A Multilevel Lin-Kernighan-Helsgaun Algorithm for the Travelling Salesman Problem

Chris Walshaw

Computing and Mathematical Sciences, University of Greenwich,
Old Royal Naval College, Greenwich, London, SE10 9LS, UK.
Email: C.Walshaw@gre.ac.uk; URL: www.gre.ac.uk/∼c.walshaw

Mathematics Research Report : 01/IM/80

September 27, 2001

Abstract

The multilevel paradigm has recently been applied to the travelling salesman problem with considerable success. The resulting algorithm progressively coarsens the problem, initialises a tour and then employs a local search algorithm to refine the solution on each of the coarsened problems in reverse order. In the original version the chained Lin-Kernighan (CLK) scheme was used for the refinement. However, a new and highly effective Lin-Kernighan variant (LKH) has recently been developed by Helsgaun. Here then we report on the modifications required to develop a multilevel LKH algorithm and the results achieved. Although the LKH algorithm, with its extremely high quality results, is more difficult to improve on than the CLK, nonetheless the multilevel framework was able to enhance the LKH performance. For example, in experiments on a well established test suite, the multilevel LKH scheme found 39 out of 59 optimal solutions as compared to the 33 found by LKH in a similar time period.

Keywords: Multilevel Refinement; Travelling Salesman; Combinatorial Optimisation.

1 Introduction

In this paper we address the Travelling Salesman Problem (TSP) which can be simply stated as follows: given a collection of 'cities', find the shortest tour which visits all of them and returns to the starting point. Typically the cities are given coordinates in the 2D plane and then the tour length is measured by the sum of Euclidean distances between each pair on the tour. However, in the more general form, the problem description simply requires a metric which specifies the distance between every pair of cities.

The TSP has been shown to be NP-hard, [2], but has a number of features which make it stand out amongst combinatorial optimisation problems. Firstly, for problems which have not yet been solved to optimality (typically with 10,000 or more cities), an extremely good lower bound can be found for the optimal tour length. This bound, known as the Held-Karp Lower Bound (HKLB), was developed in 1970 by Held & Karp, [3], and usually comes extremely close to known optimal tour lengths (often within 1%). Thus to measure the quality of an algorithm for a given set of problem instances (some or all of which have unknown solutions), we can simply calculate the average percentage excess of tours produced by the algorithm over the HKLB for each instance.

Another unusual feature of the TSP, perhaps due to the fact that the problem is so intuitive and easy to state, is that it has almost certainly been more widely studied than any other combinatorial optimisation problem. For example Johnson & McGeoch, [6], survey a wide range of approaches which run the gamut from local search, through simulated annealing, tabu search & genetic algorithms to neural nets. Remarkably, and despite all this interest, the local search algorithm proposed by Lin & Kernighan in 1973, [9], still remains at the heart of the most successful approaches. In fact Johnson & McGeoch describe the Lin-Kernighan (LK) algorithm as the world champion heuristic for the TSP from 1973 to 1989. Further, this was only conclusively superseded by chained or iterated versions of LK originally proposed by Martin, Otto & Felten, [10], in 1991.

Even as recently as 1997, and despite all the work on exotic and complex optimisation techniques, Johnson & McGeoch, [6], concluded that an iterated or chained Lin-Kernighan (ILK/CLK) scheme provides the highest quality tours for reasonable costs and that CLK/ILK variants are 'the most cost effective way to improve on Lin-Kernighan, at least until one reaches stratospheric running times'. However in 2000 an interesting and highly effective LK variant was introduced by Helsgaun, [4]. This Lin-Kernighan-Helsgaun (LKH) algorithm, at least in its multi-trial version, can significantly improve on CLK/ILK results although it suffers from runtimes which are quadratic, in fact $O(N^{2.2})$, for problems of size N.

In a further recent development, a general solution strategy known as multilevel refinement has been applied with considerable success to the TSP and in particular the CLK algorithm, [7, 13]. The multilevel approach involves recursive coarsening to create a hierarchy of approximations to the original problem; an initial solution is found for the coarsest problem and then iteratively refined at each level, coarsest to finest, [14]. When applied to the TSP it was able to significantly enhance the performance of the CLK algorithm and in particular seemed to work much better for the more clustered problem instances with which TSP algorithms traditionally have great difficulties.

