MIT Sloan School of Management

Working Paper 4463-03
July 2003

Dynamic Programming Methodologies in Very Large Scale Neighborhood Search Applied to the Traveling Salesman Problem

Özlem Ergun and James B. Orlin

Two Dynamic Programming Methodologies in Very Large Scale Neighborhood Search Applied to the Traveling Salesman Problem

Özlem Ergun
Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30339
email: oergun@isye.gatech.edu

James B. Orlin
Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139
email: jorlin@mit.edu

Abstract:

We provide two different neighborhood construction techniques for creating exponentially large neighborhoods that are searchable in polynomial time using dynamic programming. We illustrate both of these approaches on very large scale neighborhood search techniques for the traveling salesman problem. Our approaches are intended both to unify previously known results as well as to offer schemas for generating additional exponential neighborhoods that are searchable in polynomial time. The first approach is to define the neighborhood recursively. In this approach, the dynamic programming recursion is a natural consequence of the recursion that defines the neighborhood. In particular, we show how to create the pyramidal tour neighborhood, the twisted sequences neighborhood, and dynasearch neighborhoods using this approach. In the second approach, we consider the standard dynamic program to solve the TSP. We then obtain exponentially large neighborhoods by selecting a polynomially bounded number of states, and restricting the dynamic program to those states only. We show how the Balas and Simonetti neighborhood and the insertion dynasearch neighborhood can be viewed in this manner. We also show that one of the dynasearch neighborhoods can be derived directly from the 2-exchange neighborhood using this approach.

Subject classifications:

Traveling Salesman: Very large scale neighborhood search for the TSP.

Heuristics: Very large scale neighborhood search for the TSP.

Dynamic Programming: Two DP methodologies for heuristic search for the TSP.

Section 1. Introduction

Neighborhood search is a practical method for efficiently finding "good" solutions to hard combinatorial optimization problems. Let $P = \min \{cx: x \in D\}$ be an instance of an optimization problem with cost vector c and feasible set D. Given a feasible solution x' in D, a neighborhood search algorithm has an associated neighborhood function N which identifies a subset, $N(x)$, of D as the "neighbors" of x under N.

A local search algorithm follows the following basic scheme. Start with a current feasible solution, say x, and iteratively replace the current solution with a neighbor y of the current solution with lower objective value. Continue until there is no neighbor with improved objective value, at which point the current solution is called *locally optimal*. There is a large literature on local search as well as extensions of local search including simulated annealing and tabu search. For an excellent reference on local search in combinatorial optimization, see Aarts and Lenstra (1997).

In very large-scale neighborhood (*VLSN*) search, the number of solutions in a neighborhood is very large (often exponential) with respect to the size of the input. As a rule of thumb, one expects to find better locally optimal solutions assuming that one can search a larger neighborhood efficiently. Unfortunately, for many very large neighborhoods, the search time may be much larger. There are a variety of techniques for efficiently searching neighborhoods in *VLSN* search. One general approach that has been successful in searching exponentially large neighborhoods in polynomial time has been dynamic programming. See Ahuja et al. (2002) and Deĭneko and Woeginger (2000) for surveys on these techniques, including a number of papers on *VLSN* search that employ dynamic programming.

Here, we present two different approaches for developing exponentially large neighborhoods that are searchable in polynomial time using dynamic programming. In so doing, we offer frameworks that encompass many of the existing results in the literature, as well as offer general methodologies for creating new neighborhoods that are searchable in polynomial time.

In the first approach, we consider instances in which a neighborhood is recursively defined. For the examples we provide, these neighborhoods can be efficiently searched via the dynamic programs that are naturally induced by the recursions. We illustrate this approach on pyramidal tours, as developed by Sarvanov and Doroshko (1981), on twisted neighborhoods as defined by Aurenheimer (1988) and described for *VLSN* search by Deĭneko and Woeginger (2000), and for dynasearch neighborhoods as introduced by Potts and van de Velde (1995). The properties of recursively defined neighborhoods are often easily established using mathematical induction. We describe the use of recursion in defining neighborhoods in detail in Sections 3 and 4.

The second approach starts with the standard dynamic program to solve a combinatorial optimization problem and restricts attention to a polynomially large subset V of states of the dynamic programming state space. When $|V|$ is polynomially bounded in the size of the input, the time to solve the dynamic program is also polynomial. In Sections 5 and 6, we give examples in which $|V|$ is polynomially bounded, and solving the dynamic programming recursion over V is equivalent, in a technical sense that we will make clear, to searching an exponentially large neighborhood. We illustrate this approach on dynasearch neighborhoods and extensions as well as on the Balas-Simonetti neighborhood.

The approach in our paper differs from the rollout approach of using dynamic programming for heuristics (see, Bertsekas, Tsitsiklis, C. Wu. 1997) in that our approach explicitly searches a well defined neighborhood, whereas their approach uses dynamic programming combined with heuristic search to find a good solution for a combinatorial optimization problem.

The results in this paper present unifying views of dynamic programming algorithms for searching neighborhoods for the traveling salesman problem. We note that Deĭneko and Woeginger (2000) presented permutation trees as a unifying concept for dynamic programming methods for searching exponentially large TSP neighborhoods. Some of the exponential neighborhoods we survey are not representable as permutation trees (such as the twisted sequences neighborhood and the Balas-Simonetti neighborhood), and concurrently there are other exponential neighborhoods that are naturally represented in terms of permutation trees that are not as easily represented within our framework. So, neither framework subsumes the other.

We view the contributions of this paper as follows:

1. We provide two alternative unifying frameworks for using dynamic programming for the TSP for very large scale neighborhood search: recursively defined neighborhoods and dynamic programming restrictions.
2. We provide basic theory for working with restricted dynamic programs. In particular, we show how to associate a neighborhood with each restriction of the canonical dynamic program for the *TSP*.
3. We provide a method of using a dynamic programming recursion to transform a neighborhood N into a possibly larger neighborhood N' that is called the "dynamic programming expansion" of N. In the case of the 2-exchange neighborhood, the dynamic programming expansion is exponentially large, but can be searched in polynomial time.
4. Often the description of a recursively defined neighborhood is both compact and elegant.

We believe that the constructs in this paper will help to open up new lines for future research.

1. We expect that similar frameworks can be applied for other combinatorial optimization problems as well when there is a dynamic program that is much more efficient than complete enumeration.
2. Our constructs suggest the possibility of automating searching a number of exponentially sized neighborhoods. In particular, if a neighborhood can be recursively defined, perhaps one can automatically derive the dynamic programming recursion from the recursive definitions for the neighborhood.

Usually, the measure of a neighborhood search technique for a combinatorial optimization problem is how it performs in practice. In our case, several neighborhoods discussed in this paper have already been empirically tested by other researchers, including the dynasearch neighborhoods (Potts and van de Velde 1995 and Congram 2000), variants of pyramidal tours neighborhoods (Carlier and P. Villon 1990), and the Balas-Simonetti neighborhood (Balas and Simonetti 2001), all with some success for the TSP. Dynasearch neighborhoods have been applied with success on other combinatorial optimization problems as well (Agarwal et al. 2003, Congram, Potts, and van de Velde 2002, Ergun, Orlin, and Steele-Feldman 2002).

Nevertheless, we believe that the value of the methodologies in this paper should ultimately be judged on three criteria: First, will they lead to newly discovered neighborhood search techniques that are effective for some combinatorial optimization problems? Second, will they be the first step in a process that ultimately results in software that helps users to design neighborhood search techniques? Third, will they lead to further developments in the theory of *VLSN* search?

In Section 2 of this paper, we present additional background as well as our notation and definitions. In Sections 3, 4, 5, and 6 we present techniques for constructing exponentially large neighborhoods that are searchable in polynomial time using dynamic programming. In Section 7, we present a summary and conclusions.

Section 2. Definitions and Background for the TSP

In this section we offer definitions and background for the Traveling Salesman Problem (TSP).

The Traveling Salesman Problem

The traveling salesman problem tries to find the minimum distance tour on *n cities* that are labeled 1, 2, …, *n*. Let the distance from city *i* to city *j* be $c(i, j)$. We represent a *tour* using a permutation $T \in S_n$, where S_n denotes the set of all permutations of $\{1, 2, \ldots, n\}$. The permutation $T = T(1), \ldots, T(n)$ refers to the tour in which the first city visited is $T(1)$, the next city visited is $T(2)$, and so on. The cost of the tour T is denoted as $c(T) = c(T(n), T(1)) + \sum_{i=1}^{n-1} c(T(i), T(i+1))$. We refer to a pair of consecutive cities of T (including $T(n)$, $T(1)$) as an *edge* of the tour T. We denote a sequence A of k cities as $A = \langle i_1, i_2, \ldots, i_k \rangle$. If $k = n$, we refer to the sequence A as a tour. The reverse of a sequence A of cities is the cities of A in reverse order, and is denoted as *Rev*(A). For example, $Rev(\langle i_1, i_2, \ldots, i_k \rangle) = \langle i_k, i_{k-1}, \ldots, i_1 \rangle$.

If $i \leq j$, we let $[i, j]$ be shorthand for the sequence $\langle i, i+1, \ldots, j \rangle$. If $i > j$, then $[i, j] = \varnothing$.

The subset obtained from subset S of cities by deleting any city in S' will be denoted as $S \setminus S'$. We abbreviate $S \setminus \{i\}$ as $S \setminus i$.

If S is a subset or sequence of cities, then $\max(S)$ denotes the maximum index of a city of S, and $\min(S)$ denotes the minimum index of a city of S.

