ON THE SOLUTION OF THE TRAVELING SALESMAN PROBLEM:

A NOVEL HEURISTIC THAT USES

FREQUENCY OF ANCHORED NEAREST NEIGHBORS

by

Yousof Hussein Kutkut

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

May 2001

ON THE SOLUTION OF THE TRAVELING SALESMAN PROBLEM:

A NOVEL HEURISTIC THAT USES

FREQUENCY OF ANCHORED NEAREST NEIGHBORS

by

Yousof Hussein Kutkut

has been approved

April 2001

APPROVED:

Philip M. Wolfe, Chair

Richard L. Smith

Mitt Sl

Kraig Knutson

Supervisory Committee

ACCEPTED:

Department Chair

Bianca L. Bernstein

Dean, Graduate College

ABSTRACT

In this thesis, a heuristic for solving the Traveling Salesman Problem (TSP) is presented. This technique works with a pool of solutions, which has certain properties. The main goal is to shrink the size of this pool and refine its properties. Finally, the heuristic ends up with a pool that can no longer produce better solutions from its predecessors. The main property, which the heuristic works with, is the frequency of which edges (city-city connections) appear in a pool of solutions, i.e. Frequency of Anchored Nearest Neighbors (FANN).

An implementation is presented that uses a modified version of the Nearest Neighbor algorithm (anchoring edges as tours are constructed) to initialize a pool of solutions. An edge matrix representation of the frequency of which edges appear in tours of a pool is recorded and becomes the new neighborhood for constructing future solutions (pools), i.e. FANN becomes the new measure of nearness, and its edge representation acts as a candidate set for constructing future tours. As the algorithm progresses, new pools are created which have less entries in their edge matrix representations, since in constructing each new pool prior information is exploited, connecting to (and anchoring) nearest neighbors from earlier edge matrix representations, to produce new solutions. Finally, the heuristic ends up with a pool of solutions that can no longer produce better tours.

Computational results consistently found optimal or high-quality solutions in a variety of benchmark instances of the symmetrical and asymmetrical traveling salesman problems (STSP and ATSP). This encourages future research towards adopting FANN,

as a measure of nearness, in solving larger instances of the STSP, ATSP and other non-deterministic polynomial hard problems.

بسم الله الرحمن الرحيم

"وقالوا الحمد لله الذي هدانا لهذا وما كنا لنهتدي لولا أن هدانا الله"

سورة الأعراف 43

To my Parents

Hussein and Rihab

ACKNOWLEDGEMENTS

In spring of 2000, I was fortunate enough to take a class on Operations Research with Dr. Mathew Carlyle where I was introduced for the first time to the Traveling Salesman Problem (TSP). From then on I knew that this was a problem that I would definitely like to learn more about. I found that there existed a host of published work on finding solutions for the TSP. Though no efficient algorithm has been developed, there has been tremendous progress in designing fast approximate solutions and even in solving ever-larger problem instances to optimality.

This work documents my experiments with the TSP. I experienced the fascination of problem solving that, I believe, everyone studying the TSP will experience. The work presented here profited from discussions and meetings with my supervisory committee Dr. Philip Wolfe, Dr. Mathew Carlyle, Dr. Kraig Knutson and Dr. Richard Smith. I would like to thank my manager at the Behavioral Science Computer Cluster – ASU, Sharon Bushart, for without her encouragement and support this thesis may never have been completed.

Reflecting on what Roselle Mercier Montgomery once said: “Love, like Ulysses, is a wanderer, for new fields always and new faces yearning... Put by, O waiting ones, put by your weaving, unlike Ulysses, love is un-returning.” I would like to express my deepest gratitude to my parents and family, especially my sister Alicia, for their love and support that to my good fortune, never got to wander.

I am confident that the TSP will continue to stimulate further efforts and continue to serve as a classical benchmark problem for algorithmic ideas.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

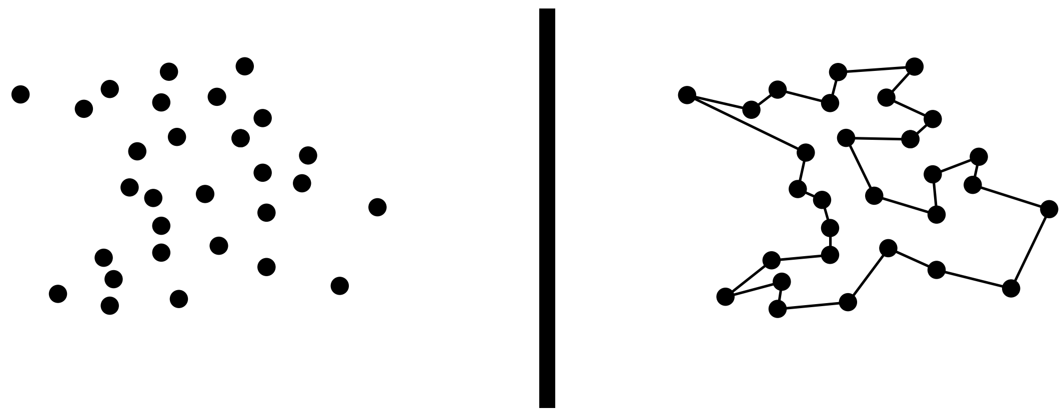
In this thesis, a heuristic for solving the Traveling Salesman Problem (TSP), using Frequency of Anchored Nearest Neighbors (FANN), is presented. In this introductory chapter, we describe the problem, outline some of the most popular techniques suggested so far for tackling the TSP and introduce some of its practical applications.

1.1 The Problem

The Traveling Salesman Problem is stated as follows: given a set of n nodes and distances between every pair of nodes, find a cycle of minimal total length visiting each node exactly once. Of course, instead of distance, other notions such as time, cost, etc., could be considered. We will use 'distance' to represent any such measure. Figure 1.1 illustrates an instance of the TSP (without distances labeled) and a possible solution. In this thesis, we examine two variants of the TSP:

- Symmetrical Traveling Salesman (STSP): the distance from node i to node j is the same as from node j to node i
- Asymmetrical Traveling Salesman (ATSP): the distance from node i to node j and the distance from node j to node i may be different

The TSP is a relatively old problem [1]; it was documented as early as 1759 by Euler (though not by that name), whose interest was in solving the knights' tour problem. A correct solution would have a knight visit each of the 64 squares of a chessboard



Input **Output**

Figure 1.1 TSP Pictorial Display

exactly once in its tour. The term 'traveling salesman' was first used in mathematical circles in 1931-32, but the term first appeared in an 1832 German book *The traveling salesman, how and what he should do to get commissions and be successful in his business*, written by a veteran traveling salesman, see [1] for detailed history.

Before discussing solution algorithms, it is worth mentioning that the ATSP can be transformed into a STSP and solved by any of the algorithms used to solve the STSP [2, 3]. The following transformation method transforms an asymmetric problem with n nodes into a problem of $2n$ nodes. Let $D = (d_{ij})$ denote the $n \times n$ distance matrix of the asymmetric problem. Then let $D' = (d'_{ij})$ be a $2n \times 2n$ symmetric matrix computed as follows:

$$d'_{n+i,j} = d'_{j,n+i} = d_{i,j} \qquad \text{for } i = 1,2,\ldots,n,\; j = 1,2,\ldots,n \text{ and } i \neq j$$

$$d'_{n+i,i} = d'_{i,n+i} = -M \qquad \text{For } i = 1,2,\ldots,n$$

$$d'_{i,j} = M \qquad \text{otherwise,}$$

where M is a sufficiently large number, e.g., $M = \sum d_{i,j}$

Any optimal solution of the new symmetric problem corresponds to an optimal solution of the original asymmetric problem. An obvious disadvantage of the transformation is that it doubles the size of the problem. Therefore, in practice it is more advantageous to use algorithms dedicated for solving asymmetric problems. We will not discuss these specialized algorithms in this context and the reader is referred to [1] for a detailed study of the asymmetric TSP.

1.2 Solution Algorithms

The TSP is easy to state, but hard to solve. The difficulty becomes apparent when one considers the number of possible tours, an astronomical figure even for a relatively small number of cities. For a problem with n cities there are $(n - 1)!$ possible tours. If n is 20, there are more than 10^{17} tours. In comparison it may be noted that the number of elementary particles in the universe has been estimated to be only 10^{87}. It has been proven that the TSP is a member of the set of NP-complete problems [4]. Problems which have known polynomial algorithms are said to be in the class *P*. A superset of class *P* is the class *NP* where NP stands for “non-deterministic polynomial”. NP consists of all problems that can be solved in polynomial time on a *non-deterministic Turing machine*. This includes all problems in P but also 'hard' problems, such as the TSP, for which all known algorithms require exponential time. Hard problems can be transformed one to the other in polynomial time. This property has been used to define a separate sub-class in NP that of *NP-complete* problems. The members of this class are related so that if a polynomial time algorithm were found for one problem, polynomial time algorithms would exist for all of them. However, it is commonly believed that no such polynomial algorithm exists. Therefore, any attempt to construct a general algorithm for finding optimal solutions for the TSP in polynomial time must (probably) fail [2]. Algorithms for solving the TSP may be divided into two classes:

- Exact algorithms
- Approximate (or heuristic) algorithms

1.2.1 Exact Algorithms

Exact algorithms are guaranteed to find the optimal solution in a bounded number of steps. The most effective exact algorithms for the TSP are cutting-plane algorithms [2]. These algorithms are quite complex, with codes on the order of 10,000 lines. In addition, the algorithms are very demanding of computer power. For example, it took roughly 3-4 years of CPU time on a large network of computers to determine the exact solution of a 7397-city problem [5].

1.2.2 Heuristic Algorithms

In contrast, heuristics (approximate algorithms) obtain good solutions but do not guarantee that the optimal solution will be found. These algorithms are usually very simple and have (relatively) short running times. Some of the algorithms give solutions that in average differ only by a few percent from the optimal solution. Therefore, if a small deviation from optimum can be tolerated, it may be appropriate to use an approximate algorithm. Heuristics may be subdivided into the following three classes:

- Tour construction algorithms

- Tour improvement algorithms
- Hybrid (Composite) algorithms

The tour construction algorithms gradually build a tour according to some construction rule, by adding a new city at each step, but do not try to improve upon this tour. The tour improvement algorithms improve upon a tour by performing various exchanges. The hybrid algorithms combine these two features.

A simple example of a tour construction algorithm is the so-called Nearest-Neighbor algorithm (NN): Start at an arbitrary city. As long as there are cities that have not yet been visited, visit the nearest city that still has not appeared in the tour. Finally, return to the first city. Several variants to the NN algorithm are also available where they try to improve on the quality of the tours constructed. Another simple construction algorithm is the Insertion type heuristic where we start with tours on small subsets and then extend these tours by inserting the remaining nodes one at a time according to some criterion. The selected node to be inserted is usually inserted into the tour at the point causing shortest increase in the length of the tour. For detailed information on different construction heuristics the reader is referred [1, 2, 6].

A simple example of a tour improvement algorithm is the so-called 2-opt algorithm: Start with a given tour. Replace 2 links of the tour with 2 other links in such a way that the new tour length is shorter. Continue in this way until no more improvements are possible. Figure 1.2 illustrates a 2-opt exchange of links, a so-called 2-

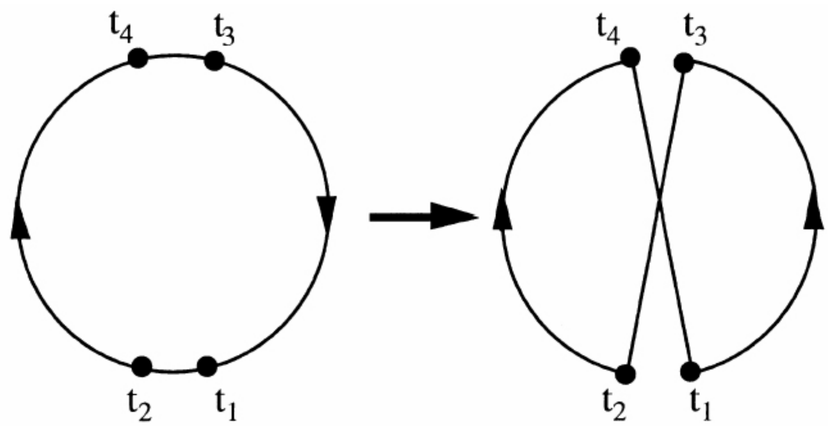


Figure 1.2 A 2-opt Move

opt move. Note that a 2-opt move keeps the tour feasible and corresponds to a reversal of a subsequence of the cities.

A generalization of this simple principle forms the basis for one the most effective approximate algorithms for solving the TSP, the Lin-Kernighan algorithm [7]. There have been a lot of implementations and variants to the original Lin-Kernighan algorithm [2, 6] among these is a really interesting algorithm [8] that makes use of computing neighbor lists based on a distance function modified by Lagrangean relaxation [9, 10].

Typically, construction heuristics get within roughly 10-15% of optimal in relatively little time, while simple 3-opt (exchanging 3 edges) can get within 3-4% of optimal. The classical Lin-Kernighan algorithm (variable-opt) usually gets within 1-2% of optimal. Variants of Lin-Kernighan can get within 0.1% [6]. Other interesting approaches have been applied to solving the TSP, among these are: Tabu Search (TS), Simulated Annealing (SA), and Genetic Algorithms (GA). These approaches try to use a systematic rule to escape from local minima. A basic ingredient is the use of randomness, which contrasts these approaches to purely deterministic heuristics. These approaches are sometimes referred to as meta-heuristics. Tabu Search is described in [2] as follows:

- Compute an initial tour *T* and start with an initial empty tabu list *L*
- As long as the stopping criteria is not satisfied perform the following steps
    - Perform a move that is not forbidden by *L*
    - Update the tabu list
- Output the best tour found by the heuristic as a solution

Having a tabu list guarantees that it is forbidden (tabu) to return to the same feasible solution. A key aspect of tabu search is the use of special memory structures to organize the way in which the space is explored.

The invention of simulated annealing actually preceded that of tabu search. Like tabu search, simulated annealing allows uphill moves. However, whereas tabu search in essence only makes uphill moves when it is stuck in local optima, simulated annealing can make uphill moves at any time. Moreover, simulated annealing relies heavily on randomization, whereas tabu search in its basic form chooses its next move in a strictly deterministic fashion (except possibly when there is a tie for the best non-tabu neighbor). Simulated Annealing has attracted the interest of many researchers and practitioners from a wide range of disciplines. The technique has its origins in statistical mechanics and it was inspired by the physical process of annealing used for the "cooling" of solids such that they form perfect crystals. SA could be described as a randomized scheme, which reduces the risk of getting trapped in local minima by allowing moves to inferior solutions [6].

The use of genetic algorithms as an approach to optimization can be traced back at least to the 1970's [6]. In this approach, a finite population of solutions is generated randomly or by other means. After that, an iterative process is applied to the population, which at each step transforms the current population to a new population. This involves selecting pairs of parent solutions from the population according to a selection scheme that takes into account their fitness values and combining them to generate offspring solutions. A special type of operator called the crossover or recombination operator (binary transformations) performs the combination of the parents. After the generation of the 'children' a second type of operator called the mutation operator (unary transformations) inflicts random changes upon them. The children are finally inserted in the population by either replacing their parents or by replacing the weakest individuals in the population. This completes an iteration of the GA, which transforms one generation of solutions to the next. The algorithm iterates until a termination criterion is satisfied based either on computational resources, the convergence of the population (high similarity between the solutions contained in the population) or both [11].

1.3 Applications

Despite the fact that the traveling salesman model applies directly to a very useful sounding situation, namely that of a salesman wishing to minimize his travel distance, most of the reported applications are quite different. Many significant real-world problems can be formulated as instances of the TSP. In this section, we introduce some applications of the TSP model. In doing so, various problem transformations may be needed to accommodate the use of the TSP model.

The traveling salesman problem has many applications; from Very Large Scale Integration chip fabrication to X-ray crystallography. The applications described below are merely intended to introduce the versatility of the TSP model. The reader is referred to [1, 2, 12] for a more comprehensive survey of applications and transformation methods.

Vehicle routing: by vehicle routing we mean the problem of determining for a fleet of vehicles which customers should be served by which vehicles, and in what order each vehicle should visit its customers. The Vehicle Routing Problem (VRP) is solvable as a TSP if there is no time constraint or if the number of vehicles is fixed (say m). In this case we obtain an m-salesmen problem. This m-salesmen problem can be transformed into an asymmetric TSP [2]. The VRP can also be solved as a symmetric TSP [12].

The Order-Picking Problem in warehouses: the problem is associated with material handling in warehouses. At a warehouse, an order arrives for a certain subset of the items stored in a warehouse. Some vehicle has to collect all the items of this order to ship them to a customer. The relationship to the TSP is immediately seen. The storage locations of the items correspond to the nodes of the graph. The distance between two nodes is given by the time needed to move the vehicle from one location to the other [2].

Scheduling: consider the problem of sequencing n jobs on a single machine. The time to process job j is t_{ij} if i is the job performed immediately before j (if j is the first job then its processing time is t_{oj}). The task is to find an execution sequence for the jobs, such that, the total processing time is as short as possible. With proper augmentation of the distance matrix, finding the shortest (directed) Hamiltonian path (each vertex is visited exactly once) can be reduced to solving an asymmetric TSP [2]. Another example is the job-shop scheduling problem. We are given n jobs that have to be processed on m machines. Each job consists of a sequence of operations (possibly more than m) where each operation has to be performed on one of the machines. The operations have to be performed one after the other in a sequence that is given in advance. This job-shop problem is restricted where, no passing is allowed (we have the same processing order of jobs on every machine), no intermediate storage is permitted and that each job visits each machine at least once. The problem is to find a schedule for the jobs that minimizes the total processing time. It can be shown that this problem can be transformed to an asymmetric TSP [12].

1.4 Overview of the Thesis

In this thesis, a heuristic for solving the Traveling Salesman Problem is presented. This technique works with a pool of solutions, which has certain properties. Our main goal is to shrink the size of this pool and refine its properties. Finally, we end up with a pool that can no longer produce better solutions from its predecessors. The main property, which we work with, is the frequency of which edges (city-city connections) appear in a pool of solutions. This property becomes our new measure of nearness, and its edge representation acts as a candidate set for constructing future tours.

The thesis is structured as follows. In the next chapter this idea for solving the TSP is introduced along with an implementation that uses a modified version of the Nearest Neighbor algorithm (anchoring edges as we construct tours and recording their frequencies) that works on a complete graph to construct our initial pool of solutions.

Chapter 3 is where computational results on a variety of benchmark instances are shown for both the symmetrical and asymmetrical variants of the TSP. The thesis concludes with Chapter 4 where we comment on our findings and where future research directions are suggested.

CHAPTER 2

METHODOLOGY

In this chapter, a heuristic that adopts Frequency of Anchored Nearest Neighbors (FANN), as a measure of nearness, is introduced. Detailed flowcharts describing each part of the methodology are presented. As each part of the methodology is introduced, outputs for the solution of a 16-city problem, ulysses16 from TSPLIB [13], are listed and discussed. At the end of this chapter, a 33-city problem that first appeared as a contest problem in 1962 is examined and solved.

2.1 Program Flow and Building Blocks

The following sections attempt to realize a simple idea where we work on a pool of solutions that has certain properties. We have two main objectives: shrinking the pool's size and refining its properties. Finally, we end up with a pool that can no longer produce better solutions from its predecessors. The main property, which we work with, is the frequency of which edges (city-city connections) appear in our pool. An implementation of this idea is presented where a modified version of the Nearest Neighbor algorithm (anchoring edges as we construct tours) is used to start us off with an initial pool of solutions.

In our treatment, we frequently use the term *function*. In this context, a function is a black box that creates a pool of solutions. Each function has *input* and *output*. Input to a function is a candidate set of edges (restricts and directs the search when constructing tours) that the function uses to construct its pool of solutions. A property of the function's pool of solutions serves as its output, and in essence is a new candidate set of edges. Functions appear in cascaded order, so the output of one serves as the input for the next.

The flow of the program constructed is illustrated in Figure 2.1 where the three basic building blocks/phases are shown. In subsequent sections, each of these phases is further examined. Figure 2.1 consists of three basic phases:

- Phase 1: the input phase is where the program reads the input data (problem instance) and prepares it for the next phase
- Phase 2: the preparation phase consists of 2 steps, and is where FANN comes to life
- Phase 3: the enumeration phase is where a number of functions are introduced. The program runs through these functions in an enumerative fashion. The program has to loop through this phase at least twice. At the end of each loop, after the first, the termination criterion is checked; if 2 consecutive loops give the same best tour length or we have looped this phase 10 times

The following sections describe what goes on in each of these phases in detail when attempting to solve a symmetrical TSP; reference is made if differences exist for the asymmetrical case. A 16-city STSP problem, ulysses16 from TSPLIB, is introduced and analyzed as we go through each of these phases.

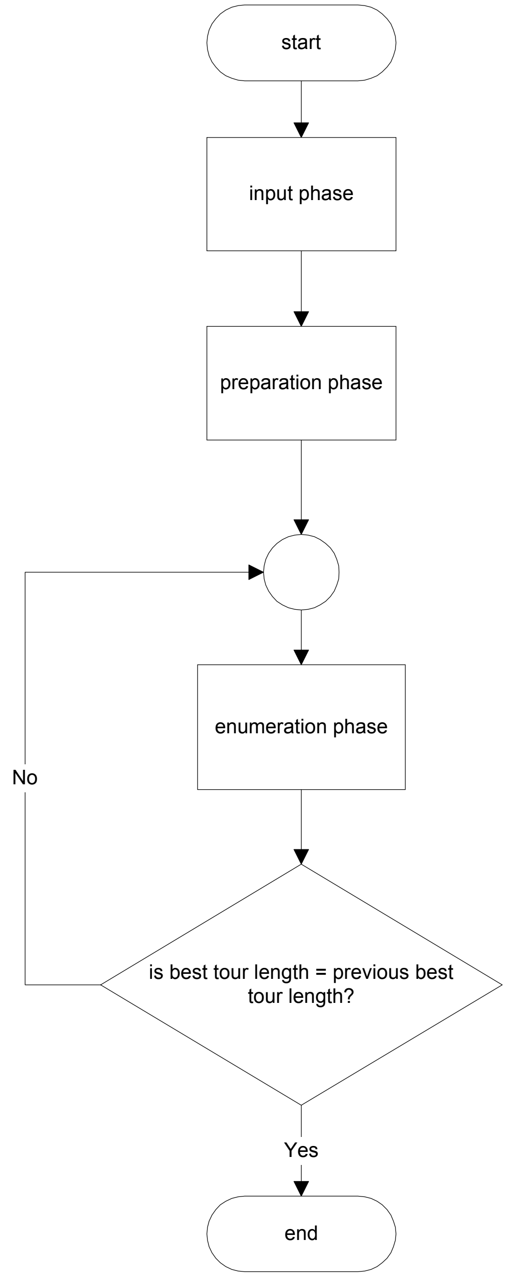


Figure 2.1 Program Flow and Building Blocks

2.1.1 Phase 1: Input Phase

In this phase, the program reads the input data for the instance at hand and prepares it for the next phase. The main purpose of the input phase is to produce a Distance Matrix that serves as a candidate set of edges for constructing tours in the next phase. Figure 2.2 shows the flow for this phase.

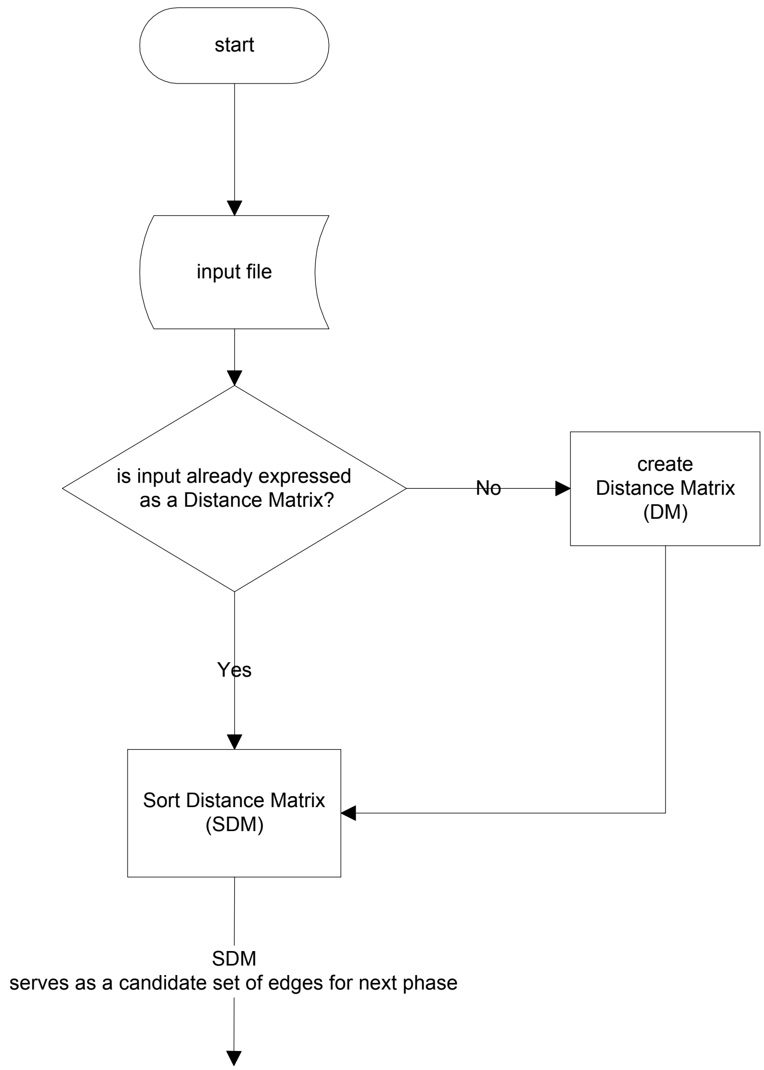


Figure 2.2 Input Phase

The input file for ulysses16 is shown in Figure 2.3. Ulysses16 is a 16-city problem (symmetrical TSP) of type GEO, in other words a geographical problem. The node coordinates give the geographical latitude and longitude of the corresponding point on the earth. Section 3.2.2 explains this input format in detail.

```
NAME: ulysses16.tsp
TYPE: TSP
COMMENT: Odyssey of Ulysses (Groetschel/Padberg)
DIMENSION: 16
EDGE_WEIGHT_TYPE: GEO
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
 1 38.24 20.42
 2 39.57 26.15
 3 40.56 25.32
 4 36.26 23.12
 5 33.48 10.54
 6 37.56 12.19
 7 38.42 13.11
 8 37.52 20.44
 9 41.23 9.10
 10 41.17 13.05
 11 36.08 -5.21
 12 38.47 15.13
 13 38.15 15.35
 14 37.51 15.17
 15 35.49 14.32
 16 39.36 19.56
 EOF
```

Figure 2.3 ulysses16: Input File

The program reads this input file, and since this input format does not express the distances between the 16 cities in matrix form a Distance Matrix (DM) is created. Each entry in the DM is the distance between two cities in kilometers, i.e. their distance on the idealized sphere. The Distance Matrix is created and is as shown in Table 2.1.

Table 2.1 ulysses16: Distance Matrix

Distance of (city#/city#)

1	509	501	312	1019	736	656	60	1039	726	2314	479	448	479	619	150
509	1	126	474	1526	1226	1133	532	1449	1122	2789	958	941	978	1127	542
501	126	1	541	1516	1184	1084	536	1371	1045	2728	913	904	946	1115	499
312	474	541	1	1157	980	919	271	1333	1029	2553	751	704	720	783	455
1019	1526	1516	1157	1	478	583	996	858	855	1504	677	651	600	401	1033
736	1226	1184	980	478	1	115	740	470	379	1581	271	289	261	308	687
656	1133	1084	919	583	115	1	667	455	288	1661	177	216	207	343	592
60	532	536	271	996	740	667	1	1066	759	2320	493	454	479	598	206
1039	1449	1371	1333	858	470	455	1066	1	328	1387	591	650	656	776	933
726	1122	1045	1029	855	379	288	759	328	1	1697	333	400	427	622	610
2314	2789	2728	2553	1504	1581	1661	2320	1387	1697	1	1838	1868	1841	1789	2248
479	958	913	751	677	271	177	493	591	333	1838	1	68	105	336	417
448	941	904	704	651	289	216	454	650	400	1868	68	1	52	287	406
479	978	946	720	600	261	207	479	656	427	1841	105	52	1	237	449
619	1127	1115	783	401	308	343	598	776	622	1789	336	287	237	1	636
150	542	499	455	1033	687	592	206	933	610	2248	417	406	449	636	1

Each row of this DM is then sorted in ascending order; a new table is needed to keep track of the cities we are dealing with. The Sorted Distance Matrix (SDM) is shown in Table 2.2 and its corresponding city representation is shown in Table 2.3. The matrix of Table 2.3 serves as a candidate set of edges for constructing tours in the next phase. More on the importance of this SDM and how it affects optimal solutions is deferred to Chapter 4.

Table 2.2 ulysses16: Sorted Distance Matrix

Distance/distance

1	60	150	312	448	479	479	501	509	619	656	726	736	1019	1039	2314
1	126	474	509	532	542	941	958	978	1122	1127	1133	1226	1449	1526	2789
1	126	499	501	536	541	904	913	946	1045	1084	1115	1184	1371	1516	2728
1	271	312	455	474	541	704	720	751	783	919	980	1029	1157	1333	2553
1	401	478	583	600	651	677	855	858	996	1019	1033	1157	1504	1516	1526
1	115	261	271	289	308	379	470	478	687	736	740	980	1184	1226	1581
1	115	177	207	216	288	343	455	583	592	656	667	919	1084	1133	1661
1	60	206	271	454	479	493	532	536	598	667	740	759	996	1066	2320
1	328	455	470	591	650	656	776	858	933	1039	1066	1333	1371	1387	1449
1	288	328	333	379	400	427	610	622	726	759	855	1029	1045	1122	1697
1	1387	1504	1581	1661	1697	1789	1838	1841	1868	2248	2314	2320	2553	2728	2789
1	68	105	177	271	333	336	417	479	493	591	677	751	913	958	1838
1	52	68	216	287	289	400	406	448	454	650	651	704	904	941	1868
1	52	105	207	237	261	427	449	479	479	600	656	720	946	978	1841
1	237	287	308	336	343	401	598	619	622	636	776	783	1115	1127	1789
1	150	206	406	417	449	455	499	542	592	610	636	687	933	1033	2248

Table 2.3 ulysses16: Sorted Distance Matrix in City Format

(city#/city#)															
1	8	16	4	13	14	12	3	2	15	7	10	6	5	9	11
2	3	4	1	8	16	13	12	14	10	15	7	6	9	5	11
3	2	16	1	8	4	13	12	14	10	7	15	6	9	5	11
4	8	1	16	2	3	13	14	12	15	7	6	10	5	9	11
5	15	6	7	14	13	12	10	9	8	1	16	4	11	3	2
6	7	14	12	13	15	10	9	5	16	1	8	4	3	2	11
7	6	12	14	13	10	15	9	5	16	1	8	4	3	2	11
8	1	16	4	13	14	12	2	3	15	7	6	10	5	9	11
9	10	7	6	12	13	14	15	5	16	1	8	4	3	11	2
10	7	9	12	6	13	14	16	15	1	8	5	4	3	2	11
11	9	5	6	7	10	15	12	14	13	16	1	8	4	3	2
12	13	14	7	6	10	15	16	1	8	9	5	4	3	2	11
13	14	12	7	15	6	10	16	1	8	9	5	4	3	2	11
14	13	12	7	15	6	10	16	8	1	5	9	4	3	2	11
15	14	13	6	12	7	5	8	1	10	16	9	4	3	2	11
16	1	8	13	12	14	4	3	2	7	10	15	6	9	5	11

2.1.2 Phase 2: Preparation Phase

The preparation phase consists of two parts. The first part relates to tour construction. The second part attempts to reduce the number of cities to which each city can connect. A city# - city# connection will be referred to, from now on, as an edge. In this phase we will see what FANN is really all about. Program flow starts with the tour construction step. *Output* of the tour construction step serves as *input* for the first run step.

Starting with the tour construction step, the implementation described herein uses a modified version of the Nearest Neighbor algorithm (anchoring edges as we construct tours) to construct an initial pool of solutions. For a problem with n cities, a solution for the TSP would be a tour of n edges. In constructing a tour we have $n\times(n-1)/2$ possible edges to choose from, a complete graph. For ulusses16 we need a tour of 16 edges and will choose from a total of 120 possible edges. The following discussion assumes we want to search this whole neighborhood of edges (using the candidate set of edges based on SDM which was passed on from the input phase). This approach is definitely time consuming and could be considered as the worst-case scenario as far as running time is concerned. Other scenarios that reduce this search neighborhood (by introducing smaller candidate sets) without affecting solution quality are discussed in Chapter 4.

We start our tour at a city and end our tour at this same city, each tour can choose from $n \times (n-1)/2$ edges. To determine whether or not an edge is chosen (anchored), a tour is constructed. Therefore re-starting at each of the n cities we end up constructing a pool of $n \times n \times (n-1)/2$ tours (reducing the number of starting points also reduces running time, more on this in Chapter 4).

A matrix is formulated that stores the frequency of which each city connects to another, i.e. edge frequencies. This Frequency Matrix will be of size $n \times (n-1)$. In dealing with STSP, if we have a city1 - city2 edge in a tour then two entries in the Frequency Matrix are updated (2-way update): city1 – city2 edge and city2 – city1 edge (for the ATSP case, only city1 – city2 edge frequency is updated). Additional memory structures are used for tracking purposes, to make sure that the tours constructed are legal; no city is visited more than once. In this discussion, only structures that are essential to the understanding of the algorithm are explicitly expressed.

At the end of the tour construction step, we end up with an edge matrix representation of the frequency of which edges appeared in tours of the pool of solutions. This new matrix becomes the search neighborhood for constructing new tours, i.e. this edge matrix representation serves as a candidate set for constructing tours in the next step (the first run). In this context, the frequency of which edges appear in a tour acts as our new measure of nearness; a measure that reflects the chances of a given edge being a member of a future tour.

Now to describe the tour construction part of this phase in more detail; Figure 2.4 shows the function involved and can be explained as follows: Let's start with city1 and see how a tour starting at city1 is constructed.

- `Start tour at city1 (anchor)`
    - `Search for next anchor`
        - `Set as city2 all non-anchored cities (`*n*`-1)`
        - `Finish tour with NN (SDM)`
        - `Calculate tour length`
        - `Update Frequency Matrix`
    - `Anchor as city2 the city that gave shortest tour`

    `Repeat until (`*n*`-1) are anchored`

- `START OVER for `*n*` cities`

`Create SEL from SFM`

Having completed the tour construction part, we end up with a pool of solutions that has the following attributes (these are saved):

- Best tour (solution)
- Best tour length
- Sorted Edge List (SEL): edge matrix representation of the Sorted Frequency Matrix (SFM) that resulted. The SFM was constructed by recording edge frequencies of $n \times n \times (n-1)/2$ tours. This SEL acts as candidate set of edges that is used for constructing tours in the next step (the first run)

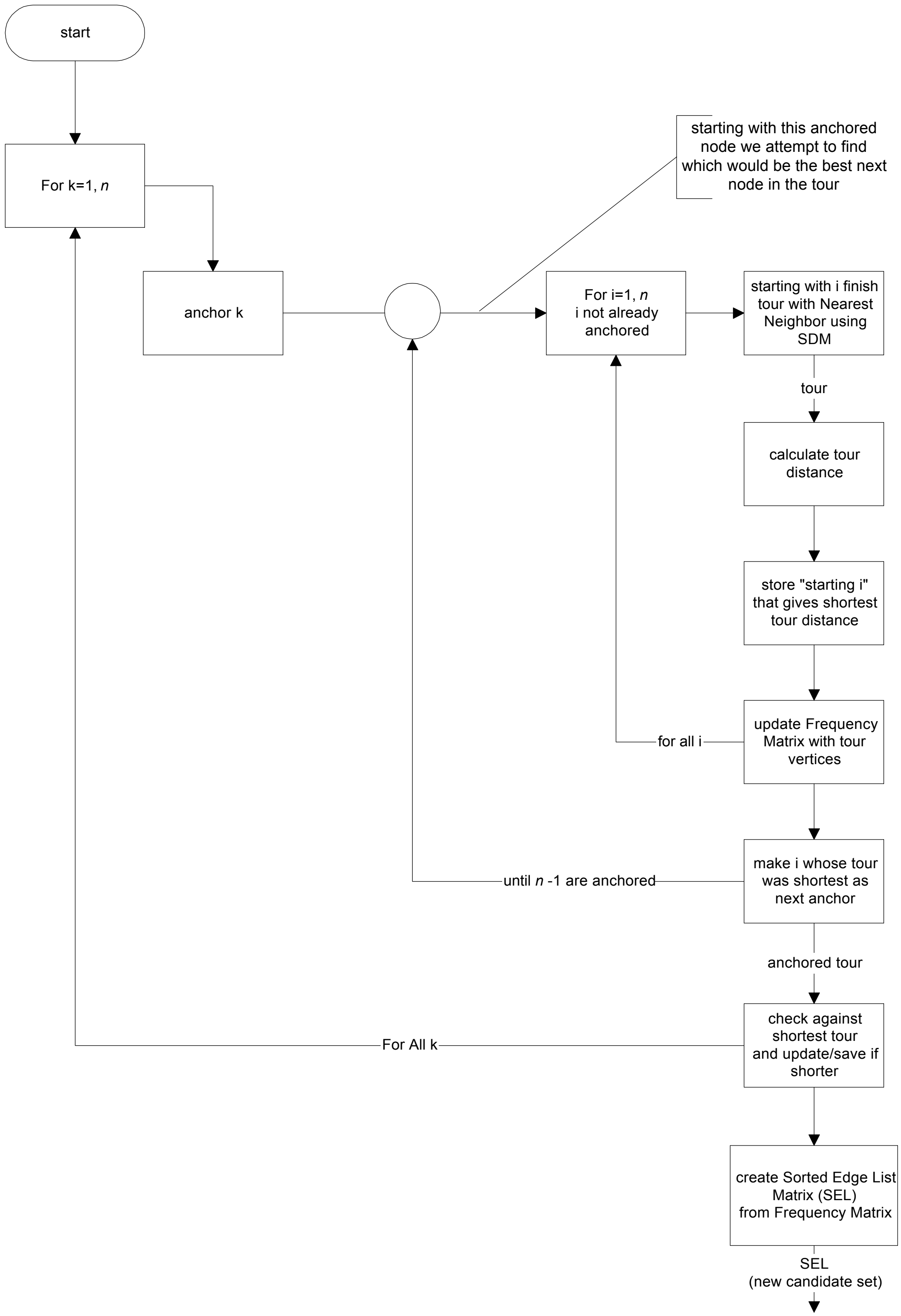


Figure 2.4 Step 1 of Preparation Phase: Tour Construction

The best tour length obtained, for ulysses16, when going through the construction step is 6909. The Frequency Matrix of edges, for ulysses16, is shown in Table 2.4. Each row represents a city and the corresponding frequency of which it connects to other cities, i.e. edge frequencies. Table 2.5 shows the edge list corresponding to this Frequency Matrix.