In this paper we aim to bridge the gap between these two developments and report on a multilevel TSP implementation which uses the LKH algorithm for the refinement stage at each level. The paper is organised as follows. In Section 2 we describe the general multilevel framework for the TSP and outline the LKH algorithm. In Section 3 we test the algorithm on a large suite of problem instances and discuss its behaviour and finally in Section 4 we summarise the paper.

It is sometimes convenient to use graph notation for the TSP in which case we refer to the cities as vertices and the problem can be specified as a complete graph with weighted edges, i.e. there is an edge between every pair of cities and the weight of the edge specifies the distance between them.

2 A multilevel LKH algorithm for the travelling salesman problem

The strength of the multilevel approach rests in the construction of a family of increasingly coarse approximations to the original problem. Each child problem should be smaller and easier to solve than its parent, but not so different that a solution for the child is no help in solving the parent problem. Moreover, if the coarsening is constructed so as to sample the solution space, the resulting family of problems are simply restrictions of the original space rather than near approximations to it and this very much facilitates the solution process, [14].

In [13] a sampling-based coarsening strategy for the TSP is derived which works by successively fixing edges into the tour. For example, given a TSP instance P of size N, if we fix an edge between cities c_a and c_b then we create a smaller problem P' of size $N-1$ (because there are $N-1$ edges to be found) where we insist that the final tour of P' must somewhere contain the fixed edge (c_a, c_b). Having found a tour T' for P' we can then return to P and look for better tours using T' as the initial tour. In fact we can fix many distinct edges in one coarsening step and hence reduce the size of the problem considerably at every level.

Figure 1 shows an example of this. The top row demonstrates the coarsening process where dotted lines represent matchings of vertices (and hence new fixed edges) whilst solid lines represent fixed edges that

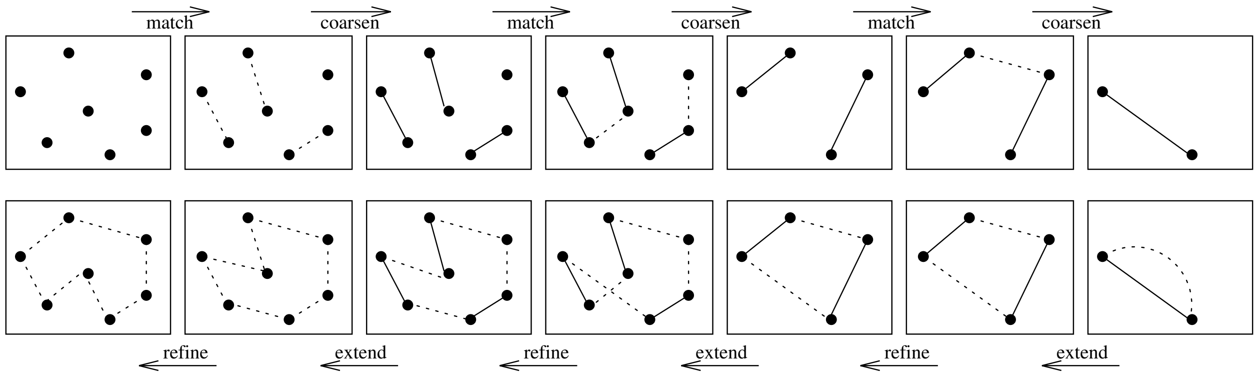


Figure 1: An example of a multilevel TSP algorithm at work

have been created in previous coarsening steps. Notice in particular that after the second coarsening step chains of fixed edges are reduced down to a single edge with a vertex at either end and any vertices internal to such a chain are removed. The coarsening terminates when the problem is reduced to one fixed edge & two vertices and at this point the tour is initialised. The initialisation is trivial and merely consists of completing the cycle by adding an edge between the two remaining vertices. The procedure then commences the extend/refine loop (bottom row, right to left). Again solid lines represent fixed edges whilst dotted lines represent free edges which may be changed by the refinement. The extension itself is trivial; we simply expand all fixed edges created in the corresponding coarsening step and add the free edges to give an initial tour for the refinement process. The refinement algorithm then attempts to improve on the tour (without changing any of the fixed edges) although notice that for the first refinement step no improvement is possible. The final tour is shown at the bottom left of the Figure; note in particular that fixing any edge during coarsening does not force it to be in the final tour since, for the final refinement step, all edges are free to be changed. However, fixing an edge early on in the coarsening does give it less possibilities for being flipped.

