Efficient Local Search With Search Space Smoothing: A Case Study of the Traveling Salesman Problem (TSP)

Jun Gu, *Senior Member, IEEE*, and Xiaofei Huang

***Abstract*—Local search is very efficient to solve combinatorial optimization problems. Due to the rugged terrain surface of the search space, it often gets stuck at a locally optimum configuration. In this paper, we give a local search method with a search space smoothing technique. It is capable of smoothing the rugged terrain surface of the search space. Any conventional heuristic search algorithm can be used in conjunction with this smoothing method. In a parameter space, by altering the shape of the objective function, the original problem instance is transformed into a series of gradually simplified problem instances with smoother terrain surfaces. Using an existing local search algorithm, an instance with the simplest terrain structure is solved first, the original problem instance with more complicated terrain structure is solved last, and the solutions of the simplified problem instances are used to guide the search of more complicated ones. A case study of using such technique to solve the traveling salesman problem (TSP) is described. We tested this method with numerous randomly generated TSP instances. We found that it has significantly improved the performance of existing heuristic search algorithms.**

***Index Terms*—Local search, search space, search space smoothing, the Traveling Salesman Problem (TSP).**

I. Introduction

The Traveling Salesman Problem (TSP) is a well-known NP-hard combinatorial optimization problem [14]. It can be stated as: A salesman tries to find the shortest closed route to visit a set of cities under the conditions that each city is visited exactly once. The distances between any pair of cities are assumed to be known by the salesman.

Up to now, there seems no way to find an algorithm which can, in general, find out an optimal solution for the TSP without suffering from exponentially growing complexity. As a result researchers in the area have developed many heuristic algorithms to solve the problem. Generally, the performance of a heuristic algorithm is measured in terms of the "closeness" between the solution it produced and an optimal solution under certain conditions.

Many local search algorithms have been developed to solve the TSP. Although local search is very efficient, due to the rugged terrain surface of the search space, a local search often gets stuck at a locally optimum configuration, i.e., a local minima. To reduce the effect of local minima points, in this paper, we introduce a local search method with search space smoothing. The basic idea of the method is simple. Given a TSP instance in a parameter space, a search space smoothing technique transforms the given problem into a series of problem instances with different terrain structures. Initially, a simplified TSP instance with a smooth terrain surface is solved which produces a solution route. Then, a more complicated TSP instance which has a rougher terrain surface is generated. It takes the solution of the previously solved TSP as an initial route and further improves the route. Eventually, the original TSP instance with the most complicated search space structure is solved. Any existing heuristic local search algorithm can be used in conjunction with this approach. We tested the performance of this method with numerous randomly generated TSP instances. We found that it has significantly improved the performance of the existing heuristic search algorithms, producing shorter routes.

The rest of the paper is organized as follows: In the next section, we will briefly review the previous work in the area. Section III briefly introduces local search and its search space. In Section IV, we describe the basic idea of search space smoothing. In Section V, a local search algorithm with a search space smoothing processing for TSP is given. In Section VI, we show the experimental results of this algorithm and its performance comparisons with other heuristic search algorithms. Finally, Section VII concludes this paper.

II. Conventional Heuristic Approaches

There are many heuristic algorithms developed to solve the TSP. Golden and Stewart gave a detailed review and empirical analysis of some TSP heuristic algorithms [3]. In this section, we first briefly review some heuristic algorithms for TSP.

The conventional heuristic algorithms fall into three categories: tour construction, tour improvement, and a composite approach with tour construction and tour improvement.

In the *tour construction* approach, we start from an initial subtour and try to extend this subtour into a complete one which is an approximately optimal tour. Usually, the initial subtour is simply a randomly chosen city or a self-loop. Some representative algorithms in this category include arbitrary insertion [21], convex hull insertion

Manuscript received April 20, 1992; revised May 29, 1993. This work was supported in part by an NSERC Strategic Grant MEF0045793 and is presently supported in part by NSERC Research Grant OGP0046423. This work was present in part in Technical Report UCECE-TR-91-006.

The authors are with the Department of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta, T2N 1N4, Canada.

IEEE Log Number 9400631.

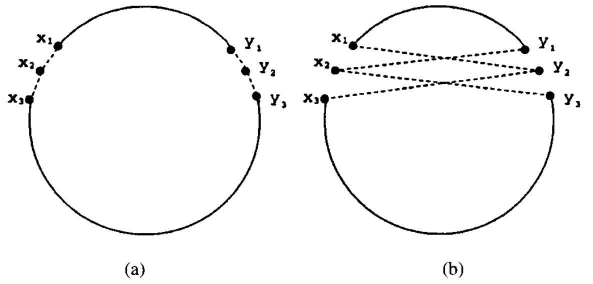


Fig. 1. Illustration of the route configuration changes using the city-swap heuristic. (a) Before swaps. (b) After swaps: $d(x_2, y_1) + d(x_2, y_3) + d(x_1, y_2) + d(x_3, y_2) < d(x_1, x_2) + d(x_2, x_3) + d(y_1, y_2) + d(y_2, y_3)$.

