Title:

Improved Large-Step Markov Chain Variants for the Symmetric TSP

Authors:
Inki Hong, Andrew B. Kahng and Byung-Ro Moon
UCLA Computer Science Department
Los Angeles, CA 90095-1596 USA

Communicating Author:
Andrew B. Kahng
UCLA Computer Science Department
3713 Boelter Hall
Los Angeles, CA 90095-1596 USA
E-mail: abk@cs.ucla.edu
Tel. 310-206-7073 (office), 310-825-7578 (fax)

Abstract: The *large-step Markov chain* (LSMC) approach is the most effective known heuristic for large symmetric TSP instances; cf. recent results of [Martin, Otto and Felten, 1991] and [Johnson, 1990]. In this paper, we examine relationships among (i) the underlying local optimization engine within the LSMC approach, (ii) the "kick move" perturbation that is applied between successive local search descents, and (iii) the resulting LSMC solution quality. We find that the traditional "double-bridge" kick move is not necessarily optimum: stronger local optimization engines (e.g., Lin-Kernighan) are best matched with stronger kick moves. We also propose use of an adaptive temperature schedule to allow escape from deep basins of attraction; the resulting *hierarchical* LSMC variant outperforms traditional LSMC implementations that use uniformly zero temperatures. Finally, a population-based LSMC variant is studied, wherein multiple solution paths can interact to achieve improved solution quality.

Key Words: large-step Markov chain; optimization; simulated annealing; traveling salesman problem

1 Preliminaries

Given a set of cities and a symmetric matrix of all inter-city distances, the symmetric traveling salesman problem (TSP) seeks a shortest tour which visits each city exactly once. The symmetric TSP is NP-hard [Garey and Johnson, 1979], and has been extensively studied both in terms of its combinatorial structure and as a testbed for exact and heuristic methods [Lawler et al., 1985] [Johnson and McGeoch, 1997]. Studies such as [Johnson, 1990] point to greedy local search (e.g., using the fast 3-Opt [Bentley, 1992] or the Lin-Kernighan (LK) [Lin and Kernighan, 1973] neighborhood structure) as the most effective approach for practical instances. Over the past decade, *iterated descent* [Baum, 1986a] [Baum, 1986b] has been shown to be an effective means of applying a given greedy local search "engine": iteratively perform a greedy descent, then perturb the resulting local minimum to obtain the starting solution for the next greedy descent. Currently, the "large-step Markov chain" (LSMC) heuristic of [Martin, Otto and Felten, 1991] and the "iterated LK" heuristic of [Johnson, 1990] are believed to be the best-performing iterated descent variants (and, indeed, the best-performing of all heuristics for obtaining near-optimal solutions [Johnson and McGeoch, 1997]); we generically refer to both of these as LSMC methods.

1.1 Previous Large-Step Markov Chain Implementations

The LSMC approach is described in Figure 1, following the presentation in [Martin, Otto and Felten, 1991]. LSMC alternately applies (i) a (greedy) local optimization procedure *Descent*, followed by (ii) a "kick move" which perturbs the current local minimum solution in order to obtain a starting solution for the next *Descent* application. Local search (i.e., *Descent*) procedures used in previous implementations include LK as well as k-Opt methods.[1] [Martin, Otto and Felten, 1991] used both LK and a fast implementation of 3-Opt; [Johnson, 1990] used LK only. Kick move perturbation of the current local minimum tour is typically achieved using a k'-change, with k' not necessarily equal to k. Both [Martin, Otto and Felten, 1991] and [Johnson, 1990] use random "double-bridge" 4-change kick moves, illustrated in Figure 2. According to [Martin, Otto and Felten, 1991], the double-bridge kick move is chosen for its ability to produce large-scale changes in the current tour without destroying the solution quality via too large a random perturbation. Furthermore, it is a relatively small perturbation whose "non-sequential" structure [Lin and Kernighan, 1973] is not easily reproduced by the Lin-Kernighan algorithm.

As can be seen from Figure 1, the LSMC method actually performs simulated annealing over local minima, with { kick move + *Descent* } as the neighborhood operator. In other words, if a new local minimum has lower cost than its predecessor, it is always adopted as the current solution; otherwise, it is adopted with probability given by the Boltzmann acceptance criterion (cf. Step 2.3 in Figure

---

[1] Some fairly standard terminology: A k-change operation modifies the current tour by removing up to k existing edges and reconnecting the resulting tour fragments into a new tour. A k-Opt method uses greedy search within the k-change neighborhood structure. A tour is "k-Opt" if it is locally minimum with respect to the k-change neighborhood structure.

```
Algorithm LSMC
Input: TSP instance,
       iteration bound $M$,
       temperature schedule $temp_i$, $i = 1, \ldots, M$
Output: heuristic tour $T_{best}$

1. Generate a random tour $T_{init}$;
   $T_1 = Descent(T_{init})$;
   $T_{best} = T_1$;

2. for $i = 1$ to $M$
   2.1. $T_i^* = kick\_move(T_i)$;
   2.2. $T_i^{**} = Descent(T_i^*)$;
   2.3. $diff = cost(T_i^{**}) - cost(T_i)$;
        if ( $diff < 0$ ) $T_{i+1} = T_i^{**}$;
        else {
            generate a random number $t \in [0, 1)$;
            if ( $t < \exp(-diff/temp_i)$ );
                $T_{i+1} = T_i^{**}$;
            else
                $T_{i+1} = T_i$;
        }
   2.4. if ( $cost(T_{best}) > \mathrm{cost}(T_i^{**})$ ) $T_{best} = T_i^{**}$;

3. return $T_{best}$;
```

Figure 1: The LSMC algorithm

1). Based on experimentally derived intuition, the authors of [Martin, Otto and Felten, 1991] set the temperature used for Boltzmann acceptance to be zero in most cases.[2] Johnson [Johnson, 1990] implicitly also sets the temperature to be uniformly zero. Thus, in both previous studies a new local minimum is generally not accepted unless it strictly improves over the current solution.[3]

It is worth noting that early work of Baum [Baum, 1986a] [Baum, 1986b] proposed an *iterated descent* approach that is essentially similar to the more recent LSMC strategies. However, Baum's implementation suffered from poor choices of the *Descent* and kick move elements: (i) he used 2-Opt and 3-Opt for local optimization, with 2-change kick moves applied until the search was considered to be trapped in a suboptimal basin of attraction; and (ii) he then applied two successive random

---

[2] In [Martin, Otto and Felten, 1991], the use of non-zero temperature was discussed in the context of the LIN318 example, but no details of the implementation were given.

[3] Martin and Otto [Martin and Otto, 1996] have recently conjectured that for instances of "moderate" size (e.g., the ATT532 instance from TSPLIB) zero-temperature LSMC can be expected to return the global minimum solution, since the LSMC cost surface has at most only a few local minima. That zero-temperature LSMC has very few basins of attraction is also in some sense an implicit assumption of Applegate et al. [Applegate et al., 1995], who use the union of edges in a very small number (10) of LSMC solutions to generate a good bounding solution within a branch-and-bound approach; this method has resulted in several recent "world's records" for optimal solution of large TSP instances.

