Kick Strength and Online Sampling for Iterated Local Search

Thomas Liddle
School of Mathematics, Statistics and Operations Research
Victoria University of Wellington
New Zealand
thomas.liddle@msor.vuw.ac.nz

Abstract

A key decision when using Iterated Local Search (ILS) as an optimisation metaheuristic is the determination of the kick's nature and strength. This paper investigates methods of improving ILS through discovering the nature of the neighbourhoods in the search using two methods: adjusting the kick strength *before* the search and sampling the neighbourhood of local optima at each iteration to discover the best search direction to explore. In addition a new method of dynamically adjusting the kick strength *during* the search is developed to avoid needing to test multiple kick strengths separately. These methods are tested on the Travelling Salesman Problem using two-opt and dynasearch local search, and it can be concluded that: increasing the kick strength can significantly improve the performance of ILS but the best strength is problem instance dependent; dynamically sampling the kick strength during the search can provide better performance than a single kick and is an effective way of getting better results without explicitly testing multiple kick strengths; online sampling can improve the performance of ILS, but the best method of sampling is problem instance dependent; and dynasearch significantly outperforms two-opt local search for small kick strengths.

Key words: Metaheuristics, iterated local search, travelling salesman problem, dynasearch, online sampling.

1 Introduction

Iterated Local Search (ILS) (Lourenço, Martin, and Stützle 2003) is a metaheuristic search technique which aims to find good solutions to difficult combinatorial optimisation problems in a reasonable amount of computational time. It is an iterative procedure which performs a biased sampling of local optima (hilltops) in the search space by randomly changing the current solution at each iteration (performing a "kick") and then using a local search heuristic to find a new local optima. Thus ILS prevents getting stuck in local optima (the main disadvantage of heuristics). ILS can operate with “black-box” components; it does not need to know the local search heuristic or how the kick is performed.

The current solution at each iteration in ILS has a *neighbourhood*, which is the set of solutions that all possible locally optimised kicks can reach (the set of all neighbouring hilltops). Standard implementations of ILS perform a single kick blindly without searching for the “best” kick to perform (to reach the best neighbouring hilltop). In addition, the *strength* of the kick (how much change is made to the current solution) determines the solutions that are contained within the neighbourhood (hilltops closer or further away).

In this paper we set out to discover whether adjusting the kick strength in ILS can improve the performance; whether sampling the neighbourhood at the current solution to find a better search direction can improve the performance; whether a method of dynamically adjusting the kick strength *during* the search can improve or maintain the performance without needing to test multiple kick strengths; and whether using a more powerful local search method affects the outcome of neighbourhood sampling and dynamic kick strength choice.

The remainder of this paper is structured as follows: Section 2 provides some background information, Section 3 proposes two new variations of ILS, Section 4 outlines the experimental setup, Section 5 gives results and discussion, and Section 6 offers some conclusions and suggestions of further work.

2 Background

We use the Travelling Salesman Problem (TSP) to test the proposed variations of ILS, as it is an NP-hard combinatorial optimisation problem which is used as a benchmark for testing new optimisation techniques (Lawler et al. 1985). The TSP is the problem of finding the shortest-distance tour around n cities, and can be informally described by imagining a salesman who starts out at his home city and needs to visit a set of other cities exactly once before returning home, wanting to find the best visiting order to minimize his distance travelled.

ILS is a metaheuristic that has been applied to many different optimisation problems, including the TSP and many scheduling problems. The general procedure can be seen in Algorithm 1 (Lourenço, Martin, and Stützle 2003).