To describe the implementation of such an algorithm, first we discuss all the elements required for the multilevel framework (detailed more fully in [13]) and then in §2.2 outline the modifications required to use the LKH algorithm for the refinement.

2.1 The multilevel framework

Matching and coarsening. The coarsening process works by matching pairs of vertices and then fixing edges between them. In fact however, it is more convenient for the data structure in the code to use edge objects and so in practice a matching of edges is created at each level. Initially each edge is of zero length and has the same vertex at either end; however after the first coarsening most edges will have different vertices at either end. The matching algorithm creates a randomly ordered list of edges and visits them one by one, matching each edge with its most appropriate neighbouring edge until every edge is either matched or has no unmatched neighbours. The definition of neighbouring in this context is given below. Figure 2(a) shows an example of this process where the edges (v_1, w_1) & (v_2, w_2) are matched together.

The aim during the matching process should be to fix those edges that are most likely to appear in a high quality tour thus allowing the refinement to concentrate on the others. For example, consider Figure 2(b); it is difficult to imagine an optimal tour which does not include the edge (u, v) and so ideally the matching should fix it early on in the process. Indeed, if by some good fortune, the matching ***only*** selected optimal edges then the optimal tour would have been found by the end of the coarsening and the refinement would have no possible improvements. However, in the absence of any other information about the optimal tour, vertices are matched with their nearest unmatched neighbours. The implementation of this process uses a superimposed grid of spacing h, e.g. Figure 2(b), to avoid $O(N)$ searches for finding neighbours, [14].

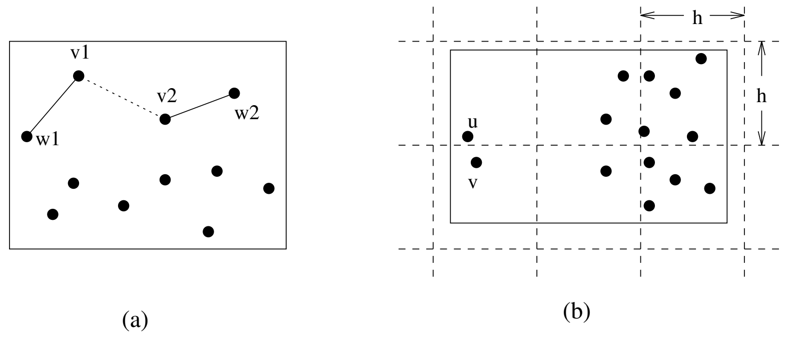


Figure 2: TSP matching examples

An important implementation detail is that at each level vertices are only allowed to match with other vertices within a distance h (thus giving a definition to the concept of *neighbouring* edges above) However, at each level h is increased (by keeping the average number of vertices per grid cell constant) to prevent the coarsening from terminating prematurely. This process not only aids fast searches for nearby vertices but also prevents long-range matching on the lower levels of the coarsening (which appears to have an important effect on the results).

Initialisation. The coarsening ceases when there remain only two vertices with a fixed edge between them. This is guaranteed to occur because each coarsening level will match at least one pair of vertices and so the problem size will shrink. Initialisation is then trivial and consists of adding an edge between the two vertices to complete the tour (the other edge of the tour being the fixed one).

