Implementation Report: Variable Neighbourhood Search

Authors: E.K. Burke, P. Cowling and R. Keuthen
Contact:
Ralf Keuthen
University of Nottingham
School of Computer Science & IT
Jubilee Campus
Nottingham NG8 1BB, UK
email: rxk@cs.nott.ac.uk

The local improvement heuristic HyperOpt introduced in [?] and briefly described in the HyperOpt implementation report was integrated with the Variable Neighbourhood Search meta-heuristic approach recently proposed by Hansen and Mladenović [?][?][?]. The hybridisation of HyperOpt and Variable Neighbourhood Search has already been investigated in [?] where it was applied to a sequencing problem arising in electronic manufacture.

Variable Neighbourhood Search:

The basic idea of Variable Neighbourhood Search is to systematically change the neighbourhood of the current solution which is searched in order to explore an increasingly large region of the solution space. A neighbouring solution is only accepted if it presents an improvement to the best solution found so far, otherwise it is abandoned. In their papers Hansen and Mladenović [?][?][?] suggest two basic strategies which they refer to as Variable Neighbourhood Search and Variable Neighbourhood Descent. Our implementation is based upon the Variable Neighbourhood Search strategy, which was applied in [?][?][?] to a variety of combinatorial optimization problems including the TSP using 2-opt for local search and rabdom k-opt moves, with $k > 2$, to change neighbourhood and escape a 2-opt local optimum.

Briefly, the concept of basic Variable Neighbourhood search can be described as follows. In the initialization stage a set of different neighbourhood structures is selected, an initial solution determined and a suitable stopping criteria chosen. The second stage represents the actual neighbourhood search. Starting with the first neighbourhood structure a random neighbour of the initial solution is determined, Hansen and Mladenović refer to this process as *shaking*, and a local search heuristic is applied to this solution. If the local optimum found at this stage represents an improvement to the initial solution, the move is accepted and the same process is repeated starting with this new solution. Otherwise, the local optimum is abandoned, a different usually larger neighbourhood is selected for shaking and the process starts over again.

Computational results are included for two approaches using either 2- or 3-HyperOpt for local search. Results are reported for both approaches where the number of shakes is limited to 100 and 1000. More details on the implementation can be found below or in the HyperOpt implementation report.

Implementation Details

Initialization and Local Search: As for local search, the Variable Neighbourhood Search approach we implemented uses either 2- or 3-HyperOpt for local search [?], which have been implemented in the very same way as described in the HyperOpt implementation report. The initial tour is constructed using the *Multiple Fragment* algorithm which has been implemented as described by Bentley [?].

Except for the first local search, which is directly applied to the initial Multiple Fragment tour, HyperOpt uses a stack to ensure that it only searches for improvements on hyperedges which have been affected by the shaking process described below.

Shaking: The vital part of variable neighbourhood search is the *shaking* process which, for the TSP, is similar to the *kicks* procedure performed by chained Lin-Kernighan [?][?]. In contrast to the random shakes suggested by Hansen and Mladenović in [?][?] we implemented a different approach guiding the shakes based on information gained in a stage of preprocessing and during the search process. These guided shakes work as follows.

After the first local search has been performed a list of "bad" hyperedges is determined which will be considered when "shaking". Here, a hyperedge $H = (p_0, p_1, \ldots, p_h)$ of consecutive vertices in the initial tour, is considered to be of "bad" quality if

$$\sum_{i=0}^{h-1} distance(p_i, p_{i+1}) - distance(p_i, nearest(p_i)),$$

(where $nearest(p_i)$ denotes the nearest neighbour of p_i) is large. The length of this "bad hyperedge" list depends on the number of shakes to be performed and has been fixed at $\#shakes/3$.

The chosen neighbourhood structures for shaking are an extended form of randomized HyperOpt moves. The basic idea of this shake is to remove three hyperedges from the tour and then restore the tour in a random manner. The outline of this shaking process is illustrated in figure 1. One of the 3 hyperedges involved in the shaking operation is chosen at random from the list of "bad hyperedges", determined as described above. Then we choose two hyperedges overlapping the nearest neighbours of the starting city of this "bad hyperedge", avoiding overlapping hyperedges for the random shake. In order to prevent HyperOpt

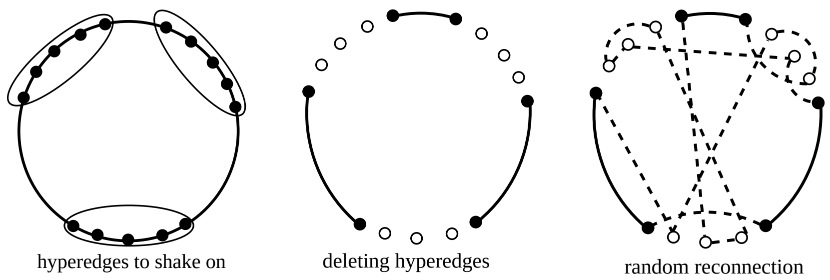

Figure 1: Schema of a hyper-shake

from falling back to a local optimum already visited, the hyperedges used for shaking start at a length of $k = (h + 2)$, where h denotes the neighbourhood size used by HyperOpt for local search. If a shake and local search do not yield an improvement in tour quality over the best tour found so far then the length of the shaking hyperedges k is increased by one, the tour is abandoned and the process starts over again at the best tour found so far. However, in order not to destroy too much of the tour found already we imposed an upper bound of 15 on k.

Technical details:

The heuristic described has been coded in GNU C++ using the `-O2` compiler option running on a Linux 2.2.13 kernel. The calculations were carried out on a Pentium II 400 MHz PC with 128 Mb memory. Computational results for the test battery of up to 3 million cities are included in the file `TestSet`. The code was written by Peter Cowling and Ralf Keuthen.

Computational Results:

Computational results for various Variable Neighbourhood Search approaches, using HyperOpt for local search, are included in the the following files:

VNS, using 2-HyperOpt, 100 Iterations: All Euclidean instances of up to 316,228 cities.
File: `Results_BurkeCowlingKeuthen_VNS_HyperOpt2_100It`.

VNS, using 2-HyperOpt, 1000 Iterations: All Euclidean instances of up to 316,228 cities.
File: `Results_BurkeCowlingKeuthen_VNS_HyperOpt2_1000It`.

VNS, using 3-HyperOpt, 100 Iterations: All Euclidean instances of up to 316,228 cities.
File: `Results_BurkeCowlingKeuthen_VNS_HyperOpt3_100It`.

VNS, using 3-HyperOpt, 1000 Iterations: All Euclidean instances of up to 316,228 cities.
File: `Results_BurkeCowlingKeuthen_VNS_HyperOpt3_1000It`.