Table 2.4 ulysses16: Frequency Matrix for Step 1 of Preparation Phase

City#/frequency															
1	19	126	141	166	17	17	1553	75	21	182	17	36	23	34	1413
2	19	1733	1562	23	20	19	25	24	25	253	21	30	18	46	22
3	126	1733	123	49	123	123	207	51	263	432	127	41	123	61	258
4	141	1562	123	70	18	18	1248	76	33	207	19	53	23	34	215
5	166	23	49	70	207	27	254	318	558	502	25	18	161	1359	103
6	17	20	123	18	207	1640	17	51	271	99	197	19	436	705	20
7	17	19	123	18	27	1640	17	143	448	105	1003	46	167	47	20
8	1553	25	207	1248	254	17	17	20	21	164	17	32	17	62	186
9	75	24	51	76	318	51	143	20	1509	931	25	22	20	207	368
10	21	25	263	33	558	271	448	21	1509	130	295	23	27	111	105
11	182	253	432	207	502	99	105	164	931	130	135	99	133	169	299
12	17	21	127	19	25	197	1003	17	25	295	135	1040	570	305	44
13	36	30	41	53	18	19	46	32	22	23	99	1040	1599	122	660
14	23	18	123	23	161	436	167	17	20	27	133	570	1599	487	36
15	34	46	61	34	1359	705	47	62	207	111	169	305	122	487	91
16	1413	22	258	215	103	20	20	186	368	105	299	44	660	36	91

Table 2.5 ulysses16: Edge List Matrix for Step 1 of Preparation Phase

City#/city#															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	1	2	4	5	6	7	8	9	10	11	12	13	14	15	16
4	1	2	3	5	6	7	8	9	10	11	12	13	14	15	16
5	1	2	3	4	6	7	8	9	10	11	12	13	14	15	16
6	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16
7	1	2	3	4	5	6	8	9	10	11	12	13	14	15	16
8	1	2	3	4	5	6	7	9	10	11	12	13	14	15	16
9	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16
10	1	2	3	4	5	6	7	8	9	11	12	13	14	15	16
11	1	2	3	4	5	6	7	8	9	10	12	13	14	15	16
12	1	2	3	4	5	6	7	8	9	10	11	13	14	15	16
13	1	2	3	4	5	6	7	8	9	10	11	12	14	15	16
14	1	2	3	4	5	6	7	8	9	10	11	12	13	15	16
15	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16
16	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The main purpose of the tour construction step is to introduce a new measure of nearness that is based on frequency. Therefore, for each row of this Frequency Matrix, we reorder the frequency entries in descending order, i.e. higher frequency occurrences first. This reordered matrix is called the Sorted Frequency Matrix (SFM) and is shown in Table 2.6. The edges corresponding to these frequencies are displayed in Table 2.7. This matrix is referred to as the Sorted Edge List Matrix (SEL), and serves as the candidate set that will be manipulated when constructing future tours (of the first run). From the resulting pool of solutions, the best tour (city sequence), best tour length, and SEL are updated as attributes of our new best solution.

Table 2.6 ulysses16: SFM for Step 1 of Preparation Phase

City#/Frequency

1	1553	1413	182	166	141	126	75	36	34	23	21	19	17	17	17
2	1733	1562	253	46	30	25	25	24	23	22	21	20	19	19	18
3	1733	432	263	258	207	127	126	123	123	123	123	61	51	49	41
4	1562	1248	215	207	141	123	76	70	53	34	33	23	19	18	18
5	1359	558	502	318	254	207	166	161	103	70	49	27	25	23	18
6	1640	705	436	271	207	197	123	99	51	20	20	19	18	17	17
7	1640	1003	448	167	143	123	105	47	46	27	20	19	18	17	17
8	1553	1248	254	207	186	164	62	32	25	21	20	17	17	17	17
9	1509	931	368	318	207	143	76	75	51	51	25	24	22	20	20
10	1509	558	448	295	271	263	130	111	105	33	27	25	23	21	21
11	931	502	432	299	253	207	182	169	164	135	133	130	105	99	99
12	1040	1003	570	305	295	197	135	127	44	25	25	21	19	17	17
13	1599	1040	660	122	99	53	46	41	36	32	30	23	22	19	18
14	1599	570	487	436	167	161	133	123	36	27	23	23	20	18	17
15	1359	705	487	305	207	169	122	111	91	62	61	47	46	34	34
16	1413	660	368	299	258	215	186	105	103	91	44	36	22	20	20

Table 2.7 ulysses16: SEL for Step 1 of Preparation Phase

City#/city#

1	8	16	11	5	4	3	9	13	15	14	10	2	7	12	6
2	3	4	11	15	13	10	8	9	5	16	12	6	1	7	14
3	2	11	10	16	8	12	1	7	4	14	6	15	9	5	13
4	2	8	16	11	1	3	9	5	13	15	10	14	12	7	6
5	15	10	11	9	8	6	1	14	16	4	3	7	12	2	13
6	7	15	14	10	5	12	3	11	9	2	16	13	4	1	8
7	6	12	10	14	9	3	11	15	13	5	16	2	4	1	8
8	1	4	5	3	16	11	15	13	2	10	9	7	14	6	12
9	10	11	16	5	15	7	4	1	6	3	12	2	13	8	14
10	9	5	7	12	6	3	11	15	16	4	14	2	13	8	1
11	9	5	3	16	2	4	1	15	8	12	14	10	7	13	6
12	13	7	14	15	10	6	11	3	16	5	9	2	4	8	1
13	14	12	16	15	11	4	7	3	1	8	2	10	9	6	5
14	13	12	15	6	7	5	11	3	16	10	4	1	9	2	8
15	5	6	14	12	9	11	13	10	16	8	3	7	2	4	1
16	1	13	9	11	3	4	8	10	5	15	12	14	2	6	7

Now we consider the next step of the preparation phase, the *first run*. In the first run, a pool of solutions is constructed from n anchored tours. Edge frequencies of these anchored tours are recorded into a Frequency Matrix. An edge representation of this Frequency Matrix is then constructed (SEL). The implementation details are as follows: The function used in this step is similar to one used for the tour construction step shown in Figure 2.4, but here it constructs tours according to the SEL (candidate set that resulted from the tour construction step) instead of the SDM (initial candidate set from input phase). In this first run, the frequency of which edges appear in a solution is updated, but only for anchored tours. The flowchart for this first run function is shown in Figure 2.5. This first run has a main purpose: to decrease the number of edges in its resulting SEL. Hence, the number of entries in the SEL, after running this step, is less than $n \times (n\text{-}1)$. This enables all future manipulations of the resulting SEL to run faster. The steps involved can be described as follows:

- Start tour at city1 (anchor)
    - o Search for next anchor
        - Set as city2 all non-anchored cities (n-1)
        - Finish tour with NN (SEL)
        - Calculate tour length
    - o Anchor as city2 the city that gave shortest tour

  Repeat until (n-1) are anchored
  Update Frequency Matrix with edges of anchored tour + save best solution

- START OVER for n cities

Create SEL from SFM

Having completed the first run, we end up with a pool of solutions that has the same three attributes:

- Best tour (solution)
- Best tour length

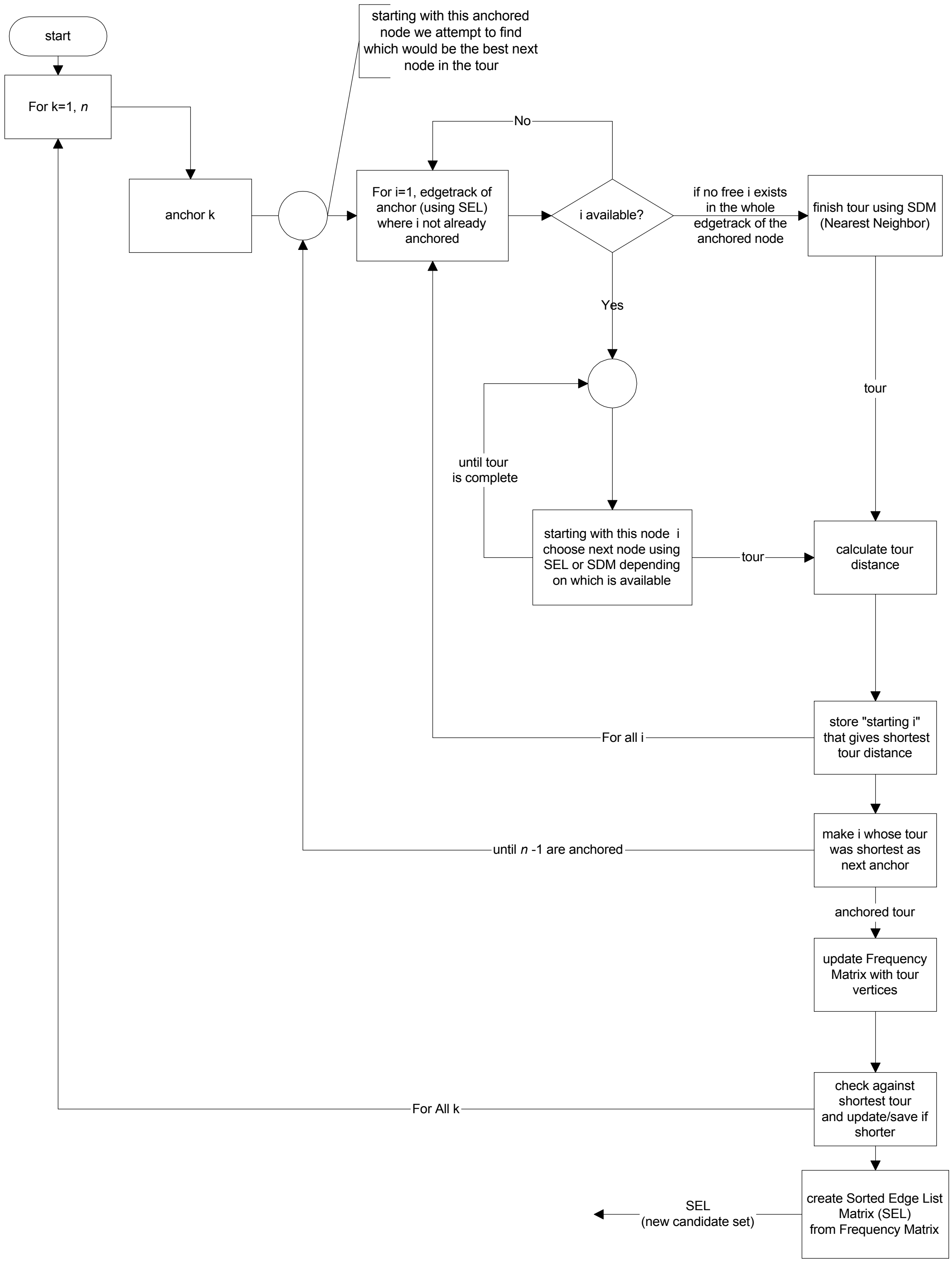


Figure 2.5 Step 2 of Preparation Phase: First Run

- Sorted Edge List: edge matrix representation of the Sorted Frequency Matrix that resulted. The SFM was constructed by recording edge frequencies of *n* anchored tours

The only attribute that is of interest to us from this pool of solutions is its SEL, which serves as input for the enumeration. Figure 2.6 summarizes the preparation phase.

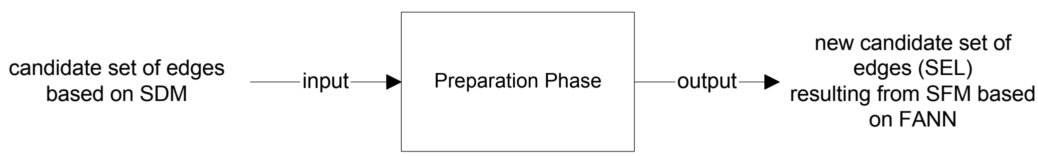


Figure 2.6 Preparation Phase Summary

The best tour length obtained for ulysses16 when going through the first run is 6870. Table 2.8 shows the Frequency Matrix reordered according to highest frequency occurrences first. The edges corresponding to these frequencies are displayed in Table 2.9. This resulting Sorted Edge List Matrix has less number of edges for each row as anticipated (smaller candidate set). The city with the most number of edges is city10 with 11 edges.

Table 2.8 ulysses16: SFM for Step 2 of Preparation Phase

City#/Frequency											
1	15	10	4	1	1	1					
2	16	15	1								
3	16	7	4	2	1	1	1				
4	15	15	1	1							
5	16	15	1								
6	16	6	5	2	1	1	1				
7	16	9	4	3							
8	15	15	1	1							
9	16	14	2								
10	14	4	4	2	2	1	1	1	1	1	1
11	16	16									
12	9	9	5	5	2	1	1				
13	14	9	7	1	1						
14	14	5	5	4	3	1					
15	15	6	4	4	2	1					
16	10	7	7	5	2	1					

Table 2.9 ulysses16: SEL for Step 2 of Preparation Phase

City#/city#											
1	8	16	15	13	10	12					
2	3	4	10								
3	2	16	10	15	4	6	8				
4	2	8	3	10							
5	11	15	6								
6	7	15	14	10	12	3	5				
7	6	12	10	14							
8	1	4	3	10							
9	11	10	12								
10	9	7	3	6	16	1	8	4	13	14	2
11	5	9									
12	7	13	14	16	9	6	1				
13	14	12	16	10	1						
14	13	12	6	15	7	10					
15	5	6	1	14	3	16					
16	1	3	13	12	10	15					

2.1.3 Phase 3: Enumeration Phase

This phase manipulates the SEL in a fashion shown in Figure 2.7 (for the STSP). Figure 2.7 displays a partial enumeration of all the possible paths through this tree structure. In this tree structure, we have 15 different functions arranged in 4 columns, each having 9, 11, 3, 2 functions (nodes), respectively (for STSP we have $9 \times 11 \times 3 \times 2 = 594$ possible paths through this tree).

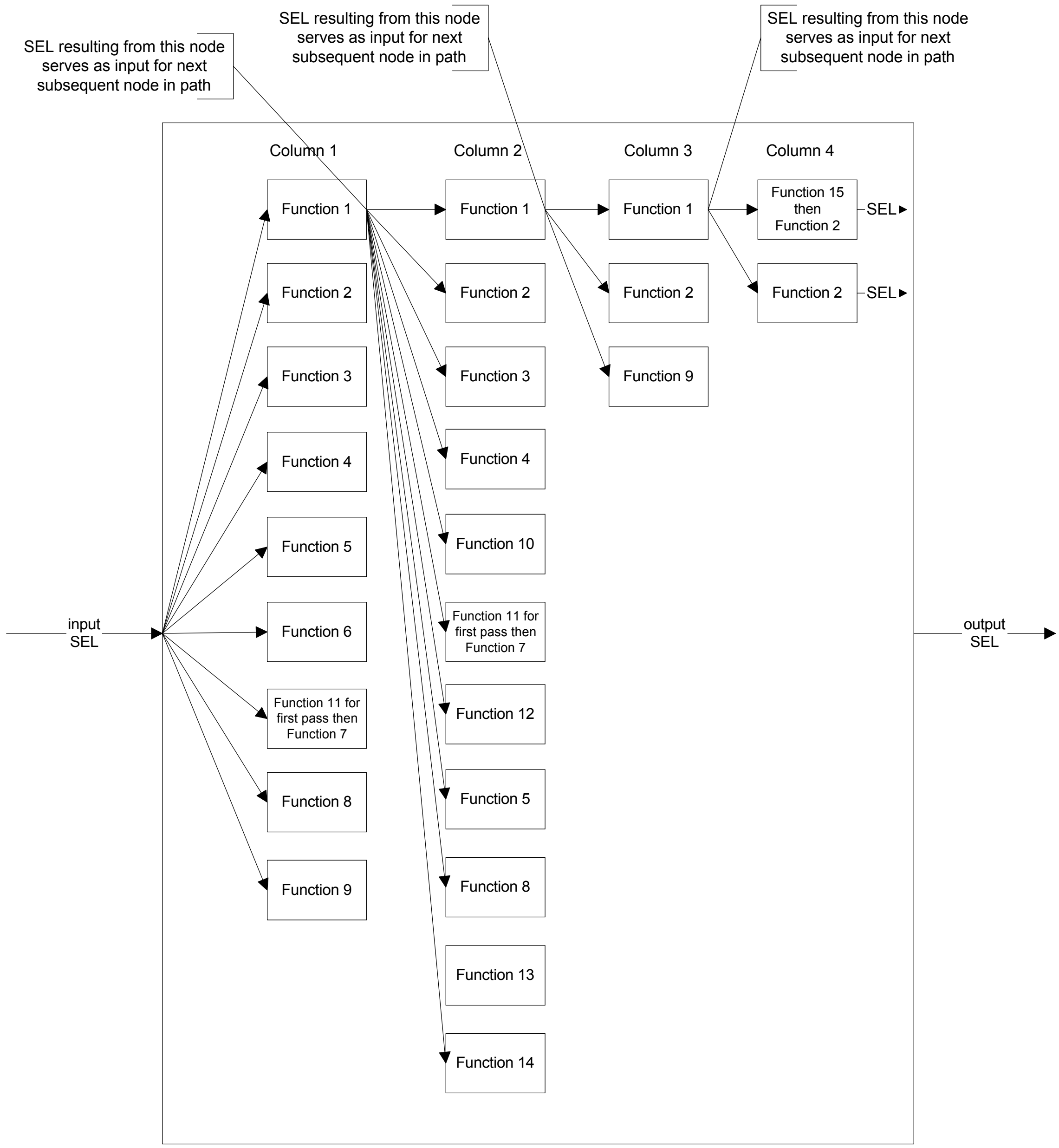


Figure 2.7 STSP Enumeration Phase: Partial Enumeration through Tree Structure

Every enumeration path is formed by passing through the 4 columns (one node in every column). Each node constructs a new pool of solutions, which consists of *n* anchored tours (at each node, FANN is the measure of nearness used to produces a new SEL, i.e. a candidate set for constructing tours in the next node of the enumeration path). Not all 15 functions appear in each column. Actually, these functions can be grouped under 2 general types that will be examined shortly. This layout of the 15 functions is not concrete, in a sense that, some columns could have fewer functions; hence less time would be needed to go through this phase, more on this in Chapter 4. For the ATSP case the layout of nodes differs slightly than that of Figure 2.7 to 11, 12, 4 and 2 nodes appearing in columns 1-4, respectively.

Enumerating through this tree, and at each node of every path (594 possible paths, each having 4 nodes), a new pool of solutions results that has the same three attributes:

- Best tour (solution)
- Best tour length
- Sorted Edge List: edge matrix representation of the Sorted Frequency Matrix that resulted from *n* anchored tours, which serves as a candidate set for constructing tours in the next node of the enumeration path

When branching through this tree structure, and at each node of every path, we need to check whether the resulting pool (from a node) has better attributes than the one already saved as *best*. It should be kept in mind that the resulting tour length and SEL attributes, of this new pool, are the main factors that decide whether or not we need to update our previously saved best pool attributes:

- If the new tour length is less than the stored best tour length then the new pool's attributes are saved and become the new best pool attributes

- If the new tour length equals the stored best tour length then the two SELs are compared. If the new pool has an SEL with more entries, then it is chosen to be the new best (attributes are updated as necessary). The reasoning behind this, is that, for two pools having the same tour length attribute, the one that has more entries in its SEL is less likely to cause nodes, which use this SEL as input, to get stuck at a local minimum. It's like having a neighborhood for local search; the bigger the neighborhood we search (more entries in the input SEL), the better our chances of stumbling across a *good* solution

Having completed the enumeration phase, the first time (pass1), we end up with attributes of a so-called *best pool*. Now, we need to go through this phase again. The stored best pool's SEL (from previous pass, pass1 in this case) serves as input for all nodes in the first column of the second pass. If the best tour length attribute obtained for the second time around the enumeration phase is equal to the first pass's best tour length, the program stops, otherwise it keeps on looping until the best pools of two consecutive passes have the same tour length attribute or we have looped 10 times, whichever comes first (termination criterion). Once the termination criterion is satisfied, the program stops and we end up with a pool whose best tour attribute we consider as the 'BEST TOUR' the algorithm can provide.

The 15 functions previously referred to, can be categorized of being one of two types:

- Type 1: a flowchart for this type is shown in Figure 2.8, which is identical to the function of Figure 2.5 but adds to it that the Frequency Matrix is also updated

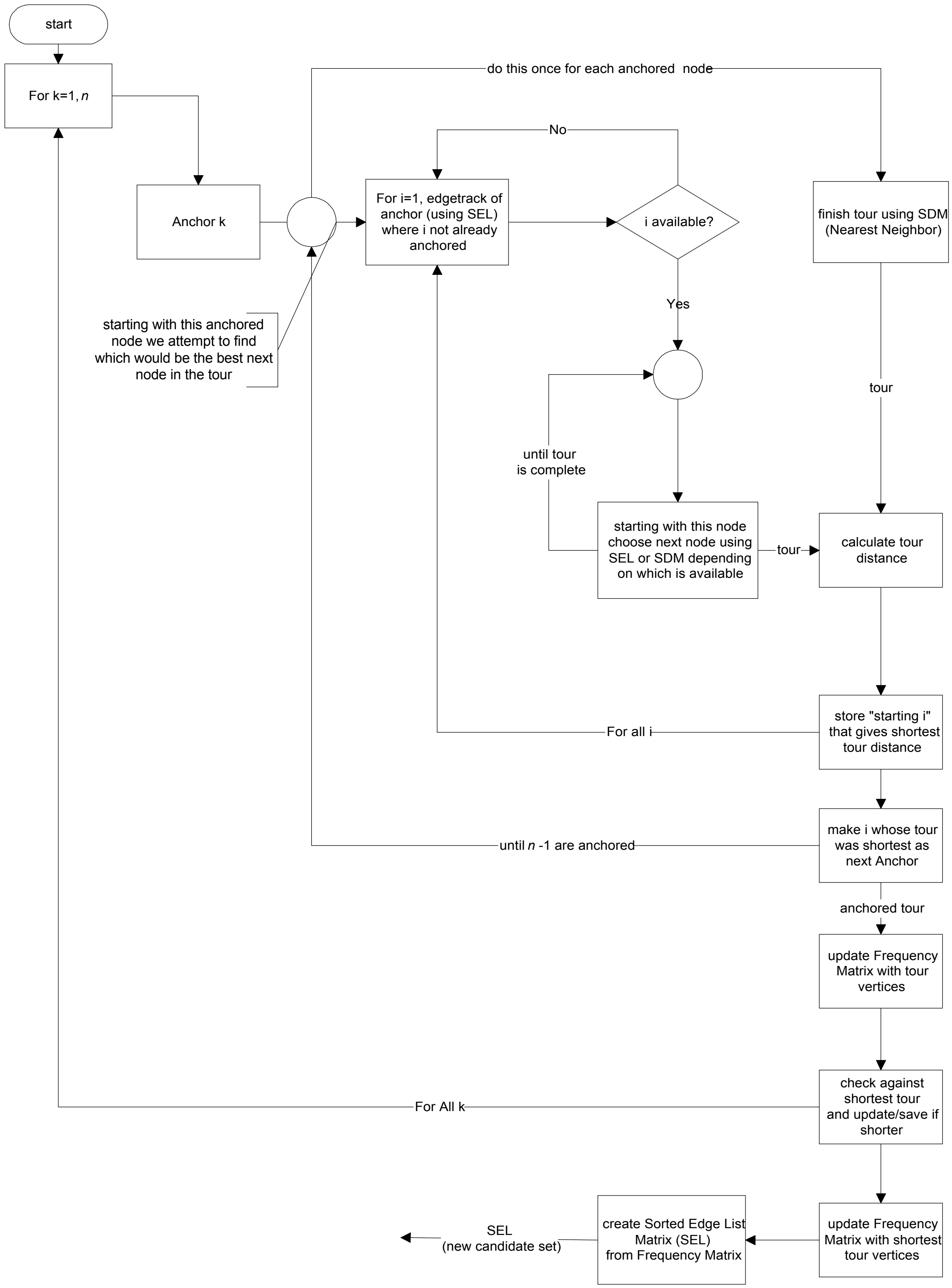


Figure 2.8 Function Type 1

with edges of the best tour (best of the *n* anchored tours). This function can be described as follows:

```
•  Start tour at city1 (anchor)
      o  Search for next anchor
            ▪  Set as city2 all non-anchored cities from candidate set
               of city1 (SEL)
            ▪  Finish tour with NN (SEL) or NN (SDM) as necessary
            ▪  Calculate tour length
      o  Anchor as city2 the city that gave shortest tour
      Repeat until (n-1) are anchored
      Update Frequency Matrix with edges of anchored tour + save best
      solution

•  START OVER for n cities
•  Update Frequency Matrix with edges of best tour
Create SEL from SFM
```

Variants of this type are constructed where:

- Different weights are given to the edges of the best tour (best of the *n* anchored tours) when updating the Frequency Matrix, i.e. +1, +*n*, etc.
- Only half or two thirds of a tour's vertices are anchored, the rest of the tour is constructed via NN that manipulates the SDM. Anchoring fewer nodes requires less time, and finishing a tour with NN that uses SDM introduces variability
- For the ATSP case, we have the advantage that we can introduce variability by having a two-way edge frequency update instead of the regular one-way update

- Type 2: a flowchart for this type is shown in Figure 2.9. This type is more involved than the previous type and can be described as follows:

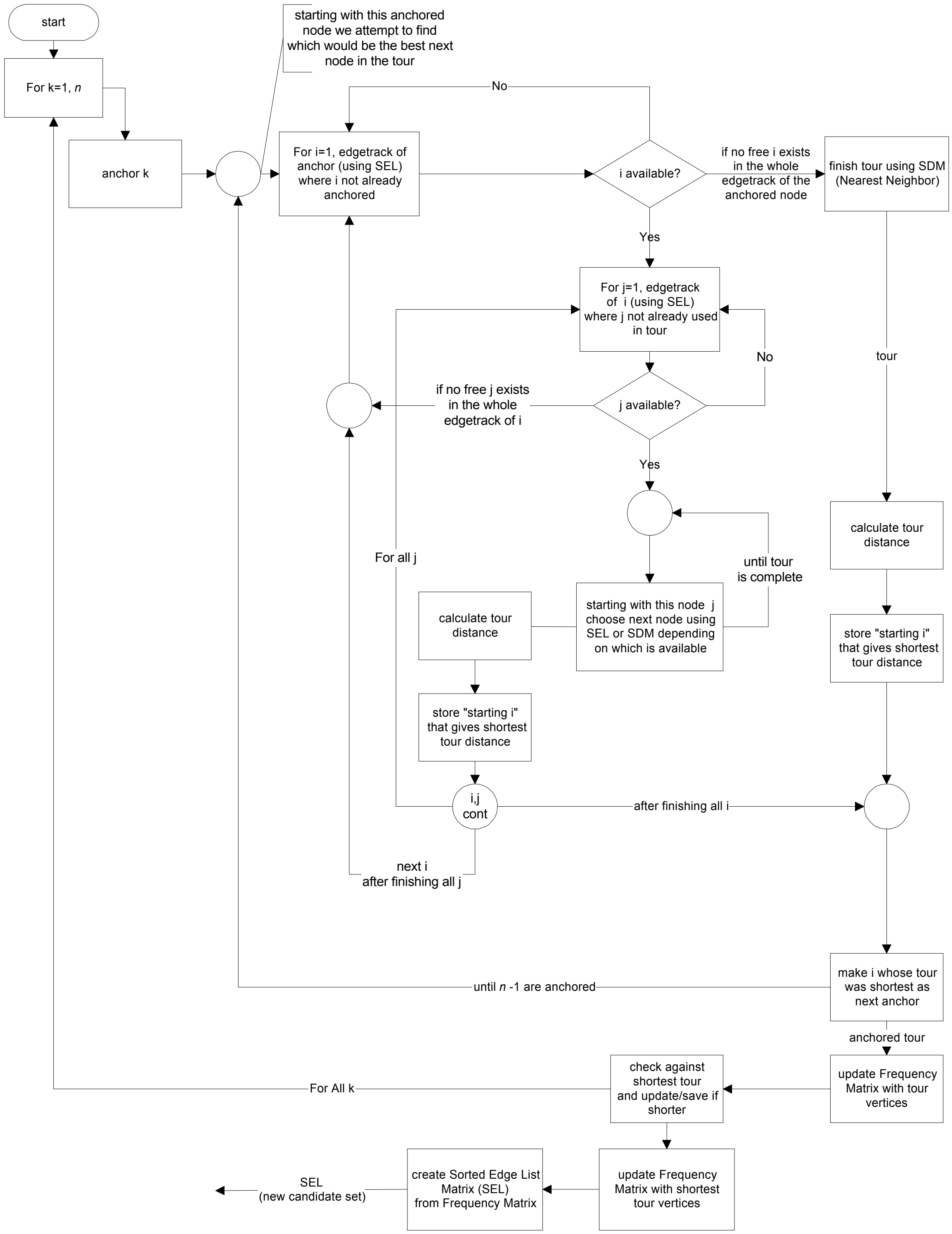


Figure 2.9 Function Type 2

```
•   Start tour at city1 (anchor)
        o  Search for next anchor
               ▪  Set as city2 all non-anchored cities from candidate set
                  of city1 (SEL)
               ▪  LOOK AHEAD: Set as city3 all city2 non-anchored cities
                  (SEL)
                      •  Finish tour with NN (SEL) or NN (SDM) as
                         necessary
                      •  Calculate tour length
        o  Anchor as city2 the city that gave shortest tour
        Repeat until (n-1) are anchored
        Update Frequency Matrix with edges of anchored tour + save best
        solution

•   START OVER for n cities
•   Update Frequency Matrix with edges of best tour
Create SEL from SFM
```

Function Type 2 performs an extra level search when attempting to anchor a city, in that, it looks ahead one level and constructs tours starting at each of the non-anchored entries in the edge list of this anchor candidate, rather than just connecting to the first non-anchored entry in the SEL. Methods used to construct variants of this type are similar to the methods used for constructing variants of Function Type 1

In the following discussion, we show outputs for ulysses16 as we construct an enumeration path that passes through the first node of columns 1-4 of Figure 2.7. The best tour length obtained when going through the first node of the enumeration tree (Function1 in column1) in Figure 2.7 is 6865. Table 2.10 shows the Frequency Matrix reordered according to highest frequency occurrences first. The edges corresponding to these frequencies are displayed in Table 2.11. This Sorted Edge List Matrix has less number of edges for each row as anticipated. The highest number of edges is 6 for both city6 and city10. The tour length, tour itself (city sequence), and SEL are updated as attributes of our new best pool (since 6865 < 6870).

Table 2.10 ulysses16: SFM after Function1 in First Column

City#/Frequency						
1	32	18	9	5		
2	32	30	2			
3	32	27	3	2		
4	32	30	2			
5	32	30	2			
6	32	23	3	2	2	2
7	32	25	4	3		
8	32	32				
9	32	32				
10	32	20	4	3	3	2
11	32	32				
12	25	25	7	5	2	
13	32	18	7	7		
14	32	25	3	2	2	
15	30	23	9	2		
16	27	20	7	5	5	

Table 2.11 ulysses16: SEL after Function1 in First Column

City#/city#						
1	8	13	15	16		
2	3	4	10			
3	2	16	10	4		
4	8	2	3			
5	11	15	6			
6	7	15	10	12	14	5
7	6	12	10	14		
8	1	4				
9	10	11				
10	9	16	7	6	3	2
11	5	9				
12	7	14	13	16	6	
13	14	1	12	16		
14	13	12	7	6	15	
15	5	6	1	14		
16	3	10	13	12	1	

Next in our path is Function1 (node1) in column2 of Figure 2.7. The previous SEL (that resulted from Function1 of column1) serves as input for this Function1 of column2. The best tour length obtained is 6859; it will be shown in Chapter 3 that this is the optimal tour length for this instance. Table 2.12 shows the Frequency Matrix reordered according to highest frequency occurrences first. The edges corresponding to these frequencies are displayed in Table 2.13. This resulting Sorted Edge List Matrix has less number of edges for each row. Actually most of them only have 2 edges, which is the least that any city entry could have, since each city in a tour needs to connect to two other cities. The tour length, tour itself (city sequence), and SEL are updated as attributes of our new best pool (since 6859 < 6865).

Table 2.12 ulysses16: SFM after Function1 in Second Column

City#/Frequency			
1	32	19	13
2	32	32	
3	32	32	
4	32	32	
5	32	32	
6	32	32	
7	32	32	
8	32	32	
9	32	32	
10	32	32	
11	32	32	
12	32	19	13
13	32	19	13
14	32	19	13
15	32	32	
16	32	32	

Table 2.13 ulysses16: SEL after Function1 in Second Column

City#/city#			
1	8	14	13
2	3	4	
3	2	16	
4	2	8	
5	11	15	
6	7	15	
7	6	12	
8	1	4	
9	10	11	
10	9	16	
11	5	9	
12	7	13	14
13	14	12	1
14	13	1	12
15	5	6	
16	3	10	

Next in our path is Function1 (node1) in column3 of Figure 2.7. The SEL that resulted from Function1 of column2 serves as input for this function. The best tour length obtained is again 6859. Table 2.14 shows the Frequency Matrix reordered according to highest frequency occurrences first. The edges corresponding to these frequencies are displayed in Table 2.15. This resulting Sorted Edge List Matrix has less number of edges than that of the previous pool, so the best pool attributes are not updated. Table 2.15 clearly shows how the edge list converges towards the same first two entries of the SEL of Table 2.13. The last node in this enumeration path would be the first node of column4.

Table 2.14 ulysses16: SFM after Function1 in Third Column

City#/Frequency		
1	32	32
2	32	32
3	32	32
4	32	32
5	32	32
6	32	32
7	32	32
8	32	32
9	32	32
10	32	32
11	32	32
12	32	32
13	32	32
14	32	32
15	32	32
16	32	32

Table 2.15 ulysses16: SEL after Function1 in Third Column

City#/city#		
1	8	14
2	3	4
3	2	16
4	2	8
5	11	15
6	7	15
7	6	12
8	1	4
9	10	11
10	9	16
11	5	9
12	7	13
13	12	14
14	1	13
15	5	6
16	3	10

Figure 2.10 is a pictorial representation of the solution progress (partial enumeration is illustrated for the best trail, in addition to a few branches along the way), for ulysses16, through the first pass of the enumeration phase. Figure 2.10 shows the two

attributes involved in evaluating whether a resulting pool should be considered for *best pool*: its best tour length and the number of entries in its resulting SEL. The

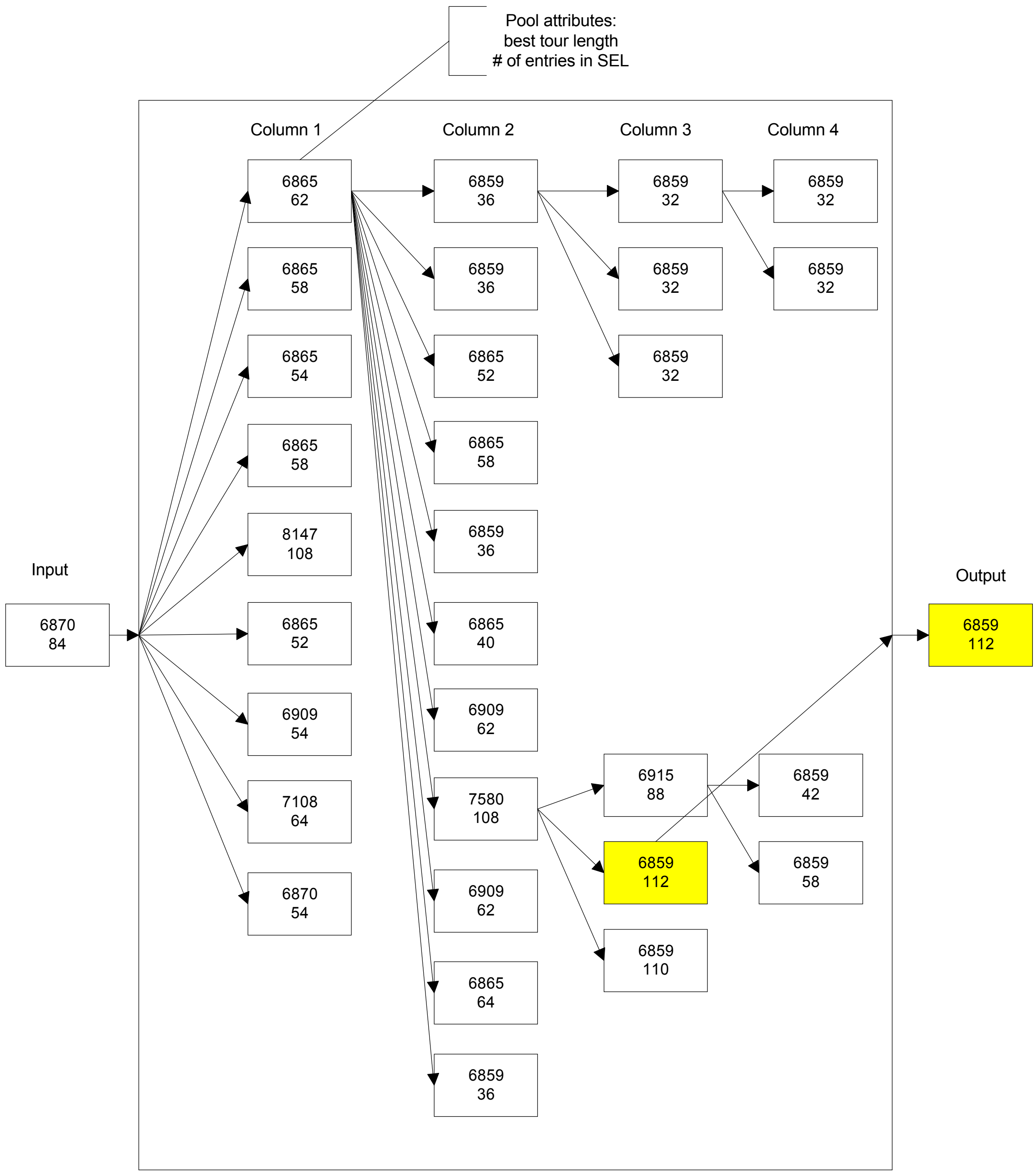


Figure 2.10 ulysses16: Solution Progress in First Pass of Enumeration Phase – Partial Branching Illustrating Enumeration Trail for Best Solution

node (within an enumeration path) that produced the best attributes is the one whose attributes are saved and passed on as input for pass2.