2.2 Refinement: the Lin-Kernighan-Helsgaun algorithm

Typically TSP tour refinement takes place by 'flipping' edges. For example, if the tour contains the edges (v_1, w_1) & (w_2, v_2) in that order, then these two edges can always be flipped to create (v_1, w_2) & (w_1, v_2). This sort of step forms the basis of the 2-opt algorithm due to Croes, [1], which is a steepest descent approach, repeatedly flipping pairs of edges if they improve the tour quality until it reaches a local minimum of the objective function and no more such flips exist. In a similar vein, the 3-opt algorithm of Lin, [8], exchanges 3 edges at a time. The Lin-Kernighan (LK) algorithm, [9], also referred to as variable-opt, however incorporates a limited amount of hill-climbing by searching for a sequence of exchanges, some of which may individually increase the tour length, but which combine to form a shorter tour. A vast amount has been written about the LK algorithm, including much on its efficient implementation, and for an excellent overview of techniques see the surveys of Johnson & McGeoch, [6, 7].

Recently a new and highly effective variant of the LK algorithm has been developed by Helsgaun, [4]. This scheme employs a number of important innovations including sequential 5-opt moves and the use of sensitivity analysis to direct the search. It has been shown to compute solutions extremely close to the optimal (where known) but suffers from the drawback of runtimes which are quadratic in N, the problem size, and so may not be suitable for large problem instances (e.g. if $N = 10{,}000$ the runtime for N trials runs to days whilst for $N = 30{,}000$ it can be measured in weeks). The algorithm is most effective in its multi-trial version where the optimisation is run multiple times, each with a different initial tour, the construction of which is biased by the edges in the existing champion tour. This is slightly different from the kicks used in the CLK algorithm where each initial tour is constructed by perturbing the existing champion tour and appears to be a very effective modification.

Three variants of the multilevel LKH algorithm were produced:

- The first version treated the LKH software entirely as a black box. Initial (incomplete) testing proved unpromising and closer inspection of the user manual, [5], revealed that if the code is given an initial tour, T, it is assumed to be of high quality and the code is discouraged from flipping the edges of T (a software feature rather than an algorithmic bug). However in the multilevel version, not only is the code ***always*** given an initial tour, T, but in fact a large proportion of the edges of T will have been fixed in the level above and so the code should be positively encouraged to try flipping them.

- The second version fixed this problem by treating all edges in the initial tour the same way as any other edge. Testing of the multilevel code with the single-trial version of LKH then gave good results but at great expense. This appeared to arise from the preprocessing (referred to as the *ascent*) where a vector of *penalty* values (π-values), one for each vertex, is calculated. This preprocessing is a costly part of the scheme (although it only requires to be carried out once for a multi-trial version) and in particular appears to be crippling to the multilevel version if carried out at each level of the problem.
- The third and current version used for the results below makes the assumption that the same π-values can be used at each problem level (by simply selecting the values corresponding to the vertices of the original problem which do appear). The π-values are thus computed once only prior to the multilevel coarsening.

3 Experimental results

We have evaluated the multilevel LKH algorithm using a test suite of TSP problems drawn from the 90 instances compiled for the 8th DIMACS implementation challenge[1] (aimed at characterising approaches to the TSP). However, because of the quadratic behaviour of the LKH algorithm, all problem instances with more than 20,000 vertices were omitted from the suite. Additionally, the 8 instances which are specified solely by an distance matrix were also omitted because the matching scheme requires coordinate information (although this is not an inherent attribute of the multilevel algorithm and it should be possible to develop suitable matching algorithms). The 67 instances used are then in three groups:

(I) 31 symmetric and geometric instances of 1,000 or more vertices from `TSPLIB`[2], a collection of sample TSP instances, including some from real-life applications, compiled by Reinelt, [11, 12].

(II) 18 randomly generated instances with uniformly distributed vertices. These range in size from 1,000 to 10,000 vertices, going up in size gradations of $\sqrt{10}$ and were constructed by Johnson, Bentley & McGeoch specifically to study asymptotic behaviour in tour finding heuristics.

(III) 18 randomly generated instances with randomly clustered vertices. These range in size from 1,000 to 10,000 and have the same origin and purpose as (II) although clustered examples such as these are generally considered to be more difficult to solve.

