item j consumes an amount $w_{ij} \geq 0$ from each resource i. The goal according to (3) is to select a subset of items with maximum total profit; chosen items must, however, not exceed resource capacities, see (4). The 0–1 decision variables x_j indicate which items are selected.

The MKP first appeared in the context of capital budgeting (Lorie and Savage, 1955; Manne and Markowitz, 1957). A comprehensive overview of practical and theoretical results for the MKP can be found in the monograph on knapsack problems by Kellerer et al. (2004). A recent review of the MKP was given by Fréville (2004). Besides exact techniques for solving small to moderately sized instances based on dynamic programming (Gilmore and Gomory, 1966; Weingartner and Ness, 1967) and branch-and-bound (Shih, 1979; Gavish and Pirkul, 1985), many kinds of metaheuristics have already been applied to the MKP.

To our knowledge, the method currently yielding the best results, at least for commonly used benchmark instances, was described by Vasquez and Hao (2001) and has recently been refined by Vasquez and Vimont (2005). It is a hybrid approach based on tabu search. The search space is reduced and partitioned via additional constraints, thereby fixing the total number of items to be packed. Bounds for these constraints are calculated by solving a modified LP-relaxation. For each remaining part of the search space, tabu-search is independently applied, starting with a solution derived from the LP-relaxation of the partial problem. The improvement described in Vasquez and Vimont (2005) lies mainly in an additional variable fixing heuristic.

Various other metaheuristics have been described for the MKP (Glover and Kochenberger, 1996; Chu and Beasley, 1998), including several variants of hybrid evolutionary algorithms

(EAs); see Raidl and Gottlieb (2005) for a recent survey and comparison of EAs for the MKP. The authors of the current article have developed different approaches, combining exact methods with metaheuristics for solving the MKP (Puchinger et al., 2005, 2006; Puchinger, 2006).

4 Relaxation Guided VNS for the MKP

We now focus on the problem-specific details of our RGVNS implementation for the MKP, introducing the used neighborhoods and their relaxations, and present results for indicating the effectiveness of the new approach in comparison to standard VNS.

4.1 Representation and Initialization

Solutions are directly represented by binary strings, and all our neighborhoods are defined on the space of feasible solutions only. Previous studies such as Raidl and Gottlieb (2005) indicate that for the MKP, this seems to be in general a wise decision, since approaches also considering infeasible solutions to a larger degree turned out to be typically less effective. We denote by $I_1(x^f) = \{j \mid x^f_j = 1\}$ the index-set of the items contained in the knapsack of a current solution x^f and by $I_0(x^f) = \{j \mid x^f_j = 0\}$ its complement.

The initial solution for our VNS is generated using a greedy first-fit heuristic, considering the items in a certain order that is determined by sorting the items according to decreasing values x_j of the solution to the MKP's LP-relaxation; ties are broken randomly. See Raidl and Gottlieb (2005) for more information on this randomized greedy heuristic.

4.2 ILP-Based Neighborhoods

Instead of simple bit-flip or k-exchange neighborhoods that would inherently lead to infeasible solutions, we consider more sophisticated neighborhoods based on the MKP's ILP model.

We want to force a certain number of items of the current feasible solution x^f to be removed from or added to the knapsack. This is realized by adding neighborhood-defining constraints depending on x^f to the ILP formulation of the MKP.

In the first neighborhood, *ILP-Remove-and-Fill IRF*(x^f, κ), we force precisely κ items from I_1 to be removed from the knapsack and any combination of items from I_0 is allowed to

be added to the knapsack as long as the solution remains feasible. This is accomplished by adding the following equation to (3)–(5):

$$\sum_{j \in I_1(x^f)} x_j = \sum_{j \in I_1(x^f)} x_j^f - \kappa. \tag{6}$$

In the second neighborhood, *ILP-Add-and-Remove* $IAR(x^f, \kappa)$, we force precisely κ items not yet packed, i.e. from I_0, to be included in the knapsack. To achieve feasibility any combination of items from I_1 may be removed. This is achieved by adding the following equation to (3)–(5):

