ANALYSIS OF EVOLUTIONARY ALGORITHMS FOR CONSTRAINED ROUTING PROBLEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ERDEM DEMİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF

MASTER OF SCIENCE

IN

INDUSTRIAL ENGINEERING

JUNE 2004

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Çağlar Güven
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science

Asst. Prof. Dr. Haldun Süral
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Nur Evin Özdemirel	(METU - IE)	
Prof. Dr. Murat Köksalan	(METU - IE)	
Assoc. Prof. Dr. Canan Sepil	(METU - IE)	
Assoc. Prof. Dr. Selim Aktürk	(Bilkent Uni. - IE)	
Asst. Prof. Dr. Haldun Süral	(METU - IE)	

ABSTRACT

ANALYSIS OF EVOLUTIONARY ALGORITHMS FOR CONSTRAINED ROUTING PROBLEMS

Demir, Erdem
M.S., Industrial Engineering
Supervisor : Asst. Prof. Dr. Haldun Süral

June 2004, 149 pages

This study focuses on two types of routing problems based on standard Traveling Salesman Problem, which are TSP with pickup and delivery (TSPPD) and TSP with backhauls (TSPB). In both of these problems, there are two types of customers, i.e. “delivery customers” demanding goods from depot and “pickup customers” sending goods to depot. The objective is to minimize the cost of the tour that visits every customer once without violating the side constraints. In TSPB, delivery customers should precede the pickup customers, whereas the vehicle capacity should not be exceeded in TSPPD.

The aim of the study is to propose good Evolutionary Algorithms (EA) for these two problems and also analyze the adaptability of an EA, originally designed for the standard TSP, to the problems with side constraints. This effort includes commenting on the importance of feasibility of the solutions in the population with respect to these side constraints. Having this in mind, different EA strategies involving feasible or infeasible solutions are designed. These strategies are compared by quantitative experiments realized over a set of problem instances and the results are given.

Keywords: TSP with Pickup and Delivery, TSP with Backhauls, Evolutionary algorithms, Heuristics.

ÖZ

EVRİMSEL ALGORİTMALARIN YAN KISITLI ROTALAMA PROBLEMLERİNDE İNCELENMESİ

Demir, Erdem
Yüksek Lisans., Industrial Engineering
Tez Yöneticisi: Y. Doç. Dr. Haldun Süral

Haziran 2004, 149 sayfa

Bu çalışma Gezgin Satıcı Probleminin iki yan kısıtlı hali üzerinde yoğunlaşır. Bu problemler, Dağıtım ve Toplamalı Güzergah Bulma Problemi (DTGBP) ve Geri Yüklemeli Gezgin Satıcı Problemi (GYGSP)'dir. Problemlerde iki çeşit müşteri vardır: ana depodan ürün talep eden “dağıtım müşterileri” ve ana depoya ürün göndermek isteyen “toplama müşterileri”. Problemlerin amacı yan kısıtları sağlayan en az maliyetli turu bulmaktır. Uyulması gereken kısıtlar, birinci problemde araç kapasitesi, ikinci problemde ise sıralama kısıtıdır.

Bu çalışmanın amacı, DTGBP ve GYGSP için iyi evrimsel algoritmalar (EA) geliştirmenin yanı sıra, kısıtsız problem için iyi işleyen bir EA'nın kıstlı problemlere uyarlanmasının incelenmesidir. Algoritma için toplumdaki bireylerin yan kısıtlara göre olurluğunun önemi üzerine yorum yapmak esastır. Bu bakış açısıyla, olurlu ve olursuz bireylerle çalışan değişik EA'lar önerdik. Bunlar bilgisiyar ortamında yapılan deneylerle karşılaştırıldı. Sonuçta önerilen EA'ların iyi işlediği görüldü.

Anahtar Kelimeler: Dağıtım ve Toplama Güzergahı Bulma Problemi (DTGBP), Geri Toplamalı Gezgin Satıcı Problemi (GTGSP), Evrimsel Algoritmalar, Sezgisel Yöntemler

To Mom and Dad

ACKNOWLEDGEMENTS

Firstly, I would like to express my gratitude to Asst. Prof. Dr. Haldun Süral for his patient guidance and understanding. I feel myself lucky to have the chance to work with such a great researcher and a great person. I must add that this work would not have been such an enjoyable experience without him.

I would like to thank Assoc. Prof. Nur Evin Özdemirel for her help and guidance in the initial phases of the study. I should also include Prof. Dr. Murat Köksalan and Assoc. Prof. Dr. Yasemin Serin in the list for their kind and valuable comments on statistical analyses realized in the study.

My family is always there for me. It is never enough no matter how much I thank them. Anyway I will repeat once more. I would like to thank my dearest mom, Cevriye Demir, my dad, İsmail Demir, and my brother, Umut Demir. Their love and support were with me throughout the study.

I would like to thank all my friends. Special thanks go to Tolga Işıkyıldız for his existence, for his understanding, support, and care, and Koray Başar for his valuable company. It is very hard to write what I feel about you guys.

I felt the warm support of my old pals Müge Çavdaroğlu and Özge Şahin all the time even if they were miles away and I thank them here for what they add to this study and to my personality as well.

I think the value of this educational experience remains faint compared to what it brought on the friendship basis. I would like to thank Ertuğrul Yurdakul, Gökçe İnce, Burcu Yalım, Elvin Karana for what they brought to my life.

In my hardest times METU IE assistants were always near. I would like to thank all of them especially my roommates Kevser Öztürk, Burcu Balçık, and Selin Bilgin.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Due to very large costs associated with physical distribution in logistics systems, the routing problems have been attracting attention of transportation scientists as well as operational researchers. In past 30 years, many papers have been devoted to introducing and defining various routing problems, which have different objectives and constraints, and to developing efficient solution techniques that can be implemented in real life easily. Bodin and Golden (1981) summarize benefits of designing and managing the routing systems efficiently and point on significant savings that can be achieved.

The distribution systems cannot simply be considered as a delivery system. Generally, distribution of goods involves also collection of some other related goods such as empty bottles for a brewery distribution company. Further efficiency can be achieved by integrating these delivery and pickup activities in the same routes. In literature, this option is commonly called "backhauling". The term backhaul is also used interchangeably by "pickup and delivery" in some references. Bodin et al. (1983) discuss the importance of delivering and picking up in the same routes, providing its real life applications. Goetschalckx and Jacobs-Blecha (1986) referring to the report of Kearney, point on the large annual distribution costs for US, and report potential savings that can be achieved by integrating pickups and deliveries. They provide an example from the grocery store industry, where the saving due to this integration is $165 million.

In this study, we have narrowed our attention to the single vehicle routing problems with pickup and delivery, where only one route is to be designed. Single vehicle problems constitute a basis for multi-vehicle cases, which are harder to deal with. Specifically, we have selected the traveling salesman problem with pickup and delivery (TSPPD) and the traveling salesman problem with backhauls (TSPB). In both problems, a single vehicle visits two customer sets, namely, delivery and pickup customers, and it satisfies the demands of all customers. The vehicle capacity is the main constraint in TSPPD. When there is no free capacity on the vehicle, it cannot visit a pickup customer to collect the goods. The two problems differ from each other in the sense that in TSPB, pickup customers can only be visited after visiting all delivery customers. These precedence relations among customers are the main constraints in TSPB.

The traveling salesman problem (TSP), where a set of customers is to be visited by a single vehicle, constitutes a natural basis for these problems. We can refer to this problem as the unconstrained case of our problems, since the vehicle capacity constraint and the precedence constraint are relaxed and the differentiation between customer sets is removed. TSP is known to be NP-hard. Since the problems under consideration are generalization of it, they are also NP-hard, which implies that they will resist, like TSP, all efforts to find a good optimization algorithm.

Exact solution algorithms for TSP can solve instances with size up to 3000 customers. However, Rego and Glover (2002) point on the impractical times required for solving instances of size larger than 1000 at optimality. Even for modest size problems, exact methods require substantially greater computation time than leading heuristic methods. Heuristics are capable of finding optimal or very-close to-optimal solutions for instances larger than those reasonably attempted by exact methods. Even when the optimal solutions are sought for real life problems despite the large time requirements, the implementation requires extensive OR and computing expertise, which consolidate the impracticality of these methods. When side constraints accompany the standard TSP, like in backhauling case, heuristics gain importance. The exact methods attempting to solve these constrained cases may not deal with very large instances at all.

Among the approximation techniques, metaheuristics are gaining popularity. In both Computer Science and Operations Research literatures, there are many successful metaheuristic applications for solving TSP. In the second domain, metaheuristics were also used for the constrained cases of this problem. These modern methods can provide solutions where the conventional heuristics cannot find, but they require more time. Specifically, Evolutionary Algorithms (EA), which incorporate continuous improvement of a population of solutions, proved to give successful results for TSP. However, the literature lacks EA applications for TSP with side constraints.

Actually, applying EAs to the problems with side constraints is not a trivial task regarding feasibility. Even for TSP, such algorithms should be modified in order to eliminate subtours. By the help of problem specific encoding and reproduction schemes, this problem has been overcame at the expense of the original binary structure and the initial "Genetic Algorithms" name. For the constrained cases, the feasibility with respect to these additional constraints makes the problem harder. In the literature, several constraint handling techniques are reported. Unfortunately, to our knowledge, no work on comparing these techniques for generalizations of TSP exists.

In this regard, we tried to analyze and compare these different constraint handling techniques in the domain of single vehicle routing problems with pickup and deliveries. Initially, we select an EA that has been proved to work well for the unconstrained case, i.e., the "naked" TSP. To deal with additional constraints (i.e., the capacity constraint for TSPPD and the precedence constraint for TSPB), we implement several different constraint handling techniques. Our techniques can be grouped into two. The first group works with feasible solutions only: specifically, simply rejecting infeasibility, modifying the crossover for ensuring feasibility, and repairing the infeasible solutions. In the second group infeasible solutions are permitted to exist in the population, however, the chance for them to pass their genetic code to a new solution is reduced. This later technique is known as penalizing infeasible solutions.

Although we specifically work on TSPPD and TSPB, the work aims at providing a general insight for the performance of the specific constraint handling techniques on the routing problems with side constraints. In the solution procedures proposed, we simply try to find good solutions for the constrained cases while realizing the search in the solution space of the unconstrained problem. In this respect, these two problems differ in hardness to find feasible solutions during the search. The solution space of TSPB is a subset of the solution space of TSPPD, and therefore, feasibility is harder to maintain for TSPB. Problem focusing on problems differing in "hardness" of the associated constraints provides a better insight for generalization. Nevertheless, we also want to propose good EAs for these specific problems at the end.

In this study, we basically adapt an EA that works well for TSP to solve the constrained TSP problems. In this adaptation, five versions of the algorithm by utilizing the constraint handling techniques mentioned above are proposed. In the first version, the algorithm keeps producing new solutions until a feasible child is produced. The infeasible solutions are simply discarded. The second version uses a modified crossover operator, which produces solutions that are feasible with respect to side constraints. The third one repairs any infeasible solution produced. Fourth and fifth versions permit infeasible solutions to enter the population, however, penalize their fitness values with different penalizing schemes. In order to measure the performance of these strategies, we conduct computational experiments on test beds taken from the literature. Promising results are obtained for some of the algorithms.

This study is organized as follows. In Chapter 2, our specific problems, namely, TSPPD and TSPB, are defined and their related literature is reviewed. Chapter 3 presents the EA algorithms proposed. Firstly, the features of the general algorithm are explained and then the specific strategies differing in utilized constraint handling techniques are explained. The experimental results of these algorithms for TSPPD and TSPB are provided in Chapter 4. The study is finalized by the conclusive remarks in Chapter 5.

CHAPTER 2

LITERATURE SURVEY

The traveling salesman problem (TSP) is a well known combinatorial optimization problem, where **n** customers are to be visited on a tour of a single vehicle. The tour should start at the depot. After visiting all of the customers once it should end at the depot. The objective is to minimize the length of this tour. The problem can be defined on an undirected complete graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where **V** represents the nodes located at the customer points and the depot, and **E** represents the edges between the nodes. For every edge $\{\mathbf{i},\mathbf{j}\} \in \mathbf{E}$, there is a cost $\mathbf{c_{ij}}$ associated with it.

This chapter, focusing on the constrained cases of TSP, provides a literature survey of related problems and solution approaches. Specifically, the problems studied are the traveling salesman problem with pickup and delivery (TSPPD) and the traveling salesman problem with backhauls (TSPB). The main difference between them and TSP is that there are two sets of customers in these variants: **D**, the set of delivery customers, and **P**, the set of pickup customers.

In the following sections, the relation of TSPPD and TSPB with the other constrained routing problems is discussed. The examples of applications are provided from industry. The chapter also includes the solution approaches proposed in the literature. The chapter is finalized by a brief overview of metaheuristic applications for routing problems.

2.1 The Traveling Salesman Problem with Pickup and Delivery

In the traveling salesman problem with pickup and delivery, each delivery customer demands $\mathbf{d_i}$ $(\mathbf{i} \in \mathbf{D})$ units of load from the central depot; whereas the pickup customers are required to send $\mathbf{p_j}$ $(\mathbf{j} \in \mathbf{P})$ units of load to the depot. In this problem, a single vehicle with a capacity **Q** has to visit every customer once while satisfying the requirements. The main difference from TSP is due to the additional constraint on vehicle capacity that can not be exceeded. For feasibility, **Q** should be greater than or equal to the maximum of total delivery load and total pickup load; otherwise, the problem is immediately infeasible.

In the literature, there are two different environments referring to this problem (Nagy and Salhi 2004). In the first one, each customer is either a delivery customer or pickup customer (i.e., *mixed* version). In the other environment, at each customer the vehicle should leave some amount of load and pick some other amount at the same visit (i.e., *simultaneous* version). In fact, the second environment can be treated as the first one by computing the net demand $\mathbf{t_i}$ of the customer **i** as the difference between the quantity to be picked up from and the quantity to be delivered to. If the net demand of a customer is negative, then it can be regarded as a delivery customer, otherwise the customer turns out to be a pickup customer. Note that a customer with "0" net demand does not influence the vehicle capacity, but still has to be included in the tour.

The problem gets harder to solve when the total amount of delivery loads equals to the total amount of pickup loads and to vehicle capacity. In this case the vehicle starts the tour with full delivery load, visits every customer, and returns to the depot with full of pickup loads. In our study, we focused on this setting of the problem.

Süral and Bookbinder (2003) proposed a mathematical formulation for the single vehicle routing problems with backhauls. This formulation can be easily modified for TSPPD. Let **n** be the total number of customers, $\mathbf{n} = |\mathbf{D}| + |\mathbf{P}|$. Let **TD** and **TP** denote the total delivery load and the total pickup load, respectively.

Let $\mathbf{x_{ij} = 1}$ if the customer **i** immediately precedes customer $\mathbf{j}$ $(\mathbf{i \neq j})$; 0 otherwise,
$\mathbf{y_i}$ = total load on the vehicle to be delivered after serving customer **i**,
$\mathbf{z_i}$ = total load picked-up just before serving customer **i**,

Minimize $\sum_{i}\sum_{j}\mathbf{c_{ij}x_{ij}}$ (2.1)

st.

$$\sum_{\mathbf{i}=0}^{\mathbf{n}} \mathbf{x_{ij}} = 1, \qquad \forall \mathbf{j} \qquad (2.2)$$

$$\sum_{\mathbf{j}} \mathbf{x_{ij}} - \sum_{\mathbf{k}} \mathbf{x_{ki}} = 0, \qquad \forall \mathbf{i} \qquad (2.3)$$

$$\mathbf{y_j - y_i + TDx_{ij} \le TD - d_j} \qquad \forall \mathbf{i}, \mathbf{j}(\mathbf{i}, \mathbf{j} \ne 0) \qquad (2.4)$$

$$\mathbf{z_i - z_j + TPx_{ij} \le TP - p_i} \qquad \forall \mathbf{i}, \mathbf{j}(\mathbf{i}, \mathbf{j} \ne 0) \qquad (2.5)$$

$$\mathbf{y_i + z_i \le Q - d_i - p_i} \qquad \forall \mathbf{i}(\mathbf{i} \ne 0) \qquad (2.6)$$

$$\mathbf{y_i}, \mathbf{z_i} \ge 0; \mathbf{x_{ij}} = 0 \text{ or } 1 \qquad \forall \mathbf{i}, \mathbf{j} \qquad (2.7)$$

The objective function (2.1) gives the total cost of traversed edges. The constraints (2.2) and (2.3) are the assignment constraints. (2.4) and (2.5) are adaptations of the Miller Tucker Zemlin subtour elimination constraints for TSP. The first one updates the delivery load on the vehicle whereas the second updates the pickup load. Constraints (2.6) restrict the solution to be feasible with respect to vehicle capacity. Constraints (2.7) are the nonnegativity and integrality constraints.

2.2 The Traveling Salesman Problem with Backhauls

The traveling salesman problem with backhauls resembles TSPPD in customer differentiation aspect. However, there is an additional precedence constraint, which forces any pickup customer to be visited only after all of the delivery customers are visited. Here, the vehicle capacity is not a restricting factor as long as the total amount of load to be delivered and picked up does not exceed vehicle capacity.

The formulation given for TSPPD can be used for TSPB also. However, an additional constraint should be introduced into the formulation, which rejects any tour including an edge from a pickup node to a delivery node. Therefore, the model should include (2.1) - (2.5), (2.7), and the following additional constraint.

$$\mathbf{x}_{ij} = 0 \qquad \forall \mathbf{i}, \mathbf{j}(\mathbf{i} \in \mathbf{P}, \mathbf{j} \in \mathbf{D}) \qquad (2.8)$$

TSPB can also be formulated as an asymmetric TSP by updating the cost matrix (Gendreau et al. 1997). When the costs of edges directed from any pickup node to any delivery node are penalized by adding a sufficiently large number to the initial costs of these edges, the optimal TSP tour will surely be free from these unattractive edges, and will make the resulting tour feasible with respect to precedence constraints.

To give an example, the following cost matrix is provided in Table 2.1 for a TSPB instance. Node 0 represents the depot. Nodes 1, 2, and 3 stand for the delivery customers while nodes 4 and 5 stand for the pickup customers. Table 2.2 shows the updated cost matrix where entries indicate that the edges directed from pickups to deliveries are penalized by a constant M. M should be large enough in order to make any solution violating "the precedence constraint" worse than the optimal solution to TSPB. An immediate value for M is the sum of row maximums of the matrix. A looser value for M can be defined as the sum of all nonnegative edge costs.

Table 2.1 The original cost matrix for a TSPB instance

	0	1	2	3	4	5
0	-	12	6	4	3	5
1	9	-	8	7	12	4
2	16	7	-	3	13	8
3	12	3	11	-	9	12
4	11	8	4	9	-	11
5	2	13	17	6	6	-

Table 2.2 The updated cost matrix for the TSPB instance

	0	1	2	3	4	5
0	-	12	6	4	3	5
1	9	-	8	7	12	4
2	16	7	-	3	13	8
3	12	3	11	-	9	12
4	11	8+M	4+M	9+M	-	11

5	2	13+M	17+M	6+M	6	-

If the updated cost matrix is solved by an exact TSP solver, any solution that uses a penalized edge ceases to be optimal TSP tour. The solutions that do not use these edges will surely be feasible with respect to the TSPB constraints. Therefore, the optimal TSP tour for this cost matrix equals the optimal TSPB tour.

In Figure 2.1, tours for TSPPD and TSPB are provided for an example problem instance involving 10 customers. White nodes and black nodes represent delivery and pickup customers, respectively. The white box represents the depot. The black numbers around nodes represent the net quantity demanded. The gray numbers on arcs represent the total load on vehicle while traversing that edge.

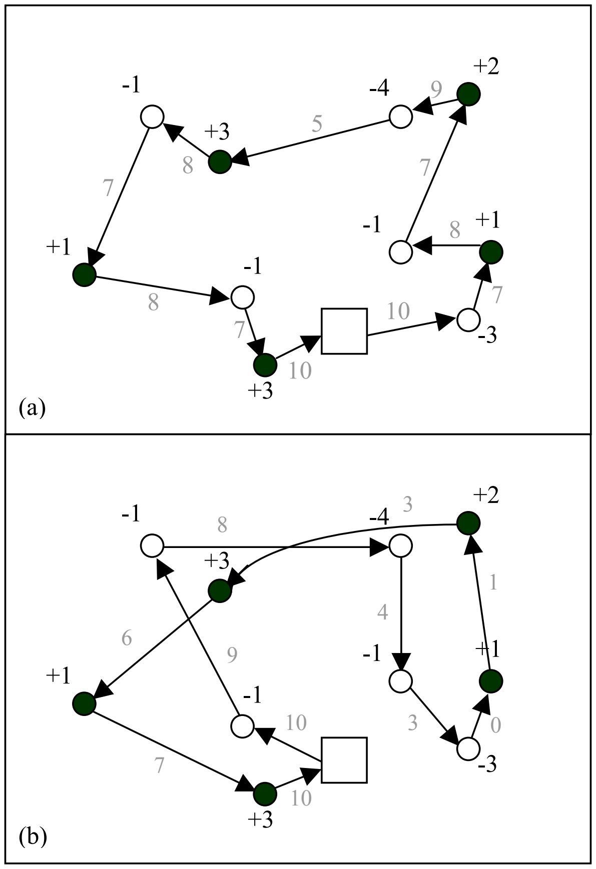


Figure 2.1 (a) A feasible route for TSPPD; (b) a feasible route for TSPB

2.3 Relation to Other Routing Problems

TSPPD and TSPB are important sub-problems for multi-vehicle routing applications. If a cluster first-route second approach is considered to solve the multi-vehicle case, the routing problem in each cluster can be regarded as a TSPPD or TSPB.

Süral and Bookbinder (2003) provide a classification of the routing problems with backhaul options. In their α/β/γ representation, α stands for the number of vehicles associated, β refers to backhaul service options and γ identifies whether a precedence restriction between pickup and delivery customers exists or not. The problems under consideration for our work are of "α=1 and β=must" type, which implies that only one vehicle is utilized and all of the pickup customers are to be serviced. Other than these problems, the authors also investigated "β=free" problems, referring to the cases where visiting pickup nodes is optional. The vehicle collects revenue from each pickup customer and only the profitable pickup nodes are included in the tour. The assumption beneath this type of environments is that discarded nodes are to be served by common carriers. In "γ=any" problems, there is no precedence relationship between delivery and pickup customers. However, in "γ=prec" type problems, the delivery customers should precede pickup customers. Following this notation, TSPPD and TSPB can be referred as 1/must/any and 1/must/prec, respectively.

In their work, Süral and Bookbinder (2003) also investigate the relationship between these variants and TSP. Figure 2.2 provides a schematic representation of the relationship between these problems. In this figure, an arrow means that the problem which it emanates from is a special case of the problem that the arrow points. Therefore, any algorithm that can be used to solve the pointed problem can also solve the pointing problem. In this sense, TSP can be regarded as a special case of TSPPD and TSPB, where one of customer sets is empty. Hence any algorithm that solves these problems in linear time can also solve TSP, which proves that both problems are NP-hard problems. 1/must/any problems can be transformed to 1/must/prec problems by adding a large constant M on the cost of between delivery and pickup nodes and of arcs from depot to pickup nodes. Any algorithm that can solve 1/free/any problem can also solve 1/must/any as the former problem can be transformed to the later one by adding sufficiently large constant M to the revenues of the pickup customers.

