Neighborhoods Revisited: An Experimental Investigation into the Effectiveness of Variable Neighborhood Descent for Scheduling*

Matthijs den Besten Thomas Stützle

Intellectics Group, Darmstadt University of Technology
Alexanderstraße 10, D-64283 Darmstadt
Email: {matthijs,tom}@intellektik.informatik.tu-darmstadt.de

1 Introduction

Variable neighborhood search (VNS) systematically exploits the idea to alternate between neighborhoods within a local search [4]. Several VNS variants of have been described so far and many of them have been applied with success to a variety of combinatorial optimization problems; see [3, 4] for an overview. Yet, even though scheduling is a central problem in production environments, we are not aware of any VNS applications to scheduling problems. In this paper we focus on the application of a particular VNS technique called variable neighborhood descent [3] to scheduling problems in which a solution can be represented as one permutation of all jobs. This class of scheduling problems includes single machine problems as well as a variety of multiple machine problems. In this abstract, we present some results of VND when applied to single machine problems minimizing the total tardiness problem, the total weighted tardiness, and the sum of weighted completion times, as well as to the permutation flow shop problem. In addition we investigate whether VND local search performs better than a single neighborhood local search algorithm when the local search is used as a sub-procedure within a metaheuristic like iterated local search [6].

2 Variable neighborhood descent for scheduling problems

An iterative improvement local search algorithm is characterized by its solution *representation*, the type of applicable solution modifications defined by some *neighborhood*, and the strategy, also known as *pivoting rule*, it employs to define which neighbored solution replaces the current one. In the scheduling problems we consider, a solution can be represented as a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of n jobs. The standard neighborhoods for permutation scheduling problems are transpose, interchange and insert. *Transpose* is the smallest neighborhood. It consists of all permutations that can be obtained by swapping adjacent jobs in the permutation and has size $n-1$. The *interchange* neighborhood consists of all permutations that can be obtained by swapping two elements at the ith and jth position $(i \neq j)$ regardless of their adjacency. The size of this neighborhood is $n \cdot (n-1)/2$. Finally, the *insert* neighborhood contains all permutations that can be obtained by removing an element at the ith position and inserting it in the jth position $(i \neq j)$, resulting in a neighborhood size of $(n-1)^2$. Obviously, the transpose neighborhood is a subset of the interchange and insert neighborhoods, while the interchange

*This work was partially supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

procedure *basic VND*
  Initialize: $x = \mathsf{GenerateInitialSolution}$, Select set of neighborhood structures $\mathcal{N}_k,\ k = 1, \ldots, k_{\max}$
  repeat
    $k = 1$
    repeat
      $x' = \mathsf{BestNeighbor}(\mathcal{N}_k(x))$
      if (x' better than x)
        $x = x'$
      else
        $k = k + 1$
    until $k = k_{\max}$
  until no improvement
end

Figure 1: Outline of variable neighborhood descent

procedure *IPS*
  Initialize: $x = \mathsf{GenerateInitialSolution}$
  repeat
    $x' = \mathsf{ModifySolution}(x)$
    $x'_1 = \mathsf{A}_1(x');\ \ldots;\ x'_k = \mathsf{A}_k(x'_{k-1})$
    if (accept(x'_k))
      $x = x'_k$
  until termination criterion is met
end

Figure 2: Outline of iterated piped search

and the insert neighborhood are complementary parts of a bigger *2-opt* neighborhood that contains both permutations that result from a normal exchange of elements and permutations obtained after an exchange of an element with an *empty* element. Finally, to fully define an iterative improvement local search algorithm, a pivoting rule has to be defined. Standard pivoting rules are *best* improvement that applies the best possible move of the neighborhood and *first* improvement that immediately applies a move as soon an improved solution is found. In the last case, also the order of the scan is important. So, already for the implementation of standard local search, a lot of design issues have to be faced. The VNS metaheuristic opens up the configuration even further: Not just one neighborhood should be chosen, but a *set* of neighborhoods, and on top of the standard neighborhood scan, one has to define in which order to scan the different neighborhoods.