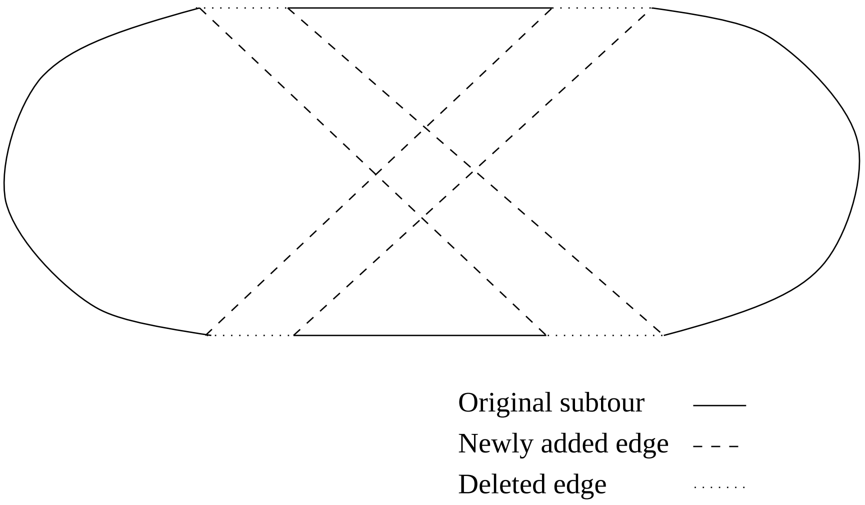

Figure 2: Illustration of the double-bridge 4-change kick move.

2-changes (i.e., a type of 4-change) as a kick move in order to escape the current basin of attraction. Interestingly, Baum's choice of 2-change as the basis of his kick move stemmed from his belief that if a 2- or 3-Opt solution is perturbed by a 2-change, then the next greedy descent will almost always lead to a new local minimum. On the other hand, [Martin, Otto and Felten, 1991] states that the above kick move is generally unable to escape the current basin of attraction (i.e., local optimization will return to the same local minimum); the double-bridge 4-change kick move is partly motivated by this belief. (Our experiments with the LIN318 instance from TSPLIB indicate that a new local optimum is obtained just under 10 percent of the time when 2-change and local optimization are applied to a 2- or 3-Opt solution.)

As a related development in the literature, we note that periodic kick moves of varied size (and strength) are a fundamental theme of the tabu search approach called strategic oscillation [Glover, 1977], which has been applied effectively to diverse combinatorial optimization problems (see, e.g., the survey of [Glover, 1996]). Applications to graph problems often take the form of alternately deleting and then adding (or adding and then deleting) different collections of elements in controlled steps that destroy and then recover feasibility – a pattern that is also achieved by a k-change kick move in the TSP setting. This provides an interesting connection between the strategic oscillation approach and the LSMC approach.

1.2 Scope of the Present Study

In this work, we study the performance of LSMC on randomly-generated symmetric TSP instances[4] using a variety of local optimization heuristics and kick moves. In particular, we examine relationships among (i) the underlying local optimization engine within the LSMC approach, (ii) the "kick

---

[4] Our instances have distance matrices either (i) corresponding to randomly-generated pointsets in the Euclidean plane, or (ii) corresponding to independent random inter-city distances taken from a uniform distribution.

move" perturbation that is applied between successive local search descents, and (iii) the resulting LSMC solution quality. In Section 2, we show that the traditional "double-bridge" kick move is not necessarily optimum: stronger local optimization engines (e.g., Lin-Kernighan) are best matched with stronger kick moves. In Section 3, we also propose use of a simple adaptive temperature schedule to allow escape from deep basins of attraction; the resulting *hierarchical* LSMC variant outperforms traditional LSMC implementations that use uniformly zero temperatures. Finally, Section 4 studies a population-based LSMC variant wherein multiple solution paths are allowed to cooperate in order to possibly achieve improved solution quality.

Our experiments use 2-Opt, 3-Opt and LK local optimization engines provided by [Boese, 1995]. These codes implement speedups of 2- and 3-Opt which are described in [Bentley, 1992] and reviewed in [Johnson and McGeoch, 1997]. Using a strategy originally due to Lin and Kernighan [Lin and Kernighan, 1973], a *neighbor-list* stores the 25 nearest neighbors for each city; this is implemented via the *k-d* tree [Bentley, 1990] data structure and simplifies finding improving moves or verifying local optimality. Following the terminology of [Johnson, 1990] [Baum, 1986a] [Baum, 1986b], we call the associated LSMC implementations Iterated 2-Opt, Iterated 3-Opt and Iterated LK, respectively.

As testbeds, we use three instances whose optimal tour costs are known. LIN318 and ATT532 are well-known geometric TSP instances used in many other studies, e.g., [Martin, Otto and Felten, 1991] [Martin, Otto and Felten, 1992] [Martin and Otto, 1996] [Johnson, 1990]; the former has 318 city locations and is due to Lin and Kernighan [Lin and Kernighan, 1973], while the latter is derived from the locations of 532 cities in the continental United States. The optimal tour lengths are known to be 42029 (LIN318) and 27686 (ATT532).[5] The third instance is S800, an 800-city random symmetric instance with integer inter-city distances randomly chosen from the interval $[1, 10000]$. The optimal tour length for S800 is known to be 20328 [Applegate, 1995].

Generally, all performance comparisons in this paper are based on the averages of solution costs over 50 trials.[6] The average running times are reported in CPU seconds on an HP-Apollo 9000/735 workstation.

2 Studies of Kick Move Strength

The studies of [Baum, 1986a] [Baum, 1986b] and [Martin, Otto and Felten, 1991] clearly point out the effect of kick move choice on solution quality. Intuitively, the kick move size should be such that (i) it is likely that the search can reach a new basin of attraction, yet (ii) the current level of solution quality is not lost through random perturbation. As noted above, both [Martin, Otto and Felten, 1991] and

---

[5] City locations are specified by integer coordinates in the Euclidean plane. Inter-city distances are maintained as double-precision reals, and tour costs are rounded to the nearest integer.

[6] The exception is the Iterated LK results for instance S800; running times were such that we are able to report only the average of 2 trials.