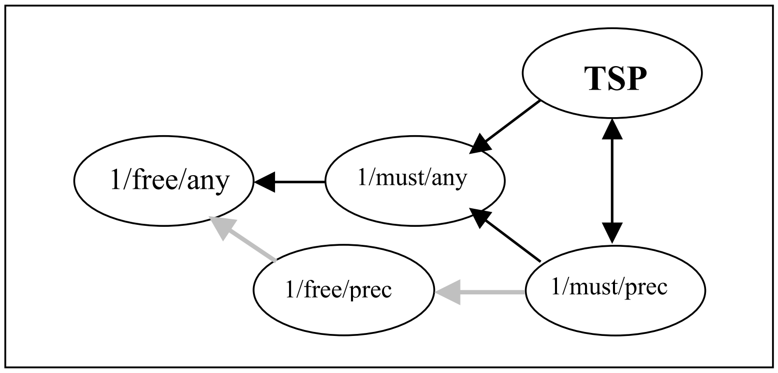


Figure 2.2 Relationships among 1/β/γ problems and TSP

1/free/prec is not discussed in Süral and Bookbinder (2003). For the sake of completeness, we introduce the 1/free/prec problem and relate it to the other variants in this work. 1/must/prec is a special case of 1/free/prec where the revenues to be gathered from pickup customers are irresistibly large. In this case, any algorithm that can solve 1/free/prec can also solve 1/must/prec. Therefore, we add a new arrow (shown in gray) pointing 1/free/prec from 1/must/prec. One can also transform 1/free/any into 1/free/prec by adding a sufficiently large M value to the costs of those arcs from pickup nodes to delivery nodes and also to those from depot to pickup nodes. The arrow between 1/free/any and 1/free/prec represents the possibility of such a transformation.

The TSPPD can find fairly short tours but visiting pickup customers at any order along the tour may create inefficiencies regarding loading/unloading activities. On the other extreme, when pickups constrained to be visited after finishing all deliveries, loading and unloading can be realized easily, However, this time tour length increases considerably. Wade and Salhi (2002) introduce a problem between TSPPD and TSPB, where a vehicle can start to visit pickups only after visiting some deliveries. By this way it is possible to obtain a tour not as inefficient as a TSPPD tour can be regarding loading and unloading, and not as poor as a TSPB tour regarding the tour length. Another variant of these problems is studied by Daganzo and Hall (1993), where a vehicle can not visit more than a particular number of deliveries but pickups.

Both TSPPD and TSPB can be related to the general pickup and delivery problem. The general framework provided by Savelsbergh and Sol (1995) is capable of handling various problems including the pickup and delivery problem, the dial-a-ride problem, and the vehicle routing problem. The first two problems can also be referred as one-to-one type problems, However, the third one is an one-to-many (or many-to-one) type problem. If TSPPD and TSPB are problems where two commodities are to be transported, then they can be classified as one-to-many/many-to-one type problems. The central depot represents the first commodity's supplier and delivery customers are "demanders", whereas pickup customers are the suppliers for the second commodity and only demand point is located on the depot.

Anily and Bramel (1999) study the capacitated TSPPD to transport a commodity from a set of suppliers to a set of demand points. At each supplier one unit of commodity is supplied while the requirement amount is one unit at demand points. In Moon et al. (2002), the traveling salesman problem with precedence constraints is defined. In the problem for a node to be visited, all of its predecessors should be visited in advance. A looser version of this problem is clustered TSP (Jongens and Volgenant 1985), where precedence relations are not defined on individual nodes but on clusters. TSPB can be seen as a clustered TSP having only two clusters.

2.4 Applications in Industry

The routing problems with pickup and delivery or with backhauls are highly important for distribution systems (Bodin et al. 1983). In practice, if vehicles do not permit loading/unloading to be realized quickly and efficiently, it is more appropriate to design a route in which its total load is delivered first and then the goods are picked up. However, if a vehicle with side-loading capabilities is utilized, the pickup customers can well be visited along any order, without consuming too much time for loading and unloading (Süral and Bookbinder 2003). So, both problems can be observed in the same distribution system differing according to the vehicles used.

These type of problems can be observed in supply chains like grocery store chains, retail department store chains, quality stores, where the vehicles transporting goods to customers can be also used for gathering materials or inbound products from suppliers (Yano et al. 1987, Goetschalckx and Jacobs-Blecha 1989, Potvin et al. 1996, Toth and Vigo 1997, Ghaziri and Osman 2003). Gendreau et al. (1999) mention a similar example for beer and soft drinks delivery system. In this example, full bottles should be delivered to customers and empty bottles should be collected from customers.

As a specific example of TSPPD, Mosheiov (1994) discusses an application for the transportation of under-privileged children. In his example, the aim of a non-profit organization is to provide under privileged children two-week long vacation opportunities. Since the visiting dates are determined in advance, it is possible to carry back the children finishing their vacations and to pickup the ones who are just starting their vacation with the same vehicle. In this TSPPD, the depot is the main vacation site, the delivery customers are the families of the children ending their vacation and pick up customers are the families of the other children. Anily and Mosheiov (1994) provide another application from mailing parcel systems, such as UPS, where the mail processed at the depot is to be delivered to recipients, while picking up the mail from senders for processing at the depot. Anily and Bramel (1999) mention about the importance of TSPPD in the context of inventory repositioning. The specific examples of TSPB can be seen in automated warehouse routing and in operation sequencing on numerically controlled machines (Gendreau et al. 1996).

2.5 Solution Procedures

Due to the complexity of TSPPD and TSPB problems, studies in the literature generally focused on the approximation algorithms. However, there also exist some exact solution procedures for each problem. Since TSPB appears before in literature, the related works are more compared to those of TSPPD. Amount of works on

TSPPD has increased with introduction of trucks with side-loading capabilities. In the following section, the solution algorithms for TSPPD, TSPB, and multi-vehicle cases of these problems are summarized.

Solution procedures for TSPPD

The works intending to solve TSPPD at optimality are infrequent in the literature. One can refer to the commodity flow formulation of Mosheiov (1994) or general MIP formulation of Süral and Bookbinder (2003) for solving the problems with modest sizes by commercial MIP solvers. However, as problem sizes get larger, these formulations are not very successful. Baldacci et al. (2003) introduce of new valid inequalities namely, flow, subtour, and capacity inequalities, and propose lower bounds for the problem. The tightness of the lower bounds reported are comparable with that of TSP lower bounds. These lower bounds are used in their exact algorithm based on branch and cut. For Euclidean instances, they could solve problems with 199 customers within one hour, whereas the solvable size decreases to 100 customers for randomly generated problems. In this work, the authors also mention about a genetic algorithm that is used to find good upper bounds for the exact method. However, no algorithmic details are presented in the paper. In personal contact, the authors stated that the heuristic was designed just for obtaining bounds and it should not be considered as a practical solution method.

The first example of heuristic methods for TSPPD comes from Mosheiov (1994). Considering the similarities between TSP and TSPPD, Mosheiov adapts TSP heuristics for TSPPD. Pickup and delivery along optimal tour (PDOT) heuristic starts with constructing the optimal TSP tour along customer nodes. The depot is then inserted to any location that makes the resulting tour feasible. The worst case bound of this heuristic was stated to be 2. For the cases in which it is very hard to find an optimal subtour, the author proposes PαDT, where the subtour is obtained by a TSP heuristic. The worst case bound "α" of the heuristic used for finding the initial subtour would also be valid for the resulting TSPPD tour.

Cheapest Feasible Insertion (CFI), another heuristic proposed in the same work, starts with construction of an α-optimal tour visiting every delivery customers and depot. At the next step, pickup nodes are inserted into the tour. At each iteration, the node increasing the total cost least while not disturbing the feasibility is selected. Although the worst case bound was reported to be infinity for CFI, the computational experiments revealed that there is no significant dominance between PαDT and CFI for problems of size varying between 60 and 200 customers.

Anily and Mosheiov (1994), with a motivation to reduce these worst case bounds, adapt the Christofides heuristic of TSP for this problem. Their heuristic first finds the minimum-spanning tree rooted at the depot node. After doubling all of the arcs, a Hamiltonian cycle starting and finishing at the root node is constructed. The algorithm gives priority for the subtrees with smaller total net demand for visiting. In such a subtree a delivery customer is satisfied immediately. A pickup customer should be a leaf node or all nodes along its subtree should be satisfied before this customer is satisfied. Although this heuristic has a worst case bound of 2, the computational results showed that it is not better than CFI in terms of solution quality.

Gendreau et al. (1999) propose a linear time algorithm for finding the optimal TSPPD on a cycle. Their heuristic firstly finds a TSP cycle using the Christofides heuristic, and then uses this linear time algorithm to find the optimal TSPPD tour on this cycle. A feasible 2-edge exchange procedure is utilized for improvement purposes. The heuristic outperformed PαDT and CFI, regarding the solution quality and time, for Euclidean instances. The authors also propose a tabu search procedure in this work. The tabu search produced slightly better results than all heuristics in term of solution quality at the expense of significantly more time.

Demirel (2001) proposes efficient improvements on the PDOT algorithm of Mosheiov. After a tour for all customers is constructed, it is improved by 2-edge exchange heuristic and then the depot is inserted into the best feasible location. The results showed that the improvement steps improved the solutions by 10.7% on the average. In this work, the adaptations of TSP heuristics are introduced, called Nearest Feasible Insertion and Farthest Feasible Insertion. A delivery tour is constructed via Nearest Neighborhood (or Farthest Insertion) heuristic. Then pickup customers are inserted into the tour as it is realized in CFI. Modified versions of insertion heuristics are also experimented for which the entire tour is constructed at once inserting all customers one by one without violating the feasibility. The farthest insertion heuristic performed better than cheapest insertion in construction of the initial delivery tour.

Solution Procedures for TSPB

Since TSPB can be transformed to an asymmetric TSP, any algorithm that can be used to solve TSP solves TSPB.

In Gendreau et al. (1996) a TSP heuristic, GENIUS, is introduced. At the first part of the heuristic (GENI) a tour is constructed by inserting nodes one at a time. GENI differs from regular insertion schemes as it performs a local optimization scheme at each iteration. First, it selects an unvisited node **v** arbitrarily, and then evaluates several insertion moves. These moves involve deletion of three edges from the tour and addition of four new edges. Two of the new edges are introduced due to connecting **v** to two nodes in the tour, $\mathbf{v_r}$ and $\mathbf{v_s}$. The other two nodes $\mathbf{v_{r+1}}$ and $\mathbf{v_{s+1}}$ previously incident to $\mathbf{v_r}$ and $\mathbf{v_s}$ are connected to nodes $\mathbf{v_k}$ and $\mathbf{v_{k+1}}$, which are already on the tour. Due to time considerations, the possible moves are restricted with the notion of p-neighborhood. That is to say, $\mathbf{v_r}$ and $\mathbf{v_s}$ are selected from the closest p nodes to **v**. Likely $\mathbf{v_k}$ should be among the closest p nodes to $\mathbf{v_{s+1}}$. For different selections of $\mathbf{v_r}$, $\mathbf{v_s}$, and $\mathbf{v_k}$, alternative moves can be obtained. From the possible moves the best one is selected, and the algorithm continues until a complete tour is obtained. GENI is succeeded by the improvement step, US. Here, each node is in turn removed from the complete tour using reverse GENI algorithm and reinserted in the tour using GENI. The algorithm halts when no further improvement can be obtained.

Gendreau et al. (1996), having previously satisfied by the performance of GENIUS for TSP, develop its six different versions for TSPB. H1 solves the corresponding TSP instance of TSPB with updated cost matrix. H2 constructs two cycles for delivery nodes and pickup nodes separately using GENIUS algorithm. These two cycles and the depot are combined by the best possible way. H3 slightly differs from H2 so that the initial cycles include the depot this time. One edge connected to the depot is removed from each cycle and the resulting paths are combined into a tour by introducing an edge between end nodes of the paths. H4 utilizes cheapest feasible insertion scheme for tour construction and US for tour improvement. The authors also propose a feasible Or-opt improvement algorithm. H5 and H6 use this algorithm for improvement. In H5 tour is constructed by GENI, while it is constructed by cheapest feasible insertion in H6.

In their computational studies, GENI turned out to be a better construction heuristic than cheapest feasible insertion, and also US is indicated to dominate feasible Or-opt procedure. In overall H1 is performed well in terms of solution quality and time for problems of size 100 and 200 customers. As problem size increases H2, H3, and H4 show more or less the same performance.

Gendreau et al. (1997) present an adaptation of the Christofides TSP heuristic to TSPB with the worst case bound of 3/2. No metaheuristic application is reported in the literature for TSPB. In the work of Ghaziri and Osman (2003), a neural network algorithm is adapted to TSPB. Its performance is compared with the heuristic proposed in Gendreau et al. (1996). The results show that the proposed algorithm produces competitive results with these algorithms but require much more time.

Solution Procedures for VRPPD

In vehicle routing problem with pickup and delivery (VRPPD), instead of one vehicle, a fleet of vehicles are to satisfy the requirements of pickup and delivery customers. VRPPD is an extremely hard problem to solve. Tzoreff et al. (2002) study the VRPPD problem on special graphs such as path, tree, cycle and so on. Using graph theoretical properties of these structures, the authors propose exact algorithms for these special cases.

Dethloff (2002) proposes an insertion heuristic for the VRPPD. The heuristic differs from the traditional saving based insertion heuristics in determining the saving value. Here, aside from the traditional criterion based on increase of the tour length, excess vehicle capacity after the insertion of the candidate node, and its distance to the depot are also taken into account while determining the node to be inserted.

A thorough review for the multi-vehicle routing with pickup and delivery is provided by Nagy and Salhi (2004). They first classify the problems into three main categories: simultaneous pickup and deliveries, mixed pickup and deliveries, which correspond to VRPPD according to the nomenclature used in our work, and delivery-first pickup second VRPPD, which we call VRPB. Their work is primarily related with the first two types. They proposed a route first-cluster second algorithm. Firstly a giant tour including every customer node and depot is constructed via TSP heuristics. Then, with the help of a direct cost network representing the length of the paths on this giant tour, the clustering, is achieved in best possible way. During this clustering the maximum of total pickup load and delivery load is sought to be less than the vehicle capacity. Nevertheless the resulting tours may result in a weakly feasible tour. Therefore, the authors propose several improvement procedures with feasibility maintaining, such as 2-opt, 3-opt, node shift, node exchange, etc. They also adapt the heuristic to be capable of handling the multi depot case. The results of the heuristic integrated with different improvement modules prove to dominate the previously proposed algorithms for VRPPD both for single depot and for multi depot case.

Solution Procedures for VRPB

For the multi-vehicle case of TSPB, there are a number of exact methods proposed in the literature. Toth and Vigo (1997) propose a branch and bound algorithm for both symmetric and asymmetric cases of the problem. With variable reduction and feasibility check the algorithm's performance is enhanced. It is reported that the algorithm solves instances with 100 customers at optimality.

In the work of Mingozzi et al. (1999), a new BIP model for VRPB is developed. The model is based on defining feasible path sets. Two heuristics for finding a feasible solution to the dual problem are proposed and the solutions found are used in variable reduction in the exact solution method. Their method was able to solve problems up to 100 customers within a time limit of 25,000 seconds.

Very first example of heuristic methods for this problem is given in Goetschalckx and Jacobs-Blecha (1989). The authors use a space filling curve methodology for both clustering and routing decisions. The algorithm provides initial solutions efficiently. The routing subproblems in clusters are improved with 2-opt and 3-opt, and the resulting tours are within 1% of optimal tour values.

Toth and Vigo (1999) propose a cluster first-route second algorithm for the problem. K delivery clusters and K linehaul clusters are formed at the end of the clustering phase based on Lagrangian relaxation. Afterwards, the delivery and pickup clusters are merged according to the edges included in the Lagrangian solution or the results of an assignment problem. At the last phase, a TSPB for each cluster is solved using the farthest insertion heuristic. The results of heuristics were better than those of Goetschalckx and Jacobs-Blecha.

There are also metaheuristic applications for VRPB in the literature. Genetic algorithm of Potvin et al. (1996) and Tabu search of Osman and Wassan (2002) can be mentioned as examples. In Potvin et al., VRPB with time windows is studied. An algorithm is proposed to construct a VRPB tour by inserting the nodes one by one according to a specified order. This algorithm is incorporated into a GA scheme where "order of insertion" is evolved throughout generations. The results provided were 1% of the optimum. In reactive tabu search of Osman and Wassan (2002), there are mainly two types of moves. The 1-interchange mechanism shifts one single customer from one route to another or exchanges two nodes, whereas the 2-consecutive-node interchange mechanism shifts or exchanges two-consecutive nodes between the route pairs. The initial tours are constructed using saving insertion heuristic and saving assignment heuristic.

2.6 Metaheuristics and Complex Routing Problems

There is a vast literature about the metaheuristics proposed for the standard routing problems, TSP and VRP. The most commonly used metaheuristics are Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithms (GA). The reader is referred to Johnson and McGeoch (1997), Rego and Glover (2002) for general discussions about metaheuristics for TSP, and Golden et al. (1998) for VRP. For a detailed discussion of GAs that work well for the TSP, the reader can refer Sönmez (2003). In general, SA and TS work better than GA for VRP (Golden et al. 1998). For TSP, good genetic algorithms are also reported by many works (Johnson and McGeoch 1997, Nagata and Kobayashi 1997, Sönmez 2003).

The literature lacks a general reference for metaheuristics for constrained routing problems, including our problems. The work of Van Breedam (2001) provides a general comparison scheme for improvement heuristics. The author compares several metaheuristics, specifically TS and SA, with conventional improvement heuristics based on descent methods (DH), for VRP. All heuristics analyzed use the same basic moves, which are string cross, string exchange, string relocation and string mix. Extensive experiments are performed with different parameters on general characteristics (i.e., geometry of node locations, side constraints on the problem, etc.), and on algorithmic characteristics (i.e., move type, initial solutions, etc.). According to the experimental results, it is reported that DH halts execution more quickly than SA and TS and the best solution of DH is better than the intermediate solutions that are produced with more or less the same effort. However, TS and SA produce better results as they proceed. No superiority is reported for one over the other among these two metaheuristics. Although VRPPD is included in the experiments as a problem type variant, no specific results are provided for it.

When we narrowed our attention to the realm of GAs, we see that a need exists for a general analysis of GAs for constrained problems. The main difficulty in applying GAs to even standard combinatorial optimization problems, like TSP, is the problem of feasibility of produced solutions. The generic crossover operators are incapable of handling the constraints of combinatorial optimization problems. Reeves (1997) summarizes major strategies to handle constraints in GAs. By modifying operators, repairing or penalizing infeasible solutions, using multiple objectives or modifying the formulation, GAs can tackle the feasibility problems. For TSP a common approach is to modify crossover operators while using proper solution representation schemes (Michalewicz and Fogel 1998). The solution representation schemes alienating to the original schemes proposed of Holland (1975) and crossover operators getting more complex, caused the algorithms to be differentiated as Evolutionary Algorithms (EA) by conservative GA theorists.

The feasibility issue for constrained routing problems gains more importance. If we adapt EAs originally designed for standard routing problems to constrained routing problems, the additional constraints will probably be violated even the basic subtour elimination constraints are satisfied. Fortunately the strategies summarized by Reeves (1997) can be applied here also. In the literature, there are some EA applications on the specific constrained routing problems such as Potvin et al. (1996), Moon et al. (2002), preserving feasibility of the solutions by using one of the formerly stated ways. However, the author could not note any work comparing the performance of feasibility maintaining methods in EAs for constrained routing problems. This deficiency of literature constitutes the core motivation of our study.

CHAPTER 3

PROPOSED ALGORITHMS

As it is mentioned before, EA, when designed appropriately, is a dependable tool for solving TSP. There are various successful EA algorithms proposed for this problem in the literature. However, the problem is defined with inevitable accompanying constraints in many real life applications and the algorithms that are specialized in solving the standard case is of limited use for industry. Kilby et al. (2000) mention about the necessity for the standard procedures for handling the additional constraints on routing problems. They also note the unstable and unpredictable nature of these side constraints. We believe that the standard procedures provide a natural base for the constrained cases. Through this respect, in our study, we adapt an EA that works well for TSP to the constrained cases.

In the previous section, the main handicaps of this adaptation and overcoming efforts are stated briefly. The feasibility regarding the side constraints are the main difficulties during this process. From now on, both of the terms of "feasibility" and "infeasibility" will be referred with respect to these side constraints instead of the regular tour constraints of TSP.

Michalewicz and Fogel (2000) provide a broad investigation of constraint handling techniques in EA. The methods can be classified in two classes: feasibility seeking methods and infeasibility penalizing methods. The most common feasibility seeking methods are rejecting infeasible solutions, repairing infeasible solutions and maintaining a feasible population using special representations and variation operators. The authors note that the global optimum may generally occur at the boundaries of the solution space and to accept some attractive infeasible solutions in the population may be a good method for some problems. In this context, the problem of evaluating the attractiveness of infeasible solutions arises. The general evaluation method is to utilize the penalty functions to decrease the attractiveness of these individuals. The most attractive penalizing functions are the adaptive ones

(Reeves 1997). These functions update the penalizing coefficients throughout generations considering the difference between best infeasible and best feasible solutions found so far. In this study, we restrict ourselves to deal with only these most common infeasibility handling strategies. In order to provide a fair comparison basis, we apply these strategies in the simplest and the basic form.

In the following subsections, the algorithms are defined for the two constrained routing problems, namely TSPPD and TSPB. The chapter starts with identifying the general EA components and providing the algorithm for the unconstrained problem. Then, this algorithm is modified to the constrained problems for each constraint handling strategy in the following sections. In each section, we first outline the implementation of the specific strategy and then give the general algorithm that provides basic structures that can be adapted to different side constraints without too much effort. Each section includes specific modification of general algorithms for TSPPD and TSPB also.

3.1 General EA components

The main components of EAs can be itemized as coding scheme, fitness function, initial population, selection strategy, reproduction operators, replacement strategy, and stopping criteria. Beasley et al. (1993) provide a summary for these general components. Sönmez (2003) proposes an EA for TSP, which finds high quality solutions in reasonably small time. This study constitutes the basis of ours. Most of the components of our EA are decided according to the findings and discussions in Sönmez (2003).

Coding Scheme

Sönmez (2003) provides a thorough listing of representation schemes for TSP. These representation schemes are grouped into two major categories: Vector and matrix representations. Binary, path, adjacency, ordinal, and rank representations are mentioned under the vector representations group. Among the listed alternatives the path representation is the most natural and logical one for TSP. A feasible TSP tour is represented by a sequence of numbers indicating the sites to be visited. Many successful EAs utilize this scheme (Nagata and Kobayashi 1997). Therefore, the path representation is selected in our study.

Fitness Function

The core of EAs is providing reproduction chances for the good solutions, which are expected to yield better ones. For this purpose one should evaluate the goodness of the solutions, which are generally realized by fitness functions. Although there are examples of works questioning the benefits of using fitness functions (Chen et al. 1999), commonly they are among the key features of EAs. We used the length of the tour as the fitness value of a solution, as it is the most sensible alternative.

Initial Population

One of the main components of an EA is the initial population that is composed of a certain number of solutions, which are produced in advance with any procedure. Two characteristics regarding the initial population should be decided in the very beginning. The first one is the population size, which generally remains constant throughout the generations. The second one is the nature of starting solutions, i.e., the procedure used to produce these solutions. Sönmez (2003) experimented with different population sizes varying from 50 to 200 for the algorithm she proposed. The finding was that a population size of 50 was appropriate for the problems with sizes up to 250 customers. Sönmez uses a population of 100 individuals for the larger problems. As we utilized a different reproduction scheme, namely, steady-state reproduction scheme, we did not use a population size as it is suggested in Sönmez (2003). We have realized a preliminary experiment about the influence of the population size on the solution quality for TSP instances. According to this experiment, the larger number is set as the size of population in our study. The results are provided in Section 4.3.