The basic variable neighborhood descent algorithm [3] is an extension of standard local search (see Figure 1 for an outline). Basic VND iteratively explores neighborhoods $\mathcal{N}_k,\ k = 1, \ldots, k_{\max}$ for the descent and applies a best improvement strategy to each of the neighborhoods. VND can be successful because a local optimum within one neighborhood is not necessarily a local optimum for a different neighborhood. So, changing the neighborhood can result in better local optima. However, the performance of the algorithm depends crucially on the choice of the neighborhoods and the way they are ordered. In applications to permutation scheduling problems, the transpose neighborhood can only be used as the first neighborhood, because the set of moves it considers is a subset of the moves of the other two neighborhoods. Still, the use of the transpose neighborhood may speed-up the local search in total, because this neighborhood can be explored very quickly (results with the transpose neighborhood will only be presented in the full paper). The best order of the interchange and the insert neighborhood will be problem dependent, but intuitively one might think that in most situations it may be more reasonable to first apply the smaller neighborhood, which in this case is the interchange neighborhood.

In our VND approach we assume that the local search algorithm for each neighborhood $\mathcal{N}_i$ is given as a black box procedure A_i [5]. Our VND is therefore a concatenation of “black boxes”. Since we *pipe* the output of procedure A_{i-1} as the input into procedure A_i, we denote this concatenation as $\mathsf{A}_{i-1}|\mathsf{A}_i$ (the $|$ symbol is chosen analogous to the Unix shell conventions on piping). Obviously, piping algorithms is only useful, if the procedures are truely different. In VND this is the case if the procedures implement local search algorithms using different neighborhoods. Otherwise, if $\mathcal{N}_{i-1} \subset \mathcal{N}_i$ mainly a speed advantage may arise by exploring first smaller neighborhoods. Yet, the idea of piping algorithms is much more general than VNS: The algorithms may differ in completely other aspects than the neighborhood to be used in a local search. For example, phases of search space exploration and exploitation can be alternated by piping algorithms implementing these two features. A different idea may be to pipe procedures which use different representations of a problem to avoid the problem of local optimality. In fact, this is the idea underlying the *shifting* approach [1]. Our implementation of Variable Neighborhood Descent is an example of such an "iterated piped search"(see Figure 2).

3 Scheduling problems

In this abstract we present results on the effectiveness of VND for four $\mathcal{NP}$–hard scheduling problems.

Single machine total tardiness problem (SMTTP). In the SMTTP each of n jobs has to be processed without any interruption on a single machine that can only process one job at a time. Each job has a processing time p_j, and a due date d_j associated and the jobs become available for processing at time zero. The tardiness of a job is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j in the current sequence of jobs. The goal in the SMTTP then is to find a sequence of the jobs which minimizes the sum of the tardiness given by $\sum_{i=1}^{n} T_i$.

Single machine total weighted tardiness problem (SMTWTP). The SMTWTP, a variant of the SMTTP, associates each job with an integer weight w_j (reflecting the importance of the job) and the goal becomes to minimize the sum of the weighted tardiness given by $\sum_{i=1}^{n} w_i \cdot T_i$.

Single machine total weighted completion time problem (SMWCP). In the SMWCP we have given n jobs and each job i has associated a positive weight w_i, a processing time p_i, and a non-negative release date r_i, before which it is unavailable for processing on the machine. The goal in the SMWCP is to minimize $\sum_{i=1}^{n} w_i C_i$, where C_i is the ith job's completion time.

Flow shop problem (FSP). In the FSP, each of n jobs has to be processed on m machines $1, \ldots, m$ in this order. The processing time of job i on machine j is t_{ij} where the t_{ij} are fixed and nonnegative. At any time, each job can be processed on at most one machine, and each machine can process at most one job. The jobs are available for processing at time 0 and the processing of a job may not be interrupted. Here, we focus on the permutation flow shop problem (PFSP), where the job order is the same on every machine. The objective is to find a job sequence π that minimizes the completion time (called makespan) of the last job.

