April, 1999

# New parallel randomized algorithms for the traveling salesman problem

Leyuan Shi, *University of Wisconsin-Madison*
Sigurdur Olafsson, *University of Wisconsin-Madison*
Ning Sun, *University of Wisconsin-Madison*

# New Parallel Randomized Algorithms for the Traveling Salesman Problem

Leyuan Shi[1] Sigurdur Ólafsson Ning Sun

Department of Industrial Engineering
University of Wisconsin-Madison
Madison, WI 53706
email: leyuan, olafsson, sunn@ie.engr.wisc.edu

March 27, 1997

[1]Corresponding Author

The final verision of this article is available at https://doi.org/10.1016/S0305-0548(98)00068-9

## Statement of Scope and Purpose

The *Traveling Salesman Problem* involves finding the shortest route between a number of cities. This route must visit each of the cities exactly once and end in the same city as it started. As easy as it is to describe, this problem is notoriously difficult to solve. It is widely believed that there is no efficient algorithm that can solve it accurately. On the other hand, this problem is very important since it has many applications in such areas as routing robots through automatic warehouses and drilling holes in printed circuit boards. We present a new method, the *Nested Partitions* method, for solving the traveling salesman problem. The method is very flexible in that it is capable of finding good solutions rapidly and given enough time will identify the optimal solution. This new method is also very important because it is naturally compatible with parallel processing capabilities.

**Abstract**

We recently developed a new randomized optimization framework, the Nested Partitions (NP) method. This approach uses partitioning, random sampling, promising index, and backtracking to create a Markov chain that has global optima as its absorbing states. This new method combines global search and local search (heuristic) procedures in a nature way and it is highly matched to emerging massively parallel processing capabilities. In this paper, we apply the NP method to the Traveling Salesman Problem. Preliminary numerical results show that our method generates high quality solutions compared to well known heuristic methods and it can identify a set of *near* optimal solutions very rapidly.

*Key Words:* Optimization, Randomized Algorithm, Parallel Algorithm, Traveling Salesman Problem.

# 1 Introduction

The Traveling Salesman Problem (TSP) is well known to be the most prominent member of the rich set of combinatorial optimization problems [7, 12]. Originally formulated as the problem of finding the shortest route for a traveling salesman to visit all of his customers, the problem has found many important applications such as routing robots through automatic warehouses, sending couriers to automatic teller machines, and drilling holes through printed circuit boards. The list of applications continues, making the TSP one of the most important combinatorial optimization problems.

As easy as the TSP is to describe, it is notoriously hard to solve. In fact it belongs to the class of NP-complete problems, for which there is no known *deterministic* optimal search algorithm that runs in polynomial time. Some progress has been made in solving specific problems to optimality and much research effort in continuously applied to that task. As an example Applegate et.al. [2] recently solve twenty previously unsolved problems from the TSPLIB problem library [11] to optimality. However, in practice the TSP is usually solved by using heuristics that converge to a locally optimal solution [12].

A popular and effective approach for escaping such local optima is to use *randomized algorithms*. This has given rise to many randomized optimization methods for optimizing deterministic problems. Among these methods are *simulated annealing* [5], *evolutionary strategies* and *genetic* algorithms [9, 19], *tabu search* [6, 8], and *neural networks* [15]. All of these methods make use of randomness to perturb the current solution and hence escape from a local optima. Many of these methods give high quality solutions, although from the running time point of view, it is hard for these methods to compete with the deterministic heuristics [12].

We recently developed a new randomized optimization framework, the Nested Partitions (NP) method [16, 17]. This method can be applied to both deterministic and stochastic discrete optimization problems. The NP method systematically partitions the feasible region into smaller subregions until some of the subregions contain only one point. It then moves between regions based on information obtained by random sampling. This procedure is shown to generate a Markov chain [16]. The method keeps track of which part of the feasible region is the *most promising* in each iteration and the sampling is always concentrated in this region. This makes the NP method very efficient for problems that are well structured. By this we mean that good solutions to these problems tend to be clustered together. In [16], we have shown that the NP method converges to an optimal solution with probability one. One remarkable feature of the NP method is that it can combine global search and local search (heuristic) procedures in a natural way. The NP method is easy to implement and can be applied without prior knowledge of a starting point. In particular, the NP method is parallel in nature and is therefore highly compatible with parallel computer edgehitectures. Preliminary results show that the NP algorithms are very efficient compared with other known heuristics.

In this paper we discuss how the NP method can be applied to the TSP and show the efficiency of the NP method through a few examples. In particular, we present some new findings about the NP method for the traveling salesman problem. In Section 2, we briefly review the TSP and some well-known heuristics which will be used in the paper. In Section 3, we review the general procedure of the NP method. In Section 4, we provide a detailed procedure for the implementation of the NP method for the traveling salesman problem. We show how different methods of partitioning and sampling can affect the efficiency of the method and how we can incorporate efficient heuristics in calculating the promising index of each region. We also present a construction heuristic for the TSP which, as far as we know has not been previously reported in the literature. In Section 5, computational experiences with the NP method is reported. Concluding remarks and future research are given in the final section. We expect that all of these result can be readily transferred to other combinatorial problems.

# 2 The Traveling Salesman Problem

The TSP is the task of finding a route with the shortest possible length through a given set of cities. The problem consists of a number of cities, represented by vertices in a graph, and a number of connections, or edges, between the cities. Each edge is associated with a cost which represents the cost of traveling between the two cities connected by the edge. The objective is to find the tour that passes through each city exactly once and returns to the starting point such that the overall cost of traveling is minimized. Therefore, given a cost matrix $C = (c_{ij})_{i,j=1,2,\ldots,n}$, where $c_{ij}$ is the cost of going from city $i$ to city $j$, the TSP problem can be stated as follows

$$\min_{\theta \in \Theta} f(\theta) \equiv \min_{\theta \in \Theta} (c_{i_1 i_2} + c_{i_2 i_3} + ... + c_{i_n i_1}). \quad (1)$$

where $\theta = (i_1, i_2, ..., i_n)$ is a permutation of $\{1, 2, ..., n\}$ and $\Theta$ is the set of all such permutations.

A TSP is symmetric, and is simply called TSP, if $c_{ij} = c_{ji}$; that is, the cost of traveling from city $i$ to city $j$ is exactly the same as that from city $j$ to city $i$. On the other hand, if $c_{ij} \neq c_{ji}$, it becomes an Asymmetric Traveling Salesman Problem (ATSP). If the vertices are set of points in the plane the TSP is called an *Euclidean* TSP. For a detailed discussion of this problem see [7] and for practical solution methods see [12].

Many heuristic methods have been developed to solve the TSP [7, 12]. Most of these methods use a local search algorithm where the local search depends on a neighborhood structure. A local search algorithm starts with an initial feasible solution and successively moves to neighboring solutions until no further improvement is possible. In the following, we provide a brief review of a few well-known local search heuristics. A completely review can be found in [12].

### Nearest Neighbor Heuristic

We divide heuristics for the TSP can into two groups, *construction heuristics* and *improvement heuristics*. Construction heuristics take as an input only the graph representing the TSP and construct a feasible tour that is believed to have low cost. The simplest method of constructing such a feasible tour is to start with an arbitrary city and pick the next edge with uniform probability from the set of feasible edges. This method creates a feasible tour that is uniformly sampled from the set of all feasible tours. However, many heuristics have been devised that can construct tours with much lower cost. Perhaps the simplest one is the *nearest neighbor* heuristic. Using this scheme, the next edge is always selected as the feasible edge that has the lowest cost. This is hence a purely greedy search. Several variants exist and we refer the reader to [12] for a discussion.