As per Deĭneko and Woeginger (2000), if A is a sequence of cities, and B is a different sequence of cities with no city in common with A, then $A \bigstar B$ is the sequence obtained by concatenating A with B. For example, $\langle 3, 1, 7 \rangle \bigstar \langle 2, 6, 4 \rangle = \langle 3, 1, 7, 2, 6, 4 \rangle$.

Notation for Neighborhoods for the Traveling Salesman Problem

We first describe a neighborhood for the tour $T^I = \langle 1, 2, \ldots, n \rangle$, which is also the identity permutation. *A* neighborhood $N(T^I)$ is a collection of tours, that is $N(T^I) \subseteq S_n$.

Any neighborhood of T^I can be extended to a neighborhood of any other tour $T \in S_n$ as follows. If $\sigma \in N(T^I)$, then the tour $T \circ \sigma = \langle \sigma(T(1)), \sigma(T(2)), \ldots, \sigma(T(n)) \rangle \in N(T)$. Mathematically, $T \circ \sigma$ is the permutation obtained by composing T and σ. If we let $f_T(i, j) = c(T(i), T(j))$, then the cost of the tour

$T \circ \sigma$ is $f_T(\sigma(n),\sigma(1)) + \sum_{i=1}^{n-1} f_T(\sigma(i),\sigma(i+1))$, which is the usual formula for the cost of σ except that c is replace by f_T.

In subsequent sections, we assume that the initial tour is $T^I = \langle 1, \ldots, n \rangle$, and we provide dynamic programs for determining the minimum distance neighbor of T^I. However, if one wants to search $N(T)$ instead, then it suffices to replace c by f_T in the recursions.

In general, we are treating neighborhoods for different sized *TSPs*. Sometimes, when the number of cities n is permitted to vary, we let N_n denote the neighborhood set for problems with n cities. In the case that the number of cities is obvious from context, we drop the index, and denote the neighborhood set as N. We assume that the identity permutation $T^I \in N_n$, that is $\langle 1, 2, \ldots, n \rangle \in N_n$ for all n.

All of our recursions define both tours as well as sequences of fewer than n cities. We let N^* refer to all sequences and tours defined by the recursion, and we let $N = N^* \cap S_n$ denote the neighborhood. We will refer to the set N^* as a *superneighborhood*.

Section 3. Recursively Defined Neighborhoods

In this section, we discuss recursive definitions as part of *VLSN* search for the *TSP*. In particular, we will provide recursive definitions for well known exponentially sized neighborhoods together with dynamic programs based on the recursions for searching these neighborhoods efficiently. We illustrate our approach on one of the dynasearch neighborhood as well as some "relatives" of this neighborhood, the pyramidal tour neighborhood, and the twisted sequences neighborhood.

As above, we will let N denote a neighborhood, and let N^* denote all sequences created by the recursions in the construction of N. In each case, rather than present a dynamic program, we will create an equivalent state space graph. Our objective in so doing is to make explicit a very clear connection between the recursion that defines the neighborhood and the resulting dynamic program.

To illustrate our recursion-based approach, we first consider a recursion for generating the neighborhood S_n, that is the neighborhood of all tours. The superneighborhood N^* that we generate will consist of all sequences of cities in which the first city is 1.

The Held and Karp (1962) dynamic program for the TSP can be viewed as being based on the following recursion:

The Complete Neighborhood N_{TSP} for the TSP.

1. $1 \in N^*_{TSP}$.

2. If $A \in N^*_{TSP}$ and $k \notin A$, then $A \bigstar k \in N^*_{TSP}$.

3. $N_{TSP} = N^*_{TSP} \cap S_n$.

The state space graph corresponding to the dynamic program for the *TSP* can be obtained as follows. It includes a node for each state of the dynamic program as well as a special destination node t.

State Space Graph $G_{TSP} = (V_{TSP}, E_{TSP})$.

1. $(\{1\}, 1) \in V_{TSP}$ and $t \in V_{TSP}$.

2. If $(S, j) \in V_{TSP}$, and if $k \notin S$, then $(S \cup \{k\}, k) \in V_{TSP}$, and there is an arc from (S, j) to $(S \cup \{k\}, k)$ in E_{TSP} with cost $c(j, k)$.

3. If $(S, j) \in V_{TSP}$ and if $|S| = n$, then there is an arc from (S, j) to t with cost $c(j, 1)$.

Let $g_{TSP}(S, j)$ denote the optimal value for state $(S, j) \in V_{TSP}$ in the Held and Karp dynamic program for the TSP. Then $g_{TSP}(S, j)$ is the shortest length of a sequence A whose initial city is 1, whose terminal city is j, and such that A includes all of the cities of S.

The following are well known properties of the state space graph and its relation to the Held and Karp dynamic program DP_{TSP}.

1. The shortest path from node $(\{1\}, 1) \in V_{TSP}$ to node (S, j) has length $g_{TSP}(S, j)$.
2. Any shortest path from node $(\{1\}, 1) \in V_{TSP}$ to node t corresponds to a minimum length tour. If we let (S_j, i_j) denote the j-th node of V_{TSP} on the shortest path, then the shortest length tour is $\langle i_1, \ldots, i_n \rangle$.

Properties 1 and 2 above extend to the other dynamic programming state space graphs as defined in this section and the next.

The Independent Compounded 2-Exchange Neighborhood: Dynasearch

We next present a class of exponentially sized neighborhoods developed in Potts and van de Velde (1995), Congram (2000) and Congram, Potts, and van de Velde (2002) that is obtained from the 2-exchange neighborhood by compounding sets of independent 2-exchanges.

We let $h(i, j)$ be the cost associated with sequence $[i, j] = \langle i, i+1, \ldots, j \rangle$. Thus $h(i, j) = \sum_{k=i}^{j-1} c(k, k+1)$. One can first compute $h(1, j)$ and $h(j, n)$ for all j in $O(n)$ steps. Subsequently computing $h(i, j) = h(1, n) - h(j+1, n) - h(1, i-1)$ takes O(1) additional steps.

Recall that for $i \le j$, $Rev[i, j]$ denote the sequence $\langle j, j-1, \ldots, i \rangle$. We let $RevMove[i, j]$ be the move that reverses the orders of cities in positions i to j. For example, if we apply $RevMove[i, j]$ to the identity permutation T^I, we obtain $\langle 1, \ldots, i-1 \rangle \bigstar Rev[i, j] \bigstar \langle j+1, \ldots, n \rangle$. If we apply $RevMove[3, 4]$ to $\langle 1, 5, 2, 3, 4 \rangle$, we would obtain $\langle 1, 5, 3, 2, 4 \rangle$.

A *2-exchange* of a tour T is a permutation obtained from T by the operation $RevMove[i, j]$ for some $i < j$. Equivalently, the 2-exchange of the tour $\langle 1, \ldots, n \rangle$ can be viewed as breaking edges $(i-1, i)$ and $(j, j+1)$ and adding edges $(i-1, j)$ and $(i, j+1)$. We say that two 2-exchanges $RevMove[i_1, j_1]$ and $RevMove[i_2, j_2]$ are *independent* if $j_1 < i_2 - 1$ or $j_2 < i_1 - 1$. If $j_1 < i_2$ or $j_2 < i_1$, we say that the two 2-exchanges are *weakly independent*. Potts and van de Velde (1995), and Congram, Potts, and van de Velde (2002) introduced neighborhoods based on compounding (or applying) independent moves under the name "dynasearch", a term which we use here as well. We refer to the neighborhood obtained from T^I by compounding independent 2-exchanges as the *2-exchange dynasearch neighborhood*. We refer to the neighborhood obtained by compounding weakly independent 2-exchanges as the *weak 2-exchange dynasearch neighborhood*.

Exponential neighborhoods based on compounding moves were designed and computationally tested on classes of single and parallel machine scheduling and vehicle routing problems as well as the traveling salesman problem and the linear ordering problem (Agarwal et al. 2003, Congram 2000, Congram, Potts, and van de Velde 2002, Ergun 2001, Ergun, Orlin, and Steele-Feldman 2002).

The size of the compounded independent moves neighborhood is $\Omega(1.7548^n)$, as may be computed from a simple recursion. See Congram (2000) and Ergun (2001) for derivations. The dynasearch neighborhoods can be searched in $O(n^2)$ by dynamic programs as in Potts and van de Velde (1995), Congram (2000), Congram, Potts, and van de Velde (2002), and by network flows techniques as in Agarwal et al. (2003), Ergun (2001), Ergun, Orlin, and Steele-Feldman (2002) as well as by the approach given in this section.

The approach developed here to define the neighborhood is different from the approach developed in the original papers on dynasearch; however, the dynamic programming recursions developed here are very similar.

The 2-Exchange Dynasearch Neighborhood N_{DS} for the *TSP*.

1. $1 \in N_{DS}^*$.

2. If $A \in N_{DS}^*$ and $\max(A) = i < n$, then

a. $A \star i{+}1 \in N_{DS}^*$, and

b. $A \star Rev[i{+}1, j{-}1] \star j \in N_{DS}^*$ for $i{+}3 \le j \le n$, and

c. $A \star Rev[i{+}1, n] \in N_{DS}^*$.

3. $N_{DS} = N_{DS}^* \cap S_n$.

The state space graph for the dynasearch neighborhood is constructed in a similar manner to the graph G_{TSP}.

State Space Graph $G_{DS} = (V_{DS}, E_{DS})$.

1. $(\{1\}, 1) \in V_{DS}$ and $t \in V_{DS}$.

