Variable Neighborhood Descent with Self-Adaptive Neighborhood-Ordering

Bin Hu and Günther R. Raidl
Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{hu|raidl}@ads.tuwien.ac.at*

Abstract

In Variable Neighborhood Descent (VND) it is often difficult to decide upon the ordering in which a different types of neighborhoods are considered. This arrangement typically strongly affects the quality of finally obtained solution as well as computation time. We present a VND variant which orders the neighborhoods dynamically in a self-adaptive way during the optimization process. Each neighborhood structure has associated a rating which is updated according to observed success probabilities and required times for evaluation. In this way, more effective neighborhood structures come to the fore and will be applied more frequently. An experimental comparison to a classical VND with a static neighborhood ordering is performed on the generalized edge biconnected network design problem. Results indicate that the self-adaptive VND requires substantially less time for finding solutions of comparable quality.

Keywords: Variable Neighborhood Search, Self-Adaption, Network Design

1 Introduction

Variable Neighborhood Descend (VND) is an enhanced local improvement strategy which is commonly used as a subordinate in Variable Neighborhood Search (VNS) and other metaheuristics [3]. The idea behind VND is to systematically switch between different neighborhood structures $N_1, \ldots, N_n$. Starting with the first structure N_1, VND performs local search until no further improvements are possible. From this local optimum, it continues local search with neighborhood structure N_2. If an improved solution could be found with this structure, VND returns to using N_1 again; otherwise, it continues with N_3, and so forth. If the last structure N_n has been applied and no further improvements are possible, the solution represents a local optimum with respect to all neighborhood structures and VND terminates.

Keeping this concept in mind, it is obvious that the application order of the neighborhood structures is crucial for the performance of VND. The first neighborhood types are searched more often than the ones at the end of the queue. If the times required for examining the neighborhoods differ substantially, it is reasonable to order them according to increasing costs. However, this criterion is not always applicable, in particular when the times for searching the neighborhoods are similar, or they vary strongly. The latter is often the case when applying a next-improvement strategy instead of best-improvement. In these situations, the best suited neighborhood ordering may also depend on specific properties of the particular problem instance and current state of the search. Research in the direction of controlling and dynamically adapting the ordering of neighborhood structures is yet limited. For example, Puchinger and Raidl [6] presented an approach in which relaxations of the neighborhoods are quickly evaluated in order to choose the most promising neighborhood next. This method is effective, however, it requires the existence of fast methods for solving relaxations of the neighborhoods. In this work, we propose a more generally applicable self-adaptive strategy.

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

2 Self-Adaptive Variable Neighborhood Descend

In Self-Adaptive Variable Neighborhood Descent (SAVND) neighborhood structures are dynamically rearranged according to their observed benefits in the past. An initial neighborhood ordering, i.e., a permutation $\lambda = (\lambda_1, \ldots, \lambda_n)$ of $\{1, \ldots, n\}$ is chosen in some intuitive way (or even at random). Each structure N_i, $i = 1, \ldots, n$, gets assigned a rating $w_i > 0$, which is initially set to some constant value W being a rough estimation of the average time for evaluating a neighborhood. During the search process, when a neighborhood $N_{\lambda_i}(x)$ of a current solution x has been investigated, rating w_{λ_i} is updated in dependence of the success and the computation time t_{λ_i} required for the evaluation: If an improved solution has been found in $N_{\lambda_i}(x)$, w_{λ_i} becomes halved and $\frac{t_{\lambda_i}}{\alpha}$ is added; α is a strategy parameter controlling the influence of the evaluation time in this case. If the search of $N_{\lambda_i}(x)$ was not able to identify a superior solution, we add time t_{λ_i} to w_{λ_i}. Permutation λ is not immediately updated after processing a neighborhood in order to avoid too rapid and strong adaptions in case of a temporarily limited extraordinary good or bad behavior. Only when an updated rating w'_{λ_i} is smaller than the so far minimum rating $\min_{j=1,\ldots,n} w_j$ or larger than the maximum rating $\max_{j=1,\ldots,n} w_j$, we redetermine permutation λ by sorting the neighborhood structures according to increasing ratings. SAVND then continues with the structure that would have also been chosen according to the old ordering. Algorithm 1 shows the whole procedure in detail.

3 The Generalized Edge Biconnected Network Design Problem

We will compare SAVND with classical VND using a static neighborhood ordering on the Generalized Edge Biconnected Network Design Problem (GMEBCNP) [1, 4]. Given an undirected graph $G = (V, E)$ where edges $e \in E$ have associated costs $c_e > 0$ and the node set V is partitioned into disjoint clusters $V_1, \ldots, V_r$, the objective is to find a minimum cost edge-biconnected subgraph $G_S = (V_S, E_S)$ that spans exactly one node of each cluster, see Figure 1.

We apply five types of neighborhoods, following different ideas of exchanging the spanned

Algorithm 1: SAVND(solution x)

$w_1 = w_2 = \ldots = w_n = W$
$w_{\min} = w_{\max} = W$
$\lambda = (1, 2, \ldots, n)$
$i = 1$
repeat
  find the best neighbor $x' \in N_{\lambda_i}(x)$, requiring time t_{λ_i}
  if x' better than x **then**
    $x = x'$
    $w_{\lambda_i} = \frac{w_{\lambda_i}}{2} + \frac{t_{\lambda_i}}{\alpha}$
    $i = 1$
  else
    $w_{\lambda_i} = w_{\lambda_i} + t_{\lambda_i}$
    $i = i + 1$
  if $w_{\lambda_i} < w_{\min} \vee w_{\lambda_i} > w_{\max}$ **then**
    nextN $= \lambda_i$ *// store the neighborhood to be considered next*
    sort $\lambda_1, \ldots, \lambda_n$ s.t. $w_{\lambda_1} \leq w_{\lambda_2} \leq \ldots \leq w_{\lambda_n}$
    $w_{\min} = w_{\lambda_1}$
    $w_{\max} = w_{\lambda_n}$
    reset i s.t. $\lambda_i =$ *nextN*
until $i > n$