$$\sum_{j \in I_0(x^f)} x_j = \kappa. \tag{7}$$

As relaxations $IRF^R(x^f, \kappa)$ and $IAR^R(x^f, \kappa)$ for RGVND we use the corresponding LP-relaxations in which the integrality constraints (5) are replaced by $0 \leq x_j \leq 1$, $j = 1, \ldots, n$. Note that depending on the specific instance's characteristics, both neighborhoods may become quite large even for $\kappa = 1$. Nevertheless, the LP-relaxations can be solved to optimality very quickly by means of standard LP algorithms. For searching the (integer) neighborhoods we use a general purpose ILP-solver – CPLEX 9.0 in our case – with a certain time limit.

4.3 Relaxation Guided RGVND

Our Relaxation Guided Variable Neighborhood Decent (RGVND) for the MKP uses the previously defined neighborhoods $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$ with $\kappa = 1, \ldots, \kappa_{\max}$, where $\kappa_{\max}$ is a prespecified upper limit on the number of items that shall be removed or added. It otherwise follows the general algorithm already presented in Section 2, i.e. the order in which the neighborhoods are considered is always dynamically determined by solving the LP-relaxations and sorting the neighborhoods according to decreasing solution values. Ties are broken by considering neighborhoods with smaller κ earlier.

4.4 Shaking

When RGVND terminates with a solution that is locally optimal with respect to $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$ for $\kappa = 1, \ldots, \kappa_{\max}$, shaking is performed within the VNS framework. Our shaking flips $l \geq 1$ different, randomly selected variables x_j of the current solution and applies the following fast repair and local improvement procedures that have originally been proposed by Chu and Beasley (1998).

Both, repair and local improvement, are greedy first-fit strategies and guarantee that any resulting candidate solution lies at the boundary of the feasible region, where optimal solutions are always located. The repair procedure considers all items in a specific order Π and removes selected items ($x_j = 1 \ \rightarrow \ x_j = 0$) as long as any capacity constraint is violated. Local improvement works vice-versa: It considers all items in the reverse order $\overline{\Pi}$ and includes items not yet appearing in the solution as long as no capacity limit is exceeded.

Crucial for these strategies to work well is the choice of the ordering Π. Items that are likely to be selected in an optimal solution should appear near the end of Π. As described by Chu and Beasley (1998), Π is determined according to pseudo-utility ratios (efficiency values)

$$u_j = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}}, \tag{8}$$

where we set the relevance factors r_i to the dual variable values of the solution to the LP-relaxation of the MKP. Puchinger (2006) and Puchinger et al. (2006) evaluated in a slightly different context various efficiency measures and concluded that the dual variable values are in general a very good choice.

As usual in general VNS, l runs from 1 to some $l_{\max}$ and is reset to 1 if an improved solution is found.

4.5 Experimental Comparison

We compare RGVNS to standard VNS with a fixed neighborhood ordering and a VNS variant in which the neighborhoods are always considered in random order (RandVNS). All three strategies use the same ILP-based neighborhoods $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$, for $\kappa = 1, \ldots, \kappa_{\max} = 10$ for their VND. In standard VNS the neighborhood-ordering is $IRF(x^f, 1), IAR(x^f, 1), IRF(x^f, 2), IAR(x^f, 2), \ldots IRF(x^f, 10), IAR(x^f, 10)$, i.e. the neighborhoods are sorted according to increasing complexity and we always switch between IRF and IAR. For shaking the maximum number of variable flips $l_{\max}$ was set to n to not restrict the theoretical possibility of reaching any solution in the search space.

The algorithms were implemented in C++ using the ILP-solver CPLEX 9.0. The neighborhoods were not always fully explored to provable optimality since this turned out to be less effective. In our experiments, we allowed a maximum of two seconds per neighborhood evaluation. If CPLEX did not terminate within this time, the so far best neighboring solutions was used. The total run-time given to the algorithms was limited to 500 seconds. All experiments were performed on a 2.4GHz Intel Pentium 4 machine.