Figure 2.11 shows the solution trail in each of the two passes through the enumeration phase that ended up giving the best solution for ulysses16. Let's explain what the results for the first pass mean: 'best' is the best tour length obtained in the first pass and is 6859. The sequence 1, 8, 2, 0 depicts the solution trail (enumeration path) referencing Figure 2.7, i.e. this best solution came about as the program branched through the tree, and constructed a path that passed node1 of column1, node8 of column2 and node2 of column3. A pictorial representation of this trail is shown in Figure 2.10. This best solution, which resulted from node2 of column3, had started its tour from city9. The best solution (city tour) is then listed. A graphical display of this best tour can be found in Figure 6.40 of Appendix B.

```
Results for PASS 1:
best is  6859 starting at  9 in loop       1, 8, 2, 0

Results for PASS 2:
best is  6859 starting at  5 in loop       3, 8, 2, 0

The best solution obtained is: 6859
The solution vector is as follows
 1
 8
 4
 2
 3
 16
 10
 9
 11
 5
 15
 6
 7
 12
 13
 14
 -1
```

Figure 2.11 ulysses16: Best Solution

In conclusion, each of the 4 nodes in every enumeration path produces a candidate set based on FANN. This candidate set is used to construct tours for the next node in the path (input for next node), and so on. This ongoing refinement of candidate sets (SEL), based on FANN (new solutions are the product of past successful solutions), to produce high quality solutions, is the main objective of the enumeration phase.

2.2 33-city Problem Example

In this section, we consider another instance of the GEO TSP, to give an idea of what other factors could affect a solution. "Proctor and Gamble ran a contest in 1962. The contest required solving a TSP on a specified 33 cities. There was a tie between many people who found the optimum. An early TSP researcher, Gerald Thompson, was one of the winners" [14]. Figure 2.12 states the problem in a concise manner.

In order to solve this problem, we need the geographical co-ordinates of the cities involved. Unfortunately [14] does specify if geographical co-ordinates were originally provided and what the official rules of the contest were. Looking up all the cities in question, Table 2.16 displays their latitude and longitude co-ordinates.

This is a STSP problem of type GEO, as described in 3.3.2, which has been solved by means of our current implementation of FANN. In the following paragraphs, we give a *C*-implementation for the computations involved. Let x[i] and y[i] be coordinates for city *i* in the above format. First the input is converted to geographical

Figure 2.12 Proctor and Gamble TSP Contest Problem [14]

Table 2.16 33-city Problem: City Co-ordinates

	City	State	Lattitude	Longitude
1	Chicago	Illinois	41.51	-87.39
2	Indianapolis	indiana	39.46	-86.09
3	Marion	Ohio	40.35	-83.07
4	Erie	Pennsylvania	42.07	-80.05
5	Carlisle	Pennsylvania	40.12	-77.11
6	Wana	West Virginia	39.42	-80.17
7	Wilkesboro	North Carolina	36.08	-81.09
8	Barnwell	South Carolina	33.14	-81.21
9	Bainbridge	Georgia	30.54	-84.34
10	Baton Rouge	Louisiana	30.27	-91.09
11	Little Rock	Arkansas	34.44	-92.17
12	Chattanooga	Tennessee	35.02	-85.18
13	Kansas City	Missouri	39.05	-94.34
14	La Crosse	Wisconsin	43.48	-91.14
15	Blunt	South Dakota	44.30	-99.59
16	Lincoln	Nebraska	40.48	-96.40
17	Wichita	Kansas	37.41	-97.20
18	Amarillo	Texas	35.13	-101.49
19	Truth Or Consequences	New Mexico	33.08	-107.14
20	Manuelito	New Mexico	35.25	-108.59
21	Colorado Springs	Colorado	38.50	-104.49
22	Marble Canyon	Arizona	36.48	-111.38
23	Mexican Hat	Utah	37.07	-109.53
24	Salt Lake City	Utah	40.45	-111.53
25	Twin Falls	Idaho	42.33	-114.27
26	Boise	Idaho	43.36	-116.12
27	Butte	Montana	46.00	-112.32
28	Lewiston	Idaho	46.25	-117.01
29	Portland	Oregon	45.31	-122.40
30	Redding	California	40.35	-122.23
31	Reno	Nevada	39.31	-119.48
32	Gustine	California	37.15	-120.59
33	Lone Pine	California	36.36	-118.03

latitude and longitude given in radians as follows:

```
PI = 3.141592;
deg = nint( x[i] );
min = x[i] - deg;
latitude[i] = PI * (deg + 5.0 * min / 3.0 ) / 180.0;
deg = nint( y[i] );
min = y[i] - deg;
longitude[i] = PI * (deg + 5.0 * min / 3.0 ) / 180.0;

The function 'nint' truncates a real number and takes its integer part
```

The distance between two cities (*i* and *j*) in kilometers, i.e. their distance on the idealized sphere is then computed as follows:

```
RRR = 6378.388;
q1 = cos( longitude[i] - longitude[j] );
q2 = cos( latitude[i] - latitude[j] );
q3 = cos( latitude[i] + latitude[j] );
dij = nint(  RRR * acos( 0.5*((1.0 + q1)*q2 - (1.0 - q1)*q3) ) +  1.0);

The function 'acos' is the inverse of the cosine function
The function 'nint' truncates a real number and takes its integer part
```

Running the heuristic, the best tour length obtained was 13064. The best tour is shown in Table 2.17 and is graphed in Figure 2.13.

Table 2.17 33-city Problem: Best Tour - Truncating

Best Tour	City	State
1	Chicago	Illinois
2	Indianapolis	Indiana
3	Marion	Ohio
4	Erie	Pennsylvania
5	Carlisle	Pennsylvania
6	Wana	West Virginia
7	Wilkesboro	North Carolina
8	Barnwell	South Carolina
12	Chattanooga	Tennessee
9	Bainbridge	Georgia
10	Baton Rouge	Louisiana
11	Little Rock	Arkansas
13	Kansas City	Missouri
16	Lincoln	Nebraska
17	Wichita	Kansas
18	Amarillo	Texas
19	Truth Or Consequences	New Mexico
20	Manuelito	New Mexico
23	Mexican Hat	Utah
22	Marble Canyon	Arizona
33	Lone Pine	California
32	Gustine	California
31	Reno	Nevada
30	Redding	California
29	Portland	Oregon
28	Lewiston	Idaho
27	Butte	Montana
26	Boise	Idaho
25	Twin Falls	Idaho
24	Salt Lake City	Utah
21	Colorado Springs	Colorado
15	Blunt	South Dakota
14	La Crosse	Wisconsin
-1	Chicago	Illinois

In calculating the distance between two cities (*i* and *j*) in kilometers, i.e. in:

```
dij = nint(  RRR * acos( 0.5*((1.0 + q1)*q2 - (1.0 - q1)*q3) ) +  1.0);
```

If we were to round to the nearest integer rather than truncate, i.e. 'nint' has the following definition:

```
long int nint (double x){
        if (x >= 0)
                return (long int)(x + 0.5);
        else
                return (long int)(x - 0.5);
}
```

Then 13080 becomes the best tour length. The best tour (city sequence) remains as that of Table 2.17. Another scenario would be if we were to use this new rounding 'nint' function in both converting to geographical latitude and longitude and for distance calculations. Doing so, we end up with a best tour length of 13146 KM and a best tour depicted in Figure 2.14. We notice here:

- Whether 13064 KM or 13080 KM or 13146 KM is the best tour length depends on what the function 'nint' does
- The best tour (city sequence) is affected by which implementation of 'nint' is used

In conclusion: which tour is the $10,000 winning solution, that of Figure 2.13 or is it that of Figure 2.14? To answer that we need to know:

- The proper co-ordinates used for latitude and longitude (could be different than Table 2.16)
- The method used for calculating the GEO distance
- How 'nint' behaves?
- The optimal tour to compare to (since someone already won in 1962!)

Figure 2.13 33-city Problem: Best Tour - Truncating

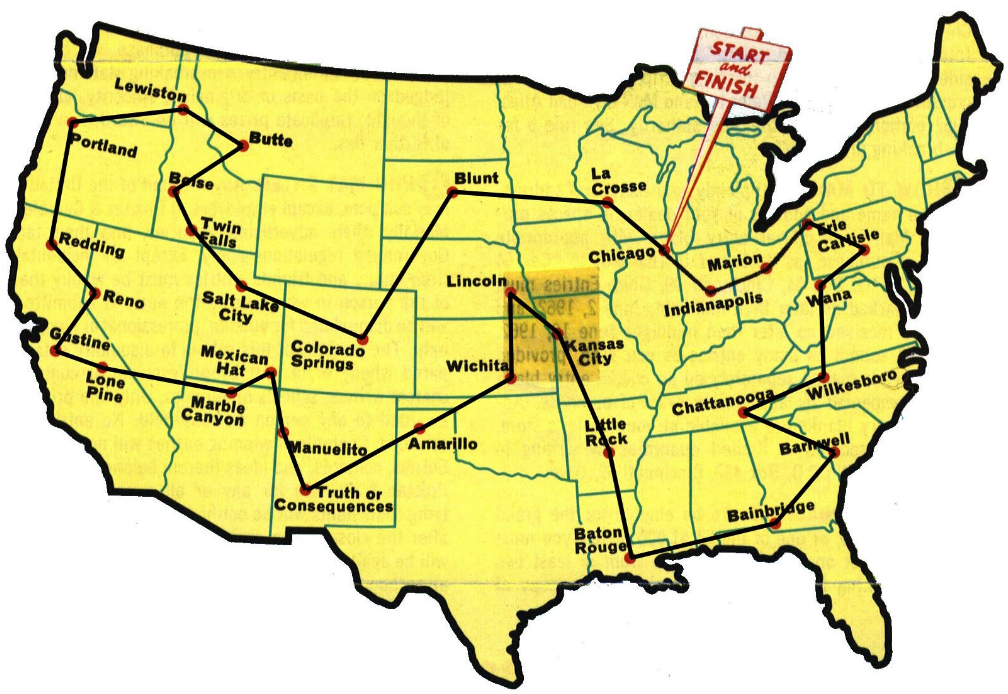


Figure 2.14 33-city Problem: Best Tour - Rounding

CHAPTER 3

COMPUTATIONAL RESULTS

Every heuristic can be evaluated in terms of two key parameters: its running time and the quality of the tours that it produces. In measuring the quality of tours we compare to optimal tour lengths given in TSPLIB [13]. Running time comparisons are a bit more difficult, as rules of thumb for relating times between machines are far from exact and are highly dependent on the actual code, compiler and operating system used. In this chapter, the problem instances chosen in our tests are presented. We report the results on this set of instances on a fixed computer.

3.1 Problem Instances

Throughout this chapter, we use a set of sample problems from TSPLIB. These instances are compiled from different sources, have different types and range in size from 14 to 225 cities for STSP and from 17 to 101 cities for the ATSP. This chosen problem set consists of 52 STSP and 16 ATSP instances, all of which have optimal solutions that are known (and proven!). According to [15] none of the TSPLIB symmetrical TSP instances are contrived to be hard, with the exception of ts225, and none are contrived to be easy. A listing of the STSP testbed instances is given in Table 3.1. ATSP testbed instances are listed in Table 3.2. The problem types examined can be classified according to their edge weights:

- GEO: weights are geographical distances
- EUC 2D: weights are Euclidean distances in 2-D

- ATT: special distance function for problem att48
- FULL MATRIX: weights are given by a full matrix
- LOWER DIAG ROW: lower triangular matrix (row-wise including diagonal entries)
- UPPER ROW: upper triangular matrix (row-wise without diagonal entries)
- UPPER DIAG ROW: upper triangular matrix (row-wise including diagonal entries)

Instances whose edge weights are given in matrix form do not necessarily obey the triangle inequality; the shortest distance between two cities is the shortest path connecting them ($c_{ik} \leq c_{ij} + c_{jk}$ for all i, j and k).

3.2 The Distance Functions

For the various choices of edge weight types (non-matrix types), the following describes the computations of the respective distances as prescribed in TSPLIB [13]. "In each case a *C*-implementation for computing the distances from the input coordinates is given. All computations involving floating-point numbers are carried out in double precision arithmetic. The integers are assumed to be represented in 32-bit words. Since distances are required to be integral, we round to the nearest integer." [13]. This is the *C* rounding function 'nint' used:

```
long int nint (double x){
               if (x >= 0)
                      return (long int)(x + 0.5);
               else
                      return (long int)(x - 0.5);
}
```

Table 3.1 STSP Test Problem Instances

	Name	Edge Weight Type	Comment
1	Att48	ATT	48 capitals of the US (Padberg/Rinaldi)
2	Bayg29	UPPER_ROW	29 Cities in Bavaria, geographical distances (Groetschel,Juenger,Reinelt)
3	Bays29	FULL_MATRIX	29 cities in Bavaria, street distances (Groetschel,Juenger,Reinelt)
4	berlin52	EUC_2D	52 locations in Berlin (Groetschel)
5	bier127	EUC_2D	127 Biergaerten in Augsburg (Juenger/Reinelt)
6	brazil58	UPPER_ROW	58 cities in Brazil (Ferreira)
7	Brg180	UPPER_ROW	Bridge tournament problem (Rinaldi)
8	burma14	GEO	14-Staedte in Burma (Zaw Win)
9	Ch130	EUC_2D	130 city problem (Churritz)
10	Ch150	EUC_2D	150 city Problem (churritz)
11	d198	EUC_2D	Drilling problem (Reinelt)
12	dantzig42	LOWER_DIAG_ROW	42 cities (Dantzig)
13	Eil101	EUC_2D	101-city problem (Christofides/Eilon)
14	Eil51	EUC_2D	51-city problem (Christofides/Eilon)
15	Eil76	EUC_2D	76-city problem (Christofides/Eilon)
16	Fri26	LOWER_DIAG_ROW	26 Staedte (Fricker)
17	gr120	LOWER_DIAG_ROW	120 cities in Germany (Groetschel)
18	gr137	GEO	America-Subproblem of 666-city TSP (Groetschel)
19	gr17	LOWER_DIAG_ROW	17-city problem (Groetschel)
20	gr202	GEO	Europe-Subproblem of 666-city TSP (Groetschel)
21	gr21	LOWER_DIAG_ROW	21-city problem (Groetschel)
22	gr24	LOWER_DIAG_ROW	24-city problem (Groetschel)
23	gr48	LOWER_DIAG_ROW	48-city problem (Groetschel)
24	gr96	GEO	Africa-Subproblem of 666-city TSP (Groetschel)
25	hk48	LOWER_DIAG_ROW	48-city problem (Held/Karp)
26	KroA100	EUC_2D	100-city problem A (Krolak/Felts/Nelson)
27	KroA150	EUC_2D	150-city problem A (Krolak/Felts/Nelson)
28	KroA200	EUC_2D	200-city problem A (Krolak/Felts/Nelson)
29	KroB100	EUC_2D	100-city problem B (Krolak/Felts/Nelson)
30	KroB150	EUC_2D	150-city problem B (Krolak/Felts/Nelson)
31	KroB200	EUC_2D	200-city problem B (Krolak/Felts/Nelson)
32	KroC100	EUC_2D	100-city problem C (Krolak/Felts/Nelson)
33	KroD100	EUC_2D	100-city problem D (Krolak/Felts/Nelson)
34	KroE100	EUC_2D	100-city problem E (Krolak/Felts/Nelson)
35	lin105	EUC_2D	105-city problem (Subproblem of lin318)
36	pr107	EUC_2D	107-city problem (Padberg/Rinaldi)
37	pr124	EUC_2D	124-city problem (Padberg/Rinaldi)
38	pr136	EUC_2D	136-city problem (Padberg/Rinaldi)
39	pr144	EUC_2D	144-city problem (Padberg/Rinaldi)
40	pr152	EUC_2D	152-city problem (Padberg/Rinaldi)
41	pr76	EUC_2D	76-city problem (Padberg/Rinaldi)
42	rat195	EUC_2D	Rattled grid (Pulleyblank)
43	rat99	EUC_2D	Rattled grid (Pulleyblank)
44	rd100	EUC_2D	100-city random TSP (Reinelt)
45	si175	UPPER_DIAG_ROW	(M.~Hofmeister)
46	st70	EUC_2D	70-city problem (Smith/Thompson)
47	swiss42	FULL_MATRIX	42 Staedte Schweiz (Fricker)
48	ts225	EUC_2D	225-city problem (Juenger,Raecke,Tschoecke)
49	tsp225	EUC_2D	A TSP problem (Reinelt)
50	U159	EUC_2D	Drilling problem (Reinelt)
51	ulysses16	GEO	Odyssey of Ulysses (Groetschel/Padberg)
52	ulysses22	GEO	Odyssey of Ulysses (Groetschel/Padberg)

Table 3.2 ATSP Test Problem Instances

	Name	# cities	Edge Weight Type
1	br17	17	Full Matrix
2	ft53	53	Full Matrix
3	ft70	70	Full Matrix
4	ftv33	34	Full Matrix
5	ftv35	36	Full Matrix
6	ftv38	39	Full Matrix
7	ftv44	45	Full Matrix
8	ftv47	48	Full Matrix
9	ftv55	56	Full Matrix
10	ftv64	65	Full Matrix
11	ftv70	71	Full Matrix
12	ftv90	91	Full Matrix
13	ftv100	101	Full Matrix
14	kro124	100	Full Matrix
15	p43	43	Full Matrix
16	ry48p	48	Full Matrix

3.2.1 Euclidean Distance

For edge weight type EUC 2D, floating-point coordinates are specified for each node (city). Let x[i], y[i] be the coordinates of node *i*. In the 2-dimensional case the distance between two points *i* and *j* is computed as follows:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
dij = nint( sqrt( xd*xd + yd*yd) );
where 'sqrt' is the C square root function.
```

3.2.2 Geographical Distance

If the traveling salesman problem is a geographical problem, then the nodes correspond to points on the earth and the distance between two points is their distance on the idealized sphere with radius 6378.388 kilometers. The node coordinates give the geographical latitude and longitude of the corresponding point on the earth. Latitude and longitude are given in the form DDD.MM where DDD are the degrees and MM the minutes. Positive latitude is assumed to be 'North', negative latitude means 'South'. Positive longitude means 'East', negative longitude is assumed to be 'West'. Let x[i] and y[i] be coordinates for city *i* in the above format. First, the input is converted to geographical latitude and longitude given in radians as follows:

```
PI = 3.141592;
deg = nint( x[i] );
min = x[i] - deg;
latitude[i] = PI * (deg + 5.0 * min / 3.0 ) / 180.0;
deg = nint( y[i] );
min = y[i] - deg;
longitude[i] = PI * (deg + 5.0 * min / 3.0 ) / 180.0;
```

The distance between two nodes (*i* and *j*) in kilometers, i.e. their distance on the idealized sphere is then computed as follows:

```
RRR = 6378.388;
q1 = cos( longitude[i] - longitude[j] );
q2 = cos( latitude[i] - latitude[j] );
q3 = cos( latitude[i] + latitude[j] );
dij = nint(  RRR * acos( 0.5*((1.0 + q1)*q2 - (1.0 - q1)*q3) ) +  1.0);

The function 'acos' is the inverse of the cosine function.
```

Although TSPLIB indicates that the function 'nint' (round to the nearest integer) is used in the above edge weight calculations of GEO instances. After checking the *optimal tours* for the instances of type GEO provided in TSPLIB, specifically: ulysses16, ulysses22, gr96, gr137, gr202, and calculating their corresponding edge weights using rounding to nearest integer. We found that adding these edges did not give the *optimal tour lengths* provided in TSPLIB. Therefore, for us to keep in accordance with the optimal tour lengths provided in TSPLIB, when dealing with GEO edge calculations, we found that 'nint' corresponds to *truncating* the fractional part of a real number. The heuristic uses type casting when calculating edge weights of type GEO (*C* implementation for truncating the fractional part of a real number).

3.2.3 Pseudo-Euclidean Distance

The edge weight type ATT corresponds to a special 'Pseudo-Euclidean' distance function. Let x[i] and y[i] be the coordinates of node *i*. The distance between two points *i* and *j* is computed as follows:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
rij = sqrt( (xd*xd + yd*yd) / 10.0 );
tij = nint( rij );
if (tij<rij) dij = tij + 1;
else dij = tij;
```

3.3 Experimental Settings

In this section, we report on the results obtained when running the heuristic on all 68-problem instances. The implementation has the following assumptions:

- Preparation Phase: each tour is constructed from searching $n\times(n-1)/2$ edges (complete graph) and each of the n cities serves as a starting point for a tour
- Enumeration Phase: for every node in the enumeration tree, each of the n cities serves as a starting point for a tour

Concerning CPU times that are either explicitly given or presented in a graphical display, the following remarks apply:

- All CPU times are given in seconds on a PIII 750 MHz with 128 MB RAM running Microsoft Windows 98 SE
- All software has been written in *C* and was compiled using Borland C++ 4.5

3.3.1 Solution Quality

Solution quality helps in providing information that will help answer the questions: how well will the algorithm perform? (how near to optimal will the tours be?), and is measured by the percent excess above the optimal tour length provided in TSPLIB. The following formula calculates excess:

$$Excess = \frac{\text{Tour length} - \text{Optimal tour length}}{\text{Optimal tour length}} \times 100$$

Table 3.3 and Table 3.4 show this measure for each of our testbed instances. The solution trail for each of these instances can be found in Table 5.1 and Table 5.2.

Table 3.3 STSP Solution Quality Measure

	Name	**Excess (%)**
1	att48	0.0
2	Bayg29	0.0
3	Bays29	0.0
4	berlin52	0.0
5	Bier127	0.0
6	brazil58	0.0
7	brg180	0.0
8	burma14	0.0
9	ch130	0.0
10	ch150	0.0
11	d198	0.019
12	dantzig42	0.0
13	Eil101	0.0
14	Eil51	0.0
15	Eil76	0.0
16	Fri26	0.0
17	gr120	0.0
18	gr137	0.0
19	gr17	0.0
20	gr202	0.0
21	gr21	0.0
22	gr24	0.0
23	gr48	0.0
24	gr96	0.0
25	hk48	0.0
26	KroA100	0.0

	Name	**Excess (%)**
27	KroA150	0.0
28	KroA200	0.0
29	KroB100	0.0
30	KroB150	0.0
31	KroB200	0.146
32	KroC100	0.0
33	KroD100	0.0
34	KroE100	0.0
35	Lin105	0.0
36	pr107	0.0
37	pr124	0.0
38	pr136	0.0
39	pr144	0.0
40	pr152	0.0
41	pr76	0.0
42	Rat195	0.129
43	Rat99	0.0
44	rd100	0.0
45	si175	0.0
46	st70	0.0
47	swiss42	0.0
48	ts225	0.0
49	tsp225	0.0
50	U159	0.0
51	ulysses16	0.0
52	ulysses22	0.0

Table 3.4 ATSP Solution Quality Measure

	Name	Excess (%)
1	br17	0.0
2	ft53	0.0
3	Ft70	0.0
4	ftv33	0.0
5	ftv35	0.0
6	ftv38	0.0
7	ftv44	0.0
8	ftv47	0.0
9	ftv55	0.0
10	ftv64	0.0
11	ftv70	0.0
12	ftv90	0.0
13	ftv100	0.0
14	kro124	0.0
15	p43	0.0
16	Ry48p	0.0

Three instances did not give optimal solutions in the STSP case. Further analysis of these 3 instances is postponed until Chapter 4. Graphical displays of optimal solutions for STSP instances, that can be graphed, can be found in Appendix B.

3.3.2 Performance Graphs

The following graphs illustrate CPU times for the different phases of our current implementation. Table 5.3 and Table 5.4 list the CPU times for STSP and ATSP instances, respectively. Starting with the STSP instances, Figure 3.1 and Figure 3.2 show similarity since each works on a SDM (tour construction) and a SEL (first run) of the same size $n\times(n-1)$. Comparing Figure 3.1 to Figure 3.2, we notice that it takes less time for an instance to complete the first run since updating the Frequency Matrix is only performed for anchored tours.

The time needed to complete the STSP preparation phase is shown in Figure 3.3 where CPU times from the previous two parts (tour construction and first run steps) are added. Despite the fact that we are working with instances of different types, this graph shows a polynomial relationship between time and n. Since for each of the 2 steps of the preparation phase we are constructing $n \times n \times (n-1)/2$ tours (each step is of order $O(n^3)$), time complexity for the preparation phase is therefore $O(n^3)$.

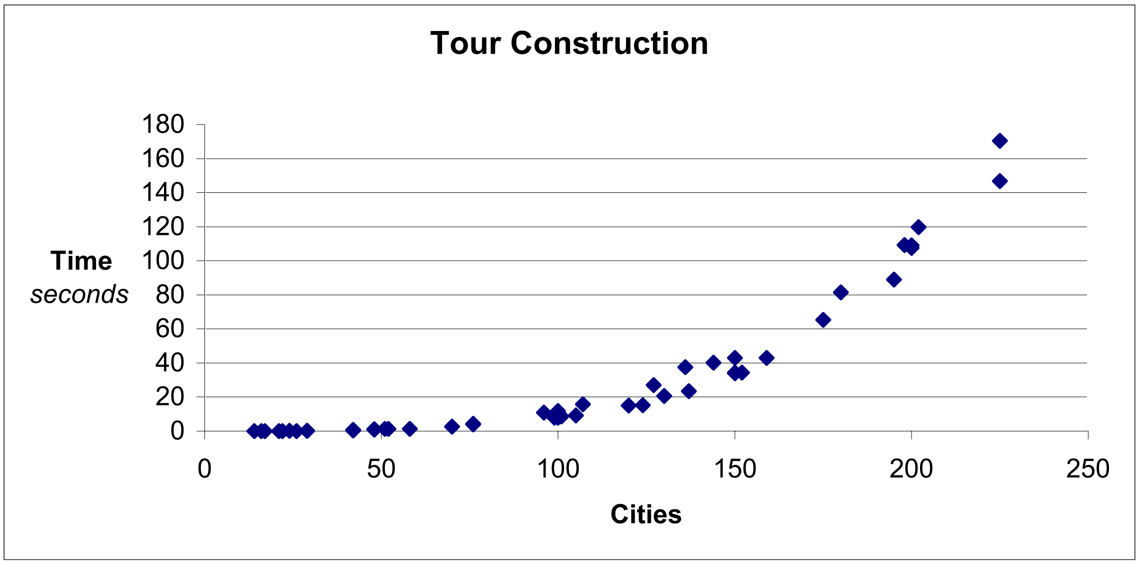


Figure 3.1 STSP Performance: Tour Construction Step

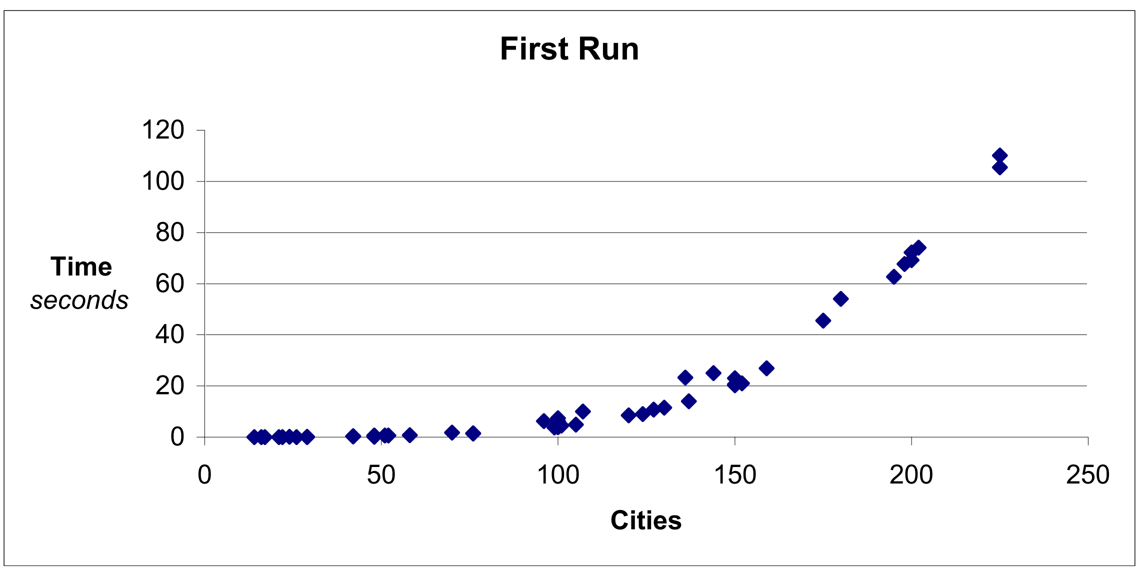


Figure 3.2 STSP Performance: First Run Step

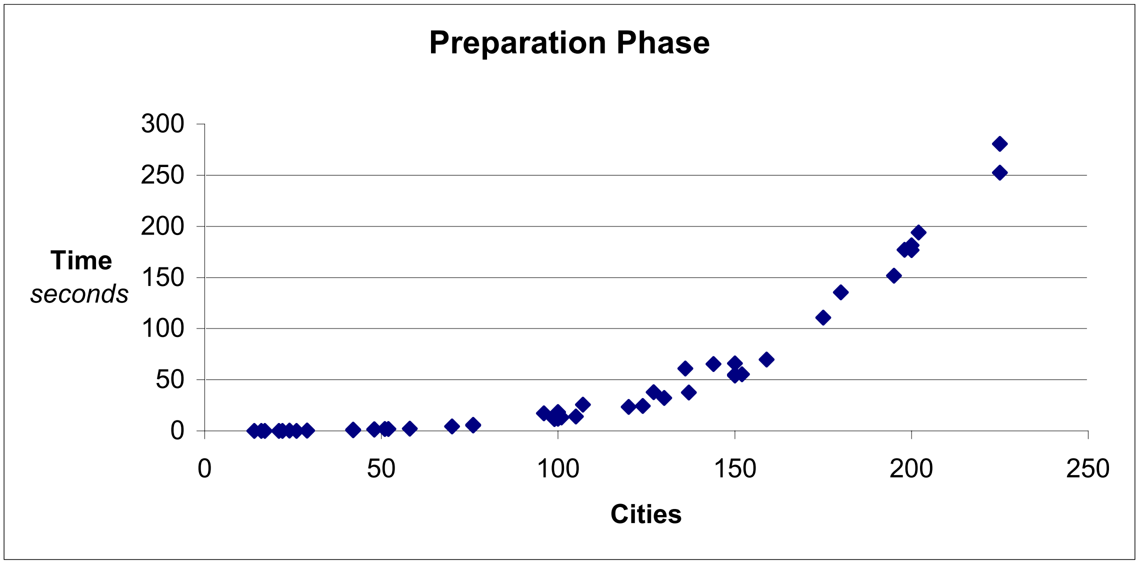


Figure 3.3 STSP Performance: Preparation Phase

The time it takes each instance to complete the enumeration phase for the first time is shown in Figure 3.4. The first time through the enumeration phase is called *first pass*. The enumeration procedure is identical for all instances, for each pass through the enumeration phase we enumerate a fixed number of times (Figure 2.7: for STSP 9×11×3×2 = 594 enumeration paths, each having 4 nodes) and select the pool (that resulted from a node in a path) that gives best pool attributes as 'best' for that pass as was shown in Figure 2.10 for ulysses16 (attributes of this best pool serve as *input* for the next pass). The number of entries in the SEL of which each node in the enumeration phase works on differs, worst case being $n \times (n-1)$. Therefore, for each of the 4 nodes of every path (for STSP = 594 paths), as a worst case, we construct $n \times n \times (n-1)/2$ tours. Hence, the worst case time complexity for every pass through the enumeration phase is also $O(n^3)$. Looking at Figure 3.4, we notice that performance is better than the projected worst case

O(n^3) since the number of entries in any SEL, which serves as a candidate set for constructing tours in a node, is much less than the worst case $n\times(n-1)$.

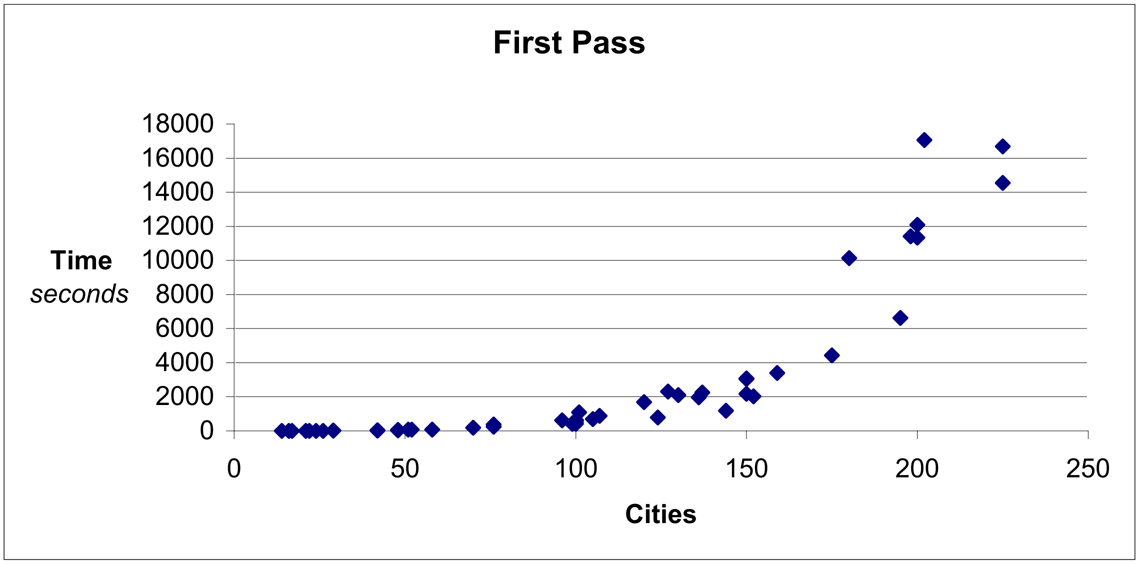


Figure 3.4 STSP Performance: First Pass

Figure 3.5 shows the total time needed to solve each of the STSP testbed instances. The number of passes needed, for the heuristic to halt, varied from one instance to another as shown in Table 5.1 and illustrated in Figure 3.6. The longest being 7, and shortest 2. Assuming that none of the instances needs more than 10 passes to satisfy the termination criterion, then it is safe to conjecture that the heuristic has a worst case running time complexity of O(n^3). Since for each pass it's O(n^3) and we have a fixed worst case number of passes = 10 and complexity for the preparation phase was also O(n^3). It is worth mentioning that all passes after the first require less time to run, since the number of entries in the SEL of a best pool decreases as we progress through passes.

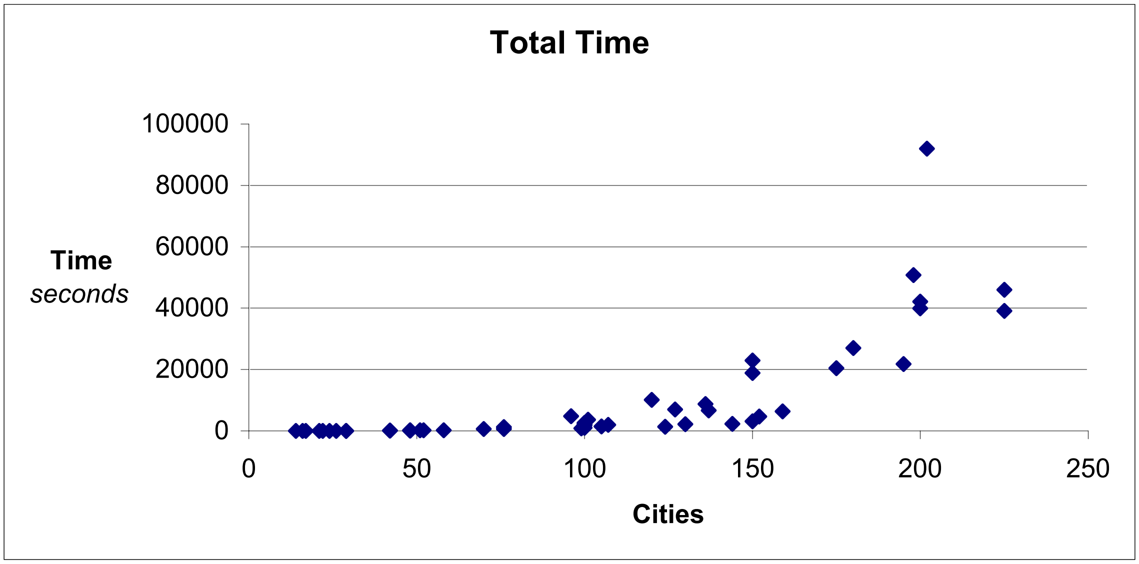


Figure 3.5 STSP Performance: Total Time

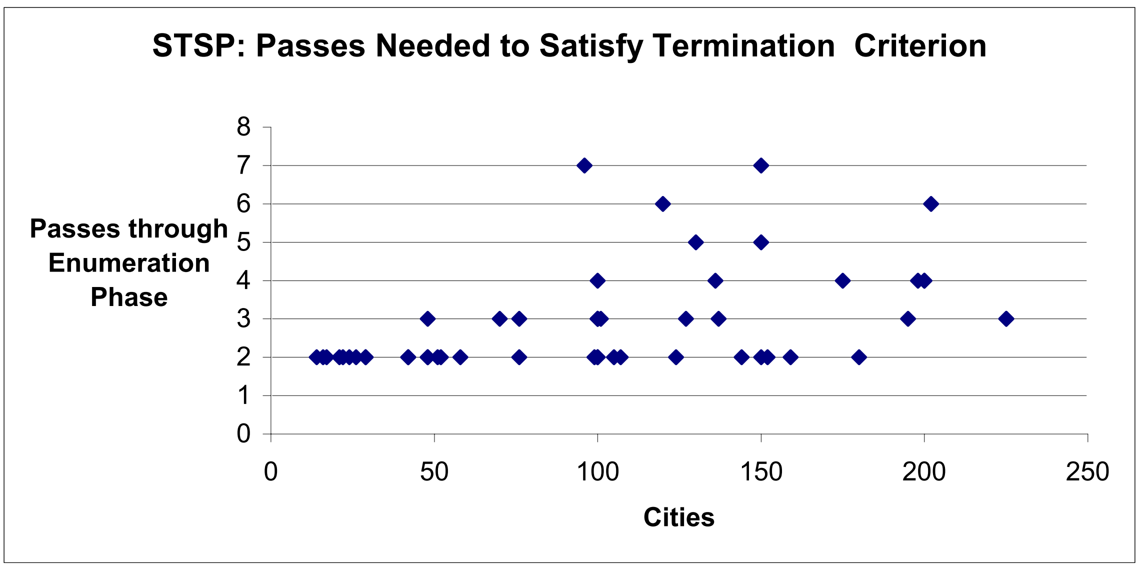


Figure 3.6 STSP Performance: Number of Passes Needed through Enumeration Phase

For the ATSP instances, the heuristic needs more time for every pass through the enumeration phase, since we are dealing with a tree that has more nodes (for ATSP:

11×12×4×2 = 1056 enumeration paths, each having 4 nodes). Figures 3.7 through Figure 3.9 display ATSP times for the preparation phase, first pass and total time, respectively.