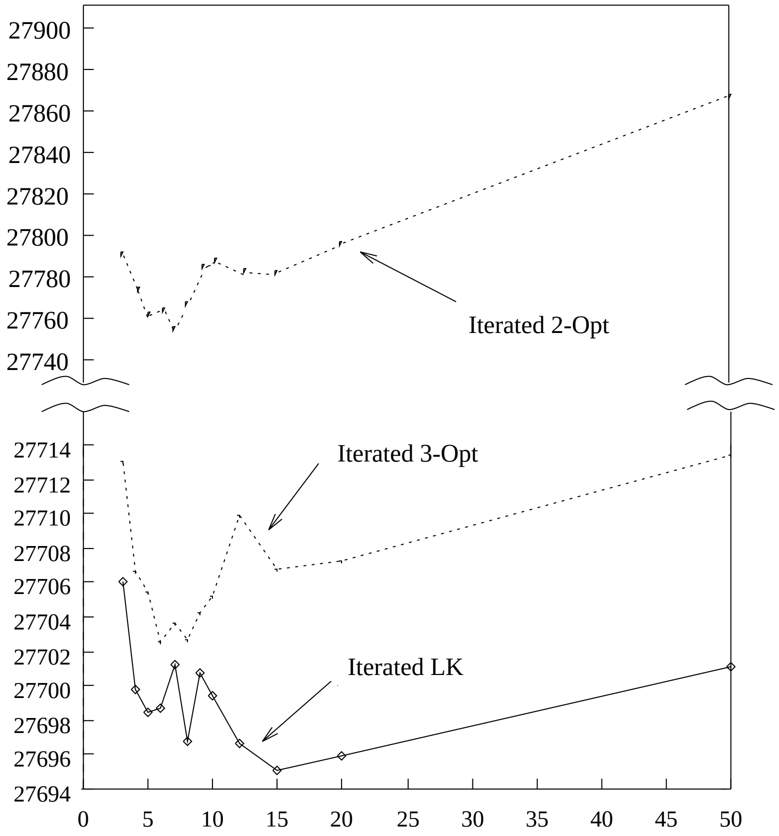

Figure 3: Average solution cost plotted against kick move size for ATT532.

[Johnson, 1990] used the double-bridge 4-change kick move. [Martin, Otto and Felten, 1991] allowed only kick moves that did not increase the tour length by more than some "small integer multiple" of the smallest inter-city distance[7] while [Johnson, 1990] imposed no such restrictions.

In this section, we describe studies of the relationship between kick move size (i.e., strength of the perturbation) and LSMC performance. For each of Iterated 2-Opt, Iterated 3-Opt, and Iterated LK, we studied k-change kick moves over the range of values $k = 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 50$. In our implementation, a k-change randomly removes k edges from the current tour and then reconnects the tour fragments according to a fixed template.[8]

Tables 8, 9 and 10 in the Appendix give the experimental data for Iterated 2-Opt. In each table, "Cost" indicates the average solution cost expressed as a percentage above optimum. The

---

[7] Efforts to contact the authors of [Martin, Otto and Felten, 1991] in order to determine the value of this "small integer" have been unsuccessful.

[8] In other words, there may be many ways to reconnect the tour fragments in, say, a 7-change – but we always reconnect the fragments in the same way. There were two exceptions to this practice: for 3-change we randomly chose among the four possible patterns that could reconnect the tour fragments, and for 4-change we used the (unrestricted) double-bridge 4-change as in [Johnson, 1990].

tables show solution costs at intervals of 1000 iterations (up to 5000, 10000 and 10000 iterations for LIN318, ATT532 and S800, respectively). Tables 11, 12 and 13 show results for the same experimental protocol, using Iterated 3-Opt. Finally, Tables 14, 15 and 16 show results from the same experimental protocol, using Iterated LK. However, for this experiment we report solution costs up to 1000, 2000 and 2000 iterations, at intervals of 200 iterations.[9] Figure 3 gives a graphical summary of the solution quality data for the ATT532 instance (maximum number of iterations).

For Iterated 2-Opt, the best kick moves are 5-change, 7-change and 4-change for LIN318, ATT532 and S800, respectively. For Iterated 3-Opt, the best kick moves are 8-change, 8-change and 3-change, respectively. And for Iterated LK, the best kick moves are $\{8, 15, 20, 50\}$-change, 15-change and 3-change, respectively. For the geometric instances (LIN318 and ATT532), stronger local optimization engines appear more suited to stronger kick moves. For the random symmetric instance (S800), 3-change always seems to produce the best results. This evidence suggests that the traditional double-bridge 4-change kick move is suboptimal, particularly for the geometric instances that are so frequently studied in the literature.

3 Adaptive Temperature Schedules

Recall that LSMC may be viewed as a variant of simulated annealing that moves among local optima, and that the annealing temperature in previous LSMC studies was typically set to zero (or left unspecified if non-zero temperatures were used). Also recall that the success of zero-temperature LSMC has led [Martin and Otto, 1996] [Applegate et al., 1995] to explicitly conjecture or implicitly assume that the LSMC cost surface (under the { kick move + *Descent* } neighborhood operator) has very few local minima. However, it is our experience that zero-temperature LSMC is often trapped in local minima which are not globally optimum. Thus, in this section we propose use of stronger perturbations which enable escape from spurious local minima. This results in a *hierarchical LSMC* approach, with non-zero annealing temperatures constituting a "higher-level kick move" that is applied when the LSMC search is perceived to be trapped. Baum [Baum, 1986a] [Baum, 1986b] also raised the issue of escaping spurious local minima, and proposed applying a (structurally) stronger kick move while remaining within the zero-temperature regime.

In our simple variant of hierarchical LSMC, the original zero-temperature LSMC is considered "stuck" if no local minimum has been accepted in the last $2n$ iterations (for Iterated 2-Opt and Iterated 3-Opt) or in the last 100 iterations (for Iterated LK). When the zero-temperature LSMC becomes stuck, the annealing temperature is set to the current solution cost $cost(T_i)$ divided by 200, for 100 iterations, after which it is reset to zero.[10] Table 1 compares the performance of our

---

[9] Johnson and McGeoch [Johnson and McGeoch, 1997] uses n Iterated LK iterations for an n-city TSP instance. Thus, our tables also include data for $n/8$, $n/4$, $n/2$ and n iterations of Iterated LK, as well as data for $n/2$ and n iterations of the other two methods.

[10] Hence, the probability of acceptance is $exp(-200 \cdot diff/cost(T_i))$, e.g., acceptance probability $1/e$ for an 0.5% cost increase. We also tried two other hierarchical LSMC variants. In the first, the temperature is raised to $cost(T_i)/100$

Instance	LIN318				ATT532				S800			
Number of Iterations	4-change		5-change		4-change		7-change		4-change		3-change	
	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$
1000	.872	.840	.822	.805	.699	.689	.688	.670	77.3	77.3	77.8	77.8
2000	.658	.524	.596	.535	.528	.512	.486	.453	66.3	66.3	65.4	65.4
3000	.548	.469	.463	.413	.475	.458	.416	.386	59.6	59.6	58.9	58.9
4000	.458	.356	.410	.324	.432	.412	.354	.342	55.4	55.4	54.6	54.6
5000	.405	.309	.381	.308	.401	.375	.335	.311	51.9	51.9	51.4	51.4
6000					.384	.362	.312	.298	49.3	49.3	48.9	48.9
7000					.371	.349	.300	.288	47.0	47.0	46.9	46.9
8000					.353	.321	.278	.267	44.9	44.9	45.2	45.2
9000					.342	.289	.262	.251	43.3	43.3	43.7	43.7
10000					.330	.262	.250	.239	41.7	41.7	42.2	42.2