As in many previous publications for the MKP, we use Chu and Beasley's benchmark library for our experiments, which is available at Beasley's OR-Library[1]. In particular, we consider here the largest instances with $n = 500$ items, $m \in \{5, 10, 30\}$ constraints, and tightness-ratios $\alpha \in \{0.25, 0.5, 0.75\}$. Each instance has been generated randomly such that $c_i = \alpha \cdot \sum_{j=1}^{n} w_{ij}$ for all $i = 1, \ldots, m$. In order to evaluate the results of our experiments statistically, we performed 30 independent runs on the first instances for each of the nine parameter combinations.

For a solution with objective value z, we measure its quality by the $\%{-}gap = \left(z^{\mathrm{LP}} - z\right) / z^{\mathrm{LP}} \cdot 100\%$, where z^{LP} is the solution value of the MKP's LP-relaxation. Table 1 lists average and median percentage gaps together with standard deviations (in parentheses) for the final solutions of VNS, RandVNS, and RGVNS. We also performed Wilcoxon rank sum tests to see how significant the observed differences in the results of the three algorithms are. Columns $p_{\mathrm{VNS,RGVNS}}$ and $p_{\mathrm{RandVNS,RGVNS}}$ show obtained error probabilities for the hypotheses that the mean gap of VNS is larger than the one of RGVNS and the mean gap of RandVNS is larger than the one of RGVNS, respectively.

For seven out of the nine instances RGVNS yields significantly better results than the classical VNS approach with fixed neighborhood ordering. For one of the test cases ($m = 5,\ \alpha = 0.75$) all obtained results were equal, whereas for the ($m = 10,\ \alpha = 0.75$) case the standard approach obtained on average slightly better results than RGVNS, but without statistical significance. When comparing RGVNS to RandVNS, one can observe that in four cases RGVNS was significantly better, in one case results were identical, in two cases RandVNS was better, and in two cases no conclusion can be drawn. As expected the random ordering yields better results than the fixed order, but the relaxation guided approach outperforms both of the naive orderings.

5 Extending RGVNS for the MKP

The pure RGVNS approach of the previous section yields relatively good results, but it cannot directly compete with today's state-of-the-art metaheuristics for the MKP. We therefore extend the RGVNS described in the previous section by some standard neighborhood structures which can be searched in faster ways. These simpler neighborhood structures are

[1] http://people.brunel.ac.uk/~mastjjb/jeb/info.html

not ordered according to relaxations, but explored in a fixed order, before relying on the relaxation based ordering for calling $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$ with $\kappa = 1, \ldots, \kappa_{\max}$.

Furthermore, we introduce an additional parameter $\beta_{\max}$ that limits the number of ILP-based neighborhoods to be explored within one call of RGVND. In this way, shaking is performed more frequently and diversification is emphasized. If, for example, we choose $\beta_{\max} = 10$ and $\kappa_{\max} = 10$, a total of 20 ILP-based neighborhoods are sorted according to their LP-relaxations, but only the 10 most promising are actually evaluated in the current call of VND.

5.1 Swap Neighborhood

The first neighborhood we use is a simple swap $SWP(x^f)$, where a pair of items $(x_i^f, x_j^f) \mid i \in I_1$ and $j \in I_0$ is exchanged, i.e. $x_i^f := 0$ and $x_j^f := 1$. Infeasible solutions are discarded. Note that this neighborhood is contained in both, $IRF(x^f, 1)$ and $IAR(x^f, 1)$. Therefore, including it in the relaxation based ordering does not make sense. Its main advantage is its simplicity and speed.

5.2 Greedy Neighborhoods

Based on the ideas of Chu and Beasley (1998) and as another simplification of IRF and IAR but an extension of SWP, we define two additional neighborhoods based on greedy concepts.

