A Novel Local Search Algorithm for the Traveling Salesman Problem that Exploits Backbones*

Weixiong Zhang and Moshe Looks
Department of Computer Science and Engineering
Washington University in Saint Louis
Saint Louis, MO 63130, USA
Email: zhang@cse.wustl.edu

Abstract

We present and investigate a new method for the Traveling Salesman Problem (TSP) that incorporates backbone information into the well known and widely applied Lin-Kernighan (LK) local search family of algorithms for the problem. We consider how heuristic backbone information can be obtained and develop methods to make biased local perturbations in the LK algorithm and its variants by exploiting heuristic backbone information to improve their efficacy. We present extensive experimental results, using large instances from the TSP Challenge suite and real-world instances in TSPLIB, showing the significant improvement that the new method can provide over the original algorithms.

1 Introduction

Given a set of cities and the distances between them, the traveling salesman problem (TSP) is to find a complete, minimal-cost tour visiting each city once. The TSP is a well-known NP-hard problem with many real-world applications, such as jobshop scheduling and VLSI routing [5]. The TSP has often served as a touchstone for new problem-solving strategies and algorithms; and many well-known combinatorial algorithms were first developed for the TSP, including the Lin-Kernighan local search algorithm [10]. In this paper, we consider the symmetric TSP, where the distance from a city to another is the same as the distance in the opposite direction.

Local search (LS) algorithms have been shown very effective for the TSP. In LS algorithms for the TSP, one defines a neighborhood structure of tours, in which tour T is a neighbor of tour T' if T can be changed into T' by some local perturbation, such as by exchanging a pair (2-Opt) or triplet (3-Opt) of edges between cities [9; 10]. Starting from a complete tour, LS repeatedly improves the current tour until it is the best among its neighbors; such a tour is called a local minimum. This process can be applied multiple times using different initial tours. Although LS algorithms do not guarantee the optimality of the best solution found, they have been routinely applied in practice, and are among the best methods for the TSP. The best known LS algorithm for the TSP is the Lin-Kernighan (LK) algorithm [10]. Since its inception three decades ago, this classic algorithm has inspired much research on LS. Today, the best LS algorithms for the TSP are its variants, which include chained and iterated LK algorithms [11; 9; 6]. These algorithms can provide high-quality, near optimal solutions for problems with several thousand cities [9]. It is nontrivial to find solutions better than those from the Lin-Kernighan family. Nonetheless, improving these algorithms is of practical importance; even small improvements can have substantial financial impacts for many applications, as in manufacturing, where selected TSP tours need to be routinely traversed.

A major deviation of most modern LS variants from the LK algorithm is the use of starting tours that are closely related to the previous local minimum or the best tour found so far, rather than independently generated [9]. Such starting tours can be typically generated by perturbations to the final tours of previous runs using neighborhood structures different from the ones used by the main procedure. For instance, double-bridge 4-Opt moves have been used extensively for this purpose [9].

The intuition behind using chained starting tours in an iterated LS is that high-quality local minima tend to reside in a small vicinity of a neighborhood structure. Therefore, it is more effective and efficient to search for a better local minimum from a known one. This intuition is supported by the observation that local minima of a good LS method usually share many common partial structures [1; 2; 10]. Such observations have also led to the "big valley" hypothesis [1], which suggests that high-quality local minima tend to have many edges in common, forming a single cluster around the optimal tour(s), and that a better local minimum tends to have more common edges shared with an optimal solution than a worse local minimum.

Besides the iterated LS methods, common structures of local minima have been exploited in at least two other ways. The first is reduction [10], which first collects a small set of local minima and then locks in the edges appearing in all of them in the subsequent runs. This method has two effects: it can speed up the search, as the problem becomes smaller, and provides a means of directing search among a set of otherwise indistinguishable tours [10]. In [10], Lin and Kernighan reported experimental results on instances up to 318 cities using reduction. The second and more recent way to exploit common structures of local minima is called tour merging [2]. Given a set of local minima, this method constructs a graph containing the union of their edges. Thanks to the large number of shared edges among the local minima, the resulting graph is sparse. An optimal tour within the sparse graph is then uncovered as

*The research was supported NSF grant EIA-0113618.

an approximate solution, which is very often optimal. Our recent algorithm on maximum satisfiability (max-SAT) also explicitly exploited the information in a cluster of local minima and incorporated it in the Walksat algorithm [15], resulting in a significant performance improvement on diverse large max-SAT problems [16]. The current research was inspired by these previous results.