(a) Iterated 2-Opt

Instance	LIN318				ATT532				S800			
Number of Iterations	4-change		8-change		4-change		8-change		4-change		3-change	
	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$
1000	.175	.143	.107	.095	.142	.142	.136	.136	12.83	12.83	13.19	13.19
2000	.147	.059	.091	.037	.106	.097	.101	.103	11.46	11.46	11.37	11.37
3000	.129	.037	.065	.007	.099	.082	.087	.091	10.55	10.55	10.37	10.37
4000	.124	.019	.054	.006	.093	.073	.082	.080	9.93	9.93	9.65	9.65
5000	.105	.000	.047	.000	.088	.064	.080	.072	9.36	9.36	9.18	9.18
6000					.081	.062	.069	.064	9.01	9.01	8.81	8.81
7000					.079	.060	.064	.059	8.73	8.73	8.45	8.45
8000					.076	.059	.062	.048	8.42	8.42	8.17	8.17
9000					.075	.057	.060	.046	8.17	8.17	7.91	7.91
10000					.075	.056	.060	.044	7.95	7.95	7.70	7.70

(b) Iterated 3-Opt

Instance	LIN318				ATT532				S800			
Number of Iterations	4-change		8-change		4-change		15-change		4-change		3-change	
	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$	$t = 0$	$t \neq 0$
200	.153	.151	.093	.089	.125	.125	.129	.129	1.641	1.641	1.796	1.796
400	.082	.048	.020	.017	.095	.093	.091	.090	1.589	1.590	1.471	1.471
600	.047	.017	.012	.007	.078	.076	.071	.070	1.368	1.411	1.378	1.378
800	.012	.005	.003	.002	.068	.065	.064	.063	1.281	1.282	1.378	1.376
1000	.005	.000	.000	.000	.063	.060	.060	.058	1.250	1.282	1.378	1.169
1200					.061	.057	.051	.046	1.250	1.275	1.173	1.169
1400					.058	.053	.050	.043	1.250	1.272	1.095	1.095
1600					.056	.051	.042	.037	1.250	1.258	1.095	1.092
1800					.052	.048	.038	.033	1.218	1.258	1.095	1.088
2000					.050	.045	.033	.031	1.218	1.257	1.067	1.059

(c) Iterated LK

Table 1: Performance of hierarchical LSMC ($t \neq 0$) versus zero-temperature LSMC ($t = 0$).

hierarchical LSMC implementation with that of zero-temperature LSMC for Iterated 2-Opt, Iterated

---

and linearly cooled back to zero over 100 iterations; in the second, the temperature is raised to $+\infty$ for a small number of iterations (both 3 and 5 were tested) and then reset to zero. Results for these variants were essentially identical to the results that we report here.

3-Opt and Iterated LK. We report data for both the 4-change kick move and the most successful kick move among those tested in Section 2.[11] For the two geometric instances, hierarchical LSMC clearly outperforms zero-temperature LSMC. The advantage of hierarchical LSMC is less obvious for the S800 instance, except when used with the Iterated LK engine. Generally, for S800 we find that zero-temperature LSMC is only rarely "stuck"; understanding this phenomenon is an interesting future direction.

```
Algorithm Population-based (zero-temperature) LSMC
Input: TSP instance,
       iteration bound $M$,
       population size $p$,
       update interval $I$
Output: heuristic tour $T_{best}$

1. For $i = 1$ to $p$,
       generate a random tour $T_i$;
       $T_i = Descent(T_i)$;
   $T_{best}$ = minimum-cost tour among all $T_i$'s;

2. do {
     2.1 For each solution $T_i$, do the following $I$ times:
             $T_i^* = kick\_move(T_i)$;
             $T_i^{**} = Descent(T_i^*)$;
             if ( $cost(T_i^{**} < cost(T_i)$ ) $T_i = T_i^{**}$;
             if ( $cost(T_{best}) > cost(T_i^{**})$ ) $T_{best} = T_i^{**}$;
     2.2 Choose a solution $T_j$ based on proportional selection;
     2.3 $T_j^* = kick\_move(T_j)$;
     2.4 $T_j^{**} = Descent(T_j^*)$;
     2.5 if ( $cost(T_{best}) > cost(T_j^{**})$ ) $T_{best} = T_j^{**}$;
     2.6 Replace $T_j^{**}$ with a solution in the population;
     } while ( number of $Descents$ performed is $\leq M$ );

3. return $T_{best}$;
```

Figure 4: A template for population-based LSMC

4 Population-Based Search

LSMC executes a "monolithic" or "single-threaded" search, i.e., only one solution is saved at any given time. It is intriguing to consider the contrast between the single-threaded LSMC search

---

[11] Due to the definition of the "stuck" criterion, there is no difference between the zero-temperature and hierarchical LSMC implementations for n or fewer iterations; thus, we do not report results for small iteration bounds.

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
1000	.129	.038	.066	.050	.067
2000	.105	.014	.021	.023	.021
3000	.077	.011	.016	.014	.010
4000	.076	.011	.016	.009	.005
5000	.068	.011	.016	.005	.004

(a) LIN318

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
1000	.128	.144	.148	.138	.151
2000	.105	.101	.099	.099	.104
3000	.093	.091	.091	.087	.088
4000	.088	.083	.083	.076	.081
5000	.082	.081	.080	.072	.077
6000	.073	.077	.076	.070	.072
7000	.071	.076	.073	.065	.067
8000	.071	.070	.072	.060	.064
9000	.067	.070	.066	.056	.061
10000	.066	.068	.066	.055	.060

(b) ATT532

Table 2: Population-based LSMC with 3-Opt descent engine: Relationship between update interval and solution cost for population size 5.

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
1000	.063	.079	.077	.064	.082
2000	.053	.031	.019	.012	.027
3000	.049	.018	.005	.002	.008
4000	.041	.010	.004	.001	.003
5000	.037	.001	.004	.000	.003

(a) LIN318

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
1000	.141	.195	.186	.203	.204
2000	.108	.116	.102	.126	.125
3000	.094	.088	.088	.098	.091
4000	.085	.078	.071	.084	.076
5000	.082	.072	.068	.074	.070
6000	.077	.065	.063	.069	.063
7000	.074	.061	.060	.067	.061
8000	.074	.060	.059	.063	.057
9000	.073	.059	.054	.060	.055
10000	.069	.057	.054	.057	.052

(b) ATT532

Table 3: Population-based LSMC with 3-Opt descent engine: Relationship between update interval and solution cost for population size 10.

and, e.g., “adaptive multi-start” techniques [Boese, Kahng and Muddu, 1994] which exploit struc-

Iter.	$I=0$	$I=5$	$I=10$	$I=20$	Independent
1000	.049	.100	.117	.128	.125
2000	.016	.039	.035	.043	.037
3000	.011	.015	.020	.018	.013
4000	.011	.004	.004	.009	.005
5000	.011	.001	.002	.004	.002

(a) LIN318

Iter.	$I=0$	$I=5$	$I=10$	$I=20$	Independent
1000	.169	.260	.264	.264	.269
2000	.121	.153	.166	.161	.174
3000	.100	.116	.123	.121	.123
4000	.087	.086	.106	.097	.104
5000	.083	.072	.086	.081	.091
6000	.080	.063	.080	.073	.081
7000	.078	.060	.073	.069	.072
8000	.072	.058	.069	.064	.065
9000	.072	.053	.065	.057	.061
10000	.069	.049	.061	.054	.057

(b) ATT532

Table 4: Population-based LSMC with 3-Opt descent engine: Relationship between update interval and solution cost for population size 20.

tural relationships among local optima [Boese, 1995].[12] Taking advantage of relationships between high quality solutions is also a feature of tabu search intensification strategies, which employ strongly determined and consistent variables together with memory-based processes to reinforce or reinstate attributes of elite solutions (see, e.g., [Glover, 1977], [Glover and Laguna, 1993]). In this section, we note that *population-based search* offers a convenient framework for hybridizing monolithic search and multi-start search. Within this framework, we ask whether solution quality can be improved by using a population of local search processes which have "evolutionary" interdependencies. For example, some fixed number of solutions may be evolved according to evolutionary programming [Fogel, Owens and Walsh, 1966] or "go with the winners" [Aldous and Vazirani, 1994] paradigms, each of which uses an evolving population with no concept of "crossover".