[25], greatest angle insertion [17], [18], and ratio times difference insertion [19].

In the *tour improvement* approach, usually, we start from an initial complete route and try to reduce the length of the route as long as the route remains complete. A frequently used procedure of tour improvement is edge exchange [4], [15], [16]. A recent new development in this direction includes using the stochastic method to solve the TSP [1], [13], [22],

The third approach is a composite approach of tour construction and tour improvement. The tour construction procedure provides an initial, complete route and the tour improvement procedure further improves the initial route.

In the rest of this paper, we will focus on the discussion of the edge exchange (tour improvement) techniques since they are more effective methods. A straightforward method in tour improvement is trying to swap two cities on a route. If the swap reduces the total length, simply keep it; otherwise, try another swap. This is so called the *city swap heuristic*. Fig. 1 illustrates the tour configuration changes using the city-swap heuristic.

Although the city-swap heuristic is simple and straightforward, empirically, it is much less effective than other tour improvement heuristics. The best known and more effective heuristics are r-opt ($r = 2, 3, \ldots$) [16] and Or-opt [19] procedures. Number "r" in the r-opt algorithm is referred as the number of edges to be exchanged. Fig. 2 illustrates an example of the 2-opt heuristic. The r-opt procedure works as follows:

Step 1 Delete r edges from a route and add r new edges from the other parts of the route as long as the result remains as a complete route. If this exchange results in a shorter route, keep the exchange; otherwise, try other exchanges by deleting/adding the different edges.

Step 2 Repeat Step 1 as long as there are improvements after some exchanges. If no more improvement can be made by all the possible exchanges, i.e., a local minima point is met, then exit and output the final route as a result.

A large r leads to more powerful heuristics which may provide solutions closer to the optimal solutions. On the other hand, a large r requires the testing of a large number of possible exchanges which increases the computing time dramatically.

The Or-opt procedure is a modified 3-opt procedure. It improves the r-opt heuristic by reducing the number of

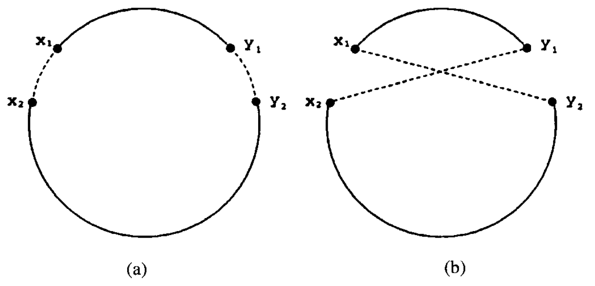


Fig. 2. Illustration of the route configuration change using the 2-opt heuristic. (a) Before exchanges. (b) After exchanges: $d(x_1, y_2) + d(x_2, y_1) < d(x_1, x_2) + d(y_1, y_2)$.

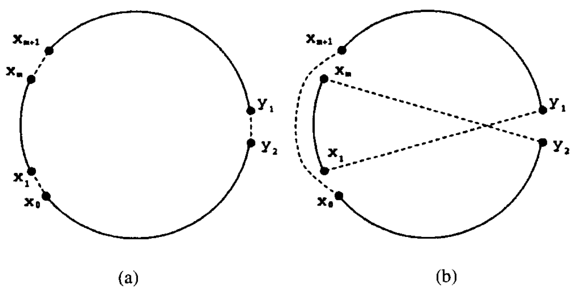


Fig. 3. Illustration of the route configuration changes using the Or-opt heuristic. (a) Before changes. (b) After changes: $d(x_0, x_{m+1}) + d(x_1, y_1) + d(x_m, y_2) < d(x_0, x_1) + d(x_m, x_{m+1}) + d(y_1, y_2)$.

exchanges to be tested. It considers those exchanges which are obtained by cutting a piece of route and insert it between two other cities, as illustrated in Fig. 3.

In the next section, we describe briefly the local search approach and the local minima problem associated with its search space.

III. Local Search and the Search Space

Local search was one of the early techniques proposed during the mid-sixties to cope with the overwhelming computational intractability of NP-hard combinatorial optimization problems. Given a minimization (maximization) problem with object function f and feasible region F, a typical local search algorithm requires that, with each solution point $x_i \in R$, there is associated a predefined *neighborhood* $N(x_i) \subset R$. (Note that in the areas of artificial intelligence and operations research, a point x_i in a local search framework is called a *solution point* even if it is not a *final* solution point of the given problem.) Given a current solution point $x_i \in R$, the set $N(x_i)$ is searched for a point x_{i+1} with $f(x_{i+1}) < f(x_i)$ ($f(x_{i+1}) > f(x_i)$). If such a point exists, it becomes the new current solution point, and the process is iterated. Otherwise, x_i is retained as a *local optimum* with respect to $N(x_i)$. Then, a set of feasible solution points is generated, and each of them is "locally" improved within its neighborhood. To apply local search to a particular problem, one needs only to specify the neighborhood structure and the randomized procedure for obtaining a feasible starting solution point.