Our research was also motivated by the recent advances in characterizing typical case features of combinatorial problems by their phase transitions and backbones [7; 12]. A problem of fundamental importance and practical interest is to utilize inherent problem information, such as phase transitions and backbones, in a search algorithm to cope with problem difficulty. The research along this line is limited. Besides the published work of utilizing backbone information in local search for SAT and max-SAT [16], previously published results include exploiting phase transitions in tree search problems in an approximation algorithm [13] and applying heuristic backbone information in a systematic search for SAT [3].

Our new method explicitly exploits the structure of the local minima of LS algorithms, namely the possible backbone information embedded in the local minima, to improve the performance of the algorithm. Here, a backbone variable for a TSP refers to an edge between two cities that appears in all optimal TSP tours. Unlike the reduction and tour merging methods, our new heuristic does not freeze the common edges in all local minima in subsequent searches; it rather applies estimated backbone information to guide a local search to the region of the search space that is more likely to have better approximate, and hopefully optimal, solutions. Specifically, we treat local minima from a local search as if they were optimal solutions, and use edge appearance frequencies to estimate the probabilities of backbone variables. We then apply the estimated backbone probabilities to alter the perturbations made by the local search algorithm so that a variable having a higher backbone probability will be less likely to be swapped out of the current tour than a variable having a smaller backbone probability, and conversely, will be more likely to be swapped in.

The paper is organized as follows: in Section 2, we describe the general idea of backbone-guided local search for the TSP. We then in Section 3 discuss the backbone-guided LK algorithms. We present the experimental results in Section 4, discuss related work in Section 5, and conclude in Section 6.

2 Backbone Guided Local Search

If all backbone variables of a TSP were known, they could provide a useful clue to how edges between two cities should be swapped in or out during a local search. If an edge is a part of the backbone, i.e., it appears in all optimal solutions, obviously the edge should be swapped in if it is not included in the current tour. Moreover, we can extend the concept of backbone to backbone frequency of an edge, which is the percentage of optimal solutions that have the edge. This means that a backbone edge has a backbone frequency of one and an edge that does not appear in any optimal solutions has a backbone frequency of zero. Therefore, the backbone frequency of an edge is an indicator of how often that edge should be swapped in (or out) if it is (or it is not) part of the current tour. This can be exploited as a heuristic for selecting edges in local search.

Unfortunately, exact backbone frequencies are hard to come by without solving the problem exactly. The idea to bypass this problem was inspired by the fact that many approximation algorithms for the TSP, the LK algorithm and its variants in particular, have superior performance. They can often reach local minima that are within a small percentage of a global optimum and have common structures shared with a global optimal solution, as discussed in [1; 9]. Therefore, we can treat local minima as if they were optimal to compute *pseudo-backbone frequencies* to approximate the true backbone frequencies. We call this general approach *backbone guided local search* or BGLS.

We define the pseudo-backbone frequency of an edge as the frequency of its appearances in the local minima sampled. Thus, if $\mathcal{S}$ is the set of local minima, and a given edge e appears in a subset $\mathcal{S}_e$ of $\mathcal{S}$, the pseudo-backbone frequency of e is simply $|\mathcal{S}_e|/|\mathcal{S}|$.

Which local minima to use will affect the quality of the pseudo-backbone frequencies. Ideally, we want the local minima to be an unbiased sample of all high-quality approximate solutions. One leading factor in reaching such an ideal is the set of initial tours: the more distinct the starting tours are, the more different the final tours will generally be. Therefore, even though local minima reached from greedily generated starting tours are superior to those reached from random starting tours, a pseudo-backbone constructed from the latter generally leads to better overall performance.

Pseudo-backbone information can be incorporated in LS algorithms to "bias" the search. In LS, moves are evaluated by the difference between the total cost of the edges to be removed from the tour and the total cost of the edges to be added. If this value is positive, the move is taken. In backbone guided search, we can make biased moves in two different ways; one only uses pseudo-backbone frequencies and the other combines pseudo-backbone frequencies and the distances between cities. Let $\mathcal{B}$ be the set of backbone edges of a TSP and $\mathcal{T}$ the set of local minima from which pseudo-backbone frequencies were computed. Let X and Y be the candidate set of edges to be removed and added, respectively, at a step of searching for a k-Opt move. We prefer to replace X by Y if X has a smaller possibility of having more backbone variables than Y. If we assume edges to be independent of each other, we prefer to replace X by Y if $\sum_{x_i \in X} P(x_i \in \mathcal{B}|\mathcal{T}) < \sum_{y_i \in Y} P(y_i \in \mathcal{B}|\mathcal{T})$, where $P(x_i \in \mathcal{B}|\mathcal{T})$ is the backbone frequency of edge x_i, computed from the set of local minima $\mathcal{T}$. This method has been shown effective on maximum satisfiability [16].