2. Suppose that $(S, i) \in V_{DS}$, and $i < n$. Then

a. $(S \cup \{i{+}1\}, i{+}1) \in V_{DS}$, and there is an arc from (S, i) to $(S \cup \{i{+}1\}, i{+}1)$ in E_{DS} with cost $c(i, i{+}1)$,

b. for $i{+}3 \le j \le n$, $(S \cup \{i{+}1, \ldots, j\}, j) \in V_{DS}$, and there is an arc from (S, i) to $(S \cup \{i{+}1, \ldots, j\}, j)$ in E_{DS} with cost $c(i, j-1) + h(i+1, j-1) + c(i+1, j)$,

c. $(\{1, \ldots, n\}, i{+}1) \in V_{DS}$, and there is an arc from (S, i) to $(\{1, 2, \ldots, n\}, i{+}1)$ in E_{DS} with a cost of $c(i, n) + h(i{+}1, n)$.

3. If $(S, j) \in V_{DS}$ and if $|S| = n$, there is an arc from (S, j) to t with cost $c(j, 1)$.

As before, an optimal tour in N_{DS} can be obtained by finding the shortest path in G_{DS} from node $(\{1\}, 1)$ to node t.

Theorem 1. The set $N_{DS} = N^*_{DS} \cap S_n$ is the 2-exchange dynasearch neighborhood. The corresponding state space graph has O(n) nodes and O(n^2) edges. The time to find a minimum distance neighbor is O(n^2).

Proof. Suppose first that σ is in the 2-exchange dynasearch neighborhood and that σ is obtained by k independent two exchanges. We prove the result by induction on k. If k = 0, then $\sigma = [1, n] \in N_{DS}$. Suppose instead that the compounded independent 2-exchanges to obtain σ are $RevMove[i_{2j-1}, i_{2j}]$ for j = 1 to k for some $k \geq 1$, and where indices are in increasing order. Then

$$\sigma = [1, i_1\text{-}1] \star Rev[i_1, i_2] \star [i_2\text{+}1, i_3\text{-}1] \star Rev[i_3, i_4] \star [i_4\text{+}1, i_5\text{-}1] \star \ldots \star Rev[i_{2k\text{-}1}, i_{2k}] \star [i_{2k}\text{+}1, n].$$

Recall that $[i, j] = \varnothing$ for $i > j$.

It is easy to see that σ can be obtained by the recursion for N_{DS} by starting with city 1, and at each application of Step 2 in the construction of σ, concatenating a single node or else concatenating $Rev[i_{2j-1}, i_{2j}] \star i_{2j}+1$ for some j, or else appending $Rev[i_{2k-1}, i_{2k}]$ if $i_{2k} = n$.

Conversely if $\sigma \in N_{DS}$, then for some k

$$\sigma = [1, i_1\text{-}1] \star Rev[i_1, i_2] \star [i_2\text{+}1, i_3\text{-}1] \star Rev[i_3, i_4] \star [i_4\text{+}1, i_5\text{-}1] \star \ldots \star Rev[i_{2k\text{-}1}, i_{2k}] \star [i_{2k}\text{+}1, n].$$

In this case, σ is obtained by compounding $RevMove[i_{2j-1}, i_{2j}]$ for j = 1 to k, and thus σ is in the 2-exchange dynasearch neighborhood.

Let $V' = \{(\{1, \ldots, j\}, j) : j = 1 \text{ to } n\} \cup \{(\{1, \ldots, n\}, j) : j = 2 \text{ to } n\text{-}1\}$. We now claim that $V_{DS} = V'$.

We first note that it is easily established that $V' \subseteq V_{DS}$. We next assume that $(S, i) \in V_{DS}$, and we will show that $(S, i) \in V'$. Suppose $|S| > 1$, and it was created by applying Step 2 to some state (S', i'). Moreover, $|S'| < |S|$, and so we assume inductively that $(S', i') \in V'$, and so $S' = \{1, \ldots, i'\}$. It is easily verified that if 2a, or 2b or 2c is applied to (S', i'), the resulting state $(S, i) \in V'$, thus showing that $V_{DS} \subseteq V'$. It also follows that each state (S, j) can be represented in O(1) space by the pair (i, j) where $S = \{1, \ldots, i\}$.

Thus $|V_{DS}| = O(n)$ and $|E_{DS}| = O(n^2)$, and the time to create G_{DS} is O(n^2) since we can compute $h(i, j)$ in O(1) steps after an initial pre-computation. The running time to find the shortest path from node $(\{1\}, 1)$ to node t is O(n^2) since G_{DS} is acyclic, completing the proof. (See, for example, Ahuja, Magnanti, and Orlin (1993) for information about shortest path algorithms.) ♦

In the proof, we commented on the storage space of a state. This storage is important because in creating G_{DS} or any other state-space graph, we need to know whether a node (S, j) derived from rule 2 is already present in the graph. If it is already present, we need to be able to identify it in O(1) steps. For each of the neighborhoods considered in this paper, except the Held-Karp dynamic program, one can store each state in O(1) space, and identify its presence in O(1) time.

Weakly Independent Compounded 2-exchanges

In this subsection, we base dynasearch on the relaxed concept of weak independence in order to develop a larger neighborhood that is also searchable in polynomial time.

The Weak 2-Exchange Dynasearch Neighborhood N_{WDS} for the TSP

1. $1 \in N^*_{WDS}$.

2. If $A \in N^*_{WDS}$ and $\max(A) = i$, then $A \bigstar Rev[i+1, j] \in N^*_{WDS}$ for $i+1 \le j \le n$.

3. $N_{WDS} = N^*_{WDS} \cap S_n$.

This neighborhood is more straightforward to define than the dynasearch neighborhood, as is its state space graph.

The State Space Graph $G_{WDS} = (V_{WDS}, E_{WDS})$:

1. $(\{1\}, 1) \in V_{WDS}$.

2. Suppose $(S, k) \in V_{WDS}$, and let $\max(S) = i < n$. Then for each j with $i < j \le n$, $(S \cup \{i+1, \ldots, j\}, i+1) \in V_{WDS}$, and there is an arc from (S, k) to $(S \cup \{i+1, \ldots, j\}, i+1)$ in E_{WDS} with cost $c(k, j) + h(i+1, j)$.

3. If $(S, j) \in V_{WDS}$ and if $|S| = n$, there is an arc from (S, j) to t with cost $c(j, 1)$

Theorem 2. The set $N_{WDS} = N^*_{WDS} \cap S_n$ is the weak 2-exchange dynasearch neighborhood. The corresponding state space graph G_{WDS} has $O(n^2)$ nodes and $O(n^3)$ edges. The time to find a minimum distance neighbor is $O(n^3)$.

Proof. We first show that the weak 2-exchange dynasearch neighborhood is contained in N_{WDS}. Suppose that σ can be obtained by compounding weakly independent 2-exchanges. By permitting null operations such as $RevMove[i, i]$, σ can be obtained by performing $RevMove[i_{2j-1}, i_{2j}]$ for $j = 1$ to k, where (1) $i_1 = i_2 = 1$, (2) $i_{2k} = n$, and (3) $i_{2j+1} = i_{2j} + 1$ for $j = 1$ to k-1. It follows that σ can be obtained by starting with 1, and concatenating $Rev[i_{2j-1}, i_{2j}]$ for $j = 2$ to k, and thus $\sigma \in N_{WDS}$. Conversely, if $\sigma \in N_{WDS}$ and can be obtained by starting with 1 and concatenating a sequence of reversals, then it is clear that σ can be obtained by compounding weakly independent 2-exchanges.

We next claim that for any state (S, j) with $\max(S) = i$, it follows that $S = \{1, \ldots, i\}$. The claim is clearly true if $i = 1$, and it follows directly by induction because of Step 2 of the construction of G_{WDS}. Moreover, the state $(\{1, \ldots, i\}, j)$ is obtainable for $2 \le j \le i$ by starting with $\{\{1, \ldots, i\text{-}1\}, i\text{-}1\}$ and then applying Step 2 of the construction of G_{WDS}. The only state (S, j) with $j = 1$ is $(\{1\}, 1)$. We conclude that

$$V_{WDS} = \{(\{1, \ldots, \mathrm{i}\}, k) : \text{for } 1 \le k \le \mathrm{i} \le n\}.$$

Thus there are $O(n^2)$ nodes, and $O(n)$ arcs emanating from each by the construction in Step 2 of G_{WDS}, leading to $O(n^3)$ arcs. The time to create G_{WDS} is $O(n^3)$ since we can compute $h(i, j)$ in O(1) steps after an initial pre-computation of $h(1, j)$ for $j = 1$ to n, and we can create each arc in O(1) step assuming that we store $\max(S)$ for each state (S, j). The running time to find the shortest path from node 1 to all other nodes is $O(n^3)$ since we are solving the shortest path problem on an acyclic graph. This completes the proof. ♦

Other Dynasearch Neighborhoods.

The results of the previous sections on dynasearch applied to independent and weakly independent 2-exchanges can be extended to other neighborhoods as well. For example, let Insert[i, j] = [i+1, j] ★ i , and let *InsertMove*[*i*, *j*] be the move that takes the city in position i and inserts it directly after city j. For example, if we apply *InsertMove*[*i*, *j*] for $1 < i < j < n$ to the tour $\langle 1, \ldots, n \rangle$, we obtain the sequence [1, *i*-1] ★ *Insert*[*i*, *j*] ★[*j*+1, *n*]. Similarly, we can define *Swap*[*i*, *j*] = *j* ★ [*i*+1, *j*-1] ★ *i*, and we can define *SwapMove*[*i*, *j*] to be the move that swaps the cities in positions i and j of a tour. The insert neighborhood is the neighborhood of $\langle 1, \ldots, n \rangle$ obtained by permitting at most one InsertMove, and the swap neighborhood is obtained by permitting at most one SwapMove. We let the *composite neighborhood* be the neighborhood obtained by permitting at most one 2-exchange or InsertMove or SwapMove.