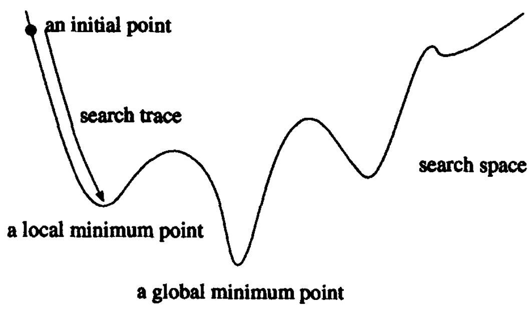


Fig. 4. An example of a simplified, 1-dimensional search space.

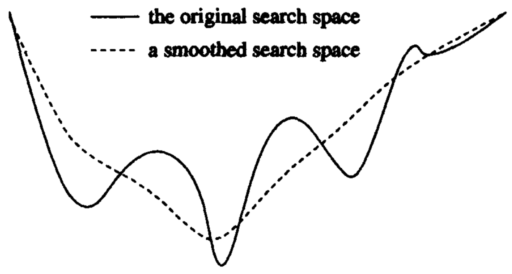


Fig. 5. The illustration of smoothing a search space. Many local minimum points are "filled" after a smoothing preprocessing, resulted in a simplified problem instance.

Local search is very efficient in several aspects. First, at the beginning of the search, using a full assignment of all variables in the search space, it reduces a larger search space to a much manageable one. Secondly, it searches for improvement within its local neighborhood using a testing for improvement and, if there is any improvement, takes an action for improvement. Since the object function has a polynomial number of input numbers, both "testing" and "action" are relatively efficient and often perform very well for a search problem. A major weakness of local search is that the algorithm has a tendency to get stuck at a locally optimum configuration, i.e., a local minima point, as illustrated in Fig. 4. In recent years, local search has been used to solve the n-queen problem [24]–[28], the satisfiability problem [5], [6], [7], and many practical application problems [9], [21], often with significant performance improvement.

For the traveling salesman problem, each feasible route defines a feasible solution point. The search space consists of all the feasible solution points. Two feasible solution points, i.e., two feasible routes, are considered as *neighbors* if one can be derived from the other via a definite number of edge exchanges. Different exchange schemes, such as city-swap, r-opt, and Or-opt, affect neighborhood structures in the search space. The object function in a TSP is defined as the length of the route. Each solution point in the search space gives a specific value to the object function.

Different neighborhood structures result in different terrain surface structures of the search space and produce different numbers of local minimum points. The effectiveness of a heuristic local search algorithm relies on the number of local minimum points in the search space. But no matter how one changes the heuristics in a local search, its computing power is limited.

Theorem 1 [20] If A is a local search algorithm whose neighborhood search time is bounded by a polynomial, then, assuming $\mathcal{P} \neq \mathcal{NP}$, A cannot be guaranteed to find a tour whose length is bounded by a constant multiple of the optimal tour length, even if an exponential number of iterations is allowed.

This theorem indicates that, for the traveling salesman problem in general, there are many local minimum points in the search space. Thus, the capability of a primitive local search is limited no matter how one defines the neighborhood structure.

In the next section, we will introduce a local search method with a search space smoothing technique. It is able to dynamically reconstruct the problem structure and smooth the rugged terrain surface of the search space. This smoothing method could "hide" some local minimum points in a simplified problem instance, therefore, improving the performance of the traditional local search algorithms.

IV. Search Space Smoothing

Local minimum points make a search problem hard. The less the number of local minimum points, the more effective a local search algorithm is. In order to improve the performance of a local search algorithm, some methods, e.g., city-swap and r-opt heuristics, attempt to limit the number of local minimum points by changing the topological structures in the search space.

As an alternative, from scrambling search, we developed a search space smoothing technique to limit the number of local minimum points in the search space. Informally the basic idea of the method can be explained as follows. Assume there is a search space with many local minimum points (see Fig. 5), where a solution point could be easily trapped. We use a smoothed search space to approximate the original search space. After search space smoothing, some local minimum points are temporarily "filled" and they will no longer cause any "trapping" problems. So, as illustrated in Fig. 5, the number of local minimum points is "reduced." A smoothing process only changes the metric characteristics of a search space and leaves its topological structure untouched.