In the first case, the *Remove-and-Greedy-Fill* neighborhood $RGF(x^f, \kappa)$, κ items are removed from x^f, i.e. a κ-tuple of variables from $I_1(x^f)$ are inverted. The resulting solution is then locally optimized using the greedy first-fit heuristic from Section 4.4.

In the second case, the *Add-and-Greedy-Repair* neighborhood $AGR(x^f, \kappa)$, κ items are added to x^f, i.e. κ variables from $I_0(x^f)$ are inverted. The resulting solution, which is usually infeasible, is then repaired and locally improved using the greedy algorithms from Section 4.4.

5.3 Computational Experiments

In these experiments, we combined the simpler neighborhoods, which were explored using a best improvement strategy, with the ILP-based neighborhoods. This time, we used the whole set of instances of Chu and Beasley's benchmark library with $n = 500$ items, $m \in$

$\{5, 10, 30\}$ constraints, and tightness-ratios $\alpha = \{0.25, 0.5, 0.75\}$. Ten different instances exist for each parameter combination yielding 90 instances in total. As before, the experiments were performed on a 2.4GHz Intel Pentium 4 machine, and each run was terminated after 500s of CPU-time. The neighborhoods are ordered as follows: $\mathcal{N}_1$:=$SWP(x^f)$, $\mathcal{N}_2$:=$RGF(x^f, 1)$, $\mathcal{N}_3$:=$AGR(x^f, 1)$, $\mathcal{N}_4$:=$RGF(x^f, 2)$, $\mathcal{N}_5$:=$AGR(x^f, 2)$.

In Table 2 we show results of the following algorithm variants: VNS with neighborhoods $\mathcal{N}_1$ to $\mathcal{N}_3$ only (VNS–$\mathcal{N}_{1-3}$), VNS with neighborhoods $\mathcal{N}_1$ to $\mathcal{N}_5$ only (VNS–$\mathcal{N}_{1-5}$), VNS with the ILP-based neighborhoods (VNS), RGVNS with the ILP-based neighborhoods (RGVNS), RGVNS with additionally $\mathcal{N}_1$ to $\mathcal{N}_3$, and RGVNS with additionally $\mathcal{N}_1$ to $\mathcal{N}_5$. Listed are average percentage gaps and numbers *#-best* indicating for how many instances each approach could find the overall best solutions among the ten instances per group. In the last column (VV05) the percentage gaps of the approach presented in Vasquez and Vimont (2005) are shown, note that run-times of more than 80 hours were needed for some instances.

We can observe a clear performance difference between $\mathcal{N}_{1-3}$, $\mathcal{N}_{1-5}$, and the ILP-based neighborhoods. RGVNS+$\mathcal{N}_{1-3}$ yields the best total average percentage gap. Furthermore the two RGVNS+ methods yield the highest number of best solutions found.

The fact that the RGVNS variants in general yield the best average percentage gaps over all categories together with the results from the previous section clearly documents the benefits of sorting the neighborhoods according to a dynamically determined potential for improvement.

The RGVNS approach presented here is not able to compete with the approach from Vasquez and Vimont (2005) in terms of solution quality. Enhancing RGVNS with other ideas such as the MKP cores or parallel-execution of different algorithms (Puchinger et al., 2005, 2006; Puchinger, 2006) can strongly increase its capabilities, and results close to the ones obtained in Vasquez and Vimont (2005) have been achieved in significantly shorter running-times.

6 Conclusion

We presented an extension of traditional variable neighborhood search: Relaxation Guided Variable Neighborhood Search (RGVNS). The order in which the neighborhoods are investigated is dynamically determined by estimating their improvement-potential using quickly determined solutions to relaxations. This idea seems to be particularly useful if the order of the neighborhoods is not obvious and their relaxations can be quickly solved and yield

relatively tight bounds. We tested this approach on standard benchmark instances of the multidimensional knapsack problem. The results obtained in our computational experiments show a clear advantage of RGVNS compared to VNS without guidance. In the future we want to apply RGVNS on other problems in order to gain further experience with the presented approach. Also, we will study further possibilities of relaxations, such as surrogate and Lagrangian relaxations.

