Relaxation Guided Variable Neighborhood Search [*]

Jakob Puchinger[1], Günther R. Raidl[1]

[1]Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{japu|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 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.

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

1 Introduction

We want to investigate in depth a new variant of Variable Neighborhood Search (VNS) [2, 3]: *Relaxation Guided Variable Neighborhood Search.* It is based on a standard VNS scheme and a new Variable Neighborhood Descent (VND) algorithm. How to order the given neighborhoods is often

[*]This work is supported by the RTN ADONET under grant 504438 and the Austrian Science Fund (FWF) under grant P16263-N04.

a difficult and performance-significant decision. We guide VND by always sorting the neighborhoods according to estimations of the improvement-potentials in dependence of 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 VNS 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. The main purpose of this work is to evaluate our new VNS variant.

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) is following the general VNS scheme [2, 3] incorporating an improved VND which we call Relaxation Guided Variable Neighborhood Descent (RGVND).

Let us assume that the neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_{l_{\max}}$ are to be used within VND. A significant question, which is often of crucial importance for the algorithm's performance, is the order in which the neighborhoods shall be considered. Often, rules of thumb, such as searching smaller neighborhoods or neighborhoods which are considered to be more promising in some sense first, are used. However, in many situations it is not straight-forward to find the ideal ordering. Furthermore, the ordering that is best suited in a particular situation might in general depend on the current solution. 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 main point of our extended variant of VND is that we control the order in which the neighborhood structures are processed by estimating improvement-potentials. These potentials are devised by quickly solving a relaxation of each neighborhood structure. We expect that this scheme allows to explore more promising neighborhoods earlier, yielding better and faster overall results.

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

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

with S being a finite set of solutions and $f(x) : S \to \mathbb{R}$ an objective function. We can now introduce the following formal definition of a relaxation [9].

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

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

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 [9] yields a bound for COP.

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

Often, it is substantially faster to calculate the optimal solution of a relaxation than of the original problem. An example is the widely used linear programming (LP) relaxation of an integer linear programming (ILP) formulation of a COP, which can be solved in polynomial time. It is a prerequisite for the RGVND scheme that the used relaxations can be solved to optimality much faster than their corresponding original neighborhood structures.

A second precondition on the used neighborhoods is that they are not fully contained in each other since this would lead to trivial orderings and make our approach meaningless. Therefore $\mathcal{N}_l \not\subseteq \mathcal{N}_{l'}$ and $\mathcal{N}_{l'} \not\subseteq \mathcal{N}_l$ must hold for any $\mathcal{N}_l, \mathcal{N}_{l'}$ with $l \neq l'$.

In Algorithm 1 the pseudocode of RGVND is given. The significant differences to the standard VND scheme, as described in [2, 3], 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_l^{R}, and then sorting the neighborhoods according to decreasing z_l^{R}. Ties are broken arbitrarily or according to some static heuristic rules.

Algorithm 1: Relaxation Guided VND (RGVND)

```
   Input: A feasible solution $x$
1  $l \leftarrow 1$
2  $\pi$ = DetermineOrderOfNeighborhoods($x$)
3  repeat
4      Find the best neighbor $x^* \in \mathcal{N}_{\pi(l)}(x) \mid f(x^*) \geq f(x') \; \forall x' \in \mathcal{N}_{\pi(l)}(x)$
5      if $f(x^*) > f(x)$ then
6          $x \leftarrow x^*$
7          $l \leftarrow 1$
8          $\pi$ = DetermineOrderOfNeighborhoods($x$)
9      else
10         $l \leftarrow l + 1$
11 until $l = l_{\max}$
12 return $x$
```

Algorithm 2: DetermineOrderOfNeighborhoods(x)

```
1  for $l = 1, \ldots, l_{\max}$ do
2      Solve $\mathcal{N}_l^{\mathrm{R}}(x)$ yielding solution value $z_l^{\mathrm{R}}$
3  Sort $\pi = (1, \ldots, l_{\max})$ according to decreasing $z_l^{\mathrm{R}}$
4  return $\pi$
```

3 The Multidimensional Knapsack Problem

In order to evaluate RGVNS we used the MKP, which is a commonly used benchmark for metaheuristics. The MKP can be defined by the following Integer Linear Program (ILP):

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

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i = 1, \ldots, m \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3}$$

Given are n items with profits $p_j > 0$ and m resources with capacities $c_i > 0$. The 0–1 decision variables x_j indicate which items are selected. Each item j consumes an amount $w_{ij} \geq 0$ from each resource i. The goal is to select a subset of the items with maximum total profit, see (1); chosen items must, however, not exceed resource capacities, see (2).