The solutions in initial population may be produced as the good starting solutions or they can simply be produced randomly. For the sake of simplicity in this study the initial solutions are produced randomly. By this way, we can focus on the core of the study and compare the performances of the proposed constraint handling strategies without any other effect. When good solutions are also included in the population, it can be quite complex to identify the underlying reason for superiority of one strategy over another. We proposed three methods to generate initial solutions, each of which is differing in the treatment of feasibility.

In method 1 only the feasible solutions are produced. It first constructs random solutions and then the resulting solutions are repaired if necessary. The general method is provided in Figure 3.1. Please refer to Section 3.2.3 for the repair algorithms utilized for TSPPD and TSPB.

```
(a)     Procedure_method1
▪     Start with the depot node
▪     Repeat until all nodes are visited
        ▪     Pick an unvisited node randomly add it to the end of the chain
▪     Repair the resulting tour if necessary
```

Figure 3.1 The pseudo code of method 1 for initial population generation

Method 2 produces feasible solutions also. However, this time, the feasibility is sought during construction of the tour. In order to explain the steps of this algorithm let’s turn back to network definition of the problem, where the customers and the depot are represented by the nodes. The algorithm starts with the depot node. Among the customer nodes whose additions do not violate feasibility, one is randomly selected to place at the end of the path. The procedure is repeated until no node is left. The algorithms of this method for TSPPD and TSPB are provided in Figure 3.2. In the figure, the variable, *vehicle_load* keeps the amount of the load on vehicle.

Method 3 constructs the tour by randomly selecting the nodes. As it can be guessed, this method does not guarantee feasibility. For tightly constrained cases such as in TSPB, a great portion of the solutions produced are expected to be infeasible because solution space of the constrained problem is much smaller than the standard case.

```
(a)     Procedure_method2_for_TSPPD
  ▪ Start with the depot node
  ▪ Assign the vehicle_load as the total delivery load
  ▪ Repeat until all nodes are visited
      ▪ Pick an unvisited node if its addition does not violate capacity randomly and add it to
        the end of the chain
      ▪ Update the load on the vehicle


(b)     Procedure_method2_for_TSPB
  ▪ Start with the depot
  ▪ Repeat until all nodes are visited
      ▪ If all delivery nodes are not visited
            Pick an unvisited delivery node randomly
        Else
            Pick an unvisited pickup node randomly
      ▪ Add the node at the end of the chain
```

Figure 3.2 The pseudo code of method 2 for initial population generation

```
(a)     Procedure_method3
  ▪ Start with the depot node
  ▪ Repeat until all nodes are visited
      ▪ Pick an unvisited node randomly and add it to the end of the chain
```

Figure 3.3 The pseudo code of method 3 for initial population generation

In our experiments, we investigate three initial population settings. In the first setting, all solutions are feasible ones: half of the population is produced by method 1 and the remainder is produced by method 2. In the second setting, each of method 1 and method 2 produces a quarter of the population. The remaining half is produced by method 3. This type is expected to have infeasible solutions in the initial population. The third setting, in which the number of infeasible solutions is expected to be larger than the former two settings, is produced by method 3 only.

Selection Strategy

In each iteration, EAs select parents from which new solutions are to be generated. The basic consideration in selection is to favor the good solutions in providing chance to reproduction; therefore the common approach selects parents from the population according to their fitness. However, there are examples of random selection strategies in literature.

The selection methods mentioned in Sönmez (2003) were roulette selection, tournament selection, and GENIE selection. The comparisons between these selection schemes yield that ranking and tournament schemes give better results than the regular roulette scheme. In fact, both of these schemes are appreciable as they do not require any rescaling procedures to overcome the effect of the fairly good solutions to dominate the populations, which is a problem in roulette selection. The examples of good algorithms using ranking schemes are available in literature, such as Whitley's GENITOR (1989). The GENIE selection strategy of Chen is given as an example for random selection in Sönmez (2003). Due to the improvement capability of the crossover operator, the fitness information is disregarded for selection in this heuristic.

Among the selection strategies summarized here, ranking based strategies can be implemented easily. We use the linear fitness function scheme as proposed in Reeves (1995). All solutions in the population are ordered according to the fitness function values (i.e., in non-decreasing order of the tour length). The probability of selecting a solution is assigned a value, inversely proportional to the rank of the solution, where the probability mass function follows a decreasing trend for increasing ranks. Specifically, let p_i be the probability of selecting the solution of rank i as a parent, and let n be the size of candidate population. The value of p_i can be computed as:

$$p_i = \frac{(n+1)-i}{\sum_{k=1}^{n} k}$$

After the first parent is selected, the corresponding solution is taken out of consideration, the size of the candidate population is decreased by one, and the probabilities are computed again in the same fashion for selecting the second parent.

Reproduction Operators

The reproduction operators can be analyzed under two main classes: crossover operators and mutation operators. Crossover operators basically generate combinations of two solutions (named as parents). Usually, applying the crossover operator to selected two parents, two new solutions (named as offspring or children) are produced. In the literature there is a vast source on crossover operators for TSP. Sönmez (2003) provides a classification of these operators according to the preservation characteristics of operators. These groups are stated as position, order, and edge preserving crossover operators. Edge preserving operators are found more appropriate as the edges are agents directly affecting the solution quality. The order and position are of little use in speaking of the goodness of a solution.

In Sönmez (2003), the idea of using conventional heuristics as the crossover operators is proposed. The crossover proposed mainly constructs a union graph of selected parent solutions and applies the nearest neighbor (NN), the greedy and the insertion type TSP heuristics on that union graph. The experiments of Sönmez revealed that the NN works better than the others regarding solution quality and time; therefore it is taken as the crossover operator of our algorithm.

The NN operator firstly selects a random starting node from the union graph. At each iteration the nearest reachable unvisited node is added to the tour. Whenever the partial graph cannot yield to an unvisited node, the nearest unvisited node is selected using the edges of the complete graph. The algorithm halts if all nodes are visited. This is the deterministic version of NN. The stochastic version uses a probability scheme to decide which node to visit next. The probability figures are inversely proportional to the distance of the unvisited nodes to the current one on the tour. As it is noted in Sönmez, the initial edges traversed in this operator are relatively shorter ones. As the algorithm goes on, since the number of alternative routes to proceed decreases, the edges added may get worse. In fact, the two edges that connect the last node to be added to the tour do not involve any distance consideration. If the parent solutions are the same solutions, the operator cannot produce a solution different from these parents.

In this work, from two parents, two children are produced by applying the NN crossover operator twice. The first child is produced starting with a random point. The last node added to this tour is used as the starting node in the production of the second child. To illustrate we provide an example with the following symmetric cost matrix in Table 3.1.

Table 3.1 The cost matrix for a symmetric TSP problem

Nodes	0	1	2	3	4	5
0	-	6	9	7	5	6
1		-	12	4	5	9
2			-	8	11	14
3				-	3	13
4					-	15
5						-

Example:

Parent 1: 0 1 2 3 4 5

Parent 2: 0 2 1 3 5 4

Let node 1 be the starting node for the procedure

Offspring 1: 1 3 4 0 5 2, where edge 5-2 is taken from the complete graph.

Since node 2 is the last node, it will be the starting node in offspring 2.

Offspring 2: 2 3 4 0 1 5, where edges 1-5 and 2-5 are taken from the complete graph.

If two parents are the same, the union graph constructed cannot allow any other solution to be produced. Only the direction of the resulting tour may change. This case is illustrated in Figure 3.4. In the construction of the first child, the algorithm starts with node 1. In a case where node 0 is closer to node 1 than node 2, the resulting child follows the sequence of the parents in the opposite direction. In construction of child 2, obtaining a tour same as the parents is exemplified. Here the starting node is node 4. If node 5 is closer to node 4 than the other alternative (i.e., node 3), the same tour as the parent tours is obtained.

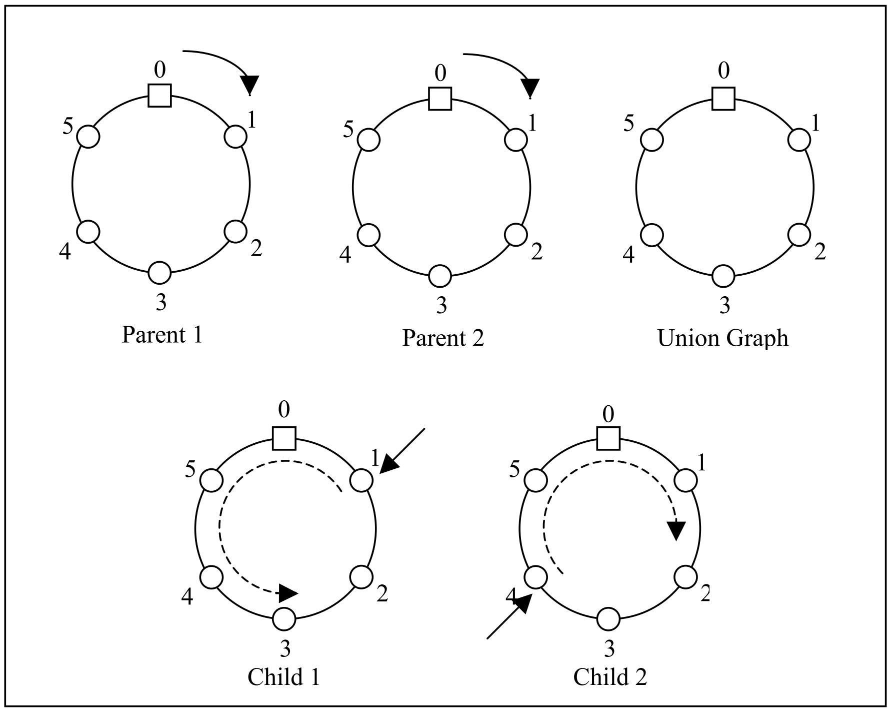


Figure 3.4 Illustration of reproduction when two parents are the same

In general, mutation operators that alter the genes of the solution randomly can be seen as operators that produce a new solution from an initial solution. They are essential in maintaining variability in population and in preventing premature convergence to a sub-optimal solution (Beasley et al. 1993). It is very convenient and common way of using mutation operators for improvement purposes (Jog et al. 1989). The results of Sönmez (2003) reveal that when EA starts with random population it is possible to improve the solution by 4% with a proper mutation operator. Although the results will most probably get better due to usage of a good mutation operator, in our study no such operator is used, as the main purpose of this study is to compare the raw strategies.

Replacement Strategy

Beasley et al. (1993) define the concept of “generational gap” corresponding to “the proportion of individuals in the population which are replaced in each generation”. The traditional algorithms, which are called “generational algorithms”, use a generational gap of 1. A new population is produced and this new population replaces the old one, in these algorithms. In a relatively newer scheme, at each generation only two solutions are produced per generation and new solutions replace two solutions of the population. This strategy, which is called “Steady State” replacement strategy, is proved to work well by many examples (Whitley 1989, Davis 1991).

There are many replacement strategies proposed for steady state algorithms. Beasley et al. (1993) summarizes the two of them: (i) selecting the leaving individuals randomly, or (ii) considering their fitness values. In GENITOR of Whitley (1989), which is a steady state algorithm, the worst two individuals of the population are dismissed from population, and the offspring directly enter to the population.

In our study, we use a steady-state replacement strategy, where only two children are produced from two parents at each generation. When we applied the replacement strategy as applied in GENITOR our preliminary results were unimpressive. Then we revised it. In the revised case, we select the solutions to be removed from the population among the parent pair and the children. The solutions are basically sorted and the best two solutions enter the population. The other two are deleted. To slow down the convergence, we try to delete the parents that are the same as either one of the children. In this stage, we assumed that two solutions having the same tour length are the same, which may not be the case. Actually, this strategy may dismiss a parent solution of same length with the child, despite the difference between the two. This drawback may be overcome by devoting additional effort for identifying the sameness. However, as this scheme provided fairly good results in the preliminary experiments, we gave up the additional effort of identifying sameness.

Note that after deleting these parents directly, there may be less than four solutions to be sorted. This replacement scheme may cause population average to get worse; if two fairly good parents, both having a fitness value **F**, produce one child with the same fitness value and one child with the poorer fitness value **K** (**K** > **F**), both parents are removed and both children enter into the population and finally population average gets worse.

Stopping Criteria

Sönmez (2003) provides a list of stopping conditions. These conditions are number of generations, computation time, fitness threshold, and population convergence. There are various ways to decide on the population convergence. Some of these are differences between population best and population worst or population average.

In this work, we used several stopping conditions. In one group of experiments the number of generations is fixed. In the other group, the algorithms halt when the population best does not change for a fixed number of generations. Both of the stopping conditions are equipped with an additional criterion examining the population convergence. The algorithm terminates whenever the value of the population average deviates from the value of the population best by no more than 0.1%.

General algorithm

In Figure 3.5, the flowchart of the general structure of our EA is provided. The iterations start just after the generation of initial population (INITIALIZATION). At every iteration, two parents are selected from the population (SELECTION), crossover operator is applied and two children are produced (REPRODUCTION). The parents and the children are sorted. While the worst two leave the population, the good ones enter the population (REPLACEMENT). In this step, if a child has the same tour value with a parent (i.e., if a tie occurs), the parent is removed from the population. The resultant population is ordered and the algorithm proceeds to the next iteration, unless the one of the stopping criteria is met.

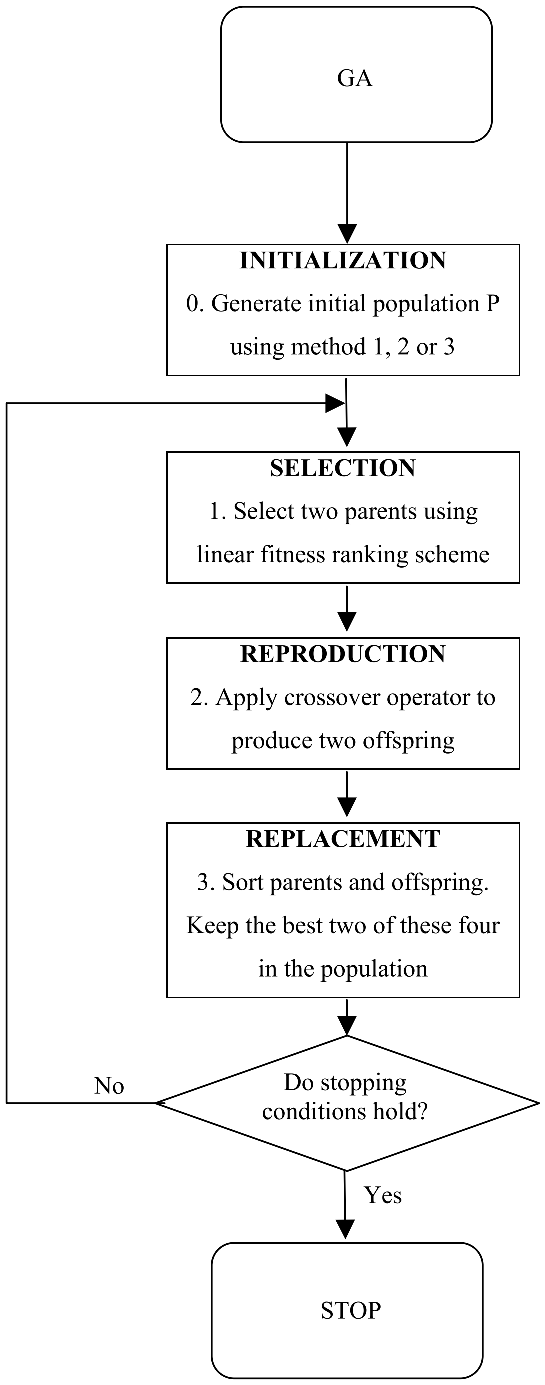

GA
INITIALIZATION
0. Generate initial population P using method 1, 2 or 3
SELECTION
1. Select two parents using linear fitness ranking scheme
REPRODUCTION
2. Apply crossover operator to produce two offspring
REPLACEMENT
3. Sort parents and offspring. Keep the best two of these four in the population
Do stopping conditions hold?
No
Yes
STOP

Figure 3.5 The flowchart of the general EA

3.2 Feasibility Seeking Algorithms

In this section, our algorithms proceeding with feasible populations are proposed. The problems for which the feasible solutions are hard to obtain from infeasible solutions, spending time with infeasible solutions may not be worthwhile. The strategies that insist on producing feasible solutions may be a proper way of treating these problems. Unfortunately, for many cases, it is not known in advance whether keeping only feasible solutions in the population should be forced or not. In addition, to our knowledge, there is no work discussing the benefits of dealing with only feasible solutions for the constrained routing problems in literature. Therefore, three main feasibility seeking approaches are proposed in the study: namely, rejecting the infeasible solutions until finding a feasible solution, using a modified crossover operator approaches, and repairing the infeasible solutions to make them feasible. Note that the second and third approaches produce feasible offspring directly even if the parent solutions are infeasible. Following subsections discuss these approaches in detail.

3.2.1 Rejecting Infeasible Solutions (REJECT)

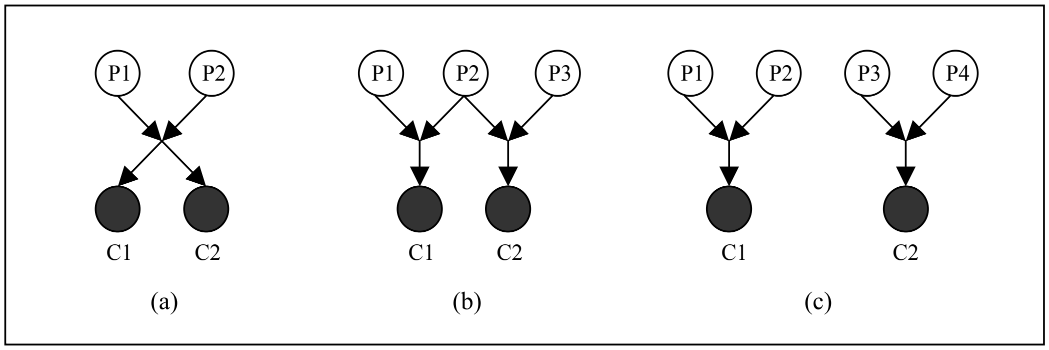
Rejecting infeasible solutions, as "death penalty" (Michalewicz and Fogel 2000), is the simplest and easiest approach that can be used for handling constraints. This strategy can be adapted to a variety of side constraints by just changing the feasibility checking procedures. Hence, it provides a common ground for all of the side constraint classes. However, this strategy works well when the feasible solution space is convex and constitutes a larger portion of the solution space without side constraints. Convex feasible solution space exists when it is guaranteed to produce feasible solutions provided that the parents are feasible. The approach is not very beneficial for which the solution spaces of the standard and the constrained problems rarely coincide.

The algorithm, called REJECT, enumerates all possible starting points and parent combinations in order to find two new feasible solutions. If a parent combination yields less than two feasible solutions after trying every node as the starting node, the other parent combinations are tried until two solutions are found. The algorithm iterates if more than one solution is found, otherwise it terminates. The general structure given in Figure 3.5 can be updated for this strategy by simply implementing SELECTION and REPRODUCTION routines recursively and in a combined fashion, until finding two feasible solutions. Let *starting_node* denote the random node used as the starting node in the crossover operator and *feas_child* keep the number of feasible children found in an iteration. Also let *not_mated* be the set of individuals that are not tried before. Define *parent1* and *parent2* as the individuals selected to be first and second parents respectively. Figure 3.6 provides the combined SELECTION and REPRODUCTION routines for adapting the general EA for REJECT. The complete REJECT algorithm can be obtained by replacing SELECTION and REPRODUCTION routines in Figure 3.5 with the one provided in this figure.

```
▪ Initialization: not_mated = P, feas_child = 0
▪ Repeat until () or (not_mated = { })
    ▪ Choose parent1 from not_mated according to linear fitness ranking, update
      not_mated
    ▪ Repeat until ( feas_child = 2) or all other parents are tried with parent1
        ▪ Choose parent2 from not_mated according to linear fitness ranking, so that
          parent2 is not mated with parent1 previously
        ▪ Construct the union graph
        ▪ Choose starting_node which is not tried before
        ▪ Repeat until (feas_child = 2) or all nodes are used as a starting point
            ▪ Apply standard NN crossover operator
            ▪ If the solution is feasible with respect to the side constraint then
              feas_child = feas_child + 1
            ▪ If (feas_child < 2) then choose a random starting_node which is not
              tried before
```

Figure 3.6 Combined SELECTION and REPRODUCTION routines for REJECT

In this strategy, the reproduction is more complex than it is in general EA and can result in one of three cases. The regular case occurs when two feasible children are found from the same parent pair. The second case occurs when a specific pair can yield only one feasible child. If this is the case, the algorithm tries to mate the first parent (*parent1*) with other individuals. If the algorithm finds another feasible solution in these trials then there will be three parents and two children. The last case may occur when the children are produced from different parent pairs. All of these cases are shown in Figure 3.7.



*Pi denotes parent i and Cj denotes child j.

Figure 3.7 Alternative reproduction schemes in REJECT

For case (a) the replacement scheme is already discussed in page 30. In case (b) is realized, if the fitness value of a parent is equal to that of either children, this parent is removed from the population. However, if all three parents have the same value as the child, then only two of them are removed. After removing these parents, if there exist at all, the remaining parents and children are ordered and best three are selected to join the population. In the overall two of five solutions are dismissed. For case (c), as there are two separate groups ordering is realized separately. The best two of each group join the population.

The sole modification required in the combined SELECTION and REPRODUCTION routine to apply the algorithm for TSPPD and TSPB is related with determining the *feas_child* value. Actually, when a child is produced its feasibility is checked regarding the side constraints associated. If it is found feasible *feas_child* is increased by one. For TSPPD, a feasible child is recognized by checking the vehicle load after visiting each node along the tour. Whenever, this quantity exceeds vehicle capacity, the algorithm marks the solution as "infeasible". Whereas for TSPB, detection of a pickup node to be visited before all of the delivery nodes are visited is sufficient to deem the solution infeasible.

Let Ω_{TSP}, Ω_{TSPPD}, and Ω_{TSPB} be solution spaces of TSP, TSPPD, and TSPB respectively. In an environment where both pickup load and delivery load do not exceed the vehicle capacity we can set the following relationship:

$$\Omega_{TSP} \supseteq \Omega_{TSPPD} \supseteq \Omega_{TSPB}$$

Therefore, considering the time consumed for the infeasible solutions, we can say that REJECT algorithm will surely work faster for TSPPD than it works for TSPB, as the solution space of TSPB is a subset of TSPPD. For a TSP where there are $\mathbf{p} + \mathbf{d}$ many customer nodes, there are $(\mathbf{p} + \mathbf{d})!/\ 2$ solutions in the solution space. Whereas the solution space of TSPB consists of $\mathbf{p}!*\mathbf{d}!$ many solutions.

3.2.2 Modified Crossover Operator (CONSTRUCT)

This strategy is a problem specific approach of handling side constraints, which exploits the problem's side constraint characteristics. The solution is constructed from scratch without violating the side constraints. The crossover operator includes a feasibility-check procedure in which at each step a node is included into the tour. For different constraints the feasibility check in this procedure should be updated.