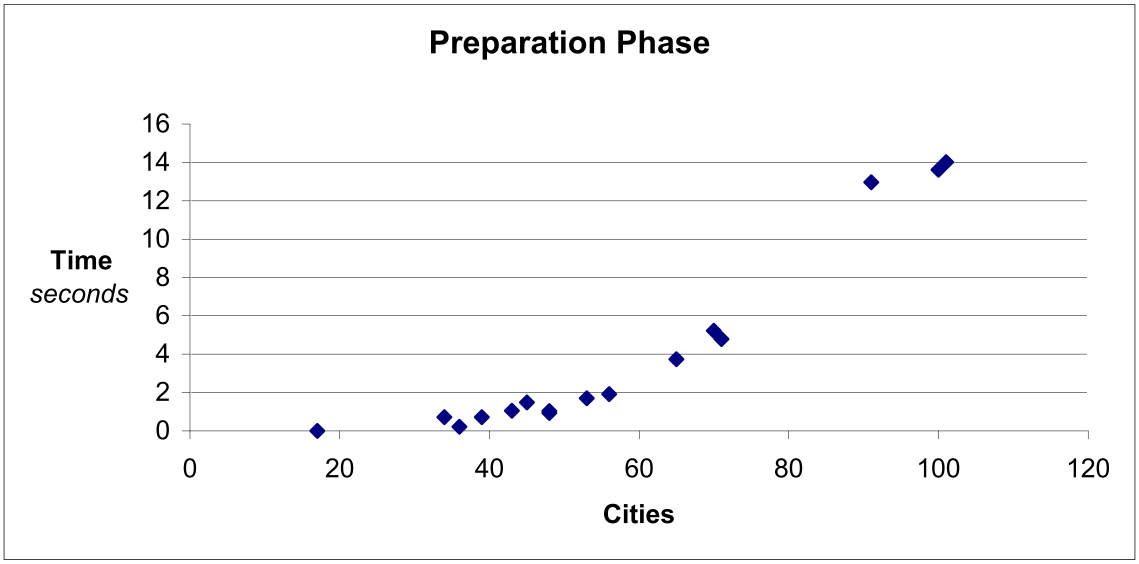


Figure 3.7 ATSP Performance: Preparation Phase

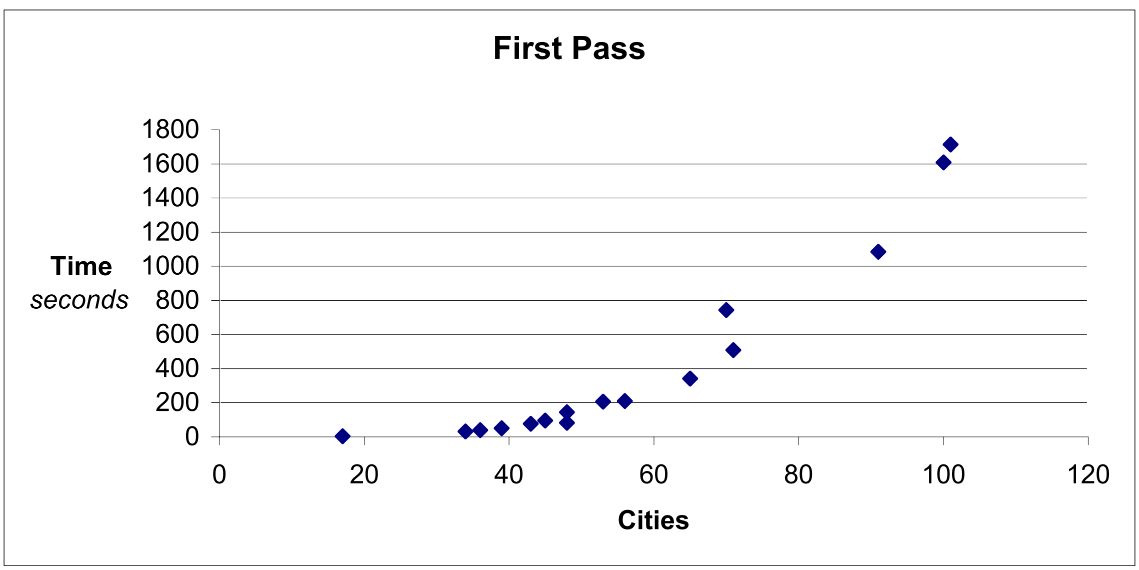


Figure 3.8 ATSP Performance: First Pass

The number of passes needed, for the heuristic to halt, varied from one ATSP instance to another as shown in Table 5.2 and illustrated in Figure 3.10.

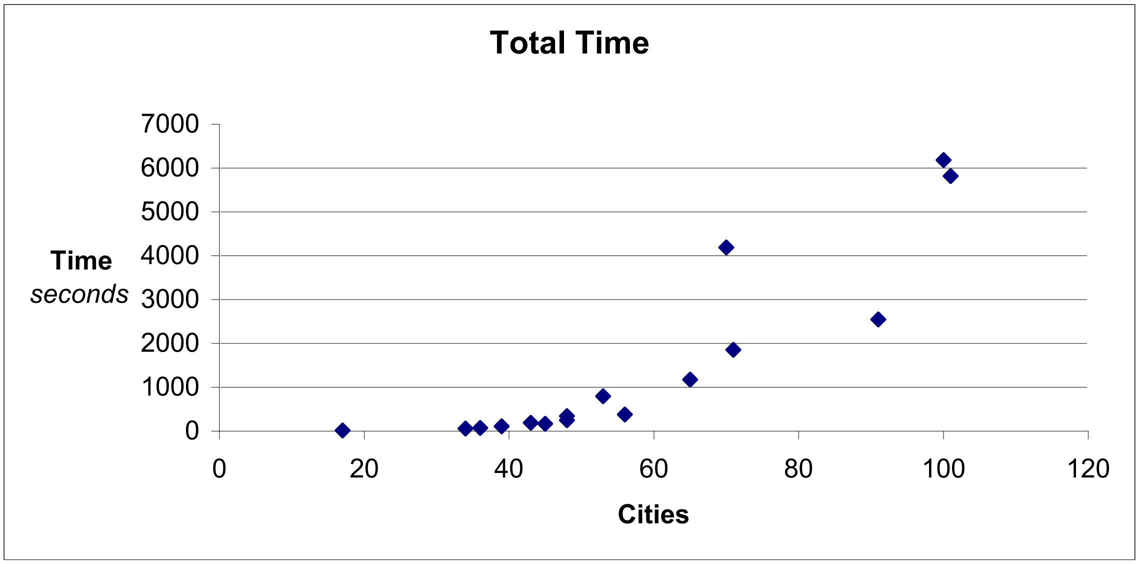


Figure 3.9 ATSP Performance: Total Time

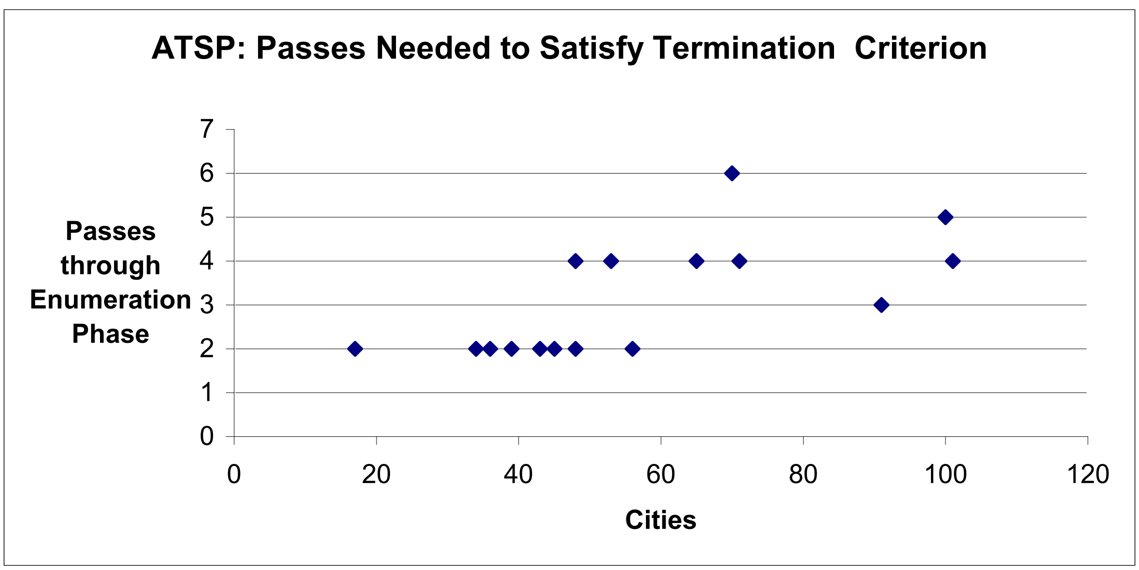


Figure 3.10 ATSP Performance: Number of Passes Needed through Enumeration Phase

ATSP time complexity analysis follows the same line of reasoning presented for the STSP case. Hence, it is safe to conjecture that for the ATSP, the heuristic has a worst case running time complexity of $O(n^3)$.

In the next chapter, we discuss several alterations to the current implementation that improve on these computational results, both in time and quality.

CHAPTER 4

CONCLUSIONS AND FUTURE RESEARCH

In the implementation suggested in Chapter 2, solution quality for all instance types was the main concern and it was clearly at the expense of running time as the computational results of Chapter 3 indicate. In this chapter, we discuss prospects of future research on this subject and suggest alterations to some of the techniques adopted that would enhance the time factor considerably.

4.1 Suggested Enhancements

The following sections suggest alterations to the current implementation that would enhance the computational results presented in Chapter 3. Some of these alterations affect solution quality, while others considerably decrease the time needed for the heuristic to run.

4.1.1 Tour Construction Algorithms & Candidate Sets

In the construction part of the preparation phase, we used the Nearest Neighbor algorithm (NN) to construct our tours. Other construction algorithms could be adopted, and we continue to anchor edges and record their frequency, as we construct tours. Even alterations to the neighborhood that NN searches when constructing tours could be examined. It is intuitively clear, that most of the possible connections would never occur because they are too long. It is therefore a reasonable idea to restrict attention to 'promising' edges. Instead of searching $n \times (n-1)/2$ possible connections (a complete graph, i.e. the Sorted Distance Matrix), we could set up candidate sets of promising edges that restrict and direct the search. There are numerous studies on this topic [1, 2]; most appealing are those that use 1-trees to introduce a measure of nearness as described in [8-10].

4.1.2 Reordering Equal Entries

In this section, we elaborate on the current implementation and why it failed to provide optimal solutions for three of the STSP testbed instances. When examining the Sorted Distance Matrices (SDM) of our testbed instances, we notice consecutive entries in these matrices that are of equal magnitude. To describe this phenomenon and the effect it has on optimal solutions, we examine the SDM of ulysses16. The same reasoning could be applied to equal entries appearing in Sorted Frequency Matrices (SFM) and their effect on the corresponding Sorted Edge List representations (SEL).

The Sorted Distance Matrix of ulysses16 that was first presented in Table 2.2 is shown here in Table 4.1, highlighting its equal distance entries. For this instance only 2 entries have equal consecutive entries. In other instances of our testbed, the percentage of equal distance entries to the total number of entries in the SDM could be much higher. This might not be the case for real practical problems.

This equal distance phenomenon was found to have an effect while constructing tours using NN, in the construction step of the preparation phase. Through experimenting with these equal distance entries (their city representations), it was found that changing their (city representation) relative positions sometimes had an effect on whether or not the heuristic produced optimal results. For others it had the effect of speeding up the heuristic; the heuristic would satisfy the termination criterion in a fewer number of passes (for some no effect whatsoever).

The way the current implementation deals with equal distance entries in the SDM can be described by examining the SDM city representation of ulysses16 that is shown in Table 4.2. Here city1 has an equal distance to each of cities 14 and 12. What we are looking for, is the answer to: which city should be placed before the other in the SDM city representation? Trying to answer this, we construct a tour starting at city1 then city14 and finish it off with NN. We construct another tour that starts at city1 then city12 and finish it off with NN. We then compare the lengths of these two tours. The city whose tour was shortest is placed before the other in the SDM. In the case of these two cities, city14 had a shorter tour than city12. In the case of cities 1 and 8 (same distance to city14), city8 gets to be placed before city1. This new order is shown in Table 4.3 and is identical to that of Table 2.3.

In the current implementation, any SDM that had equal entries of 15% or less of its total number of entries, had its equal entries re-sorted. Why 15%? For the SDM of Table 4.3, trying to reorder the equal distance entries again (by constructing tours using NN), we find that there is no need to re-adjust the position of any of the entries, i.e. the SDM converged. For instances that have a higher percentage of equal entries in their SDM this would not be the case. They would never converge! Therefore, reordering the

Table 4.1 ulysses16: Sorted Distance Matrix (highlighting equal entries)

Distance/distance

1	60	150	312	448	479	479	501	509	619	656	726	736	1019	1039	2314
1	126	474	509	532	542	941	958	978	1122	1127	1133	1226	1449	1526	2789
1	126	499	501	536	541	904	913	946	1045	1084	1115	1184	1371	1516	2728
1	271	312	455	474	541	704	720	751	783	919	980	1029	1157	1333	2553
1	401	478	583	600	651	677	855	858	996	1019	1033	1157	1504	1516	1526
1	115	261	271	289	308	379	470	478	687	736	740	980	1184	1226	1581
1	115	177	207	216	288	343	455	583	592	656	667	919	1084	1133	1661
1	60	206	271	454	479	493	532	536	598	667	740	759	996	1066	2320
1	328	455	470	591	650	656	776	858	933	1039	1066	1333	1371	1387	1449
1	288	328	333	379	400	427	610	622	726	759	855	1029	1045	1122	1697
1	1387	1504	1581	1661	1697	1789	1838	1841	1868	2248	2314	2320	2553	2728	2789
1	68	105	177	271	333	336	417	479	493	591	677	751	913	958	1838
1	52	68	216	287	289	400	406	448	454	650	651	704	904	941	1868
1	52	105	207	237	261	427	449	479	479	600	656	720	946	978	1841
1	237	287	308	336	343	401	598	619	622	636	776	783	1115	1127	1789
1	150	206	406	417	449	455	499	542	592	610	636	687	933	1033	2248

Table 4.2 ulysses16: Sorted Distance Matrix in City Format (highlighting equal entries)

(city#/city#)

1	8	16	4	13	14	12	3	2	15	7	10	6	5	9	11
2	3	4	1	8	16	13	12	14	10	15	7	6	9	5	11
3	2	16	1	8	4	13	12	14	10	7	15	6	9	5	11
4	8	1	16	2	3	13	14	12	15	7	6	10	5	9	11
5	15	6	7	14	13	12	10	9	8	1	16	4	11	3	2
6	7	14	12	13	15	10	9	5	16	1	8	4	3	2	11
7	6	12	14	13	10	15	9	5	16	1	8	4	3	2	11
8	1	16	4	13	14	12	2	3	15	7	6	10	5	9	11
9	10	7	6	12	13	14	15	5	16	1	8	4	3	11	2
10	7	9	12	6	13	14	16	15	1	8	5	4	3	2	11
11	9	5	6	7	10	15	12	14	13	16	1	8	4	3	2
12	13	14	7	6	10	15	16	1	8	9	5	4	3	2	11
13	14	12	7	15	6	10	16	1	8	9	5	4	3	2	11
14	13	12	7	15	6	10	16	1	8	5	9	4	3	2	11
15	14	13	6	12	7	5	8	1	10	16	9	4	3	2	11
16	1	8	13	12	14	4	3	2	7	10	15	6	9	5	11

Table 4.3 ulysses16: Sorted Distance Matrix in City Format (re-sorting equal entries)

(city#/city#)

1	8	16	4	13	14	12	3	2	15	7	10	6	5	9	11
2	3	4	1	8	16	13	12	14	10	15	7	6	9	5	11
3	2	16	1	8	4	13	12	14	10	7	15	6	9	5	11
4	8	1	16	2	3	13	14	12	15	7	6	10	5	9	11
5	15	6	7	14	13	12	10	9	8	1	16	4	11	3	2
6	7	14	12	13	15	10	9	5	16	1	8	4	3	2	11
7	6	12	14	13	10	15	9	5	16	1	8	4	3	2	11
8	1	16	4	13	14	12	2	3	15	7	6	10	5	9	11
9	10	7	6	12	13	14	15	5	16	1	8	4	3	11	2
10	7	9	12	6	13	14	16	15	1	8	5	4	3	2	11
11	9	5	6	7	10	15	12	14	13	16	1	8	4	3	2
12	13	14	7	6	10	15	16	1	8	9	5	4	3	2	11
13	14	12	7	15	6	10	16	1	8	9	5	4	3	2	11
14	13	12	7	15	6	10	16	8	1	5	9	4	3	2	11
15	14	13	6	12	7	5	8	1	10	16	9	4	3	2	11
16	1	8	13	12	14	4	3	2	7	10	15	6	9	5	11

equal entries would be of no practical use. Experimenting with our testbed instances, 15% was found to be the cutoff for convergence, and hence for reordering.

This reordering based on NN is clearly not the best resolution to this phenomenon, even though it showed acceptable results. A different reordering scheme was examined. The new reordering technique was tested on the three instances that did not give optimal results in Chapter 3. These three instances had nodes that were specified by coordinates in 2-D space. We sort the equal entry points in the SDM with respect to their x-coordinates and y-coordinates, respectively and in ascending order. Solution quality changes are depicted in Table 4.4, and solution trails are shown in Table 4.5.

Another alternative would be to perform a simple transformation of the original Distance Matrix. An interesting transformation is described in [9, 10] where the length of all edges incident to a node are changed with the same amount, π_i, hence any optimal tour remains optimal (the length of every tour is increased by $2\sum\pi_i$).

Table 4.4 Solution Quality (re-sorting equal entries)

Previous results	
Name	Excess (%)
d198	0.019
KroB200	0.146
rat195	0.129

New results	
Name	Excess (%)
d198	0.0
kroB200	0.0
rat195	0.086

Table 4.5 Solution Trail (re-sorting equal entries)

d198	best is 15880 starting at 33 in loop 5, 3, 1, 1
	best is 15819 starting at 14 in loop 2, 4, 1, 1
	best is 15808 starting at 14 in loop 2, 10, 2, 0
	best is 15790 starting at 152 in loop 7, 11, 3, 2
	best is 15780 starting at 141 in loop 9, 4, 1, 2
	best is 15780 starting at 147 in loop 2, 3, 1, 0
kroB200	best is 29751 starting at 72 in loop 7, 1, 1, 1
	best is 29506 starting at 183 in loop 5, 11, 3, 1
	best is 29437 starting at 181 in loop 6, 1, 2, 2
	best is 29437 starting at 172 in loop 7, 2, 2, 1
rat195	best is 2334 starting at 14 in loop 4, 1, 1, 1
	best is 2325 starting at 14 in loop 4, 3, 2, 1
	best is 2325 starting at 29 in loop 6, 8, 1, 0

As suggested in the previous section though, a better approach when constructing tours in the construction step, would be to utilize a small candidate set that is based on a measure of nearness that better describes a given edge of being a member of an optimal tour, rather than using the SDM as an initial candidate set. Using an initial candidate set, other than SDM in the construction step, would produce a pool of solutions that has better attributes and fewer entries in its resulting SEL. The SEL that results from the construction step, is of great importance since it serves as the candidate set for constructing tours in the *first run* step. With a smaller candidate set (SEL from construction step) to work on, the first run would run faster and its resulting SEL would have fewer entries. Therefore, the total running time for the heuristic decreases dramatically (SEL resulting from the first run is *input* for the enumeration phase).

Some of the methods that introduce new measures of nearness (suggested to be used in the construction step) perform better (produce better candidate sets) when working on a transformed Distance Matrix [8]. The initial candidate set, could also be used to re-sort equal entries in the SFM. This would affect the corresponding SELs that results. In the current implementation, no criterion was used to re-sort equal entries in the SFM.

A note worth mentioning concerning the current implementation is that bubble sort was the sorting algorithm used whenever we needed to sort entries (whenever we create a SEL). Bubble sort is clearly not the best sorting algorithm to use. A host of faster sorting algorithms are described in [16].

4.1.3 Nodes in the Enumeration Tree

Examining Table 5.3 and Table 5.4, we note that most of the heuristic's running time is spent in the enumeration phase. Therefore, decreasing the time spent in this phase would decrease the total running time of the heuristic. The previous sections discussed the effect of introducing a smaller initial candidate set on the running time of the enumeration phase. Other alterations to the current implementation that would also decrease the running time of this phase include:

- Decreasing the number of possible enumeration paths in the first pass of the enumeration phase: this can be accomplished by decreasing the number of nodes present in the enumeration tree of the first pass. The time needed to complete the first pass is the longest, since the first pass's search neighborhood (for all nodes in the first column of Figure 2.7) is dictated by the output of the preparation phase, whereas all subsequent passes have their search neighborhoods assigned by prior passes (less entries in the SEL as we progress from one pass to the next). In the current implementation, we decreased the number of nodes in the first column of Figure 2.7 for the first pass to 7 (STSP case). Therefore, the number of possible enumeration paths for the first pass decreased from $9\times11\times3\times2 = 594$ to $7\times11\times3\times2 = 462$
- Decreasing the number of possible enumeration paths in the enumeration phase for subsequent passes: the same reasoning regarding reducing the time needed to complete the first pass applies to all passes

- Introducing new functions: in the current implementation, we experimented with 15 functions (that appear as nodes in an enumeration tree), arranged in 4 columns as depicted in Figure 2.7. As can be seen from the results of Table 5.1 and Table 5.2, some functions (nodes) were used more extensively than others. Therefore, introducing new functions that outperform the ones currently used is something worth considering. In this study, we experimented with 2 main types of functions and created variants from them. Introducing new types of functions, along with introducing new ways to construct variants from them is something worth considering
- Introducing ATSP specific functions: in this study, the same 15 functions were used for both the STSP and ATSP cases. This need not be the case if we are interested in enhancing running time

4.1.4 Randomness

Introducing randomness definitely enhances the time factor. If for every function and when constructing tours, instead of re-staring at each city (n starting points), we re-started a fixed number of times with cities that were randomly selected. This would reduce the time complexity to O(n^2) for all functions. Most heuristics adopt this technique (random restarts was one of the first methods proposed to escape local minima) and specify the average of excess in their resulting tours over the total number of restarts.

4.1.5 Termination Criterion & Running Time

Different heuristics adopt different termination criteria. While working on the benchmark instances of TSPLIB, there are heuristics that specify their termination criterion, as that, the algorithm should stop if it reaches the optimum specified in TSPLIB. In recording time, there are heuristics that do not include the time needed for constructing initial tours (tour construction part) in their *running time* measure. In our current implementation, time was recorded from the beginning of the input phase until the termination criterion was satisfied; when two consecutive passes through the enumeration phase have identical best tour lengths or if we had looped the enumeration phase10 times, whichever comes first. Considering other termination criteria would save time, such as comparing the heuristic's best tour length to a computed lower bound of the optimal tour length [6]. Using this suggested new termination criterion, in contrast to the one adopted in the current implementation, would reduce running time by the time it takes the heuristic to complete an extra pass through the enumeration phase.

4.2 Summary

For the TSP, a tour of size n is constructed from $n \times (n-1)/2$ possible connections. Using FANN we tried to reduce the number of these possible connections relying on how frequently connections appeared in a pool of solutions, i.e. FANN becomes our new measure of nearness, and its edge representation acts as a candidate set for constructing future tours. FANN played a major role in determining how future solutions were constructed (restricting and directing the search). New search neighborhoods are the product of past successful solutions (pools).

Experimental evaluation of the heuristic, on benchmark instances of the symmetrical and asymmetrical TSP, compared very favorably to famous general and specialized heuristic algorithms. Several possible modifications to the current implementation, that decrease the running time of the heuristic, were suggested earlier in this chapter. This encourages future research towards adopting FANN, as a measure of nearness, in attempting to solve larger instances of the TSP and other NP hard problems.

REFERENCES

[1] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D. B. Shmoys, *The Traveling salesman problem: a guided tour of combinatorial optimization*. Chichester [West Sussex] ; New York: Wiley, 1985.

[2] G. Reinelt, *The traveling salesman: computational solutions for TSP applications*. Berlin ; New York: Springer-Verlag, 1994.

[3] R. Jonker and T. Volgenant, "Transforming Asymmetric into Symmetric Traveling Salesman Problems," *Operations Research Letters*, vol. 2, No. 4, 1983.

[4] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. San Francisco: W. H. Freeman, 1979.

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

[6] E. H. L. Aarts and J. K. Lenstra, *Local search in combinatorial optimization*. Chichester [England] ; New York: Wiley, 1997.

[7] S. Lin and W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Operations Research*, vol. 21, pp. 498-516, 1973.

[8] K. Helsgaun, "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic," *European Journal of Operational Research*, vol. 126, pp. 106-130, 2000.

[9] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, pp. 1138-1162, 1970.

[10] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees: Part II," *Mathematical Programming*, vol. 1, pp. 6-25, 1971.

[11] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*, 3rd rev. and extended ed. Berlin ; New York: Springer-Verlag, 1996.

[12] J. K. Lenstra and A. H. G. R. Kan, "Some simple applications of the traveling salesman problem," *Oper. Res. Quart.*, vol. 26, pp. 717-733, 1975.

[13] G. Reinelt. (1995, June 15). *TSPLIB*, [Online]. Available: http://softlib.rice.edu/softlib/tsplib/.

[14] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. (2001, February 15). *A Pictorial Survey of the History of the Traveling Salesman Problem*, [Online]. Available: http://www.keck.caam.rice.edu/tsp/history.html.

[15] M. Holtsclaw. (1996). *A Farmer's Daughter in Our Midst - An Interview with Vasek Chvátal*, [Online]. Available: http://www.cs.rutgers.edu/~mcgrew/Explorer/1.2/#Farmers.

[16] S. S. Skiena, *The algorithm design manual*. Santa Clara, Calif.: TELOS--the Electronic Library of Science, 1998.

APPENDIX A

PERFORMANCE TABLES

Table 5.1 STSP Performance: Solution Trail

	Name	**Initial Passes**	**Final Pass**
1	att48	best is 10628 starting at 3 in loop 6, 5, 2, 0	best is 10628 starting at 23 in loop 3, 4, 2, 2
2	bayg29	best is 1610 starting at 2 in loop 5, 6, 1, 0	best is 1610 starting at 5 in loop 7, 3, 1, 0
3	bays29	best is 2020 starting at 2 in loop 2, 8, 0, 0	best is 2020 starting at 2 in loop 7, 8, 0, 0
4	Berlin52	best is 7542 starting at 7 in loop 5, 8, 2, 1	best is 7542 starting at 30 in loop 2, 8, 0, 0
5	bier127	best is 118743 starting at 10 in loop 1, 2, 2, 1 best is 118282 starting at 105 in loop 6, 10, 2, 1	best is 118282 starting at 5 in loop 6, 6, 2, 0
6	Brazil58	best is 25395 starting at 8 in loop 2, 7, 2, 1	best is 25395 starting at 17 in loop 4, 3, 2, 0
7	brg180	best is 1950 starting at 1 in loop 4, 2, 0, 0	best is 1950 starting at 56 in loop 8, 4, 2, 0
8	burma14	best is 3323 starting at 5 in loop 1, 8, 0, 0	best is 3323 starting at 5 in loop 7, 8, 0, 0
9	ch130	best is 6137 starting at 22 in loop 4, 9, 1, 1 best is 6121 starting at 13 in loop 3, 9, 1, 1 best is 6115 starting at 104 in loop 4, 5, 2, 0 best is 6110 starting at 17 in loop 5, 8, 2, 1	best is 6110 starting at 9 in loop 5, 5, 2, 1
10	ch150	best is 6528 starting at 46 in loop 1, 7, 2, 2	best is 6528 starting at 138 in loop 5, 5, 2, 0
11	d198	best is 15898 starting at 34 in loop 2, 1, 3, 1 best is 15817 starting at 126 in loop 2, 3, 2, 1 best is 15783 starting at 123 in loop 2, 10, 2, 1	best is 15783 starting at 129 in loop 2, 0, 0, 0
12	dantzig42	best is 699 starting at 39 in loop 5, 3, 1, 0	best is 699 starting at 28 in loop 5, 3, 2, 0
13	eil101	best is 633 starting at 87 in loop 4, 5, 2, 1 best is 629 starting at 53 in loop 9, 8, 1, 1	best is 629 starting at 14 in loop 5, 2, 2, 0
14	eil51	best is 426 starting at 20 in loop 5, 3, 2, 2	best is 426 starting at 13 in loop 1, 8, 2, 2
15	eil76	best is 538 starting at 62 in loop 2, 6, 2, 0	best is 538 starting at 1 in loop 8, 5, 2, 1
16	fri26	best is 937 starting at 8 in loop 5, 0, 0, 0	best is 937 starting at 1 in loop 7, 8, 0, 0
17	gr120	best is 7053 starting at 16 in loop 5, 5, 3, 1 best is 7013 starting at 1 in loop 5, 1, 1, 2 best is 6999 starting at 92 in loop 6, 4, 1, 1 best is 6975 starting at 89 in loop 4, 3, 1, 1 best is 6942 starting at 28 in loop 1, 2, 2, 1	best is 6942 starting at 32 in loop 4, 6, 1, 0
18	gr137	best is 70314 starting at 91 in loop 3, 1, 1, 1 best is 69853 starting at 2 in loop 8, 2, 1, 1	best is 69853 starting at 67 in loop 6, 3, 2, 2
19	gr17	best is 2085 starting at 1 in loop 4, 8, 0, 0	best is 2085 starting at 1 in loop 5, 8, 0, 0
20	gr202	best is 41225 starting at 130 in loop 3, 1, 3, 1 best is 40472 starting at 121 in loop 8, 5, 1, 1 best is 40378 starting at 41 in loop 4, 2, 1, 1 best is 40269 starting at 173 in loop 1, 10, 1, 2 best is 40160 starting at 128 in loop 3, 2, 1, 0	best is 40160 starting at 1 in loop 5, 6, 2, 2
21	gr21	best is 2707 starting at 11 in loop 4, 8, 0, 0	best is 2707 starting at 4 in loop 4, 8, 0, 0
22	gr24	best is 1272 starting at 6 in loop 5, 7, 2, 2	best is 1272 starting at 6 in loop 4, 10, 2, 2
23	gr48	best is 5057 starting at 18 in loop 1, 8, 1, 2 best is 5046 starting at 1 in loop 1, 10, 2, 2	best is 5046 starting at 1 in loop 2, 0, 0, 0

	Name	Initial Passes	Final Pass
24	gr96	best is 55467 starting at 46 in loop 5, 7, 1, 1 best is 55403 starting at 35 in loop 5, 7, 2, 2 best is 55394 starting at 74 in loop 6, 4, 2, 2 best is 55291 starting at 11 in loop 1, 8, 2, 1 best is 55259 starting at 1 in loop 3, 4, 1, 2 best is 55209 starting at 70 in loop 5, 2, 2, 0	best is 55209 starting at 69 in loop 1, 8, 2, 1
25	hk48	best is 11461 starting at 1 in loop 3, 5, 2, 1	best is 11461 starting at 23 in loop 1, 8, 0, 0
26	KroA100	best is 21282 starting at 5 in loop 2, 2, 2, 2	best is 21282 starting at 55 in loop 8, 3, 2, 2
27	KroA150	best is 26790 starting at 21 in loop 1, 10, 1, 1 best is 26727 starting at 44 in loop 4, 2, 2, 2 best is 26626 starting at 28 in loop 4, 11, 1, 1 best is 26534 starting at 2 in loop 6, 6, 1, 2 best is 26525 starting at 25 in loop 4, 6, 1, 2 best is 26524 starting at 8 in loop 8, 1, 1, 1	best is 26524 starting at 8 in loop 4, 5, 2, 0
28	KroA200	best is 29514 starting at 63 in loop 7, 1, 1, 1 best is 29440 starting at 37 in loop 5, 1, 2, 1 best is 29368 starting at 139 in loop 3, 3, 3, 1	best is 29368 starting at 91 in loop 2, 0, 0, 0
29	KroB100	best is 22179 starting at 4 in loop 3, 1, 1, 1 best is 22141 starting at 55 in loop 1, 4, 2, 0	best is 22141 starting at 43 in loop 1, 4, 2, 0
30	KroB150	best is 26333 starting at 2 in loop 6, 4, 1, 2 best is 26328 starting at 27 in loop 4, 4, 1, 1 best is 26254 starting at 51 in loop 8, 11, 2, 1 best is 26130 starting at 45 in loop 2, 2, 0, 0	best is 26130 starting at 97 in loop 2, 1, 1, 0
31	KroB200	best is 30012 starting at 44 in loop 7, 1, 2, 1 best is 29705 starting at 159 in loop 1, 8, 1, 1 best is 29480 starting at 102 in loop 1, 3, 1, 1	best is 29480 starting at 144 in loop 4, 6, 2, 1
32	KroC100	best is 20749 starting at 50 in loop 2, 2, 2, 2	best is 20749 starting at 24 in loop 1, 7, 2, 0
33	KroD100	best is 21563 starting at 59 in loop 1, 8, 2, 2 best is 21383 starting at 42 in loop 2, 2, 1, 2 best is 21294 starting at 11 in loop 4, 2, 1, 0	best is 21294 starting at 73 in loop 5, 5, 2, 1
34	KroE100	best is 22127 starting at 66 in loop 4, 1, 1, 1 best is 22068 starting at 4 in loop 2, 6, 1, 1	best is 22068 starting at 9 in loop 4, 2, 0, 0
35	lin105	best is 14379 starting at 18 in loop 5, 2, 3, 1	best is 14379 starting at 15 in loop 3, 10, 2, 0
36	pr107	best is 44303 starting at 7 in loop 5, 8, 3, 1	best is 44303 starting at 2 in loop 2, 7, 2, 0
37	pr124	best is 59030 starting at 19 in loop 1, 8, 1, 1	best is 59030 starting at 17 in loop 3, 8, 1, 1
38	pr136	best is 98955 starting at 109 in loop 4, 4, 1, 1 best is 96916 starting at 9 in loop 5, 11, 3, 1 best is 96772 starting at 5 in loop 2, 7, 3, 1	best is 96772 starting at 2 in loop 2, 8, 2, 2
39	pr144	best is 58537 starting at 1 in loop 3, 7, 2, 0	best is 58537 starting at 59 in loop 5, 2, 2, 0
40	pr152	best is 73682 starting at 4 in loop 5, 7, 1, 2	best is 73682 starting at 34 in loop 5, 6, 2, 1
41	pr76	best is 108406 starting at 27 in loop 3, 5, 2, 1 best is 108159 starting at 14 in loop 1, 8, 2, 2	best is 108159 starting at 59 in loop 3, 2, 2, 1
42	rat195	best is 2339 starting at 71 in loop 4, 1, 1, 1	best is 2326 starting at 15 in loop 6, 6, 2, 2

	Name	Initial Passes	Final Pass
		best is 2326 starting at 14 in loop 7, 3, 2, 1	
43	rat99	best is 1211 starting at 64 in loop 3, 8, 2, 0	best is 1211 starting at 50 in loop 4, 8, 2, 2
44	rd100	best is 7930 starting at 14 in loop 3, 5, 1, 1 best is 7910 starting at 15 in loop 5, 6, 2, 0	best is 7910 starting at 4 in loop 7, 6, 2, 2
45	si175	best is 21442 starting at 164 in loop 6, 4, 1, 1 best is 21410 starting at 153 in loop 3, 1, 1, 2 best is 21407 starting at 34 in loop 8, 1, 1, 1	best is 21407 starting at 64 in loop 1, 10, 2, 0
46	st70	best is 676 starting at 2 in loop 2, 4, 2, 1 best is 675 starting at 16 in loop 7, 8, 1, 2	best is 675 starting at 1 in loop 6, 2, 0, 0
47	swiss42	best is 1273 starting at 1 in loop 5, 8, 1, 1	best is 1273 starting at 10 in loop 5, 0, 0, 0
48	ts225	best is 126995 starting at 43 in loop 7, 8, 1, 1 best is 126643 starting at 6 in loop 2, 4, 2, 1	best is 126643 starting at 13 in loop 4, 5, 2, 2
49	tsp225	best is 3921 starting at 95 in loop 5, 2, 2, 1 best is 3916 starting at 78 in loop 1, 4, 2, 1	best is 3916 starting at 174 in loop 7, 8, 2, 0
50	u159	best is 42080 starting at 155 in loop 5, 5, 1, 1	best is 42080 starting at 7 in loop 2, 3, 2, 2
51	ulysses16	best is 6859 starting at 9 in loop 1, 8, 2, 0	best is 6859 starting at 5 in loop 3, 8, 2, 0
52	ulysses22	best is 7013 starting at 1 in loop 1, 8, 1, 0	best is 7013 starting at 3 in loop 8, 2, 0, 0