Search space smoothing is a special technique of *multispace search* developed in recent years [8], [9], [10]. A traditional search algorithm optimizes by changing values in the value space, it is difficult for a value search algorithm to handle some pathological phenomena in optimization problems. Multispace search not only alters values in the value space but also scrambles across the variable space and other active spaces, dynamically changes the problem structure in terms of variables, parameters, and other components, and incrementally constructs the final solution to the search problem. The idea of multispace search was derived from principles in nonequilibrium thermodynamic evolution that structural changes are fundamental than quantitative changes and evolution depends on the growth of new structure in biological system rather than

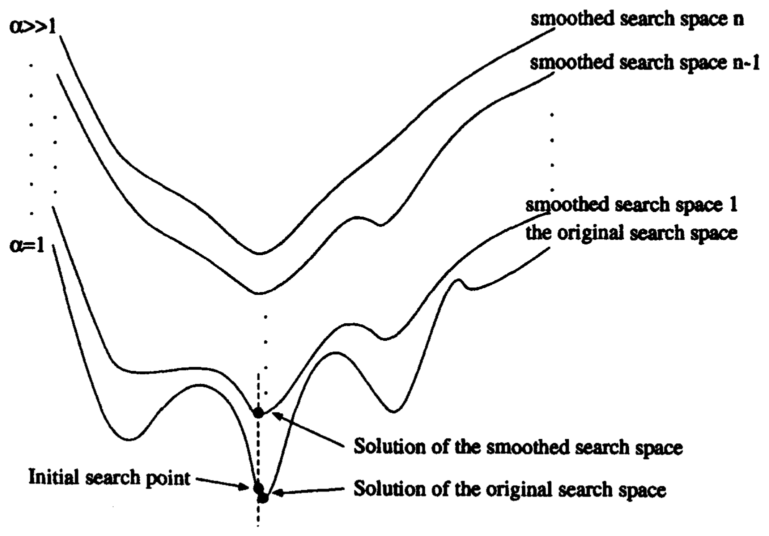

α>>1
α=1
smoothed search space n
smoothed search space n-1
smoothed search space 1
the original search space
Solution of the smoothed search space
Initial search point
Solution of the original search space

Fig. 6. A series of smoothed search spaces is generated. The solutions of the smoother search spaces are used to guide the search of those of the rugged search spaces.

just information transmission. Structural multispace operations disturb the environment of forming local minima, this makes multispace search a very natural approach to handle the difficult optimization problems [8], [9], [10]

There are several advantages to using a search space smoothing technique. We discuss its roles in finding the global minimum point in a smoothed search space and in finding the global minimum point in the original search space.

First, we note that smoothing makes the search of the global minimum solution point in a smoothed search space easier. After a smoothing operation, the number of local minimum points in a smoothed search space is "reduced." Thus, for a search process, the probability of being trapped in a local minimum point is minimized and the chance of finding a global optimum point is increased. In the ideal case, after a smoothing operation, there might be only one local minimum point left, which is the global minimum point in the smoothed search space. In this case, a local search algorithm with search space smoothing would find the global minimum point quickly.

Secondly, since a smoothed search space has qualitatively accumulated the topological structure information of the original search space, a smoothing processing facilitates the search of the global minimum point in the original search space. Using an appropriate smoothing scheme, e.g., a gradually approximated smoothing scheme (see Section V.A), the global minimum point in the smoothed search space could be set very close to the global minimum point in the original search space. If we use the global minimum point in the smoothed search space as the initial starting point in the original search space, as illustrated in Fig. 6, then the probability of finding the global minimum point in the original search space could be increased considerably.

A smoothing operation has different levels of strength, resulting in a search space with a varying degree of smoothness. We use a *smoothing factor*, α, to characterize the degree of a smoothing operation and the smoothness of the resulting search space. If $\alpha = 1$, no smoothing operation is applied, the search space is the same as the original search space. If $\alpha > 1$, a smoothing operation is applied, the smoothed search space is flatter than that of the original search space. If $\alpha \gg 1$, the smoothing operation has a stronger effect, resulting in a nearly flat search space.

To apply search space smoothing techniques to local search, one is often faced with a contradictory situation. That is, if one applies a weaker smoothing operation, the topological structure of the smoothed search space is similar to the original one. The heuristic guidance information of the original search space is thus strong. A weaker smoothing operation, however, results in less reduction in the number of local minimum points in the original search space. In order to increase the chance of finding the global minimum points in the smoothed search space, we expect a strong smoothing operation that produces a flatter search space, but we may lose some heuristic guidance information. This contradictory situation can be resolved using a series of search space smoothing operations. By altering the shape of the objective function in a parameter space, i.e., the α space, a series of smoothed search spaces with their structures varying from a flatter search space to the original search space, as illustrated in Fig. 6, is generated. Each (upper) search space is a further smoothing of the lower search space. The solutions of a smoothed, flatter search space are used to guide the search of those in the more rugged search spaces.

V. A Local Search Algorithm With Search Space Smoothing

In this section, we apply the search space smoothing technique to TSP problem solving and give a local search algorithm implementing the idea.

A. Smoothing the Search Space for TSP

There are many ways to smooth a search space. The smoothing method employed here is to simplify the search space of a hard TSP problem instance into a series of simple ones, each is a gradually smoothed approximation of the original search space (as discussed in the last section). The number of local minimum points in a simplified search space can be reduced by such a simplification.

A *trivial case* of the TSP is the case where all the distances among cities are equal. In such a case, any route is an optimal one. The search space is *flat* since there is no local minimum point in the search space. For a given TSP, let n be the number of cities, d_{ij} be the distances between cities i and j[1] $(i, j = 1, 2, \ldots, n)$, and $\bar{d}$ be the average distance among all the cities, then

$$\bar{d} = \frac{1}{n(n-1)} \sum_{i \neq j} d_{ij}. \tag{1}$$

So a trivial case of TSP is one with $d_{ij} = \bar{d}$, for $i, j = 1, 2, \ldots, n$.