Unfortunately, local search based exclusively on local perturbations using merely pseudo-backbone frequencies is not very effective on the TSP. One possible factor contributing to this discrepancy between the TSP and max-SAT is the different sizes of their search spaces. The TSP has $2^{|E|} = O(2^{n^2})$ states in its search space, where $|E| = n(n-1)/2$ is the number of edges for an n city TSP. These states are embedded in a constraint structure, in which, for example, taking one edge preventing many other edges from being taken. In comparison, the search space of max-SAT has only 2^n states for n Boolean variables. The estimated backbone frequencies could thus be less reliable on the TSP than on max-SAT. The deficiency of local perturbations based purely on pseudo-backbone frequencies indicates that the actual intercity distances should not be ignored for the TSP. This constitutes one of the main differences between backbone-guided local search for the TSP and that for max-SAT.

In the LK algorithm, moves are evaluated by summing the costs of the edges to be removed from the current tour, and subtracting the sum of the costs of the edges to be added. The same evaluation principle can be applied in backbone guided LS, but with cost computed differently. Instead of taking the cost of a given edge to be the distance between two cities, w, it is taken to be $w \cdot (1 - p)$, where p is the pseudo-backbone frequency of that edge. Thus, the cost of an edge will decrease linearly in proportion to its pseudo-backbone frequency. The extreme cases are edges not in the pseudo-backbone, which have their original costs ($p = 0$), and edges in all local minima, which their costs set to zero ($p = 1$).

When this method is used, the results can be improved even more by carrying out regular LK search using original distances from the local minima found by the biased search. This is effective because the biased search actually searches in a new TSP instance that has been created from the original by applying a "pseudo-backbone transform" to its edge weights. Local minima in this new instance will not generally correspond to local minima in the original, so continued search of the original graph will improve the results, even after biased search has reached a local minimum.

Furthermore, pseudo-backbone information can be employed to generate starting tours. It is known that the effectiveness and speed of local search can be improved by using greedy starting tours [9]. The greedy tour construction begins by randomly picking a starting city, and adding the shortest edge exiting the city to the tour. Then, edges are greedily added one-by-one until the tour is complete. We can modify this process to naturally utilize the pseudo-backbone by redefining the "best" edge in terms of the pseudo-backbone-transformed weights, rather than the original weights. This gives us a greedy pseudo-backbone tour generation heuristic.

3 Backbone Guided LK and ILK

Applying the above ideas and considerations to LK, we have the backbone guided LK algorithm (BGLK). Similar to LK, BGLK runs many cycles, each of which starts from a new starting tour and reaches a local minimum. Different from LK, BGLK has two phases. The first is a learning phase that runs a fixed number of iterations of original LK with *random* starting tours. The local minima from these runs are used to compute pseudo-backbone frequencies. The second is a backbone-guided improvement phase where biased k-Opts are utilized. In addition, in the second phase, biased starting tours can be used as well to improve local minima and speed up the search. In our experiments, we found that setting aside 30% of the total runs for learning was generally effective.

A few issues must be dealt with to combine pseudo-backbone utilization with ILK, the iterated LK algorithm, in deriving iterative BGLK (IBGLK). First, pseudo-backbone must be constructed from "unbiased samples" of local minima. Since each local minimum found via ILK typically differs in only a small number of edges from its progenitor, a sampling of local minima from such a chained process is biased, and should not be averaged into the backbone. To deal with this, we construct a backbone in our experiments from 30 independent rounds, i.e., restarts, of ILK, each of which is allowed a smaller number of iterations, such as 1% of the total number of iterations. Another issue is that we would like to follow each round of BGLK by a round of regular search using the original distances. However, since BGLK uses "weighted" costs, it is impossible to ignore potential k-Opts involving only unchanged cities, as when moving between successive rounds of ILK with only a single search method. Our solution to this has been to alternate between BGLK and regular LK at a higher level of granularity: one of these search mechanisms is employed until it "fails", at which point we begin employing the alternate mechanism until it too "fails", then switch back to the original, and the alternation repeats. In this context, we have defined failure at going through k rounds (restarts) of optimization without finding an improved tour. For our experiments, we have set $k = N/20$, where N is the number of cities.

4 Experimental Evaluation

We have carried out two sets of experiments to study different ways of applying pseudo-backbone information in BGLS. In our experiments we used random instances from the TSP Challenge suite [8], which includes problem classes of uniform Euclidean (Uniform), clustered Euclidean (Clustered), and distance matrix (Matrix). We have also used the large instances from the TSPLIB [14].