Table 5.2 ATSP Performance: Solution Trail

No	Name	Initial Passes	Final Pass
1	br17	best is 39 starting at 6 in loop 5, 9, 0, 0	best is 39 starting at 6 in loop 10, 0, 0, 0
2	ft53	best is 6980 starting at 29 in loop 4, 2, 2, 2 best is 6915 starting at 4 in loop 8, 8, 1, 3 best is 6905 starting at 49 in loop 2, 10, 2, 2	best is 6905 starting at 30 in loop 2, 9, 0, 0
3	ft70	best is 39116 starting at 31 in loop 2, 2, 1, 2 best is 38836 starting at 57 in loop 1, 12, 2, 2 best is 38781 starting at 3 in loop 1, 1, 1, 0 best is 38694 starting at 66 in loop 9, 3, 2, 2 best is 38673 starting at 4 in loop 1, 12, 2, 0	best is 38673 starting at 67 in loop 7, 10, 2, 3
4	ftv33	best is 1286 starting at 28 in loop 10, 10, 1, 0	best is 1286 starting at 12 in loop 4, 9, 0, 0
5	ftv35	best is 1473 starting at 34 in loop 1, 8, 2, 0	best is 1473 starting at 19 in loop 7, 9, 0, 0
6	ftv38	best is 1530 starting at 10 in loop 9, 3, 2, 0	best is 1530 starting at 34 in loop 10, 0, 0, 0
7	ftv44	best is 1613 starting at 12 in loop 8, 7, 1, 3	best is 1613 starting at 2 in loop 4, 12, 2, 3
8	ftv47	best is 1776 starting at 3 in loop 5, 1, 1, 2	best is 1776 starting at 3 in loop 10, 0, 0, 0
9	ftv55	best is 1608 starting at 7 in loop 1, 6, 1, 3	best is 1608 starting at 11 in loop 6, 9, 0, 0
10	ftv64	best is 1851 starting at 8 in loop 7, 8, 1, 2 best is 1848 starting at 14 in loop 3, 7, 1, 3 best is 1839 starting at 6 in loop 1, 4, 2, 2	best is 1839 starting at 6 in loop 1, 4, 2, 2
11	ftv70	best is 1971 starting at 26 in loop 6, 2, 1, 1 best is 1954 starting at 18 in loop 4, 7, 1, 3 best is 1950 starting at 6 in loop 4, 2, 2, 1	best is 1950 starting at 3 in loop 7, 11, 0, 0
12	ftv90	best is 1599 starting at 66 in loop 8, 9, 1, 2 best is 1579 starting at 32 in loop 8, 7, 1, 2	best is 1579 starting at 19 in loop 9, 11, 0, 0
13	ftv100	best is 1844 starting at 62 in loop 3, 8, 2, 2 best is 1815 starting at 28 in loop 9, 9, 1, 2 best is 1788 starting at 28 in loop 6, 11, 1, 2	best is 1788 starting at 20 in loop 9, 2, 2, 0
14	kro124	best is 37376 starting at 23 in loop 4, 8, 1, 2 best is 36750 starting at 63 in loop 1, 6, 1, 3 best is 36447 starting at 13 in loop 1, 6, 3, 3 best is 36230 starting at 13 in loop 4, 12, 1, 2	best is 36230 starting at 13 in loop 2, 9, 0, 0
15	p43	best is 5620 starting at 15 in loop 6, 8, 2, 2	best is 5620 starting at 27 in loop 2, 10, 0, 0
16	ry48p	best is 14459 starting at 21 in loop 11, 10, 1, 2 best is 14429 starting at 37 in loop 7, 3, 3, 2 best is 14422 starting at 2 in loop 8, 9, 0, 0	best is 14422 starting at 39 in loop 7, 7, 2, 2

Table 5.3 STSP Performance: Time Elapsed

	Name	**# Cities**	**Tour Construction** (seconds)	**First Run** (seconds)	**First Pass** (seconds)	**Total Time** (seconds)
1	att48	48	0.990	0.170	45	100
2	bayg29	29	0.220	0.060	11	24
3	bays29	29	0.220	0.110	10	25
4	berlin52	52	1.320	0.600	68	161
5	bier127	127	27.020	10.760	2303	7036
6	brazil58	58	1.370	0.770	75	181
7	brg180	180	81.510	54.100	10133	27049
8	burma14	14	0.000	0.000	2	5
9	ch130	130	20.700	11.480	2107	2178
10	ch150	150	34.380	20.660	2183	3089
11	d198	198	109.250	67.720	11408	50831
12	dantzig42	42	0.330	0.330	37	85
13	eil101	101	8.680	4.560	1084	3681
14	eil51	51	1.320	0.650	74	164
15	eil76	76	4.340	1.480	246	577
16	fri26	26	0.050	0.060	7	16
17	gr120	120	15.050	8.560	1683	10166
18	gr137	137	23.450	14.010	2250	6705
19	gr17	17	0.000	0.000	3	6
20	gr202	202	119.840	74.040	17058	91959
21	gr21	21	0.050	0.000	4	10
22	gr24	24	0.110	0.160	8	17
23	gr48	48	1.100	0.380	50	203
24	gr96	96	10.830	6.310	619	4834
25	hk48	48	0.990	0.600	40	95
26	kroA100	100	11.760	6.260	496	992
27	kroA150	150	42.950	23.070	3045	22900
28	kroA200	200	107.490	69.150	11336	39924
29	kroB100	100	11.150	7.410	615	1905
30	kroB150	150	33.890	20.210	3086	18889
31	kroB200	200	109.140	72.170	12089	42118
32	kroC100	100	8.300	3.900	415	1076
33	kroD100	100	8.950	4.010	605	2572
34	kroE100	100	9.830	4.010	645	2003
35	lin105	105	9.230	4.940	700	1485
36	pr107	107	15.700	10.060	887	1974
37	pr124	124	15.220	9.060	788	1369
38	pr136	136	37.620	23.240	1979	8739
39	pr144	144	40.210	25.050	1174	2321
40	pr152	152	34.380	21.040	2025	4669
41	pr76	76	3.950	1.380	379	1301
42	rat195	195	89.040	62.720	6627	21761
43	rat99	99	7.800	3.740	416	863
44	rd100	100	7.960	3.960	628	2159
45	si175	175	65.250	45.540	4431	20470
46	st70	70	2.580	1.760	195	633
47	swiss42	42	0.820	0.280	28	71
48	ts225	225	146.870	105.460	14541	39075
49	tsp225	225	170.440	110.070	16677	45994
50	u159	159	42.890	26.860	3398	6388
51	ulysses16	16	0.000	0.000	3	7
52	ulysses22	22	0.000	0.000	5	12

Table 5.4 ATSP Performance: Time Elapsed

	Name	**# Cities**	**Tour Construction** (seconds)	**First Run** (seconds)	**First Pass** (seconds)	**Total Time** (seconds)
1	br17	17	0.000	0.000	4.00	11.00
2	ft53	53	1.040	0.660	207.00	798.00
3	ft70	70	3.630	1.590	744.00	4189.00
4	ftv33	34	0.280	0.440	31.00	58.00
5	ftv35	36	0.170	0.050	40.00	72.00
6	ftv38	39	0.270	0.440	51.00	113.00
7	ftv44	45	1.040	0.440	95.00	167.00
8	ftv47	48	0.550	0.490	144.00	249.00
9	ftv55	56	1.100	0.820	211.00	378.00
10	ftv64	65	2.300	1.430	341.00	1176.00
11	ftv70	71	2.690	2.090	508.00	1855.00
12	ftv90	91	7.800	5.160	1085.00	2547.00
13	ftv100	101	9.170	4.840	1713.00	5819.00
14	kro124	100	9.560	4.060	1609.00	6183.00
15	p43	43	0.770	0.280	77.00	192.00
16	ry48p	48	0.600	0.330	83.00	346.00