The algorithm CONSTRUCT is the general EA algorithm provided in Figure 3.5 with the modified crossover operator. The crossover operator, called Nearest Feasible Neighbor (NFN), searches for the nearest unvisited feasible nodes on the union graph. If there are multiple nodes that can be added to the tour without violating feasibility, then the nearest one is selected. When no such node can be reached through the edges of union graph, the edges of complete graph are used. The algorithm starts with the depot, than a feasible node is picked depending on the distance of the nodes to the depot. The probability of picking a node among all possible nodes is the inverse distance of this node to the depot over the sum of inverse distances of all feasible nodes. The pseudo code of the updated REPRODUCTION routine for this strategy is provided in Figure 3.8.

```
▪ Construct the union graph of parent1 and parent2
▪ Select a feasible starting_node considering the distance, connect this node to the depot,
▪ Repeat until all nodes are visited
    ▪ If there is an unvisited feasible node that can be reached from the current via the
      edges of the union graph go to this node
      Else go to the nearest unvisited feasible node by using complete graph edges
▪ Connect the lastly added node and the depot
```

Figure 3.8 NFN crossover operator routine

In the preliminary experiments, several settings are tried for selecting the *starting_node*. One of them was pure random selection among the possible nodes. The other was selection depending on the square of the inverse distance. However, these alternatives turned out to give results worse than the previously stated setting.

Another thing to mention here is the construction of two different children. For the first child the algorithm executes in a forward fashion, i.e., the later a node is added to the partial tour in the construction phase, the later it is visited in the resulting tour. Whereas the second child is constructed by the backward execution of the crossover operator, i.e., the later a node is added in construction, the earlier it is visited in the final tour.

The feasibility checking procedure should be updated in order to adapt this general operator for TSPPD. Generally, the algorithm should keep the vehicle load information at each iteration and the load should never be more than the vehicle capacity. The procedure differs among the children construction. For the first child, the nodes are added in a forward fashion. The *feasible* nodes to be added at the end of the chain satisfy the following inequality:

$$\mathbf{Vload\_a\ [i]} \leq \mathbf{Q}$$

where **Vload_a** [**i**] denotes the load on the vehicle after visiting node **i**. **Vload_a** [0] is the total delivery load. If node **i** is visited just after node **j**, **Vload_a** [**i**] is computed as follows:

Vload_a [**i**] = **Vload_a** [**j**] + **load** [**i**]

where **load**[**i**] keeps the demand information for customer **i**. Note that for delivery customer **i**, **load**[**i**] ≤ 0.

During the construction of the first child the delivery customers can be visited at any step. Provided that **Vload_a** is currently less than **Q**, it can never exceed **Q** after adding any delivery node because **load** entries are nonpositive for these nodes.

In the construction of the second child, the algorithm proceeds backward. Hence, **Vload_b** keeps the load on the vehicle before visiting the nodes. Consequently, the inequality restricting the *feasible* nodes is updated similarly:

Vload_b [**i**] ≤ **Q**

The inequality ensures the capacity feasibility on the edge just before visiting any candidate node. **Vload_b** [0] keeps the total pickup load. If node j is currently the first node of the chain, **Vload_b** value or candidate node **i** can be computed as follows:

Vload_b [**i**] = **Vload_b** [**j**] **- load** [**i**]

Here the pickup nodes can be visited any time, however, the delivery loads should be less than the excess capacity on vehicle.

TSPB requires a simpler feasibility check procedure. For the first child, the *feasible* nodes are the delivery nodes unless all delivery nodes are included in the chain. When all delivery nodes are visited, all pickup nodes become feasible to add in the tour. For the second child, the pickup nodes are initially feasible and whenever there is no pickup node left to add, all delivery nodes become feasible.

3.2.3 Repairing Infeasible Solutions (REPAIR)

Repairing is a commonly used method for handling constraints. In this strategy the infeasible solutions are replaced by their repaired versions. This approach relates to Lamarckian evolution, which states that improvements gained during lifetime of an individual are coded back into genetics of that individual. In literature, there are examples of successful EAs for individual problems which utilize repairing (Ulusoy et al. 1997, Erdem and Özdemirel 2003). However, it is almost impossible to design a general repair algorithm that can handle different constraint types. In some cases like nonlinear transportation problems, repairing itself is a complex task (Michaelwicz and Fogel 2000).

Although there are some applications where only a portion of repaired individuals replace their infeasible origins, we replaced all infeasible solutions in this work. Michaelwicz and Fogel (2000) state a 5-percent rule which works well for many combinatorial optimization problems. However, the authors also state that this cannot be generalized to the realm of combinatorial optimization. Reeves (1997) reports the algorithm of Orvosh and Davis (1993), where the repaired version replaces the infeasible solution with a probability.

A repairing procedure is needed in the REPRODUCTION routine of the general EA algorithm of Figure 3.5 in order to adapt the algorithm to this strategy. After two children are constructed by the standard crossover operator, if necessary, they are repaired.

The repair algorithm of TSPPD is based on a result due to Mosheiov (1994). The author showed that for every tour, there is at least one specific starting node and direction to follow, which makes the tour feasible with respect to capacity constraints. That is to say, any tour can be made feasible by deleting the depot from its current location and inserting it to this specific location. An example is provided in Figure 3.9. In this example, white and black nodes represent delivery and pickup customers, respectively. The black numbers around nodes represent the net quantity supplied in the node and the the gray numbers on arcs represent the total load on vehicle while traversing that edge. The vehicle capacity is 10. In Figure 3.9 (a), the bold black numbers represent the vehicle load on the edges causing infeasibility. The load on vehicle exceeds capacity if the pickup customer with a supply of 3 is visited at the second place. Also after visiting the pickup customer visited in sixth order, the load exceeds the capacity. In Figure 3.9 (b), a repaired version for this tour is presented. By simply changing two incident nodes of the depot, all of these infeasibilities are avoided.

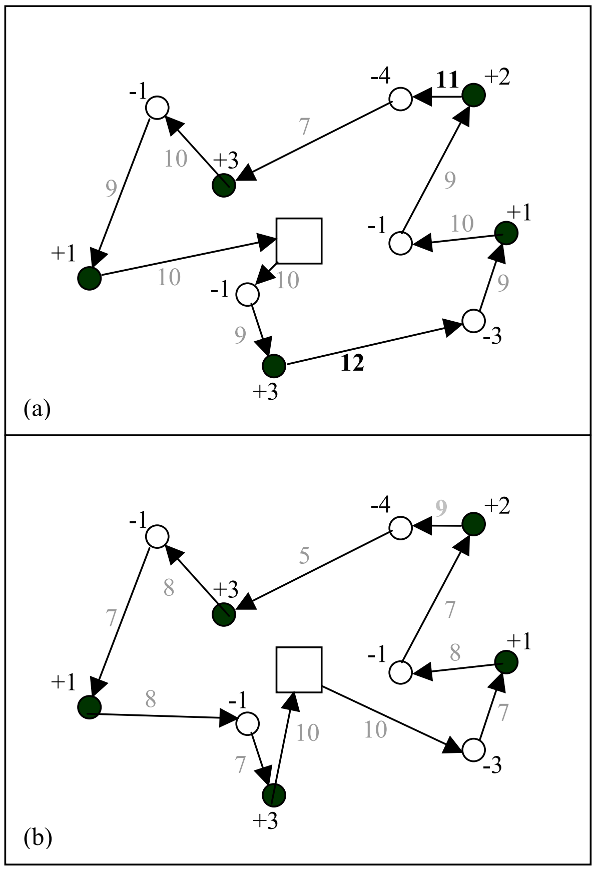


Figure 3.9 An infeasible TSPPD tour (a) and the repaired TSPPD tour (b)

Actually in the above example, there is more than one possible location to insert the depot. The illustrated repaired tour in (b) is just one of them. In our application the depot is inserted into the best location, if more than one alternative exists. By this way, this strategy is made comparable with the other alternatives. As the lastly added two edges do not consider any distance information, the repaired solutions may be very poor. To prevent this bias and to get the approach closer to the nearest neighbor notion, the best possible location is taken.

In TSPB, a repair algorithm, which is expected to work well on the Euclidean instances, is proposed. The basis of the algorithm is to preserve the relative order of delivery (pickup) nodes on the initial tour. The underlying assumption is that if two nodes are visited consecutively in a good solution then probably they are close to each other. The algorithm mainly constructs two separate chains: delivery chain and pickup chain. It starts from the depot, adds nodes one by one to the chain of their type in the order of the initial tour. Then these two chains and the depot are combined. Among four alternatives the one yielding the minimum tour length is taken. In Figure 3.10, the result of the TSPB repair algorithm is illustrated on an example in (b). Again white and black nodes represent the delivery and pickup customers, respectively. In this example, firstly, a white chain and a black chain are produced regarding the order of the nodes in (a). Then these two chains and the depot are connected in best possible way. For the example, this best alternative requires reversing of the pickup chain originally produced by following the order of (b).

3.3 Infeasibility Penalizing Algorithms

In penalizing strategies, infeasible solutions are allowed to exist in the population. However, by manipulating fitness values, their probability of survival in the coming generations is decreased. This strategy, different from rejecting strategy, may take advantage of keeping several infeasible solutions in population as they can yield good feasible offspring through little modification. For the problems where the optimal solution is at the boundaries of the feasible solution space, this method may yield good results (Reeves 1997). In order to benefit from these infeasible solutions the penalizing scheme should be designed carefully. If the penalty coefficients are selected to be insufficiently small, the search may get far from the feasible solution space. In these cases, the algorithm may even cease to find a feasible solution at the end. On the other hand, if the coefficients are unnecessarily large, the boundaries of feasible solution space may be left unexplored. Generally, these coefficients are set regarding the closeness of the solution to the feasible region, which requires a feasibility distance metric.

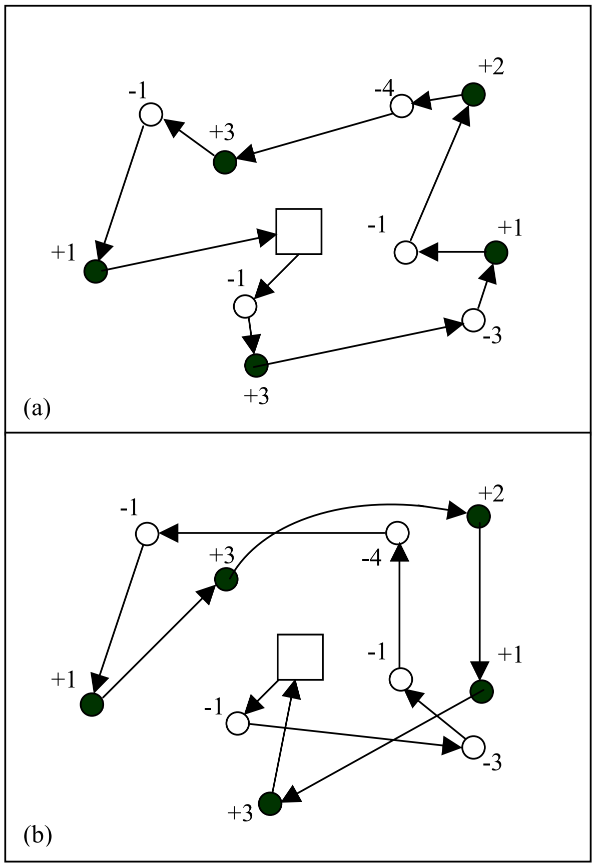


Figure 3.10 An infeasible TSPB tour (a) and the repaired TSPB tour (b)

Two examples in the literature use the amount of violation of the constraints or the number of violated constraints as the distance metric. (Beasley and Chu 1996, Coit et al. 1996). These penalizing schemes are general schemes and can be adapted to different side constraints with little effort. The general structure of the EA will not change, only the procedures computing the amount of infeasibility should be added. Hence, like rejecting strategy this scheme provides a general basis for solving the problems with side constraints.

Michalewicz and Fogel (2000) consider the effort required to repair an infeasible individual as penalizing basis. Although this infeasibility metric is intuitive and logical as it requires specific repairing procedures for each side constraint type, it cannot be considered as a general approach for problems with different side constraints.

According to Michalewicz and Fogel (2000), the appropriate penalizing method should consider the problem specific properties like the solution space topology (i.e., convexity of the feasible solution space) and the ratio between sizes of the feasible and the whole search space. If a distance metric based on constraint violation is to be used, the modeling related properties of the problem (i.e., the number of variables and constraints) and type of the constraints should also be considered in order to come up with a proper penalizing function.

The general EA algorithm that is using penalizing schemes does not differ from the one given in Figure 3.2. The REPRODUCTION routine is enlarged by feasibility checking and penalizing procedures. Then, the penalized fitness values are used for selection and replacement.

In our work two penalizing methods are proposed for TSPPD and TSPB. The first method uses the effort to repair the infeasible solution as the penalizing basis as in Michalewicz and Fogel (2000). The value of the repaired infeasible solution is set as the penalized fitness value. The second method is an adaptation of the penalizing scheme proposed by Coit et al. (1996) to our problems.

3.3.1 Penalizing by Repairing (PEN_REPAIR)

It utilizes a direct infeasibility metric. The distance of the infeasible solution to the feasible solution space is taken as the difference between the original infeasible solution value and the value of the repaired value solution. Hence, the penalized fitness value of a solution is

$$\mathbf{F_p(x) = REP(F(x))}$$

where $\mathbf{F_p(x)}$ is the penalized value of an infeasible solution, $\mathbf{F(x)}$ is the original solution value and **REP** function returns the value of the repaired solution. If the solution is feasible the function returns the original solution value.

Using the repair algorithms explained in Section 3.2.3, the penalizing function repairs any infeasible solution produced. The repaired solution replaces the original solution if it is shorter than the original solution.

3.3.2 Penalizing by Adaptive Penalizing Scheme (PEN_ADAPT)

As it is noted by Reeves (1997), static and naive penalty schemes are capable of finding good results. Coit et al. (1996) provide the example of Siedlecki and Sklansky (1989) of dynamic updating scheme for the penalty coefficients. In the earlier stages of evolution, the infeasible solutions are penalized less. However, as the algorithm proceeds the coefficients are increased in order to shift the population to the feasible solution space.

The more sensitive approach is to use adaptive penalizing schemes in which the history and the current state of the evolution are incorporated in deciding penalizing coefficients. Coit et al. (1996) proposed a general adaptive penalty method for constrained combinatorial problems; The infeasible solutions are penalized considering the difference between best feasible and best overall solution found so far, and the remoteness of the infeasible solution to the feasible solution space.

Using their terminology, an adaptive penalizing method is designed here. The penalized values are computed as follows:

$$\mathbf{F_p(x) = F(x) + (F_{feas} - F_{all}) \sum_{i=1}^{n} \left(\frac{d_i(x,B)}{NFT_i} \right)^{K_i}}$$

where $\mathbf{F_{feas}}$ is the value of the best feasible solution value found so far , and $\mathbf{F_{all}}$ is the value of the best solution, not necessarily feasible, found so far. The distance of the solution to the feasible area is kept with $\mathbf{d_i(x,B)}$ considering the ith constraint of type **B**. This metric should be defined separately for different constraint types. $\mathbf{NFT_i}$ is the Near Feasible Threshold value for the ith constraint, which sets the boundary for the acceptable feasible solutions. And $\mathbf{K_i}$ is the severity index for the ith constraint. Note that if $\mathbf{K_i}$ >1 then the “near” infeasible solutions are penalized less than the distant ones. If the current best solution is feasible, the infeasible solutions are not penalized at all. When the search finds some better infeasible values, the function begins to penalize them.

NFT can be used as a static parameter where it is possible. However, in most cases, there is not a standard, well-defined **NFT** value that can be set in advance. In such cases, it can be initialized to a reasonably large value, and can be decreased through the generations. Coit et al. (1996) provided the following equation for this dynamic setting of **NFT**.

$$\mathbf{NFT} = \frac{\mathbf{NFT}_0}{1 + \lambda \mathbf{g}}$$

where **g** stands for the generation number, and $\mathbf{NFT}_0$ is the initial value. In this setting **NFT** is a monotonically decreasing function of the generation number for the positive values of λ. In our work, this dynamic approach is used for **NFT**. This penalizing setting is originally used for the Unequal Area Shape Constrained Layout Problem and Redundancy Allocation Problem. In the first one, the objective is to minimize the total cost of flow among the rectangular facilities located on a rectangular area. For any facility, the proportion of width to length should be in some predetermined range. The facilities with the proportion value out of this range are deemed infeasible. Here the distance metric is based on the number of facilities violating the shape constraints. The authors used a static NFT value of 2. In the second problem, where the reliability of a system is to be maximized using parallel components, there are two restrictions regarding cost and weight. For this problem, the distance of the solution to the feasible space is exceeded the amount of constraints. Since the appropriate **NFT** value cannot be foreseen, the authors used a dynamic **NFT** for this case. Large $\mathbf{NFT}_0$ values such as 70% of the constraints, and slow cooling schemes such as λ =0.04 found to give better results.

The algorithm for this strategy for specific side constraints is as provided in Figure 3.5. The children are produced and penalized using $\mathbf{F}_{\mathbf{feas}}$ and $\mathbf{F}_{\mathbf{all}}$ values of the previous generation. Also $\mathbf{F}_{\mathbf{feas}}$ and $\mathbf{F}_{\mathbf{all}}$ values are not updated until the solutions that are better than these values enter the population. That is to say, if an infeasible solution with a value better than $\mathbf{F_{all}}$ is penalized highly and dismissed directly, we do not update the value of $\mathbf{F_{all}}$. When an update of these values occurs with the introduction of new solutions to the population, the penalized values, which are computed using the previous values of $\mathbf{F_{feas}}$ and $\mathbf{F_{all}}$, are recomputed using the updated values. Therefore all of the infeasible solutions in the population should be penalized again before proceeding to the next generation.

In order to adapt the general strategy for TSPPD, first of all, a suitable distance metric should be defined. The distance metric used is the total exceeding of the capacity constraint. It can be computed by using the positive difference between the vehicle load and the capacity after visiting each node. But this can cause unfair penalization of the solutions. Let us think of an instance in which there is only one pickup customer with load **m**, and **m** delivery customers with unity demands. When the vehicle visits the pickup customer first, the vehicle capacity is exceeded along every node of the tour. Hence the solution is penalized highly although it can be repaired with small modifications on its structure. In order to cope with this cumulative effect, the individual effects of the nodes are considered in computing the distance of the solution to feasibility. Since visiting a delivery node decreases the capacity violation this node is excluded in computing $\mathbf{d_i(x,B)}$. If visiting a pickup node causes the load on vehicle to exceed capacity, then the exceeded amount is taken as the constraint violation for this node. When we come to a pickup node, having already exceeded the capacity, after visiting this pickup node, the exceeding will increase. Instead of considering the total violation, we take only the contribution of this node to the total violation in computing $\mathbf{d_i(x,B)}$. The general formulation for $\mathbf{d_i(x,B)}$ can be generalized as follows:

$$\mathbf{d_i(x,B)} = \begin{cases} 0, & \text{if } \mathbf{i} \in \mathbf{D} \\ \mathbf{load}[\mathbf{i}] & \text{if } \mathbf{i} \in \mathbf{P} \text{ and } \mathbf{Vloadb}[\mathbf{i}] \geq \mathbf{Q} \\ \max(0, \mathbf{Vloadb}[\mathbf{i}] + \mathbf{load}[\mathbf{i}] - \mathbf{Q}) & \text{if } \mathbf{i} \in \mathbf{P} \text{ and } \mathbf{Vloadb}[\mathbf{i}] \leq \mathbf{Q} \end{cases}$$

where **B** denotes the capacity exceeding constraint. Here **Vloadb**[**i**] keeps the quantity of load on vehicle just before visiting node i.

The following example illustrates the computation of **d(x,B)**. For the instance provided in Figure 3.11, the encircled pickup nodes cause infeasibility. Let the first encircled node be indexed as **a** and the second one be indexed as **b**. After visiting these nodes, the vehicle capacity is exceeded by 2 for **a** and by 1 for **b**. Therefore $\mathbf{d_a(x,B)} = 2$ and $\mathbf{d_b(x,B)} = 1$ and $\mathbf{d_i(x,B)} = 0$ for every other nodes.

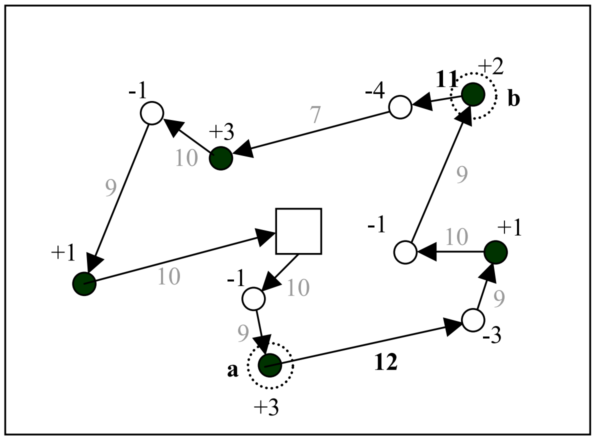


Figure 3.11 Example for computing $\mathbf{d_i(x,B)}$'s for TSPPD

A dynamic **NFT** is used for this strategy. In the preliminary experiments, $\mathbf{NFT}_0$ and λ settings of Coit et al. (1996) are found inappropriate for our problem. The reason may arise due to the difference in the number of iterations to termination, or the difference due to the type of crossover operators. The authors terminated the algorithm after 750,000 iterations, but in our work the algorithm is ended earlier. The crossover operator used in Coit et al. is a variant of the uniform crossover operator. In our work, the crossover used is based on a conventional heuristic. Therefore, the probability of having a feasible child from an infeasible parent may vary. Finally, we set $\mathbf{NFT}_0$ and λ very tightly regarding the preliminary experiments: $\mathbf{NFT}_0 = \mathbf{Q} / \mathbf{n}$ and $\lambda = \mathbf{n} / (2 * \mathbf{iternum})$ where **n** is the number of customers and **iternum** is the termination generation. When the generation number equals to **iternum, NFT** will be equalized to $(2 * \mathbf{Q} / \mathbf{n}) / (\mathbf{n} + 2)$. The numerator represents the load for a pickup customer when there are an equal number of pickup and delivery customers with equal demands. It is mentioned before that the capacity is assumed to be equal to the total delivery load. When the infeasibility occurs due to the location of just one pickup customer, **d**(**x**,**B**) value becomes equal to (2 * **Q** / **n**). **K** parameter is set as 1, as suggested in Coit et al. (1996).

For TSPB the distance metric can be designed more easily. **d**(**x**,**B**) simply holds the number of the pickup chains visited before than all delivery customers. Note that a chain may also be composed of only one customer if it is preceded and appended by a delivery node. In Figure 3.12, there are three pickup chains, two of which are visited before than all delivery customers. Chains violating the precedence constraints of TSPB are encircled on the figure and **d**(**x** , **B**) is equal to 2.

For TSPB the preliminary experiments yielded tight values for the penalizing function parameters. **NFT**$_0$ is set to 1 and λ = **n** / (2 * **iternum)** as it is in TSPPD.

