# An Efficient Implementation of a Static Move Descriptor-based Local Search Heuristic

Onne Beek[1], Birger Raa[1] and Wout Dullaert[2]
[1]*Dept. Management Information and Operations Management, Ghent University, Tweekerkenstraat 2, Ghent, Belgium*
[2]*Institute of Transport and Maritime Management Antwerp, University of Antwerp, Keizerstraat 64, Antwerp, Belgium*

Keywords: Efficient Local Search, Vehicle Routing, Static Move Descriptors, Heuristic Priority Queue.

Abstract: The Vehicle Routing Problem is a well-studied problem. Since its formulation in 1959, a great number of powerful solution methods have been designed. However, for large scale problems, few techniques are able to consistently find good solutions within an acceptable time limit. In this paper, the concept of Static Move Descriptors, a recently published technique for speeding up Local Search algorithms, is analyzed and an efficient implementation is suggested. We describe several changes that significantly improve the performance of an SMD-based Local Search algorithm. The result is an efficient and flexible technique that can easily be adapted to different metaheuristics and can be combined with other complexity reduction strategies.



## 1 INTRODUCTION

### 1.1 Vehicle Routing Problem

The Vehicle Routing Problem is a general formulation used to model a wide range of common distribution problems. The basic VRP minimizes the total distance required to visit a set of customers using multiple vehicles. This paper deals with the Capacitated VRP, where vehicles have a capacity constraint.

A CVRP instance is defined by a complete graph $G = (V,E)$, with $V$ the customer set and $E$ the edge set. Each edge in $E$ has an associated distance $c_{ij}$, i.e. the cost of traversing edge $(i, j)$. The goal is to find the minimum cost set of routes, starting and ending in $v_0$, so that each vertex in $V$ is visited. The sum of customer demands, $d_i$, assigned to a route may not exceed the capacity of the vehicle, $D_j$.

We will assume a homogenous fleet, $D_j = D \forall j$, and a symmetric distance matrix, $c_{ij} = c_{ji} \forall i, j$, that satisfies the triangle inequality, $c_{ij} \leq c_{ik} + c_{kj} \forall i, j, k$. For a thorough introduction to the VRP, we refer to (Lenstra and Kan, 1981) and (Laporte, 2009).

### 1.2 Heuristics

The solving capabilities of exact methods are limited to relatively small instances. For large scale problems only heuristic methods are a viable option. Simple heuristics, such as the Clarke and Wright savings algorithm (Clarke and Wright, 1964), find decent solutions in minimal time.

Once a solution is found, improvement heuristics, such as the Lin-Kernighan algorithm (Lin, 1965), are used to further improve the solution. However, the result can be far from optimal, as basic heuristics get stuck in local optima. When heuristic solutions are inadequate, metaheuristics are used to strategically guide the search beyond local optima. Modern metaheuristics can find near-optimal solutions, but their performance scales poorly for larger instances.

Popular, high-performance local search-based metaheuristics include Guided Local Search (Voudouris et al., 2010) (Kilby et al., 1997), Tabu Search (Gendreau et al., 1994) (Xu and Kelly, 1996) (Cordeau and Maischberger, 2011) and Variable Neighborhood Search (Mladenovic and Hansen, 1997) (Kytöjoki et al., 2007). Local Search is also used in hybrid metaheuristics (Nguyen and Yoshihara, 2007) (Lee et al., 2010).

This paper deals with Static Move Descriptors (SMD), a new concept proposed by (Zachariadis and Kiranoudis, 2010). To understand the logic behind SMDs and the impact of our suggestions in this paper, we will first discuss the fundamental building blocks of local search based algorithms in 2 and their connection to performance in 2.5. We will then turn to the SMD technique in 3. Our suggestions are explained in 4. Finally, we will compare implementations in 5.

Beek O., Raa B. and Dullaert W..
An Efficient Implementation of a Static Move Descriptor-based Local Search Heuristic.
DOI: 10.5220/0004284102770282
In *Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems* (ICORES-2013), pages 277-282
ISBN: 978-989-8565-40-2

## 2 LOCAL SEARCH

Local Search-based algorithms perform an iterative search through the solution space, by continuously evaluating and making small adjustments to a solution. The search is very intense and highly localized, but requires a guiding system to escape local minima.

A Local Search algorithm exists of several building blocks, or decision rules: the operator set, the search and acceptance strategies, feasibility handling and the guidance system.

### 2.1 Operators

The local search operators define which adjustments can be made to a solution, or which solutions can be reached at each step. Simple operators are cheap to evaluate, but have less potential for improvement. Complex operators can find larger improvements, but require more effort. Most modern algorithms use simple operators with a complex guiding system.