A general overview on practical and theoretical results for the MKP can be found in the monograph by Kellerer et al. [4]. Besides exact techniques for solving small to moderately sized instances, many kinds of metaheuristics have already been applied to the MKP. To our knowledge, the method currently yielding the best heuristic results, at least for commonly used benchmark instances, is a tabu-search / linear programming based approach described by Vasquez and Hao [7]. It was recently refined by Vasquez and Vimont [8]. Besides this tabu search approach, several variants of hybrid evolutionary algorithms have been described; see [6] for a recent survey and comparison of evolutionary approaches for the MKP. The basics of today's most effective evolutionary algorithms go back to Chu and Beasley [1]: Candidate solutions are directly represented by their 0–1 vectors x; standard crossover and mutation operators and – most importantly – clever repair and local improvement strategies are applied. In [5] we presented a collaborative strategy, in which a memetic algorithm and an ILP-based exact approach are executed in parallel exchanging information about the ongoing optimization process. The results obtained were competitive to [7] and [8].

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. 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, which is determined by sorting the items according to decreasing values of the solutions to the MKP's LP-relaxation; see [6].

4.2 ILP Based Neighborhoods

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* $\mathit{IRF}(x^f, k)$, we force precisely k 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 (1)–(3):

$$\sum_{j \in I_1(x^f)} x_j = \sum_{j \in I_1(x^f)} x_j^f - k. \tag{4}$$

In the second neighborhood, *ILP-Add-and-Remove* $\mathit{IAR}(x^f, k)$, we force precisely k 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 (1)–(3):

$$\sum_{j \in I_0(x^f)} x_j = k. \tag{5}$$

As relaxations $\mathit{IRF}^R(x^f, k)$ and $\mathit{IAR}^R(x^f, k)$ we use the corresponding LP-relaxations in which the integrality constraints (3) are replaced by $0 \le x_j \le 1$, $j = 1, \ldots, n$. Note that depending on the specific instance's characteristics, both neighborhoods may become quite large even for $k = 1$. 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) with a certain time limit.

4.3 Relaxation Guided VNS

The Relaxation Guided Variable Neighborhood Search (RGVNS) is based on the previously defined neighborhoods $\mathit{IRF}(x^f, k)$ and $\mathit{IAR}(x^f, k)$. We first solve the LP-relaxations of $\mathit{IRF}(x^f, k)$ and $\mathit{IAR}(x^f, k)$ for $k = 1, \ldots, k_{\max}$, where $k_{\max}$ is a prespecified upper limit on the number of items we want to remove or add. The neighborhoods are sorted according to decreasing LP-relaxation solution values. Ties are broken by considering smaller ks earlier.

4.4 Shaking

In the VNS framework, after RGVND has explored all neighborhoods, shaking is performed. Shaking flips κ different, randomly selected variables of the currently best solution and applies greedy repair and local improvement according to [1]: A solution is repaired by removing packed items in an order Π until the solution becomes feasible again. This order Π is determined during preprocessing by sorting all items according to increasing pseudo-utility ratios

$$u_j = \frac{p_j}{\sum_{i=1}^{m} a_i w_{ij}}, \tag{6}$$

where we set the surrogate multipliers a_i to the dual variable values (i.e. the shadow prices of the i-th constraints) of the solution to the LP-relaxation of the MKP. After repairing, a first-fit local improvement is applied, in which the items are considered in the reverse order $\overline{\Pi}$.

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

4.5 Relaxation Guided VNS versus Standard VNS

We compare RGVNS to standard VNS using the ILP based neighborhoods $\mathit{IRF}(x^f, k)$ and $\mathit{IAR}(x^f, k)$, for $k = 1, \ldots, 10$. In RGVNS the neighborhoods are ordered according to their LP-relaxations, whereas in standard VNS the neighborhoods are statically ordered according to increasing k and always switching between $\mathit{IRF}(x^f, k)$ and $\mathit{IAR}(x^f, k)$. We further compare RGVNS to RandVNS where the neighborhoods are always /ordered randomly. For shaking $\kappa_{\max}$ was set to n.