Algorithm 1 Iterated Local Search

```
  $s_0$ = GenerateInitialSolution
  $s^*$ = LocalSearch($s_0$)
  repeat
    $s'$ = PerformKick($s^*, history$)
    $s'^*$ = LocalSearch($s'$)
    $s^*$ = AcceptanceCriterion($s^*, s'^*, history$)
  until TerminationCriterion
```

ILS begins with a local optima from using the local search heuristic on an initial solution (generated by a construction heuristic such as nearest neighbour construction). It then iteratively performs a kick on the current solution (by randomly changing some elements of the solution), obtains a new local optima (using local search) and then determines if this new solution should be used as the current so-

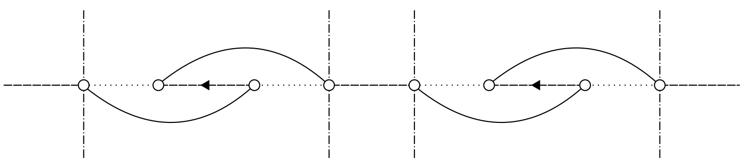


Figure 1: Dynasearch with two-exchanges.

lution for the next iteration by testing it against some *acceptance criterion.* The search is terminated when a *termination criterion* is reached.

Two local search methods are considered in this paper, two-opt local search and dynasearch (with two-exchanges). *Two-opt* search is a simple local search method which iteratively makes the best two-exchange until no improving exchange can be found (Lin 1965). A two-exchange selects and removes two non-adjacent links, joining the tour back up in the only other way possible. A single iteration of two-opt search evaluates all possible two-exchanges and performs the exchange that provides the largest improvement in length over the current tour. *Dynasearch* (Congram, Potts, and van de Velde 2002) is a local search method which uses dynamic programming to find the best set (largest total improvement in distance) of non-overlapping two-exchanges on the current tour (see Figure 1), performing all of them in a single iteration. It performs exactly the same number of exchange evaluations as two-opt local search per iteration, but will find at least as much (but likely more) improvement than two-opt can.

We adopt the “double-bridge” move (a particular four-exchange) as the kick component, as it has been found to be very effective for the symmetric TSP and has been commonly used since (Martin, Otto, and Felten 1991). The *kick strength* is the number of sequential random double-bridge moves, k, performed as a single kick, e.g. if $k = 5$, the kick component performs five random double-bridge moves.

Two *acceptance criteria* are considered: *better* acceptance, where the new solution is only accepted if its tour length is less than that of the current solution; and *probability* acceptance, where in addition to better solutions, if the new solution is worse than the current, it may be accepted with a probability p if there has been a delay of d iterations without finding an improving solution. Different balances of d and p produce a search that is closer to either better acceptance or a *random restart* search. The search terminates if either an optimal solution has been found (if the tour length is known beforehand) or a maximum time limit has been reached.

3 Proposed Variations of ILS

We propose two variations of ILS: sampling the neighbourhood of the current solution at each iteration and dynamically adjusting the kick strength at each iteration.

3.1 Sampled Iterated Local Search

We propose a variation of ILS, called Sampled Iterated Local Search (S-ILS), that samples the neighbourhood of the current solution, by performing four kicks in three different branching combinations (sampling types), each of which is locally optimised before performing the next. The best local optima found (intermediate or final) is presented as the new solution to the acceptance criterion. The three different sampling types can be seen in Figure 2 and are briefly described below.

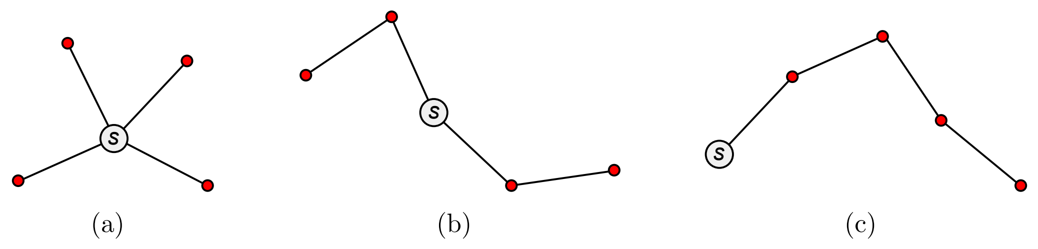


Figure 2: Different sampling types for S-ILS: Four branches of a single kick (a), two branches of two kicks each (b), and one branch of four kicks (c).