APPENDIX B

OPTIMAL TOURS

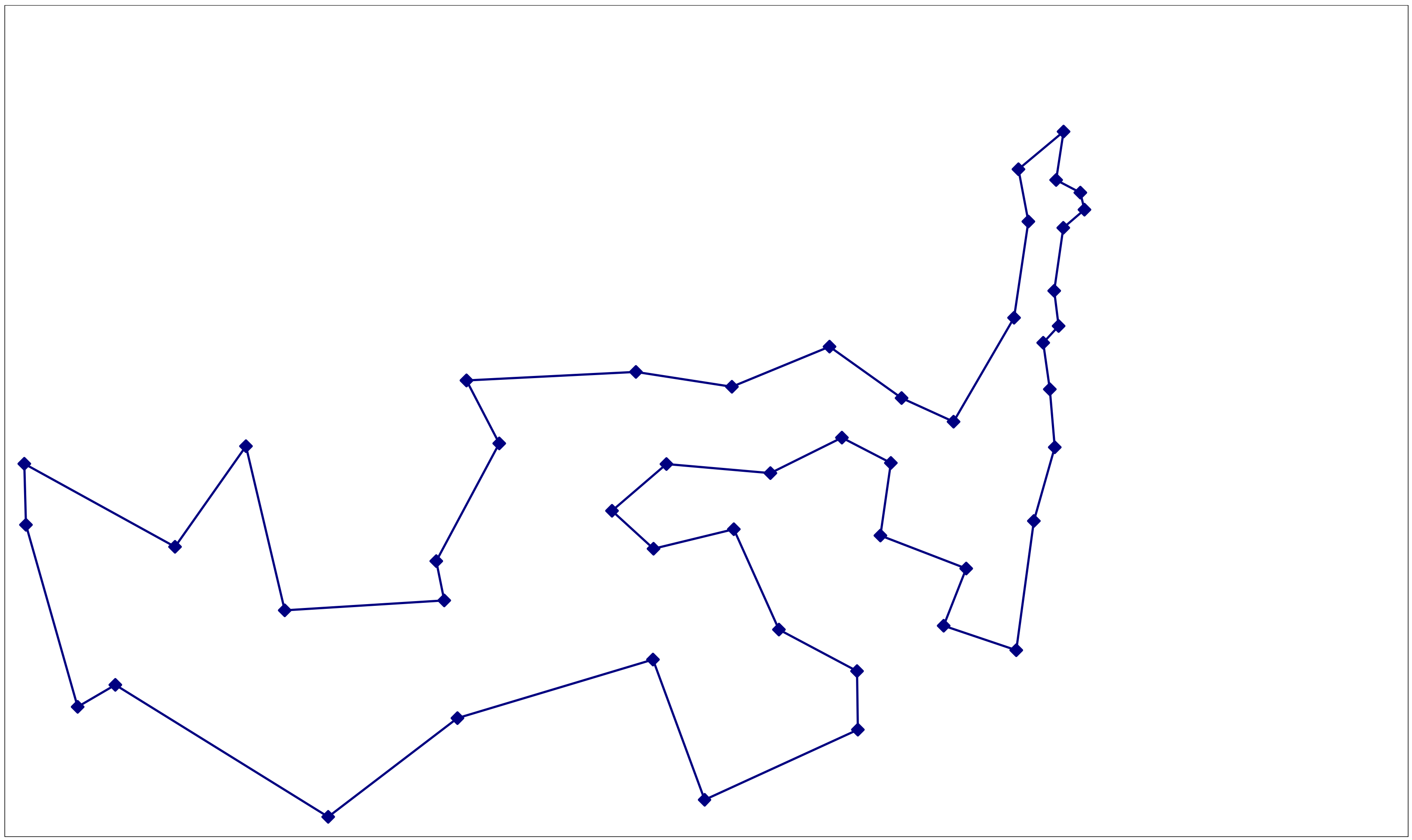


Figure 6.1 att48

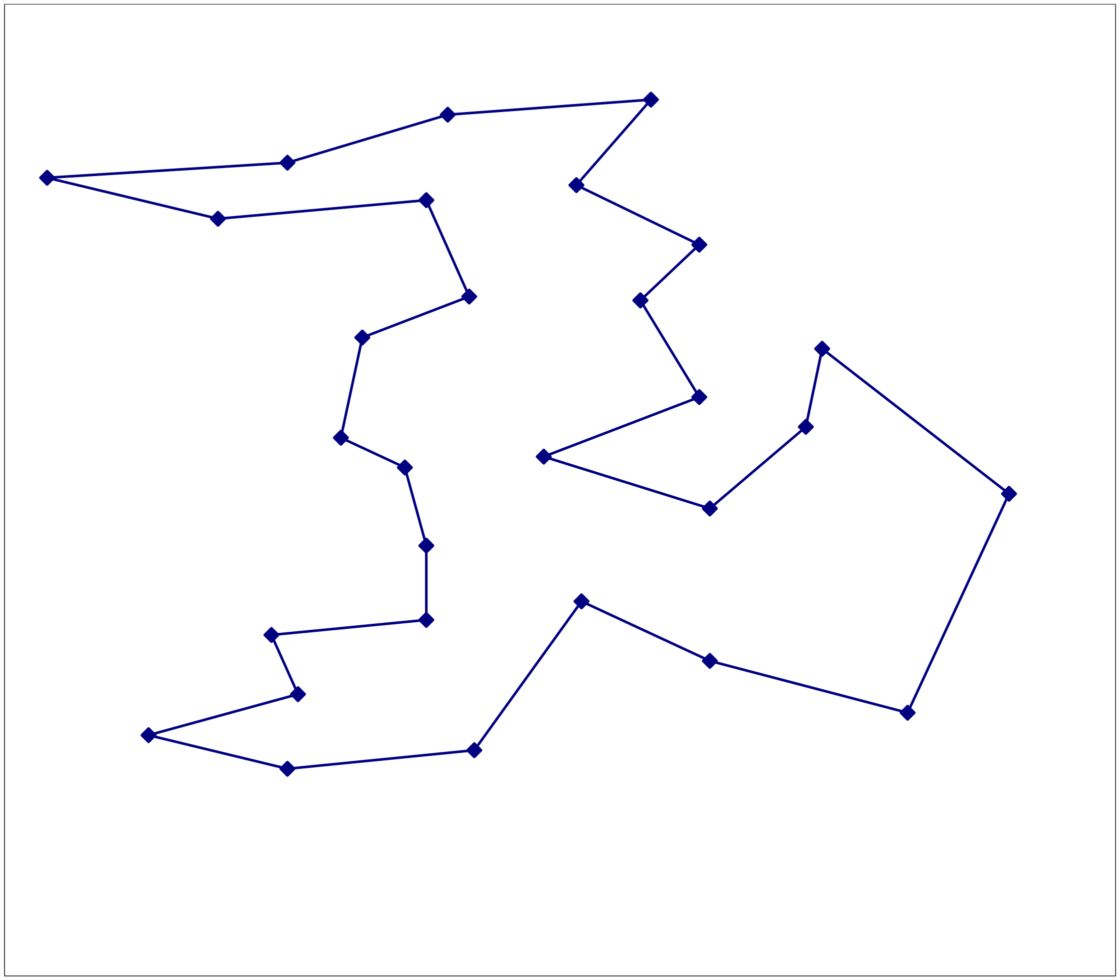


Figure 6.2 bayg29

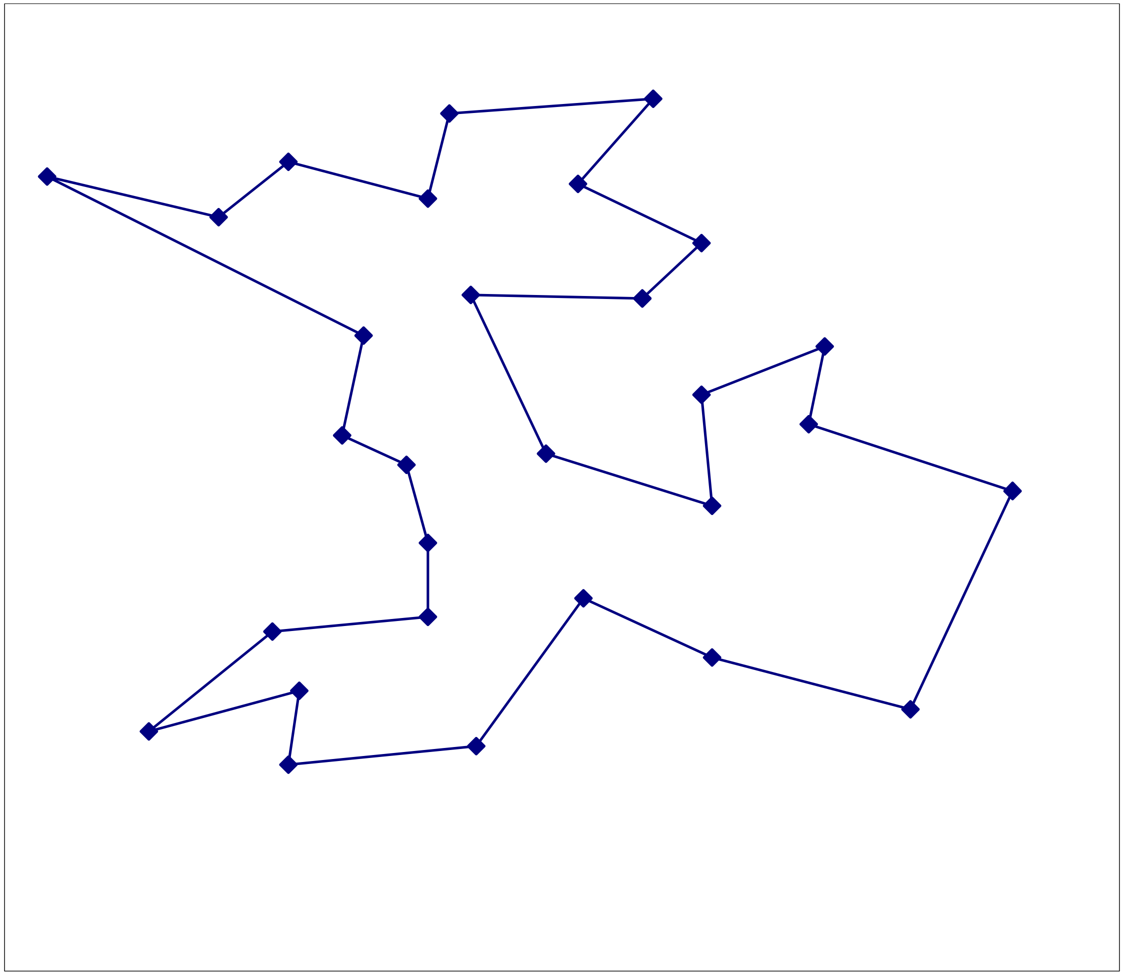


Figure 6.3 bays29

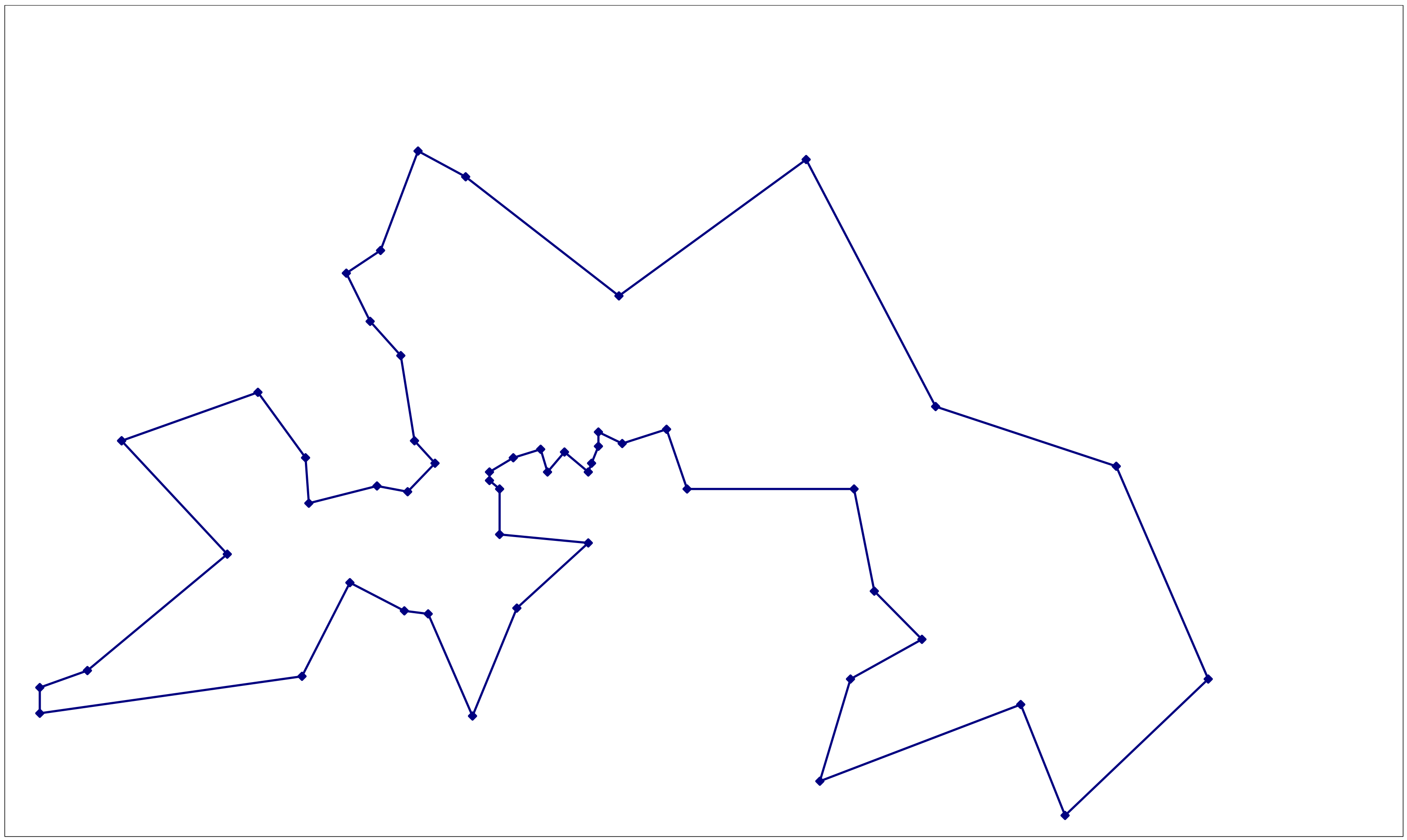


Figure 6.4 berlin52

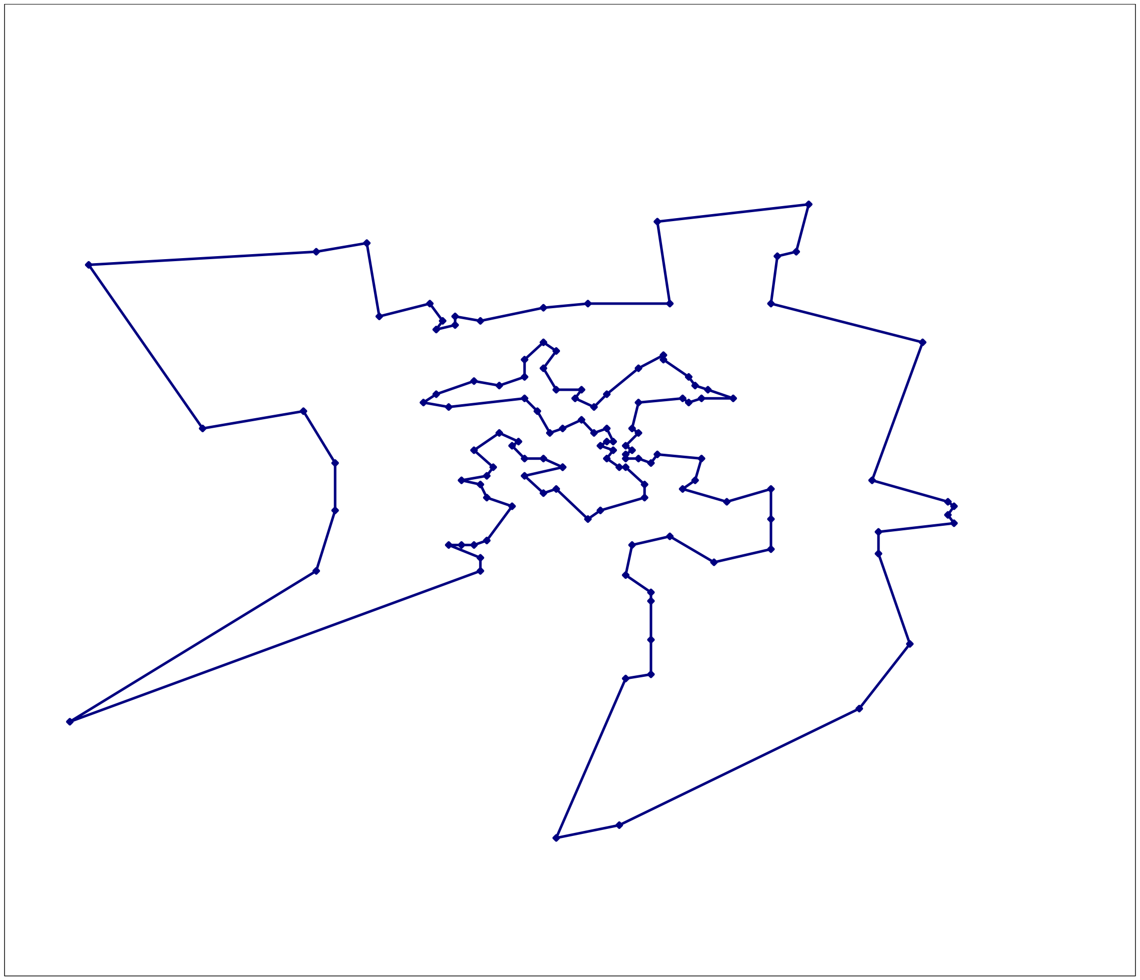


Figure 6.5 bier127

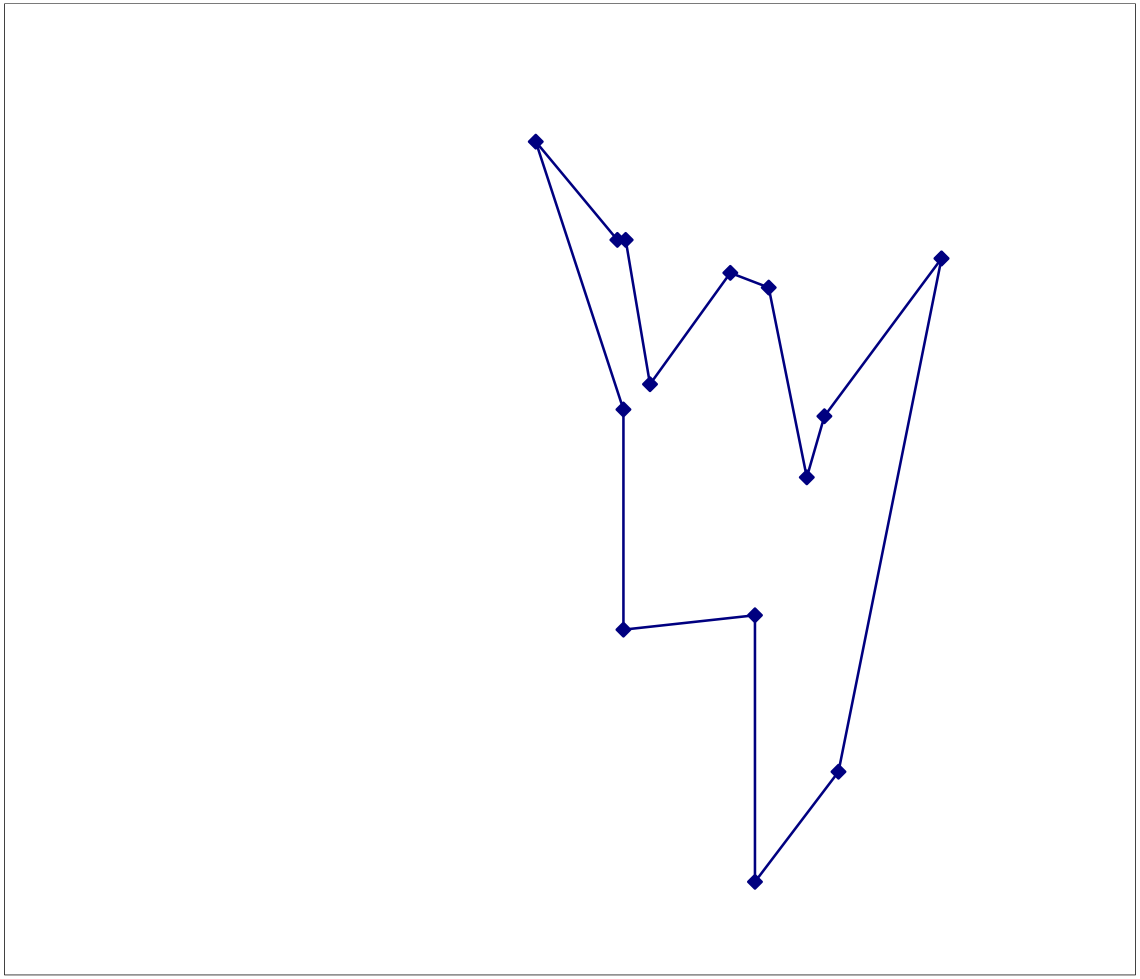


Figure 6.6 burma14

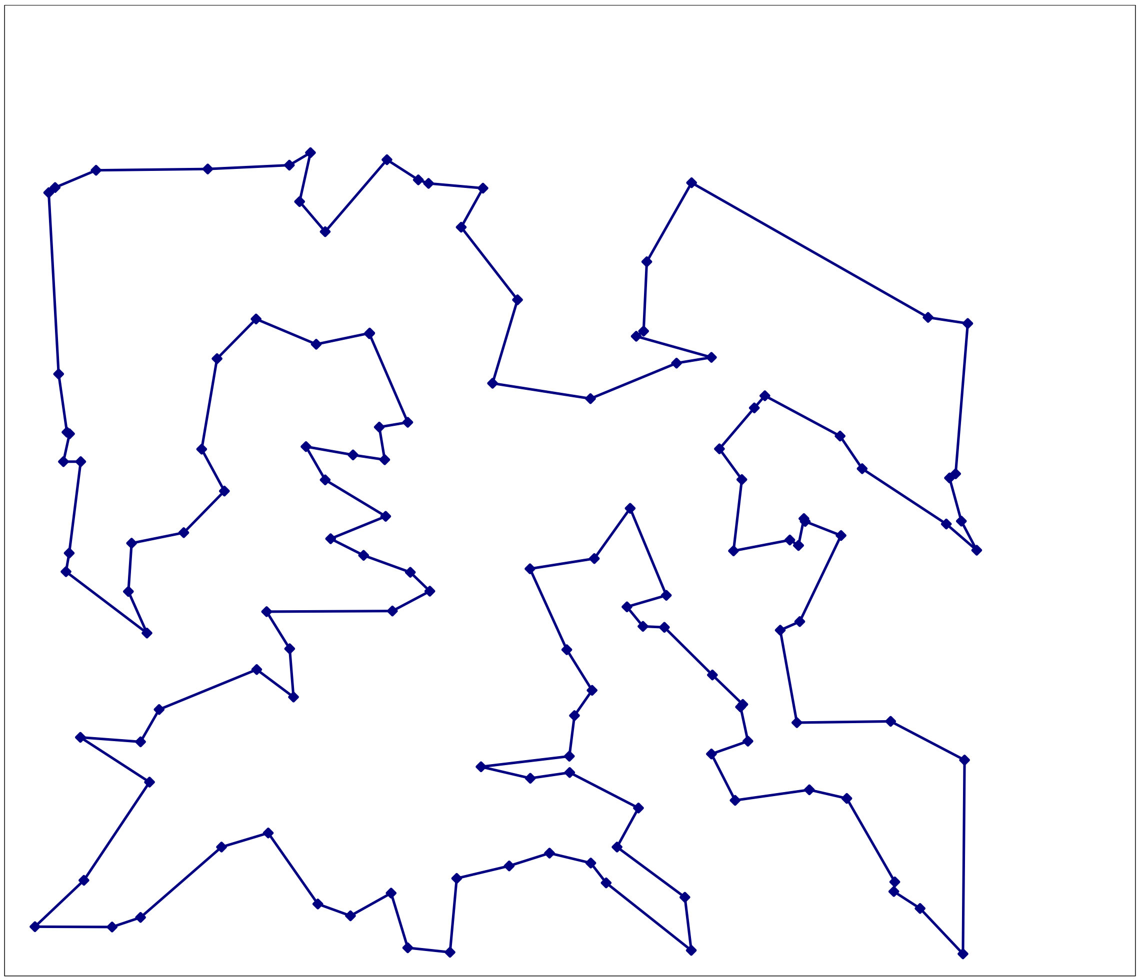


Figure 6.7 ch130

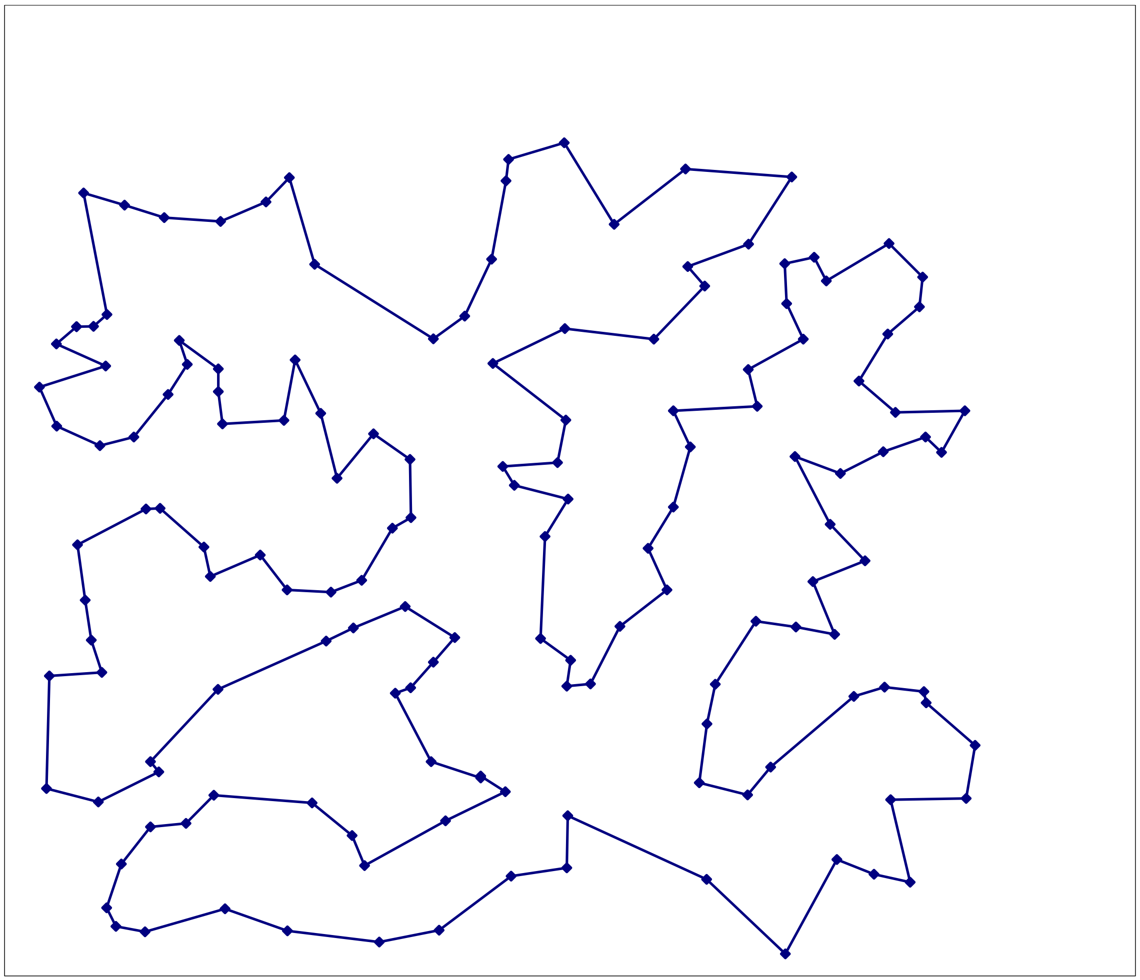


Figure 6.8 ch150

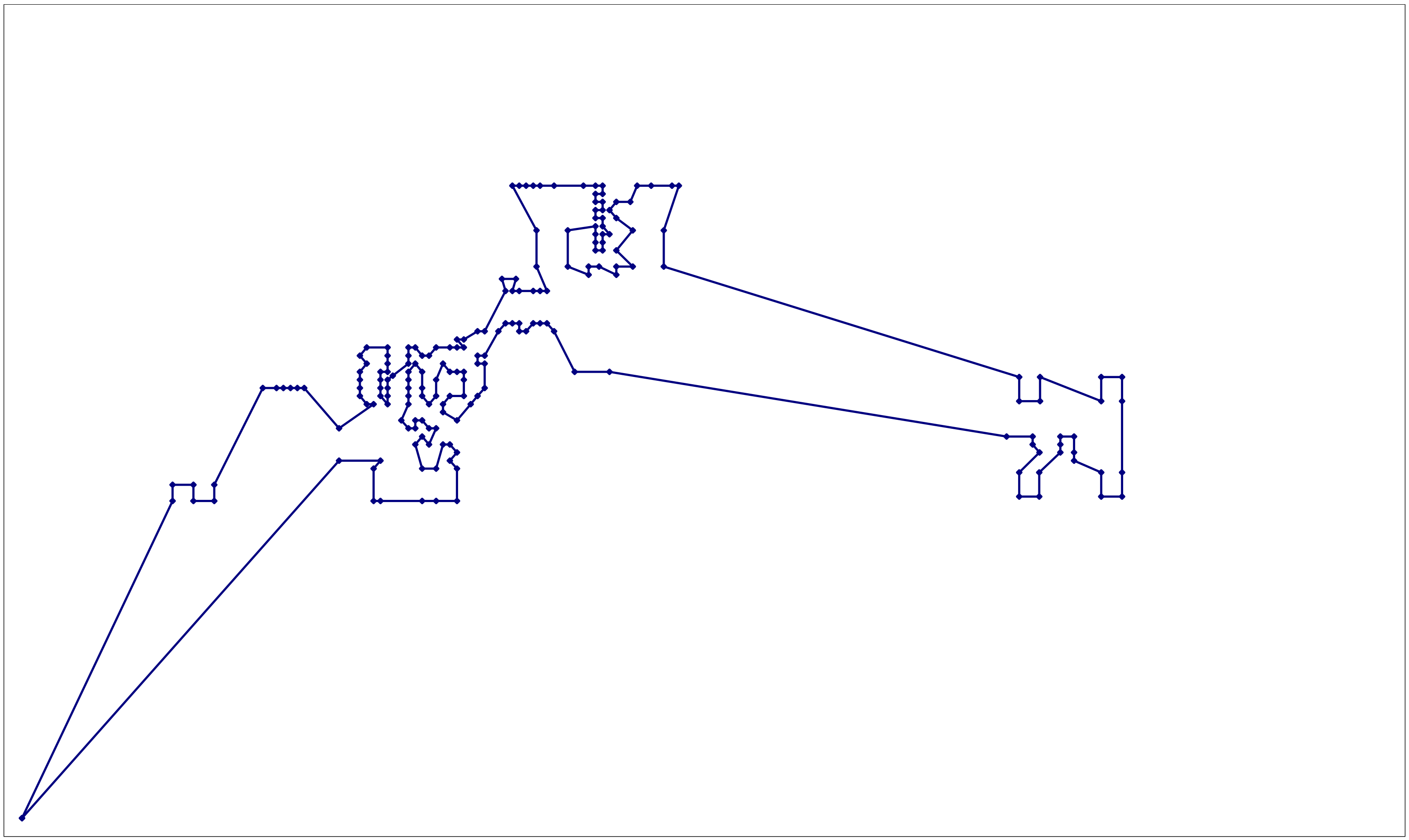


Figure 6.9 d198

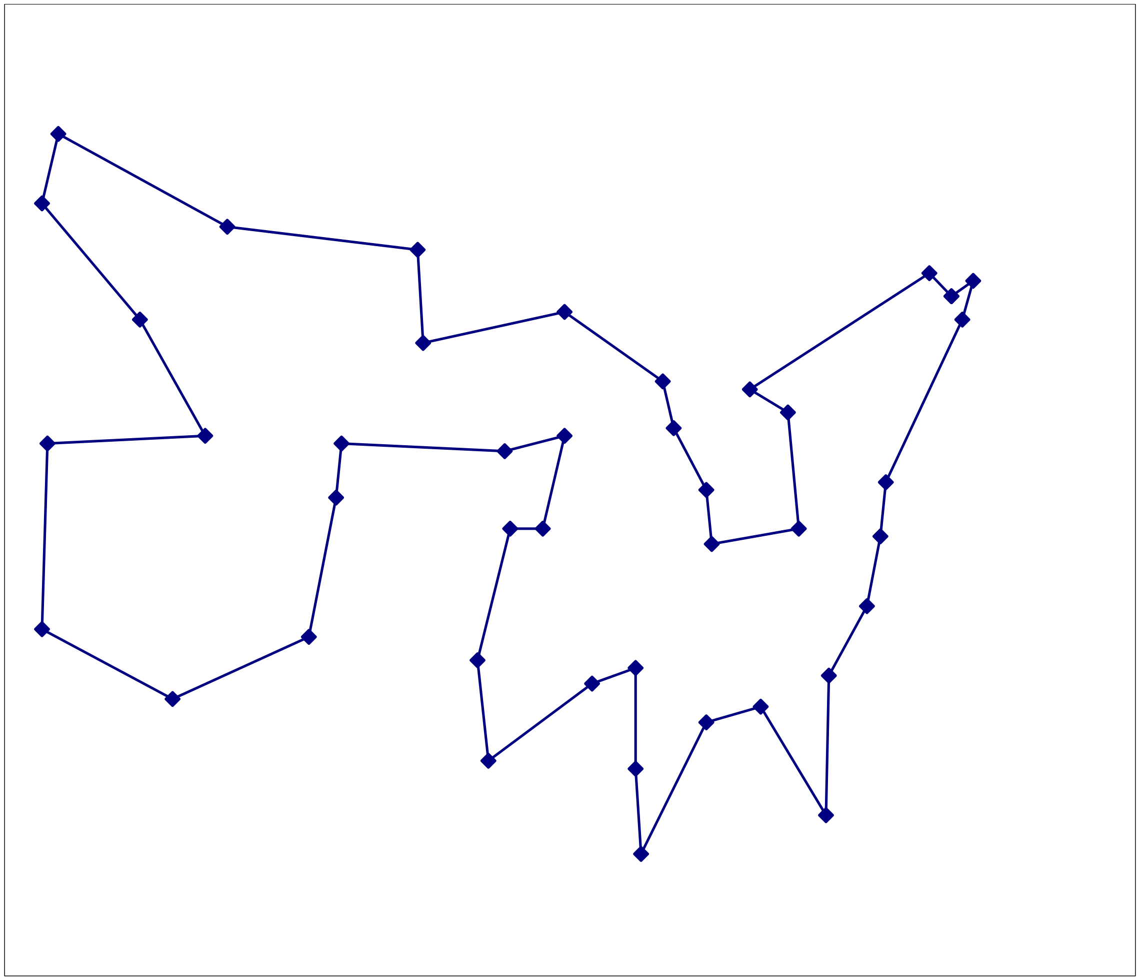


Figure 6.10 dantzig42

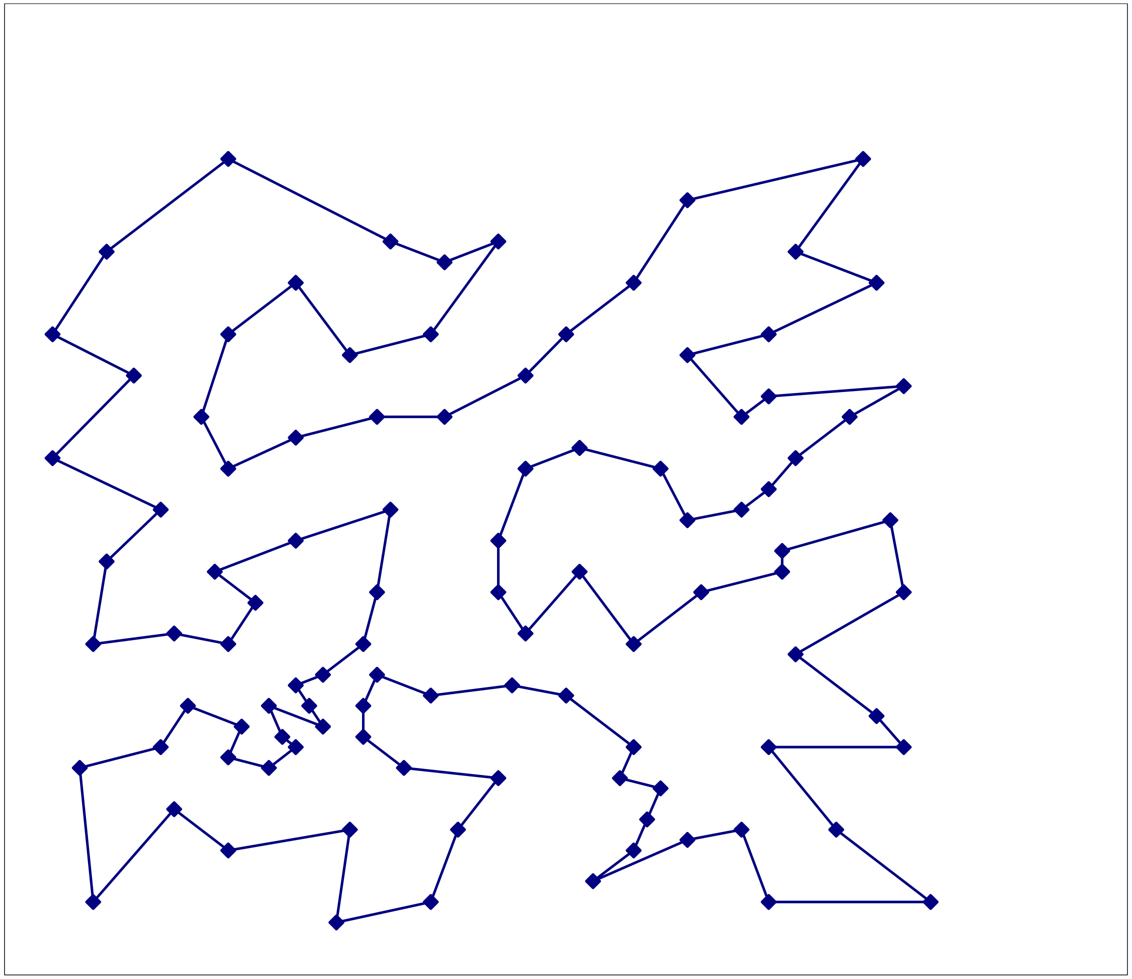


Figure 6.11 eil101

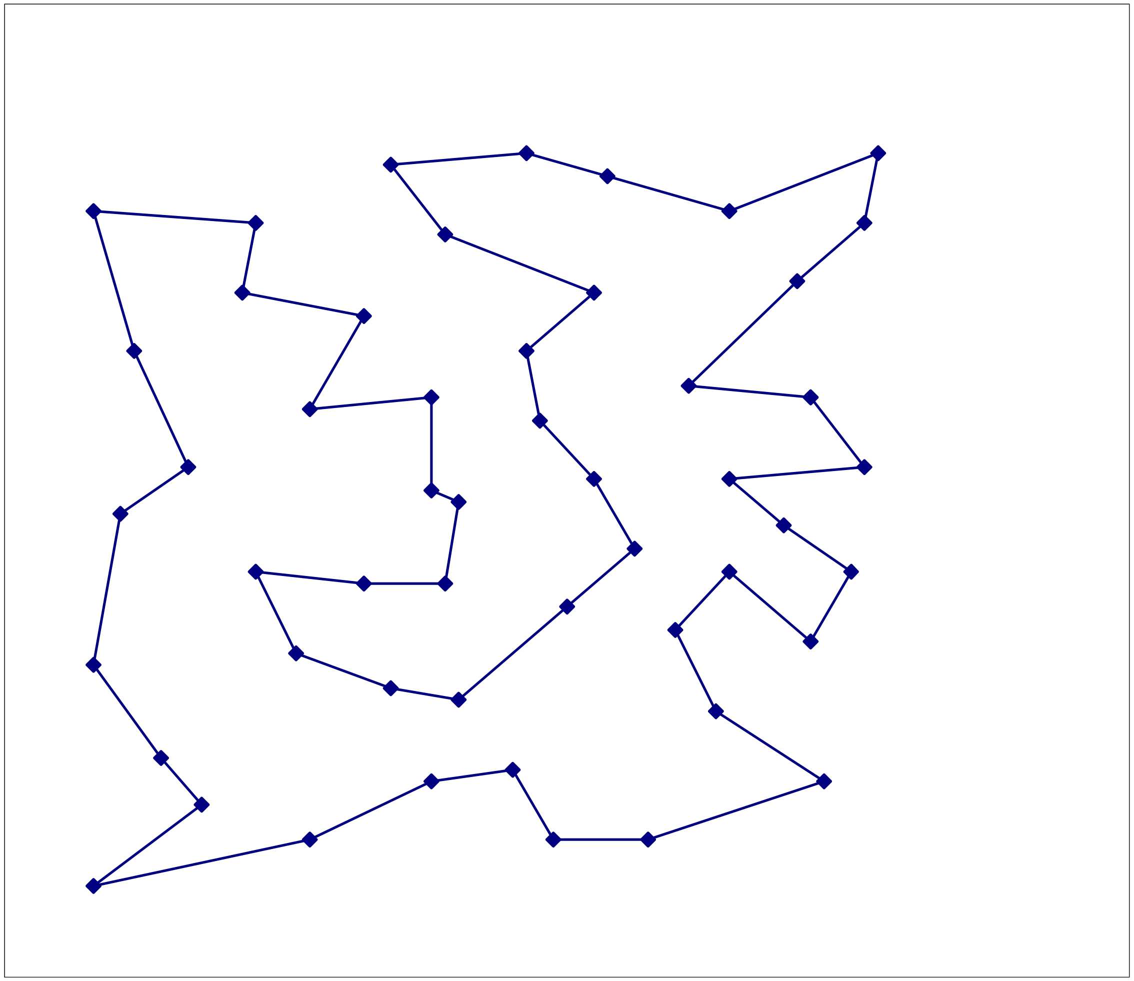


Figure 6.12 eil51

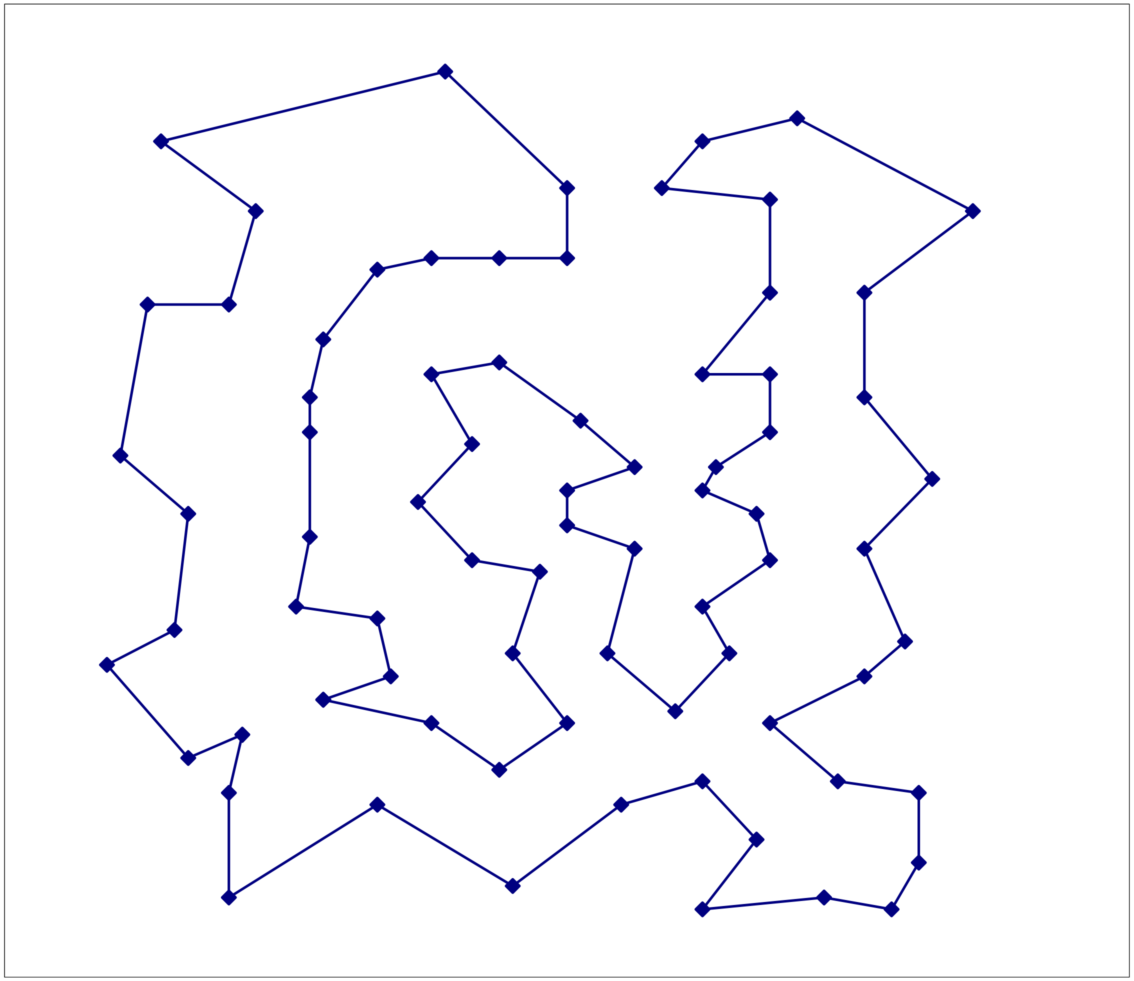


Figure 6.13 eil76

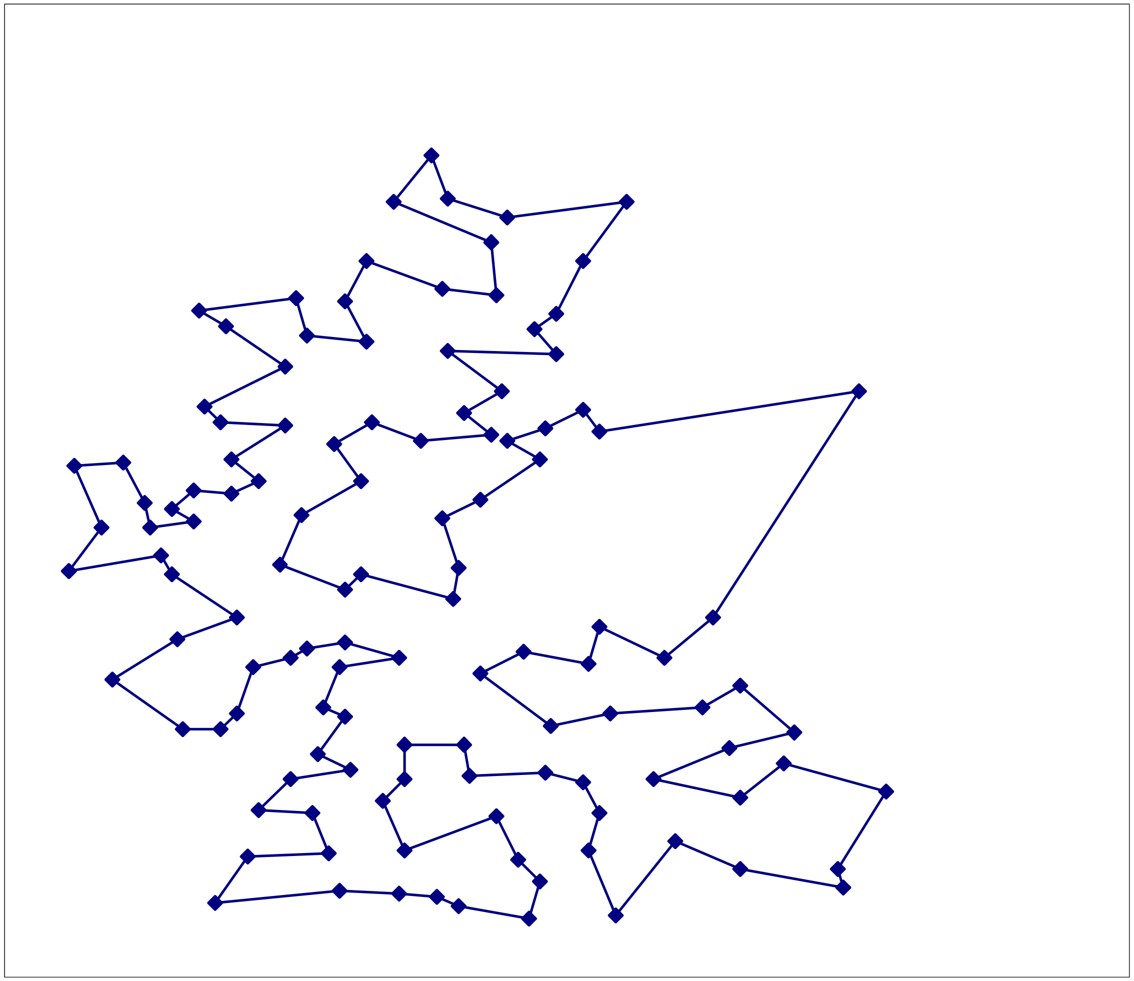