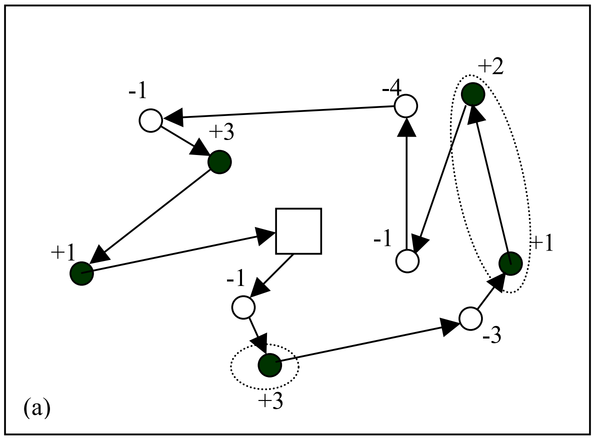


Figure 3.12 Example for computing **d$_i$**(**x**,**B**)'s for TSPB

In this chapter, five EAs differing in the strategy for handling infeasibility are proposed for constrained routing problems. These algorithms are then adapted for TSPPD and TSPB. The following chapter reveals the performance of these algorithms for TSPPD and TSPB.

CHAPTER 4

EXPERIMENTAL RESULTS

The core of this study is to analyze the performance of the EAs proposed in the previous chapter. “Performance” of a heuristic procedure is generally defined by the quality of the solutions it provides and its computational time requirement. However, additional measures can be defined in order to point at specific considerations.

In order to comment on the performance of our algorithms, several experiments were designed and carried on computer environment. We first determined performance measures and identified related statistics, which are appropriate for our purposes. The related algorithms were then coded using C programming language. These codes were run on a HP computer with a Pentium 4, 1.6 GHz processor, and 256 MB RAM.

This chapter presents our results and findings of this computational experiment study. We clarify the experiment environment in Section 4.1. The performance measures utilized and the problem set are presented. After revealing the experiment parameters in Section 4.2, the results of our experiments and the analyses of the algorithms are provided in Section 4.3. The chapter ends with the concluding remarks given in Section 4.4.

4.1 Experiment Settings

This section explains the setting of our experimental study. First, the performance measures that are used to analyze the algorithms and the related statistics that should be collected in these experiments are identified. The section continues with explanation of the test instances utilized in the study. Then, how the optimal solutions and bounds for these instances are found is discussed in the last part of this section.

Performance Measures

As usual, the most important measure is related with the solution quality. The best solution found by the algorithm is generally present in the final population. However, in penalizing schemes it may be needed to keep the best feasible solution separately due to poor penalizing schemes. In such cases, insufficiently penalized infeasible solutions may dominate the population, which may cause deletion of the best feasible solution from the population. In either case, it is essential to keep the best solution information. We also defined several other measures. A complete list of these measures and statistics are provided in Table 4.1.

DEV_{opt} keeps the percent deviation of the best feasible solution from the optimal value of the constrained problem. This measure is used for TSPB experiments, where we could obtain optimal tours for all of the instances. Since it is not the case in TSPPD, we have used DEV_b, the percent deviation of the best feasible solution from the lower bound value. For the lower bound value optimal TSP tours are used. FB, Value of the best solution in the final population (FB), and the population average in the final population (FA) are recorded for analyzing convergence of algorithms. As the decision on convergence is given depending on the difference between FB and FA, it is found appropriate to consider the unpenalized fitness values of infeasible solutions while computing FB and FA in the penalizing strategies. Therefore it is necessary to keep the value of the best feasible solution in the final population, which is kept by FFEAS in the penalizing algorithms.

Another performance measure, which is as important as solution quality, is the computation time. In our study, average computation time in seconds is given as CPU. CPU excludes generation of initial population, inputting and outputting times. The time elapsed for generating initial population is kept by CPU_{ini}, in seconds.

Initial population type is one of the experiment parameters in our study. The performance of the algorithms may differ regarding the population type used. Although, for all types, the initial populations are generated randomly, the number of feasible solutions in the population may differ. Hence the probability of including feasible solutions differs among these types.

Table 4.1 Performance measures and statistics used

DEV_{opt}	Percent deviation of population best from optimal
DEV_b	Percent deviation of population best from optimal TSP tour
FB	The value of best solution at termination
FA	The value of population average at termination
FFEAS[1]	The value of best feasible solution at termination
IMP_b	Percent improvement in best solution compared to the best solution
IMP_{avg}	Percent improvement in population average compared to the initial population average
IMP_{feas}[1]	Percent improvement in best feasible solution compared to the initial best feasible solution
CPU	Computation time for main algorithm (sec.)
CPU_{ini}	Computation time for generating initial population (sec.)
GEN	Number of generations at termination
C_{del}	Number of deleted children per generation
EP	Percent of edges taken from complete graph per child
ENUM[2]	Number of infeasible solutions produced per feasible child divided by problem size
TOTREP[3]	Number of total repaired children
$EDGE_{rep}$[3]	Percent of edges added due to repairing
LOC[3]	Percent of depot locations tried per repaired child to problem size
F_{avg}[1]	Number of feasible solutions per generation
F_{all}[4]	Value of best solution at termination

[1] defined for penalizing strategies, [2] defined for REJECT, [3] defined for REPAIR, [4] defined for PEN_ADAPT,

Therefore we defined IMP_b and IMP_{avg} to keep the percent improvement of the population best and the population average relative to initial population values, respectively. When the initial population lacks feasible solutions, which can be the case when we generate the entire initial population randomly, the improvement value is not computed. For the penalizing strategies, the former measures include infeasible solutions and IMP_{feas} is the percent improvement of the best feasible solution relative to initial population value.

Stopping condition is one of the key features of an EA. For the proposed algorithms, the difference between population best and population average is used for determining when to stop, as well as an upper bound on the number of generations. Since this bound is difficult to set in advance, several numbers of generations to terminate are experimented in the study. Thus, the number of generations at termination (GEN) is recorded during the experiments.

In the replacement strategy, the children produced may be deleted directly when they are poorer than their parents. The effort used for producing children that are deleted directly is unbeneficial. Therefore, the portion of the children that are directly deleted from the population is selected as a performance measure to compare the proposed algorithms. C_{del} keeps the number of deleted children per generation. The total number of solutions produced is typically two times the number of generations until termination. The number of children that are included in the population can be computed by subtracting C_{del} times the number of generations at termination from two times the number of generations at termination.

The resemblance of the children to their parents is an important issue in analyzing the crossover operator. By keeping the percent of edges taken from the complete graph per child, EP, the resemblance can be measured. This quantity is expected to vary among strategies. Since the crossover operator itself in CONSTRUCT checks feasibility, the edges taken from the complete graph is expected to be more for the other strategies. For the algorithms utilizing repairing, the edges added to the solution due to this operation are excluded in computing this measure.

Apart from general measures, some strategy specific measures were also computed in the study. The number of infeasible solutions generated until obtaining a feasible solution is an important indicator for measuring the computational effort to obtain a feasible solution in REJECT. However, the number of infeasible solutions produced may be vary with respect to the size of the instance. To provide a general measure the average number of infeasible solutions per feasible solution is divided by the size of the instance. ENUM keeps this value for REJECT.

In REPAIR strategy, we are interested in behavior of repairing procedures utilized. For example, the number of edges added by the repair algorithms should be computed in order to comment on the resemblance of the children to their parents. In fact, for TSPPD, the repair algorithm introduces at most three new edges and this quantity does not change the solution much, especially, when the problem size gets larger. On the other hand, the number of edges added is not fixed and may be very large for TSPB. Therefore, $EDGE_{rep}$, which keeps the number of edges changed due to repair operation, is defined for REPAIR strategy for TSPB only. For this strategy, the total number of children repaired, TOTREP, is also kept. In addition, the repair algorithm for TSPPD searches for new locations for depot, making the resulting tour feasible and selects the location, ensuring feasibility at minimum cost. We are interested in the question of how many such locations exist on the average. If there are too many alternatives for depot location, instead of picking the best location, picking the first location may be advisable. For that purpose, LOC keeps the average number of feasible locations per repaired solution.

The average number of feasible solutions in the population is an important quantity for penalizing strategies in order to analyze the behavior of the penalizing schemes and the distance metric. Therefore, during the experiments F_{avg}, keeping this quantity, is computed for penalizing schemes.

For PEN_ADAPT the value of the best solution found so far should be kept. F_{all} is defined for that purpose.

Test Problem Set

A test bed of 20 small-sized problems is utilized for the first part of our experiments. These problems are taken from the VRP literature. The sizes of the problems vary from 20 to 151**.** Six problems are uniformly generated, two of them are clustered problems, and the remaining problems are adapted from the real life problems, which do not have any pattern. The distances between the locations are made integer by rounding the real valued distance to the next integer. The sources and sizes of these problems are provided in Table 4.2.

Half of the nodes are selected to be pickup customers randomly for TSPPD. In cases where there are an odd number of customers, the number of delivery nodes is larger by one. The total loads are equated by adding the difference between the initial loads of two sets of customers to the load of customer set with fewer loads.

Table 4.2 The test problems

Name	Authors	Source	Size
P00	Christofides, Eilon	http://neo.lcc.uma.es/radi-aeb/WebVRP/	22
P01	Christofides, Eilon	http://neo.lcc.uma.es/radi-aeb/WebVRP/	33
P02	Christofides, Mingozzi, Toth	http://neo.lcc.uma.es/radi-aeb/WebVRP/	51
P03	Christofides, Mingozzi, Toth	http://neo.lcc.uma.es/radi-aeb/WebVRP/	101
P04	Fisher	Operations Research 42, (1994) 626-642	45
P05	Christofides, Mingozzi, Toth	http://neo.lcc.uma.es/radi-aeb/WebVRP/	121
P06	Tsigilirides	Journal of the Operational Research Society 35, (1984) 797-809	30
P07	Tsigilirides	Journal of the Operational Research Society 35, (1984) 797-809	20
P08	Tsigilirides	Journal of the Operational Research Society 35, (1984) 797-809	30
P09	Mosheiov	European Journal of Operational Research 79, (1994) 299-310	25
P10	Christofides, Eilon	http://neo.lcc.uma.es/radi-aeb/WebVRP/	23
P11	Christofides, Eilon	http://neo.lcc.uma.es/radi-aeb/WebVRP/	30
P12	Christofides, Mingozzi ,Toth	http://neo.lcc.uma.es/radi-aeb/WebVRP/	151
P13	Christofides, Eilon	http://neo.lcc.uma.es/radi-aeb/WebVRP/	76
P14	Fisher	http://neo.lcc.uma.es/radi-aeb/WebVRP/	72
P15	Fisher	http://neo.lcc.uma.es/radi-aeb/WebVRP/	135
P16	Christofides, Mingozzi, Toth	http://neo.lcc.uma.es/radi-aeb/WebVRP/	101
P17	Rinaldi, Yallow	http://neo.lcc.uma.es/radi-aeb/WebVRP/	48
P18	Augerat	http://neo.lcc.uma.es/radi-aeb/WebVRP/	34
P19	Hadjiconstantinou, Christofides, Mingozzi	www.or.deis.unibo.it/research_pages/Orinstances/VRPLIB/VRP.html	36

Finding the Optimal Solutions and Bounds

For finding the optimal TSP tours on these problem data, CONCORDE symmetric TSP solver of Applegate et al. (1998) is utilized and optimal tours are obtained within seconds by this software. Finding optimal TSPPD tours were not that easy. Formulations of Mosheiov (1994) and Süral and Bookbinder (2003) were run in CPLEX 8.1. Unfortunately, within a time limit of 24 hours, only ten problems with smaller sizes could be solved to optimality. When we compared the optimal values for TSP and TSPPD for these 10 problems, the average deviation of the optimal TSPPD values from optimal TSP values turned out to be 2.33%.

TSPB instances of the problems are transformed to asymmetric TSP (ATSP). And then these instances are transformed to symmetric TSP (STSP) instances. The resulting instances are solved by CONCORDE again. Transforming a TSPB to STSP is illustrated on example provided in Appendix A.

Values of the optimal TSP and TSPB tours for the entire test bed, and the optimal TSPPD tours for the smaller 10 instance are provided in Appendix B.

4.2 Experimental Factors

The main parameters of the experiment are five strategies utilized in handling the additional constraints, namely, REJECT, CONSTRUCT, REPAIR, PEN_REPAIR, and PEN_ADAPT. The second parameter is the type of initial population. Three different initial population types are utilized in this experiment, which are called ini_{feas}, ini_{half} and ini_{rand}. Entire population is composed of feasible solutions in ini_{feas}, whereas only half of the population is guaranteed to be feasible in ini_{half}, the other half is produced randomly. In ini_{rand}, all of the solutions are produced randomly. The detailed discussion on these initial types was provided in Chapter 3.

The last experiment parameter is the bound on the stopping condition. Unfortunately there is not a single appropriate value for this bound. The bound value can be set according to the trade-off between the computational effort and the solution quality. If we stop early, we can generate solutions quickly, but these solutions may be poor in terms of the solution quality. If we wait too much to terminate, it may take long times to obtain the solutions. For different environments, different measures may gain importance. Therefore, in this study we have tried several different bounds for stopping conditions.

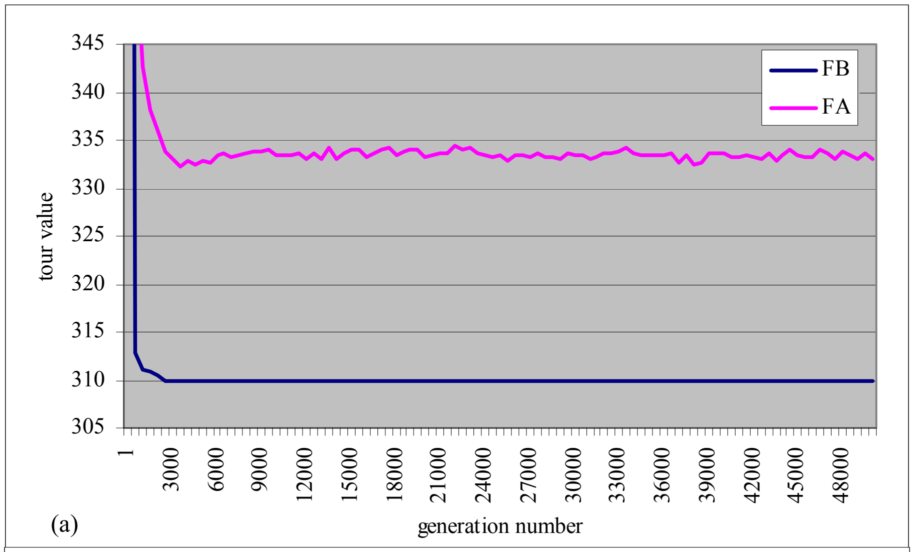
In order to take the first step in the analysis, a preliminary experiment revealing the convergence behavior and change of the solution quality through generations is performed. For this purpose, two TSPPD and TSPB instances were run for sufficiently large number of generations. A small sized problem, p00 (of size 22), and a larger sized problem, p15 (of size 135), were selected for the analysis. The next step was determining the length of this run. In Sönmez (2003), the convergence analysis has been performed by producing 50,000 new solutions for a TSP instance with 52 locations. Considering this, we have decided to run the problems 50,000 generations, in which a total of 100,000 solutions was produced. For each strategy and problem combination, 30 replications were realized. The population bests and population averages were recorded at every 500 generations. After averaging these values for 30 replications, they are plotted on graph. These graphs are provided in Appendix C**.**

For all of the strategies, the convergence cannot been achieved after 50,000 generations for both problems. For the small problem, there is a constant gap between the population best and average. The reason of the constant gap and unnoticeably slow convergence may be the replacement scheme utilized in the algorithms. The population average may get worse and actually it gets worse for many cases, which may prevent the convergence. Although the best solution achieved before 8,000th generation did not improve until the end in general, in CONSTRUCT, the optimal solution has been found in every replication before 3000 generations, and the population average was 7% higher than the optimal value at the termination. This example illustrates drawbacks of stopping the algorithm only when the convergence is obtained. The graphs for the related case are provided in Figure 4.1.

For the larger problem, the gap between population best and average is not constant and tends to decrease throughout the entire run in many replications. However, the best solutions at termination are within 1% of the population best at the 8,000th generation. Considering the huge computational effort to achieve this improvement in solution quality, the solutions found until 8000 generations are deemed good solutions. An example of the convergence behavior for the large problem is provided in Figure 4.2.

Finally, 8000 is determined to be sufficient for the upper bound on the number of generations for stopping condition. However, the performance measures and statistics are recorded for 2000th and 5000th generations also. These three stopping conditions will be called by their generation number at termination in the following sections as, stopping condition 2000, stopping condition 5000 and stopping condition 8000.

We also test another stopping condition, where the algorithm halts when the population best does not change for a number of generations. This stopping condition is designed mainly for larger problems where the population best is expected to improve after 8000 generations. It is set as 15,000 after analyzing convergence plots for the larger problem. This stopping condition is named as Fixedbest.



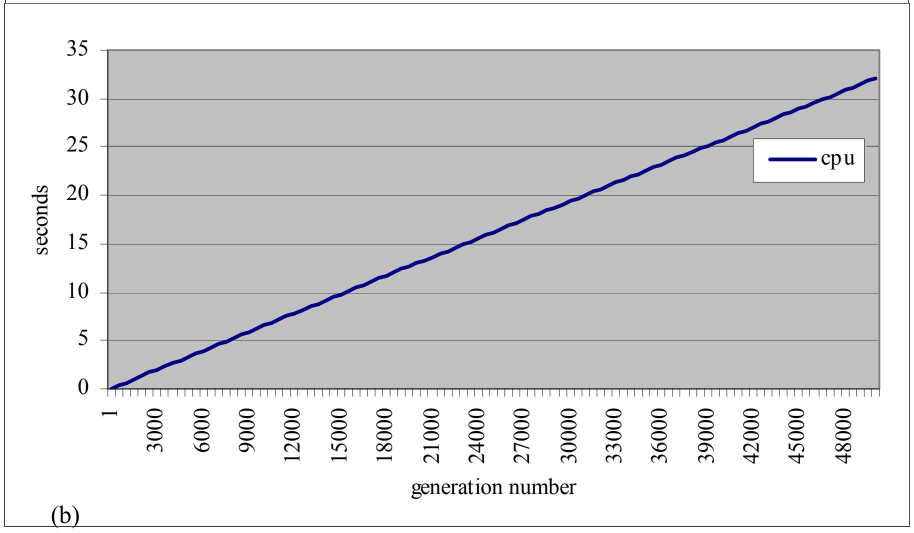
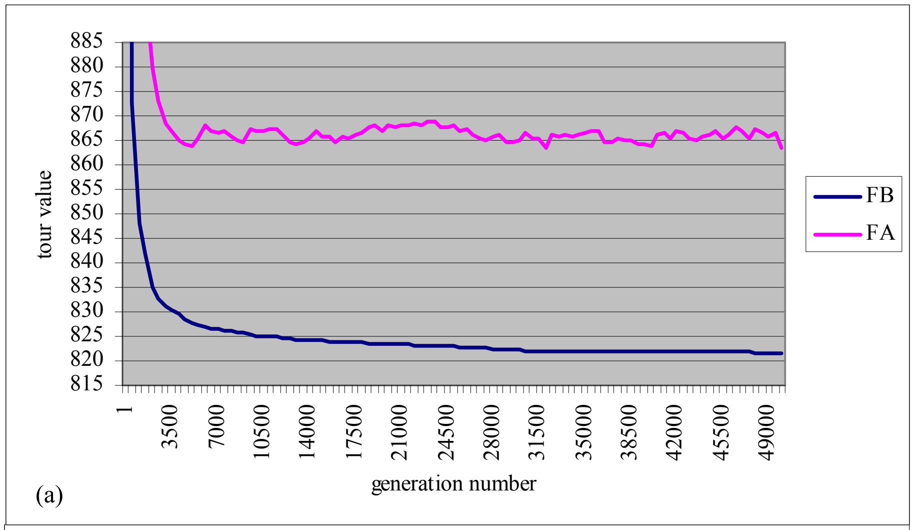


Figure 4.1 (a) FB and FA vs. number of generations and (b) CPU vs. number of generations for CONSTRUCT with ini_{rand} for the small problem instance



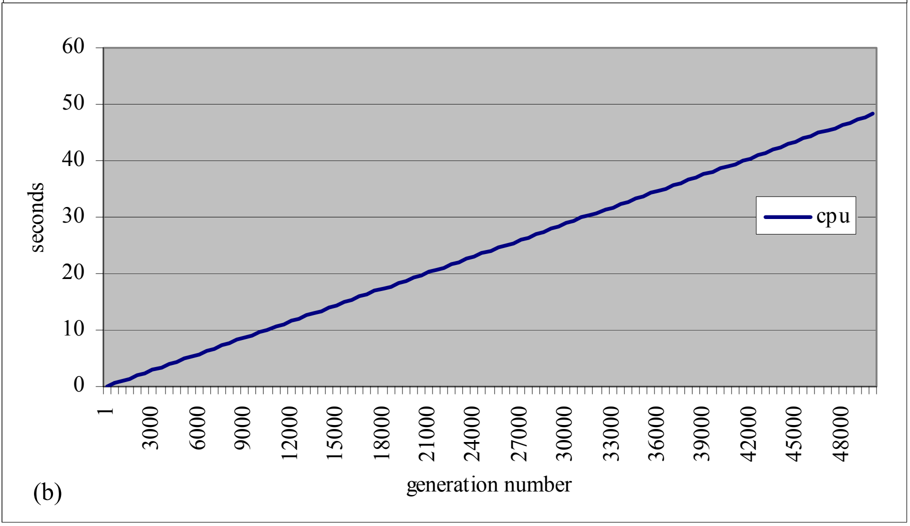


Figure 4.2 (a) FB and FA vs. number of generations and (b) CPU vs. number of generations for REPAIR with ini_{rand} for the large problem instance

4.3 Results for TSP

Although the main purpose of this work is to analyze the performance of various constraint handling techniques for different side constraints on TSP, this effort would never be complete unless the performance of the EA is analyzed for the unconstrained case ("naked" TSP). Therefore our experimentation starts with TSP. When there is no side constraint associated with the problem, differences among the proposed strategies vanish as the crossover operator utilized assures feasibility regarding TSP. Only computational time performances may vary among the algorithms, due to some algorithmic and coding differences. For example, if CONSTRUCT is to be used, the computational time will be more as a feasibility check is done at every node addition. Therefore, any proposed algorithm can be used for observing the performance of EA settings regarding the solution quality. We have chosen the algorithm based on REPAIR for this experimentation. The initial population is generated randomly.

Firstly, the influence of population size is investigated. Two population sizes experimented in Sönmez (2003), i.e., 50 and 100, are experimented for determining a proper population size. For the problems given in Table 4.2, 30 replications are realized for different stopping conditions. For each problem, average, minimum, and maximum values attained and standard deviation observed during replications are recorded. Then, these values are averaged for 20 problems. The results for population size 50 and for population size 100 are given in Table 4.3 and Table 4.4, respectively.