The version of the LKH software used is contained in the file `LKH-1.2.tgz` and we very gratefully acknowledge its author for making this code available[3]. Only minor modifications were required to the LKH code and in particular, since it allows fixed edges, the previous work-around used to prevent them from being flipped, [13, §3.3.1] was not required. The multilevel code wrapper which implements the matching and coarsening is called `sierra` and is described more fully in [13]; parameter settings established in that paper are also used here. The tests were carried out on a DEC Alpha machine with a 466 MHz CPU and 1 Gbyte of memory.

We have tested the multilevel LKH algorithm using the same methodology as in [13]. Thus to assess the convergence behaviour we vary a single parameter, the optimisation intensity, λ, and measure average solution quality (expressed as percentage excess over the HK lower bound – see Section 1) against average runtime (normalised by runtimes for the LK algorithm which may be found in [13]). The intensity parameter, λ, in this case is simply the number of trials expressed as a fraction of N the problem size, i.e. $\lambda = xN$

[1]see `http://www.research.att.com/~dsj/chtsp/`
[2]available from `http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/`
[3]from `http://www.dat.ruc.dk/~keld/research/LKH`

for some factor x. For the multilevel versions, the intensity parameter at each level, λ_l, was then set to $\lambda_l = xN_l$ where N_l is the problem size (the number of free edges) at level l. We then refer to the single-level and multilevel schemes at intensity λ as LKH$^\lambda$ and MLLKH$^\lambda$ respectively.

To give a snapshot of the results we present detailed figures for the MLLKH^N and LKH^N configurations in Table 3. For the TSPLIB instances (category I) we report the results for each example, but for the randomly generated instances we average the results for each class so that asymptotic analysis is easier. For example the row labelled 'C3k (5)' contains average values over the 5 instances with three thousand vertices, `C3k.0`, ..., `C3k.4`. For each algorithm and each instance we then present the percentage excess over the HKLB and, in the next column, the percentage excess over the optimum tour length (if known). We also give the ratio of average runtime for the instance over the average runtime for the LK algorithm for the same instance. Finally, at the bottom of the Table we average all of the results; the HKLB and runtime results are averaged over all 67 instances whilst the optimal excess figures are averaged over those 59 instances for which an optimal tour is known.

Table 1: A summary of results comparing the quality achieved for similar runtimes