References

Chu, P. C. and J. Beasley (1998). "A genetic algorithm for the multiconstrained knapsack problem." *Journal of Heuristics* 4, 63–86.

Dumitrescu, I. and T. Stuetzle (2003). "Combinations of Local Search and Exact Algorithms." In *Applications of Evolutionary Computation*, G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, and E. Marchiori, eds. Springer, volume 2611 of *LNCS*, 211–223.

Fréville, A. (2004). "The multidimensional 0-1 knapsack problem: An overview." *European Journal of Operational Research* 155, 1–21.

Gavish, B. and H. Pirkul (1985). "Efficient algorithms for solving the multiconstraint zero-one knapsack problem to optimality." *Mathematical Programming* 31, 78–105.

Gilmore, P. and R. Gomory (1966). "The theory and computation of knapsack functions." *Operations Research* 14, 1045–1075.

Glover, F. and G. Kochenberger (1996). "Critical event tabu search for multidimensional knapsack problems." In *Metaheuristics: Theory and Applications*, I. Osman and J. Kelly, eds., Kluwer Academic Publishers. 407–427.

Hansen, P. and N. Mladenović (1999). "An Introduction to Variable Neighborhood Search." In *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. Osman, and C. Roucairol, eds., Kluwer. 433–458.

Hansen, P. and N. Mladenović (2003). "A Tutorial on Variable Neighborhood Search." Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada.

Kellerer, H., U. Pferschy, and D. Pisinger (2004). *Knapsack Problems*. Springer.

Lorie, J. and L. Savage (1955). "Three problems in capital rationing." *The Journal of Business* 28, 229–239.

Manne, A. and H. Markowitz (1957). "On the solution of discrete programming problems." *Econometrica* 25, 84–110.

Puchinger, J. (2006). *Combining Metaheuristics and Integer Programming for Solving Cutting and Packing Problems*. Ph.D. thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms.

Puchinger, J. and G. R. Raidl (2005a). "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification." In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*. Springer, volume 3562 of *LNCS*, 41–53.

Puchinger, J. and G. R. Raidl (2005b). "Relaxation Guided Variable Neighborhood Search." In *Proceedings of the XVIII Mini EURO Conference on VNS*. Tenerife, Spain.

Puchinger, J., G. R. Raidl, and M. Gruber (2005). "Cooperating Memetic and Branch-and-Cut Algorithms for Solving the Multidimensional Knapsack Problem." In *Proceedings of MIC2005, the 6th Metaheuristics International Conference*. Vienna, Austria, 775–780.

Puchinger, J., G. R. Raidl, and U. Pferschy (2006). "The Core Concept for the Multidimensional Knapsack Problem." In *Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006*. Springer, volume 3906 of *LNCS*, 195–208.

Raidl, G. R. and J. Gottlieb (2005). "Empirical Analysis of Locality, Heritability and Heuristic Bias in Evolutionary Algorithms: A Case Study for the Multidimensional Knapsack Problem." *Evolutionary Computation Journal* 13(4), 441–475.

Shih, W. (1979). "A branch and bound method for the multiconstraint zero-one knapsack problem." *Journal of the Operational Research Society* 30, 369–378.

Vasquez, M. and J.-K. Hao (2001). "A Hybrid Approach for the 0–1 Multidimensional Knapsack Problem." In *Proceedings of the Int. Joint Conference on Artificial Intelligence 2001*. 328–333.

Vasquez, M. and Y. Vimont (2005). "Improved results on the 0-1 multidimensional knapsack problem." *European Journal of Operational Research* 165, 70–81.

Weingartner, H. M. and D. N. Ness (1967). "Methods for the solution of the multidimensional 0/1 knapsack problem." *Operations Research* 15, 83–103.