The most popular simple operator is the 2-Opt (Croes, 1958) and its generalization k-Opt (Lin, 1965). Excellent examples of complex operators are Node Ejection Chains (Rego, 2001) and Cyclic Transfers (Thompson and Psaraftis, 1993).

### 2.2 Search and Acceptance

Move selection is a crucial decision. Most authors use a best-accept strategy, where all possible moves are evaluated and the best is executed. Alternatively, the search can accept the first improving move it encounters. This reduces the evaluation effort, but can lead to low quality moves.

For the first-accept strategy, the order in which we evaluate moves dictates which move will be executed. A smart search strategy might find high quality moves in minimal effort, but finding such strategies has proven difficult for the VRP.

### 2.3 Feasibility

Allowing temporary constraint violations can give the search more flexibility. However, it can be difficult to guide the search back to a feasible solution. Most authors prefer to enforce feasibility and guide the search around infeasible regions.

### 2.4 Search Guiding

A local search heuristic stops when it reaches a local optimum and no operator can find further improvements. To diversify the search, a guiding system helps the local search operators move away from the incumbent by adjusting the problem. Simple systems include random restarts and mutations, where a new starting point is generated for the local search to continue. More complex systems keep track of frequent solution characteristics, which are then penalized or even forbidden to ensure new solutions are reached.

### 2.5 Complexity

The total optimization effort is decided by *the cost of a single iteration* and *the number of iterations* required to reach an optimum. The first depends on the cost of a single move evaluation, which is a small, operator-specific constant, multiplied by the number of possibilities for each operator in the set, easily determined by how many nodes are required to define a single move of that type, e.g. there are $O(n^2)$ distinct ways of swapping two nodes. The number of iterations is influenced by the quality of each move. Additionally, as each operator only affects a small part of the solution, we can intuitively see that the move count will increase with instance size.

The double impact of instance size causes bad scaling in Local Search-based metaheuristics. Reducing the impact of size on these effort factors improves the efficiency. Sadly, few general efficiency strategies exist in the VRP literature. The next paragraphs hold short descriptions of several important complexity reduction strategies.

**Candidate Lists.** Attributed to (Glover, 1990), the idea of Candidate Lists (CL) is to restrict the neighborhood by preemptively eliminating unlikely edges based on relevant characteristics, such as edge cost. The size of the List varies widely between authors. A narrow list may exclude the optimal solution.

**Variable Neighborhood Descent.** Variable Neighborhood Descent (Mladenovic and Hansen, 1997) reduces the search effort by iteratively performing LS optimizations with a single operator instead of the entire set. The operator changeover strategy can affect the performance in terms of quality and efficiency.

**Don‘t-Look Bits.** The nature of Local Search makes it so that each iteration only affects a small part of the solution. Moves that take place in unaffected regions therefore remain unchanged. DLBs (Bentley, 1990) exploit this information by tagging unimproved neighbor solutions until they are affected by an iteration. This way, unchanged parts can be ignored in the next iteration and redundant evaluations are skipped.

# 3 STATIC MOVE DESCRIPTORS

The idea of Static Move Descriptors (Zachariadis and Kiranoudis, 2010) is to build a memory structure that enables the algorithm to keep track of changes in the solution. Each move is stored in an efficient data structure, ordered by its solution-dependent cost tag. When a move is affected, its cost tag is updated. This improves the scaling of operators by an order of $n$.

The implementation of the SMD technique provides efficient access to every move so they can be easily found and updated, as well as make it easy to search for the best move available. The underlying algorithm is simple, iterating over three phases:

**Initialization Phase.** Given a starting solution, all possible SMDs, one for each move, are generated and their cost tags are calculated. The SMDs are then inserted into a Priority Queue (PQ), which keeps them organized by their current cost tag.

**Search Phase.** The best SMD is extracted from the PQ and its feasibility is checked. If infeasible, the SMD is added to a Stack and the next SMD is extracted from the PQ. If feasible, the search ends and all SMDs in the Stack are reinserted into the PQ. The Update phase starts.

**Update Phase.** The feasible SMD is executed, changing the solution. Using the SMD definition and the update rules, all affected SMDs are identified and deleted from the PQ. Their cost tags are updated with regards to the new solution and they are reinserted. The Search phase begins on the updated PQ.
The cost of the initialization phase can be amortized over all moves. By updating the minimal amount of cost tags, the scaling is improved. The downside is the overhead of keeping the PQ organized. In the original paper, the PQ is implemented as a Fibonacci Heap (Fredman and Tarjan, 1987), which is a theoretically optimal implementation for a PQ.