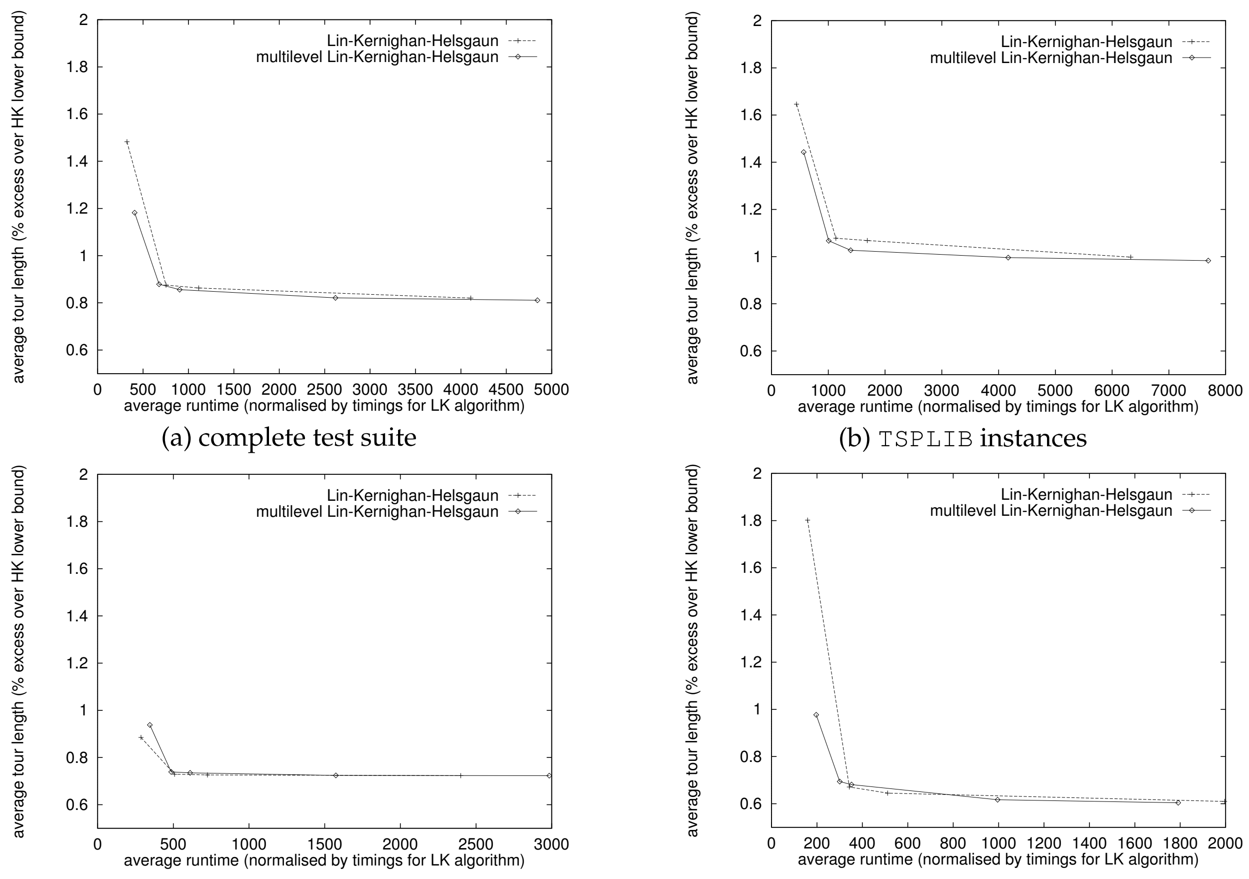
	Average % excess				Average % excess		
configuration	HKLB	opt	T/T_{LK}	configuration	HKLB	opt	T/T_{LK}
LKH^1	1.483	0.627	322.856	MLLKH^1	1.182	0.356	406.970
LKH$^{N/10}$	0.875	0.066	752.981	MLLKH$^{N/20}$	0.879	0.078	676.996
LKH$^{N/5}$	0.863	0.059	1112.439	MLLKH$^{N/10}$	0.856	0.057	903.223
LKH^N	0.820	0.016	4108.781	MLLKH$^{N/2}$	0.821	0.017	2619.229
				MLLKH^N	0.811	0.010	4842.779

Table 3 shows individual results at one particular intensity, but as we are interested in the convergence behaviour over a range of intensities Table 1 shows averaged figures (including the final line of Table 3) at intensities $\lambda = 1, N/10, N/5, N$ for LKH$^\lambda$ and $\lambda = 1, N/20, N/10, N/2, N$ for MLLKH$^\lambda$. These values were chosen because, as argued in [13], a single-level local search scheme (LS$^\lambda$) at intensity λ takes approximately the same runtime as a multilevel version (MLLS$^{\lambda/2}$) at intensity $\lambda/2$ provided the runtime of the local search is approximately linear in N. However, it is also predicted in [13] that if the local search is quadratic or even cubic then the relative multilevel overhead is much smaller and this is borne out in Table 1. For example, MLLKH$^{N/2}$ is considerably faster than LKH^N and in turn MLLKH^N is only slightly slower than that (by about 20%).

Table 2: A summary of results on subsets of the test suite

	`TSPLIB`			random uniform instances			random clustered instances		
	Average % excess			Average % excess			Average % excess		
configuration	HKLB	opt	T/T_{LK}	HKLB	opt	T/T_{LK}	HKLB	opt	T/T_{LK}
LKH^1	1.646	0.706	439.300	0.885	0.168	286.326	1.802	0.930	158.843
LKH$^{N/10}$	1.078	0.113	1133.279	0.729	0.011	508.199	0.671	0.031	342.806
LKH$^{N/5}$	1.068	0.103	1686.300	0.726	0.008	725.495	0.645	0.024	511.066
LKH^N	0.998	0.029	6328.409	0.723	0.005	2399.259	0.610	0.002	1995.609
MLLKH^1	1.443	0.491	565.317	0.938	0.221	344.774	0.977	0.229	196.458
MLLKH$^{N/20}$	1.067	0.101	1006.603	0.739	0.022	486.267	0.694	0.088	300.068
MLLKH$^{N/10}$	1.027	0.060	1393.060	0.735	0.017	610.608	0.681	0.090	352.229
MLLKH$^{N/2}$	0.996	0.027	4168.757	0.724	0.007	1574.117	0.617	0.007	995.710
MLLKH^N	0.983	0.014	7694.107	0.723	0.004	2983.738	0.604	0.010	1791.200