The tested algorithms were implemented in C++ using CPLEX 9.0. The ILP-based neighborhoods are not always fully explored but CPLEX is terminated after at most 2 seconds. The total run-time given to the algorithms was limited to 500 seconds. The experiments were performed on a 2.4GHz Pentium 4 machine.

We used standard benchmark instances for the MKP available at Beasley's OR-Library[1]. The experiments were performed on the first instance of each category with $n = 500$ items, $m \in \{5, 10, 30\}$ constraints, and tightness ratios $\alpha = c_i / \sum_{j=1}^{n} w_{ij} \in \{0.25, 0.5, 0.75\}$. The instances with 500 items are the hardest of this benchmark set, most of them cannot be solved to proven optimality in reasonable run-times using CPLEX. Computing the best known heuristic solutions described in [8] took up to 33 hours.

We performed 30 runs for each of these 9 instances. Table 1 lists the mean and median percentage gaps of the final solutions' objective values with respect to the LP-relaxation, i.e. $(z^R - z)/z^R \cdot 100\%$. Corresponding standard deviations are shown in parentheses. The $p_{\text{VNS,RGVNS}}$ columns list the error probabilities in t-tests and Wilcoxon rank sum tests of the hypotheses that differences exist. These

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

statistical tests were computed using the statistics software R[2].

		VNS		RandVNS		RGVNS		$p_{\text{VNS,RGVNS}}$		$p_{\text{RandVNS,RGVNS}}$	
m	α	mean	median	mean	median	mean	median	t-test	W-test	t-test	W-test
5	0.25	0.091 (0.011)	0.096	0.088 (0.006)	0.088	**0.082** (0.010)	**0.076**	< 0.01	0.04	< 0.01	< 0.01
	0.5	0.042 (0.005)	0.041	0.037 (0.004)	0.036	**0.034** (0.000)	**0.034**	< 0.01	< 0.01	< 0.01	< 0.01
	0.75	**0.023** (0.000)	**0.023**	**0.023** (0.000)	**0.023**	**0.023** (0.000)	**0.023**	n.a.	n.a.	n.a.	n.a.
10	0.25	0.251 (0.018)	0.251	0.229 (0.025)	0.236	**0.212** (0.016)	**0.204**	< 0.01	< 0.01	< 0.01	< 0.01
	0.5	0.115 (0.009)	**0.108**	**0.105** (0.009)	**0.108**	0.108 (0.007)	**0.108**	< 0.01	< 0.01	0.20	0.42
	0.75	0.073 (0.003)	0.075	**0.071** (0.003)	**0.070**	0.075 (0.005)	0.079	0.19	0.19	< 0.01	< 0.01
30	0.25	0.685 (0.047)	0.686	0.639 (0.025)	0.642	**0.635** (0.034)	**0.614**	< 0.01	< 0.01	0.583	0.383
	0.5	0.291 (0.032)	0.304	**0.256** (0.019)	**0.244**	0.272 (0.022)	0.277	< 0.01	< 0.01	< 0.01	< 0.01
	0.75	0.152 (0.016)	0.154	0.139 0.009	0.0136	**0.131** (0.000)	**0.131**	< 0.01	< 0.01	< 0.01	< 0.01

Table 1: Comparison of VNS and RGVNS; listed are average and median percentage gaps, standard deviations in parentheses, and error probabilities $p_{\text{VNS,RGVNS}}$ obtained by t-tests and Wilcoxon rank sum tests.

For seven out of the nine instances RGVNS yields significantly better results than the VNS approach with a 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 yielded slightly better results than RGVNS, however without statistical significance. When comparing RGVNS to RandVNS, one can observe that in four cases RGVNS was significantly better, in one case results were equal, in two cases RandVNS was better, and in two cases no conclusions 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

We now extend the RGVNS described in the previous section by some faster to solve standard neighborhood structures. Those simpler neighborhood structures are not ordered according to relaxations, but explored in a fixed order, before relying on the relaxation based ordering for calling

[2]`http://www.r-project.org/`

$\mathit{IRF}(x^f, k)$ and $IAR(x^f, k)$ with $k = 1, \ldots, k_{\max}$.

5.1 Swap Neighborhood

The first neighborhood we use is a simple swap $SWP(x^f)$, where a pair of items $(x_i^f, x_j^f) \mid i \in I_1 andj \in I_0$ 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, $\mathit{IRF}(x^f, 1)$ and $IAR(x^f, 1)$. Its main advantage is that it can be much faster explored.