## Drawbacks

The SMD technique is very promising in theory, but does have several drawbacks. These will be the focus of our own work, as we try to reduce their impact.

**Memory Usage.** Every SMD is stored during the entire runtime of the algorithm. The memory requirements increase with the number of operators and the complexity of each operator (e.g.: 2Opt $\sim O(n^2)$ vs. 3Opt $\sim O(n^3)$). Additionally, the implementation of the PQ adds significant extra costs.

**Cross-operator Effects.** Any change in the solution will affect SMDs of every operator. The update effort grows linearly with the size of the operator set. Additionally, the amount of updating rules, which determine which SMDs need to be updated, grows quadratically with the number of operators, making it harder to implement efficiently.

**Heap Overhead.** The Heap greatly speeds up the Searching phase, but its benefits are reduced by its own cost. The SMD-based algorithm requires many Heap operations and whilte a Fibonacci Heap is theoretically optima, its practical performance is poor. The effort of an operation increases with Heap size, which grows strongly with instance size.

In the next chapter, we will suggest some ways to improve the performance of SMD-based algorithms.

# 4 EFFICIENT SMD IMPLEMENTATION

Three major changes are made. First, a Variable Neighborhood search strategy (2.5) is adopted. Secondly, the Heap implementation is changed to one with better practical performance. Lastly, the Search phase is replaced with a novel heuristic heap search.

## 4.1 Variable Neighborhood Descent

By iteratively optimizing with a single operator, no cross-operator updating is needed, drastically reducing the evaluation effort. Secondly, the heap size is reduced, making all operations cheaper. However, this change requires an Initialization phase (3) at each changeover and reduces average move quality.

## 4.2 Binary Heap

The Binary Heap (Johnson, 1975) is the conceptual opposite of the Fibonacci Heap: theoretically inferior, but excellent practical performance. Preliminary tests showed reduced memory usage and much faster Update phases, but slower Search phases.

## 4.3 Heuristic Heap Search

Unlike the Fibonacci Heap, which relies on pointers to organize its elements, a Binary Heap can be implemented as an array with its Heap properties enforced implicitly. This allows us to access elements without making use of expensive Heap operations.

The array implementation maps the Tree-like structure to linear memory by means of index-relations. A node located at index $k$ will find its parent at $k/2$ and its left and right children at $2 \times k$ and $2 \times k + 1$ respectively. Combined with the Heap property ($par(k) \geq k \geq children(k)$), this means good nodes are more likely to be located early in the array.

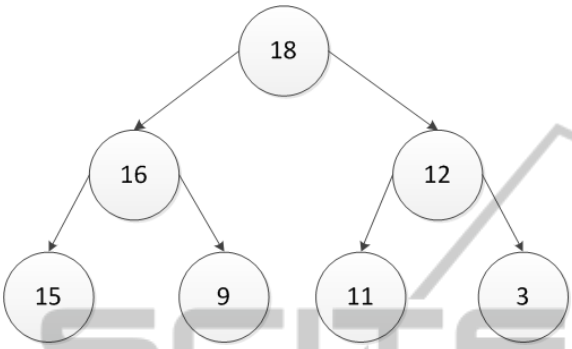


Figure 1: Tree-like structure of a Binary Heap.

Table 1: Array representation of a Binary Heap.

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Value | 18 | 16 | 12 | 15 | 9 | 11 | 3 |

We can exploit this property by replacing the expensive iterated root-extracts by simple linear traversal of the array, until a feasible SMD is found. This is not guaranteed to be the best feasible move, but it is very likely to be one of the best moves available.

This change adopts a first-of-operator acceptance strategy with a *strongly guided* search strategy.

Combining these three changes leads to significant speedups, with as only downside the forced change from best-of-all to a guided first-of-operator strategy.

# 5 RESULTS

We compare our version to the original. If we can show reduced *time per iteration* without strong reductions in *average move quality*, we can declare our version more efficient, provided our version does not consistently lead to worse final solutions.

In an effort to reduce external bias, we will compare each version relative to a naïve implementation.

## 5.1 Algorithm Overview

We implemented and tested three algorithms:

1. The BASE algorithm: a naïve implementation of a best-accept variable neighborhood search.
2. The FIB algorithm: the original SMD implementation; uses a Fibonacci Heap and best-accept.
3. The BIN algorithm: our suggested algorithm, using the heuristical heap search.

All algorithms use a VND (see 2.5) scheme to allow for a better comparison. Preliminary results showed that even for the original algorithm, VND always reduced runtimes without sacrificing final solution quality. Additionally, we believe it is a necessary change for the SMD technique to be viable, as it provides more flexibility for the operator set.