Figure 6.14 gr120

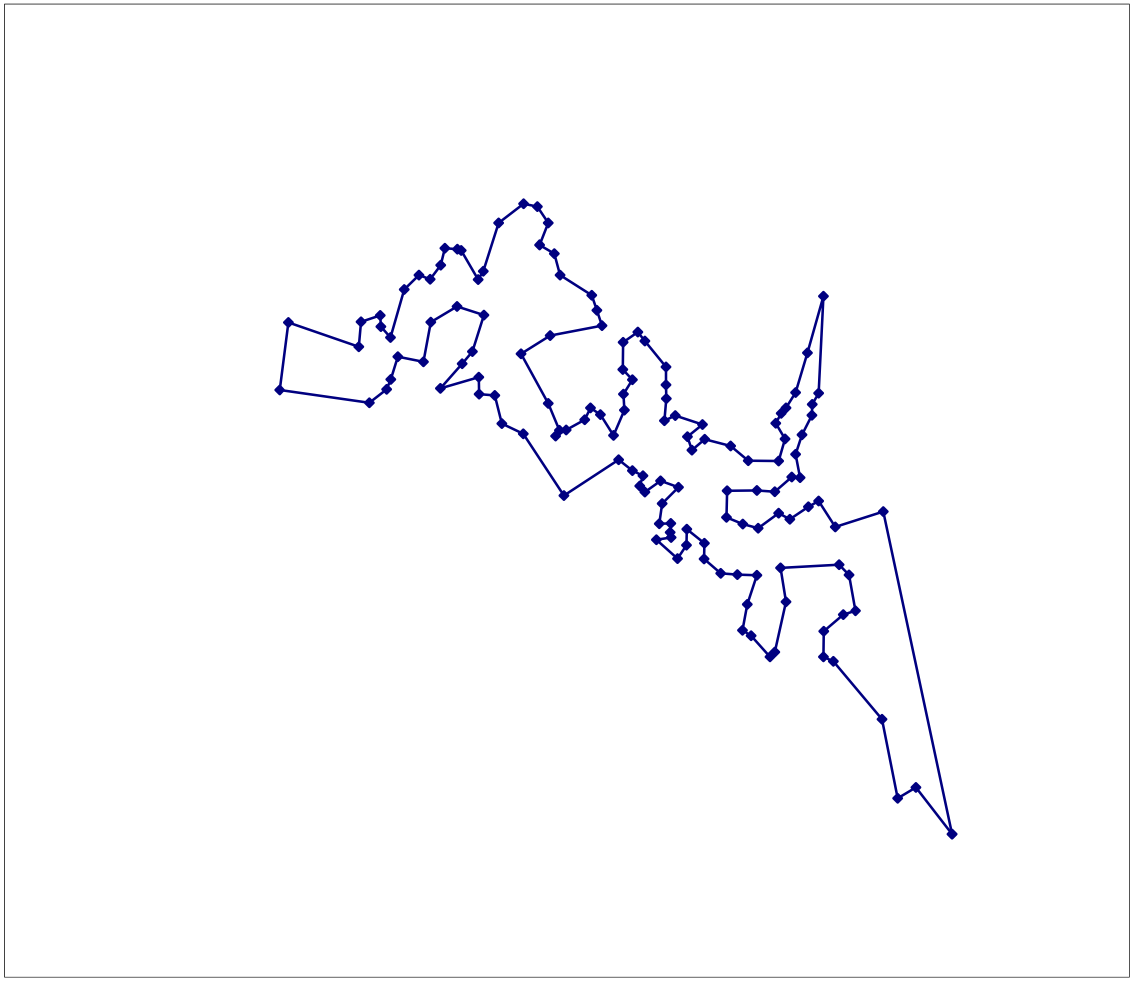


Figure 6.15 gr137

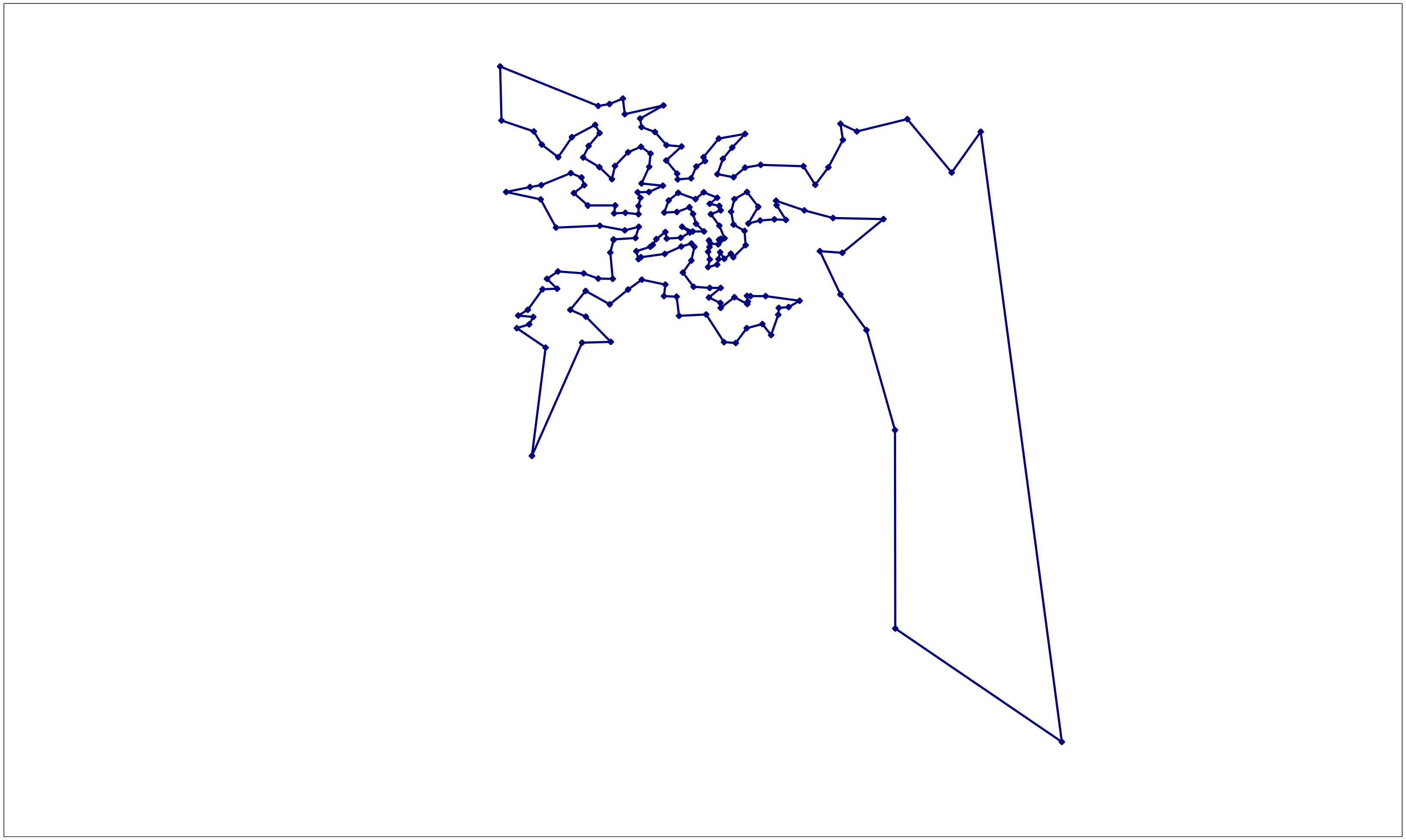


Figure 6.16 gr202

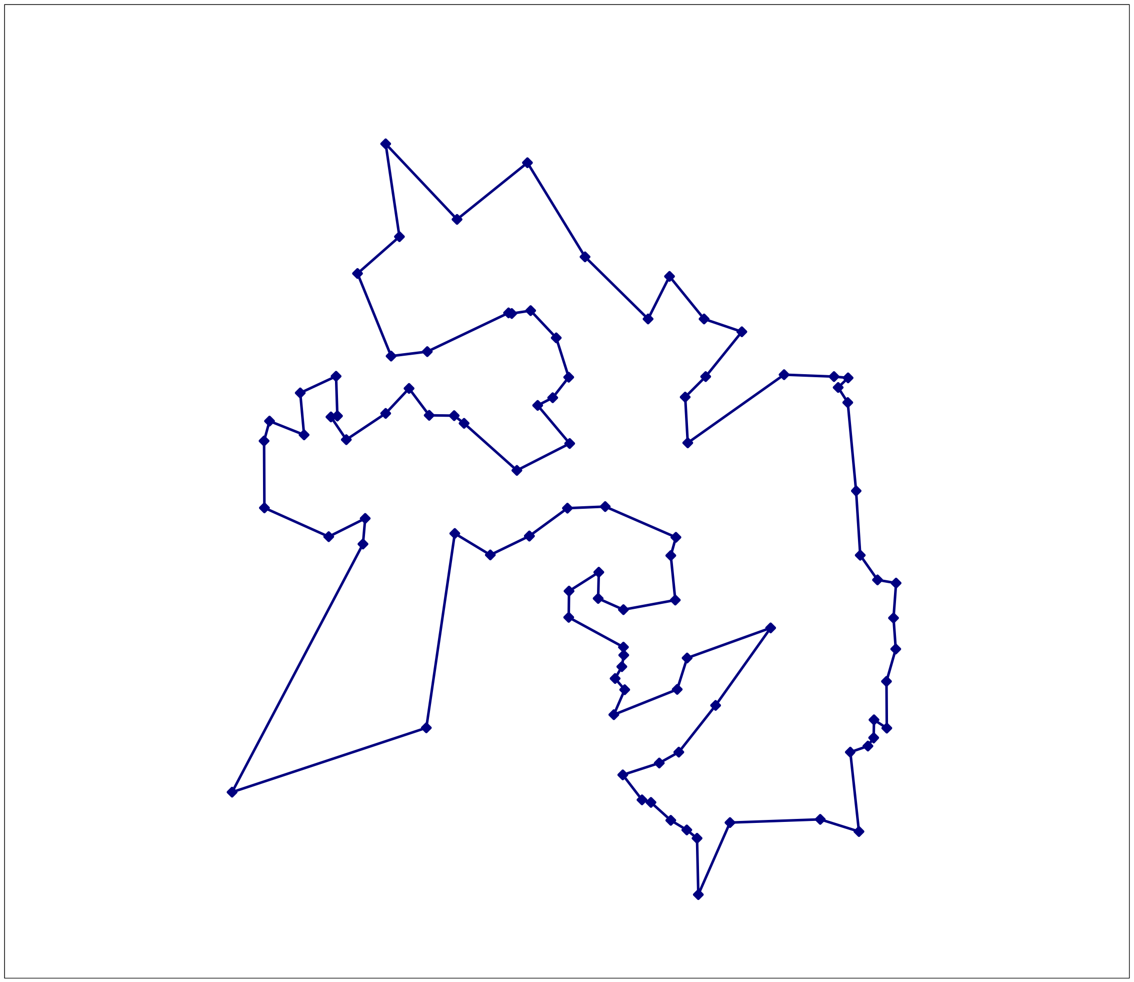


Figure 6.17 gr96

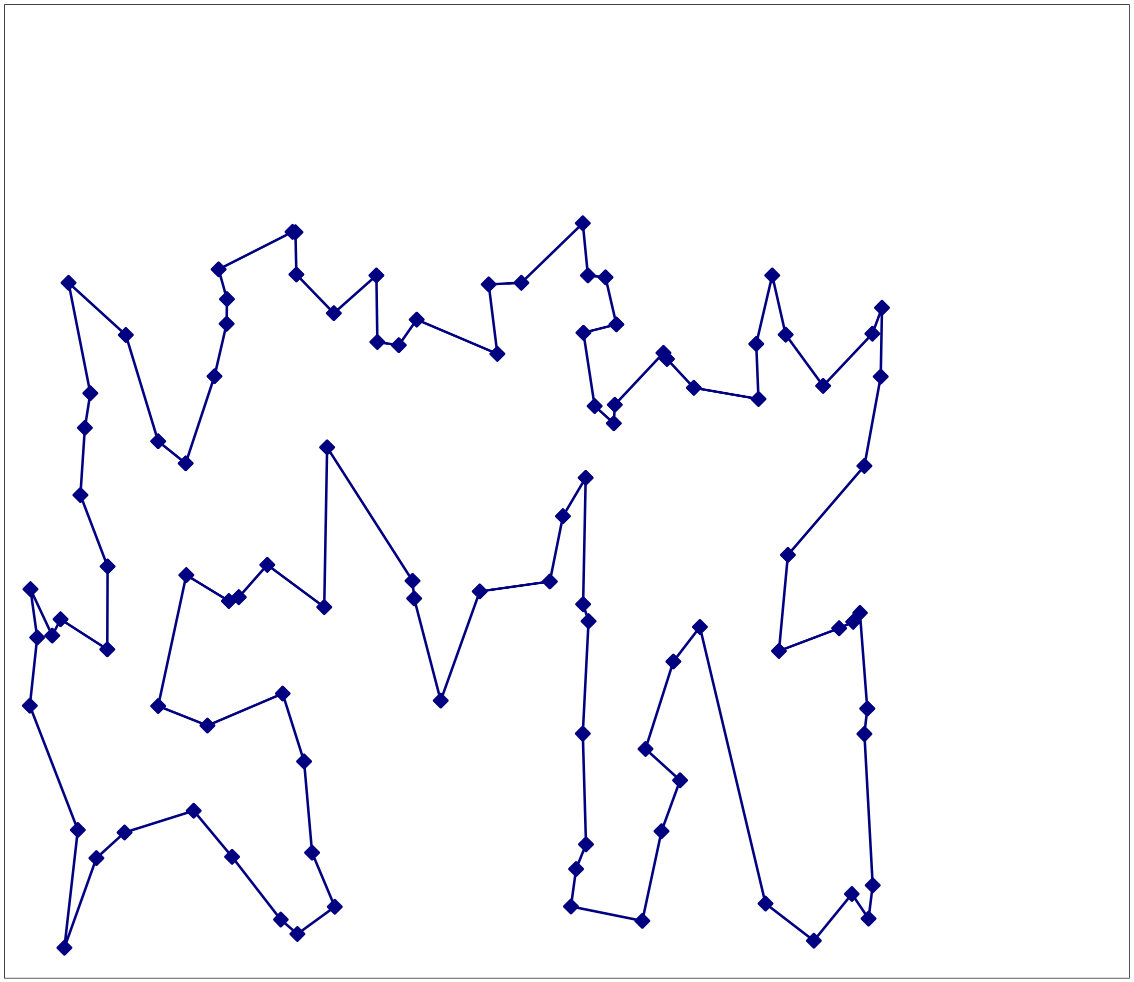


Figure 6.18 kroA100

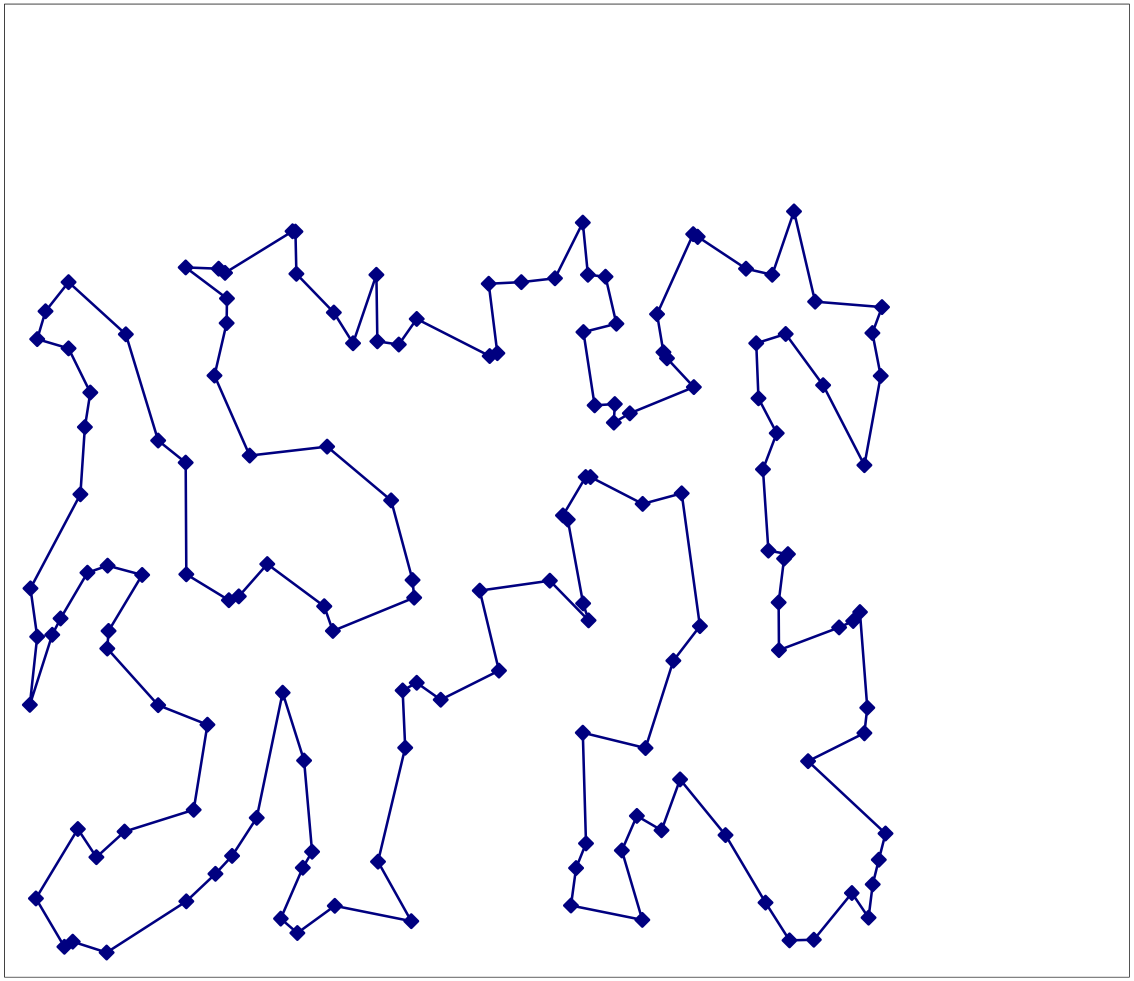


Figure 6.19 kroA150

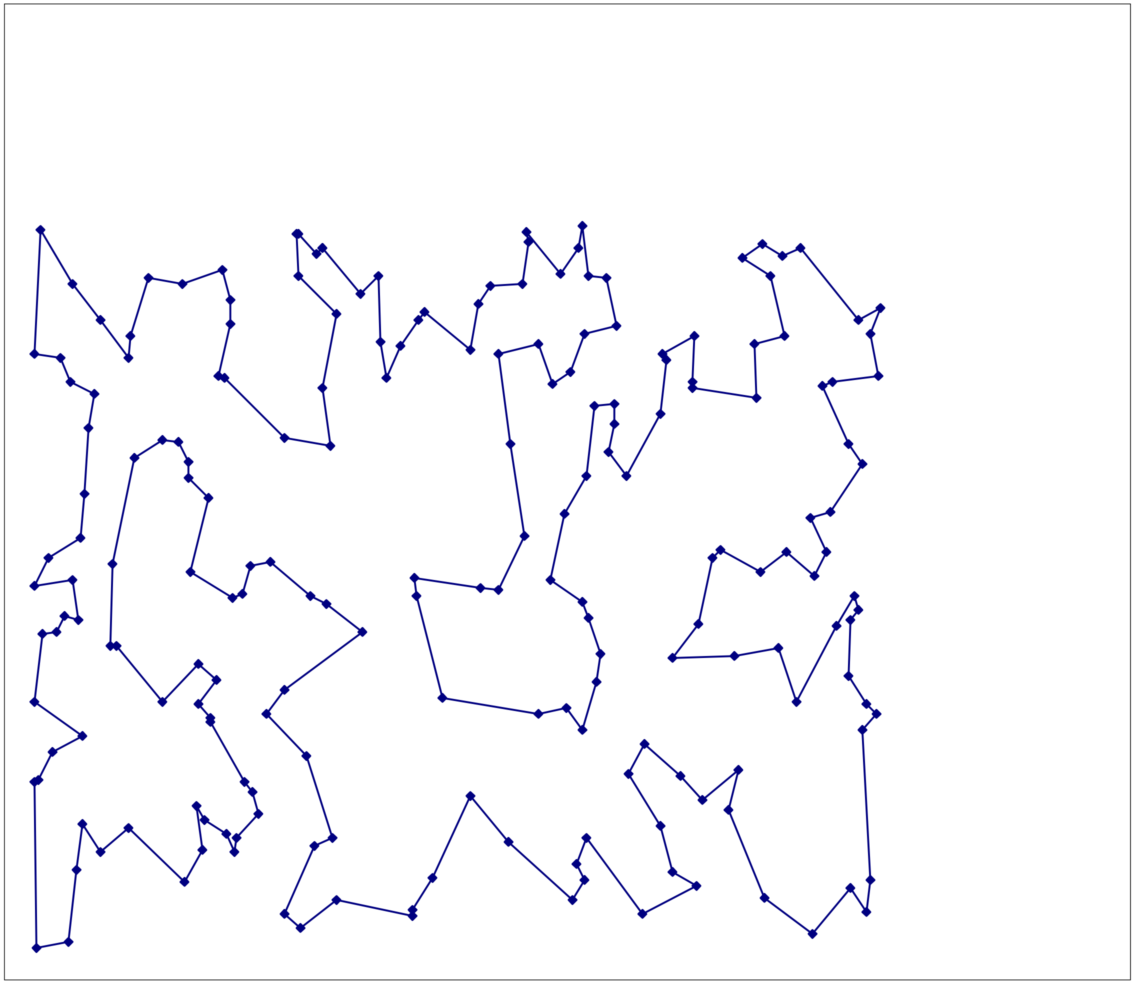


Figure 6.20 kroA200

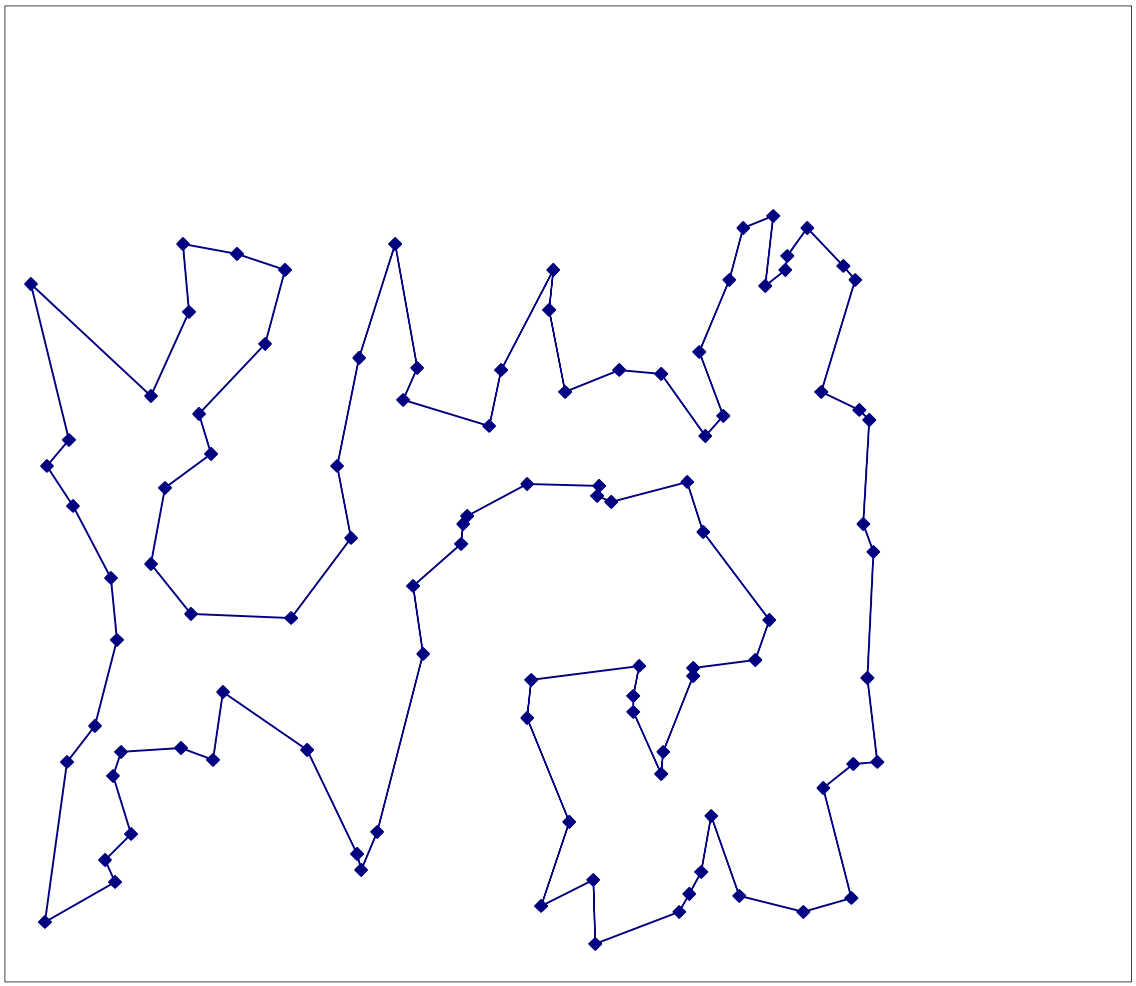


Figure 6.21 kroB100

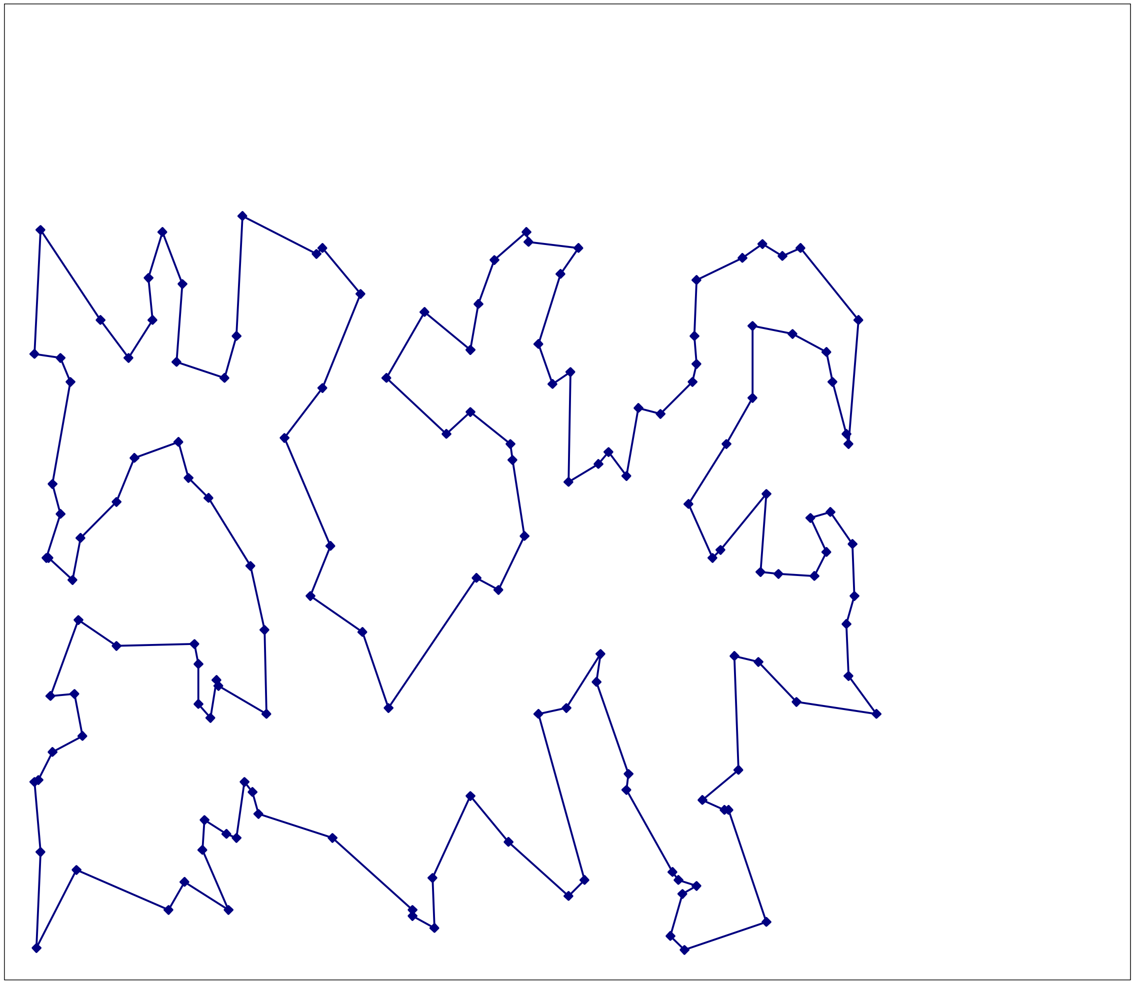


Figure 6.22 kroB150

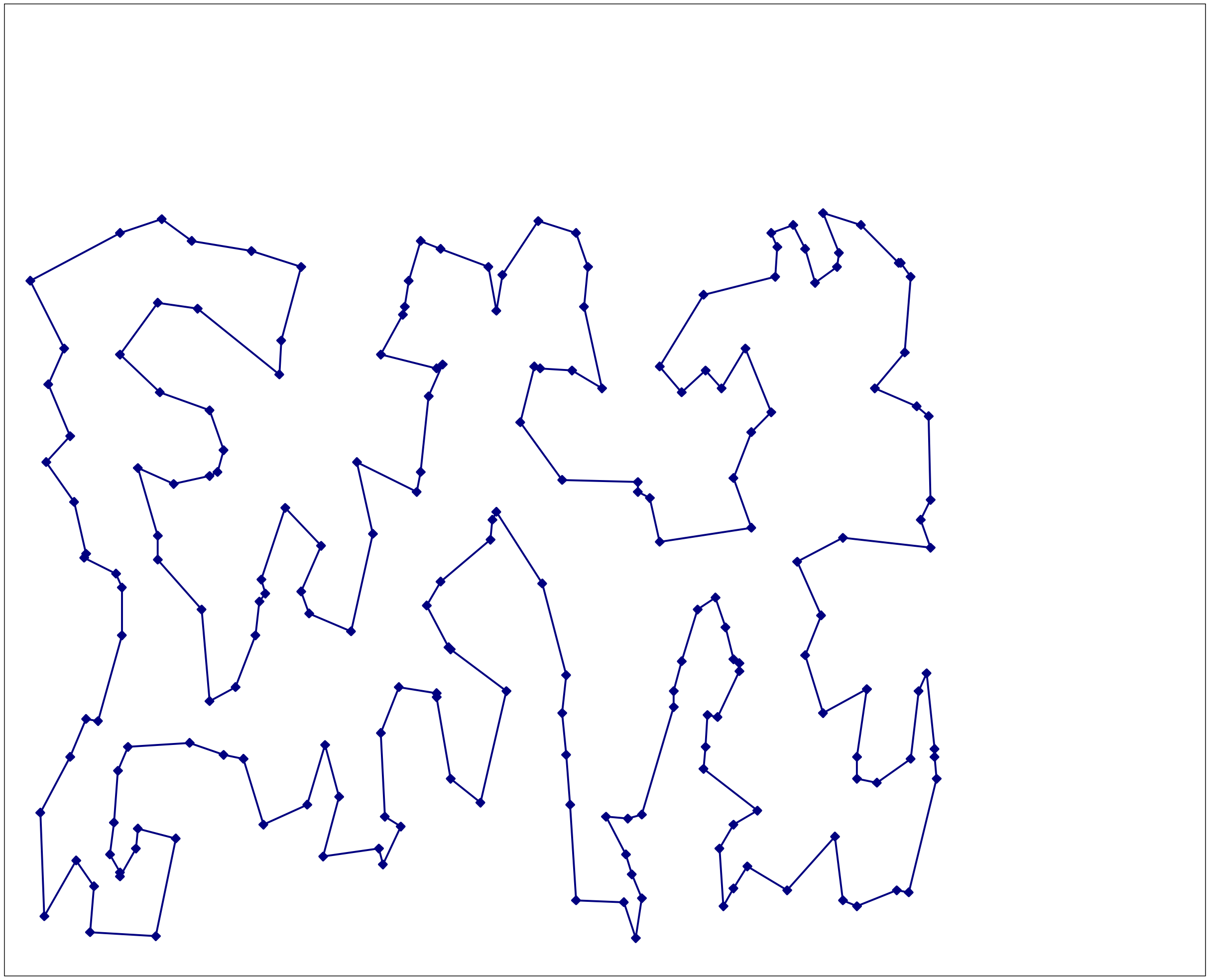


Figure 6.23 kroB200

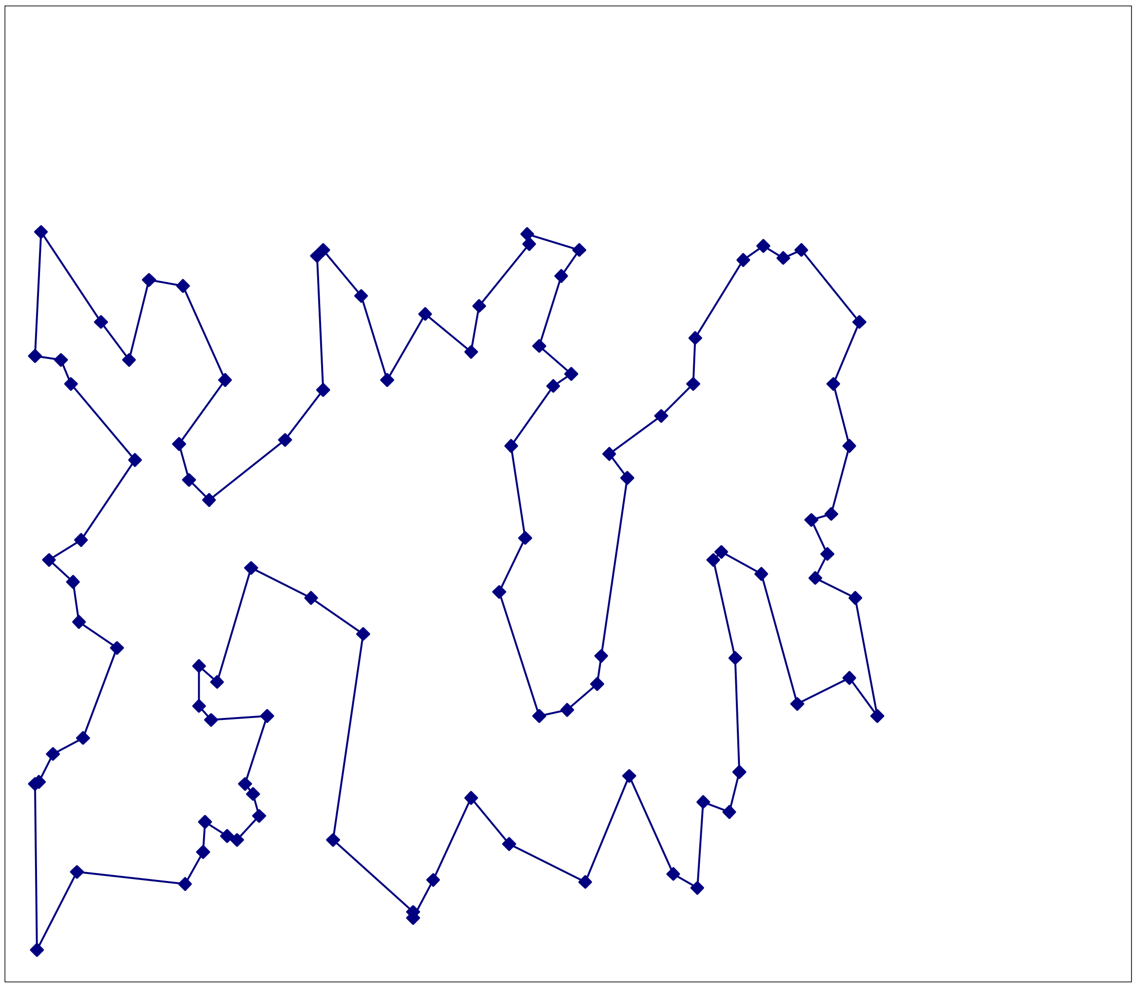


Figure 6.24 kroC100

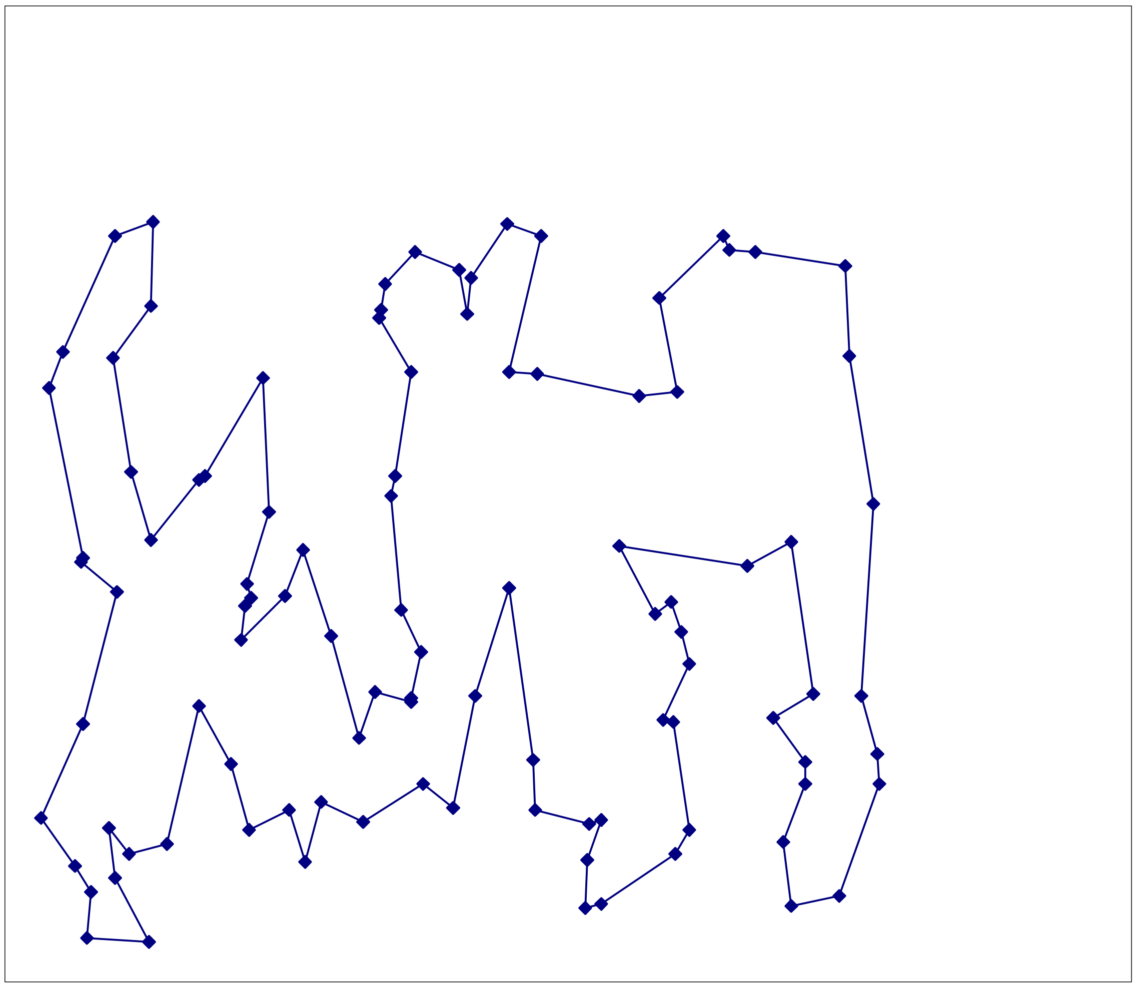


Figure 6.25 kroD100

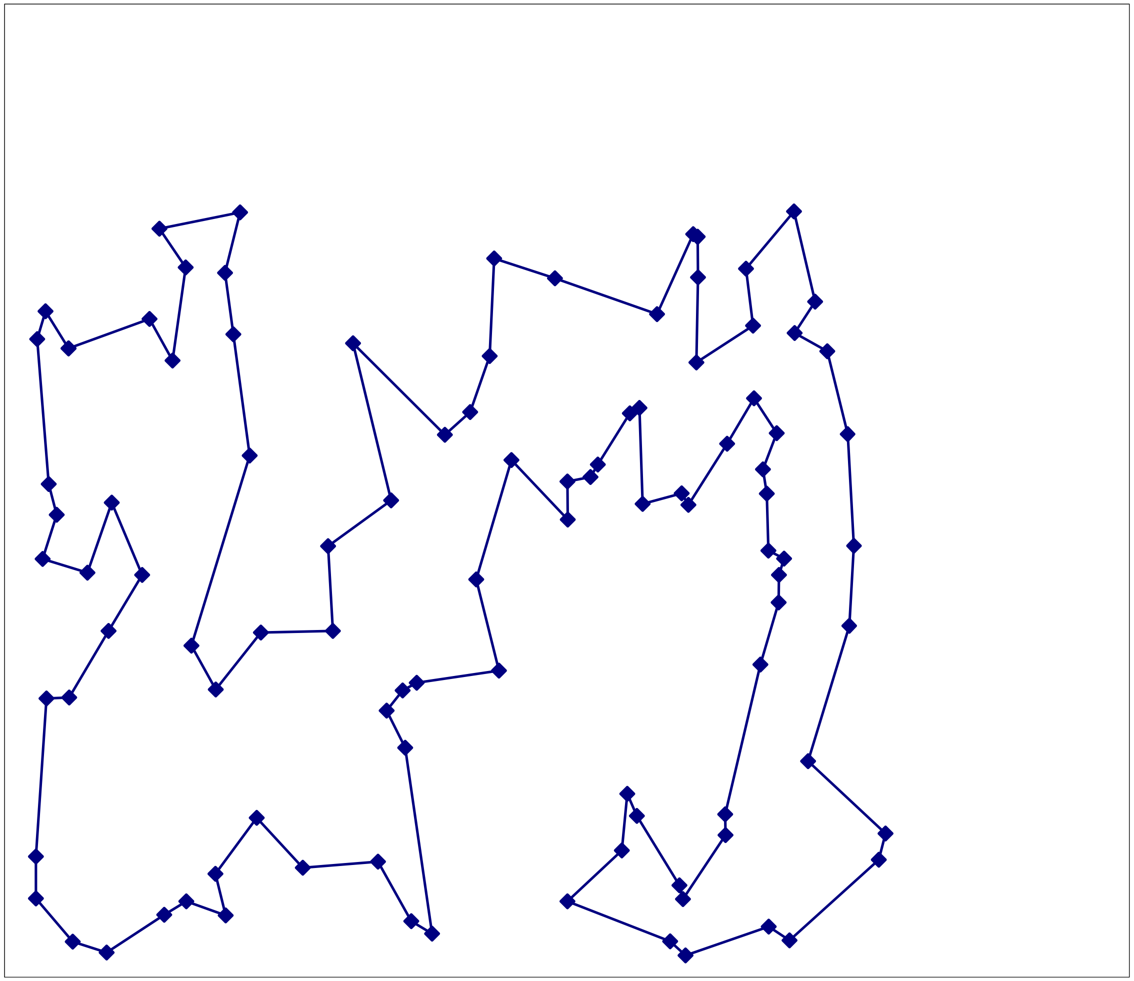


Figure 6.26 kroE100

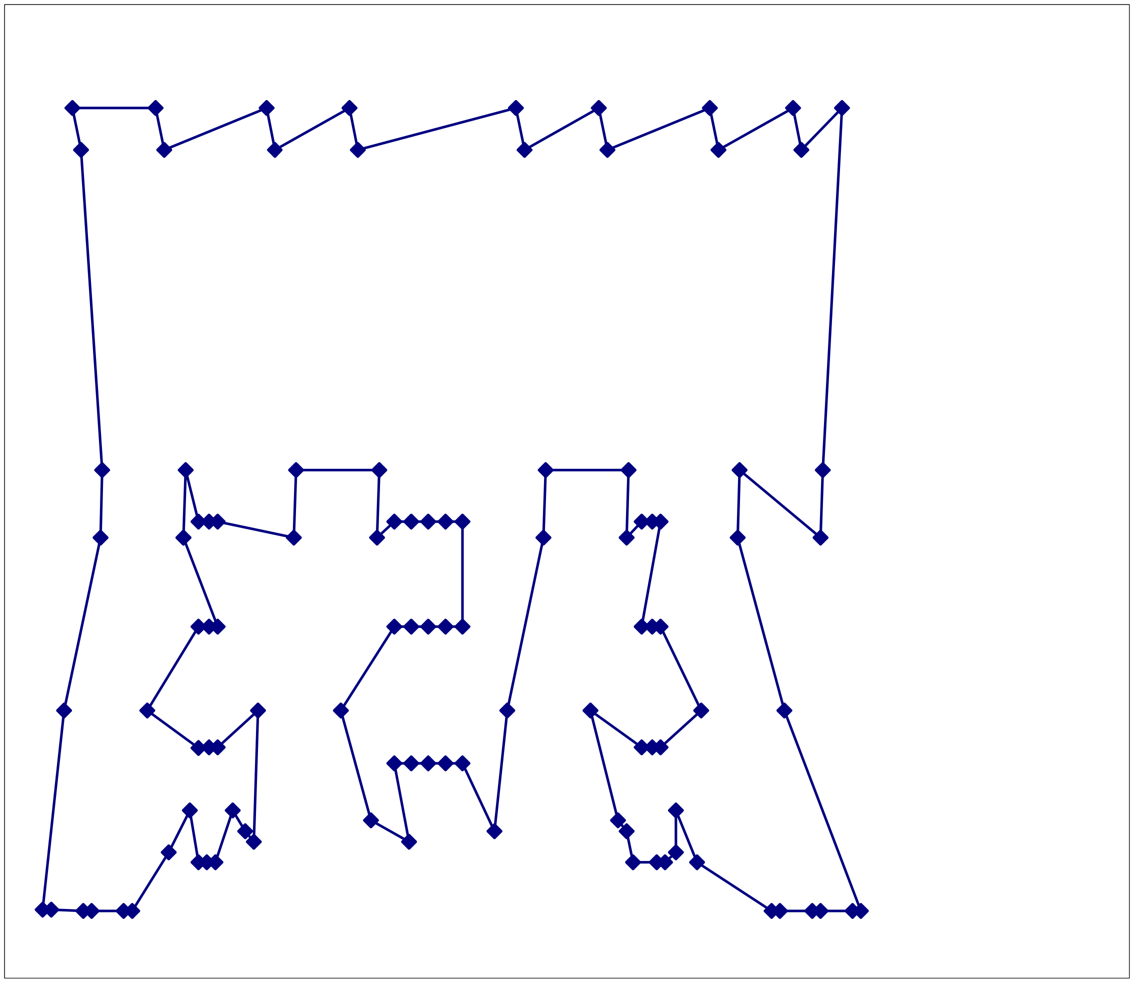


Figure 6.27 lin105