Given a fixed CPU budget of M descents, a population size p, and an "update interval" I, our *population-based LSMC* approach follows the outline of Figure 4: we maintain a population of p separate LSMC searches that interact every $p \cdot I$ descents. More precisely, we initially compute p local minima and then repeat the following main loop until M descents have been performed. In the loop, I zero-temperature { kick move + *Descent* } cycles are applied to each of the p existing solutions. Then, one solution (T_j) is chosen based on a roulette-wheel selection scheme [Goldberg, 1989] wherein

---

[12] LSMC might be viewed as constituting search along the "boundary" of the "big valley" that governs 2-Opt, 3-Opt and LK local minima [Boese, Kahng and Muddu, 1994] [Boese, 1995]. Adaptive multi-start techniques attempt to restart the local search in the "interior" of this big valley. Cf. hybrid genetic-local search metaheuristics, e.g., [Mühlenbein, Georges-Schleuter and Krämer, 1988] [Ulder et al., 1990].

Iter.	$I=0$	$I=5$	$I=10$	$I=20$	Independent
200	.089	.120	.078	.077	.093
400	.039	.049	.027	.013	.035
600	.020	.034	.013	.011	.014
800	.000	.014	.006	.000	.011
1000	.000	.000	.000	.000	.000

(a) LIN318

Iter.	$I=0$	$I=5$	$I=10$	$I=20$	Independent
200	.105	.174	.159	.177	.161
400	.077	.111	.095	.123	.099
600	.064	.082	.075	.090	.079
800	.060	.068	.070	.073	.068
1000	.057	.061	.062	.058	.061
1200	.056	.057	.058	.051	.060
1400	.052	.051	.048	.047	.054
1600	.048	.046	.043	.046	.052
1800	.048	.044	.040	.043	.051
2000	.048	.043	.037	.040	.049

(b) ATT532

Table 5: Population-based LSMC with LK descent engine: Relationship between update interval and solution cost for population size 5.

the best solution is four times as likely to be selected as the worst solution. If one more { kick move + *Descent* } cycle applied to T_j produces a new solution T_j^{**} that is better than T_j, then T_j^{**} replaces T_j in the current population. Otherwise, T_j^{**} replaces the worst solution in the current population, as long as T_j^{**} has better cost than this solution.

A wide spectrum of population-based LSMC implementations is possible, based on the value of I. At one extreme, $I = 0$ and the p solution streams have the strongest possible interaction: no iterations are made before the next solution is selected for possible replacement. At the other extreme, $I = \infty$ is equivalent to multi-start LSMC with p independent runs of M/p descents each. In our tests, we tried 3-Opt and LK as the local optimization engine with the double-bridge 4-change kick move; examining all possible combinations was beyond the scope of our study. Values of $I = 0, 5, 10, 20$ and ∞ were tested with population sizes $p = 5, 10$ and 20. A fixed CPU budget was used for all experiments; for Iterated 3-Opt (Iterated LK), we used $M = 5000$ ($M = 1000$) descents for LIN318 and $M = 10000$ ($M = 2000$) descents for ATT532. Tables 2, 3 and 4 show that for Iterated 3-Opt, the combination ($I = 20$, $p = 10$) was best for LIN318, while the combination ($I = 5$, $p = 20$) was best for ATT532. Tables 5, 6 and 7 show that for Iterated LK, the combination ($I = 5$, $p = 10$) was best for LIN318, while the combination ($I = 10$, $p = 5$) was best for ATT532. Note that the original Iterated 3-Opt (Iterated LK) studied in Section 2 averages .105% (.005%) above optimum for LIN318 and .075% (.050%) above optimum for ATT532, while the best population-based LSMC variant with 3-Opt (LK) local optimization engine averages 0% (0%) above optimum for LIN318 and

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
200	.083	.113	.149	.125	.125
400	.014	.033	.038	.043	.048
600	.014	.000	.013	.025	.024
800	.000	.000	.000	.011	.009
1000	.000	.000	.000	.005	.000

(a) LIN318

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
200	.163	.184	.193	.185	.185
400	.099	.124	.112	.133	.126
600	.079	.094	.091	.084	.096
800	.075	.078	.075	.067	.082
1000	.067	.066	.069	.062	.068
1200	.063	.062	.062	.055	.057
1400	.061	.057	.055	.051	.050
1600	.057	.052	.055	.049	.046
1800	.052	.048	.049	.044	.046
2000	.049	.044	.049	.039	.041

(b) ATT532

Table 6: Population-based LSMC with LK descent engine: Relationship between update interval and solution cost for population size 10.

.049% (.037%) above optimum for ATT532.

In our population-based LSMC approach, the parameter I trades off between coupling of the p search processes and diversity among the p current solutions. Our experimental data indicate that even pure multi-start LSMC ($I = \infty$) is superior to the original LSMC. At the same time, it seems obvious that for very small CPU budgets, lower values of I (e.g., $I = 0$) will be the most successful. Tuning the values of p and I to a given instance and CPU budget M remains an open issue.

5 Conclusion

We have provided extensive experimental studies of traditional zero-temperature LSMC, following the original implementations reported by [Martin, Otto and Felten, 1991] and [Johnson, 1990]. Experiments with various k-change kick moves suggest that the traditional double-bridge 4-change kick move is not optimum, and that the best kick move strongly depends on both the underlying local optimization engine and the type of instance.[13] We have also proposed a *hierarchical LSMC* strategy which significantly improves performance over the previous zero-temperature LSMC implementations.

---

[13] Recall that we used a fixed "template" to implement the k-change kick move for most values of k studied. It is possible that other k-change templates, or the use of random k-changes, would yield different results. At present, we cannot assess the "strength" of a k-change kick move as a function of k; it would be useful to develop a quantitative measure which enables more precise study of the relationship between perturbation strength and LSMC performance.