We can define independent and weakly independent InsertMoves, SwapMoves, and composite moves as before and modify the dynasearch and the weak dynasearch neighborhoods accordingly. In general, one would expect the dynasearch neighborhood to require $O(n^2)$ time to search, and the weak dynasearch neighborhood would require $O(n^3)$ time. However, the weak insertion dynasearch neighborhood can be searched even more efficiently when it is formulated with dynamic programming restrictions as we will see in Section 5.

Section 4. Other recursively defined neighborhoods

In this section, we consider recursive ways for defining pyramidal tours and the twisted sequences neighborhood.

In the dynasearch neighborhood of Section 3 as well as in the standard dynamic program for the traveling salesman problem, every single tour began with the city 1, and the state (*S*, *j*) referred to tours that started with city 1, ended with city *j*, and visited all of the cities in *S*. In this section, we no longer assume that the initial city in a sequence is city 1. Rather, we let (*i*, *S*, *j*) refer to sequences *A* such that the first city in *A* is *i*, the last city in *A* is *j*, and *A* includes all of the cities in *S*.

The Pyramidal Tours Neighborhood

The pyramidal tours neighborhood for the *TSP* was first introduced by Sarvanov and Doroshko (1981). We say that a sequence *A* (which is not necessarily a tour) is *pyramidal* if it consists of cities $\{i, i+1, \ldots, n\}$ for some $i \geq 1$, and if the cities in *A* increase in index to city *n*, and then decrease in index. For example, if $n = 7$, then $\langle 4, 7, 6, 5, 3 \rangle$ is pyramidal.

The pyramidal neighborhood consists of all pyramidal tours. For example, if $n = 7$, then the tour $\langle 3, 4, 7, 6, 5, 2, 1 \rangle$ is in the pyramidal neighborhood. Our definition differs slightly from the usual definition of pyramidal tour in that we do not require the first city to be city 1; that is, city 1 may be the last city instead. Thus, our definition includes all the usual pyramidal tours as well as their reversals. We use this slightly non-standard definition in order to simplify the description of the recursion, but it would be easy to use the original definition as well. The size of the pyramidal neighborhood is 2^n.

Note that the edges (1, 2) and (*n*-1, *n*) of the original tour belong to all of its neighbors. This is a drawback in using the pyramidal tour neighborhood in practice. To circumvent this drawback, Carlier

and Villon (1990) consider all n rotations associated with a given tour, and search the pyramidal neighborhoods of each of these rotations. The size of this composite neighborhood is $\theta(n2^n)$ and it can be searched in $O(n^3)$ time by a dynamic program described in Carlier and Villon (1990) or using n iterations of a shortest path algorithm on an appropriately created auxiliary graph (see Ahuja et al. (2002) for details), or using the n iterations of the procedure listed below.

We now present a recursion which defines the pyramidal neighborhood, as well as its corresponding state space graph.

The Pyramidal Tours Neighborhood N_{PT} for the TSP.

1. $n \in N_{PT}^*$.

2. If $A \in N_{PT}^*$ and $\min(A) = i$, then

a. $A \bigstar i\text{-}1 \in N_{PT}^*$ and

b $i\text{-}1 \bigstar A \in N_{PT}^*$.

3. $N_{PT} = N_{PT}^* \cap S_n$.

The State Space Graph $G_{PT} = (V_{PT}, E_{PT})$.

1. $(n, \{n\}, n) \in V_{PT}$.

2. Suppose $(j, S, k) \in V_{PT}$, and let $\min(S) = i > 1$. Then

a. $(j, S \cup \{i\text{-}1\}, i\text{-}1) \in V_{PT}$, and there is an arc from (j, S, k) to $(j, S \cup \{i\text{-}1\}, i\text{-}1)$ in E_{PT} with a cost of $c(k, i\text{-}1)$.

b. $(i\text{-}1, S \cup \{i\text{-}1\}, k) \in V_{PT}$, and there is an arc from (j, S, k) to $(i\text{-}1, S \cup \{i\text{-}1\}, k)$ in E_{PT} with a cost of $c(i\text{-}1, j)$.

3. If $(j, S, k) \in V_{PT}$ and if $|S| = n$, there is an arc from (j, S, k) to t with cost $c(k, j)$.

Theorem 3. The superneighborhood N_{PT}^* consists of all pyramidal sequences, and the neighborhood N_{PT} is the pyramidal tour neighborhood. The corresponding state space graph G_{PT} has $O(n^2)$ nodes and $O(n^2)$ edges. The time to find a minimum distance neighbor is $O(n^2)$.

Proof. We first claim that for $A \in N_{PT}^*$, A is pyramidal. It is true if $|A| = 1$, since in that case $A = \langle n \rangle$. We now consider A' and assume inductively that the claim is true for all sequences with fewer cities than A'. The sequence A' is created in Step 2 of the recursion, and either (i) $A' = A \bigstar i\text{-}1$ or (ii) $A' = i\text{-}1 \bigstar A$, where $A \in N_{PT}^*$ and $\min(A) = i$. By inductive hypothesis A is pyramidal and thus A consists of cities $\{i, \ldots, n\}$, and the sequence in A increases in index to city n and then decreases in index. It follows that A' is also pyramidal.

We next establish that every pyramidal sequence is in N_{PT}^*. So suppose that A is pyramidal, and let A_j be A as restricted to cities $\{j, j+1, \ldots, n\}$. Since A is pyramidal, it follows that A_j is also pyramidal for each j. We assume inductively that $A_{j+1} \in N_{PT}^*$ for some $j < n$. It is easily verified that $A_j = j \bigstar A_{j+1}$ or else $A_j = A_{j+1} \bigstar j$. It follows that $A_j \in N_{PT}^*$, completing the proof of the first sentence of the theorem.

We now establish that the state space graph has $O(n^2)$ nodes. Suppose that (i, S, j) is a state. Then $S = (k, k+1, \ldots, n\}$ for some k and $k = i$ or j. It follows that there are $O(n)$ different sets for S and $O(n)$ ways that the pair (i, j) can be chosen for a specified set. And for each state (i, S, j), the state can be represented by the triple $(i, \min(S), j)$, which takes O(1) space, and can be recognized in O(1) time.

Because the nodes (i, S, j) and (j, S, i) for $S = \{i, i+1, \ldots, n\}$ both have two arcs emanating, we conclude that $|E_{PT}| = O(n^2)$, and G_{PT} can be created in $O(n^2)$ time. Moreover, the optimal tour in N_{PT} is induced by the shortest path from node $(n, \{n\}, n\}$ to node t, which can be obtained in $O(n^2)$ steps, completing the proof. ♦

The Twisted Sequences Neighborhood

We once again consider 2-exchanges, each of which is denoted by an operation *RevMove*$[i, j]$ for some i and j with $1 < i < j \leq n$. Let R be an ordered set of 2-exchanges on a tour on n cities. For example, suppose that $n = 9$, and $R = \{$*RevMove*[3, 5], *RevMove*[2, 7], *RevMove*[4, 6]$\}$. Applying the operations in R is to carry out the reversals in the order that they appear in R. For example, applying *RevMove*[3, 5] to $A_0 = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9\rangle$ would lead to the sequence $A_1 = \langle 1, 2, 5, 4, 3, 6, 7, 8, 9\rangle$. Applying *RevMove*[2, 7] to A_1 would lead to the sequence $A_2 = \langle 1, 7, 6, 3, 4, 5, 2, 8, 9\rangle$ since it reverses the positions of the elements currently in positions 2 to 7. Applying *RevMove*[4, 6] to A_2 leads to the sequence $A_3 = \langle 1, 7, 6, 5, 4, 3, 2, 8, 9\rangle$.

We say that one reversal *RevMove*$[j_1, k_1]$ *is contained in RevMove*$[j_2, k_2]$ if $[j_1, k_1] \subseteq [j_2, k_2]$. We say that a subset S of cities is *consecutive* if $S = \{i, i+1, \ldots, j\}$ for some $1 \leq i \leq j \leq n$.

Aurenhammer (1988) defined twisted sequences in the context of sorting of data. We provide a definition that Aurenhammer shows is equivalent to his original definition (modulo he specified the operations slightly differently.). We say that a sequence A on a consecutive set $A = \{i, i+1 \ldots, j\}$ of cities is a *twisted sequence* if it can be obtained by carrying out a sequence R of 2-exchanges on the permutation $\langle i, i+1\ldots, j\rangle$ such that the following two properties hold:

TS Property 1. Each 2-exchange *RevMove*(k, l) of R is contained in $[i, j]$.
TS Property 2. If *RevMove*(k, l) precedes *RevMove*(k', l') in R, then either the two moves are weakly independent or else, *RevMove*(k, l) is contained in *RevMove*(k', l').

For a sequence R of 2-exchanges satisfying the two *TS* Properties, the same tour is obtained regardless of the order in which the moves in R are carried out. However, we shall assume that they are carried out in the order they appear in R.

This definition is equivalent to the definition given by Aurenhammer in the case that the sequence consists of all cities in $\{1, 2, \ldots, n\}$. We note that the sequence A_3 obtained from R above is a twisted sequence because A_3 can be obtained by a single reversal even though R does not satisfies the properties 1 and 2 above.

Aurenhammer showed that the twisted sequences neighborhood contains at least $\Omega(2^n)$ and at most $O(c^n)$ tours for some constant c. Deĭneko and Woeginger (2000) claimed that the twisted sequences neighborhoods contains $O(6^n)$ tours. They also presented a dynamic program which finds the best tour in

the twisted sequences neighborhood in $O(n^7)$ time. Congram (2000) proved that the size of the neighborhood grows asymptotically $(3+\sqrt{3})^n$. In particular, it is $\Omega(5.8284^n)$ and $O(5.8285^n)$.

We now present a recursion which defines the twisted sequences neighborhood.