The operator set exists of the same three quadratic operators as in (Zachariadis and Kiranoudis, 2010): Swap, Relocate and 2Opt. Swap takes two nodes, $i$ and $j$, and swaps their positions. Relocate takes two nodes, $i$ and $j$, removes $i$ from its current position and reinserts it after $j$. The 2Opt has two variants: if $i$ and $j$ are part of the same route, the route segment between $i$ and $j$ is reversed; if $i$ and $j$ belong to different routes, both routes are cut at $i$ or $j$ and the start of one route is combined with the end of the other. Each version uses the same cheapest-insertion construction heuristic. We follow the VND scheme that consistently lead to better results for each algorithm: 2Opt, then Swap, then Relocate, repeat.

Tests were performed on 36 instances provided by (Zachariadis and Kiranoudis, 2010), but due to space constraints we will limit the report to 7 sets: 3 small sets to analyze the scaling and 4 large sets for the actual performance.

- Small sets: g01, g04, 24 (240, 480, 720).
- Large sets: zk1 - zk4 (3000).

## 5.2 Time per Move

First we analyze the time per iteration. The BASE algorithm performs a full evaluation and executes the best. The SMD-based algorithms perform an Update and Search phase, except in the first iteration following an operator changeover, in which case they perform an Initialization and Search phase. Frequent changeovers increase the average time per iteration because of the required Initialization step.

Table 2 shows the average time per iteration for the small sets per operator. Note that 2Opt is significantly more expensive overall, because of its dual (inter- and intraroute) nature, which results in additional work for each evaluation.

Comparing between sets, we see the BASE algorithm scales poorly: tripling the instance size leads to 13 to 20 times slower iteration times. The FIB algorithm performs poorly on small sets due to internal

overhead, but scales well with instance size: increasing instance size by 3 leads to an average slowdown factor of 5. The BIN algorithm scales slightly better still and runs significantly faster across the board, up to 50 times faster than the FIB algorithm.

Table 2: Average time (ms) per iteration on small sets.

| | Set | g01 | g04 | 24 |
|---|---|---|---|---|
| | Size | 240 | 480 | 720 |
| BASE | Swap | 1.16 | 4.14 | 17.28 |
| | Reloc | 0.88 | 3.76 | 11.51 |
| | 2Opt | 3.72 | 12.59 | 73.72 |
| FIB | Swap | 1.38 | 3.30 | 6.41 |
| | Reloc | 0.91 | 2.87 | 5.33 |
| | 2Opt | 4.60 | 10.89 | 22.13 |
| BIN | Swap | 0.07 | 0.11 | 0.19 |
| | Reloc | 0.06 | 0.10 | 0.17 |
| | 2Opt | 0.11 | 0.22 | 0.57 |

Table 3: Average time (ms) per iteration on large sets.

| | Set | zk1 | zk2 | zk3 | zk4 |
|---|---|---|---|---|---|
| | Size | 3000 | 3000 | 3000 | 3000 |
| BASE | Swap | 181.82 | 180.17 | 167.16 | 168.18 |
| | Reloc | 118.15 | 116.75 | 109.56 | 114.91 |
| | 2Opt | 430.06 | 432.00 | 441.79 | 430.42 |
| FIB | Swap | 66.01 | 66.31 | 66.94 | 67.38 |
| | Reloc | 64.16 | 64.50 | 67.74 | 67.16 |
| | 2Opt | 132.58 | 146.44 | 124.93 | 136.62 |
| BIN | Swap | 2.20 | 1.91 | 2.08 | 2.11 |
| | Reloc | 1.63 | 1.44 | 1.44 | 1.47 |
| | 2Opt | 2.83 | 2.69 | 2.62 | 2.61 |

In table 3 we see the same comparison for the large sets, which are roughly 12 times larger than set $g01$. We see relative average slowdown factors of 130, 50 and 27 for the BASE, FIB and BIN algorithms respectively, further proving the scaling benefits of the SMD technique. In total, the FIB algorithm works roughly 3 times faster than BASE, while the our BIN algorithm works roughly 110 times faster on the large sets. We conclude that the BIN implementation can find and execute improvements much faster.

### 5.3 Move Quality

To verify our assumption that our novel heuristic heap search leads to high quality moves, we measured the number of iterations each algorithm needed to reach a solution. To account for differences in optimization paths, we used the quality of the worst final solution of the three algorithms as a cutoff point for our data. We also verified that neither technique consistently lead to better final solutions. On the large sets, each algorithm got within 12% to 5% of the best known solutions, which is normal for a basic heuristic.