Wolsey, L. A. (1998). *Integer Programming*. Wiley-Interscience.

Table 1: Comparison of VNS, RandVNS, and RGVNS; listed are average and median percentage gaps, standard deviations in parentheses, and error probabilities of Wilcoxon rank sum tests indicating the significance of differences.

		VNS		RandVNS		RGVNS		$p_{\text{VNS,RGVNS}}$	$p_{\text{RandVNS,RGVNS}}$
m	α	mean	median	mean	median	mean	median	W-test	W-test
5	0.25	0.091	0.096	0.088	0.088	**0.082**	**0.076**	0.04	< 0.01
		(0.011)		(0.006)		(0.010)			
	0.5	0.042	0.041	0.037	0.036	**0.034**	**0.034**	< 0.01	< 0.01
		(0.005)		(0.004)		(0.000)			
	0.75	**0.023**	**0.023**	**0.023**	**0.023**	**0.023**	**0.023**	n.a.	n.a.
		(0.000)		(0.000)		(0.000)			
10	0.25	0.251	0.251	0.229	0.236	**0.212**	**0.204**	< 0.01	< 0.01
		(0.018)		(0.025)		(0.016)			
	0.5	0.115	**0.108**	**0.105**	**0.108**	0.108	**0.108**	< 0.01	0.42
		(0.009)		(0.009)		(0.007)			
	0.75	0.073	0.075	**0.071**	**0.070**	0.075	0.079	0.19	< 0.01
		(0.003)		(0.003)		(0.005)			
30	0.25	0.685	0.686	0.639	0.642	**0.635**	**0.614**	< 0.01	0.383
		(0.047)		(0.025)		(0.034)			
	0.5	0.291	0.304	**0.256**	**0.244**	0.272	0.277	< 0.01	< 0.01
		(0.032)		(0.019)		(0.022)			
	0.75	0.152	0.154	0.139	0.0136	**0.131**	**0.131**	< 0.01	< 0.01
		(0.016)		0.009		(0.000)			

Table 2: Average results of the different approaches, using the whole Chu and Beasley 500-variables instance set.

		VNS–$\mathcal{N}_{1-3}$		VNS–$\mathcal{N}_{1-5}$		VNS		RGVNS		RGVNS+$\mathcal{N}_{1-3}$		RGVNS+$\mathcal{N}_{1-5}$		VV05
m	α	$\overline{\%-gap}$	$\#$-$best$	$\overline{\%-gap}$	$\#$-$best$	$\overline{\%-gap}$	$\#$-$best$	$\overline{\%-gap}$	$\#$-$best$	$\overline{\%-gap}$	$\#$-$best$	$\overline{\%-gap}$	$\#$-$best$	$\overline{\%-gap}$
5	0.25	0.124	0	0.109	0	0.113	2	0.090	2	0.088	4	**0.087**	6	0.074
	0.5	0.065	0	0.053	0	0.049	0	**0.042**	7	0.043	5	**0.042**	4	0.038
	0.75	0.041	1	0.029	2	0.032	1	**0.026**	7	0.027	6	**0.026**	8	0.024
10	0.25	0.357	0	0.293	0	0.271	1	0.234	5	**0.230**	6	0.232	4	0.174
	0.5	0.180	0	0.137	0	0.131	0	0.108	4	0.108	3	**0.103**	8	0.082
	0.75	0.103	0	0.083	0	0.084	1	**0.069**	7	**0.069**	6	0.072	5	0.057
30	0.25	0.890	0	0.825	0	0.716	0	0.609	5	**0.595**	6	0.615	4	0.482
	0.5	0.414	0	0.332	0	0.310	0	0.265	4	**0.263**	6	0.268	3	0.210
	0.75	0.232	0	0.216	0	0.189	1	**0.167**	3	0.168	3	**0.167**	3	0.135
Average		0.267	0.1	0.231	0.2	0.211	0.7	0.179	4.9	**0.177**	5.0	0.179	5.0	0.142