- *Type 1: Four branches of one kick.* A single kick of strength k is performed away from the current solution four independent times. Local search is performed on each to produce four local optima.
- *Type 2: Two branches of two kicks each.* Two independent sampling branches are followed, each of which consists of two consecutive kicks of strength k (each locally optimised before performing the next). This type looks "ahead" further than Type 1, but looks "around" less. The best of the four local optima (two intermediate and two final) is proposed as the new solution, giving the same amount of improvement opportunity as Type 1.
- *Type 3: One branch of four kicks.* A single branch of four consecutive kicks of strength k (each locally optimised before performing the next) is followed. This type consists entirely of looking ahead, and does not look around the current solution. It may be able to discover a search direction that looks promising in the short term but is in the long term.

3.2 Dynamic Iterated Local Search

We propose another variation of ILS, called Dynamic Iterated Local Search (D-ILS), which avoids the need to test multiple kick strengths to find the best one; instead it dynamically samples from a set of available kick strengths K, choosing more often those kick strengths in K that seem to be performing well so far. For ILS, $|K|$ experiments (one for each kick strength) are needed to find the best $k \in K$, but D-ILS has the potential to achieve the same or similar results using only a single experiment. It can also be incorporated into S-ILS as it only affects how the kick is performed.

We use *weights* w_k for each $k \in K$ in a probabilistic selection method. These w_k are updated during the search. This concept is similar to the use of *pheromones* in Ant Colony Optimization (Dorigo and Stützle 2004). K is specified by lower and upper kick strength limits k_l and k_u, thus $K = \{k_l, k_l + 1, \ldots, k_u - 1, k_u\}$, with $m = k_u - k_l + 1$ possible kick strengths to sample from.

A simple implementation of dynamic sampling is to use a single weights vector $\mathbf{w}$, where the next kick strength is chosen according to $\mathbf{w}$ only, that is, no memory is incorporated. Alternatively, we incorporate one level of memory into the kick strength choice, in order to evolve a balance between intensification and diversification. In this case a weights *matrix* W is kept, where each row j of W ($j = 1, \ldots, m$) corresponds to the weights vector $\mathbf{w}_j$: the weights for choosing the next kick strength given that the previous kick strength was j.

Using a small kick strength has the potential for getting stuck in a local optima (Lourenço, Martin, and Stützle 2003). We introduce a *non-improvement bias* (NIB), which biases the strength of the kick increasingly towards larger values as the number of non-improving iterations increases. The NIB (set before the search) is used as a power to exponentially increase the bias for higher values of NIB in this paper.

The proposed method for probabilistically selecting a kick strength j from the set of possible kick strengths K, when performing a dynamically sampled kick in D-ILS is as follows. First determine the row vector of weights $\mathbf{w}$. If not using memory, this is just $\mathbf{w}$, the vector of weights. If using memory assign $\mathbf{w} = \mathbf{w}_i$, row i of the weights matrix W, which corresponds to the previous kick strength $i \in \{1, \dots, m\}$. Then calculate the probability vector $\mathbf{p}$ from $\mathbf{w}$ as follows. Normalise the weights to be non-negative:

$$v_j = \begin{cases} w_j - w_{min} & \text{if } w_{min} < 0 \\ w_j & \text{otherwise} \end{cases}, \quad \text{for } j = 1, \dots, m \tag{1}$$

where $w_{min} = \min_j w_j$. Calculate the normalisation increment a as:

$$a = \frac{w_{max} - w_{min}}{k_u - k_l} \tag{2}$$

where $w_{max} = \max_j w_j$. Calculate the non-improvement biases b_j:

$$b_j = \begin{cases} 0 & \text{if NIB} = 0 \\ (\beta j)^{\text{NIB}} & \text{otherwise} \end{cases}, \quad \text{for } j = 1, \dots, m \tag{3}$$

where β is the number of non-improving iterations that have elapsed. Add the normalisation increment and non-improvement biases to $\mathbf{v}$:

$$v_j = w_j + a + b_j, \quad \text{for } j = 1, \dots, m \tag{4}$$

Finally, calculate p_j for each $j \in K$ by normalising the v_j between 0 and 1:

$$p_j = \frac{v_j}{\sum_{i=1}^{m} v_i}, \quad \text{for } j = 1, \dots, m \tag{5}$$