Table 4: Iterations required to reach similar solution quality.

| | g01 | g04 | 24 | zk1 | zk2 | zk3 | zk4 |
|---|---|---|---|---|---|---|---|
| BASE | 276 | 494 | 679 | 11334 | 11329 | 9968 | 9794 |
| FIB | 285 | 476 | 656 | 10311 | 10475 | 9284 | 9884 |
| BIN | 357 | 651 | 1069 | 13242 | 10985 | 9867 | 10610 |

Table 4 shows that even the BASE and FIB algorithms, which use the same strategy, still lead to slightly different results. This can be explained by difference in tie-breaking as well as precision errors. While FIB may seem to outperform BASE consistently, tests on other sets have disputed this. We have taken the average of the two algorithms as comparison point for BIN. The difference with our BIN algorithm is more profound, requiring up to 30% more iterations to reach a similar solution, although we see the relative difference becomes smaller for larger instances.

Our results show that our new implementation benefits greatly from the intelligent searching of the SMD technique, leading to improved scaling factors, while suffering less from the overhead that haunted the original implementation proposed by (Zachariadis and Kiranoudis, 2010). We feel the trade-off, being at most 30% more iterations versus a speedup factor of 50, is greatly in favor of our implementation.

## 6 CONCLUSIONS

In this paper, we have taken a closer look at the fascinating concept of Static Move Descriptors (Zachariadis and Kiranoudis, 2010), a strategy to speed up Local Search-based algorithms by eliminating unnecessary reevaluations. We have revealed several flaws and suggested changes to its implementation.

Firstly, we have changed the search strategy to a Variable Neighborhood. This speeds up the search and allows costless expansion of the operator set.

Secondly, we change the underlying data structure used to keep the SMDs organized: the theoretically optimal Fibonacci Heap is replaced by the practically efficient Binary Heap. This speeds up the Update phase tremendously, even for large instance sizes.

Lastly, we change the exact search phase to a novel heuristic heap search, avoid expensive heap operations. This makes the Search phase very efficient.

Our results show that the changed strategy does not affect final solution quality. Moreover, we have shown that the average move quality remains high, while each iteration is sped up significantly, leading to impressive speedups overall.

## 7 FUTURE RESEARCH

The concept of Static Move Descriptors is very promising and with our changes it is a practically viable and flexible speedup strategy. However, its full potential has not yet been explored.

Firstly, our implementation has not yet been tested inside a metaheuristic. The flow of the algorithm lends itself perfectly for a Guided Local Search or a Tabu Search strategy. Either strategy could be implemented without much difficulty and without hurting performance.

Secondly, the flexibility of the SMD concept allows for combination with other speedup techniques, such as Candidate Lists. This could significantly speed up the various parts of the algorithm.

## REFERENCES


Bentley, J. L. (1990). Experiments on traveling salesman heuristics. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 91–99. Society for Industrial and Applied Mathematics.

Clarke, G. and Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

Cordeau, J. and Maischberger, M. (2011). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*.

Croes, G. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812.

Fredman, M. and Tarjan, R. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615.

Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290.

Glover, F. (1990). Tabu search-part II. *ORSA Journal on computing*, 2(I):4–32.

Johnson, D. (1975). Priority queues with update and finding minimum spanning trees. *Information Processing Letters*, 4(3).

Kilby, P., Prosser, P., and Shaw, P. (1997). Guided local search for the vehicle routing problem. *Proceedings of the 2nd International Conference on Meta-heuristics*.

Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34(9):2743–2757.

Laporte, G. (2009). Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416.

Lee, C., Lee, Z., Lin, S., and Ying, K. (2010). An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. *Applied Intelligence*, 32(1):88–95.

Lenstra, J. and Kan, A. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*.

Mladenovic, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100.

Nguyen, H. and Yoshihara, I. (2007). Implementation of an effective hybrid GA for large-scale traveling salesman problems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(1):92–9.

Rego, C. (2001). Node-ejection chains for the vehicle routing problem: Sequential and parallel algorithms. *Parallel Computing*, 27(3):201–222.

Thompson, P. M. and Psaraftis, H. N. (1993). Cyclic Transfer Algorithm for Multivehicle Routing and Scheduling Problems. *Operations Research*, 41(5):935–946.

Voudouris, C., Tsang, E., and Alsheddy, A. (2010). Guided local search. *Handbook of metaheuristics*, pages 185–218.

Xu, J. and Kelly, J. (1996). A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30(4):379–393.

Zachariadis, E. and Kiranoudis, C. (2010). A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & Operations Research*, 37(12):2089–2105.