Table 4.3 Performance of EA in TSP when population size is 50

		GEN	FB	FA	IMP_b	IMP_{avg}	CPU	CPU_{ini}	EP	DEV_{opt}
2000	avg	1990.66	2408.46	2441.21	72.01	72.68	0.32	0.00	1.26	2.06
	std	37.06	37.21	31.42	1.90	1.45	0.01	0.01	0.39	0.89
	min	1816.05	2327.05	2360.06	67.55	69.23	0.30	0.00	0.71	0.65
	max	2000.00	2470.50	2499.60	75.40	75.29	0.34	0.02	2.30	4.13
5000	avg	4482.78	2407.15	2425.92	72.03	72.86	0.73	0.00	0.71	1.98
	std	765.26	37.95	31.89	1.90	1.42	0.12	0.01	0.31	0.90
	min	2500.70	2326.60	2355.95	67.57	69.48	0.42	0.00	0.33	0.58
	max	5000.00	2470.25	2483.19	75.43	75.45	0.83	0.02	1.65	4.09
8000	avg	6233.11	2406.25	2423.37	72.03	72.89	1.01	0.00	0.60	1.97
	std	1769.24	38.37	32.48	1.90	1.42	0.29	0.01	0.30	0.90
	min	2650.70	2326.60	2353.54	67.57	69.54	0.43	0.00	0.23	0.58
	max	8000.00	2470.00	2481.92	75.43	75.48	1.32	0.02	1.49	4.06
Fixedbest	avg	9925.13	2399.53	2415.60	72.04	72.95	1.68	0.00	0.53	1.93
	std	4534.58	36.66	34.21	2.01	1.49	0.80	0.00	0.31	0.85
	min	3168.40	2335.70	2343.35	67.42	69.44	0.51	0.00	0.13	0.62
	max	19214.80	2475.25	2479.16	75.73	75.81	3.40	0.02	1.40	3.89

Having noticed that the larger population size give better results, we decided to use a population composed of 100 solutions for analyzing the performance of our EAs on constrained problems.

Table 4.4 The performance of EA in TSP when population size is 100

		GEN	FB	FA	IMP_b	IMP_{avg}	CPU	CPU_{ini}	EP	DEV_{opt}
2000	avg	2000.00	2398.93	2475.16	71.46	72.50	0.52	0.01	2.03	1.64
	std	0.00	32.00	42.43	1.76	1.12	0.01	0.00	0.34	0.54
	min	2000.00	2347.80	2397.26	67.47	70.00	0.51	0.01	1.39	0.79
	max	2000.00	2453.40	2606.75	74.83	74.80	0.53	0.02	2.79	2.84
5000	avg	4985.54	2388.96	2424.69	71.50	72.89	1.28	0.01	1.09	1.42
	std	45.56	31.25	34.47	1.75	1.12	0.01	0.00	0.29	0.50
	min	4804.75	2346.95	2374.39	67.53	70.43	1.25	0.01	0.62	0.66
	max	5000.00	2446.40	2542.97	74.85	75.21	1.30	0.02	1.77	2.49
8000	avg	7845.54	2388.31	2414.15	71.51	72.99	2.01	0.01	0.79	1.39
	std	363.45	31.58	26.77	1.75	1.10	0.07	0.00	0.26	0.50
	min	6559.85	2345.40	2369.48	67.53	70.56	1.71	0.01	0.42	0.64
	max	8000.00	2446.15	2472.42	74.87	75.29	2.05	0.02	1.44	2.44
Fixedbest	avg	15186.93	2383.54	2401.62	71.43	73.02	4.01	0.01	0.53	1.33
	std	3398.09	27.87	24.77	1.85	1.10	0.92	0.00	0.26	0.54
	min	7688.65	2320.80	2348.86	67.24	70.47	1.95	0.01	0.21	0.40
	max	22980.65	2438.20	2449.13	75.01	75.29	6.27	0.02	1.27	2.55

From the reproduction aspect, our algorithms are based on Sönmez (2003); the crossover operator utilized in the study is taken from her work. However, regarding selection, replacement, and generational gap aspects, the algorithms are different. Sönmez's EA is an example of a generational GA. Specifically, it first generates a mating pool by replicating each solution in the population twice. Then, it selects random pairs from the population until all the individuals are mated with another. From each pair one solution is produced. In total, N new solutions are produced, where N is the original population size. The population size is doubled with the addition of these newborns to the population. In order to decrease the population size back to N, all new solutions and individuals in the current population are ordered and worst N individuals are deleted from the population.

Sönmez (2003) reports an average deviation of 3.10% of the population best from the optimal value for the Nearest Neighbor crossover operator. The average computation time is reported to be 0.38 seconds. The experiment was carried on a set of ten problems from TSP library, sizes of which are varying from 52 to 226. Although this result is not directly comparable with our results due to size difference in test beds, it shows that the performance of our EA is at least as good as her algorithm.

Our EA produced better results even for the early stopping conditions: the average deviations of the population best from the optimal value are 1.64, 1.42, and 1.39 for the stopping conditions of 2000 generations, 5000 generations and 8000 generations, respectively. The algorithms of Sönmez converge after producing about 2300 solutions on the average. However, even if we stop at 2000th generation, we produce 4000 solutions. The main reason for our algorithm to give better results may be that increase in the number of solutions produced. Moreover, the steady state replacement scheme, or specific selection and replacement schemes used in our study may be accounted for these better results also. Anyway, we will not concentrate on the goodness of our algorithm. We will just concentrate on the results. Having found that our EA performs well for TSP, we thought that it would be worthwhile to test the performance of handling side constraints on TSPs using our scheme.

4.4 Results for TSPPD

In this section the performance of the algorithms are tested for TSPPD. 30 replications for each problem in the problem set described in Section 4.1 are realized. For each algorithm (i.e., the algorithms based on different strategies), four different stopping conditions (2000, 5000, 8000, and Fixedbest) and three different initial population types (ini_{feas}, ini_{half}, and ini_{rand}) are experimented. In the following subsections, we first give the performance measure values in tables for each strategy. Then the statistical analyses of the two experiment factors are performed for each algorithm separately. Afterwards the proposed algorithms are compared by again statistical means. The significance level is considered to be 0.05 for every statistical analysis.

We want to determine the effect of the initial population types and indicate the favorable types if any. Through this respect, Analysis of Variance (ANOVA) is conducted for the effect of the initial population types on solution quality and time.

This analysis is conducted once for the earliest stopping condition, 2000, and once for the latest stopping condition, Fixedbest, to point on the changes throughout the generations if any.

Having determined 8000 as the appropriate stopping condition, we want to check if there are significant differences among this stopping condition and the others regarding our main performance measures. The main question was "Can we stop earlier without a significant decrease in the solution quality?". Therefore, we first look at the difference between 5000 and 8000 by conducting ANOVA. When we look at the average figures, the minimum difference between two stopping conditions is observed between 5000 and 8000. Hence, once the difference between 5000 and 8000 is found significant, we assume that 8000 is significantly different from the others also.

For both initial population and stopping condition analyses, ANOVA is conducted in the same manner. Normality and residual vs. fit plots are drawn in order to check the validity of the assumptions of ANOVA. The main responses used in the analyses are DEV, regarding solution quality, and CPU, regarding the computation time. However, when the plots are found inappropriate for these responses, logarithms and square roots of these measures are considered in order to conform the assumptions. For the analyses, General Linear Model option of MINITAB 13.32 is utilized. Main fixed factor is initial population type and stopping condition in the first and second analyses. All analyses are realized in randomized block design, where the instances are taken to be the random effects. The ANOVA tables and plots of initial population and stopping condition analyses are provided in Appendix D and Appendix E, respectively.

The main aim of this study is to compare the effect of different strategies. For that purpose another ANOVA with randomized block design is conducted. The algorithms are compared regarding solution quality and time for each stopping condition and for the initial population types determined before.

Stating that algorithms are significantly "different" from each other is not sufficient. Therefore, Tamhane's T2 test, a post-hoc multiple comparison test, which does not assume equal variances for different factor levels, is used for ordering the algorithms. The reader is referred to Toothaker (1993) for a discussion on this test.

After testing this assumption with Levene's test, and observing that in all cases the hypothesis is rejected, we find this type of comparison test to be more appropriate. This comparison is realized in SPSS 12.0 for Windows. Results of the Levene's and Tamhane's tests are given in Appendix F.

4.4.1 Results for REJECT (Strategy-1)

For this strategy, because of its large computation time requirements three stopping conditions and three initial population types are experimented here. Fixedbest stopping condition was not experimented. Results are given in Table 4.5.

When we look at the effect of initial population for 2000 stopping condition, ini_{rand} seems to be worse than the others on the average. However, statistical significance could not be achieved. The p-value for the test (when the response was logarithm of DEV_b) was 0.375, which is far from the standard p-value of 0.05. However, when the response was logarithm of CPU, the p-value decreased to 0.079. Nevertheless it is still more than 0.05. Therefore, the initial population type is deemed not influential on the performance for early termination. The p-values get worse for 8000. Hence, the former decision is valid for this condition too. Note that, the mentioned responses gave the most appropriate residual plots in the tests.

As the initial population type is determined to be not influential to the performance of the algorithm, we combined the data coming from every initial population type for this strategy for comparing this strategy with others. However, we also realized a comparison using only the results of the experiment with ini_{rand} as it is the most basic and easy to generate type.

The difference between stopping conditions is tested for the ini_{feas}. Logarithm of DEV_b is the response in this test. The p-value of the test was 0.011, which implies statistical significance. However, the best normality and residual vs. fitted value plots that could be attained were not very dependable. Therefore, stopping earlier will cause a significant worsening in the solution quality by 0.30%, but this will decrease the computation time required by 5 seconds on the average. In cases where computational time is extremely important, stopping earlier may considered as feasible alternative for this strategy.

Table 4.5 Performance of REJECT for TSPPD

			GEN	FB	FA	IMP_b	IMP_{avg}	CPU	CPU_{ini}	EP	ENUM	C_{del}	DEV_b
2000	ini_{feas}	avg	2000.00	2412.50	2651.10	66.42	70.15	9.43	0.01	3.62	0.32	0.81	5.30
		std	0.00	1.33	72.96	1.27	0.79	3.27	0.01	0.84	0.13	0.17	1.30
	ini_{half}	avg	2000.00	2416.00	2579.80	67.01	70.72	9.33	0.01	3.71	0.33	0.82	5.30
		std	0.00	1.32	83.16	1.37	0.72	3.54	0.01	1.02	0.12	0.18	1.30
	ini_{rand}	avg	2000.00	2412.80	2505.10	70.44	71.30	9.70	0.01	3.52	0.33	0.79	5.40
		std	0.00	1.33	44.30	1.81	1.22	3.09	0.01	0.89	0.13	0.19	1.30
5000	ini_{feas}	avg	5000.00	2404.70	2500.10	66.47	71.41	15.14	0.01	2.31	0.25	0.69	5.00
		std	0.00	1.23	51.22	1.26	0.57	7.47	0.01	0.86	0.13	0.22	1.20
	ini_{half}	avg	5000.00	2408.70	2484.70	67.07	71.54	15.19	0.01	2.42	0.25	0.71	5.10
		std	0.00	1.28	49.97	1.36	0.54	8.03	0.01	1.06	0.12	0.23	1.30
	ini_{rand}	avg	4996.44	2404.40	2466.20	70.49	71.65	15.00	0.01	2.25	0.25	0.68	5.20
		std	19.52	1.24	37.88	1.80	1.21	7.35	0.01	0.92	0.12	0.24	1.20
8000	ini_{feas}	avg	7988.41	2404.00	2466.60	66.48	71.69	19.83	0.01	0.78	0.23	0.64	5.00
		std	52.15	1.23	41.15	1.26	0.43	11.32	0.01	0.33	0.13	0.24	1.20
	ini_{half}	avg	7997.44	2407.80	2461.70	67.08	71.72	20.48	0.01	0.82	0.23	0.65	5.10
		std	14.05	1.26	39.91	1.36	0.46	12.54	0.01	0.37	0.12	0.24	1.20
	ini_{rand}	avg	7978.46	2403.90	2456.60	70.51	71.74	19.53	0.01	0.78	0.23	0.62	5.10
		std	118.01	1.24	35.70	1.80	1.21	11.39	0.01	0.36	0.12	0.26	1.20

Observe that ENUM does not vary among the initial population types even when all of the solutions in the initial population are feasible. This case seems to be the indicator of domination of population by the solutions that can give feasible offspring when NN is applied. That is to say, in the initial steps, it is hard to produce feasible solutions from the random solutions in population. However, as we proceed, the solutions having short "feasible" edges begin to exist in the population, and it becomes easier to find feasible breed from these solutions with NN. The initial effort for generating solutions that can produce feasible solutions relatively easily, can be seen on CPU vs. number of generation graphs provided in Appendix C. As it is given in Figure 4.3, for p15 the rate of change in CPU decreases considerably around 1000th generation. This indicates a decrease in the total infeasible solutions produced per feasible solution per generation. When we stop at 5000 and 8000, the number of infeasible solutions per feasible solution decreases for all of the population types. A similar pattern can be recognized in EP and C_{del}. These measures also decrease as the algorithm runs more. This may indicate that, as we proceed, the children like their parents are produced in this strategy. Recall that if a child with the same value of its parent is produced it is accepted to enter the population.

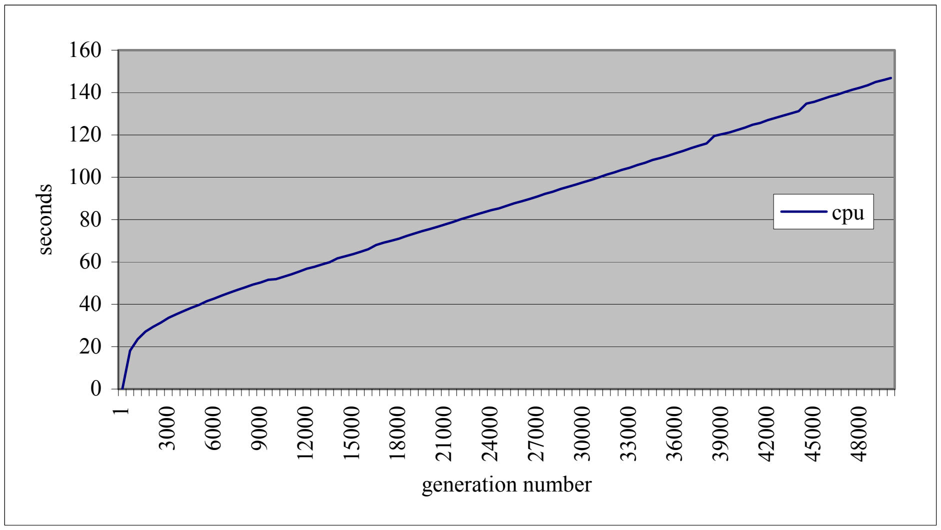


Figure 4.3 CPU vs. number of generations for REJECT in ini_{feas} for p15

4.4.2 Results for CONSTRUCT (Strategy-2)

There is no statistical evidence that the initial population types differ in performance when the stopping condition is 2000 for CONSTRUCT. When we used Fixedbest as the stopping condition, a close to significant p-value of 0.060 is observed for DEV_b. In this stopping condition ini_{rand} gives slightly better results regarding solution quality. However, at the end we state that the initial population type is insignificant for this algorithm. Results for CONSTRUCT algorithm is provided in Table 4.6.

Table 4.6 Performance of CONSTRUCT for TSPPD

			GEN	FB	FA	IMP_b	IMP_{avg}	CPU	EP	C_{del}	DEV_b
2000	ini_{feas}	avg	2000	2435.9	2775.3	66.35	68.57	0.57	11.7	1.55	5.74
		std	0	34.44	86.83	1.33	0.76	0.01	0.34	0.02	1.04
	ini_{half}	avg	2000	2441.6	2715.7	67.01	69.24	0.57	11.7	1.55	5.75
		std	0	36.28	78.71	1.39	0.76	0.01	0.35	0.02	1.06
	ini_{rand}	avg	2000	2442.2	2623.9	70.46	69.94	0.57	11.7	1.55	5.85
		std	0	25.35	35.91	1.87	1.26	0.01	0.33	0.02	1.01
5000	ini_{feas}	avg	5000	2411.3	2607.2	66.59	70.09	1.42	11.62	1.67	4.7
		std	0	30.23	57.54	1.3	0.54	0.01	0.41	0.03	0.83
	ini_{half}	avg	5000	2420.1	2599.6	67.24	70.26	1.42	11.59	1.67	4.76
		std	0	26.64	45.61	1.37	0.5	0.01	0.44	0.03	0.77
	ini_{rand}	avg	5000	2422.7	2578.9	70.67	70.42	1.42	11.6	1.67	4.77
		std	0	25.48	31.94	1.85	1.25	0.01	0.42	0.02	0.81
8000	ini_{feas}	avg	8000	2406.4	2572	66.66	70.44	2.24	11.57	1.7	4.4
		std	0	29.16	34.42	1.29	0.41	0.01	0.44	0.03	0.77
	ini_{half}	avg	8000	2412.3	2576.3	67.31	70.48	2.24	11.53	1.7	4.43
		std	0	27.64	31.36	1.36	0.42	0.01	0.46	0.03	0.71
	ini_{rand}	avg	8000	2417.7	2572.6	70.73	70.73	2.24	11.55	1.71	4.44
		std	0	25.5	21.44	1.85	1.6	0.01	0.45	0.03	0.72
Fixedbest	ini_{feas}	avg	25246.2	2398.8	2556.4	66.74	70.63	8.59	11.37	1.74	3.91
		std	7236.59	25.37	27.59	1.36	0.31	2.73	0.46	0.03	0.66
	ini_{half}	avg	24776.7	2391.8	2544.6	67.37	70.64	8.3	11.39	1.74	3.86
		std	6849.74	29.25	30.42	1.31	0.36	2.59	0.48	0.03	0.67
	ini_{rand}	avg	24951.1	2392.4	2555.4	70.94	70.68	8.19	11.37	1.75	3.82
		std	6451.1	20.22	25.66	1.75	1.16	2.31	0.46	0.03	0.58

The difference between the stopping conditions is tested for the ini_{rand}. If we decide to stop earlier, the solution quality gets worse by 0.3%, and this decrease is statistically significant. As the difference between 5000 and 8000 in computation time is only 1 second, waiting for additional generations is an appropriate way. In fact, when we stop with Fixedbest, we wait for 6 more seconds on the average, and achieve a decrease about 0.5-0.6%. Stopping with Fixedbest takes less than 10 seconds for this strategy, which is equal to the time required by stopping with 2000 at strategy-1. Therefore stopping with Fixedbest is also a feasible alternative. Differences between 5000 and 8000 are reported significant, but the best plots are doubtful about the validation of the assumptions.

Looking at the overall, the algorithm produces competitive results. On the average, values deviate 4.4% from TSP optimal values. Recalling the difference between TSP and TSPPD optimal for 10 small problems (i.e., 2.33%) it can be assumed that the solutions deviate from optimal values around 2 and 2.5% on the average.

There is a slight decrease in EP values and a considerable increase in C_{del}, as the number of generations increases. The increase in EP is expected, since good solutions dominate the population as the number of generation increases. However, as the parents get better it is harder to find better children. Its reason could be the edge added between the first visited node (last visited for the second child) and the depot. Recall that this edge is necessary to be able to produce children different from their parents.

4.4.3 Results for REPAIR (Strategy-3)

Initial population type is not influential. For the stopping condition analysis, the results of ini_{rand} are utilized. There is no significant difference between 5000 and 8000 as a result of the analysis. Although statistical tests could not differentiate the solution quality of these two stopping conditions, average falls by 0.2%. If we are looking for computational efficiency we can stop at 5000, which improves the time by 1 second on the average. However, Fixedbest produces better results than 5000. The results of our third strategy are provided in Table 4.7.

Table 4.7 Performance of REPAIR for TSPPD

			GEN	FB	FA	IMP_b	IMP_{avg}	CPU	CPU_{ini}	EP	LOC	TOTREP	C_{del}	DEV_b
2000	ini_{feas}	avg	2000.00	2447.09	2754.32	66.49	69.11	0.55	0.01	3.08	2.91	2931.85	1.22	4.91
		std	0.00	33.61	95.14	1.39	1.44	0.01	0.01	0.38	0.1	189.34	0.09	0.94
	ini_{half}	avg	2000.00	2448.88	2668.51	67.22	69.75	0.55	0.01	3.04	2.91	2929.75	1.23	4.96
		std	0.00	35.61	72.00	1.47	0.76	0.01	0.01	0.38	0.1	187.32	0.09	0.94
	ini_{rand}	avg	2000.00	2447.91	2601.47	70.53	70.34	0.55	0.01	3.05	2.91	2939.45	1.23	4.87
		std	0.00	31.63	39.27	1.74	1.22	0.01	0.01	0.36	0.09	187.27	0.08	0.85
5000	ini_{feas}	avg	5000.00	2429.94	2600.30	66.47	70.36	1.34	0.01	2.15	2.9	6689.86	1.18	4.48
		std	0.00	32.82	52.03	1.43	1.17	0.01	0.01	0.33	0.08	500.71	0.12	0.82
	ini_{half}	avg	5000.00	2418.33	2562.94	67.26	70.55	1.34	0.01	2.13	2.9	6635.42	1.19	4.33
		std	0.00	34.89	41.30	1.35	0.54	0.01	0.01	0.35	0.1	504.53	0.13	0.87
	ini_{rand}	avg	5000.00	2431.11	2562.94	70.75	70.68	1.34	0.01	2.13	2.9	6662.52	1.19	4.38
		std	0.00	35.52	36.70	1.74	1.22	0.01	0.01	0.35	0.09	501.19	0.12	0.85
8000	ini_{feas}	avg	8000.00	2429.81	2567.41	66.79	70.67	2.12	0.01	1.87	2.89	10330.96	1.15	4.21
		std	0.00	32.4	45.16	1.26	1	0.02	0.01	0.34	0.08	829.35	0.13	0.79
	ini_{half}	avg	8000.00	2420.82	2549.41	67.38	70.71	2.12	0.01	1.87	2.89	10349.78	1.16	4.13
		std	0.00	31.53	35.81	1.28	0.45	0.02	0.01	0.35	0.1	885.56	0.14	0.85
	ini_{rand}	avg	8000.00	2429.61	2558.23	70.52	70.56	2.12	0.01	1.84	2.89	10304.77	1.16	4.31
		std	0.00	28.03	31.04	1.78	1.3	0.02	0.01	0.32	0.09	815.82	0.12	0.76
Fixedbest	ini_{feas}	avg	21610.23	2422.13	2542.03	66.78	70.78	6.42	0.01	1.58	2.87	28268.92	1.14	3.94
		std	5445.37	33.02	42.05	1.3	1.25	1.95	0.01	0.3	0.09	8923.4	0.14	0.71
	ini_{half}	avg	20983.85	2417.24	2523.15	67.29	70.79	6.2	0.01	1.57	2.87	27159.26	1.13	3.93
		std	5125	38.42	33.47	1.36	0.4	1.84	0.01	0.28	0.12	7678.23	0.14	0.67
	ini_{rand}	avg	21097.46	2421.65	2534.78	70.87	70.81	6.32	0.01	1.57	2.87	27357.31	1.15	3.96
		std	4871.88	32.36	34.98	1.74	1.19	1.72	0.01	0.31	0.11	7446.28	0.14	0.77

The results of this strategy is promising like the previous one. On the average solutions deviate by 4.2% from the TSP optimal. The computation time is again reasonably less compared to REJECT strategy.

The average value for LOC is around 3, which shows that taking the best location is an appropriate repairment procedure. Referring to TOTREP we can say in general, 70% of the population is repaired at the early stages of the algorithm. This number decreases as the number of generations increases, which indicates that the probability of obtaining feasible solutions increases as the better feasible solutions increases in the population.