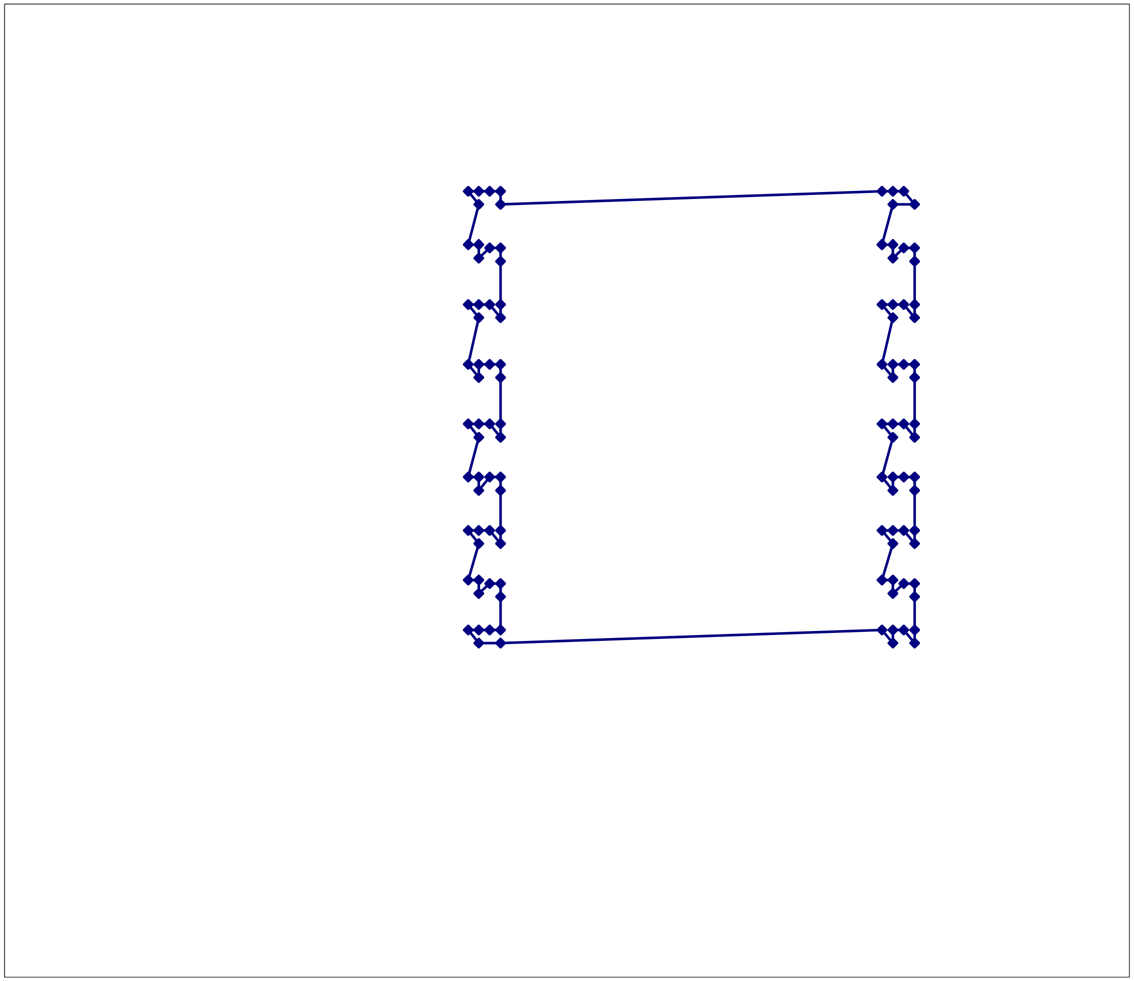


Figure 6.28 pr107

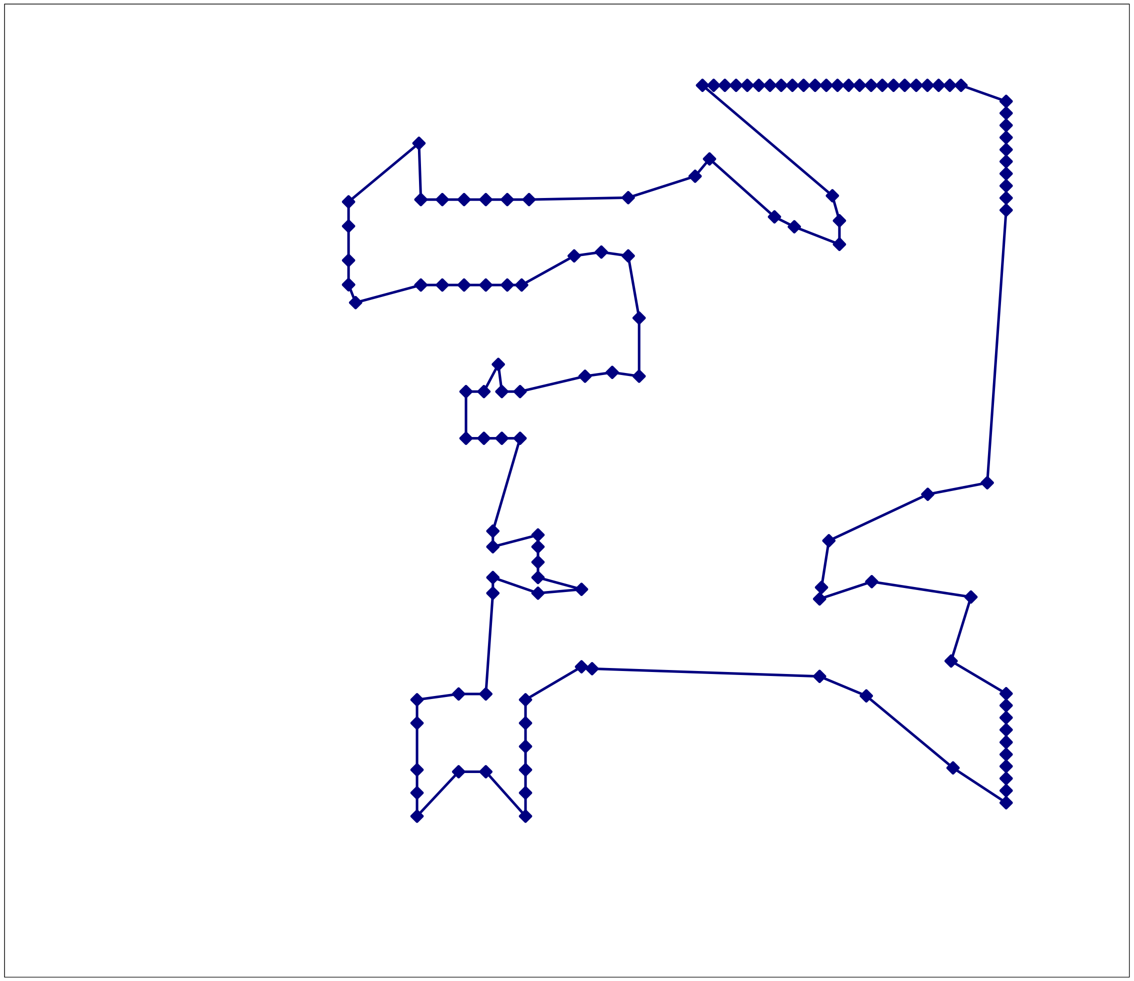


Figure 6.29 pr124

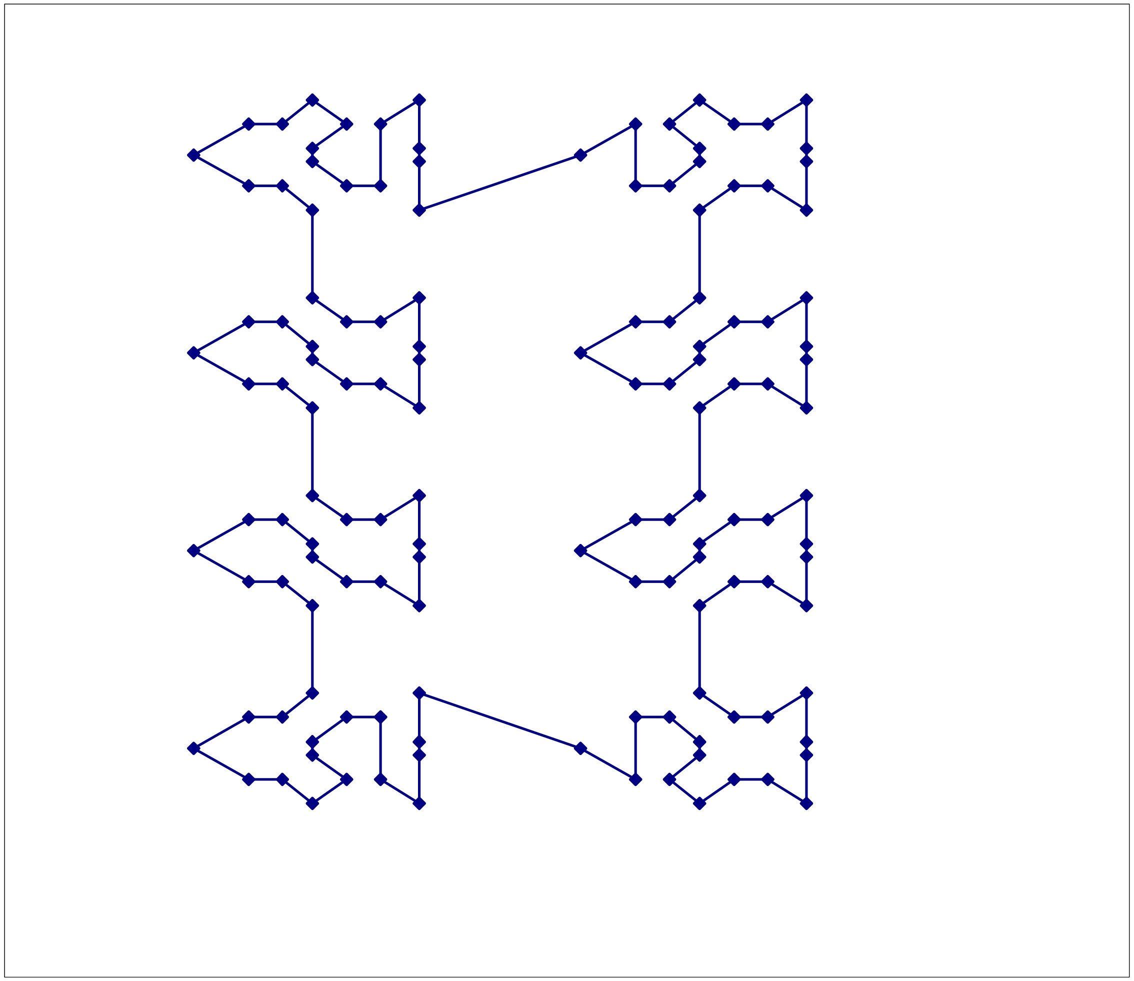


Figure 6.30 pr136

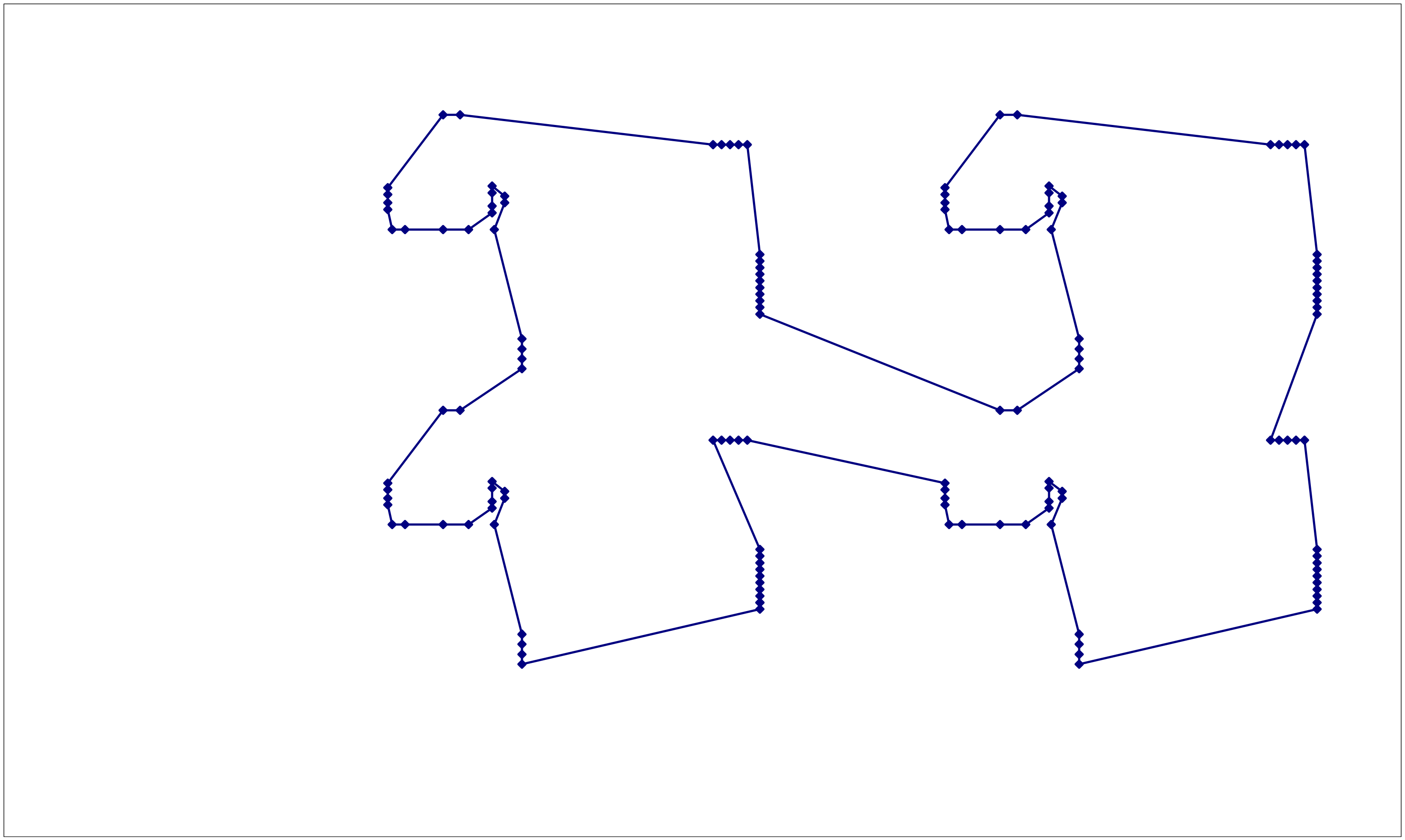


Figure 6.31 pr144

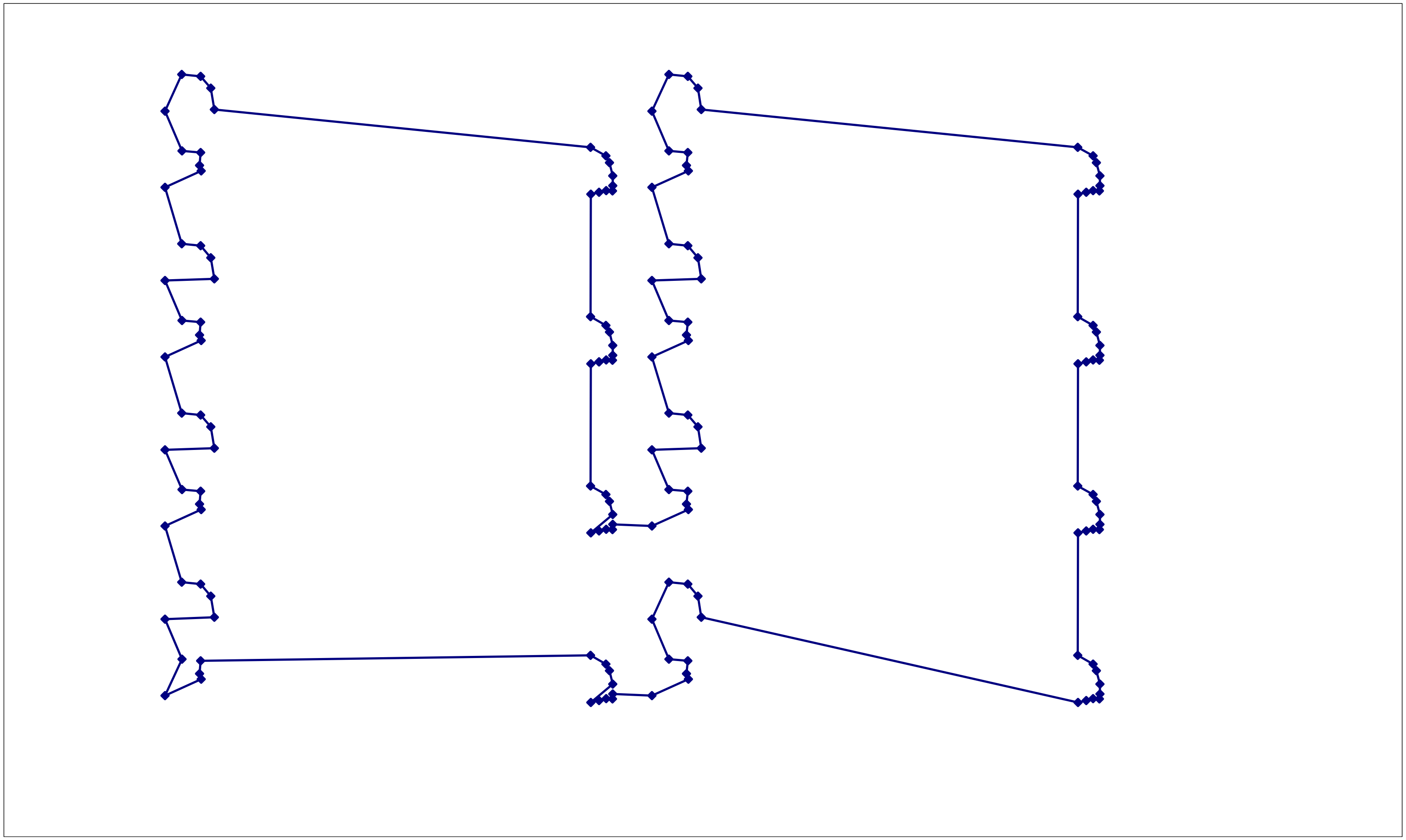


Figure 6.32 pr152

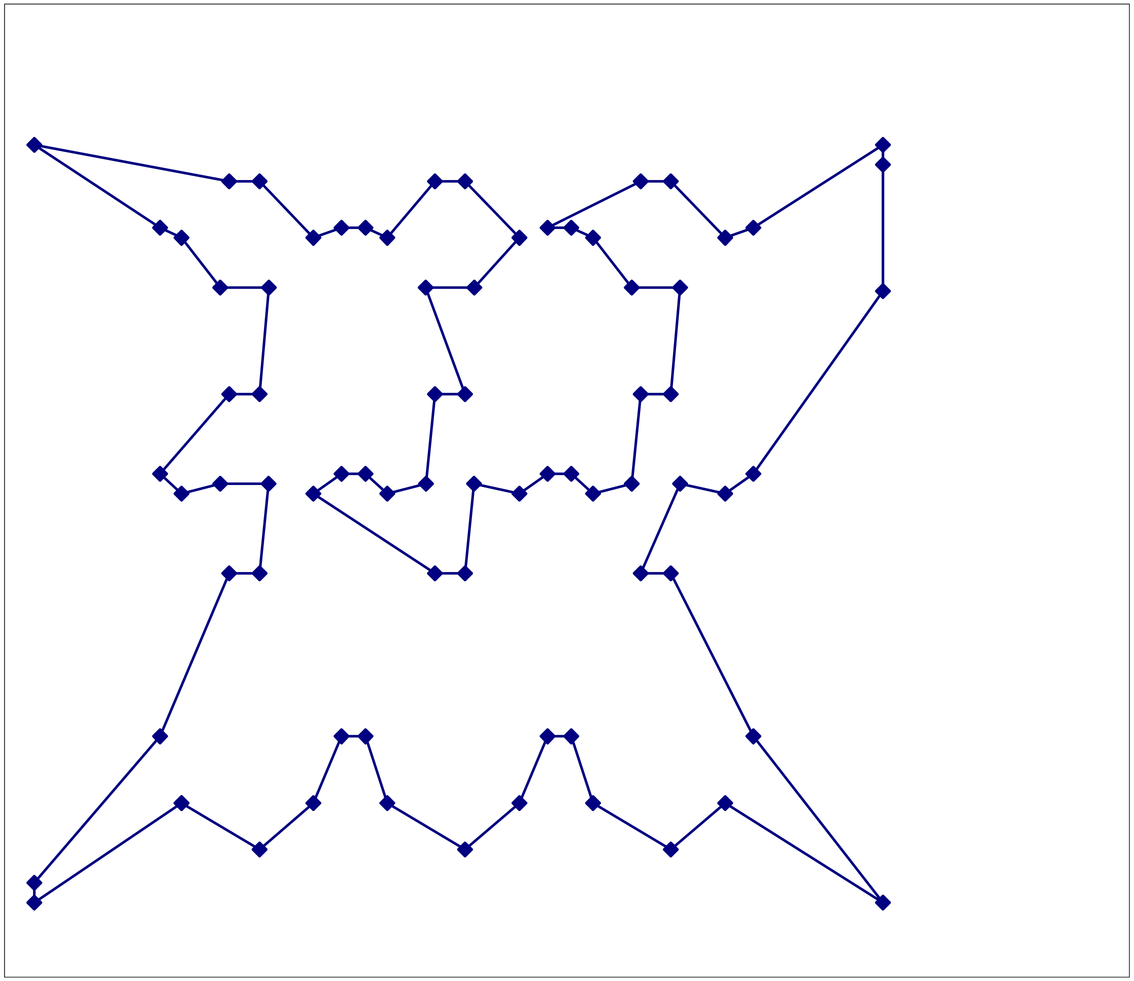


Figure 6.33 pr76

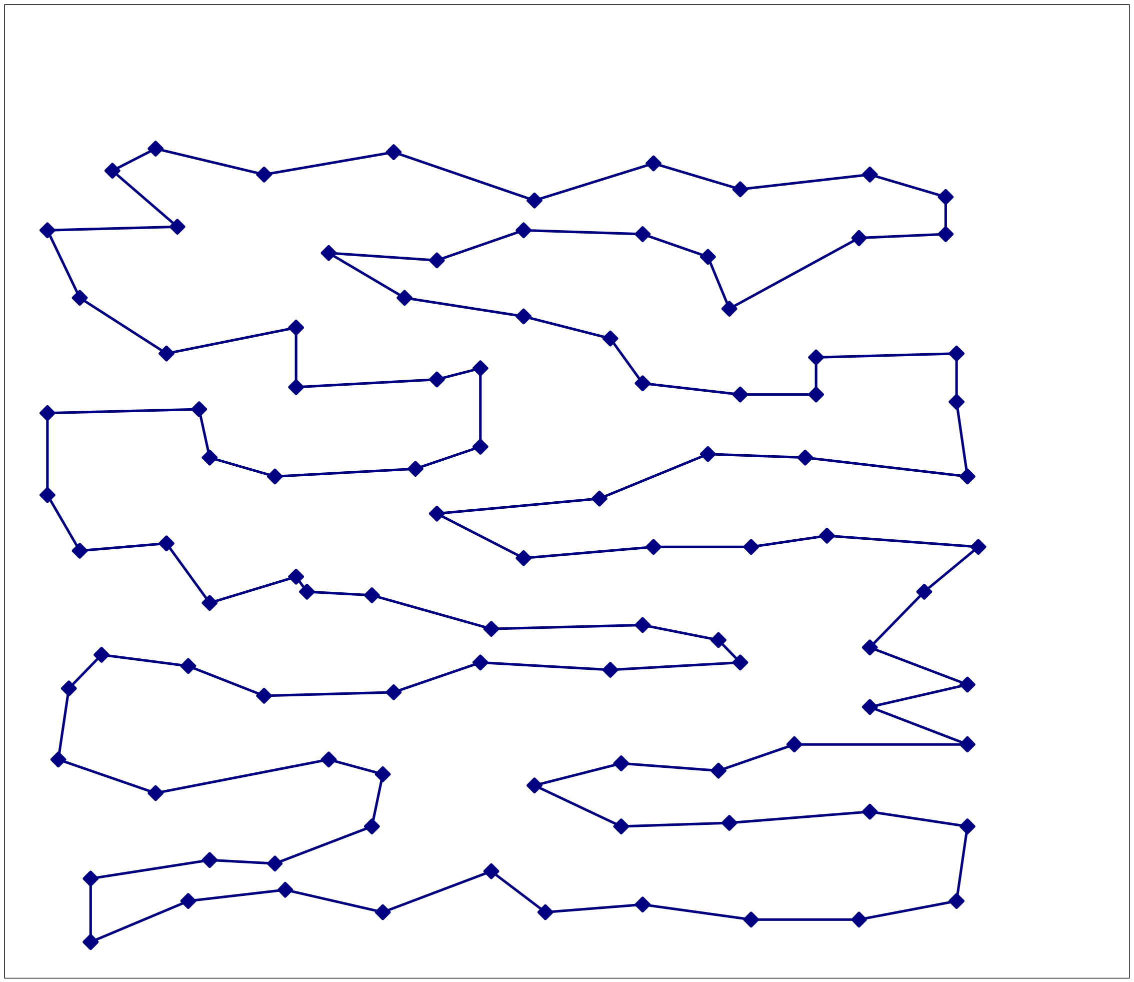


Figure 6.34 rat99

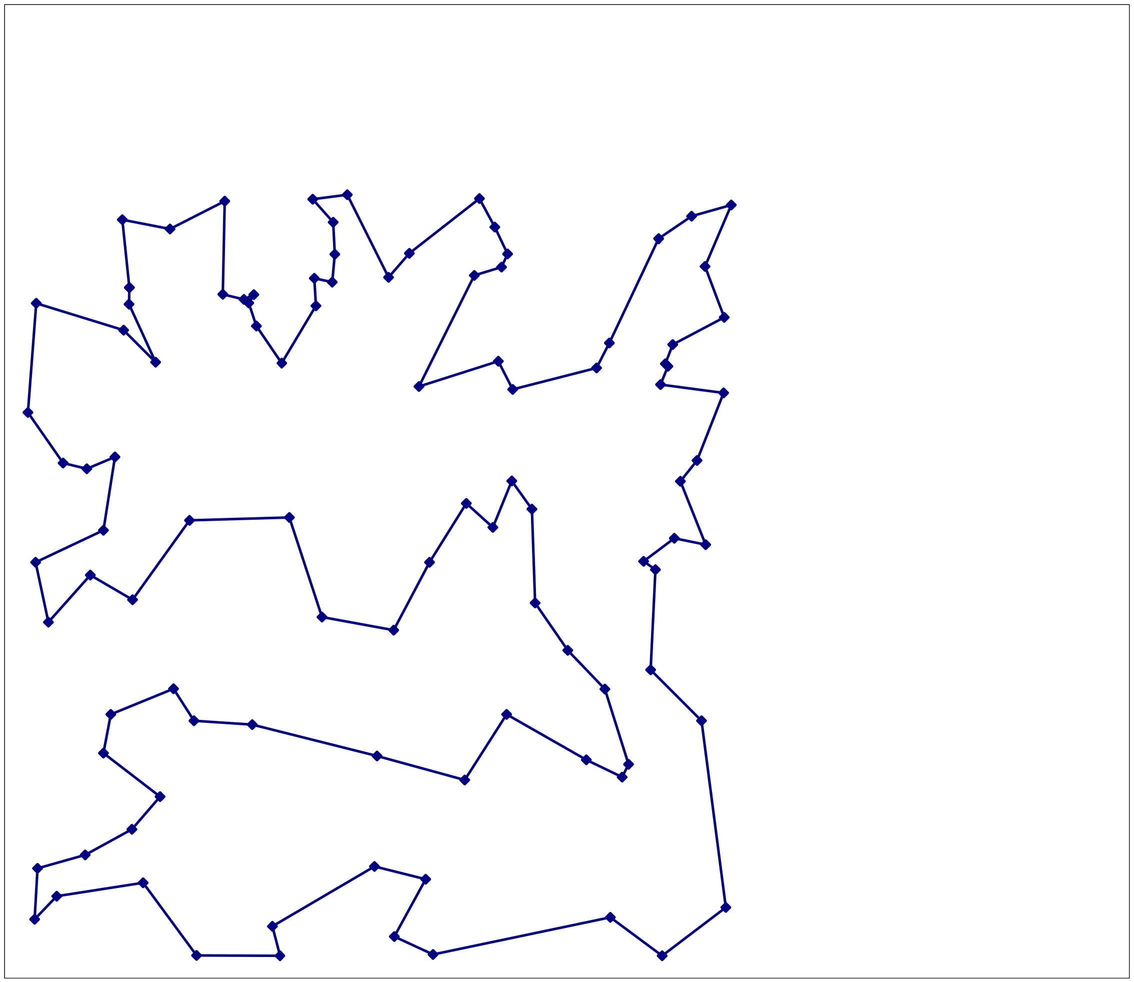


Figure 6.35 rd100

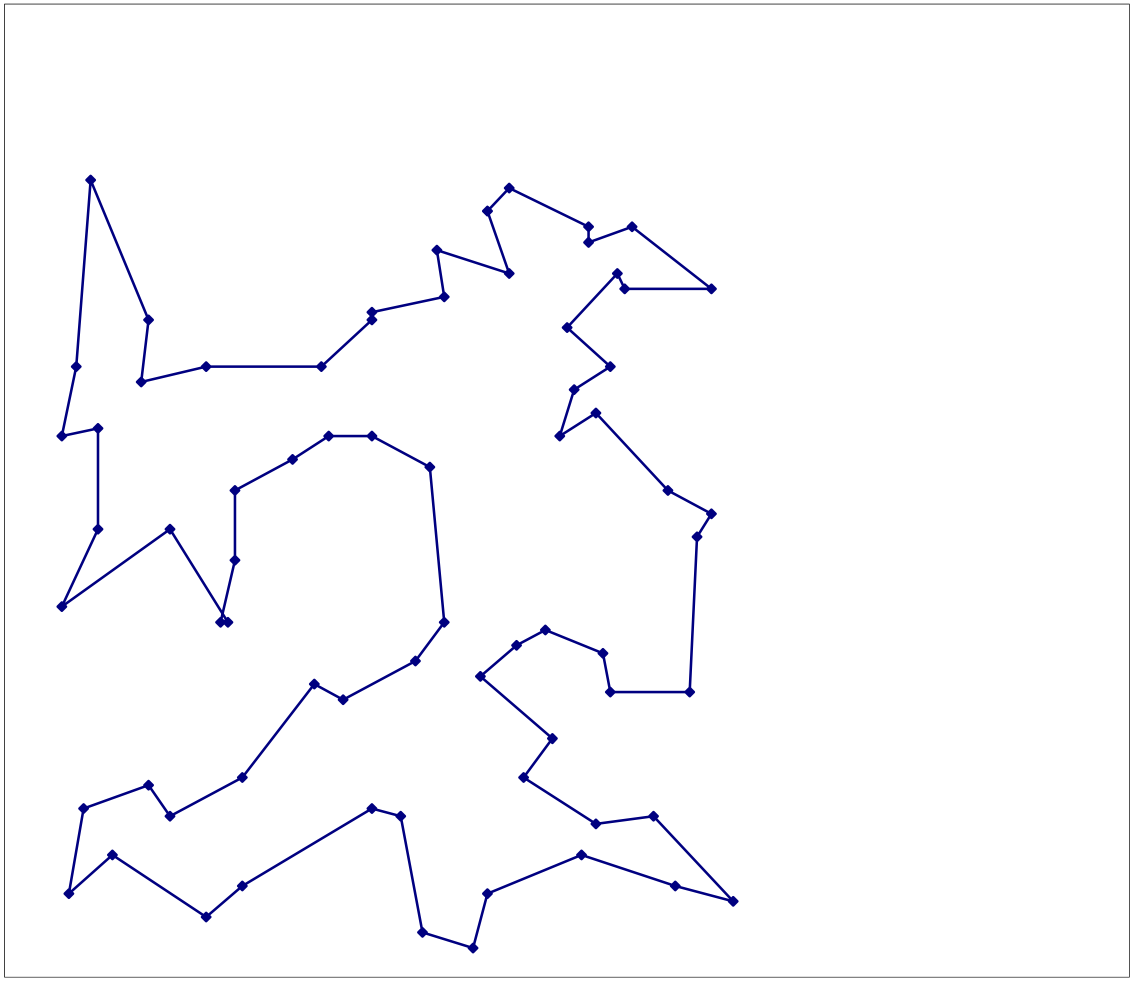


Figure 6.36 st70

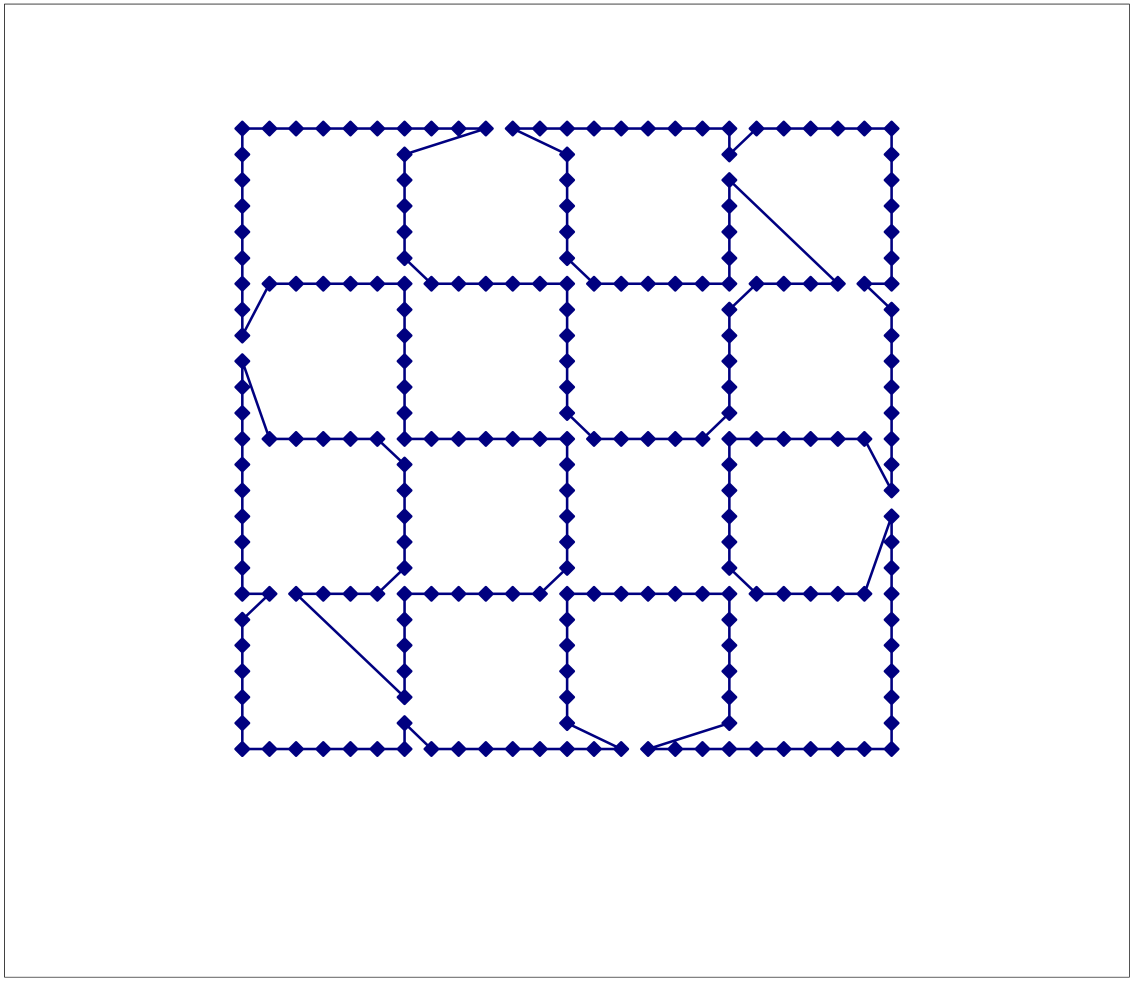


Figure 6.37 ts225

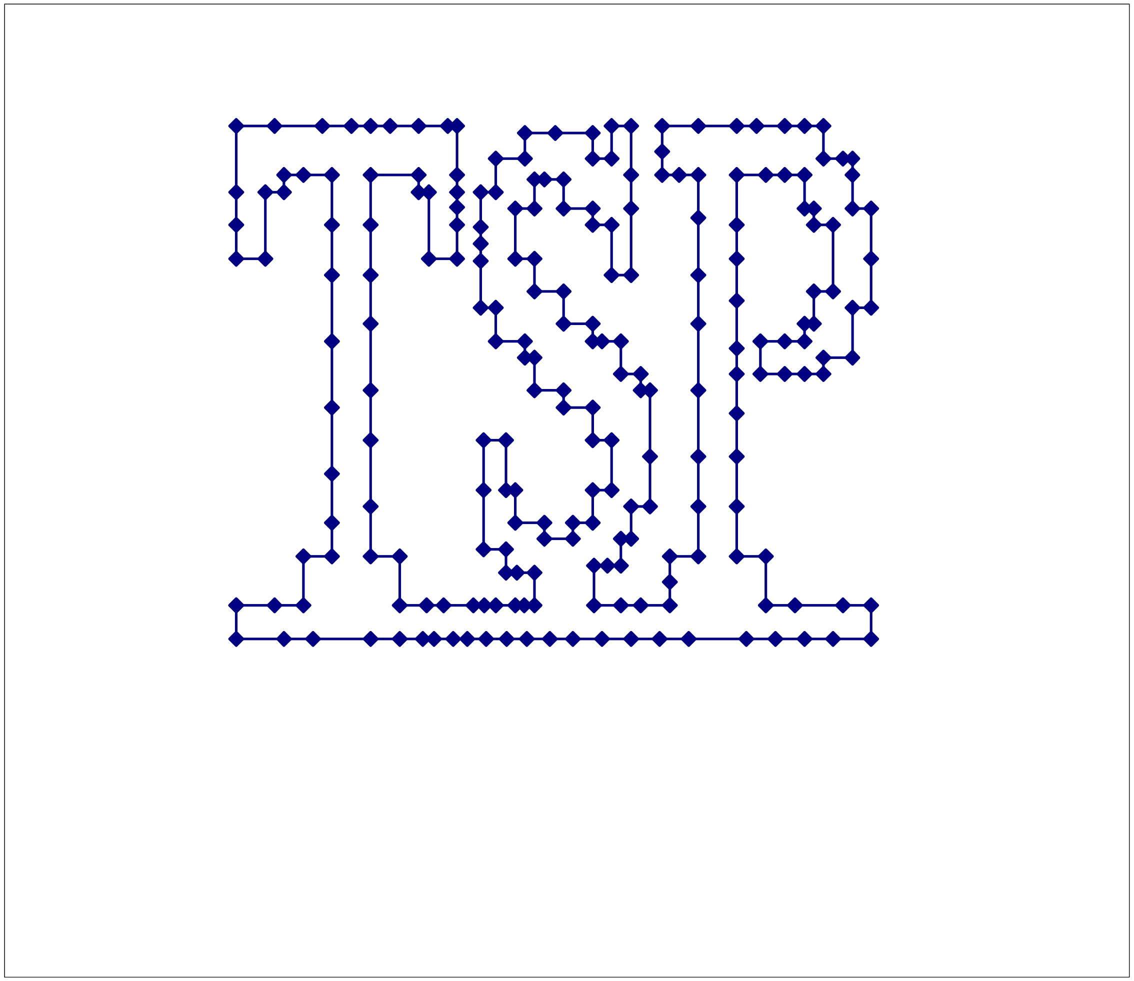


Figure 6.38 tsp225

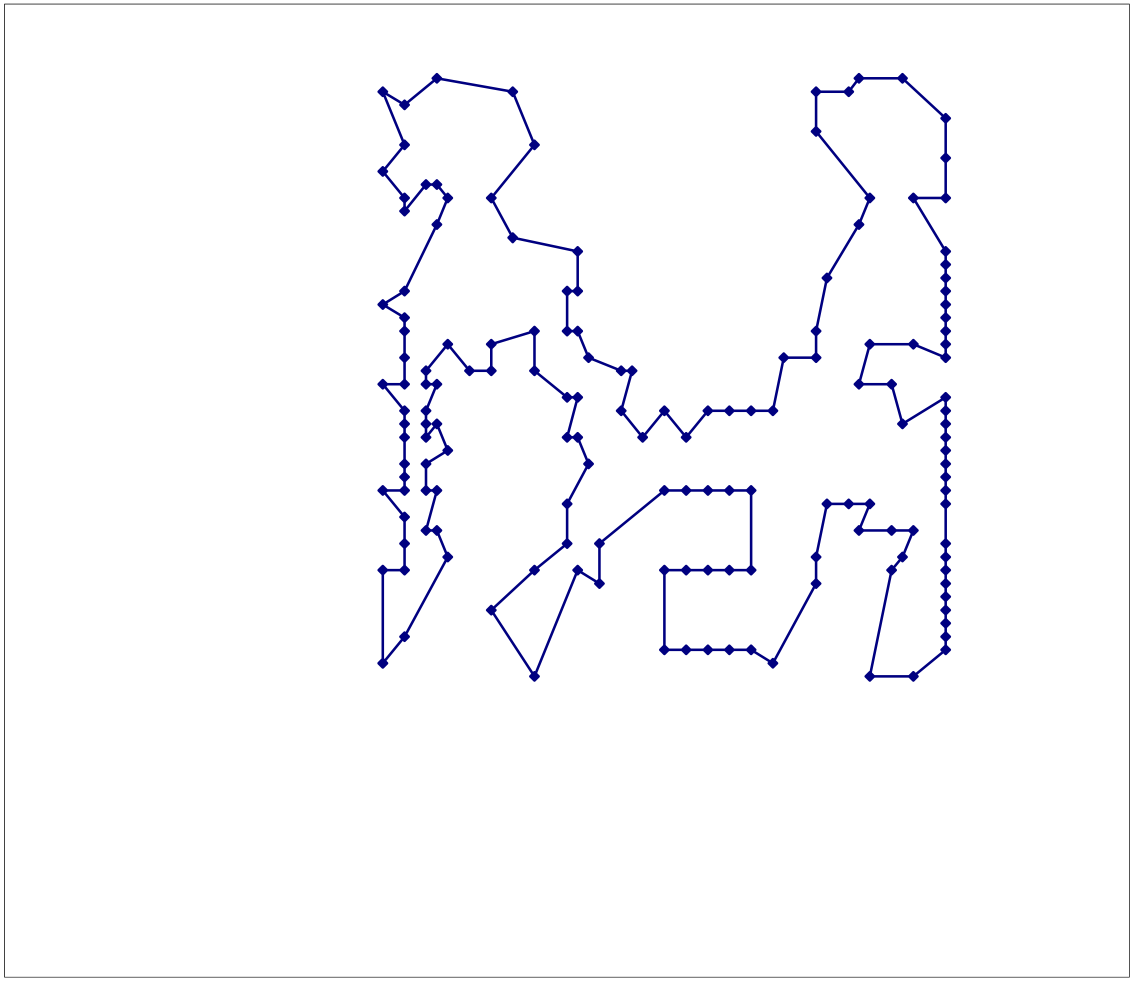


Figure 6.39 u159

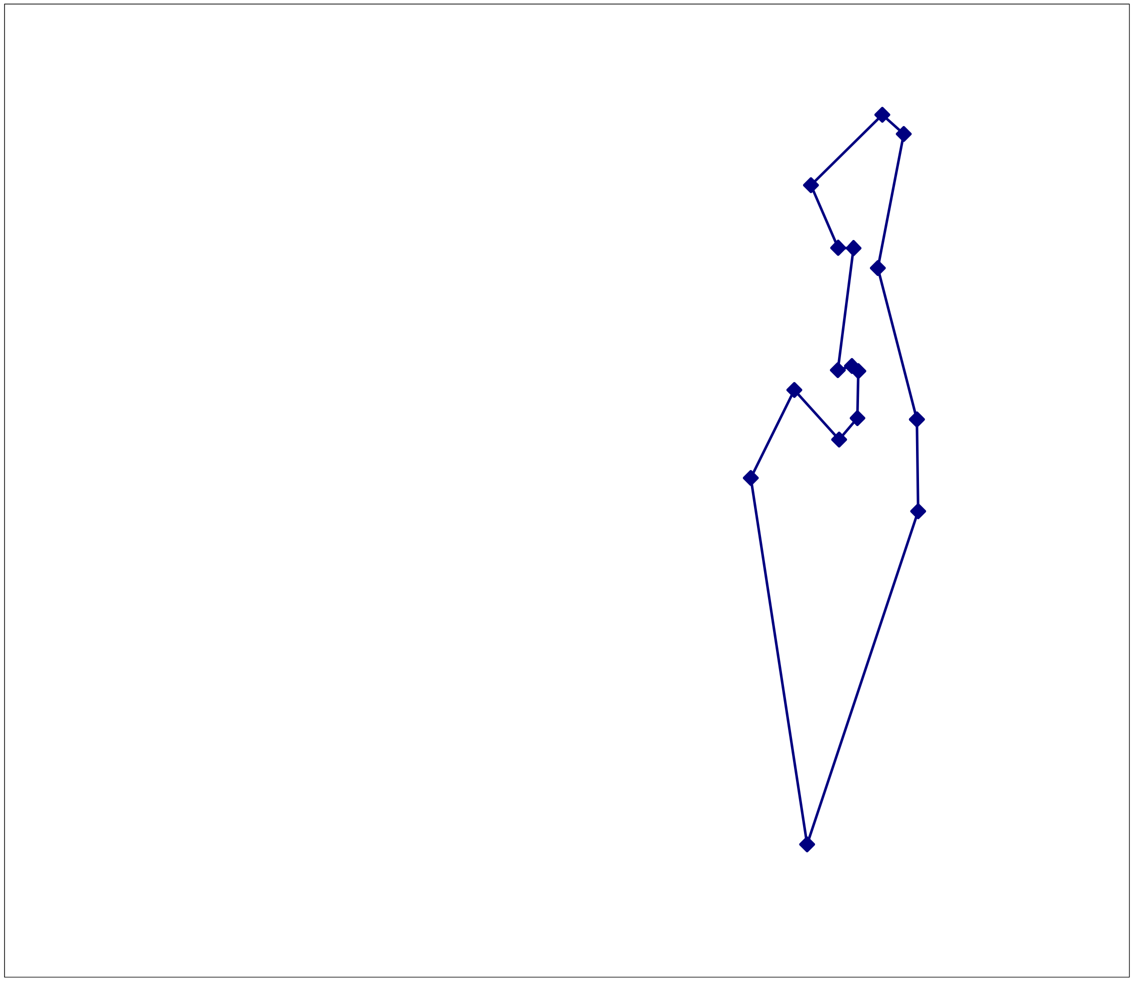


Figure 6.40 ulysses16

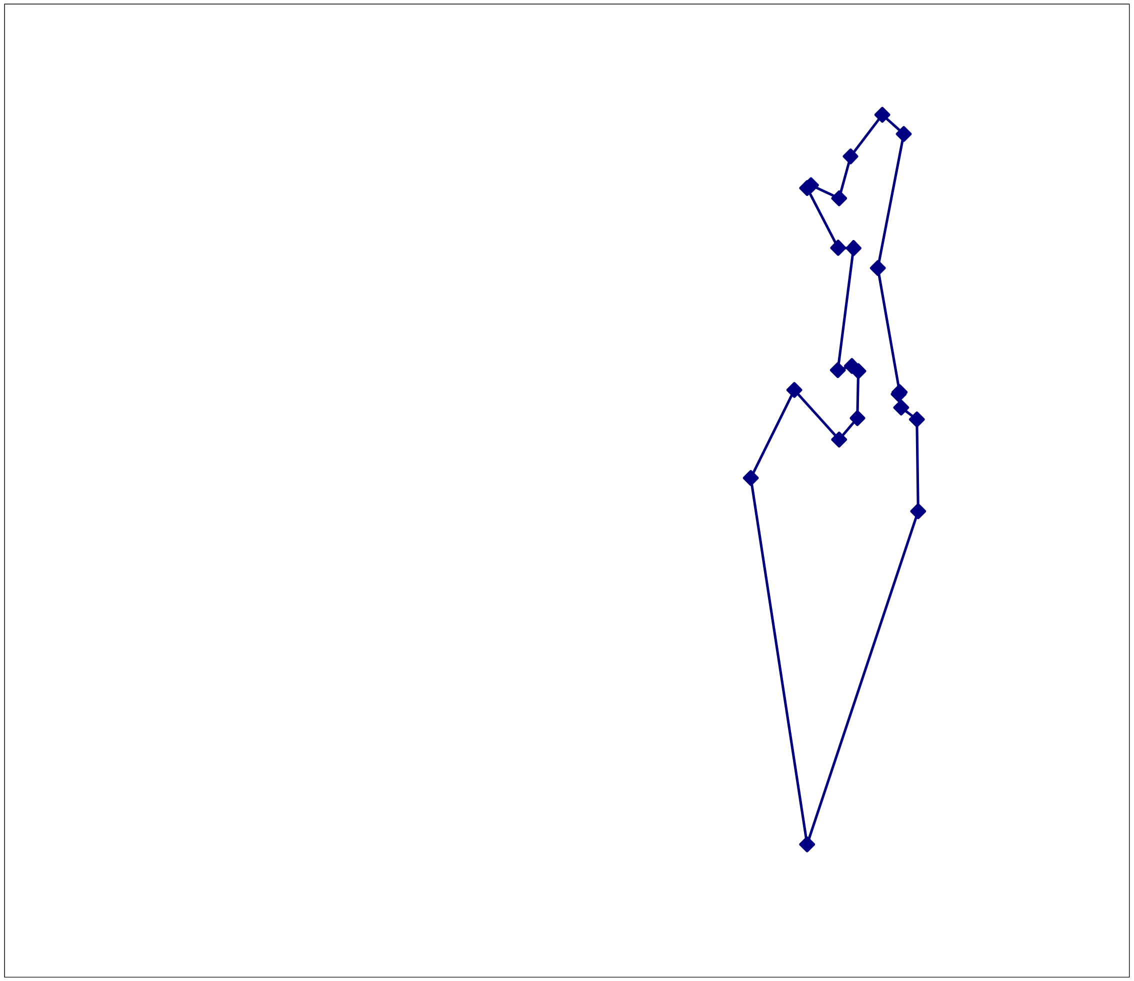


Figure 6.41 ulysses22