The Twisted Sequences Neighborhood N_{TS} for the TSP.

1. $j \in N_{TS}^*$ for each $j = 1$ to n.

2. If $A \in N_{TS}^*$, $B \in N_{TS}^*$, and $\max(A) = \min(B) - 1$, then $A \bigstar B \in N_{TS}^*$ and $\text{Rev}(A \bigstar B) \in N_{TS}^*$.

3. $N_{TS} = N_{TS}^* \cap S_n$.

Before presenting the state space graph for the twisted sequences neighborhood, we establish that N_{TS}^* is the set of all twisted sequences, and N_{TS} is the twisted sequences neighborhood.

Theorem 4. The superneighborhood N_{TS}^* consists of all twisted sequences, and the neighborhood N_{TS} is the twisted sequences neighborhood.

Proof. We first note for any sequence $A \in N_{TS}^*$, the set of cities in A is consecutive. This is easily established via induction on the number of operations needed to create A. We also observe that by Rule 2 in the creation of N_{TS}^*, if $A \in N_{TS}^*$, then $\text{Rev}(A) \in N_{TS}^*$.

We next claim that if $A' \in N_{TS}^*$, then A' is a twisted subsequence. The claim is clearly true if $A' = \langle j \rangle$ for some j. We assume inductively that the claim is true for sets $A \in N_{TS}^*$ with $|A| < |A'|$. By construction, $A' = A \bigstar B$ or else $A' = Rev(A \bigstar B)$, where $A \in N_{TS}^*$ is defined on cities $\{i, \ldots, j\}$ for some i and j, and $B \in N_{TS}^*$ is defined on cities $\{j+1, \ldots, k\}$. By inductive hypothesis, there are sets of 2-exchanges R_A and R_B that satisfies *TS* Properties 1 and 2, and which applied to $\{i, \ldots, k\}$ yields A and B. Then A' is either obtained from $R = R_A, R_B$ or else it is obtained from $R' = R_A, R_B, RevMove(i, k)$. This completes the proof of the first claim.

We next claim that every twisted sequence is in N_{TS}^*. To see this, let A' be a twisted sequence on $\{i, i+1, \ldots, j\}$, and let R be the set of 2-exchanges satisfying *TS* Properties 1 and 2, and which when applied to $\langle i, \ldots, j \rangle$ yields A'. If A' can be obtained in multiple such ways via 2-exchanges, let R be the set with the smallest number of 2-exchanges that generates A'.

We inductively assume that the claim is true for all twisted sequences with fewer cities than A'. We also assume inductively that the claim is true for all twisted sequences A with the same number of cities as A' but with fewer 2-exchanges needed to generate A.

We consider two cases. In the first case, we assume that there is **no** 2-exchange that changes the position of city i. In this case, if we apply the reversals in R to $\langle i+1, \ldots, j \rangle$, we obtain $A' \setminus i$, that is, we obtain the subsequence obtained from A by deleting city i. By inductive hypothesis $A' \setminus i \in N_{TS}^*$, and by Rule 2, $A' = i \bigstar A' \setminus i \in N_{TS}^*$. So, if no 2-exchange contains i, then $A' \in N_{TS}^*$.

We now assume that there is a reversal that contains i, and let $RevMove(i, k)$ be the last such reversal in R. We now consider the subcase that $k = j$. In this case, the reversal $Rev(A')$ of sequence A' is obtained from $\{i, \ldots, j\}$ by applying the 2-exchanges in $R' = R \setminus RevMove(i, j)$. Since $|R'| < |R|$, by inductive hypothesis $Rev(A') \in N_{TS}^*$, and so $A' \in N_{TS}^*$.

In the remaining subcase $k < j$. By *TS* Property 2, the set R may be partitioned into two sets R_1 and R_2 where each 2-exchange of R_1 is contained in $\{i, .., k\}$, and each 2-exchange of R_2 is contained in $\{k+1, \ldots, j\}$. We assume that the order of the 2-exchanges in R_1 and R_2 are consistent with their ordering in R. Let A be generated by R_1 and let B be generated by R_2. By inductive hypothesis, $A \in N_{TS}^*$ and $B \in N_{TS}^*$. Moreover, $A' = A \bigstar B$. It follows from Rule 2 that $A' \in N_{TS}^*$. ♦

We now provide the nodes of the state space graph for twisted sequences, and show that the resulting dynamic program solves twisted sequences in $O(n^7)$ time, which is the same bound obtained by Deĭneko and Woeginger (2000). We do not create the state space graph in its entirety since the optimal solution to the dynamic program does not correspond in a simple way to the shortest path in the state space graph. We let $g_{TS}(i, S, j)$ be the minimum cost of a twisted sequence starting at city i, ending at city j, and passing through each city of S. .

The Feasible States V_{TS} and the Dynamic Programming Recursion DP_{TS} .

1. $(j, \{j\}, j) \in V_{TS}$ for each $j = 1$ to n.
$g_{TS}(j, \{j\}, j) = 0$ for each j.

2. Suppose $(i_1, S_1, j_1) \in V_{TS}$ and $(i_2, S_2, j_2) \in V_{TS}$ and $\max(S_1) = \min(S_2) - 1$.
Then $(i_1, S_1 \cup S_2, j_2) \in V_{TS}$ and $(j_2, S_1 \cup S_2, i_1) \in V_{TS}$.

3. For each consecutive subset $S = \{k_1, \ldots, k_2\}$ of cities, and for every pair i_1 and j_2 cities of S with $i_1 < j_2$, $g_{TS}(i_1, S, j_2) = g_{TS}(j_2, S, i_1) =$

$$\min \quad g_{TS}(i_1, \{k_1, \ldots, k_3\}, i_2) + g_{TS}(j_1, \{k_3+1, \ldots, k_2\}, j_2) + c(i_2, j_1)$$
$$\textbf{s.t.} \quad k_1 \le i_2 \le k_3 \quad \text{and} \quad k_3 + 1 \le j_1 \le k_2.$$

4. The min cost of a tour in N_{TS} is $\min\{g_{TS}(i, S, j) + c(j, i) : |S| = n, \text{ and } i, j \in \{1, \ldots, n\}\}$.

Theorem 5. The dynamic program DP_{TS} finds the optimal twisted sequence in the neighborhood N_{TS} in $O(n^7)$ time.

Proof. The number of states in the state graph is $O(n^4)$ since each state is of the form (i, S, j) where S is consecutive and i and j are cities in S. If one looks at the minimization in Rule 3 of the dynamic programming recursion, one sees that one needs to minimize over choices of k_3, i_2 and j_1, which results in $O(n^3)$ time to determine $g(i_1, S, j_2)$. Thus the running time is $O(n^7)$. ♦

The running time matches the one developed by Deĭneko and Woeginger (2000), and the dynamic program is essentially the same.

Section 5. Optimization over DP *Restrictions.*

In this section and in the next section, we apply dynamic programming restrictions; that is, we take the Held-Karp dynamic program for the traveling salesman problem as given by the state space graph G_{TSP} in

Section 3, and solve this dynamic program as restricted to a subset $V \subseteq V_{TSP}$. We denote the state space graph as $G_{TSP}[V]$, which is standard graph theoretic notation for the subgraph of G_{TSP} induced by the subset V of nodes and all edges with both endpoints in V. This process induces a neighborhood of the original tour, which we describe in this section. We are particularly interested in examples in which the number of states is polynomially bounded and the number of tours in the induced neighborhood is exponentially large.

There is a sizeable literature in approximate dynamic programming for hard combinatorial optimization problems that is based on dynamic programming restrictions. As with the dynamic programming restrictions of this section, approximate dynamic programming is an approach that tries to circumvent the "curse of dimensionality" by contracting the state space. Usually, in approximate dynamic programming and other forms of dynamic programming restrictions, the size of the dynamic program is decreased by simultaneously aggregating some of the states and eliminating some of the constraints. Thus, the problem solved is a relaxation of the original, rather than a restriction of the original. Typically, the solution value obtained in approximate dynamic programming is a lower bound (for minimization problems) on the optimal value, whereas in the approach we discuss in this section, the optimal solution for the neighborhood is feasible for the *TSP* and thus is an upper bound on the optimal value. Christofides, Mingozzi and Toth (1981) used state space reductions to derive lower bounds for the *TSP*, the *VRP*, and variants. They give several different relaxations of the states, show how these correspond to known relaxations of the original problems, and that these relaxations are equivalent to Lagrangian relaxations. Similar techniques are applied to machine scheduling problems in, Abdul-Razaq and Potts (1988), Hariri and Potts (1994), Ibaraki and Nakamura (1994), and Potts and van Wassenhove (1987).

In order to associate neighborhoods with subsets of states, we need some additional notation and definitions. For a given sequence $A = \langle i_1, i_2, i_3, \ldots, i_k \rangle$, with $i_1 = 1$, we let $State(A) = (S, i_k)$, where $S = \{i_1, i_2, \ldots, i_k\}$. For a sequence $A = \langle i_1, i_2, i_3, \ldots, i_k \rangle$, we refer to the subsequences $A_j = \langle i_1, i_2, i_3, \ldots, i_j \rangle$ for $j = 1$ to k as the *initial subsequences* of A. The canonical dynamic program creates a tour A by starting with city 1 and concatenating one city at a time. With this in mind, for each tour A we let

$$V_{TSP}[A] = \{t\} \cup \{State(A_j) : A_j = \langle 1, \ldots, i_j \rangle \text{ is an initial subsequence of } A \text{ for } j = 1 \text{ to } |A|\}.$$

For a given collection $V \subseteq V_{TSP}$, let $N_{TSP}[V] = \{A \in S_n : V_{TSP}(A) \subseteq V\}$. In other words, $N_{TSP}[V]$ contains all tours A such that the states needed to generate A are all in V. Similarly, we let $N^*_{TSP}[V] = \{A: V_{TSP}(A) \subseteq V\}$. We will soon show that $N_{TSP}[V]$ is the neighborhood induced by solving the dynamic program on state space graph $G_{TSP}[V]$, and $N^*_{TSP}[V]$ is the corresponding superneighborhood.