The probability vector $\mathbf{p}$ can then be used to sample a kick strength to use from K. After a kick strength j is selected, the kick is performed, a local optima is found using local search, and the weight corresponding to the chosen kick strength is then updated via

$$w_j \leftarrow w_j + length_{last} - length_{new} \tag{6}$$

where $length_{last}$ and $length_{new}$ are the tour length of the current (before the kick) and new solutions' local optima respectively. If selection with memory is used, then replace w_j with w_{ij} in Equation (6) where i is the kick strength chosen the previous iteration.

Studies of Ant Colony Optimization (ACO) have shown that initialising the pheromones (which serve a similar purpose to the weights in D-ILS) to an arbitrary value can adversely affect the algorithm's performance (Dorigo and Stützle 2004). Performance increases when the pheromones are initialised to a value that is indicative of the average change in pheromone level. In the context of D-ILS, we propose to estimate this value (as it cannot be found without running a full search) by performing a kick and local search of each available kick strength on the generated initial solution of ILS independently of each other, and then setting the each weight to be the average of the improvement found by each kick strength. The general procedure for D-ILS can be seen in Algorithm 2.

Algorithm 2 Dynamic Iterated Local Search

```
$s_0$ = GenerateInitialSolution
$s^*$ = LocalSearch($s_0$)
$\mathbf{w}$ = InitialiseWeights($s^*$)
**repeat**
  $s'$ = PerformDynamicKick($s^*, \mathbf{w}, history$)
  $s'^*$ = LocalSearch($s'$)
  $\mathbf{w}$ = UpdateWeights($s^*, s', \mathbf{w}$)
  $s^*$ = AcceptanceCriterion($s^*, s'^*, history$)
**until** TerminationCriterion
```

Table 1: Problem instance properties and maximum time limits for all experiments.

	berlin52	eil101	tsp225	a280	pcb442
Number of cities	52	101	225	280	442
Optimal tour length	7542	629	3919	2579	50778
Search time limit (s)	10	20	120	120	300

4 Experimental Setup

This section describes the experimental setup for testing the proposed methods, including the problem instances tested and the configurations of the parameters in each method used in the experiments.

Five instances of the symmetric TSP with Euclidean distances varying in size have been chosen from (Heidelburg University 2010) for testing the proposed variations (see Table 1).

Initial experiments of ILS with two-opt local search using better acceptance and different combinations of $p \in \{0.05, 0.25, 0.5, 1\}$ and $d \in \{1, 5, 10, 20\}$ for probability acceptance showed that better acceptance generally performed better than probability acceptance; of the 16 probability acceptance configurations, the best was the one with smallest p and largest d ($p = 0.05$ and $d = 20$); and ILS performed much better than random restart search (increasingly so as the problem instance size increased). Hence, each problem instance was tested using better acceptance and probability acceptance with $p = 0.05$ and $d = 20$, and each of two-opt and dynasearch local search. ILS and each S-ILS sampling type were tested on each of $k \in \{1, 2, \dots, 10\}$ for the berlin52 and eil101 instances, and each of $k \in \{1, 2, \dots, 20\}$ for the other instances. D-ILS used these same ranges for the available kick strength set K, and each of NIB $\in \{0, 1, 2\}$ with and without memory, a total of 24 D-ILS configurations. Each experiment was run on the same 50 different random seeds. Table 1 shows the maximum time limits allowed for all experiments for each instance.

5 Results and Discussion

Table 2 shows results graphs for ILS and each type of S-ILS on three of the problem instances (eil101, tsp225 and pcb442), using better acceptance and each of two-opt and dynasearch local search heuristics. The distribution of results is shown by using the median and lower and upper quartile values of the percentage extra distance travelled over the optimal solution. The graphs show a banana-like shape in the distribution for two-opt, indicating that adjusting the kick strength can significantly reduce the extra distance travelled over the optimal solution. There is a much steeper decrease in performance for larger kick strengths for dynasearch, indicating that optimising the kick strength may just be a matter of trying a few small kick strengths. For small kick strengths, dynasearch performs much better than two-opt (achieving the optimal solution for the tsp225 instance), but significantly worse for large kick strengths.