We compared BGLK and IBGLK against LK, ILK, the reduction method of [10], and the search space smoothing method of [4]. The smoothing method is related because it modifies the distances of TSP instances in an attempt to make it easier for local search to eventually reach local minima of higher quality. It uses a fixed formula, $d' = \bar{d} \pm |d - \bar{d}|^{\alpha}$, to transform distance from d to d', where $\bar{d}$ is the average distance over all original distances, and α takes a series of decreasing values, typically $5, 4, \cdots, 1$ to help slowly move from an initially "smooth" instance, in which all distances look similar to one another, to the actual problem. We implemented this method in the LK algorithm for the test.

The particular version of LK-based algorithms that we used was implemented and provided by Johnson and McGeoch, described and analyzed in [9]. We leave the details of the implementation to its original description, while simply pointing out that all of our tests were carried out with the default settings for this implementation, namely length-20 neighbor lists for all levels of the search, don't-look bits, and the 2-Level Tree tour representation. We incorporated the reduction method and the space smoothing method in LK to generate two variants. We have also followed Johnson and McGeoch [9] in configuring ILK, allowing it the most flips, $10N$, where N is the problem size. This is our baseline ILK algorithm, abbreviated as *ILK-1-run*. In addition, we used two different configurations of ILK in our experiments: the best of five runs of $2N$ iterations (i.e., ILK with four random restarts), which is named as *ILK-5-runs*, and the best of ten runs of N iterations, which is called *ILK-10-runs*. We configured IBGLK as follows. It first runs 30 rounds of ILK, each of which starts from a random tour and is allowed $N/10$ iterations. It then executes 70 rounds of ILK with biased moves and biased greedy initial tours, each of which is also allocated $N/10$ iterations. All algorithms used the same total number of flips to give a relatively fair comparison.

Our experiments were run on 1.6 and 2.0 GHz AMD Athelon machines with 2 gigabytes of memory. All runtimes are normalized for a 500 Mhz Alpha, as described in [9], so that our results and the previous results can be directly compared.

4.1 The TSP Challenge suite

We experimentally validate our claim that BGLS allows local search methods to reach better local minima, using problem

		N=1000		N=3162		N=10K		N=31K	
U	Opti	0.74	—	0.70	—	—	—	—	—
	LK	1.38	(8.8)	1.56	(33.0)	1.79	(83.2)	1.83	(232.9)
	Redu	1.30	(5.2)	1.59	(18.1)	1.80	(43.8)	1.84	(117.0)
	Smoo	1.21	—	1.46	—	1.58	—	1.55	—
	BGLK	**1.07**	(12.1)	**1.15**	(53.4)	**1.2**	(147.0)	**1.19**	(441.2)
C	Opti	0.54	—	0.59	—	—	—	—	—
	LK	1.81	(226.9)	3.42	(625.0)	5.67	(1241.7)	5.63	(3013.3)
	Redu	1.51	(103.8)	3.36	(290.7)	5.19	(494.9)	6.01	(1132.8)
	Smoo	2.16	—	3.70	—	6.08	—	6.08	—
	BGLK	**1.30**	(322.3)	**2.54**	(1032.3)	**4.90**	(2120.8)	**4.58**	(5238.7)
M	Opti	0.02	—	0.00	—	0.00	—	—	—
	LK	2.45	(19.9)	3.87	(63.1)	5.30	(339.3)	—	—
	Redu	2.24	(16.0)	3.72	(49.1)	5.19	(270.3)	—	—
	Smoo	1.60	—	2.79	—	4.27	—	—	—
	BGLK	**0.90**	(30.9)	**1.77**	(79.0)	**2.58**	(405.0)	—	—

Table 1: Comparison of LK, reduction, smoothing and BGLK on problems from the Challenge suite on Uniform (U), Clustered (C) and matrix (M) problem instances. The numbers are tour costs over Held-Karp bounds in %, and normalized CPU times in seconds (in parentheses).