For the extended version of the article we also will add results for the application of VND to FSPs with job weights and the weighted completion time objective, SMWCP with common job weights, as well as other variants of single machine problems. The choice of the problems and their objective functions is motivated by the desirability to have (i) problems in a weighted and unweighted form, (ii) problems where each job has a direct influence or only a rather indirect influence on the objective function (e.g. in the PFSP only the completion time of the last job is measured, while if the objective is the sum of weighted completion times each job's completion time directly goes into the goal), and (iii) the number of machines. We expect these criteria to influence the performance improvement obtained by using VND over the use of a single neighborhood in the local search.

4 Experimental Results

We have run iterative improvement algorithms using the interchange and the insert neighborhood as well as the two VND variants on a large set of benchmark instances. The benchmark instances for the SMTTP, the SMTWTP, and the PFSP are taken from ORLIB (available at `http://mscmga.ms.ic.-ac.uk/`); the SMWCP instances are a subset of the instances available at `http://ebbets.poly.edu/-SCHED/onerj.html`. The computational results for the different local search procedures are given in Tables 1 to 4.

We can summarize the computation results regarding the solution quality as follows: For the SMWCP and the PFSP the insert neighborhood is strongly preferable over the interchange neighborhood, while for the SMTTP and the SMTWTP the performance of the two basic local search

Table 1: Comparison of the local search effectiveness for the SMTTP. Results on the 100 job instances using the interchange, the insert, and the VND variants. We give the average percentage deviation from the best known solutions (Δ_{avg}), the number of best-known solutions found (n_{opt}), and the average CPU time in seconds (t_{avg}) averaged over 125 benchmark instances. The computational results are given for two different construction heuristics (EDD, MDD) for the initial solution.

	interchange			insert			insert | interchange			interchange | insert		
init	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}
EDD	0.53	33	0.19	0.73	41	0.17	0.12	73	0.20	0.07	76	0.20
MDD	0.46	64	0.02	0.12	83	0.02	0.10	93	0.03	0.11	87	0.03

Table 2: Comparison of the local search effectiveness for the SMTWTP. Results on the 100 job instances from ORLIB with the interchange, the insert, and the VND local searches. See Table 1 for a definition for the table entries.

	interchange			insert			insert | interchange			interchange | insert		
init	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}	Δ_{avg}	n_{opt}	t_{avg}
EDD	1.19	38	0.29	2.09	26	0.23	0.46	49	0.30	0.52	42	0.28
MDD	1.31	36	0.32	1.03	33	0.16	0.42	42	0.33	0.37	44	0.20
AU	0.56	39	0.11	0.81	33	0.05	0.34	48	0.12	0.63	50	0.08

algorithms is rather similar and, in addition, the relative performance depends on the initial solution. If we combine the two local search algorithms in a VND local search, resulting in two VND local search variants depending on the order of the interchange and insert local searches, we observe especially on the single machine problems a significant improvement in the solution quality. This improvement can be observed in the much lower percentage deviation from best known solutions over the first local search algorithm applied in the VND. Only for the PFSP the performance improvement through the VND was not very impressive. This is rather due to the fact that only the completion time of one job is reflected in the objective function rather than the PFSP being the only problem with more than one machine. With respect to the order of the interchange and the insert local search in the VND, in most case it seems to be preferable to apply the insert local search, contradicting somewhat the intuition of first applying smaller neighborhoods.

Regarding computation times, it is noteworthy that in all cases the VND local search takes less time than the sum of the computations times necessary for each of the two component local searches when starting from the initial solution. This effect is caused by the good starting solution for the second local search and the fewer number of improvement steps necessary to reach a local optimum. This run-time effect together with the possible improvement in solution quality shows that when applied to scheduling problems, VND can significantly enhance local search performance and that VND may be an easily implementable alternative to more complex local search algorithms like variable descent or dynasearch for such problems.