In the following, for each $A = \langle i_1, \ldots, i_k \rangle$, we let $\hat{c}(A)$ denote the cost of sequence A where A is viewed as a path; that is $\hat{c}(A) = \sum_{j=1}^{k-1} c(i_j, i_{j+1})$. If $k < n$, then $c(A) = \hat{c}(A)$. If A is a tour, then $c(A) = \hat{c}(A) + c(i_n, i_1)$.

The following theorem shows that finding the shortest path from node $(\{1\}, 1)$ to all other nodes in $G_{TSP}[V]$ corresponds to finding the best tour in $N_{TSP}[V]$.

Theorem 6. Let V be any subset of states of V_{TSP}. Let (S, j) be any state in V, and let $g(S, j)$ be the minimum cost of a path from $(\{1\}, 1)$ to (S, j) in $G_{TSP}[V]$. Then

$$g(\mathrm{S}, j) = \min \{ \hat{c}(A) : State(A) = (S, j) \text{ and } A \in N^*_{TSP}[V] \}, \text{ and}$$
$$g(t) = \min \{ c(A) : A \in N_{TSP}[V] \}.$$

Proof. We first claim that $g(S, j) = \min \{ \hat{c}(A) : State(A) = (S, j) \text{ and } A \in N^*_{TSP}[V] \}$. This claim is clearly true for $|S| = 1$, since the only state of V_{TSP} with one element is $(\{1\}, 1)$. Let us assume inductively that the claim is true for any state (S', j') with $|S'| < |S|$, and suppose that $|S| > 1$. In the restriction of the Held-Karp dynamic program for the *TSP*,

$$g(S, j) = \min \{g(S \setminus j, k) + c(k, j) : k \in S \setminus j \text{ and } (S \setminus j, k) \in V \},$$

and thus by induction,

$$g(S, j) = \min \{ \hat{c}(A') + c(k, j) : A' \in N^*_{TSP}[V] \text{ and } State(A') = (S \setminus j, k)\}.$$

Let A'' be the set that causes the minimization in the previous relation. Since $A'' \in N^*_{TSP}[V]$, $State(A'') = (S \setminus j, k)$, and $(S, j) \in V$, it follows that $A'' \bigstar j \in N^*_{TSP}[V]$, and thus

$$g(S, j) = \min \{ \hat{c}(A) : State(A) = (S, j) \text{ and } A \in N^*_{TSP}[V] \}.$$

From the above claim it also follows that $g(t) = \min \{ \hat{c}(A) : A \in N_{TSP}[V] \}$. ♦

We next apply the results of Theorem 6 to the neighborhood first developed by Balas (1996), and later extended by Balas and Simonetti (2001).

The Balas-Simonetti Neighborhood

Balas (1996) considered the neighborhood consisting of all sequences A with the following properties: (1) the first city of A is city 1, and (2) there is a parameter K of the neighborhood, such that i follows j in A if $j + K \leq i$. We consider the following equivalent formulation: If i precedes j in A, then $i < K + j$. Balas presented a dynamic programming recursion for finding the minimum distance tour in the neighborhood that runs in $O(K^2 2^K n)$ time, which is linear in n for fixed K and polynomial in n for $K = O(\log n)$. The number of tours in the neighborhood is $(\Omega((k-1)^n / e^n)$, which is exponential in n whenever k is slowly growing in n. (Here, exponential means non-polynomial.)

Balas and Simonetti (2001) generalized the neighborhood to require a condition that is equivalent to the following: If i precedes j in A, then $i < K(j) + j$, where $K(j)$ is a positive integer bounded above by K. They also showed how to implement the dynamic program effectively by searching the state space graph, and they carried out extensive computational experiments. Here we show that the Balas-Simonetti neighborhood can be constructed in a very natural way using recursion. Moreover, the recursion immediately suggests an implementation very similar to the one that Balas and Simonetti applied. In the following, if A is a sequence of cities or a set, we let $\overline{A} = \{1, \ldots, n\} \setminus A$, that is the cities not in A.

The Balas-Simonetti Neighborhood N_{BS} for the *TSP*.

1. $1 \in N^*_{BS}$.

2. Suppose $A \in N^*_{BS}$ and choose $i' \in \bar{A}$ so that $i' + K(i') = \min \{i + K(i): i \in \bar{A}\}$. Then for each $j \in \bar{A}$ with $j < i' + K(i')$, $A \bigstar j \in N^*_{BS}$.

3. $N_{BS} = N^*_{BS} \cap S_n$.

State Space Graph $G_{BS} = (V_{BS}, E_{BS})$:

1. $(\{1\}, 1) \in V_{BS}$.

2. Suppose $(S, k) \in V_{BS}$, and choose $i' \in \bar{A}$ so that $i' + K(i') = \min \{i + K(i): i \in \bar{S}\}$. Then for each $j \in \bar{S}$ with $j < i' + K(i')$, $(S \cup \{j\}, j) \in V_{BS}$, and there is an arc from (S, k) to $(S \cup \{j\}, j)$ with cost $c(k, j)$.

3. If $(S, j) \in V_{BS}$ and if $|S| = n$, then there is an arc from (S, j) to t with cost $c(j, 1)$.

We next show that N^*_{BS} is the Balas-Simonetti neighborhood.

Lemma 1. The tour $A \in N_{BS}$ if and only if whenever i precedes j in A, then $i < K(j) + j$.

Proof. Suppose $A = \langle 1, i_2, i_3, \ldots, i_n \rangle$, and let $A_j = \langle 1, i_2, i_3, \ldots, i_j \rangle$. Let us assume first that $A \in N_{BS}$, and thus $A_k \in N^*_{BS}$ for each k. It follows directly from the construction of A_k in Step 2 of the N_{BS} neighborhood, that $i_k < i_l + K(i_l)$ for all $l > k$.

We next assume that whenever i precedes j in A, then $i < K(j) + j$. We will claim that $A_k \in N^*_{BS}$ for all k, and thus $A \in N_{BS}$. The claim is clearly true for $k = 1$, and we assume inductively that the claim is true for $k - 1$. By inductive hypothesis $A_{k-1} \in N^*_{BS}$, and by our choice of A, $i_k < i_l + K(i_l)$ for all $l > k$. It follows that $A_k \in N^*_{BS}$, completing the proof. ♦

The recursion for N_{BS} adds one city at a time to the end of a sequence A such that the newly added city satisfies a condition. Hence the recursion for N_{BS} is a restriction of the Held and Karp recursion for N_{TSP} and $V_{BS} \subseteq V_{TSP}$. Furthermore, by Theorem 6 finding the shortest path from node $(\{1\}, 1)$ to node t in $G_{TSP}[V_{BS}]$ corresponds to finding the best tour in $N_{TSP}[V_{BS}]$ which is equivalent to N_{BS}.

The following lemma is proved in Balas and Simonetti for their neighborhood. We give an alternative proof here that follows from the recursive construction of N^*_{BS}.

Theorem 7. Consider the Balas-Simonetti neighborhood and suppose that $K < \log n$. Then $|V_{BS}| = O(n K 2^k)$, and $|E_{BS}| = O(n K^2 2^K)$. The time to construct G_{BS} is $O(n K^2 2^K)$, and the time to find a minimum distance neighbor is $O(n K^2 2^K)$.

Proof. Let $Q_r = \{S : (S, j) \in V_{BS}$ for some j, and $r = \min(\bar{S})\}$. Then for $1 \le k < r$ it follows that $k \in S$. For $r + K(r) \le k \le n$, it follows that $k \in \bar{S}$. Thus for any $(S, j) \in Q_r$, $S = \{1, \ldots, k\text{-}1\} \cup S'$, where $S' \subseteq \{r + 1, \ldots, r + K(r)\text{-}1\}$. It follows that there are at most $2^{K(r)} \le 2^K$ choices for S' and thus for S. In

addition, there are at most K choices for j, and so $|Q_r| = O(K\,2^K)$. Moreover, $V_{BS} = \bigcup_{r=1}^{n} Q_r$, and so $|V_{BS}| = O(n\,K\,2^K)$.

We note in Step 2 in the creation of E_{BS}, there are at most K edges emanating from any state (S, j), and so $|E_{BS}| = O(n\,K^2\,2^K)$.

We next show that we can store (S, j) in O(1) space for all states $(S, j) \in V_{BS}$. So suppose $(S, j) \in Q_r$. We represent $\overline{S}$ by the pair $(r, L(\overline{S}))$ where $L(\overline{S}) = \sum_{k \in \overline{S} \cap [r,\, r+K]} 2^k$. Then $1 \le L(\overline{S}) \le 2^{K+1} \le 2n$, and the pair $(r, L(\overline{S}))$ uniquely identifies and characterizes S. We refer to $(r, L(\overline{S}))$ as the *identifier* for S, and we refer to the triple $(j, r, L(\overline{S}))$ as the *identifier* for (S, j). For any state (S, j) created in Step 2 of the recursion of G_{BS}, it takes O(K) time to determine the identifier. Moreover, it takes O(K) time to determine all of the arcs emanating from (S, j) . So, the time to create G_{BS} is $O(n\,K^2\,2^K)$, and the time to solve the shortest path problem is also $O(n\,K^2\,2^K)$. ♦

The Weak Insertion Dynasearch Neighborhood

In this subsection we show that the $\boldsymbol{N_{WIDS}}$ can be searched efficiently in $O(n^2)$ when described with a recursion which is a restriction of the Held and Karp recursion for $\boldsymbol{N_{TSP}}$. First we provide a characterization of the weak independent insertion neighborhood, and then give the recursive description.