As in [13] it is again of interest to examine the convergence behaviour on the three different subclasses of problem instance and Table 2 therefore splits the results in Table 1 into their 3 subclasses. Figure 3 then illustrates the results in Tables 1 & 2 graphically by plotting the HKLB excess against runtime. As can be seen, with the exception of the random uniform instances, Figure 3(c), if only a single trial is used, the multilevel variant, MLLKH^1 appears to offer considerably better performance than LKH^1. However



(a) complete test suite (b) `TSPLIB` instances (c) random uniform instances (d) random clustered instances

Figure 3: Plots of convergence behaviour for single-level and multilevel versions of LKH

comparisons with the MLCLK and CLK results in [13] suggest that neither LKH^1 nor MLLKH^1 offers a cost effective solution strategy considering the amount of computation time required.

On the other hand the multi-trial versions of LKH offer extremely high quality solutions and results elsewhere (e.g. [4, 7]) indicate that solution quality can be very close to optimal (for those instances where the optimal solutions are known). This allows the multilevel version very little margin for improvement (because it is obviously impossible to improve on an optimal solution) and as can be seen the MLLKH and LKH curves follow each other very closely although MLLKH does appear to achieve slightly better results. Also interestingly, and in common with the results in [13], the multilevel versions seem to give the least benefit to the random uniform instances. It seems likely that this is because the matching choices are not clear-cut during the coarsening of such instances, [14].

Finally returning to Table 3, if we consider those 59 instances for which an optimal solution is known, perhaps the most impressive statistic is that in a similar runtime MLLKH^N found 39 of the optimal solutions as compared to 33 for LKH^N. (Note that for these instances the very highly regarded CLK algorithm with N kicks was unable to find any optimal solutions, although admittedly its runtime is linear and it is thus ***very*** much faster on the larger instances.)

4 Summary

We have discussed the modifications required to use the multilevel TSP scheme originally developed in [13] in combination with the powerful Lin-Kernighan-Helsgaun algorithm from [4]. The results appears to indicate that the multilevel approach can enhance the convergence behaviour of the LKH algorithm, at least for two out of the three subclasses of problem instances tested (the same two subclasses as for the chained Lin-Kernighan algorithm) although the single-level LKH algorithm is so good that any improvement found by the multilevel version will only be very small. However the resulting scheme MLLKH^N managed to find an optimal tour for 2/3 of the tested instances with a known solution. In addition, the results here do appear to demonstrate further the flexibility and generality of the multilevel approach. Furthermore they add further evidence to the suggestion in [14] that the multilevel paradigm is better suited to problem instances with inherent sparsity and back up the claim, [13, 14], that the multilevel runtime overhead for quadratic local search schemes (in this case around 20%) is considerably less than the factor of two for linear schemes.

Acknowledgements. The author very gratefully acknowledges Keld Helsgaun for making his software available and Gerd Reinelt, David Johnson, Lyle McGeoch & others for providing the test cases.

References

[1] G. A. Croes. A method for solving traveling salesman problems. *Oper. Res.*, 6:791–812, 1958.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[3] M. Held and R. M. Karp. The Traveling Salesman Problem and Minimum Spanning Trees. *Oper. Res.*, 18:1138–1162, 1970.

[4] K. Helsgaun. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *Eur. J. Oper. Res.*, 126:106–130, 2000.

[5] K. Helsgaun. LKH User Guide. (available via `http://www.dat.ruc.dk/~keld`), 2001.

[6] D. S. Johnson and L. A. McGeoch. The travelling salesman problem: a case study. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, 1997.