		N=1000		N=3162		N=10K		N=31K	
U	Optimal	0.74	—	0.70	—	—	—	—	—
	ILK-1-run	0.82	(58)	0.78	(294)	0.81	(1619)	0.89	(8371)
	ILK-5-run	**0.77**	—	0.80	—	0.80	—	0.85	—
	ILK-10-run	0.78	—	0.82	—	0.86	—	0.98	—
	IBGLK	**0.77**	(78)	**0.77**	(411)	**0.77**	(2311)	**0.77**	(11760)
C	Optimal	0.54	—	0.59	—	—	—	—	—
	ILK-1-run	0.55	(1469)	**0.62**	(6645)	0.84	(22859)	1.14	(88077)
	ILK-5-run	**0.54**	—	**0.62**	—	0.76	—	1.06	—
	ILK-10-run	**0.54**	—	0.63	—	**0.73**	—	1.01	—
	IBGLK	0.55	(1281)	**0.62**	(5764)	0.78	(23311)	**0.94**	(93759)
M	Optimal	0.02	—	0.00	—	0.00	—	—	—
	ILK-1-run	0.58	(106)	1.34	(860)	2.70	(8230)	—	—
	ILK-5-run	0.85	—	1.34	—	2.09	—	—	—
	ILK-10-run	0.89	—	1.94	—	3.44	—	—	—
	IBGLK	**0.20**	(139)	**0.79**	(1221)	**1.48**	(11360)	—	—

Table 2: Comparison of ILK (ILK-1-run), its variations (ILK-5-run and ILK-10-run), and IBGLK on problems from the Challenge suite. The legend and interpretation of the table are the same as those for Table 1.

instances from the Challenge suite. We examine the results from four different perspectives.

In the first test, we compared BGLK against LK, LK with reduction, and LK with search space smoothing. The LK algorithm has 100 runs; the reduction method uses an initial probing stage of 30-run of LK, followed by 70-run of LK with reduction; the smoothing method has 20 runs total, each of which runs LK five times, for $\alpha = 5, 4, \cdots, 1$. These algorithms all used greedy initial tours and had a total of 100 runs. BGLK executes an initial 30-run of LK, using random starting tours, to learn pseudo-backbone, and then 70 runs of LK with biased moves and biased initial tours. The sizes N of problem instances are 1000, 3162, 10000, and 31623, increased by a factor of $\sqrt{10}$, following the scheme proposed and used in [9]. For each problem class and size, 100 random problem instances were used for each algorithm. Table 1 shows the averaged results. For comparison, percent over the Held-Karp bounds for optimal tours is listed when known.

As shown, while the reduction and smoothing methods provide modest improvements over LK in certain instance classes and sizes, they are respectively the worst among all algorithms tested on the Euclidean and clustered Euclidean classes. In contrast, BGLK is consistently superior, outperforming all the other three methods across all instance classes and sizes. The improvement of BGLK over these algorithms varies from 0.3% to 1.7% on the largest problem instances of these classes. Note that the average running time of BGLK was no more than twice the time by LK. Interestingly, BGLK actually ran *faster* than LK on some instances, presumably due to its faster focus on promising areas. Note also that the reduction method's running times significantly less than that of IL, while the smoothing method's runtimes are essentially identical to LK's.

In the second experiment, we compared IBGLK against ILK (ILK-1-run) and its two variants (ILK-5-runs and ILK-10-runs). We used the same problem instances as in the first experiment, and computed the results the same way as in that experiment. The results are shown in Table 2, where we adopted the same reporting scheme as in Table 1. As shown, the two ILK variations do not provide much improvement over the baseline ILK, while IBGLK is able to push significantly closer to optimal than ILK for most cases where ILK does not perform very well. For uniform Euclidean instances, as ILK does not reach the optimum for these instances, IBGLK has a good chance to improve tour quality. IBGLK's performance appears to remain constant (i.e., 0.77%) relative to the Held-Karp bound as the problem size increases, while ILK's performance degrades. Although IBGLK runs longer than ILK, due to the extra time needed to construct and apply the pseudo-backbone, the overhead never exceeds 50% of ILK's runtime, and it appears that IBGLK's and ILK's runtimes are asymptotically close to each other. For clustered problem class, IBGLK, ILK and its variations' performances are compatible with one another. Interestingly, IBGLK actually runs *faster* than ILK on some of the clustered instances. For this instance class, IBGLS's lack of significant improvement over ILK can be attributed to the fact that ILK is already effective and close to the optimum; there is little room for improvement. On the distance matrix class, similar to BGLK versus LK, IBGLK is significantly superior to ILK on each of the individual instances of different sizes, and reduces average tour costs by more than 0.6% on the largest ($N = 10,000$) instances tested. In addition, among ILK and its variants, ILK-1-run has the fastest running times, and its two variants are 5%-10% slower.