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
200	.138	.157	.159	.156	.159
400	.033	.044	.056	.028	.048
600	.014	.028	.015	.005	.014
800	.005	.006	.006	.000	.000
1000	.000	.000	.000	.000	.000

(a) LIN318

Iter.	$I = 0$	$I = 5$	$I = 10$	$I = 20$	Independent
200	.194	.275	.270	.270	.270
400	.109	.185	.181	.185	.185
600	.088	.132	.139	.134	.143
800	.075	.109	.114	.119	.116
1000	.070	.096	.091	.088	.090
1200	.065	.082	.082	.082	.080
1400	.059	.072	.075	.079	.070
1600	.059	.068	.068	.073	.066
1800	.058	.063	.063	.065	.060
2000	.056	.061	.061	.062	.058

(b) ATT532

Table 7: Population-based LSMC with LK descent engine: Relationship between update interval and solution cost for population size 20.

Further studies might address in more detail the relationship between optimal temperature schedules and structural parameters of the cost surface. Finally, we have proposed the use of *population-based LSMC* strategies which again seem to offer improvements over the previous LSMC implementations. Here, the key open issue involves tuning the update interval and population size parameters to the given problem instance and CPU budget.

Acknowledgments

We thank Kenneth D. Boese for providing source codes for the local optimization engines used in this study. We also thank David S. Johnson and Kenneth D. Boese for their valuable comments and careful reading of this manuscript.

Appendix

	Multi-Start		2-change		3-change		4-change		5-change		6-change		7-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
159	3.47	14	6.4	0	2.022	1	1.961	1	1.858	2	2.022	2	2.034	2
318	3.17	28	6.4	1	1.472	2	1.442	3	1.378	3	1.523	4	1.478	4
1000	2.90	91	6.4	2	.937	6	.872	9	.822	10	.923	12	.831	12
2000	2.70	182	6.4	4	.706	12	.658	18	.596	20	.682	24	.604	24
3000	2.65	274	6.4	6	.641	18	.548	27	.463	30	.596	36	.549	35
4000	2.59	365	6.4	8	.597	24	.458	35	.410	40	.540	48	.512	48
5000	2.51	456	6.4	10	.588	30	.405	44	.381	50	.487	60	.481	60

	8-change		9-change		10-change		12-change		15-change		20-change		50-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
159	2.156	2	2.203	2	2.258	3	2.008	3	1.894	3	1.977	4	2.101	4
318	1.692	4	1.711	5	1.811	5	1.492	5	1.451	6	1.480	6	1.813	8
1000	.999	14	1.100	15	1.094	16	.875	17	.921	19	.987	21	1.274	25
2000	.759	28	.783	30	.769	31	.664	34	.700	38	.830	42	1.036	50
3000	.631	42	.605	45	.637	47	.604	51	.610	57	.741	62	.931	75
4000	.548	56	.501	59	.561	62	.569	68	.565	76	.682	83	.863	100
5000	.483	69	.449	74	.517	78	.535	85	.538	94	.624	103	.819	125

Table 8: Relationship between kick-move strength and performance of Iterated 2-Opt for LIN318.

	Multi-Start		2-change		3-change		4-change		5-change		6-change		7-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
266	4.25	80	6.12	1	1.524	4	1.336	4	1.351	6	1.340	7	1.398	7
532	3.53	160	6.12	2	1.069	8	.928	9	.961	11	.939	14	.917	13
1000	3.19	300	6.12	3	.818	14	.699	17	.690	21	.675	26	.688	25
2000	3.03	601	6.12	7	.623	28	.528	33	.509	42	.504	51	.486	51
3000	2.91	901	6.12	10	.556	42	.475	49	.428	62	.423	76	.416	76
4000	2.80	1201	6.12	13	.507	56	.432	65	.376	82	.392	100	.354	100
5000	2.75	1502	6.12	16	.485	70	.401	80	.346	103	.358	125	.335	124
6000	2.75	1803	6.12	19	.462	84	.384	96	.320	123	.330	149	.312	149
7000	2.69	2103	6.12	22	.427	98	.371	111	.303	144	.316	173	.300	173
8000	2.66	2404	6.12	25	.409	112	.353	127	.297	164	.307	198	.278	197
9000	2.62	2704	6.12	28	.397	126	.342	142	.285	185	.290	222	.262	222
10000	2.62	3004	6.12	31	.383	140	.330	158	.272	205	.283	246	.250	246

	8-change		9-change		10-change		12-change		15-change		20-change		50-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
266	1.542	9	1.755	10	1.759	11	1.394	12	1.615	12	1.676	12	1.827	12
532	1.127	18	1.333	20	1.398	22	1.098	23	1.174	23	1.250	24	1.421	24
1000	.884	34	1.033	38	1.075	42	.835	43	.826	43	.967	44	1.193	45
2000	.625	66	.723	72	.770	83	.598	84	.602	85	.728	86	1.028	88
3000	.519	97	.615	106	.633	122	.530	123	.527	124	.601	126	.925	131
4000	.440	129	.545	141	.550	161	.466	161	.463	161	.537	165	.837	173
5000	.394	160	.488	175	.504	201	.433	202	.418	204	.498	207	.789	215
6000	.362	192	.447	220	.469	240	.412	241	.397	245	.471	249	.738	257
7000	.343	223	.416	243	.434	279	.397	280	.379	286	.446	290	.711	299
8000	.330	255	.396	278	.412	318	.381	318	.362	325	.432	331	.707	341
9000	.304	286	.373	312	.392	356	.365	357	.355	366	.414	371	.665	383
10000	.294	317	.358	347	.372	395	.346	397	.345	408	.403	415	.652	424

Table 9: Relationship between kick-move strength and performance of Iterated 2-Opt for ATT532.

	Multi-Start		2-change		3-change		4-change		5-change		6-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
400	131	200	143	5	93.9	10	92.7	10	95.1	12	96.2	12
800	127	400	143	10	81.9	19	80.8	21	85.0	24	87.2	24
1000	122	501	143	12	77.8	24	77.3	26	81.6	30	84.2	32
2000	118	1000	143	24	65.4	46	66.3	49	72.9	57	76.0	62
3000	118	1502	142	36	58.9	67	59.6	71	67.2	83	71.5	91
4000	117	2001	142	47	54.6	87	55.4	92	63.7	109	68.1	119
5000	117	2499	142	59	51.4	107	51.9	113	60.7	135	65.5	148
6000	116	3002	142	71	48.9	127	49.3	134	58.5	160	63.1	175
7000	116	3503	142	82	46.9	146	47.0	154	56.7	185	61.6	203
8000	115	4007	142	94	45.2	166	44.9	174	54.9	210	60.2	231
9000	115	4510	142	106	43.7	185	43.3	194	53.3	235	58.9	258
10000	115	5011	142	117	42.2	204	41.7	214	52.0	259	57.5	285

Table 10: Relationship between kick-move strength and performance of Iterated 2-Opt for S800.