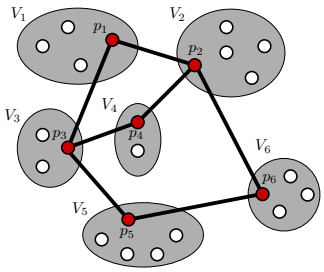

Figure 1: Example for a solution to the GMEBCNP.

nodes as well as the used edges. Some of these neighborhoods follow the concept of very large scale neighborhood search and utilize dynamic programming for an efficient evaluation. A detailed description of the neighborhoods and their evaluation strategies, as well as used greedy heuristics for creating initial solutions can be found in [5].

4 Experimental Results

For SAVND, strategy parameters were set to $W = \frac{1}{10}$ and $\alpha = 10$. The initial neighborhood ordering, as well as the one used for VND, is based on increasing computational complexity [5].

Table 1 shows results on randomly generated instances of the following types: Euclidean instances with grid clustering, Euclidean instances with random clustering, and Non-Euclidean instances; see [5] for further details. Listed are for each instance category the total number of nodes $|V|$, number of clusters r, and for both, VND and SAVND, objective values obj of final solutions and total CPU-times t. Each category consists of three instances and average values are given. All graphs are complete, i.e., $|E| = \frac{|V|\cdot(|V|-1)}{2}$. Table 2 shows further results on TSPlib[1] instances with geographical center clustering [2]. As some of the neighborhood evaluation strategies contain stochastic components, average objective values and average times over 30 runs are provided.

When we compare the final objective values of VND and SAVND, differences exist. However, they are relatively small, and over all instances, no strategy yields statistically significantly better solutions than the other. Comparing running times, the advantages of SAVND become very clear: It consistently requires significantly less time.

Table 1: Results on random instances, three instances per category.

Random Instances			VND		SAVND	
Category	$\|V\|$	r	$\overline{obj}$	$\bar{t}$ [s]	$\overline{obj}$	$\bar{t}$ [s]
Grouped Eucl. 125	125	25	178.36	5.90	180.65	3.31
Grouped Eucl. 500	500	100	766.11	161.41	762.85	95.69
Grouped Eucl. 600	600	20	115.49	63.34	116.26	48.10
Grouped Eucl. 1280	1280	64	467.70	221.28	476.15	184.61
Random Eucl. 250	250	50	4143.14	54.28	4049.57	30.67
Random Eucl. 400	400	20	1132.70	51.51	1211.81	38.73
Random Eucl. 600	600	20	948.83	107.85	1088.40	82.44
Non-Eucl. 200	200	20	486.11	6.68	492.44	3.83
Non-Eucl. 400	500	100	1141.04	66.96	1175.01	31.63
Non-Eucl. 600	600	20	266.03	34.63	253.47	25.96

[1] http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html

Table 2: Results on TSPlib instances with geographical clustering.

TSPlib Instances			VND		SAVND	
Name	$\|V\|$	r	$\overline{obj}$	$\bar{t}$ [s]	$\overline{obj}$	$\bar{t}$ [s]
gr137	137	28	505.60	10.33	490.23	5.30
kroa150	150	30	12470.77	9.32	12562.13	6.36
d198	198	40	12330.90	37.7	12435.03	17.55
krob200	200	40	13906.47	29.49	14010.97	14.43
gr202	202	41	344.53	30.84	341.13	22.67
ts225	225	45	77418.33	18.63	77691.67	13.77
gil262	262	53	1186.63	66.71	1173.23	21.30
pr264	264	54	34691.27	70.87	35506.70	28.90
pr299	299	60	24887.30	46.73	24839.63	32.66
lin318	318	64	26285.40	41.47	26535.23	22.64
rd400	400	80	7891.80	92.65	7532.13	53.47
fl417	417	84	11042.77	89.44	11003.57	53.29
gr431	431	87	1557.83	78.95	1520.57	72.27
pr439	439	88	77427.03	116.65	78338.93	66.65
pcb442	442	89	26669.00	83.05	26881.70	43.79

5 Conclusions

We proposed a technique to speed up Variable Neighborhood Descend by allowing the order in which neighborhoods are searched to self-adapt dynamically during the search process. As criteria for controlling self-adaption, we used measured times required for evaluating the neighborhoods and the the success of neighborhood structures in terms of how likely they lead to improved solutions. Experimental results on the generalized edge biconnected network design problem consistently document the advantages of the approach with respect to running time.

References

[1] C. Feremans, M. Labbe, and G. Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.

[2] M. Fischetti, J. J. Salazar, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45:378–394, 1997.

[3] P. Hansen and N. Mladenovic. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics, advances and trends in local search paradigms for optimization*, pages 433–458. Kluwer Academic Publishers, 1999.

[4] D. Hyugens. Version generalisee du probleme de conception de reseau 2-arete-connexe. Master's thesis, Universite Libre de Bruxelles, Brussels, Belgium, 2002.

[5] M. Leitner. Solving two generalized network design problems with exact and heuristic methods. Master's thesis, Vienna University of Technology, Vienna, Austria, 2006.

[6] J. Puchinger and G. R. Raidl. Relaxation guided variable neighborhood search. In P. Hansen et al., editors, *Proceedings of the 18th Mini EURO Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.