### $k$-Opt Exchange

Given an initial tour heuristic methods can be used to make improvements to this tour. The *2-Opt exchange* is motivated by the fact that in the Euclidean TSP, if two edges cross the tour can be improved by removing the edges that cross and replace them with edges that do not cross. Although not as obvious improvements can also be made using this move if edges are not crossed and for non-Euclidean problems. This improvement move gives rise to the following algorithm.

Given a tour $\theta$
**while** improvements can be made **do**
  **for** every node $i$ in $\theta$ **do**
    Consider $i$ and its successor in the tour.
    If possible, make a 2-opt move involving these two nodes to decrease the cost.
    Update the tour $\theta$.
  **end**
**end**

Clearly we can generalize this method to a $k$-opt exchange algorithm, where $k$ is any integer [12].

### Lin-Kernighan Heuristic

Simple heuristics such as the 2-opt and 3-opt exchange methods quickly get stuck at a local optima that may be quite far from the true global optima. A more powerful method is the *Lin-Kernighan* heuristic that is based on allowing the search procedure to sometimes increase the tour length slightly and hence escape from a local optima. This method builds on simpler heuristics such as 2-opt exchanges by using them as a building block for a more complicated scheme. The *iterated Lin-Kernighan* heuristics improves upon this even more by using the original Lin-Kernighan heuristic and making a random 4-opt exchange and restarting whenever the algorithm gets stuck [12].

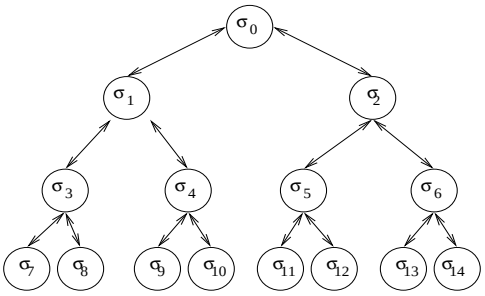


Figure 1: Example of a partitioning generated by the NP method. The feasible region is $\Theta = \sigma_0$. In each iteration the current most promising region is partitioned into two subregions.

It is clear that each of the more elaborate heuristics is capable of producing higher quality solutions, but at the same time the search time increases.

# 3 The Nested Partitions Method

The Nested Partitions (NP) method has been described in [16] for deterministic problems and in [17] for stochastic problems. For completeness, we provide a general framework for the Nested Partition (NP) algorithm for combinatorial optimization. In each iteration of the algorithm we assume that we have a region (subset) of $\Theta$ that is considered *the most promising*. We then partition this most promising region into $M$ subregions and aggregate the entire surrounding region into one region. At each iteration, we therefore look at $M+1$ disjoint subsets of the feasible region $\Theta$. Each of these $M+1$ regions is sampled using some random sampling scheme, and for each region a *promising index* is calculated. These promising indices are compared to determine which region is the most promising in the next iteration. If one of the subregions is found to be best, this subregion becomes the most promising region. If the surrounding region is found to be best, a region of less depth than the current region becomes the most promising. This new most promising region is partitioned and sampled in a similar fashion.

In the first iteration we use the entire feasible region $\Theta$ as the most promising region. Since the surrounding region is empty, we sample only from $M$ regions in the first iteration, or in any iteration where $\Theta$ is considered the most promising region. It is also clear that since $\Theta$ is finite, sooner or later we will have regions that contain only a single point in $\Theta$. We will call such regions, regions of *maximum depth*, and more generally, talk about the *depth* of any region. This is defined iteratively in the obvious manner, with $\Theta$ having depth 0 and so forth. For example, consider a feasible region $\sigma_0 = \Theta$, and assume that in each iteration we partition the current most promising region into two disjoint sets. In this case Figure 1 shows such a partitioning. Here $\sigma_1$ and $\sigma_2$ are disjoint subsets of $\sigma_0$ such that $\sigma_0 = \sigma_1 \cup \sigma_2$, $\sigma_5$ and $\sigma_6$ are similarly disjoint subsets of $\sigma_2$ such that $\sigma_2 = \sigma_5 \cup \sigma_5$ and so forth.

It should be noted that the partitioning scheme should be fixed during implementation of the method. This means that if a region $\sigma$ has been selected repeatedly as the most promising region, then the same partitioning rule should be applied to the region. Therefore, we call a region, that is constructed using the partitioning scheme described above, a *valid region* given the fixed partition. If a valid region $\eta$ is formed by partitioning a valid region $\sigma$, then $\eta$ is called a *subregion* of the region $\sigma$, and the region $\sigma$ is called a *superregion* of the region $\eta$.

We now introduce some notation that is used throughout the paper.

| | | |
|---|---|---|
| $\Theta$ | = | The feasible region. |
| $\Sigma$ | = | $\{\sigma \subseteq \Theta \mid \sigma$ is a valid region given a fixed partitioning$\}$. |
| $\sigma(k)$ | = | The most promising $\sigma \in \Sigma$ in the $k$-th iteration. |
| $I(\sigma)$ | = | The promising index for $\sigma \in \Sigma$. This is a real valued function defined on $\Sigma$. |

Using this notation, the general procedure of the NP algorithm is given as follows.

**The Nested Partitions Algorithm**

1. **Partitioning.**
   Partition the most promising region $\sigma(k)$, into $M_{\sigma(k)}$ subregions $\sigma_1(k), ..., \sigma_{M_{\sigma(k)}}(k)$, and aggregate the surrounding region $\Theta \setminus \sigma(k)$ into one region $\sigma_{M_{\sigma(k)}+1}(k)$.
   Referring back to the example in Figure 1, we could for example have $\sigma(k) = \sigma_4$, so $\sigma_1(k) = \sigma_9$, $\sigma_2(k) = \sigma_{10}$ and $\sigma_3(k) = \sigma_0 \setminus \sigma_4 = \sigma_3 \cup \sigma_2$.

2. **Random Sampling.**
   Randomly pick $N_j$ points from each of the regions $\sigma_j(k),\ j = 1, 2, ..., M_{\sigma(k)} + 1$,

   $$\theta_1^j, \theta_2^j, ..., \theta_{N_j}^j, \quad j = 1, 2, ..., M_{\sigma(k)} + 1,$$

   and calculate the corresponding performance values:

   $$f(\theta_1^j), f(\theta_2^j), ..., f(\theta_{N_j}^j), \quad j = 1, 2, ..., M_{\sigma(k)} + 1.$$

   The only requirement on the random sampling procedure is that each point in the region has a positive probability of being selected.

3. **Calculation of the Promising Index.**
   Assume that a *promising index* function $I(\sigma_j)$ has been selected. For each region $\sigma_j,\ j = 1, 2, ..., M_{\sigma(k)}+1$ calculate an estimate of the promising index. For example, we can define $I(\sigma_j)$ to be the best performance value in the region (other promising index functions will be discussed later).

   $$I(\sigma_j) = \min_{\theta \in \sigma_j} f(\theta), \quad j = 1, 2, ..., M_{\sigma(k)} + 1. \tag{2}$$

   Let $\hat{I}(\sigma_j)$ be an estimate of $I(\sigma_j)$. We can for example use

   $$\hat{I}(\sigma_j) = \min_{i=1,2,...,N_j} f(\theta_i^j), \quad j = 1, 2, ..., M_{\sigma(k)} + 1. \tag{3}$$

   We require that $I(\sigma) = f(\theta)$ if $\sigma = \{\theta\}$ is a region of maximum depth. Otherwise, no restrictions are imposed on the promising index.

4. **Backtracking.**
   Determine *the most promising region* $\sigma_{j_k}$.

   $$\jmath_k \in \arg \min_{j=1,...,M+1} \hat{I}(\sigma_j), \quad j = 1, 2, ..., M_{\sigma(k)} + 1 \tag{4}$$

   If more than one region is equally promising, the tie can be broken arbitrarily. If the index corresponds to a subregion of $\sigma(k)$, then let this subregion be the most promising region in the next iteration. Otherwise, if the index corresponds to the surrounding region, *backtrack* to a region which is determined by a pre-specified backtracking rule. For example, we could backtrack to the superregion of the current most promising region.