A *simplified* TSP instance can be defined by a specified smoothing factor, α. For a TSP instance, the distances among all the cities, i.e., $d_{ij}(\alpha)$, are determined by α as:

$$d_{ij}(\alpha) = \begin{cases} \bar{d} + (d_{ij} - \bar{d})^{\alpha} & \text{if } d_{ij} \geq \bar{d} \\ \bar{d} - (\bar{d} - d_{ij})^{\alpha} & \text{if } d_{ij} < \bar{d} \end{cases} \tag{2}$$

where $\alpha \geq 1$. When α is decreased step by step from a large number, say 10, to 1, a series of simplified TSP instances is generated. A search space generated from a larger α exhibits a smoother terrain surface, and a search space generated from a smaller α exhibits a more rugged terrain surface. Two extreme cases of the series of the TSP instances are: (1) If $\alpha \gg 1$, then $d_{ij}(\alpha) \rightarrow \bar{d}$, this is the trivial case; (2) If $\alpha = 1$, then $d_{ij}(\alpha) = d_{ij}$, which is the original problem.

Once we have a series of smoothed problem instances, we can use any existing local search algorithm to solve them. The idea of this simplification method is simple. We start from an approximate trivial case, i.e., the case with a fairly "flat" search space, generate a simplified problem instance with its smoothed search space close to the trivial one, and find the solution to this simplified problem using an available TSP local search algorithm. The solution of the problem instance is then taken as the initial route to the next problem instance that has a slightly more complicated search space. The problem is again solved using the same local search algorithm. The above procedure is repeated until the final problem instance having the original search space is solved.

The ideas described here are expressed in an algorithm form below.

B. A Local Search Algorithm With Search Space Smoothing

Based on the search space smoothing idea, we give a local search algorithm, the TSP1 algorithm, for the TSP problem (see Fig. 7). The algorithm works as follows. Initially, procedure *get_a_TSP_instance*() randomly constructs an initial route,

[1]Note that, without lossing of generality, all distances d_{ij} are normalized so that $0 \leq d_{ij} \leq 1$.

```
procedure TSP1 ()
begin
    /* Initialization */
    route := get_a_TSP_instance();
    $\alpha := \alpha_0$;

    /* Search */
    while ($\alpha \geq 1$) do
    begin
        /* Generate a Simplified TSP Instance */
        for i := 1 to n do
            for j := 1 to n do
                d[i][j] := Compute_$d_{ij}(\alpha)$;
        /* Search */
        route := TSP_local_search(d,route);
        $\alpha := \alpha - 1$;
    end;
end;
```

Fig. 7. **TSP1:** A local search algorithm with search space smoothing technique.

route, and returns route length, $L = |\text{route}|$, as the value of the object function. An initial value of the smoothing factor α, α_0, is chosen appropriately (in this case it is chosen as less than 10). During each iteration of the search, a simplified TSP instance with a smoother search space is generated. This is done by two *for* loops that compute the distances among all the cities in the TSP instance. Any available heuristic local search algorithm, i.e., TSP_*local_search*(), can be used to minimize the object function, $L = |\text{route}|$. In this case study, we have used three typical TSP local search algorithms: 2-opt, *Or*-opt, and city-swap procedures. Procedure TSP_*local_search*() finds a route and its length as the solution to the present TSP instance.

At the end of each iteration loop, α is reduced by one. The same *while* loop is repeated and a more complicated TSP instance is generated. It takes the solution produced from the previous, simpler TSP problem instance as the initial route and searches for a solution of a more difficult TSP instance. This solution is again taken as the initial route of a more complicated TSP problem instance (with a smaller α), which will be solved by the TSP local search algorithm.

In the TSP1 algorithm, α_0 iterations are required to decrease α down to 1. After α is decreased down to 1, we have the original TSP instance. The solution yielded by the algorithm at this time is thus a solution to the original problem.

The run time of the TSP1 algorithm can be estimated as follows: Procedure *get_a_ TSP_instance*() takes $O(n^2)$ time to generate an initial TSP instance since, in case of a complete graph, there are $n(n-1)/2$ arcs that need to be connected. At the beginning of the search, two *for* loops take $O(n^2)$ time to produce a simplified TSP instance. Let the run time of procedure TSP_*local_search*() be $O(\cdot)$, then the run time of the *while* loop is $O(\alpha_0 n^2) + \alpha_0 O(\cdot)$. Summarizing the above, the time complexity of the TSP1 algorithm is:

$$O(n^2) + \max\{O(\alpha_0 n^2), \alpha_0 O(\cdot)\} = \max\{O(n^2), O(\cdot)\}. \quad (3)$$

Since most TSP_*local_search* procedures, such as 2-opt, 3-opt, and Lin-Kernighan algorithms, have a higher than $O(n^2)$ run time,[2] the run time of the TSP1 algorithm can be written as $O(\cdot)$, which is determined by the local search algorithm used in the TSP1 algorithm and is not affected by the $O(n^2)$ run time of the search space smoothing procedure.