	Multi-Start		2-change		3-change		4-change		5-change		6-change		7-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
159	.75	9	1.34	1	.466	2	.347	3	.305	3	.269	4	.338	4
318	.64	18	1.25	2	.383	4	.226	5	.231	6	.207	7	.257	7
1000	.46	56	1.19	7	.320	12	.175	18	.151	21	.145	23	.161	22
2000	.39	113	1.18	14	.293	25	.147	36	.115	41	.110	45	.128	44
3000	.35	169	1.18	21	.276	37	.129	55	.109	61	.102	67	.113	65
4000	.32	226	1.18	28	.246	49	.124	73	.103	81	.075	90	.100	87
5000	.31	282	1.18	35	.237	62	.105	91	.096	102	.052	112	.075	109

	8-change		9-change		10-change		12-change		15-change		20-change		50-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
159	.257	4	.264	4	.307	4	.321	5	.286	5	.316	5	.471	6
318	.190	8	.198	8	.219	8	.271	9	.224	9	.233	10	.326	12
1000	.107	25	.148	27	.157	27	.157	29	.172	31	.173	34	.183	39
2000	.091	50	.112	53	.111	54	.108	56	.151	59	.139	63	.158	70
3000	.065	75	.074	80	.087	81	.097	83	.108	87	.094	91	.143	98
4000	.054	100	.070	106	.064	108	.084	111	.090	116	.094	120	.127	131
5000	.047	126	.067	133	.049	135	.076	138	.087	145	.087	152	.126	168

Table 11: Relationship between kick-move strength and performance of Iterated 3-Opt for LIN318.

	Multi-Start		2-change		3-change		4-change		5-change		6-change		7-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
266	.95	29	1.008	3	.318	6	.242	8	.246	9	.246	9	.264	9
532	.71	57	.881	5	.231	10	.173	15	.184	17	.184	19	.188	19
1000	.60	111	.830	10	.190	20	.142	29	.150	32	.133	36	.140	36
2000	.59	221	.821	20	.148	39	.106	56	.113	64	.103	72	.112	72
3000	.58	332	.820	29	.127	58	.099	85	.099	96	.088	108	.095	107
4000	.57	443	.791	38	.120	76	.093	112	.092	128	.081	143	.088	142
5000	.54	557	.791	47	.114	95	.088	140	.088	159	.075	179	.084	178
6000	.53	663	.791	56	.109	113	.081	168	.083	191	.072	214	.077	214
7000	.52	777	.791	65	.105	132	.079	196	.077	224	.070	250	.075	250
8000	.51	890	.791	74	.100	150	.076	224	.073	256	.067	285	.074	285
9000	.50	1002	.791	83	.098	169	.075	251	.072	288	.063	321	.068	320
10000	.49	1115	.791	92	.098	187	.075	279	.070	321	.060	356	.064	356

	8-change		9-change		10-change		12-change		15-change		20-change		50-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
266	.246	11	.275	12	.314	12	.224	13	.220	14	.249	14	.282	16
532	.173	22	.199	23	.217	24	.159	26	.163	28	.184	29	.206	32
1000	.136	42	.146	44	.164	46	.136	49	.136	52	.147	55	.172	60
2000	.101	83	.112	88	.121	91	.115	97	.111	100	.124	108	.142	116
3000	.087	124	.098	132	.101	136	.108	146	.102	149	.105	162	.132	174
4000	.082	165	.087	177	.091	180	.101	193	.096	198	.099	210	.128	232
5000	.080	205	.082	221	.087	225	.094	241	.093	248	.089	263	.116	291
6000	.069	246	.077	265	.082	270	.091	290	.085	299	.087	316	.111	350
7000	.064	287	.074	309	.078	315	.091	338	.083	348	.084	423	.107	409
8000	.062	328	.069	353	.077	360	.090	385	.083	397	.084	423	.101	468
9000	.060	369	.067	397	.073	404	.088	432	.077	445	.084	475	.101	527
10000	.060	410	.066	441	.069	449	.086	481	.075	496	.077	529	.099	585

Table 12: Relationship between kick-move strength and performance of Iterated 3-Opt for ATT532.

	Multi-Start		2-change		3-change		4-change		5-change		6-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
400	27.2	102	15.96	18	15.52	19	15.07	21	15.82	24	16.40	28
800	26.1	206	13.79	36	13.69	38	13.31	42	14.34	48	15.04	56
1000	25.4	260	13.25	45	13.19	47	12.83	52	13.83	61	14.67	70
2000	25.0	519	11.62	85	11.37	88	11.46	99	12.67	118	13.52	134
3000	24.8	781	10.70	123	10.37	129	10.55	145	11.96	173	12.85	202
4000	24.6	1040	10.22	161	9.65	168	9.93	189	11.45	228	12.40	270
5000	24.6	1305	9.91	199	9.18	207	9.36	234	11.04	282	12.04	335
6000	24.5	1567	9.77	236	8.81	245	9.01	277	10.66	335	11.77	403
7000	24.2	1829	9.64	273	8.45	284	8.73	320	10.36	388	11.52	469
8000	24.2	2090	9.52	310	8.17	321	8.42	363	10.13	440	11.35	533
9000	24.1	2349	9.46	347	7.91	359	8.17	406	9.92	493	11.16	599
10000	24.1	2613	9.42	384	7.70	397	7.95	448	9.75	545	10.98	664

Table 13: Relationship between kick-move strength and performance of Iterated 3-Opt for S800.

	Multi-Start		2-change		3-change		4-change		5-change		6-change		7-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
40	.447	96	1.29	13	.578	20	.324	26	.309	28	.288	30	.412	30
80	.353	192	1.25	26	.478	40	.214	52	.214	56	.202	60	.281	61
159	.195	370	1.21	49	.355	75	.169	96	.138	110	.124	121	.202	123
200	.171	481	1.20	66	.327	98	.153	130	.110	141	.100	152	.176	153
318	.142	760	1.15	95	.247	148	.102	195	.081	220	.069	242	.114	245
400	.119	960	1.14	125	.242	193	.082	256	.062	278	.036	300	.082	301
600	.088	1435	1.13	189	.214	289	.047	379	.051	409	.023	448	.057	450
800	.064	1921	1.11	253	.204	381	.012	506	.039	550	.011	599	.035	601
1000	.064	2407	1.11	315	.185	488	.005	635	.012	688	.005	742	.028	746

	8-change		9-change		10-change		12-change		15-change		20-change		50-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
40	.274	36	.240	37	.233	39	.371	41	.302	43	.250	47	.255	49
80	.183	72	.152	74	.143	78	.266	82	.219	86	.107	94	.145	99
159	.119	139	.079	142	.079	146	.105	161	.112	172	.076	183	.102	195
200	.093	179	.062	187	.054	194	.068	205	.086	214	.052	235	.067	248
318	.054	278	.031	285	.021	292	.060	321	.050	345	.033	367	.045	397
400	.020	355	.018	365	.016	389	.053	408	.032	421	.027	456	.033	492
600	.012	521	.009	529	.005	586	.029	611	.028	645	.000	688	.000	739
800	.003	695	.006	705	.005	779	.015	819	.015	858	.000	907	.000	980
1000	.000	870	.006	886	.005	972	.012	1028	.000	1061	.000	1135	.000	1223

Table 14: Relationship between kick-move strength and performance of Iterated LK for LIN318.