In the third experiment, we considered relatively small instances of 1,000 cities and compared the local minima from LK, ILK and IBGLK against the optimal solutions. We used 100 instances for each of the problem classes in the Challenge suite. For each instances and each algorithm, we averaged three runs with different random seeds. As the result shows, IBGLK generally produces solutions of higher quality on average. Interestingly, it seems that more powerful search methods produce not only better, but more diverse solutions, as quantified by the standard deviations of the Hamming distances between the structures of local minima and the optimal solutions (data not shown). Figure 1 graphically shows relative performances of ILK and IBGLK on individual instances in the test, indicating that IBGLK is favorable to ILK on most of the problem instances.

Additional insight can be gained by an inspection of the anytime behavior of these algorithms. As shown in Figure 2, BGLK's anytime behavior is remarkably different from that of LK and its variants. To begin with, LK outpaces BGLK, since LK is being run from greedy starting tours, while BGLK must be run from random tours to construct a reliable pseudo-backbone (see Section 3). But once BGLK begins using the pseudo-backbone to guide the search, its rate of descent increases, and the tables are turned on their performance. Qual-

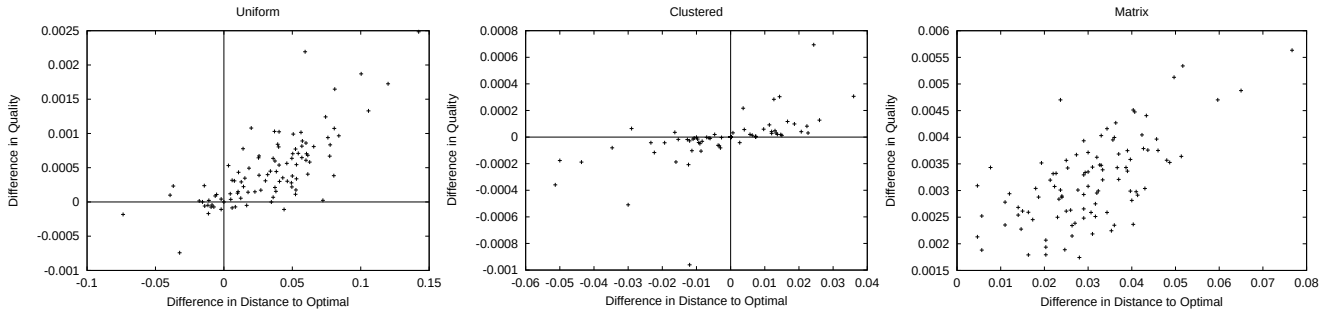

Figure 1: Comparison of local minima from ILK and IBGLK for 100 problems with 1,000 cities each. Points with $x > 0$ indicate that IBGLK's solutions are of higher quality, points with $y > 0$ indicate that IBGLK's solutions are closer to optimal. Both axes are normalized.

itatively similar anytime performance results have been found on large problem sizes and when comparing IBGLK with ILK.

4.2 Instances from the TSPLIB

Table 3 contains the results for ILK, its variants, and IBGLK, on all the instances in the TSPLIB that have at least 1,000 cities. On many of the instances, unfortunately, ILK is already at or near the optimum, so there is very little room for improvement. Again, in most places where further improvements are possible, IBGLK produces them. On this set of instances, IBGLK championed on 20 of them (the ones in bold and underlined), whereas ILK and its variants came first on 12 instances (with some overlaps). Note that on the largest instances (the ones with more than 10,000 cities), IBGLK produces significant gains over ILK. The runtimes were from our AMD Athelon machines and then normalized to a 500MHz Alpha as suggested in [9].

5 Related Methods and Discussions

The most closely related work is our previous work on backbone guided Walksat for maximum satisfiability [16]. The backbone guided local search developed here for the TSP follows the same principles developed there; and thus can be viewed as an innovative extension to the work in [16]. This research was also inspired by and builds upon the previous results of the "big valley" hypothesis on the clustering of local minima from the family of the Lin-Kernighan algorithm and its variants [1; 2; 10].

Two pieces of previous work resemble BGLK in some way. The first is the reduction method by Lin and Kernighan [10]. As discussed earlier, reduction "locks in" the common edges in all local minima. This has two effects: it can speed up the search, as the problem becomes smaller, and provides a means of directing search among a set of otherwise indistinguishable tours. The main limitation of this method is that it is brittle, depending on the quality of the "locked in" edges. If a "locked in" edge turns out not to be part of the backbone, no optimal tour will be found. Moreover, information such as "edge (a, b) appears in 90% of all local minima" cannot be utilized, thus it simply disregards potentially useful information of backbone frequencies. The results in [10], up to 318 cities, is too limited to provide a fair assessment of this method. Our results in Section 4 revealed that reduction is not competitive.