While for some scheduling problems the VND achieves significantly improved solution quality when compared to its component local search algorithms, it is not immediately clear whether these improvements apply also if VND is used inside more complex metaheuristics that frequently apply a local search algorithm as a subroutine. To investigate this issue we applied the VND algorithm presented here in iterated local search algorithms [6] for the different scheduling problems attacked in this abstract [7, 2]. The results, which we omit here due to space limitations, show that for the single machine problems the ILS algorithms, in fact, achieve best performance when applying the VND local search. Yet, for the PFSP the improvement through the VND is even detrimental: For this problem, we obtained best performance using the CPU-frugal insert local search.

In the full paper we will give a detailed account of our analysis and we extend the results shown here to the application of the VND variants to a number of other scheduling problems. We will also identify the problem features of permutation scheduling problems which make an VND local search

Table 3: Comparison of the local search effectiveness for the SMWCP. All local search algorithms start from the same random initial solutions. We give the average percentage deviation from the best known solutions (Δ_{avg}) and the average CPU time in seconds (t_{avg}) averaged over the 10 benchmark instances for each instance class with 10 instances each.

	interchange		insert		insert | interchange		interchange | insert	
instance	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}
N100A2_P3W1	30.14	0.53	1.48	1.50	0.86	1.58	6.20	1.51
N100A5_P3W1	46.08	0.28	8.44	1.24	6.75	1.33	4.59	1.14
N100A10_P3W1	29.44	0.19	4.58	1.09	2.95	1.17	10.04	0.86
N100A20_P3W1	3.94	0.22	1.83	0.97	1.14	1.04	1.34	0.45
U100A0_P2W1	17.21	1.20	0.78	1.48	0.23	1.58	0.55	2.01
U100A0_P2W2	17.04	1.26	0.84	1.51	0.24	1.61	0.63	2.13
U100A0_P2W3	24.67	0.62	0.76	1.33	0.30	1.42	0.57	1.56

Table 4: Comparison of the local search effectiveness for the PFSP. All local search algorithms start from random initial solutions. Each instance class (indicated by number jobs / number machines) contains 10 instances and we give the average percentage deviation from the best known solutions (Δ_{avg}), and the CPU time in seconds (t_{avg}).

	insert		interchange		insert | interchange		interchange | insert	
size	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}
20 / 20	3.12	0.01	6.09	0.02	2.99	0.01	3.06	0.03
50 / 10	4.92	0.03	6.72	0.23	4.81	0.07	4.91	0.24
50 / 20	5.62	0.08	7.84	0.44	5.47	0.18	5.40	0.49
100 / 20	5.19	0.37	6.61	4.88	5.08	1.18	4.95	4.98

algorithm particularly promising.

References

[1] L. Barbalescu, J.-P. Watson, and L. D. Whitley. Dynamic representations and escaping local optima: Improving genetic algorithms and local search. In *Proceedings of AAAI'2000*, 2000.

[2] M. den Besten, T. Stützle, and M. Dorigo. Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In *Proceedings of EvoWorkshops*, LNCS. Springer, Berlin, 2001.

[3] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. Les Cahiers du GERAD G-98-20, GERAD and École des Hautes Études Commerciales, Montréal, Canada, 1998.

[4] P. Hansen and N. Mladenović. 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 searchs paradigms for optimization*, pages 433–458. Dordrecht, NL, 1999.

[5] P. Hansen and N. Mladenović. Variable neighborhood search: Methods and recent applications. In *Proceedings of MIC'99*, pages 275–280, 1999.

[6] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. To appear. A preliminary version is available at `http://www.intellektik.informatik.tu-darmstadt.de/tom/pub.html`.

[7] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA–98–04, FG Intellektik, TU Darmstadt, August 1998.