5.2 Greedy Neighborhoods

Based on the ideas of Chu and Beasley [1] and as another simplification of IRF and IAR but an extension of SWP, we defined two additional neighborhoods based on greedy concepts.

In the first case, the *Remove-and-Greedy-Fill* neighborhood $RGF(x^f, k)$, k items are removed from x^f, i.e. a k-tuple of variables from $I_1(x^f)$ is flipped. 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 $AGR(x^f, k)$, k items are added to x^f, i.e. k variables from $I_0(x^f)$ are flipped. 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. We used the whole set of Chu and Beasley's 500-variable instances. These experiments were performed on a 2.8GHz Pentium 4 machine and each run was again 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 $\mathcal{N}_5$. Each algorithm was given a run-time of 500 seconds.

m	α	VNS–$\mathcal{N}_{1-3}$	VNS–$\mathcal{N}_{1-5}$	VNS	RGVNS	RGVNS+$\mathcal{N}_{1-3}$	RGVNS+$\mathcal{N}_{1-5}$
5	0.25	0.125	0.102	0.085	**0.084**	**0.084**	0.089
		(0.015)	(0.010)	(0.011)	(0.011)	(0.014)	(0.008)
	0.5	0.059	0.049	**0.042**	**0.042**	**0.042**	0.043
		(0.010)	(0.005)	(0.006)	(0.005)	(0.004)	(0.006)
	0.75	0.038	0.029	**0.026**	0.027	0.027	0.027
		(0.007)	(0.003)	(0.006)	(0.004)	(0.004)	(0.005)
10	0.25	0.319	0.276	0.242	**0.225**	0.23	0.239
		(0.042)	(0.040)	(0.023)	(0.021)	(0.017)	(0.017)
	0.5	0.152	0.131	0.108	0.106	**0.101**	0.108
		(0.014)	(0.011)	(0.009)	(0.009)	(0.010)	(0.012)
	0.75	0.100	0.082	0.073	0.071	**0.069**	0.072
		(0.010)	(0.008)	(0.009)	(0.007)	(0.007)	(0.008)
30	0.25	0.860	0.770	0.619	0.583	**0.581**	0.607
		(0.085)	(0.074)	(0.048)	(0.075)	(0.073)	(0.073)
	0.5	0.383	0.334	0.276	0.264	**0.254**	0.266
		(0.033)	(0.022)	(0.016)	(0.012)	(0.020)	(0.021)
	0.75	0.230	0.192	0.171	**0.159**	0.166	0.159
		(0.018)	(0.017)	(0.009)	(0.015)	(0.013)	(0.013)

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

We can observe a clear performance difference between $\mathcal{N}_{1-3}$, $\mathcal{N}_{1-5}$, and IP which is in most of the cases statistically significant with an error probability below 0.05. RGVNS and RGVNS+$\mathcal{N}_{1-3}$ yielded the best average percentage gaps for almost every instance class. The statistical tests indicate a difference between the relaxation guided methods and the non-guided one with an error probability below 0.1 for the ($m = 5,\ \alpha = 0.5$) and ($m = 5,\ \alpha = 0.75$) classes only. In order to see more significant differences, several runs for each of the instances would be required which was not possible due to time constraints.

Nevertheless, the fact that the RGVNS variants yielded the best average percentage gaps for 8 out of the 9 classes together with the results from the previous section clearly document the benefits of sorting the neighborhoods according to a dynamically determined potential for improvement.

6 Conclusions and Future Work

We presented a new VNS variant: 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. Furthermore we intend to include RGVNS into a cooperative strategy and execute it in parallel with other metaheuristics and exact algorithms.

References

[1] P. C. Chu and J. Beasley. A genetic algorithm for the multiconstrained knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[2] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, 1999.

[3] P. Hansen and N. Mladenović. A tutorial on variable neighborhood search. Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, 2003.

[4] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.

[5] J. Puchinger, G. R. Raidl, and M. Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *Proceedings of the sixth Metaheuristics International Conference (MIC). Vienna, Austria*, pages 775–780, 2005.

[6] G. R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation Journal*, 13(4), to appear 2005.

[7] M. Vasquez and J.-K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. In *Proceedings of the International Joint Conference on Artificial Intelligence 2001*, pages 328–333, 2001.

[8] M. Vasquez and Y. Vimont. Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165:70–81, 2005.

[9] L. A. Wolsey. *Integer Programming.* Wiley-Interscience, 1998.