As we can see, the basic idea of the algorithm is to shift the focus from the feasible region $\Theta$ to a sequence of *subsets* of $\Theta$. Eventually this procedure picks a region containing only a global optimum and at that point no further transitions are made. It is also clear from Figure 1 why the algorithm is termed a *Nested Partitions* algorithm.

The method described so far can be considered a *generic algorithm* capable of supporting different partitioning techniques, sampling schemes, index functions, and backtracking rules. For example, instead of retreating only to the superregion, we can always make the entire feasible region the most promising region *if* the promising indices indicate that a backtracking is the appropriate move. This makes it easier

for the algorithm to move from one region to another far apart, and may be appropriate for some problem structures. Notice further that this backtracking rule has slightly less overhead than the one we presented in the algorithm above, since it does not keep track of more than one most promising region. On the other hand, since each single solution consists of a region in maximum depth, we can therefore backtrack to any superregion of a maximum depth region which contains the best solution found in the current iteration. Thus, the *backtracking* step of the NP algorithm can be modified according to backtracking rules. The NP method has a number of important features worth highlighting here.

As stated in the introduction, it has been shown that this method generates a Markov chain with state space $\Sigma$ and global optima as absorbing states. The Markov property of the NP method provides us with a powerful approach to study the convergence rate of the method. This research topic is closely connected to an exciting area of modern mathematical research, the study of quantitative convergence rates of Markov chains [13]. In [18], we have started to establish connections between the NP method and the minorization conditions method for establishing convergence rate [14].

One remarkable feature of the NP method is that it can combine global search and local search procedures in a natural way. Recall that the NP method shifts the focus from the single point in $\Theta$ to a sequence of subsets of $\Theta$, and the promising index is defined on these sets. Hence we can incorporate any effective heuristic methods into the algorithm by using them to calculate the promising index function. This is possible because the only requirement for selecting an appropriate promising index is that such an index function should agree with the objective function on regions at maximum depth, that is the regions which contain only one point. Therefore, finding the optimal solution for the original optimization problem will be equivalent to finding the best solution of the promising index.

The NP algorithm can also take maximum advantage of parallel computer edgehitectures. In every iteration, the algorithm looks at $M+1$ regions, each of which can be handled independently in parallel. The algorithm always spends the most computational effort in the region that is considered the most promising at any given iteration. Therefore, the NP algorithm favors regions that have many good solutions and no bad solutions, rather than a region that has one very good solution and many bad ones. The algorithm might therefore, in *finite time*, miss the optimal solution, but usually because it is surrounded by many bad solutions. This behavior is a direct consequence of the focus being shifted from individual solutions to *sets* of solutions.

The NP algorithm is generic and can be applied both to deterministic or stochastic optimization problems [17], and it is easy to implement and highly compatible with other optimization algorithms such as branch-and-bound algorithm [10, 7] and Genetic Algorithm [4].

Last, but not least, we could provide stopping criteria based on a probability statement (confidence interval) for the algorithm, since the algorithm essentially is a random search algorithm [16]. We now discuss the NP algorithm in detail for the TSP.

# 4 The Nested Partitions Algorithm for the TSP

In this section, four issues involved in implementing the NP algorithm efficiently for the TSP are discussed. They are: (i) How to partition the solution space. (ii) How to obtain the sampling points. (iii) How to select a promising index function. (iv) How to select the backtracking rule? These four problems constitute the core of our discussion on implementing the NP method.

## 4.1 Partitioning

To apply the NP algorithm to the TSP problem, first we need to consider how to partition the solution space into subregions. This must be done in such a way that they can be partitioned further until each subregion contains only a single solution. Although the NP method does not limit the way we do partitioning, the efficiency of the algorithm depends on the partitioning strategies. If the partitioning is such that good solutions are clustered together, the NP algorithm quickly identifies a set of near optimal solutions. In this paper, we present two partitioning techniques.

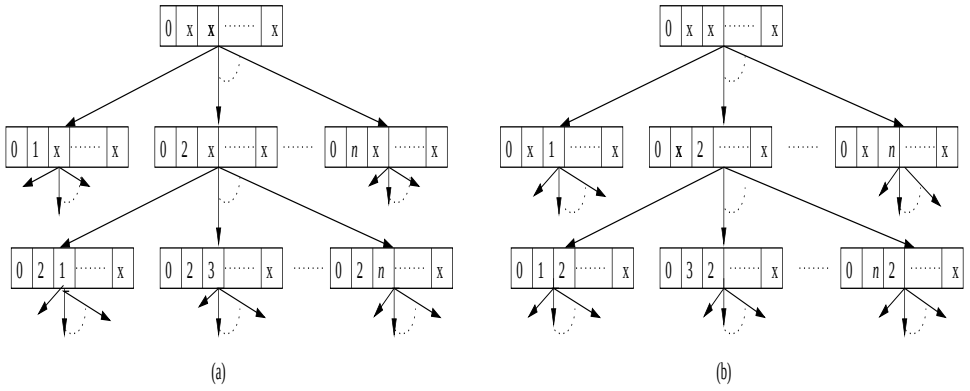


Figure 2: Two generic partitions.

### 4.1.1 Generic Partitioning

We first consider a generic way to partition the solution space. By this we mean a partitioning of the solution space that does not consider the objective function.

Given $n+1$ cities, suppose we choose city 0 as the starting point and other cities are labeled as $1, 2, 3, ...., n$. The whole solution space becomes all permutations of $\{1, 2, 3, ..., n\}$. First, we can divide this solution space into $n$ equal parts by fixing the first city on the tour to be one of $1, 2, ...,$ or $n$ (note that $M_{\sigma_0} = n$). We can further partition each such subregion into $n-1$ parts by fixing the second city as any of the remaining $n-1$ cities on the tour. This procedure can be repeated until the maximum depth is reached, when all the cities on the tour are fixed. In this way, the subregions at maximum depth contain only single solutions. Figure 2a illustrates this approach.

It should be noted that there exist many such partitions. For example, when we choose city 0 as the starting point, instead of fixing the first city on the tour, we could fix any $i$-th city on the tour to be one of cities $1, 2, ..., n$ (see Figure 2b). This partition provides a completely different set of subregions.

The advantage of the generic partitioning is that the search tree is completely predictable in general and is highly regular in terms of branching degrees and searching depths. Therefore, this type of partitions is ideal for parallel algorithms. On the other hand, the generic partitioning focuses on the solution space only. Intuitively, more efficient partitions could be constructed if the objective function was considered. This motivates our second partitioning technique.

### 4.1.2 Knowledge-Based Clustered Partitioning

The generic partitioning does not consider the objective function when partitioning the feasible region. This may lead to difficulties in distinguishing between regions and consequently the algorithm may not find any particular region to concentrate the computational effort. This has been our experience with some large problems. If the NP method is applied using the above partitioning, it may retreat frequently and not settle down in a particular region. On the other hand, the NP method is likely to perform much better if good solutions tend to be clustered together for a given partitioning. To impose such structure, we consider the following partitioning scheme through a simple example.

**Example:** Assume we have $n = 5$ cities defined by the undirected graph in Figure 3. As an initialization procedure we store the edges in an adjacency list and sort each of the linked lists that are connected to the cities (see the following table). For example, in the following adjacency list, the first row provides a linked list for city $A$, that is $E$ is the city closest to $A$, $C$ is the city second closest to $A$, $B$ is the city next closest to $A$, and $D$ is the city least close to $A$.