Across all kick strengths, S-ILS Type 1 (S-ILS(1)) performed similarly to ILS, and S-ILS(3) performed generally worst (with a narrower range of good kick strengths and a steeper decrease in performance for increasing kick strengths). The best kick strength varies between problem instances, sampling types and local search methods, but the performance of the best kick strength for each of ILS and the three S-ILS types is similar for each instance (though slightly worse for S-ILS(3) in each case). The results for D-ILS (not shown) generally sat somewhere within the "average" distribution of each sampling type, giving better results than standard ILS but not as good as ILS with the best kick strength.

Table 3 shows that these results generally hold for probability acceptance as well, with S-ILS(1) performing the best in most problem instances and S-ILS(3) the worst. This table shows the best two configurations of D-ILS for each instance and acceptance criterion, with a prevalence of Type 1 sampling being among the best configurations. It is inconclusive whether memory is useful or not, or which value of NIB to use (though other results suggest that NIB $= 2$ performs slightly worse than the others). At least one D-ILS configuration achieved the overall best performance out of all ILS and S-ILS configurations without needing to test multiple kick strengths for the berlin52, eil101 and a280 problem instances.

Table 4 shows that in four out of five problem instances, all 24 D-ILS configurations improved on ILS (the other instance gave 22 out of 24 improving configurations), which is a promising result. The amount of improvement achieved varied among configurations, however, but the best configurations gave significant reductions in extra distance travelled (up to 75% for the larger instances). The 100% reduction for the berlin52 problem instance corresponds to the optimal solution being found (at least 50% of the time) by S-ILS(2), S-ILS(3) and all D-ILS configurations, where standard ILS's median result was 0.23% from the optimal solution. Which configurations gave the best results were not consistent, but the worst performers were those with Type 3 sampling or NIB $= 2$, and removing these in future would reduce the number of D-ILS configurations to just 12. In addition, D-ILS was rather sensitive to the kick strength set, indicating that further research is required to improve the selection method. Because dynasearch's best kick strength was very low and showed significant performance degradation, the results of D-ILS when using dynasearch indicate that is unlikely to be necessary if using dynasearch as the local search heuristic. Table 4 also shows that the three S-ILS types were able to match the performance of ILS, and in some cases improve on it, though only by a small amount. The sampling types are of most use within D-ILS.

Probability acceptance generally produced worse results than better acceptance for ILS across all instances, but interestingly produced similar results for all S-ILS types, indicating that the neighbourhood sampling may be able to "weed out" the worse solutions that ILS is forced to propose.

Table 2: Distribution of the percentage extra distance travelled over the optimal solution across 50 seeds (y axis) for the eil101, tsp225 ad pcb442 problem instances using better acceptance, testing different configurations of S-ILS against ILS.