In the next section, we will show experimental results and some performance comparisons among different local search algorithms with and without a search space smoothing preprocessing.

VI. Experimental Results and Performance Comparisons

A goal of testing a TSP algorithm is to see how is the route length produced by the algorithm close to an optimal route length. For any given TSP instance, the optimal route length in most cases is unknown. Thus, researchers in the area often compare, for the same TSP instance, the route length produced from some typical TSP local search algorithms [14].

We used the 2-opt, Or-opt, and city-swap procedures in the TSP1 algorithm. We will directly make relative performance comparisons between these local search algorithms *without* smoothing preprocessing, i.e., TSP_*local_search*(), and the same algorithm *with* search space smoothing preprocessing, i.e., the TSP1 algorithm. Since TSP1 runs a local search algorithm α_0 times, we should run the same local search algorithm (without smoothing) α_0 times, and compare the route length produced from the TSP1 and the minimum route length produced from α_0 executions of the same local search algorithm. That is, if we run a local search algorithm R_{local} times, we will run the TSP1 algorithm R_{TSP1} $(= R_{\text{local}}/\alpha_0)$ times, and then compare their results. In the following experiments, we use "running times" to denote R_{local}. The initial value of α, i.e., α_0, was set to 5.

All the comparisons in the following experiments are based on the percentage improvement of the route length. The improvement percentage of algorithm A over algorithm B is defined as:

$$\frac{\text{route\_length}(B) - \text{route\_length}(A)}{\text{route\_length}(A)} \times 100\%,$$

where route_length (A) represents the best solution (i.e., the shortest route length) given by algorithm A. The same is true for the route_length (B).

In our experiments, the initial problem structure of a TSP was generated randomly. The distances among n distinct cities were set randomly within the range of [0, 1] with a uniform distribution.

A. Improvement for a TSP Instance

For a TSP instance with 50 cities, Fig. 8 shows that, as the local search execution time (R_{local}) increases, the route lengths yielded by the 2-opt local search algorithm (a solid line) and the TSP1 algorithm with search space smoothing preprocessing (a dashed line). We note from the figure that, as the local search execution time increases, the TSP1 algorithm with search space smoothing outperforms the 2-opt local search algorithm.

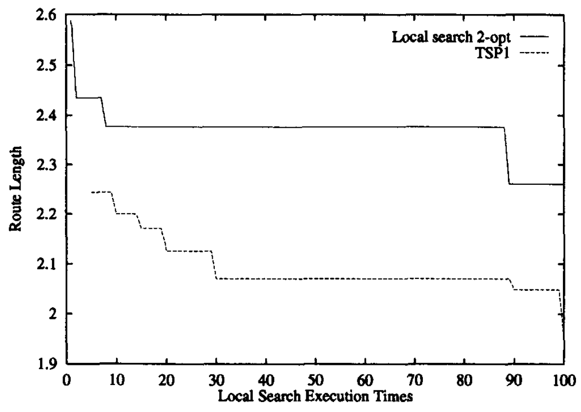


Fig. 8. Route lengthes as the increase of running times for the 2-optlocal search algorithm and the TSP1 algorithm (50 cities).

TABLE I
Performance Comparisons for the 2-Opt Local Search Algorithm and the TSP1 Algorithm With a Search Space Smoothing Preprocessing (10 Instances, 25 Executions for Each Instance, 50 Cities).

Instances	2-opt algorithm	*TSP*1 algorithm	*L* improvements
1	2.377	2.113	12.49%
2	2.381	2.112	12.75%
3	2.775	2.393	15.97%
4	2.438	2.084	16.97%
5	2.524	1.914	31.89%
6	2.433	2.226	9.284%
7	2.652	2.346	13.04%
8	3.035	2.518	20.51%
9	2.873	2.485	15.65%
10	2.423	2.250	7.712%

B. Improvements for Multiple TSP Instances

Ten randomly generated TSP instances, each with 50 cities, were used to show the performance improvement of the TSP1 algorithm with a search space smoothing preprocessing. One can observe, from Table I, that significant performance improvements, with percentages varying from 7.712% to 31.89%, were achieved for all ten TSP problem instances.

C. Improvement With Increasing Problem Size

Table II gives some statistics on the improvement percentage as the increase of the problem size. All of the results in the table were obtained based on the average over 100 TSP problem instances. As the problem size increases, better performance improvements were obtained for the city-swap algorithm. Performance figures of the 2-opt and Or-opt algorithms remain relatively stable.

[2]The worst-case running times for 2-opt, 3-opt, and Lin-Kernighan search procedures are $O(n^2)$, $O(n^3)$, and $O(n^5)$, respectively.