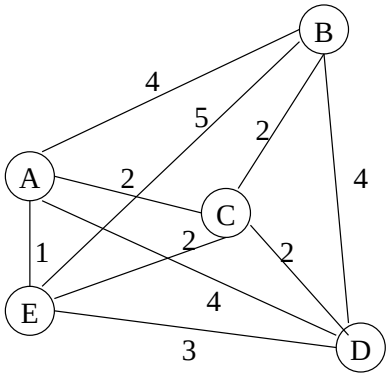


Figure 3: TSP Example

| City | Closest two | | Next two | |
|---|---|---|---|---|
| A → | E → | C → | B → | D |
| B → | C → | A → | D → | E |
| C → | A → | B → | D → | E |
| D → | C → | E → | A → | B |
| E → | A → | C → | B → | D |

This adjacency list becomes the basis of the partitioning. The entire region is all paths that start with the city A (chosen arbitrarily). If in each iteration we partition the solution space into $M = 2$ subregion then the first subregion consists of all the paths that start with either (A,E) or (A,C) as the first edge. The second subregion consists of all the paths that start with (A,B) or (A,D) as the first edge.

Now assume that the first subregion is chosen as the most promising region. Then the first subregion of that region is the region that consists of all paths that start with (A,E,A),(A,E,C),(A,C,A) or (A,C,B). The second region can be read from the adjacency list in a similar manner. Notice that one of these conditions creates an infeasible solution so there is no guarantee that all paths in a subregion will be feasible. It is, however, easy to check for feasibility during the sampling stage, and in fact this must always be done.

This partitioning is illustrated in the Figure 4 below. We notice that maximum depth regions don't restrict the feasible region to a single solution, but only to a smaller subset of solutions. For example, if we consider the first maximum depth region illustrated in Figure 4 then this region reduces the possible tours to feasible tours in the digraph shown in Figure 5. Clearly the choice of feasible tours in this graph is not large. In fact there are only three feasible tours in this graph.

## 4.2 Random Sampling

The method used to obtain random samples from each region in each iteration is not fixed by the NP algorithm. However, sampling schemes will greatly affect the efficiency of the NP algorithm. We will discuss various sampling schemes based on the abovementioned partitioning techniques.

### 4.2.1 Random Sampling with Generic Partitioning

Assume that the generic partitioning (Figure 1a) is used and the current most promising region is of depth $k$. This means that the first $k$ edges in the tour have been determined. Obtaining a sample from this region entails finding the $n-k$ remaining edges. One approach would be simply to pick the edges consecutively, such that each feasible edge has equal probability of being picked (uniform sampling). However, this approach may not always give good results in practice. The reason is the same as in the partitioning problem: uniform sampling considers only the solution space itself. To incorporate the objective function into the sampling, we present the following sampling schemes.

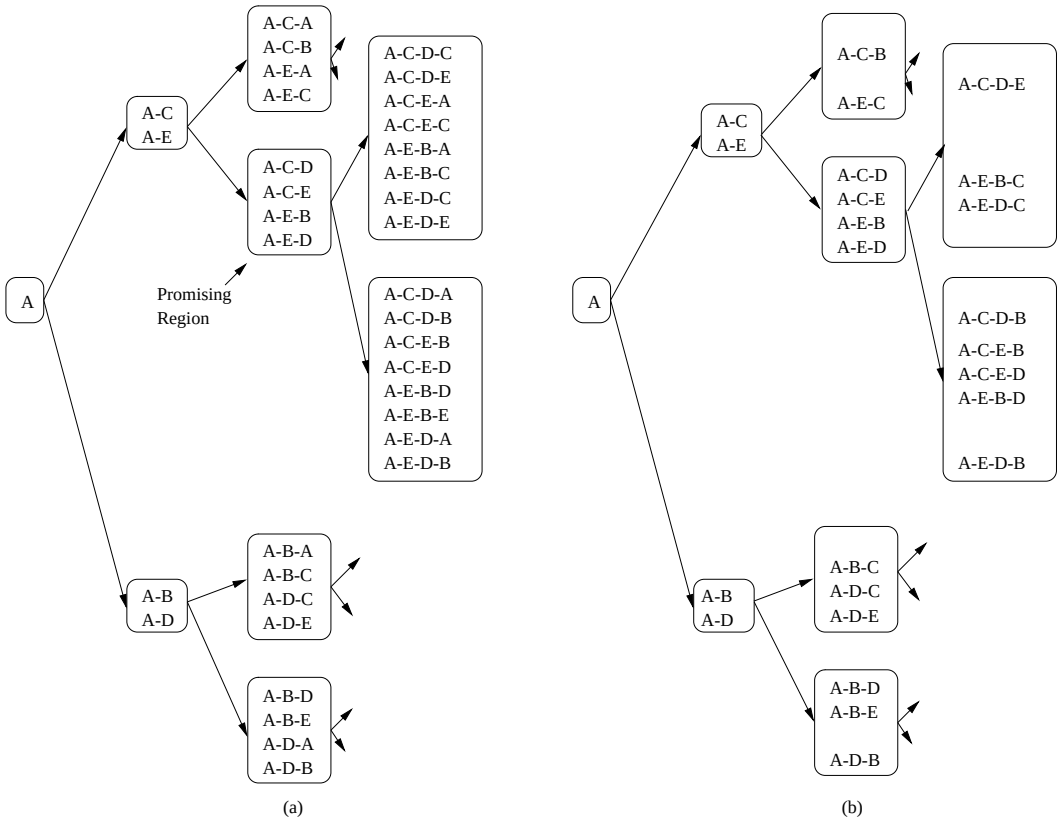


Figure 4: Knowledge-based clustered partitioning for the TSP feasible region. The figure in (a) shows the subregions including the infeasible solutions generated. The figure in (b) shows only the feasible solution. Note that the sampling procedure sees only the feasible solutions, so these are the effective subregions in practice.

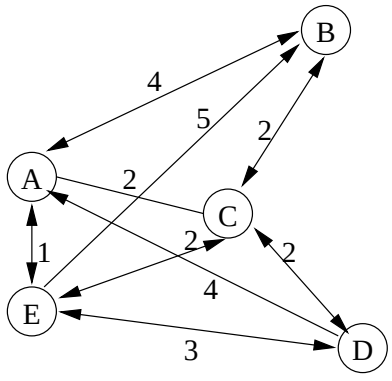


Figure 5: Restricted graph corresponding to a maximum depth region in the knowledge-based clustered partitioning. This graph has only three feasible tours.

**Weighted Sampling I**

Assume that the generic partitioning shown in Figure 3a is used and the current sampling region is $\sigma = (j_1, j_2, ..., j_k, x, x, ..., x)$ with the first $k$ cities fixed. A random sampling point (a tour) can be generated as follows.

**for** $i = k+1 : n$ **do**
  generate a random number $u$ uniformly distributed on (0,1).
  **for** $l = i : n$ do
    **let** $w_{j_{i-1},j_l} = \frac{\frac{1}{c_{j_{i-1},j_l}}}{\sum_{h=i}^{n} \frac{1}{c_{j_{i-1},j_h}}}$,
  **if** $\sum_{m=i}^{i^*} w_{j_{i-1},j_m} \leq u < \sum_{m=i}^{i^*+1} w_{j_{i-1},j_m}$
    **let** the next city $j_i$ be $j_{i*}$
  **else**,
    **let** the next city $j_i$ be a randomly selected according to a uniform distribution.
  **end**
**end**

At each iteration, weights ($w_{j_{i-1},j_l}$) are calculated and assigned to each of the remaining cities which need to be determined. Each weight $w_{j_{i-1},j_l}$ is inversely proportional to the cost of the edge from city $j_l$ to the city $j_{i-1}$. Therefore, this sampling scheme will pick shorter edges with higher probability than longer ones and pick the lowest cost edge with the maximum probability. This procedure mimics a greedy search for the TSP but still has a positive probability of selecting any feasible edge.