Lemma 2. Let $A = \langle 1, i_2, i_3, \ldots, i_n \rangle$, and let $A_j = \langle 1, i_2, i_3, \ldots, i_j \rangle$ for each $j = 2$ to n. Then $A \in$ Weak insertion dynasearch neighborhood if and only if for each j, the cities of A_j are $\{1, 2, \ldots, j+1\} \backslash k$ for some $k \in \{2, \ldots, j+1\}$.

Proof. Suppose first that A is in the weak insertion dynasearch neighborhood. If $A = \langle 1, 2, \ldots, n \rangle$, then the cities of A_j are $\{1, \ldots, j\}$, and the lemma is valid. Suppose instead that $A \ne \langle 1, 2, \ldots, n \rangle$, and let InsertMove($r$, s) be the last InsertMove in the creation of A. We assume inductively that for $j < r$, the cities of A_j are $\{1, 2, \ldots, j+1\} \backslash k$ for some $k \in \{2, \ldots, j+1\}$. Because A is formed by compounding weakly independent moves, the cities of A_{r-1} are $\{1, 2, \ldots, r-1\}$. For $r \le t \le s-1$, the cities of A_t are $\{1, 2, \ldots, t+1\} \backslash r$. Finally, for $t > s$, the cities of A_t are $\{1, 2, \ldots, t\}$. We have thus established the "only if" part of the lemma.

Suppose instead that for each j, the cities of A_j are $\{1, 2, \ldots, j+1\} \backslash k$ for some $k \in \{2, \ldots, j+1\}$. Let $S = \{r$: the cities of A_r are $\{1, 2, \ldots, r\}$, and let us denote the cities in S as $\{1, j_2 \ldots, j_k\}$. Then one can establish inductively that A can be created from $InsertMove(j_s+1, j_{s+1})$ for all s such that $j_s+1 \ne j_{s+1}$. ♦

The Weak Insertion Dynasearch Neighborhood $\boldsymbol{N_{WIDS}}$ for the TSP

1. $1 \in N^*_{WIDS}$

2. Suppose that $A \in N^*_{WIDS}$ and $\max(A) = i$. Then

- a. $A \bigstar i+1 \in N^*_{WIDS}$.
- b. if $|A| = i$, then $A \bigstar i+2 \in N^*_{WIDS}$.
- c. if $|A| = i-1$, then $A \bigstar j \in N^*_{WIDS}$, where $j = \{1, \ldots., i\} \backslash A$.

3. $N_{WIDS} = N^*_{WIDS} \cap S_n$.

State Space Graph $G_{WIDS} = (V_{WIDS}, E_{WIDS})$:

1. $(\{1\}, 1) \in V_{WIDS}$.

2. Suppose $(S, k) \in V_{WIDS}$ and $\max(S) = i$. Then $(S \cup \{j\}, j) \in V_{WIDS}$ where

a. $(S \cup \{i+1\}, i) \in V_{WIDS}$ and there is an arc from (S, k) to $(S \cup \{i+1\}, i+1)$ with cost $c(k, i+1)$.

b. if $|S| = i$, then $(S \cup \{i+2\}, i+2) \in V_{WIDS}$ and there is an arc from (S, k) to $(S \cup \{i+2\}, i+2)$ with cost $c(k, i+2)$.

c. if $|S| = i-1$, then $(S \cup \{j\}, j) \in V_{WIDS}$ for $j = \{1, \ldots, i\} \setminus A$ and there is an arc from (S, k) to $(S \cup \{j\}, j)$ with cost $c(k, j)$.

3. If $(S, j) \in V_{BS}$ and if $|S| = n$, then there is an arc from (S, j) to t with cost $c(j, 1)$.

Theorem 8. The neighborhood N_{WIDS} is the weak insertion dynasearch neighborhood. The corresponding state space graph G_{WIDS} has $O(n^2)$ nodes and $O(n^2)$ edges. The time to find a minimum distance neighbor is $O(n^2)$.

Proof. Lemma 2 establishes that N_{WIDS} is the weak insertion dynasearch neighborhood.

By Lemma 2 there are $O(n^2)$ values for (S, k) because $S = \{1, \ldots, j\}$ and $1 < k \leq j$ or else $S = \{1, \ldots, k\} \setminus \{i\}$, $1 < i < k$. In both cases, the number of arcs emanating from each state is equal to 2 and thus there are $O(n^2)$ edges. Moreover, the graph G_{WIDS} is acyclic and can be created in $O(n^2)$ steps. It follows that the time to find a shortest path in G_{WIDS} from node $(\{1\}, 1\}$ to node t is $O(n^2)$. ♦

We close this section with a simple negative result. No superset of the pyramidal tour neighborhood can be obtained as a neighborhood $N_{TSP}[V]$ if we require that $|V|$ is polynomially bounded. To see this, suppose that the pyramidal tours are a subset of $N_{TSP}[V]$ for some $V \subseteq V_{TSP}$. Such a subset V exists since $N_{TSP}[V_{TSP}]$ contains all tours. Recall that a pyramidal tour can be expressed as $\langle i_1, \ldots, i_{j-1} \rangle \star n \star \langle i_{j+1}, \ldots, i_{n-1} \rangle$, where the sequence $\langle i_1, \ldots, i_{j-1} \rangle$ is increasing in index, and the sequence $\langle i_{j+1}, \ldots, i_{n-1} \rangle$ is decreasing. There are 2^{n-2} distinct ways that one can choose the cities in $\langle i_1, \ldots, i_{j-1} \rangle$, which means that V must contain at least 2^{n-2} distinct states, contrary to the assumption that V is polynomially bounded in n.

Section 6. The Dynamic Programming Expansion of a Neighborhood.

For a given tour A, let $V_{TSP}[A]$ be the set of states of V_{TSP} associated with A, as defined in Section 5, and for a given collection N of tours, we let $V_{TSP}[N] = \bigcup_{A \in N} V_{TSP}[A]$.

In Section 5, we associated a neighborhood $N_{TSP}[V]$ with a subset V of states. In this section, we consider the inverse operation of associating a set of states $V_{TSP}[N]$ with a neighborhood. Moreover, for a given neighborhood N, we can first associate the set of states $V' = V_{TSP}[N]$, and then create a second neighborhood $N_{TSP}[V']$. We refer to the neighborhood $N_{TSP}[V'] = N_{TSP}[V_{TSP}[N]]$ as the *expansion* of neighborhood N with respect to the dynamic program DP_{TSP}, or more briefly as the *dynamic programming expansion* of N. In general, if N is any polynomially sized neighborhood, then N is a subset of the dynamic programming expansion N' of N, and N' can be searched in polynomial time. The

primary result of this section is that the dynamic programming expansion of the 2-exchange neighborhood is the dynasearch neighborhood obtained by compounding weakly independent 2-exchanges.

We discovered the above result only after observing that for the 2-exchange neighborhood N_{EX} and for the neighborhood N_{WDS} obtained by compounding weakly independent 2-exchanges, it is true that $V_{TSP}[N_{EX}]= V_{TSP}[N_{WDS}]$. Thus we learned that taking a dynamic programming expansion of a neighborhood can transform a polynomial sized neighborhood into an exponential neighborhood that is searchable in polynomial time. We have determined other cases where the dynamic programming expansion of a polynomial size neighborhood creates an exponential neighborhood that is searchable in polynomial time, but each of these cases has a very similar flavor to the dynasearch neighborhoods. It is too early to assess whether this technique of creating polynomially searchable exponential neighborhoods via dynamic programming expansion is an interesting anomaly or whether it is potentially a useful methodology in very large scale neighborhood search.

We first state four propositions about the operations $V_{TSP}[N]$ and $N_{TSP}[V]$, each of which has a very straightforward proof which is omitted. We give a proof of the fifth proposition.

Proposition 1. For a given set $N \subseteq S_n$, $V_{TSP}[N]$ is the smallest subset $V \subseteq V_{TSP}$ of states such that $N \subseteq N_{TSP}[V]$.

Proposition 2. For a given subset $V' \subseteq V_{TSP}$ of states, $N_{TSP}[V']$ is the largest subset $N \subseteq S_n$ such that $V_{TSP}[N] \subseteq V$.

Proposition 3. For a given set $N \subseteq S_n$, $N \subseteq N_{TSP}[V_{TSP}[N]]$.

Proposition 4. For a given subset $V \subseteq V_{TSP}$ of states, $V_{TSP}[N_{TSP}[V]] \subseteq V$.

Proposition 5. Suppose that N_n is a neighborhood whose size is polynomially bounded in n. Then one can find the best neighbor in $N_{TSP}[V_{TSP}[N]]$ in polynomial time.

Proof. Since $| N_n | = O(p(n))$ for some polynomial p( ), it follows that $|V_{TSP}[N]| = O(n\,p(n))$, and the number of edges of the corresponding state space graph is $O(n^2 p(n))$. Thus, the dynamic program can be solved in $O(n^2 p(n))$ time. ♦

The 2-Exchange Neighborhood and its Dynamic Programming Expansion.

Let N_{EX} denote the 2-exchange neighborhood, which can be obtained from $\langle 1, \ldots, n\rangle$ by performing at most one operation $RevMove(i, j)$ for $1< i < j \le n$. Recall that the neighborhood N_{WDS} is obtained from $\langle 1, \ldots, n\rangle$ by permitting the compounding of weakly independent 2-exchanges that exclude city 1. Thus any element of N_{WDS} can be obtained from $\langle 1, \ldots, n\rangle$ by performing a sequence of 0 or more operations $RevMove(i_1, i_2)$, $RevMove(i_3, i_4)$, …, $RevMove(i_{2j-1}, i_{2j})$, where $1 < i_1 < i_2 < \ldots < i_{2j}$.