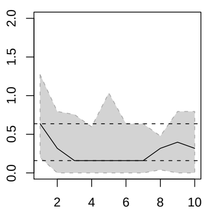
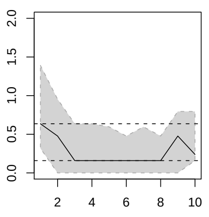
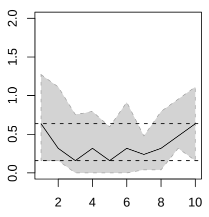
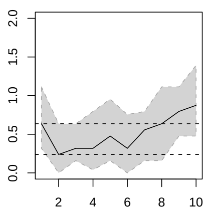
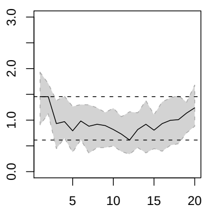
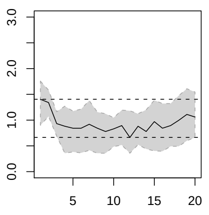
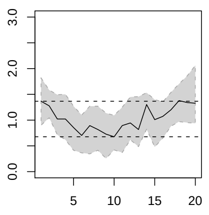
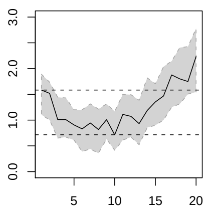
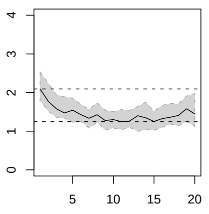
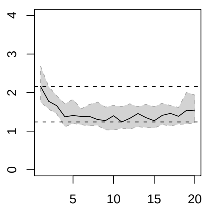
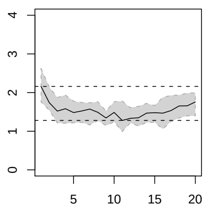
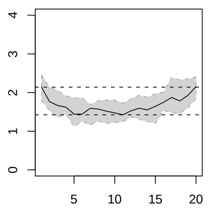
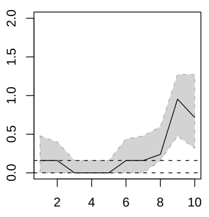
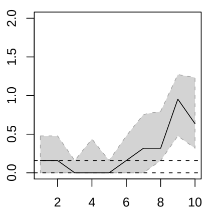
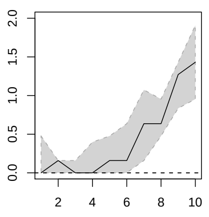
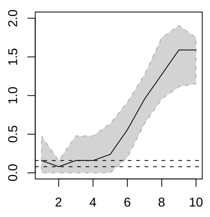
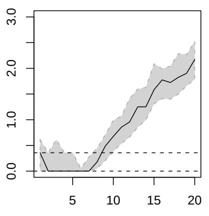
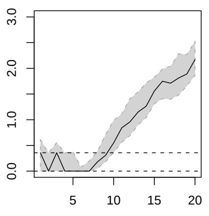
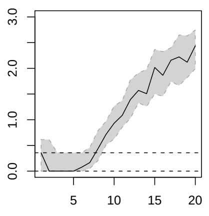
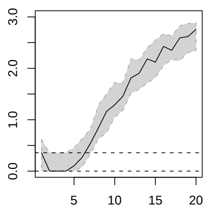
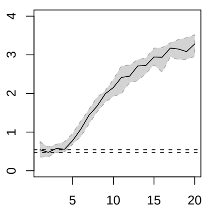
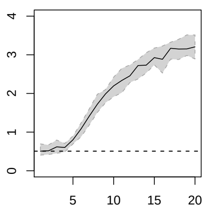
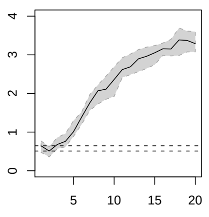
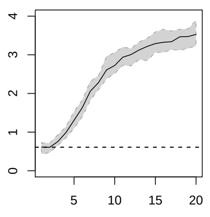
Instance	ILS	S-ILS(1)	S-ILS(2)	S-ILS(3)
		two-opt		
eil101				
tsp225				
pcb442				
		dynasearch		
eil101				
tsp225				
pcb442				

Table 3: Summary of experimental results from testing ILS, S-ILS, and D-ILS using two-opt local search, showing the best and worst performing method from ILS and the three S-ILS types, as well as the best two D-ILS configurations and whether they achieved the overall best median result.