One particular concern in this type of sampling scheme is that obtaining a weighted sample can be quite expensive (we need to calculate the weights $w_{j_{i-1},j_l}$). Therefore, we present a modified version of the weighted sampling scheme, that combines uniform sampling and nearest neighbor search as special cases.

**Weighted Sampling II**

Assume a region defined by the first $k$ cities fixed and a predetermined constant $p \in (0,1)$.
**for** $i = k+1 : n$ **do**
  generate a random number $u$ uniformly distributed on $(0,1)$.
  **if** $u < p$,
    **let** the next edge $(v_{i-1}, v_i)$ be the lowest cost edge.
  **else**,
    **let** $(v_{i-1}, v_i)$ be a random edge according to a uniform distribution.
  **end**
**end**

This sampling scheme is less expensive than the first weighted sampling scheme since we can use the adjacency list and select the first feasible edge if $u < p$. This implies that we assign a *weight* $p$ only to the lowest cost edge while the rest of edges have the same weight $\frac{1-p}{k-1}$.

It should be noted that the case $p = 1$ corresponds to a nearest neighbor search and the case $p = 0$ corresponds to uniform sampling. For $0 < p < 1$ we have a non-uniform sampling scheme that picks each feasible tour with a positive probability. Therefore the NP algorithm is guaranteed to converge to a global optimum with probability one.

As we pointed out earlier, obtaining a weighted sample can be quite expensive. Even for a uniform sampling scheme, obtaining a sampling point in a subregion with $k$ fixed edges has complexity $O(n-k)$. We hence propose another sampling scheme.

**Two-Step Sampling Scheme**

Assume that we need to generate $N_j$ random sampling points.

1. Obtain one random sample using either uniform or weighted sampling.

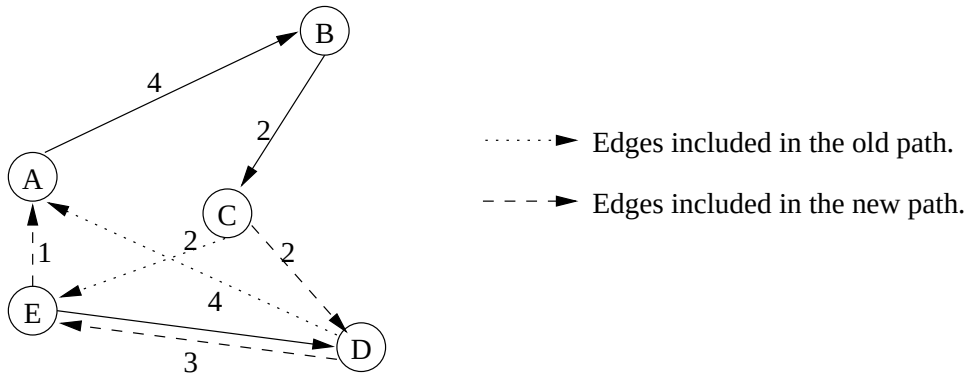


Figure 6: The second step in the Two-Step Sampling Scheme.

2. Obtain $N_j - 1$ more samples by making small perturbations of the sample generated in the first step.

In this paper, the second step involves randomly selecting two edges and connecting the first vertex of the first edge to the first vertex of the second edge, and connecting the second vertex of the virst edge to the second vertex of the second edge. This technique is similar to a 2-opt exchange, but does not consider if the performance is improved by this exchange. Other more complicated variant, with more than two edges selected at random, are easily obtained in a similar fashion.

To further illustrate the perturbation step (step 2), consider the example in Figure 6. A sample obtained in step 1 is illustrated with the combination of solid and dotted lines. Then in step 2, we select the edges (C,E) and (D,A) at random, and replace them with the edges (C,D) and (E,A). The new edges are shown as dashed lines in Figure 6. Clearly this procedure provides us with a new sample point. Finally notice that since we consider only the symmetric TSP, the edges (E,D) and (D,E) are considered to be the same edge.

We call readers' attention to the fact that the weighted sampling schemes and the two-step sampling scheme provide a new construction heuristic for the TSP, which as far as we know has not been previously reported in the literature.

#### 4.2.2 Random Sampling with Knowledge-Based Clustered Partitioning

We now describe how sampling is implemented if we use the clustered partitioning scheme. We illustrate the technique by showing how to obtain a nearest neighbor sample for the example illustrated in Figure 3. Assume that the current promising region is the region indicated in Figure 4 and we want to find the pure nearest neighbor sample. This is a two step procedure. First we consider the edges that lead to a path in the promising region, and then we consider the remaining edges needed to make up a tour.

1. First compare the edges (A,B) and (A,E) which have cost 4 and 1 respectively. Hence we pick edge (A,E) as the cheaper edge. We now compare the two edges (E,B) and (E,D) which have cost 5 and 3 respectively. Hence we pick (E,D). This leads us to the path (A,E,D) which is an item in the promising region.

2. We now start with (A,E,D) and add the nearest neighbor edge until we have a feasible tour. In this case there is only one step left. We compare the edges (D,B) and (D,C) with cost 4 and 2 respectively, so we pick edge (D,C). Now since we have to visit all the cities this means that we visit B last and we have the tour (A,E,D,C,B,A) which has total cost of 12.

Some comments need to be made. It is clear that this procedure is easily generalized and it is very efficient since we have already sorted the adjacency list and we only have to look for the first feasible edge. This procedure shows how a pure nearest neighbor tour can be obtained from any given region. It is clear that a similar procedure can be used to obtain either a uniformly distributed sample or a weighted random sample.

### 4.3 Calculating the Promising Index

After selecting $N_j$ points from each subregion, by using the abovementioned sampling schemes, we need to define the *promising index*, $I(\sigma_j)$, that will be used to determine the most promising region. However, there exist many potential candidates for the function. For example, we may define

$$I(\sigma_j) = \min_{\theta \in \sigma_j} f(\theta), \quad j = 1, 2, ..., M+1, \tag{5}$$

and let $\hat{I}(\sigma_j)$ be an estimate of $I(\sigma_j)$

$$\hat{I}(\sigma_j) = \min_{i=1,2,...,N_j} f(\theta_i^j), \quad j = 1, 2, ..., M+1, \tag{6}$$

then $\hat{I}(\sigma_j)$ is the best performance within the $N_j$ points. We may also define

$$I(\sigma_j) = \frac{1}{|\sigma_j|} \sum_{\theta \in \sigma_j} f(\theta), \quad j = 1, 2, ..., M+1, \tag{7}$$

and let $\hat{I}(\sigma_j)$ be an estimate of $I(\sigma_j)$

$$\hat{I}(\sigma_j) = \frac{1}{N_j} \sum_{i=1}^{N_j} f(\theta_i^j), \quad j = 1, 2, ..., M+1, \tag{8}$$

then $\hat{I}(\sigma_j)$ is the average performance of the $N_j$ points.

Both promising index functions agree with $f(\cdot)$ on regions at maximum depth. They are, in general, different on other regions of less maximum depth. The promising index function is defined on the set $\Sigma$ which is a collection of subsets. This enables us to incorporate many effective heuristic methods into the NP algorithm when the current subregion is not in the maximum depth. That is, we can take the $N_j$ sampling points as initial points and for each of these sampling points, perform a fixed number of improvements based on a given heuristic method. For example, we can use the $k$-opt exchange heuristic to select the promising index function for the TSP. In the numerical experiments that follow, we use the following procedure to determine the promising index function.

Assume that we want to make $m$ improvements for each sampling point.