BGLS is more general than reduction by providing a remedy to the problem of brittleness and making use of information embedded in backbone frequencies. BGLS is a random strategy, in which edges that appear in all local minima may still be swapped out of the current tour during the search, albeit with a slim possibility. This allows more tours be explored while still maintaining a focused search.

	% over the Held-Karp bound					runtime in seconds	
Name	ILK-1	ILK-5	ILK-10	IBGLK	Optimal	ILK-1	IBGLK
dsj1000	**0.61**	0.62	0.62	0.62	0.61	2097	2066
pr1002	**0.89**	**0.89**	1.04	**0.89**	0.89	300	345
si1032	0.15	**0.08**	**0.08**	**0.08**	0.08	610	772
u1060	**0.65**	0.67	0.67	0.67	0.65	870	873
vm1084	1.37	**1.35**	**1.35**	**1.35**	1.33	403	403
pcb1173	0.97	0.98	0.97	**0.96**	0.96	161	213
d1291	1.31	1.23	**1.18**	1.26	1.18	1099	1086
rl1304	**1.55**	**1.55**	**1.55**	**1.55**	1.55	457	504
rl1323	1.66	1.66	**1.65**	**1.65**	1.65	470	582
nrw1379	0.47	0.51	**0.43**	0.49	0.43	274	440
fl1400	**1.74**	**1.74**	**1.74**	**1.74**	1.74	22548	19317
u1432	0.44	0.38	0.38	**0.29**	0.29	319	617
fl1577	1.68	**1.66**	**1.66**	**1.66**	1.66	9072	10430
d1655	1.19	0.92	0.95	**0.91**	0.91	1808	1869
vm1748	**1.35**	1.38	**1.35**	1.36	1.35	893	829
u1817	1.08	1.13	1.17	**1.06**	0.90	468	680
rl1889	1.74	1.64	**1.55**	1.77	1.55	1028	1110
d2103	**1.44**	**1.44**	**1.44**	**1.44**	1.44	3407	8669
u2152	0.96	**0.77**	0.79	0.81	0.62	597	808
u2319	0.13	0.13	0.13	**0.06**	0.02	1182	2091
pr2392	**1.27**	1.28	1.37	1.36	1.22	503	715
pcb3038	0.91	**0.88**	0.93	0.94	0.81	854	1178
fl3795	**1.04**	**1.04**	1.05	1.07	1.04	45789	52460
fnl4461	0.66	0.67	0.69	**0.63**	0.55	1613	2505
rl5915	**1.58**	1.62	**1.58**	1.62	1.56	2963	4766
rl5934	1.53	**1.39**	1.56	1.54	1.38	3714	5250
pla7397	**0.61**	0.72	0.63	0.63	0.58	18338	20716
rl11849	**1.13**	1.17	1.21	1.15	1.02	9421	15301
usa13509	0.78	0.76	0.83	**0.72**	0.66	20829	26339
brd14051	0.61	0.63	0.66	**0.58**	—	28587	35048
d15112	0.66	0.68	0.72	**0.63**	0.52	21481	29709
d18512	0.63	0.64	0.70	**0.59**	—	21995	31916
pla33810	0.68	0.66	0.70	**0.65**	—	148387	150455
pla85900	0.56	0.58	0.62	**0.52**	—	256808	247597
# of bests	12	11	12	**20**			

Table 3: Comparison of all instances in TSPLIB with at least 1,000 cities. Numbers are tour costs over Held-Karp bounds in % and normalized runtimes in seconds (for a 500 Mhz Alpha).

The second related work is search space smoothing [4], which was briefly described in Section 4. Gu and Huang only experimented with this idea with a 2-Opt algorithm and on small random Euclidean problems with no more than 100 cities. They did not compare their method to other techniques. Our results in Section 4 showed that smoothing outperforms LK on uniform and matrix instances. However, it is less efficient than BGLK and ILK.

Although both smoothing and backbone guided search all modify distances, they are fundamentally different. First, modifying distances is just one of the end products of the backbone guided search for the TSP. In fact, when applied to maximum satisfiability, backbone guided local search did not change