4.4.4 Results for PEN_REPAIR (Strategy-4)

The effect of the initial population types resembles to REPAIR algorithm. No evidence could be obtained for stating that the initial population types differ widely for solution quality and CPU. Again the difference between the stopping conditions 5000 and 8000 is significant with a p-value of 0.000 when the results of ini_{rand} are used. The time can be improved by approximately 2.5 second, giving up 0.15% of the solution quality on the average by stopping at 5000 instead of 8000. If we stop with Fixedbest condition, improvements around 0.2% - 0.3% can be obtained at the expense of approximately 3.5 seconds. This stopping condition can be utilized where solution quality is more important. The results of PEN_REPAIR are given in Table 4.8.

As the number of generations increases EP and C_{del} decrease. On the average, 25% of the population is composed of the feasible solutions throughout the entire evolution. The number of the feasible solutions differs for initial population types for earlier stopping conditions. However, this quantity is equalized for all types in longer runs.

4.4.5 Results for PEN_ADAPT (Strategy-5)

For all of the stopping conditions the **iternum** is taken as 8000. Therefore the cooling parameter λ equals to **n**/16000, which decreases **NFT** in a fairly slow rate.

The influence of the initial population type for PEN_ADAPT has the most interesting pattern. When we stop the algorithm early, the difference between the initial population types regarding both solution quality and computation time is significant. When we look at the averages, ini_{rand} gives the smallest deviation and the largest computation time. ini_{feas} is the worst at solution quality and the best at the computation time.

If we wait more and stop with Fixedbest criterion, the statistical significance vanishes regarding solution quality, but remains intact regarding the computation time. In this case, ini_{rand} gives the worst and ini_{feas} gives the best average values indicating solution quality. The ordering in the computation time remains the same. For the analyses of stopping condition, the results of ini_{rand} were used. Like in many cases, the difference between the stopping conditions 5000 and 8000 regarding the solution quality is significant. Results of PEN_ADAPT are given in Table 4.9.

EP is relatively higher at the initial stopping condition, but decreases as the number of generations increases. The average number of feasible solutions is higher than the previous penalizing algorithm. For early stopping conditions, there is a considerable difference among the initial population types, which vanishes in the later stopping conditions, for this quantity. On the average, 70% of the population is feasible throughout the entire evolution process. In fact, this value is far from the value of the other algorithm.

When we look at the solution qualities of these two strategies, the benefit that can be gained from these feasible individuals are questionable. The strategy give solutions deviating between 6.3% - 6.5% from the TSP optimal on the average even for the latest stopping condition. This point indicates that an inappropriate penalizing scheme is utilized. Most probably, the infeasible solutions, which are penalized according to the constraint violation values, are harshly penalized. However, milder penalizing schemes did not yield better results in preliminary experiments.

Table 4.8 Performance of PEN_REPAIR for TSPPD

			GEN	FB	FA	FFEAS	IMP_b	IMP_{avg}	IMP_{feas}	CPU	CPU_{ini}	EP	C_{del}	F_{avg}	DEV_b
2000	ini_{feas}	avg	2000.00	2396.42	2654.60	2428.97	66.26	69.88	65.76	0.51	0.01	1.52	1.16	28.38	5.28
		std	0.00	33.47	74.14	44.77	1.25	0.79	1.38	0.01	0.01	0.20	0.11	6.55	1.23
	ini_{half}	avg	2000.00	2397.17	2636.74	2423.65	66.01	69.72	65.69	0.51	0.01	1.53	1.16	25.93	5.24
		std	0.00	32.96	88.70	42.92	1.25	0.76	1.36	0.01	0.01	0.20	0.10	5.75	1.21
	ini_{rand}	avg	2000.00	2387.08	2658.70	2413.51	65.94	69.58	65.74	0.51	0.01	1.52	1.15	23.78	5.17
		std	0.00	29.39	65.65	43.69	1.15	0.82	1.25	0.01	0.01	0.20	0.11	5.49	1.24
5000	ini_{feas}	avg	5000.00	2386.20	2494.57	2414.58	66.33	71.28	65.88	1.23	0.01	0.99	1.10	25.73	4.76
		std	0.00	30.66	50.44	38.00	1.24	0.56	1.36	0.01	0.01	0.19	0.14	7.07	1.10
	ini_{half}	avg	4999.03	2382.88	2491.09	2410.27	66.09	71.16	65.83	1.23	0.01	1.00	1.10	24.45	4.65
		std	5.32	33.55	51.40	44.90	1.24	0.55	1.34	0.01	0.01	0.20	0.13	6.50	1.11
	ini_{rand}	avg	4998.48	2372.47	2480.96	2395.59	66.01	71.04	65.85	1.23	0.01	0.99	1.11	23.87	4.65
		std	7.93	27.32	51.73	38.83	1.15	0.59	1.24	0.01	0.01	0.19	0.14	6.59	1.14
8000	ini_{feas}	avg	7990.94	2382.96	2456.48	2408.30	66.36	71.58	65.93	1.95	0.01	0.82	1.06	24.63	4.58
		std	34.42	32.33	44.56	39.55	1.24	0.45	1.35	0.02	0.01	0.19	0.15	7.05	1.04
	ini_{half}	avg	7983.73	2377.30	2462.27	2405.40	66.12	71.45	65.87	1.95	0.01	0.82	1.06	23.72	4.50
		std	48.05	36.19	44.71	42.59	1.24	0.45	1.32	0.02	0.01	0.19	0.14	6.80	1.05
	ini_{rand}	avg	7977.36	2370.95	2442.71	2392.38	66.03	71.33	65.89	1.95	0.01	0.82	1.08	23.75	4.49
		std	76.28	28.39	42.90	39.48	1.15	0.48	1.24	0.02	0.01	0.19	0.15	7.01	1.14
Fixedbest	ini_{feas}	avg	19947.64	2373.03	2439.00	2395.20	66.71	71.72	65.88	5.39	0.01	0.59	0.85	23.24	4.33
		std	4669.88	30.81	37.72	34.70	1.34	0.38	1.41	1.39	0.01	0.17	0.16	7.11	1.03
	ini_{half}	avg	20264.38	2368.78	2431.91	2390.84	66.53	71.61	65.88	5.43	0.01	0.60	0.84	23.34	4.29
		std	5223.70	31.86	37.02	36.77	1.27	0.35	1.34	1.55	0.01	0.18	0.16	7.34	1.01
	ini_{rand}	avg	20429.54	2373.76	2435.82	2395.22	66.41	71.49	65.95	5.67	0.01	0.60	0.85	24.69	4.20
		std	5242.11	27.90	32.75	29.56	1.26	0.38	1.26	1.67	0.01	0.17	0.17	8.23	0.93

Table 4.9 Performance of PEN_ADAPT for TSPPD

			GEN	FB	FA	FFEAS	$\mathrm{IMP_b}$	$\mathrm{IMP_{avg}}$	$\mathrm{IMP_{feas}}$	fall	CPU	$\mathrm{CPU_{ini}}$	EP	$\mathrm{C_{del}}$	$\mathrm{F_{avg}}$	$\mathrm{DEV_b}$
2000	$\mathrm{ini_{feas}}$	avg	2000.00	2497.35	3035.70	2503.88	64.36	62.88	64.04	2437.87	0.97	0.01	3.46	1.53	70.09	16.63
		std	0.00	78.54	222.54	78.37	1.96	3.34	1.99	51.66	0.13	0.01	0.33	0.11	8.23	6.05
	$\mathrm{ini_{half}}$	avg	1999.64	2497.91	2996.49	2507.56	64.68	64.67	65.00	2437.10	1.39	0.01	3.34	1.49	54.55	14.10
		std	1.98	73.18	216.74	77.04	1.97	2.37	2.09	47.77	0.11	0.01	0.33	0.13	7.58	5.35
	$\mathrm{ini_{rand}}$	avg	1999.32	2459.47	2804.52	2470.94	65.24	67.33	68.83	2420.86	1.69	0.01	2.95	1.42	39.47	10.59
		std	2.89	51.82	122.77	57.02	1.51	1.28	2.08	35.17	0.13	0.01	0.29	0.12	9.14	3.54
5000	$\mathrm{ini_{feas}}$	avg	4953.29	2433.31	2563.33	2438.58	65.46	69.76	65.17	2398.80	2.44	0.01	2.21	1.31	69.19	7.25
		std	78.06	55.56	110.08	59.66	1.52	1.20	1.56	38.20	0.44	0.01	0.37	0.16	10.13	2.37
	$\mathrm{ini_{half}}$	avg	4953.19	2420.34	2536.13	2426.42	65.42	70.03	65.82	2390.30	2.98	0.01	2.12	1.28	61.68	6.88
		std	82.21	52.69	96.71	53.64	1.53	0.96	1.53	33.07	0.39	0.01	0.37	0.17	9.92	2.17
	$\mathrm{ini_{rand}}$	avg	4936.36	2442.14	2555.22	2449.78	65.58	70.24	69.46	2410.13	3.23	0.01	1.76	1.25	56.88	7.13
		std	109.72	46.06	79.22	47.16	1.48	0.80	2.16	33.54	0.44	0.01	0.37	0.18	11.05	2.20
8000	$\mathrm{ini_{feas}}$	avg	7736.46	2426.67	2513.13	2436.11	65.72	70.68	65.42	2393.36	3.75	0.01	1.56	1.20	70.24	6.46
		std	379.07	56.29	93.63	59.48	1.43	0.73	1.47	39.68	0.75	0.01	0.36	0.19	11.71	2.01
	$\mathrm{ini_{half}}$	avg	7794.34	2446.26	2533.45	2450.87	65.56	70.65	66.01	2397.63	4.24	0.01	1.56	1.21	67.18	6.54
		std	248.57	66.07	96.79	67.60	1.43	0.72	1.41	41.54	0.69	0.01	0.37	0.19	11.11	2.07
	$\mathrm{ini_{rand}}$	avg	7694.83	2445.50	2529.18	2451.49	65.57	70.70	69.36	2403.93	4.38	0.01	1.25	1.15	63.84	6.66
		std	250.41	54.10	72.99	53.82	1.38	0.61	2.05	40.05	0.71	0.01	0.32	0.17	9.41	1.99
Fixedbest	$\mathrm{ini_{feas}}$	avg	17730.58	2435.67	2500.97	2440.04	65.62	70.94	65.34	2391.36	8.12	0.01	0.80	1.07	76.27	6.31
		std	3822.52	47.44	72.46	46.65	1.37	0.60	1.41	35.94	2.40	0.01	0.30	0.20	12.02	1.84
	$\mathrm{ini_{half}}$	avg	17569.07	2427.05	2468.14	2431.65	65.60	70.96	65.95	2397.52	8.64	0.01	0.79	1.06	74.40	6.35
		std	3683.13	48.55	67.84	49.15	1.38	0.58	1.60	36.14	2.35	0.01	0.28	0.21	11.53	1.97
	$\mathrm{ini_{rand}}$	avg	17155.93	2429.65	2483.31	2437.41	65.51	70.95	69.48	2395.15	8.72	0.01	0.67	1.04	73.60	6.53
		std	3665.81	53.89	69.63	56.68	1.44	0.55	1.98	37.41	2.31	0.01	0.26	0.20	11.28	2.07

4.4.6 Comparison of Strategies

The overall results of computation time and solution quality for all algorithms based on different strategies are provided in Table 4.10

Table 4.10 Overall results for TSPPD[*]

			REJECT S-1		CONSTRUCT S-2		REPAIR S-3		PEN_REPAIR S-4		PEN_ADAPT S-5	
			CPU	DEV_b	CPU	DEV_b	CPU	DEV_b	CPU	DEV_b	CPU	DEV_b
2000	ini_{feas}	avg	9.43	5.30	0.57	5.74	0.55	4.91	0.51	5.28	0.97	16.63
		std	3.27	1.30	0.01	1.04	0.01	0.94	0.01	1.23	0.13	6.05
	ini_{half}	avg	9.33	5.30	0.57	5.75	0.55	4.96	0.51	5.24	1.39	14.1
		std	3.54	1.30	0.01	1.06	0.01	0.94	0.01	1.21	0.11	5.35
	ini_{rand}	avg	9.7	5.40	0.57	5.85	0.55	4.87	0.51	5.17	1.69	10.59
		std	3.09	1.30	0.01	1.01	0.01	0.85	0.01	1.24	0.13	3.54
5000	ini_{feas}	avg	15.14	5.00	1.42	4.70	1.34	4.48	1.23	4.76	2.44	7.25
		std	7.47	1.20	0.01	0.83	0.01	0.82	0.01	1.1	0.44	2.37
	ini_{half}	avg	15.19	5.10	1.42	4.76	1.34	4.33	1.23	4.65	2.98	6.88
		std	8.03	1.30	0.01	0.77	0.01	0.87	0.01	1.11	0.39	2.17
	ini_{rand}	avg	15	5.20	1.42	4.77	1.34	4.38	1.23	4.65	3.23	7.13
		std	7.35	1.20	0.01	0.81	0.01	0.85	0.01	1.14	0.44	2.2
8000	ini_{feas}	avg	19.83	5.00	2.24	4.40	2.12	4.21	1.95	4.58	3.75	6.46
		std	11.32	1.20	0.01	0.77	0.02	0.79	0.02	1.04	0.75	2.01
	ini_{half}	avg	20.48	5.10	2.24	4.43	2.12	4.13	1.95	4.50	4.24	6.54
		std	12.54	1.20	0.01	0.71	0.02	0.85	0.02	1.05	0.69	2.07
	ini_{rand}	avg	19.53	5.10	2.24	4.44	2.12	4.31	1.95	4.49	4.38	6.66
		std	11.39	1.20	0.01	0.72	0.02	0.76	0.02	1.14	0.71	1.99
Fixedbest	ini_{feas}	avg	-	-	8.59	3.91	6.42	3.94	5.39	4.33	8.12	6.31
		std	-	-	2.73	0.66	1.95	0.71	1.39	1.03	2.4	1.84
	ini_{half}	avg	-	-	8.3	3.86	6.2	3.93	5.43	4.29	8.64	6.35
		std	-	-	2.59	0.67	1.84	0.67	1.55	1.01	2.35	1.97
	ini_{rand}	avg	-	-	8.19	3.82	6.32	3.96	5.67	4.2	8.72	6.53
		std	-	-	2.31	0.58	1.72	0.77	1.67	0.93	2.31	2.07

[*]S-i: strategy i, where i=1,...,5

After analyzing the influence of the experiment factors, we have determined a basis for a fair comparison of these algorithms. For some cases, the random initial population showed a slight difference but in most cases algorithms deemed insensitive to initial population. Therefore, first we have compared the algorithms using the data coming from the ini_{rand} experiments, as this is the initial population type easiest to generate. For a second look, we used the whole data including all initial population types. For each stopping condition, ANOVA is conducted, followed by a post-hoc multiple level comparison test, namely Tamhane's T2 test.

Specifically, this test conducts pairwise comparisons with a determined p-value and constructs indifference groups. However, in order to compute p-value of the overall group, the effect of individual grouping should be considered at the same time. That is to say, $(1-p_{overall}) = (1-p_1)* (1-p_2)…* (1-p_n)$, where n is the total number of comparisons realized and p_i is the p value for the ith comparison. The conducted tests and the resulting tables are given in Appendix F.

The resulting indifference groups are summarized in Table 4.11. The numbers in cells are the strategy numbers. The groups are ordered according to the superiority of the performance measure. In order to explain the meaning of these groups, the grouping for the stopping condition 2000 regarding DEV_b is described. The first group is composed of strategies 3 and 4; the second group is composed of strategies 4 and 1, and so on. This first two cells of the first row tell that strategy 4 did not yield significantly different solutions from 3 and 1, but 3 is significantly better than 1 at a significance level of 0.05

Table 4.11 Indifference groups constructed by the result of the Tamhane's T2 test.

		$ini_{feas} + ini_{half} + ini_{rand}$				ini_{rand}		
		1st group	2nd group	3rd group	4th group	1st group	2nd group	3rd group
Stopping Condition 2000	DEV_b	3-4	4-1	1-2	5	3-4-1	4-1-2	5
	CPU	3-4-2	5	1	-	3-4-2	5	1
Stopping Condition 5000	DEV_b	3-4-2	2-1	5	-	3-4-2	4-2-1	5
	CPU	5	3-2-4	1	-	5	3-2-4	1
Stopping Condition 8000	DEV_b	3-2-4	1	5	-	3-4-2	4-2-1	5
	CPU	5	3-2-4	1	-	5	3-2-4	1
Stopping Condition Fixedbest	DEV_b	2-3-4	5	-	-	2-3-4	5	-
	CPU	3-4	2	5	-	4-3	2	5

For all of the tests, the overall p value is smaller than 0.05. The overall p values are computed for the above example as follows:

$$(1-p_{overall}) = (1-p_{3-1})* (1-p_{3-2})* (1-p_{3-5})* (1-p_{4-2})* (1-p_{4-5})* (1-p_{1-5})$$

In the equation, p_{i-j} is the p value for comparing strategies i and j. Referring to the p values provided in appendices, the overall p for this indifference group is computed. In the tableau, the p values are displayed to three significant digits, this rounding may cause underestimating the overall p value, therefore p values displayed as 0.000 are considered to be 0.0005. All of the p values except p_{2-4} were 0.000 for this group and p_{2-4} is 0.001. With these input, the $p_{overall}$ is computed to be 0.004, which indicates significance for the whole group. The actual p-values for the pairwise comparisons can be found in Appendix F.

In general, a few differences regarding the relative orders of algorithms are observed between all initial types and random initial type only. However, differentiation power of the first is more than the second one as more data points are used. Therefore, the discussion is done considering all initial population types. When we look at the solution quality, strategy-3 (REPAIR) is always in the first group for all stopping conditions. The strategy-2 (CONSTRUCT) is relatively poorer in earlier stops, however, performs quite well at the later stopping conditions. The 4th strategy (PEN_REPAIR) is also among the good ones. 1st and 5th strategies are not performing well relative to others. The difference between strategy-2, strategy-3, and strategy-4 is not significant for all stopping conditions except for Fixedbest. Strategy-2 can find better solutions but requires more number of generations, i.e., its convergence (or getting close to convergence in our case) is slower than the others. The GEN values for the Fixedbest stopping conditions support our argument, where strategy-2 terminates at 25000th generation on the average, while the other strategies halts running around 20000th generation on the average.

Computation time values are also fair for strategy-2, 3, and 4. Strategy-5 performs quickly at the stopping condition 8000, but its performance gets worse for Fixedbest. Strategy-1 is not a good alternative both regarding solution quality and computational effort. As conclusion, we can say that strategy-2, 3, and 4, performed well compared to others.

4.4.7 Comparison with the Work of Gendreau et al.

In the previous section, strategy-2, strategy-3, and strategy-4 are found promising to solve TSPPD. However, their performances should be compared with the previous works reported in the literature in terms of solution quality and time requirement. One prominent work proposing heuristic techniques for TSPPD is the work of Gendreau et al. (1999). They proposed three algorithms one of which is conventional (HI) and the other two are Tabu Search applications (TS1 and TS2). The algorithms reported to give better results relative to the previous heuristics. The detailed description of the heuristics is provided in Chapter 2.

The authors experimented their heuristics on three different test beds. The first set is composed of the problems taken from the VRP literature, sizes of which are varying between 6 and 261. The distance matrices of all instances are used without any change in our experiments. For a customer **i,** the quantity demanded in VRP instance ($\mathbf{q_i}$) is taken as the quantity of the delivery load ($\mathbf{d_i}$) for TSPPD instance. For determining the quantity of pickup loads ($\mathbf{p_i}$) for each customer, the authors used the following equation:

$$\mathbf{p_i} = \begin{cases} \lfloor (1-\beta) * \mathbf{q_i} \rfloor & \text{if i is even} \\ \lfloor (1+\beta) * \mathbf{q_i} \rfloor & \text{if i is odd} \end{cases} \qquad i = 1, \ldots, n,$$

where $\boldsymbol{\beta}$ is the parameter determining the largeness of the net demand of a customer, and n is the problem size. $\boldsymbol{\beta}$ is reported to be a real non-negative value smaller than 1. Note that, if the index of a customer is even, the net demand of the customer ($\mathbf{d_i} - \mathbf{p_i}$) will be non-positive, which makes the customer a delivery customer. Whereas, customers with even index values will be pickup customers.

The parameter $\boldsymbol{\beta}$ deserves additional interest. When $\boldsymbol{\beta}$ is equal to 0.00, since $\mathbf{p_i}$ and $\mathbf{d_i}$ values will be equal for every customer, the resulting instance is an instance of TSP. As the value of $\boldsymbol{\beta}$ increases, quantity of net demand of each customer increases and increases in the variety of the loads makes the instance harder to solve. When $\mathbf{q_i}$'s are all unity in the original instance, the corresponding TSPPD instance will not have any customers with negative net demand. This time the instance becomes a TSP instance also.

In the work of Gendreau et al. (1999), four different **β** values, 0.00, 0.05, 0.10, 0.20, are experimented. We have experimented our heuristics with these **β** values on the test bed defined. For two instances, all of the demand quantities were 1. Therefore, these instances are excluded from our experiments with **β** values different than 0. In experiments, we have realized 30 replications for each instance and **β** value pair. The random initial population is utilized and Fixedbest stopping condition is used.

The average results of our heuristics are provided together with the results of Gendreau et al. (1999) in Table 4.12. The results show that for higher **β** values, our heuristics find better values than Gendreau et al.'s heuristics. However, for **β** = 0.00, our heuristics were generally worse than their TS heuristics. Our heuristics seem to require less computation time, but the difference between the computer properties of experiment environments should not be forgotten. The experiments of Gendreau et al. were realized on a PC486/66 while ours done on a Pentium 4, 1.6 GHz processor.

Table 4.12 Results of the heuristics of Gendreau et al. and our EAs

		CONSTRUCT	REPAIR	PEN_REPAIR	HI	TS1	TS2
$\beta = 0.00$	DEV_{tsp}	2.70	1.35	1.37	5.00	1.10	0.70
	CPU	9.71	5.84	5.72	0.32	24.63	99.69
$\beta = 0.05$	DEV_{tsp}	3.65	2.00	2.07	7.90	5.00	3.60
	CPU	10.16	8.04	9.15	0.18	16.11	68.20
$\beta = 0.10$	DEV_{tsp}	4.22	2.30	2.41	8.40	5.80	4.50
	CPU	10.36	8.53	8.70	0.14	14.44	71.66
$\beta = 0.20$	DEV_{tsp}	4.62	3.56	2.85	10.10	7.20	6.30
	CPU	11.36	10.48	8.74	0.13	15.87	69.78

The comparison revealed a counter result to the general view about the superiority of TS and Simulated Annealing over EA for routing problems. In fact, our EAs can find better results than TS heuristics of Gendreau et al. (1999) for constrained problems. Although this finding is insufficient to derive general conclusions about comparison of different metaheuristic techniques, it surely provides a particular important example where EA performs better than TS.

4.4.8 Comparison with Nearest Neighbor Heuristic with Repair

The benefit of our metaheuristic can be observed when we look at the results of the conventional heuristic results for TSPPD. The heuristic applied here is typically the Nearest Neighbor Heuristic appended by a repair algorithm. In this heuristic, for a problem, starting from different nodes, N solutions are produced by NN, where N is the size of the problem. Then, all of these solutions are repaired with the repair algorithm utilized in this work. The best solution found is then reported. In Table 4.13, results for the problem set are provided. In the table, "Best" represents the best solution found by the heuristic.