Theorem 9. The neighborhood N_{WDS} obtained by compounding weakly independent 2-exchanges is the dynamic programming expansion of the 2-exchange neighborhood N_{EX}; that is, $N_{WDS} = N_{TSP}[V_{TSP}[N_{EX}]]$.

Proof. We will first establish that $N_{WDS} \subseteq N_{TSP}[V_{TSP}[N_{EX}]]$, by proving the following claim.

Claim 1. $V_{TSP}[N_{EX}] = V_{TSP}[N_{WDS}]$.
From Claim 1, it will follow that $N_{TSP}[V_{TSP}[N_{WDS}]] = N_{TSP}[V_{TSP}[N_{EX}]]$, and by Proposition 3, $N_{WDS} \subseteq N_{TSP}[\mathrm{V}_{\mathrm{TSP}}[N_{WDS}]] = N_{TSP}[V_{TSP}[N_{EX}]]$.

Proof of Claim 1. It is clear that $V_{TSP}[N_{EX}] \subseteq V_{TSP}[N_{WDS}]$, and so it suffices to show that any state in $V_{TSP}[N_{WDS}]$ is also in $V_{TSP}[N_{EX}]$. Suppose that (S, k) is a state in $V_{TSP}[N_{WDS}]$. Then there is a sequence $A \in N_{WDS}$ such that A is obtained by carrying out by carryout out a sequence of weakly independent reversal moves. Suppose $State(A_r) = (S, k)$ for some initial subsequence A_r of A. We need to prove that $(S, k) \in V_{TSP}(N_{EX})$. If city r is not part of any reversal, then $S = \{1, 2, \ldots, r\}$ and $State(A_r) = (\{1, \ldots, r\}, r) \in V_{TSP}(N_{EX})$ and the claim is true. So, we now consider the case that city r is part of a reversal, say $Rev[i, j]$. Let B be the sequence in the 2-exchange neighborhood obtained by performing only $RevMove[i, j]$, and let B_r consist of the first r cities of B in order. Then $State(B_r) = State(A_r)$, and so Claim 1 is true.

We will soon establish that $N_{TSP}[V_{TSP}[N_{EX}]] \subseteq N_{WDS}$; but first we state a characterization of $V_{TSP}[N_{EX}]$, followed by a property of N_{WDS}. Both are elementary and stated without proof.

Claim 2. $V_{TSP}[N_{EX}] = \{(S, j) : S = \{1,\ldots, k\}$ for some $k \geq j\} \cup \{(S, j) : \mathrm{S} = \{1, \ldots, i\} \cup \{j, \ldots, k\}$ for some i, j, and k with $i+2 \leq j \leq k\}$.

Claim 3. The tour $A = \langle i_1, \ldots \mathrm{i}_n\rangle$ is in N_{WDS} if the following is true:

1. $i_1 = 1$ and
2. if $i_j > i_{j+1}$, then $i_{j+1} = i_j - 1$.

Let $V = V_{TSP}[N_{EX}]$. We now prove that any tour $A \in N_{TSP}[V]$ satisfies the properties of Claim 3, and is therefore in N_{WDS}. Let $A = \langle i_1, \ldots i_n\rangle$, and let $A_j = \langle i_1, \ldots, i_j\rangle$ for each j. It is clear that $i_1 = 1$ because the only state in V with one element is $(\{1\}, 1)$. Now suppose that $i_j > i_{j+1}$. Then $State(A_j) = (S, i_j) \in V$, and $S = \{1,\ldots, i'\} \cup \{i_j, \ldots, k\}$ for some $i' < i_{j+1}$ and some $k \geq i_j$. Then $State(A_{j+1}) = \{1,\ldots, i'\} \cup \{i_{j+1}, \ldots, k\}$, because there is no other possible state of V that is consistent with A_{j+1} and thus $i_{j+1} = i_j - 1$, and A satisfies Claim 3, and is thus in N_{WDS}. ♦

Section 7. Conclusions

In this paper we discussed two unifying perspectives for defining neighborhood search as applied to the traveling salesman problem.

1. We showed that several previously developed exponential neighborhoods for the traveling salesman problem that are solvable in polynomial time can be defined recursively. Furthermore, the exponential neighborhoods can be searched efficiently with the dynamic program induced from the recursion. We illustrated our approach on dynasearch neighborhoods, the pyramidal tour neighborhood, and the twisted sequences neighborhood, and the Balas-Simonetti neighborhood.

2. Given the canonical Held-Karp dynamic programming formulation for the TSP, one can obtain a neighborhood by restricting the dynamic program to a subset V of states. We explain the correspondence between the *DP* restriction and the neighborhood and show that the Balas-Simonetti neighborhood can easily be represented in this manner. Furthermore we show that the

weak independent insertion dynasearch neighborhood can be represented and searched efficiently as a DP restriction. We also show how one may map neighborhoods to sets of states and then back to neighborhoods leading to the dynamic programming expansion of a neighborhood. In the case that the original neighborhood is the 2-exchange neighborhood, the dynamic programming extension is the neighborhood obtained by permitting compounding of weakly independent 2-exchanges.

There are a number of future issues suggested by this research. First of all, it would be interesting to identify new neighborhoods for the *TSP* that can be obtained in the manner of this paper, and for which the neighborhood search algorithms are effective in practice. Second, it would be of value to extend this approach to other combinatorial optimization problems for which there is a dynamic programming recursion that is more efficient than complete enumeration.

Another issue that was briefly touched upon in this paper was that of "automation" of the process. In particular, is there an algorithm that could take the input for the recursion, and automatically create the dynamic programming state space graph? If such an approach could be developed, it could be of substantial value in permitting the quick creation of neighborhood search algorithms. The results in this paper offer some first hints that such an automatic procedure may exist.

Acknowledgements.

Özlem Ergun was supported in part under NSF grant DMI-0238815.James B. Orlin was supported in part under ONR grant N00014-98-1-0317 and under NSF grant DMI-0217123.

References

Aarts, E. and J.K. Lenstra. 1997. *Local Search in Combinatorial Optimization*. John Wiley & Sons, New York.

Abdul-Razaq, T.S. and C.N. Potts. 1988. Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society* 39, 141-152.

Agarwal, R., Ö. Ergun, J.B. Orlin, C.N. Potts. 2003. Improvement graphs in large-scale neighborhood search: a case study in scheduling. Working paper, School of Ind. and Sys. Engr., Georgia Institute of Technology, Atlanta.

Ahuja, R.K., Ö. Ergun, J.B. Orlin, A.B. Punnen. 2002. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123, 75-102.

Ahuja, R.K., T.L. Magnanti, J.B. Orlin. 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Englewood Cliffs, NJ.

Aurenhammer, F. 1988. On-line sorting of twisted sequences in linear time. *BIT* 28, 194-204.

Balas, E. 1996. New classes of efficiently solvable generalized salesman problems. MSRR No. 615, GSIA, Carnegie Mellon University, Pittsburgh.

Balas, E. and N. Simonetti. 2001. Linear time dynamic programming algorithms for new classes of Restricted TSPs. *Informs Journal on Computing*, 13, 56-75.

Bertsekas, D.P., J.N. Tsitsiklis, and C. Wu. 1997. Rollout Algorithms for Combinatorial Optimization. *Journal of Heuristics* 3, 245-262.

Carlier, J. and P. Villon. 1990. A new heuristic for the traveling salesman problem. *RAIRO - Operations Research* 24, 245-253.

Christofides, N., A. Mingozzi, P. Toth. 1981. State-space relaxation procedures for the computation of bounds to routing problems. *Networks* 11, 145-164.

Congram, R.K. 2000. Polynomially Searchable Exponential Neighborhoods for Sequencing Problems in Combinatorial Optimization. Ph.D. Dissertation. Faculty of Mathematical Studies, University of Southampton, UK.

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

Deĭneko V. and G.J. Woeginger. 2000. A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem. *Mathematical Programming* 87, 519-542.

Ergun Ö. 2001. New Neighborhood Search Algorithms Based on Exponentially Large Neighborhoods. Ph.D. Dissertation. Operations Research Center, MIT, Cambridge.

Ergun, Ö., J.B. Orlin, and A. Steele-Feldman. 2002. A computational study on a large-scale neighborhood search algorithm for the vehicle routing problem with capacity and distance constraints. Working paper, School of Ind. and Sys. Engr., Georgia Institute of Technology, Atlanta.

Hariri, A.M.A. and C.N. Potts. 1994. Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. *Management Science* 40, 1712-1719.

Held, M. and R. M. Karp. 1962. A dynamic programming approach to sequencing problems. *J. SIAM* 10, 196-210.

Ibaraki, T. and Y. Nakamura. 1994. A dynamic programming method for single machine scheduling. *European Journal of Operational Research* 76, 72-82.

Potts, C.N. and S.L. van de Velde. 1995. Dynasearch - iterative local improvement by dynamic programming: part I, The traveling salesman problem. Preprint, Faculty of Mathematical Studies, University of Southampton, UK.

Potts, C.N. and L.N. van Wassenhove. 1987. Dynamic programming and decomposition approaches for the single machine total tardiness problem. *European Journal of Operational Research* 32, 405-414.

Sarvanov, V.I. and N.N. Doroshko. 1981. Approximate solution of the traveling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time. *Software: Algorithms and Programs*, *Mathematics Institute of the Belorussia Academy of Science* 31, 11-13. (In Russian)