TABLE II
PERCENTAGES OF PERFORMANCE IMPROVEMENT AS THE INCREASE OF THE PROBLEM SIZE (AVERAGE OVER 100 PROBLEM INSTANCES 25 EXECUTIONS FOR EACH INSTANCE)

n	50	60	70	80	90	100
2-opt	15.91%	19.43%	22.43%	22.39%	25.43%	26.68%
Or-opt	3.781%	4.669%	6.205%	6.202%	6.208%	5.993%
city swap	19.68%	22.90%	25.53%	27.95%	29.83%	33.53%

TABLE III
PERCENTAGE IMPROVEMENT OF THE ROUTE LENGTH AS THE INCREASE OF RUNNING TIMES (AVERAGE OVER 100 INSTANCES, 50 CITIES).

Running Times	25	50	75	100
2-opt	15.91%	13.62%	15.49%	14.22%
Or-opt	3.781%	3.216%	3.398%	3.272%
city swap	19.68%	20.55%	18.95%	21.03%

For the same problem size, there are differences among the percentages of improvement for three different TSP local search algorithms. The improvement of the *city-swap* is the best, that of the 2-opt is the second, and the improvement of the Or-opt is the least. It is easy to understand these differences if we recall the original performances of these three TSP local search algorithms. The Or-opt algorithm is the best among all three algorithms. Compared to solutions given by the other two algorithms, solutions yielded from the Or-opt algorithm are closer to the optimal one. Therefore, less space is left for further improvement. The *city-swap* heuristic is the worst and there is much space left for further improvement.

D. Improvement as the Increase of Running Times

As illustrated in Table III, the percentages of improvement for different local search algorithms are relatively stable with the increasing of execution times. All the results shown in the table were obtained based on the average of 100 problem instances. Each instance has 50 cities.

E. Improvement for Different Simplification Schemes

There are different ways of controlling the strength of a search space smoothing operation. The decreasing scheme for α mentioned before is called *Scheme* 1. The use of different decreasing schemes for α results in a different series of the simplified TSP instances. To test the performance under different simplification schemes, we used a different decreasing scheme, *Scheme* 2. In Scheme 2, α is reduced by M/x, where M is a positive integer and x is incremented from 1 to M. This produces a sequence of decreasing α: $M, M/2, M/3, \ldots, M/(M-1), 1$. In the following experiment, M is set to 5. Thus, five run times were required in Scheme 2 to yield a solution to the original problem.

Scheme 1 yields a smoother search space than that of Scheme 2. Table IV demonstrates that there is not much difference in their performance variations. That is, TSP1 algorithm with search space smoothing preprocessing is less sensitive to the implementation details.

TABLE IV
PERFORMANCE COMPARISONS OF THE DIFFERENT SIMPLIFICATION SCHEMES (AVERAGE OVER 100 INSTANCES, 25 RUNNINGS FOR EACH INSTANCE, 50 CITIES)

	2-opt	Or-opt	city-swap
Scheme 1	15.91%	3.781%	19.68%
Scheme 2	15.92%	3.586%	19.19%

VII. CONCLUSION

Due to the rugged terrain surface of the search space, a traditional local search algorithm has a tendency of getting stuck at a locally optimum configuration. In this paper, we present a local search method with search space smoothing. It significantly improves the performance of the conventional local search algorithms. We have shown a case study of applying this method to solve the traveling salesman problem (TSP).

In a parameter space, a search space smoothing preprocessing transforms the original problem instance into a series of gradually simplified problem instances. And the solutions of the simplified problem instances can then be used to guide the search of more complicated instances. The power of this smoothing technique relies on the reduced complexity of solving the simplified problem instances and the effectiveness of using the intermediate solutions to guide the search of the increasingly complicated problem instances. Using randomly generated problem instances with varying problem sizes, during numerous experiments, we have observed that this smoothing technique has significantly improved the performance of several well-known local search algorithms for TSP.

REFERENCES

[1] V. Cĕrny, "A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm," *Technical report*, Institute of Physics and Biophysics, Comenius University, Bratislava, 1982.

[2] R. Durbin and D. Willshaw, "An analogue approach to the traveling salesman problem using an elastic net method," *Nature*, 326:689–691, 1987.

[3] B. L. Golden and W. R. Stewart, "Empirical Analysis of Heuristics," In *The Traveling Salesman Problem*. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, pp. 207–249. John Wiley & Sons, New York, 1985.

[4] G. A. Groes, "A method for solving traveling-salesman problems," *Operation Research*, vol. 6, pp. 791–812, 1958.

[5] J. Gu, "How to solve Very Large-Scale Satisfiability (VLSS) problems," *Technical Report*, 1988 (Present in part in, J. Gu, Benchmarking {SAT} Algorithms, Technical Report UCECE-TR-90-002, 1990).

[6] J. Gu. "Efficient local search for very large-scale satisfiability problem," *SIGART Bulletin*, vol. 3, no. 1, pp. 8–12, Jan. 1992, ACM Press.

[7] J. Gu. "Local search for satisfiability (SAT) problem," *Trans. on Systems, Man, and Cybernetics*, vol. 23, no. 4, pp. 1108–1129, 1993.