1. Make $m$ improvements for each point (tour) $\theta_i^j \in \sigma_j$ by using a 2-opt heuristic method.
2. Calculate the performance of each improved path.
3. Let the promising index be the best such performance.

Note that $m = 0$ implies that no heuristic improvements are made and $m \to \infty$ implies an exhausted heuristic search. Again, this approach guarantees the convergence property of the NP method.

### 4.4 Backtracking

The NP method provides great flexibility in selecting a backtracking rule. Perhaps the simplest rule would be to always move to the immediate superregion of the current most promising region. However many other alternatives exist. For example, we could backtrack all the way to the entire feasible region, or to any region that is in between the superregion and the entire feasible region. We can also consider the superregion of the best tour found in this iteration in the surrounding region.

To provide a systematic way for backtracking, in this paper we adopt the following approach. If the depth of current region is less than the maximum depth, then the algorithm backtracks to a superregion of the best solution found during this iteration. This superregion is determined such that it has less depth than the current region. For example, if the current most promising region is $(i_1^*, i_2^*, ..., i_k^*, x, x, ..., x)$ and the best solution, $(j_1^*, j_2^*, ..., j_n^*)$, is found in the surrounding region, then the next most promising region is determined to be $(j_1^*, j_2^*, ..., j_{k-h}^*, x, x, ..., x)$. If the current region is at maximum depth, then the algorithm

backtracks to a superregion of the best solution found in the current iteration. The superregion is determined by a predetermined depth $h > 0$ as before. For example, if the current region is at the maximum depth and the best solution, $(j_1^*, j_2^*, ..., j_n^*)$, is found in the surrounding region, then the next most promising region is determined to be $(j_1^*, j_2^*, ..., j_{n-h}^*, x, x, ..., x)$.

The advantage of the abovementioned backtracking rule is that if the ancestor of a better solution has a higher probability of containing the optimal tour than the current promising region, the algorithm will quickly identify the subregion which contains the optimal solution.

When describing the generic NP method, we assumed that the most promising region in iteration zero is the entire feasible region. This reflects the fact that at iteration zero there is no knowledge available about where good solutions might be found. However, if such knowledge is available, the NP method can use any other valid region $\sigma(0) \in \Sigma$ as the initial most promising region. The information leading to a specific initial region may for example come from previous numerical experience, but we can also go through an initialization phase to find such a region. In particular, for the TSP, we can use the nearest neighbor heuristic to quickly get a moderatly good tour. This tour can then be truncated to any specific length $k$, and the valid region defined by these first $k$ edges being fixed can be used as an initial most promising region.

The advantage of this is that the since a region of this depth only has $n-k$ subregions, the first iterations require much less computational effort than if we start at the entire feasible region and have to look at $n$ subregions.

# 5 Numerical Results

In this paper, we report our numerical results based on generic partitioning. Two sampling schemes are considered: *weighted sampling II* and the *two-step sampling*. We choose to use one of the simplest heuristics, the 2-opt exchanges, to incorporate into the promising index function, since this heuristic will be called repeatedly during the execution of the NP algorithm and it is desirable to have an overhead as low as possible.

## 5.1 Euclidean Problem with 51 cities

Our first example is the problem *eil51* from the TSPLIB library [11]. This is an Euclidean problem with 51 cities and a known optimal tour with cost 426. Due to the relative small size of this problem, we are able to try the NP algorithm for a variety of configurations.

For this experiment we use a SUN SPARCstation 20 for most of the settings.

The initial experiments focus on the *quality* of the solution as measured in percentage above the known minimum cost. The results are shown in Table 2 - Table 6.

Table 1 shows the results for the configurations that do not include any 2-opt exchanges in the promising index. The sampling procedure is the two step sampling scheme with weighted sampling II used to get the first sample. We see that the algorithm quickly gets to a solution that is about 10%-15% above the minimum cost, but seems to have trouble improving on these solutions. Furthermore, from Table 1 we see that increasing the sample size has very little effect on the performance if we use weighted sampling. If we use uniform sampling then there is clearly an effect of the number of samples.

We note that the computational effort of each iteration is, in general, dependent on the configuration, that is, the value of $p$, $N$ and in later experiments, how many improvements are made in the promising index. It is very intuitive why the number of samples ($N$) and the number of improvements effect the computational effort. What may not be as clear is that the weight ($p$) also has significant effect. The reason for this is that the number of times the algorithm retreats may depend on this weight. If the algorithm retreats frequently, it tends to stay at small depth and hence the most promising region has more subregions than if it was at a greater depth. Since more subregions implies more computational effort in each iteration, configurations where retreating is common, tend to use more computational effort in each iteration. Our experience indicates that configurations with small weight tend to retreat more than configurations with large weight.

Notice that the weighted sampling is uniformly better than the uniform sampling. We have to keep in mind, however that the weighted sampling requires more computational effort. Obtaining an uniform sample

takes $O(n)$ operations, but $O(n^2)$ operations are needed for the weighted sampling. Since sample size has very little effect for the weighted sampling, and the weighted sampling scheme is uniformly better, we let the sample size be fixed and equal to one in the remaining experiments.

Table 2 shows the results for when we incorporate 2-opt exchanges into the promising index. We observe that by using this promising indices, we get very quickly within 10% of the optimum and all of the configurations get within 6.3% in 300 iterations. We also see that varying $p$ from 0.90 to 0.99 has an effect on the objective function, particularly for the configurations where more 2-opt exchanges are performed. We also get a clear improvement in quality as we increase the number of 2-opt exchanges included in the promising index. This is to be expected, since this simply uses more computational effort. The best configurations easily find solutions that are around 1% above the global optimum.

It is clear from these results that in order to get within 1% of the optimum in less than three hundred iterations, it is necessary to incorporate the 2-opt exchanges into the promising index. Furthermore, weighted sampling outperforms uniform sampling, so we conclude that in order to obtain fast convergence of the NP algorithm, it is necessary to make intelligent choices in the sampling scheme and the promising index.

We point out that we have only used the simplest of weighted sampling schemes and the simplest improvement heuristic. This is done to minimize the computational effort. More elaborate sampling schemes and promising indices will undoubtedly improve the quality of the solutions, but will also increase the computational effort.

We repeated these experiments using the same configurations but starting at an initial most promising region that is at a depth ten above the maximum depth (41 in this case). This initial region was found by determining the nearest neighbor tour and truncating it to the desired length. The results are given in Table 3 and Table 4 below. We notice that if little effort is put into the promising index, then the results are slightly worse than before for fixed number of iterations. Note however, that the same number of iterations requires much less computational effort than before. If more computational effort is used in the promising index, the results are comparable to previous results, but with much less computational effort. We conclude that using such different initial most promising region is generally beneficial.

Finally, in Table 5 we have a comparison between using the same computational effort (path evaluations) in either *Crude Random Seedgeh* (CRS) or the NP method. We note that the NP method finds solutions that are generally 10-100 times better than the ones found by the CRS algorithm. More interestingly, we note that using more computational effort intelligently in the promising index and sampling of the NP method generates more than one hundred fold improvement, while increasing the computational effort by the same amount in CRS only improves the performance slightly.

### 5.2 Euclidean Problem with 417 cities

A much more challenging problem is the *fl417* problem from the TSPLIB library. Reineilt [12] reports the results in Table 6 for this problem.

Since replicating the previous experiment on a single workstation would be too time consuming for this problem, we used a distributed resource management system called *Condor* to submit these jobs to the pool of HP workstations available at the CAE facilities at UW-Madison. Condor is a distributed job scheduler that seeks out idle workstations and assigns waiting jobs to them. This system is therefore ideal for large and computational intensive jobs such as these experiments.

We build on the experience obtain from the *eil51* problem and use only weighted partitioning and use as the initial most promising region, a region of depth ten above the maximum depth. As before, this initial region is found using nearest neighbor search and truncation. The results can by found in Table 7 below.