	Multi-Start		2-change		3-change		4-change		5-change		6-change		7-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
67	.45	426	1.156	86	.336	136	.209	204	.217	214	.202	227	.217	228
133	.39	853	1.098	172	.249	271	.144	409	.148	428	.141	453	.155	456
200	.34	1278	1.086	257	.196	408	.125	613	.118	642	.116	680	.126	685
266	.31	1632	1.076	328	.166	502	.108	802	.105	832	.105	863	.112	862
400	.27	2553	1.038	513	.139	804	.095	1222	.086	1276	.088	1359	.088	1362
532	.26	3265	1.004	660	.116	1005	.087	1605	.072	1658	.080	1726	.080	1728
600	.26	3835	.999	782	.112	1184	.078	1833	.069	1912	.075	2039	.077	2041
800	.26	5113	.969	1052	.098	1569	.068	2442	.064	2546	.068	2716	.069	2725
1000	.25	6391	.960	1324	.087	1952	.063	3048	.060	3179	.059	3396	.062	3412
1200	.23	7668	.956	1594	.084	2334	.061	3655	.054	3809	.053	4074	.062	4121
1400	.23	8947	.956	1869	.079	2719	.058	4260	.050	4430	.051	4755	.059	4793
1600	.22	10223	.951	2144	.076	3102	.056	4865	.048	5048	.049	5431	.059	5527
1800	.22	11503	.950	2421	.075	3400	.052	5470	.048	5612	.048	6106	.059	6226
2000	.22	12781	.950	2698	.072	3787	.050	6082	.045	6251	.046	6783	.055	6805

	8-change		9-change		10-change		12-change		15-change		20-change		50-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
67	.199	240	.246	250	.220	261	.228	270	.213	275	.199	284	.218	297
133	.130	480	.159	500	.163	522	.163	540	.152	552	.140	567	.151	594
200	.103	719	.140	751	.129	786	.120	811	.129	832	.117	852	.125	890
266	.090	902	.123	968	.112	998	.101	1035	.116	1084	.101	1103	.108	1180
400	.077	1434	.101	1502	.087	1537	.087	1597	.091	1641	.095	1680	.102	1763
532	.065	1802	.087	1932	.076	1994	.087	2071	.076	2165	.080	2201	.098	2359
600	.061	2145	.080	2246	.076	2279	.078	2365	.071	2450	.079	2508	.095	2626
800	.055	2858	.074	2987	.067	3083	.074	3157	.064	3257	.075	3336	.081	3497
1000	.050	3571	.066	3726	.061	3815	.070	3902	.060	4072	.068	4171	.073	4369
1200	.046	4282	.063	4465	.056	4534	.062	4656	.051	4881	.061	5001	.069	5239
1400	.043	4993	.060	5213	.055	5327	.057	5453	.050	5690	.055	5827	.066	6104
1600	.042	5701	.059	5968	.053	6087	.053	6205	.042	6497	.051	6652	.062	6957
1800	.041	6413	.055	6712	.051	6850	.042	7010	.038	7301	.039	7479	.059	7832
2000	.039	7119	.053	7459	.048	7621	.039	7812	.033	8105	.036	8302	.055	8699

Table 15: Relationship between kick-move strength and performance of Iterated LK for ATT532.

	Multi-Start		2-change		3-change		4-change		5-change		6-change	
Iter.	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
100	2.16	1810	3.016	618	1.835	765	1.855	1045	2.174	1186	1.953	1291
200	2.05	3618	2.526	1236	1.796	1529	1.641	2087	1.982	2365	1.778	2580
400	1.98	7205	2.499	2401	1.471	3056	1.589	4171	1.621	4628	1.530	5007
600	1.98	10817	2.315	3597	1.378	4601	1.368	6359	1.621	6917	1.476	7482
800	1.92	14421	2.315	4785	1.378	6112	1.281	8420	1.488	9254	1.476	9965
1000	1.92	18035	2.315	5992	1.378	7662	1.250	10519	1.488	11428	1.476	12411
1200	1.92	21644	2.315	7106	1.173	9200	1.250	12599	1.277	13672	1.476	14890
1400	1.92	25257	2.315	8318	1.095	10717	1.250	14687	1.274	15901	1.476	17392
1600	1.92	28863	2.283	9527	1.095	12269	1.250	16760	1.274	18103	1.429	19679
1800	1.92	32479	2.179	10736	1.095	13806	1.218	18813	1.269	20388	1.429	22031
2000	1.92	36069	2.125	11917	1.067	15354	1.218	20851	1.252	22496	1.380	24389

Table 16: Relationship between kick-move strength and performance of Iterated LK for S800.

References

[Aldous and Vazirani, 1994] D. Aldous and U. Vazirani. 'Go with the winners' algorithms. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 492–501, 1994.

[Applegate et al., 1995] D. L. Applegate, R. Bixby, V. Chvatal, and W. Cook. Finding cuts in the TSP (a preliminary report). Technical Report No. 95-05, Center for Discrete Mathematics and Theoretical Computer Science, 1995.

[Applegate, 1995] D. L. Applegate, July 1995. Personal Communication.

[Baum, 1986a] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. Manuscript, 1986.

[Baum, 1986b] E. B. Baum. Towards practical 'neural' computation for combinatorial optimization problems. In *Neural Networks for Computing, AIP Conference Proceedings*, page 151, 1986.

[Bentley, 1990] J. L. Bentley. K-d trees for semidynamic point sets. In *Proc. ACM Symp. on Computational Geometry*, pages 187–197, June 1990.

[Bentley, 1992] J. L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[Boese, 1995] K. D. Boese. Cost versus distance in the traveling salesman problem. Technical Report TR-950018, UCLA CS Department, 1995.

[Boese, Kahng and Muddu, 1994] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 1994.

[Fogel, Owens and Walsh, 1966] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution.* John Wiley, 1966.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.

[Glover, 1977] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[Glover, 1996] F. Glover. Tabu search and adaptive memory programming – advances, applications and challenges. In R. Barr, R. Helgason, and J. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, 1996.

[Glover and Laguna, 1993] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–141. Blackwell Scientific Publishing, Oxford, 1993.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[Johnson and McGeoch, 1997] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search Algorithms.* Wiley and Sons, New York, 1997.

[Johnson, 1990] D. S. Johnson. Local optimization and the traveling salesman problem. In *Proc. 17th Intl. Colloquium on Automata, Languages and Programming*, pages 446–460, 1990.

[Lawler et al., 1985] E. L. Lawler, J. K. Lenstra, A. Rinnooy-Kan, and D. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley, Chichester, 1985.

[Lin and Kernighan, 1973] S. Lin and B. W. Kernighan. An effective heuristics algorithm for the traveling-salesman problem. *Operations Research*, 31:498–516, 1973.

[Martin and Otto, 1996] O. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. In G. Laporte, I. H. Osman, and P. L. Hammer, editors, *Annals of Operations Research*, volume 63, pages 57–75. 1996.

[Martin, Otto and Felten, 1991] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, June 1991.

[Martin, Otto and Felten, 1992] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224, 1992.

[Mühlenbein, Georges-Schleuter and Krämer, 1988] H. Mühlenbein, M. Georges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[Ulder et al., 1990] N. L. J. Ulder, E. H. L. Aarts, H.-J. Bandelt, P. J. M. van Laarhoven, and E. Pesch. Genetic local search algorithms for the traveling salesman problem. In *Proc. Parallel Problem Solving from Nature*, pages 109–116, 1990.