Problem instance	Acceptance criterion	ILS/S-ILS Best	ILS/S-ILS Worst	Best D-ILS	Sampling	Memory?	NIB	Overall Best?
berlin52	Better	S-ILS (3)	S-ILS (1)	1.	None	Yes	2	Yes
				2.	None	No	2	Yes
	Prob	ILS	S-ILS (1)	1.	None	Yes	2	Yes
		S-ILS (3)		2.	None	No	2	Yes
eil101	Better	S-ILS (1)	S-ILS (3)	1.	Type 2	Yes	0	Yes
				2.	None	Yes	2	Yes
	Prob	S-ILS (1)	S-ILS (3)	1.	Type 2	Yes	0	Yes
		S-ILS (2)		2.	Type 2	No	1	Yes
tsp225	Better	ILS	S-ILS (3)	1.	Type 1	No	2	No
		S-ILS (1)		2.	Type 1	No	0	No
	Prob	S-ILS (2)	ILS	1.	Type 1	Yes	0	No
		S-ILS (1)		2.	Type 1	No	2	No
a280	Better	ILS	S-ILS (3)	1.	None	No	2	Yes
		S-ILS (1)		2.	Type 2	No	1	No
	Prob	S-ILS (1)	ILS	1.	Type 1	Yes	2	No
		S-ILS (3)		2.	Type 2	No	1	No
pcb442	Better	S-ILS (1)	S-ILS (3)	1.	Type 1	Yes	1	No
		ILS		2.	None	Yes	0	No
	Prob	S-ILS (1)	ILS	1.	Type 2	No	0	No
			S-ILS (3)	2.	Type 1	Yes	1	No

Table 4: The number of configurations (of S-ILS or D-ILS) that performed strictly better than (% distance from the optimal solution), and better than or equal to each of standard ILS ($k = 1$) and ILS with the discovered best kick strength ($k = k_{best}$) shown as ($<$ ($\leq$), % reduction) triples. Comparisons are made against the median percentage distance from the optimal solution. Results are shown for two-opt local search and better acceptance.

		S-ILS (/3)		D-ILS (/24)
Instance	Configuration	$k = 1$	$k = k_{best}$	
berlin52	ILS, $k = 1$	2 (2) 100%		24 (24) 100%
	ILS, $k = k_{best}$		0 (3)	0 (24)
eil101	ILS, $k = 1$	0 (3)		22 (23) 75%
	ILS, $k = k_{best}$		0 (2)	0 (8)
tsp225	ILS, $k = 1$	2 (2) 6%		24 (24) 45%
	ILS, $k = k_{best}$		0 (0)	0 (0)
a280	ILS, $k = 1$	1 (3) 5%		24 (24) 73%
	ILS, $k = k_{best}$		0 (1)	0 (1)
pcb442	ILS, $k = 1$	0 (0)		24 (24) 58%
	ILS, $k = k_{best}$		1 (1) 1%	0 (0)

6 Conclusions

Optimizing the kick strength in ILS can significantly improve performance, although it requires multiple experiments of different kick strengths. Sampling the neighbourhood at each iteration can also improve on (or at least match) the performance of ILS, but the best method of sampling is problem instance dependent. The method of dynamic kick strength choice we developed appears quite promising and can improve on ILS when using two-opt search, but is sensitive to the range of available kick strength it takes and thus does not perform as well as using dynasearch. Dynasearch performs better than two-opt search for small kick strengths but (counter-intuitively) significantly worse for large kick strengths for ILS and S-ILS.

There is much further research to be done, including testing these methods on a wider set of problem instances and more local search methods; investigating other methods for dynamic kick strength choice to reduce its sensitivity to the available kick strengths; investigating why dynasearch exhibits extreme fluctuations in performance; development of other neighbourhood sampling types and methods; and testing of these methods on other problem types to see if the observed results hold.

References

Congram, R. K., C. N. Potts, and S. L. van de Velde. 2002. "An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem." *INFORMS Journal on Computing* 14 (1): 52–67.

Dorigo, M., and T. Stützle. 2004. *Ant Colony Optimization.* MIT Press.

Heidelburg University. 2010. TSPLIB. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`, accessed October, 2010.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. 1985. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley.

Lin, S. 1965. "Computer solutions of the traveling salesman problem." *Bell System Computer Journal* 44:2245–2269.

Lourenço, H. R., O. Martin, and T. Stützle. 2003. "Iterated local search." In *Handbook of Metaheuristics*, edited by F. Glover and G. Kichenberger, 321–353. Kluwer.

Martin, O., S. W. Otto, and E. W. Felten. 1991. "Large-step markov chains for the traveling salesman problem." *Complex Systems* 5:299–326.