## 6 Conclusions

We have shown how a new optimization methodology, the NP method, can be used to solve the traveling salesman problem. This method uses partitioning and random sampling to globally search the entire solution space, and can incorporate local search heuristic into a promising index that is used to determine where to concentrate the search. Since many powerful local heuristics have been devised for the TSP this is a very attractive property for this problem.

For each problem that the NP method is applied to we need to determine how to partition, how to sample, how to calculate the promising index, and how to retreat. For the TSP we presented a generic partitioning that can be applied to most combinatorial problems and a knowledge-based clustered partitioning that takes advantage of the special structure of the problem. We expect this partitioning to focus the computational effort better than the generic partitioning and intend to implement this partitioning method in the future.

In addition to the uniform sampling scheme that can be applied to all combinatorial problems, we presented two weighted sampling schemes that take into consideration the special structure of the TSP. Our numerical experience indicates that a simple version of the weighted sampling is superior to uniform sampling. We also proposed a two-step sampling scheme to increase the efficiency of either uniform or weighted sampling.

In our numerical experiments, we incorporated the 2-opt exchange local search heuristic into the promising index with good results. Future work will determine if it is beneficial to use more sophisticated improvement heuristics in the promising index.

We have shown that the NP method is a powerful alternative in solving combinatorial problems, such as the traveling salesman problem. It can be applied in its generic version, but also offer much flexibility in taking advantage of special structure. Another very attractive feature of the NP method is that it is parallel in nature and can therefore take maximum advantage of emerging parallel processing capabilities.

The numerical experiments indicate that the method is capable of finding good solutions quickly, and again there is much flexibility inherent in the method.

1. If time is very limited, then the NP method gives us immediately as good a solution as is obtained by a multi start 2-opt exchange heuristic.

2. If time is somewhat limited the NP method gives us very high quality solutions within a short time.

3. If time is virtually limitless the NP method converges to the optimal tour (asymptotic convergence).

It is also apparent that using the NP method has further benefits. It can equally be applied to stochastic problems, such as the *stochastic* TSP. It is readily transferable to other combinatorial problems and even to problems with a countable infinite feasible region. Future work will focus on more numerical experiments and implementing the knowledge-based clustered partitioning, as well as a parallel version of the algorithm.

# References


[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms,* Addison-Wesley, Reading, MA (1983).

[2] D. Applegate, R. Bixby, V. Chvátal and W. Cook, "Finding Cuts in the TSP (A preliminary report)," *DIMACS Technical Report* **95-05** (1995).

[3] G.S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications,* Springer, New York (1996).

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

[5] S. Kirkpatrick, C.D. Delatt Jr. & M.P.Vecchi, "Optimization by Simulated Annealing", *Science* **222**:671-680 (1983).

[6] J. Knox, "Tabu search performance on the symmetric traveling salesman problem," *Computers and Operations Research* **21**:867-876 (1994).

[7] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), *The Traveling Salesman Problem,* John Wiley & Sons, Chichester (1985).

[8] M. Malek, M. Guruswamy, H. Owens and M. Pandya, "Serial and Simulated Annealing and Tabu Seedgeh Algorithms for the Traveling Salesman Problem," *Annual of Operations Research* **21**:59-84 (1989).

[9] H. Muhlembein, M. Gorges-Schleuter and O. Kramer, "Evolution Algorithms in Combinatorial Optimization", *Parallel Computing* **7**:65-85 (1988).

[10] W.I. Norking, G.C. Pflug and A. Ruszczyński, "A Branch and Bound Method for Stochastic Global Optimization," *Working Paper,* International Institute for Applied System Analysis WP-96-065, Laxenburg, Austria (1997).

[11] G. Reinelt, "TSPLIB - A Traveling Salesman Problem Library," *ORSA J. Computing* **3**:376-384 (1992).

[12] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications*, Springer (1994).

[13] J.S. Rosenthal, "Convergence Rates for Markov Chains," *SIAM Review* **37**: 387-405 (1995).

[14] J.S. Rosenthal, "Minorization Conditions and Convergence of Markov Chain Monte Carlo," *Journal of the American Statistical Association* **90**: 558-566 (1995).

[15] D.E. Rumelhart, G.E. Hinton and J.L. McClelland, The PDP Research Group: Parallel Distributed Processing, *Explorations in the Microstructure of Cognition*, MIT Press (1996).

[16] L. Shi and S. Ólafsson, "Nested Partitions Method for Discrete Global Optimization," submitted for publication (1996).

[17] L. Shi and S. Ólafsson, "Nested Partitions Method for Stochastic Optimization," submitted for publication (1996).

[18] L. Shi and S. Ólafsson, "Convergence Rate of the Nested Partitions Method for Stochastic Optimization," *Working Paper*, Dept. of Ind. Engr., University of Wisconsin, Madison, WI (1997).

[19] N.L.J. Ulder, E. Pesch, P.J.M. van Laarhoven, H.-J. Bandelt and E.H.L. Aart, "Improving TSP exchange Heuristics by Population Genetics", *Research Report,* Erasmus Universiteit Rotterdam (1990).

[20] C.K. Wong and M.C. Easton, "An efficient method for weighted sampling without replacement," *SIAM Journal of Computing* **9**: 111-113 (1980).

| Iterations | $p = 0.00$ | | | $p = 0.90$ | | |
|---|---|---|---|---|---|---|
| | $N = 1$ | $N = 10$ | $N = 100$ | $N = 1$ | $N = 10$ | $N = 100$ |
| 1 | 216% | 236% | 133% | 34.5% | 45.8% | 34.5% |
| 10 | 216% | 214% | 133% | 29.3% | 27.2% | 29.3% |
| 50 | 182% | 189% | 110% | 16.2% | 27.2% | 16.2% |
| 100 | 182% | 189% | 110% | 15.0% | 27.0% | 15.0% |
| 200 | 182% | 189% | 110% | 15.0% | 21.4% | 15.0% |
| 300 | 182% | 189% | 110% | 15.0% | 21.4% | 15.0% |
| Iterations | $p = 0.95$ | | | $p = 0.99$ | | |
| | $N = 1$ | $N = 10$ | $N = 100$ | $N = 1$ | $N = 10$ | $N = 100$ |
| 1 | 31.2% | 28.9% | 24.2% | 25.8% | 25.8% | 23.0% |
| 10 | 22.3% | 16.2% | 14.3% | 12.9% | 16.2% | 13.4% |
| 50 | 11.5% | 16.2% | 12.2% | 11.0% | 11.7% | 12.9% |
| 100 | 11.5% | 16.2% | 12.2% | 11.0% | 11.5% | 10.6% |
| 200 | 11.5% | 16.2% | 12.2% | 11.0% | 11.5% | 10.6% |
| 300 | 11.5% | 16.2% | 12.2% | 11.0% | 11.5% | 10.6% |

Table 1: Results for the problem *eil51* when no heuristic improvements are made in the promising index and we use the entire feasible region as the initial most promising region. The percentages indicate how much the solution is above the known optimal solution.