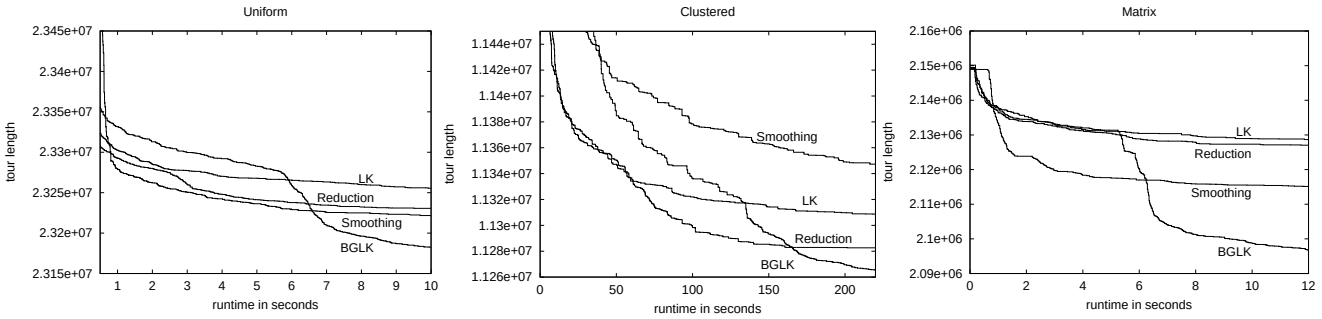

Figure 2: Anytime performance of LK, LK with reduction, LK with smoothing, and BGLK, for random instance from the TSP Challenge suite with 1,000 cities.

costs at all [16]. Estimated backbone information can be used not only to modify distances, but also in different places of a local search to make biased searches. For instance, it can be employed to generate biased initial starting tours. Secondly and more importantly, the backbone guided search does not treat every distance equally; some distances may be altered dramatically while others may remain intact to help exploit the intricate constraints among the cities.

6 Conclusion

We have presented and investigated a new method of applying backbone information to force a local search to make biased local perturbations, in the context of the TSP. Based on the "big valley" hypothesis, we developed a method for deriving heuristic backbone information by exploiting the local minima from the efficient Lin-Kernighan local search algorithm. We demonstrated its effectiveness with extensive experiments on various problem types and instances.

Our results on the TSP show that BGLK and IBGLK compare very favorably to LK and ILK, respectively, on the instances in the Challenge suite, up to 31,623 cities, and the instances from TSPLIB. For the uniform Euclidean class, BGLK and IBGLK outperform LK (and its variants) and ILK, respectively. On clustered Euclidean instances, BGLK outperforms LK by reducing tour costs by more than 1% on average on 31,623-city instances; and IBGLK and ILK have comparable performance. (Note that an improvement of a fraction of percent in solution quality is significant on large TSPs [9].) BGLK and IBGLK dominate their counterparts on tour quality on all instances in the distance matrix class we tested; on 10,000-city instances, BGLK and IBGLK find tours with costs at least 1.5% and 0.5% smaller than LK and ILK, respectively. On real instances from the TSPLIB, IBGLK finds nearly twice as many best solutions as three versions of ILK (i.e., 20 versus 12). Finally, BGLK and IBGLK have comparable running times with their counterparts, no more than twice as long as the original algorithms, while sometimes being faster.

In short, the main contribution of this work is an effective method of utilizing inherent problem features, backbones in particular, of the TSP in the LK algorithms and variants to improve their efficacy. A method of exploiting problem features in search is of fundamental interest and practical importance. The results on the TSP in this paper have indicated that the idea of backbone-guided local search is general and can lead to efficient heuristic algorithms for large, difficult problems. The source code of our algorithm will be available on the web.

References

[1] K. D. Boese. *Models for Iterative Global Optimization*. PhD thesis, UCLA/Computer Science Department, 1996.

[2] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS J. of Computing*, to appear.

[3] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formula. In *Proc. of IJCAI-01*, pages 248–53, 2001.

[4] J. Gu and X. Huang. Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). *IEEE Trans. SMC*, 24:728–735, 1994.

[5] G. Gutin and A.P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer, 2002.

[6] K. Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European J. Operations Research*, 126:106–130, 2000.

[7] T. Hogg, B.A. Huberman, and C. Williams. Phase transitions and the search problem. *Artificial Intelligence*, 81:1–15, 1996.

[8] D. S. Johnson. 8th dimacs implementation challenge: The TSP. http://www.research.att.com/∼dsj/chtsp.

[9] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer, 2002.

[10] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the Traveling Salesman Problem. *Operations Research*, 21:498–516, 1973.

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

[12] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.

[13] J.C. Pemberton and W. Zhang. Epsilon-transformation: Exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81:297–325, 1996.

[14] G. Reinelt. TSPLIB - a traveling salesman problem library. *ORSA J. Computing*, 3:376–384, 1991.

[15] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proc. of AAAI-94*, pages 337–43, 1994.

[16] W. Zhang. Configuration landscape analysis and backbone guided local search: Part I: satisfiability and maximum satisfiability. *Artificial Intelligence*, 158(1):1–26, 2004.