[8] J. Gu, "Multispace search: A new optimization approach, " In *Technical Report UCECE-TR-90-001*, Mar. 1990.

[9] J. Gu, "Constraint-Based Search," Cambridge University Press, New York, to appear.

[10] J. Gu and B. Du, "Graph partitioning by simulated evolution," Technical Report UCECE-TRT-92-001, Jan. 1992.

[11] J. J. Hopfield and D. Tank, "Neural computation of decisions in optimization problems," *Biological Cybernetics*, vol. 5, pp. 141–152, 1985.

[12] D. S. Johnson, "More approaches to the traveling salesman guide," *Nature*, vol. 330, pp. 525, 1987.

[13] S. Kirkpatrick, C. D. Gelat and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[14] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, editors. *The Traveling Salesman Problem.* John Wiley & Sons, New York, 1985.

[15] S. Lin, "Computer solutions of the traveling salesman problem," *Bell Sys. Tech. Journal*, vol. 44, no. 10, pp. 2245–2269, Dec. 1965.

[16] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Operation Research*, vol. 21, no. 498–516, 1973.

[17] J. P. Norback and R. F. Love, "Geometric approaches to solving the traveling salesman problem," *Management Science*, vol. 23, pp. 1208–1223, 1977.

[18] J. P. Norback and R. F. Love, "Heuristic for the hamiltonian path problem in euclidian two space," *J. Oper. Res. Soc.*, vol., 30, pp. 363–368, 1979.

[19] I. Or, "Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking," *Ph.D. thesis*, Northwestern University, Evanston, IL, 1976.

[20] C. H. Papadimitriou and K. Steiglitz, "On the complexity of local search for the traveling salesman problem," *SIAM J. on Computing*, vol. 6, no. 1, pp. 76–83, 1977.

[21] R. Puri and J. Gu, "An efficient algorithm for microword length minimization," *IEEE Transactions on CAD*,vol. 12, no. 10, pp. 1449–1457, Oct. 1993.

[22] D. J. Rosenkrantz, R. E. Stearns and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. on Computing*,vol. 6, pp. 563–581, 1977.

[23] B. M. Schwartzschild, "Statistical mechanics algorithm for monte carlo optimization," *Physics Today*, vol. 35, pp. 17–19, 1982.

[24] R. Sosič and J. Gu, "How to search for million queens," *Technical Report UUCS-TR-88-008,* Dept. of Computer Science, Univ. of Utah, Feb. 1988.

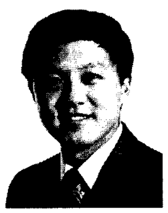
[25] R. Sosič and J. Gu, "A polynomial time algorithm for the n-queens problem," *SIGART Bulletin*, vol. 1, no. 3, pp. 7–11, Oct. 1990, ACM Press.

[26] R. Sosič and J. Gu, "3,000,000 queens in less than one minute," *SIGART Bulletin*, vol. 2, no. 2, pp. 22–24, Apr. 1991, ACM Press.

[27] R. Sosič and J. Gu, "Fast search algorithms for the n-queens problem," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-21, no. 6, pp. 1572–1576, Nov./Dec. 1991.

[28] R. Sosič and J. Gu, "Efficient local search with conflict minimization," *IEEE Trans. on Knowledge and Data Engineering*, vol. 6, 1994.

[29] W. R. Stewart, "A computationally efficient heuristic for the traveling salesman problem," In *Proc. 13th Annual Meeting of S.E. TIMS*, pp. 75–85, 1977.



Jun Gu received B.S. degree in electrical engineering from the University of Science and Technology of China in 1982 and Ph.D degree in computer science from the University of Utah in 1989, where he was twice awarded the ASC Fellowships, twice awarded the University Research Fellowships, and twice awarded the ACM/IEEE academic scholarship awards. He joined the University of Calgary in late 1989. From 1990 to 1993, he was an Associate Professor with the Department of Electrical and Computer Engineering. Since 1994, he has been a Professor of Electrical and Computer Engineering at the University of Calgary, where he has received substantial funding from federal governmental agencies and industrial sectors. His research interest includes operations research and combinatorial optimization, communication systems, intelligent vehicle-highway systems, computer architecture and parallel processing, and structured techniques for CMOS, GaAs, and MCM VLSI system design. He developed many efficient optimization algorithms handling practical constrained optimization problems with over a million of variables. In 1987 he gave several discrete and continuous local search algorithms for the satisfiability (SAT) problem and other NP-hard problems. He authored the book *Constraint-Based Search* (Cambridge University Press, 1995).

Dr. Gu was Vice Program Chair of the 1993 IEEE International TAI Conference, Boston, MA. He is an Associate Editor of *Journal of Global Optimization* and *Journal of AI Tools* and is an Adjunct Professor of the National Research Center for Intelligent Computing Systems, the University of Science and Technology of China, and the Academy of Sciences of China. He is a member of the ACM, AAAI, International Neural Network Society, and Sigma Xi.

X. Huang, biography and photo not available.