[7] D. S. Johnson and L. A. McGeoch. Experimental Analysis of Heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Travelling Salesman Problem and its Variations*. Kluwer Academic Publishers, 2001. (To appear).

[8] S. Lin. Computer solutions of the traveling salesman problem. *Bell Syst. Tech. J.*, 44:2245–2269, 1965.

[9] S. Lin and B. W. Kernighan. An effective heuristic for the traveling salesman problem. *Oper. Res.*, 21(2):498–516, 1973.

[10] O. C. Martin, S. W. Otto, and E. W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5(3):299–326, 1991.

[11] G. Reinelt. TSPLIB— A Traveling Salesman Problem Library. *ORSA J. Comput.*, 3(4):376–384, 1991.

[12] G. Reinelt. TSPLIB95. Tech. Rep., Inst. Angewandte Math., Univ. Heidelberg, 1995.

[13] C. Walshaw. A Multilevel Approach to the Travelling Salesman Problem. To appear in *Oper. Res.*, (originally published as Univ. Greenwich Tech. Rep. 00/IM/63), 2000.

[14] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. Tech. Rep. 01/IM/73, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, June 2001.

Table 3: LKH^N & MLLKH^N results

	LKH^N			MLLKH^N		
	Average % excess			Average % excess		
instance	HKLB	opt	T/T_{LK}	HKLB	opt	T/T_{LK}
dsj1000	0.610	0.003	702.179	0.612	0.004	467.857
pr1002	0.888	0.000	145.444	0.888	0.000	200.667
u1060	0.660	0.012	1281.150	0.648	0.000	1039.000
vm1084	1.327	0.000	673.000	1.327	0.000	585.529
pcb1173	0.960	0.000	375.526	0.960	0.000	509.474
d1291	1.349	0.167	850.105	1.180	0.000	1483.474
rl1304	1.547	0.000	285.143	1.547	0.000	383.333
rl1323	1.649	0.000	387.700	1.649	0.000	699.000
nrw1379	0.429	0.000	398.087	0.429	0.000	603.478
fl1400	1.926	0.184	1459.098	1.931	0.189	1204.275
u1432	0.285	0.000	894.478	0.285	0.000	1038.957
fl1577	1.723	0.063	3744.333	1.686	0.027	3156.567
d1655	0.940	0.000	678.280	0.940	0.000	860.120
vm1748	1.354	0.000	621.933	1.354	0.000	867.167
u1817	0.905	0.000	1184.792	0.947	0.042	1408.042
rl1889	1.551	0.001	671.194	1.550	0.000	685.452
d2103	1.488	0.046	3700.269	1.451	0.010	4985.769
u2152	0.684	0.065	1679.667	0.618	0.000	2293.074
u2319	0.018	0.000	2853.338	0.018	0.000	3110.954
pr2392	1.216	0.000	1162.211	1.216	0.000	1361.684
pcb3038	0.810	0.000	3512.640	0.810	0.000	2570.740
fl3795	1.179	0.142	10955.608	1.081	0.045	6265.785
fnl4461	0.552	0.003	2635.115	0.552	0.003	3657.333
rl5915	1.595	0.035	3254.656	1.585	0.025	3584.989
rl5934	1.479	0.097	5816.422	1.381	0.000	5573.244
pla7397	0.581	0.000	18954.725	0.581	0.000	22729.732
rl11849	1.024	0.006	13629.236	1.061	0.042	18371.661
usa13509	0.668	0.007	14796.812	0.669	0.007	21160.387
brd14051	0.491		28222.618	0.495		34580.087
d15112	0.533	0.009	24606.889	0.534	0.009	37055.553
d18512	0.504		46048.025	0.496		56023.919
C1k (10)	0.541	0.003	544.527	0.552	0.014	450.817
C3k (5)	0.615	0.001	1765.447	0.614	0.000	1885.290
C10k (3)	0.830		7216.154	0.761		6102.328
E1k (10)	0.741	0.005	275.122	0.740	0.004	401.178
E3k (5)	0.710	0.006	1902.020	0.709	0.004	2217.182
E10k (3)	0.684		10308.446	0.687		12869.865
Average	0.820	0.016	4108.781	0.811	0.010	4842.779