| Iterations | $\tilde{n}$ | | | | $2\tilde{n}$ | | | |
|---|---|---|---|---|---|---|---|---|
| | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ |
| 1 | 29.3% | 18.3% | 12.2% | 13.8% | 18.3% | 9.2% | 8.5% | 8.2% |
| 10 | 27.7% | 6.6% | 10.8% | 9.6% | 13.1% | 5.4% | 7.0% | 8.2% |
| 50 | 17.1% | 6.6% | 8.2% | 4.9% | 7.3% | 3.5% | 5.2% | 5.6% |
| 100 | 17.1% | 6.6% | 8.2% | 4.5% | 5.9% | 3.1% | 4.9% | 2.8% |
| 200 | 17.1% | 5.4% | 7.3% | 4.5% | 5.9% | 3.1% | 3.5% | 2.8% |
| 300 | 17.1% | 5.4% | 6.3% | 4.5% | 4.0% | 3.1% | 2.3% | 2.8% |
| Iterations | $3\tilde{n}$ | | | | $4\tilde{n}$ | | | |
| | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ |
| 1 | 16.0% | 7.7% | 8.5% | 7.5% | 11.3% | 6.1% | 7.7% | 7.5% |
| 10 | 8.0% | 6.1% | 3.8% | 7.5% | 6.3% | 6.1% | 5.9% | 4.9% |
| 50 | 4.7% | 3.5% | 3.8% | 1.2% | 4.5% | 2.3% | 4.2% | 1.4% |
| 100 | 3.5% | 3.1% | 3.3% | 1.2% | 3.5% | 2.3% | 2.8% | 1.4% |
| 200 | 2.8% | 2.8% | 3.3% | 1.2% | 3.5% | 2.3% | 2.8% | 1.4% |
| 300 | 2.8% | 2.8% | 2.1% | 1.2% | 3.5% | 2.3% | 2.3% | 1.4% |

Table 2: Results for the problem *eil51* when 2-opt exchange improvements are made in the promising index and we use the entire feasible region as the initial most promising region. The number of improvements is fixed to be $\tilde{n}$, $2\tilde{n}$, $3\tilde{n}$ or $4\tilde{n}$ where $\tilde{n}$ is the number of cities that have not been fixed in the current most promising region. The percentages indicate how much the solution is above the known optimal solution.

| Iterations | $\tilde{n}$ | | | | $2\tilde{n}$ | | | |
|---|---|---|---|---|---|---|---|---|
| | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ |
| 1 | 32.4% | 21.8% | 16.0% | 14.3% | 18.5% | 8.0% | 2.8% | 4.5% |
| 10 | 23.5% | 12.2% | 12.0% | 9.2% | 12.2% | 6.1% | 2.8% | 4.0% |
| 50 | 16.0% | 12.2% | 12.0% | 9.2% | 8.0% | 4.2% | 2.8% | 2.1% |
| 100 | 16.0% | 10.3% | 12.0% | 6.8% | 7.5% | 4.2% | 2.8% | 2.1% |
| 200 | 16.0% | 8.0% | 9.4% | 6.8% | 7.5% | 4.0% | 2.8% | 2.1% |
| 300 | 16.0% | 7.3% | 9.4% | 6.8% | 7.5% | 3.8% | 2.8% | 2.1% |
| Iterations | $3\tilde{n}$ | | | | $4\tilde{n}$ | | | |
| | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ |
| 1 | 15.5% | 7.5% | 5.4% | 8.7% | 13.6% | 7.3% | 10.0% | 9.2% |
| 10 | 9.9% | 4.9% | 5.4% | 2.8% | 3.5% | 2.6% | 7.5% | 6.6% |
| 50 | 4.5% | 4.7% | 3.5% | 2.8% | 3.5% | 2.6% | 1.2% | 2.8% |
| 100 | 4.5% | 4.5% | 2.1% | 2.8% | 3.5% | 2.6% | 1.2% | 1.9% |
| 200 | 4.0% | 3.5% | 2.1% | 1.4% | 2.6% | 2.6% | 0.9% | 1.9% |
| 300 | 4.0% | 2.8% | 2.1% | 1.4% | 2.6% | 2.3% | 0.9% | 1.9% |

Table 3: Results for the problem *eil51* when 2-opt exchange improvements are made in the promising index and we use a region of depth ten above the maximum depth as the initial most promising region. The number of improvements is fixed to be $\tilde{n}$, $2\tilde{n}$, $3\tilde{n}$ or $4\tilde{n}$ where $\tilde{n}$ is the number of cities that have not been fixed in the current most promising region. The percentages indicate how much the solution is above the known optimal solution.

| Iterations | $p = .00$ | $p = .90$ | $p = .95$ | $p = .99$ |
|---|---|---|---|---|
| 1 | 260% | 34.7% | 26.5% | 21.4% |
| 10 | 208% | 26.1% | 22.1% | 12.9% |
| 50 | 206% | 18.5% | 22.1% | 10.6% |
| 100 | 206% | 18.5% | 16.0% | 10.6% |
| 200 | 206% | 18.5% | 16.0% | 10.6% |
| 300 | 206% | 18.5% | 16.0% | 10.6% |

Table 4: Results for the problem *eil51* when no improvements are made in the promising index and we use a region of depth ten above the maximum depth as the initial most promising region. The percentages indicate how much the solution is above the known optimal solution.

| | 0 | | $\tilde{n}$ | | $2\tilde{n}$ | | $3\tilde{n}$ | | $4\tilde{n}$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | $p = .00$ | $p = .99$ | $p = .00$ | $p = .99$ | $p = .00$ | $p = .99$ | $p = .00$ | $p = .99$ | $p = .00$ | $p = .99$ |
| NP | 206% | 10.6% | 16.0% | 6.8% | 7.5% | 2.1% | 4.0% | 1.4% | 2.8% | 2.1% |
| 2-Opt | 32.9% | 42.7% | 10.6% | 13.6% | 11.3% | 11.3% | 11.0% | 12.2% | 5.6% | 10.8% |
| CRS | 217% | 208% | 167% | 178% | 178% | 173% | 173% | 172% | 166% | 163% |

Table 5: Comparison with Crude Random Search (CRS) and 2-opt exchange heuristic using the same number of path evaluations. The percentages indicate how much the solution is above the known optimal solution. We see that increasing the computational effort has a large effect when we apply the NP method, but increasing the computational effort by the same amount does not produce significant effect when we apply the CRS.

| Method | Performance |
|---|---|
| 2-opt exchange, random starting tour | 47.4 % |
| 3-opt exchange, random starting tour | 9.1 % |
| 3-opt exchange, nearest neighbor starting tour | 6.2 % |
| 2-opt exchange, nearest neighbor starting tour | 5.7 % |
| Lin-Kernighan, random starting tour | 3.1 % |
| Iterated Lin-Kernighan | 2.5 % |

Table 6: Results for the *fl417* problem obtain through various heuristics. The percentages indicate how much the solution is above the known optimal solution.

| Iterations | $\tilde{n}$ | | $2\tilde{n}$ | | $3\tilde{n}$ | | $4\tilde{n}$ | |
|---|---|---|---|---|---|---|---|---|
| | $p = .90$ | $p = .99$ | $p = .90$ | $p = .99$ | $p = .90$ | $p = .99$ | $p = .90$ | $p = .99$ |
| 1 | 28.0% | 18.8% | 15.3% | 14.4% | 12.4% | 12.2% | 11.4% | 9.1% |
| 10 | 26.9% | 13.5% | 11.6% | 11.4% | 10.2% | 8.5% | 10.4% | 8.5% |
| 50 | 25.6% | 11.9% | 10.0% | 11.4% | 7.3% | 8.2% | 7.0% | 8.5% |
| 100 | 25.6% | 11.9% | 9.2% | 9.3% | % | % | % | % |
| 200 | 25.6% | 11.9% | % | % | % | % | % | % |
| 300 | 24.5% | 11.9% | % | % | % | % | % | % |

Table 7: Results for the problem *fl417* when 2-opt exchange improvements are made in the promising index and we use a region of depth ten above the maximum depth as the initial most promising region. The number of improvements is fixed to be $\tilde{n}$, $2\tilde{n}$, $3\tilde{n}$ or $4\tilde{n}$ where $\tilde{n}$ is the number of cities that have not been fixed in the current most promising region. The percentages indicate how much the solution is above the known optimal solution.