Table 4.13 Results of the NN with repair heuristic for TSPPD

	Best	CPU	DEV_b
p00	344.00	0.00	17.01
p01	567.00	0.02	22.73
p02	545.00	0.00	17.71
p03	701.00	0.02	19.83
p04	761.00	0.00	16.90
p05	821.00	0.03	30.11
p06	105.00	0.00	8.25
p07	69.00	0.00	25.45
p08	109.00	0.00	2.83
p09	4999.00	0.00	12.82
p10	570.00	0.00	17.77
p11	495.00	0.00	23.13
p12	864.00	0.05	6.67
p13	678.00	0.00	12.62
p14	293.00	0.00	16.27
p15	955.00	0.03	19.97
p16	850.00	0.02	19.89
p17	42949.00	0.02	28.01
p18	621.00	0.00	21.53
p19	390.00	0.00	5.41
	avg	0.01	17.25
	std	0.01	7.39

The results were very poor regarding the solution quality. However, the computation time is almost negligible. Results for our best strategies with the earliest stopping condition were around 5% of the TSP bound, which is quite less than the results of the conventional heuristic. The additional computation time required is around 5-6 seconds, however, it does worth the additional effort, as the improvement in solution quality is drastic.

4.5 Results for TSPB

The experiment methodology in the previous section is followed for TSPB too. The same problem parameters and same stopping conditions are tried. The statistical analyses are carried on similarly to those of TSPPD. The ANOVA tables and plots related with the effect of the initial population type and the stopping condition on the algorithm's performance are provided in Appendix G and Appendix H, respectively.

4.5.1 Results for REJECT (Strategy-1)

From the TSPPD results we know that this strategy requires relatively more computational effort than the other strategies. As shown in Chapter 3, the solution space of TSPB is generally a subspace of TSPPD. Therefore, obtaining feasible solutions without any intervention is less likely. When we start the experimentation of this strategy, we determined to generate 30 replications for all problems in test bed for each stopping conditions. However, due to large computational time required for the experiments, we have narrowed our attention to a smaller test bed composed of the smallest 10 problems, and only 20 replications are realized here. Table 4.14 summarizes the results for these 10 problems for only 2000 stopping condition.

Table 4.14 Performance of REJECT for TSPB

			GEN	FB	FA	IMP_b	IMP_{avg}	CPU	CPU_{ini}	EP	ENUM	DEV_{opt}
2000	ini_{feas}	avg	2000.00	1013.17	1099.96	44.66	51.37	9.18	0.00	0.87	2.00	3.24
		std	0.00	18.01	21.66	2.34	0.97	2.23	0.01	0.23	0.51	1.89
	ini_{half}	avg	2000.00	1021.56	1096.78	45.69	51.62	27.35	0.00	0.77	6.02	4.05
		std	0.00	23.76	24.30	2.51	1.20	5.65	0.01	0.26	1.25	2.31
	ini_{rand}	avg	No feasible solution obtained									
		std										

CPU figures are extremely large with respect to the results of 2000 stopping condition of TSPPD. ENUM results point to another important fact: on the average the number of infeasible solutions produced per feasible child is 2 times the problem size, when we start with all feasible population. This number increases to 6 times the problem size when there are also infeasible solutions in the initial population. When we start with a random population, no feasible offspring could be produced and the algorithm halts running at the first generation, which is another drawback of this strategy.

The results of this strategy are compared with the results of the other strategies for the smallest 10 problems. In Table 4.15, the summary of the results for all strategies on this small test bed is provided. Considering relatively large computational requirements for this strategy and the poor solution quality obtained with respect to CONSTRUCT, REPAIR, and PEN_REPAIR, this strategy was deemed to be a poor one. Therefore, this strategy is taken out of consideration for the following analysis.

Table 4.15 Comparison of REJECT with other strategies regarding computational time and solution quality for the small test bed

			REJECT		CONSTRUCT		REPAIR		PEN_REPAIR		PEN_ADAPT	
			CPU	DEV_{opt}	CPU	DEV_{opt}	CPU	DEV_{opt}	CPU	DEV_{opt}	CPU	DEV_{opt}
2000	ini_{feas}	avg	9.18	3.24	0.25	1.18	1.40	0.82	0.38	3.21	0.42	24.13
		std	2.23	1.89	0.01	0.56	0.02	0.46	0.01	1.26	0.01	13.67
	ini_{half}	avg	27.35	4.05	0.25	1.23	1.47	0.79	0.38	3.08	0.42	29.64
		std	5.65	2.31	0.01	0.69	0.02	0.53	0.01	1.15	0.01	15.25
	ini_{rand}	avg	-	-	0.25	1.29	1.53	0.81	0.38	3.03	-	-
		std	-	-	0.01	0.68	0.02	0.56	0.01	1.07	-	-

4.5.2 Results for CONSTRUCT (Strategy-2)

This strategy is experimented in the regular way defined as TSPPD. All of the stopping conditions and all of the initial population types are experimented with the entire problem set realizing 30 replications for each problem. Results of this experiment are provided in Table 4.16.

An important finding is that starting with ini_{rand} did not produce any feasible solution in any replications. Therefore, percent improvement figures for this initial population type can not be computed.

From the ANOVA tables, we could not detect any significant difference regarding solution quality and computation time when we stop early. Although ANOVA reports statistical significance for influence of the initial population on solution quality in Fixedbest stopping condition, due to poor normality and residual plots we cannot be totally sure about the significance. When we include the infeasible solutions in the initial population, the algorithms perform worse. The difference between 5000 and 8000 regarding solution quality and computation time is significant. Looking at overall results, we can say that this strategy is also performs well for TSPB. 2% deviations from optimal in approximately 2 seconds is a promising result regarding the performances of the previous heuristics such as Gendreau et al. (1996).

When we compare the EP values in Table 4.16 with those in Table 4.6, we see that EP values are generally higher for TSPB relative to TSPPD. This case is expected as the nearest neighbor approach is a myopic approach in a sense. There will be many cases, in which the algorithm cannot proceed to an unvisited node using the union graph edges. This increase in the EP value with respect to TSPPD case, can be interpreted as the hardness of this constraint relative to the former constraint as well.

C_{del} decreases as the number of generations increases. A possible reason could be similar to the explanation in TSPPD case.

Table 4.16 Results for CONSTRUCT for TSPB

			GEN	FB	FA	IMP_b	IMP_{avg}	CPU	CPU_{ini}	EP	C_{del}	DEV_{opt}
2000	ini_{feas}	avg	2000.00	3274.12	3564.40	57.95	61.77	0.54	0.01	8.93	1.49	2.72
		std	0.00	25.37	76.21	1.54	0.70	0.01	0.01	0.27	0.03	0.74
	ini_{half}	avg	2000.00	3271.27	3509.43	58.87	62.42	0.54	0.01	8.95	1.49	2.77
		std	0.00	28.71	61.99	1.53	0.71	0.01	0.01	0.28	0.03	0.82
	ini_{rand}	avg	2000.00	3278.24	3404.37	-	-	0.54	0.01	9.03	1.49	2.81
		std	0.00	27.48	28.47	-	-	0.01	0.01	0.26	0.03	0.83
5000	ini_{feas}	avg	5000.00	3257.19	3394.46	58.10	63.03	1.34	0.01	8.72	1.60	2.21
		std	0.00	29.12	50.83	1.51	0.54	0.01	0.01	0.29	0.03	0.61
	ini_{half}	avg	5000.00	3256.45	3390.60	59.02	63.17	1.34	0.01	8.71	1.60	2.28
		std	0.00	27.90	60.26	1.52	0.53	0.01	0.01	0.30	0.04	0.69
	ini_{rand}	avg	5000.00	3261.46	3364.67	-	-	1.34	0.01	8.74	1.60	2.35
		std	0.00	31.07	28.76	-	-	0.01	0.01	0.30	0.04	0.73
8000	ini_{feas}	avg	8000.00	3254.82	3373.31	58.14	63.29	2.13	0.01	8.66	1.63	2.10
		std	0.00	29.40	40.44	1.50	0.43	0.01	0.01	0.29	0.04	0.56
	ini_{half}	avg	8000.00	3249.07	3365.81	59.06	63.32	2.13	0.01	8.64	1.63	2.15
		std	0.00	29.22	45.96	1.51	0.47	0.01	0.01	0.31	0.04	0.69
	ini_{rand}	avg	8000.00	3258.44	3360.15	-	-	2.13	0.01	8.66	1.63	2.24
		std	0.00	30.68	35.51	-	-	0.01	0.01	0.30	0.04	0.66
Fixedbest	ini_{feas}	avg	20878.61	3240.95	3347.84	58.18	63.39	6.45	0.01	8.53	1.66	1.93
		std	4893.86	28.25	29.39	1.53	0.38	1.84	0.01	0.29	0.04	0.63
	ini_{half}	avg	20424.91	3248.69	3352.96	58.97	63.38	6.28	0.01	8.55	1.66	2.01
		std	4579.40	32.85	33.53	1.55	0.42	1.73	0.01	0.31	0.04	0.68
	ini_{rand}	avg	20420.64	3257.93	3355.69	-	-	6.23	0.01	8.57	1.66	2.01
		std	4592.08	28.25	30.39	-	-	1.66	0.01	0.28	0.04	0.64

4.5.3 Results for REPAIR (Strategy-3)

Our standard experiment setting is realized for REPAIR. Like CONSTRUCT, no feasible solution exists in the initial population when it is produced by ini_{rand} method. For either 2000 or Fixedbest stopping conditions, the solution quality does not differ among initial population types. However, the computation time reveals significant difference in both stopping conditions. Therefore, it is advisable to use all feasible initial population. Despite the poor ANOVA plots, the stopping condition 8000 turned out to yield significantly better solutions in significantly more time with respect to stopping condition 2000. When we look at the average figures, for this approximately 0.05% improvement in solution, we should wait for 4 more seconds on the average. The overall results for REPAIR given in Table 4.17, reveals a performance close to CONSTRUCT. The solution quality is approximately 0.2% better but time requirement is considerably more, almost by 7.5 seconds.

The most surprising outcome of this experiment was the small number of total repaired children relative to the case in TSPPD. However, we could not supply a convincing reasoning for this outcome. Another finding is that this quantity depends on the initial population type. This finding is logical when we accept that the probability of generating a feasible solution from feasible parents is high. In fact, as the number of generations increases, the feasible solutions are expected to dominate the population and hence, these values become almost equal.

C_{del} is relatively smaller than that of TSPPD and decreases further as the number of generations increases, which implies that this strategy has a lower probability of producing worse children than the other strategies. As expected, the percent of edges added due to repair operation is more than that of TSPPD. On the average, 10 % of the edges in an infeasible solution is replaced by the edges coming from the repair operation.

Table 4.17 Results for REPAIR for TSPB

			GEN	FB	FA	IMP$_b$	IMP$_{avg}$	CPU	CPU$_{ini}$	EP	EDGE$_{rep}$	TOTREP	C$_{del}$	DEV$_{opt}$
2000	ini$_{feas}$	avg	2000.00	3284.44	3551.76	58.02	62.74	2.36	0.01	11.82	11.25	2850.64	0.91	2.22
		std	0.00	36.83	80.47	1.63	0.77	0.06	0.01	1.54	0.60	215.59	0.18	0.75
	ini$_{half}$	avg	2000.00	3282.51	3481.33	59.02	63.46	2.52	0.01	12.30	11.81	2866.02	0.91	2.16
		std	0.00	42.36	71.27	1.58	0.72	0.05	0.01	1.45	0.54	206.21	0.18	0.80
	ini$_{rand}$	avg	2000.00	3280.70	3397.12	-	-	2.69	0.01	12.86	12.35	2889.51	0.92	2.22
		std	0.00	40.48	39.94	-	-	0.06	0.01	1.58	0.53	217.31	0.17	0.80
5000	ini$_{feas}$	avg	5000.00	3273.47	3377.86	58.11	64.30	5.91	0.01	8.95	10.56	6184.17	0.74	1.87
		std	0.00	37.12	51.26	1.62	0.55	0.13	0.01	1.55	8.58	599.98	0.64	0.71
	ini$_{half}$	avg	4997.34	3267.59	3364.85	59.12	64.45	6.30	0.01	9.20	10.41	6202.72	0.69	1.83
		std	14.57	41.58	46.47	1.56	0.53	0.14	0.01	1.47	8.53	581.74	0.47	0.70
	ini$_{rand}$	avg	5000.00	3272.02	3344.19	-	-	6.73	0.01	9.47	10.82	6244.33	0.73	1.92
		std	0.00	41.56	38.67	-	-	0.14	0.01	1.68	9.21	638.57	0.67	0.75
8000	ini$_{feas}$	avg	7999.15	3271.71	3345.97	58.12	64.63	9.47	0.01	8.02	9.74	9360.65	0.56	1.84
		std	4.66	37.38	36.33	1.61	0.44	0.21	0.01	1.51	0.74	947.07	0.23	0.70
	ini$_{half}$	avg	7986.41	3266.44	3335.12	59.12	64.67	10.09	0.01	8.22	9.97	9376.98	0.56	1.81
		std	56.95	41.34	33.87	1.56	0.45	0.23	0.01	1.47	0.72	963.48	0.24	0.70
	ini$_{rand}$	avg	7986.72	3270.52	3331.58	-	-	10.77	0.01	8.39	10.14	9416.32	0.57	1.87
		std	34.01	42.52	40.38	-	-	0.24	0.01	1.66	0.74	1032.18	0.25	0.75
Fixedbest	ini$_{feas}$	avg	17133.26	3263.42	3316.96	58.31	64.84	17.30	0.01	7.02	9.26	17015.64	0.41	1.79
		std	2405.36	36.68	36.23	1.48	0.35	3.22	0.01	1.37	0.73	3032.73	0.22	0.71
	ini$_{half}$	avg	17377.55	3249.18	3302.57	59.07	64.80	20.85	0.01	7.28	9.38	17609.84	0.42	1.79
		std	2719.01	35.03	31.91	1.53	0.45	3.86	0.01	1.47	0.73	3528.53	0.23	0.71
	ini$_{rand}$	avg	17167.60	3259.75	3313.59	-	-	23.99	0.01	7.46	9.63	17472.86	0.45	1.82
		std	2470.89	35.37	32.26	-	-	4.03	0.01	1.58	0.87	3535.26	0.26	0.74

4.5.4 Results for PEN_REPAIR (Strategy-4)

Our standard experiment setting is tried for this strategy also. In stopping condition 2000, the statistical analyses yield significant differences in computation time regarding the initial population type used. SS value for initial population type is fairly small, however, as this value is smaller for the error coefficient, the hypothesis of equivalence of means is rejected. However, no considerable difference is observed when we look at the average values with two significant digits. Therefore, this statistical difference is discarded. There is no significance of initial population for neither solution quality nor CPU for other stopping condition. When we stop at 8000, we have significantly better solutions than stopping at 5000, however, the time requirement is more for 8000. The solutions of this strategy are deviating 4.3% from the optimal solutions on the average, which is a worse result. However, regarding both solution quality and computation time, this strategy is still a reasonable one to use, and therefore requires additional experimentation.

Table 4.18 summarizes the results for this strategy. One of the important findings is the low value of average number of infeasible solutions. In fact, the value of F_{avg} drops to one digit numbers for TSPB, from two digit numbers for TSPPD. Similarly, the relative performance regarding solution quality gets worse, which may be interpreted that the benefit of working with infeasible solutions is less and the value of feasible for finding good solutions is more for TSPB.

4.5.5 Results for PEN_ADAPT (Strategy-5)

For this strategy, the standard experiment is realized. However, the solution quality that can be achieved were surprisingly poor. The deviation of the resulting solution from the optimal solution is around 100% in general, which is totally unacceptable. The other drawback of this strategy is its imprudence in finding the feasible solutions when started with a random initial population. No feasible solution is obtained for ini_{rand} for all of the replications.

Table 4.18 Results for PEN_REPAIR for TSPB

			GEN	FB	FA	FFEAS	IMP_b	IMP_{avg}	IMP_{feas}	CPU	CPU_{ini}	EP	C_{del}	F_{avg}	DEV_{opt}
2000	ini_{feas}	avg	2000	2411.99	2733	3346.33	67.19	69.83	56.88	0.82	0.01	2.58	0.98	9.22	5.16
		std	0	40.07	95.15	66.48	1.22	0.85	1.72	0.01	0.01	0.4	0.15	0.97	1.28
	ini_{half}	avg	2000	2412.88	2746.85	3357.24	66.52	69.29	56.54	0.82	0.01	2.53	0.98	6.63	5.19
		std	0	35.37	108.39	66.94	1.26	0.85	1.71	0.01	0.01	0.37	0.13	1.12	1.23
	ini_{rand}	avg	2000	2404.56	2741.61	3366.64	66.24	68.76	56.22	0.82	0.02	2.5	0.97	3.58	5.15
		std	0	35.18	95.32	70.44	1.14	0.87	1.66	0.01	0.01	0.39	0.14	0.90	1.21
5000	ini_{feas}	avg	5000	2409.1	2580.81	3330.73	67.24	71.16	57.04	2.01	0.01	1.51	0.74	4.28	4.6
		std	0	40.51	78.79	68.85	1.22	0.7	1.7	0.01	0.01	0.41	0.21	0.63	1.22
	ini_{half}	avg	5000	2409.93	2591.99	3332.79	66.57	70.59	56.73	2.01	0.01	1.51	0.74	3.06	4.57
		std	0	36.32	83.3	58.85	1.25	0.64	1.68	0.01	0.01	0.37	0.19	0.67	1.11
	ini_{rand}	avg	5000	2399.59	2598.31	3343.96	66.28	69.99	56.4	2.01	0.02	1.48	0.74	1.58	4.58
		std	0	34.73	79.56	67.52	1.13	0.68	1.64	0.01	0.01	0.38	0.20	0.50	1.1
8000	ini_{feas}	avg	7999.95	2408.5	2552.81	3327.85	67.25	71.46	57.08	3.19	0.01	1.13	0.60	2.78	4.49
		std	0.26	40.95	81.69	67.84	1.21	0.61	1.7	0.01	0.01	0.4	0.23	0.49	1.21
	ini_{half}	avg	8000	2409.74	2552.41	3329.9	66.58	70.89	56.76	3.19	0.01	1.13	0.60	1.99	4.47
		std	0	36.22	67.46	60.51	1.24	0.55	1.68	0.01	0.01	0.36	0.21	0.48	1.11
	ini_{rand}	avg	7997.48	2398.99	2563.91	3339.87	66.29	70.27	56.43	3.19	0.02	1.11	0.59	1.02	4.49
		std	13.82	35.25	71.5	68.14	1.13	0.58	1.63	0.02	0.01	0.36	0.21	0.36	1.09
Fixedbest	ini_{feas}	avg	18041.1	2402.48	2546.71	3317.94	67.38	71.58	57.26	7.58	0.01	0.67	0.26	0.49	4.41
		std	2999.85	42.6	69.36	50.88	1.08	0.56	1.51	1.52	0	0.33	0.14	0.70	1.18
	ini_{half}	avg	18234.05	2402.12	2568.45	3316.05	66.55	70.99	56.63	7.65	0.01	0.69	0.26	0.40	4.3
		std	2669.94	31.41	67.05	54.56	1.24	0.5	1.69	1.36	0	0.29	0.14	0.45	1.16
	ini_{rand}	avg	18279.94	2410.09	2547.69	3311.41	66.27	70.44	56.43	7.7	0.02	0.67	0.25	0.26	4.34
		std	3003.82	36.98	57.81	56.04	1.2	0.52	1.68	1.56	0	0.31	0.14	0.23	1.16

Considering the poor results of this strategy, we have designed another penalizing scheme, V_2, in which the total constraint violation is multiplied with the difference between unpenalized value of the infeasible solution and the best feasible solution found so far, which is a milder penalizing scheme. The results were worse than the previous case. Finally in order to comment on closeness of the infeasible solutions in the final population to the feasible search space, we designed a scheme, V_{rep}, where all infeasible solutions in the population are repaired at termination. The results were not very promising either. All deviation results are provided in Table 4.19. The original version is referred as V_1 in the table.

Table 4.19 Deviation of the best solution found from the optimal solution value for different versions of PEN_ADAPT proposed for TSPB

		Stopping Condition 2000			Stopping Condition 5000			Stopping Condition 8000			Stopping Condition Fixedbest		
		V_1	V_2	V_{rep}	V_1	V_2	V_{rep}	V_1	V_2	V_{rep}	V_1	V_2	V_{rep}
ini_{feas}	avg	124.73	144.71	16.20	115.59	139.96	15.91	111.56	138.60	16.01	97.56	132.78	16.05
	std	24.30	27.10	2.07	24.83	30.37	2.10	26.35	32.05	2.21	31.22	33.23	2.19
ini_{half}	avg	136.71	159.41	16.03	125.64	155.79	15.99	120.25	154.11	16.15	110.20	151.25	16.50
	std	21.23	28.08	1.97	23.78	31.21	2.02	25.66	33.25	2.17	29.77	34.45	2.12
ini_{rand}	avg	-	-	17.45	-	-	17.08	-	-	17.05	-	-	17.25
	std	-	-	2.23	-	-	2.32	-	-	2.40	-	-	2.50

In Table 4.20, overall results for as V_1 are provided. Although CPU is relatively smaller than the previous algorithms for the same stopping conditions, the number of generations are not increased further, as the solution quality offered do not seem to increase at the expense of the additional seconds. Due to the poor performance of this strategy, no statistical analysis is realized regarding the influence of the parameters. This strategy is directly excluded from the final comparison.

Table 4.20 Results for PEN_ADAPT for TSPB

			GEN	FB	FA	FFEAS	IMP_b	IMP_{avg}	IMP_{feas}	F_{all}	CPU	CPU_{ini}	EP	C_{del}	F_{avg}	DEV_{opt}
2000	ini_{feas}	avg	2000.00	3618.67	5415.00	6119.14	48.76	38.09	30.69	3149.74	0.76	0.01	7.46	1.79	42.20	124.73
		std	0.00	236.65	503.84	957.08	4.07	3.52	7.12	151.98	0.01	0.01	0.54	0.06	5.22	24.30
	ini_{half}	avg	2000.00	2966.09	4955.67	6388.98	60.32	46.25	29.28	2758.32	0.76	0.01	7.30	1.73	24.10	136.71
		std	0.00	123.33	398.09	923.66	2.19	2.33	7.24	76.90	0.01	0.01	0.40	0.04	3.01	21.23
	ini_{rand}	avg	2000.00	2522.33	2958.57	-	65.69	67.05	-	2458.22	0.71	0.01	4.55	1.51	0.00	-
		std	0.00	54.70	85.17	-	1.37	1.15	-	37.22	0.01	0.01	0.62	0.10	0.00	-
5000	ini_{feas}	avg	5000.00	3417.74	4757.75	5891.94	51.82	43.55	31.63	2986.35	1.87	0.01	6.67	1.77	33.50	115.59
		std	0.00	200.83	332.64	1044.41	3.84	3.06	7.71	137.33	0.02	0.01	0.65	0.08	4.92	24.83
	ini_{half}	avg	5000.00	2828.48	4328.93	6228.74	61.38	50.56	31.58	2648.50	1.87	0.01	6.61	1.77	20.50	125.64
		std	0.00	99.04	229.04	957.37	2.03	2.15	7.30	54.31	0.02	0.01	0.48	0.05	2.98	23.78
	ini_{rand}	avg	5000.00	2525.70	2757.72	-	65.75	68.94	-	2451.03	1.73	0.01	3.15	1.46	0.00	-
		std	0.00	48.94	77.35	-	1.39	1.06	-	36.43	0.03	0.01	0.75	0.17	0.00	-
8000	ini_{feas}	avg	8000.00	3325.59	4567.08	5843.53	53.35	45.62	32.20	2917.82	2.98	0.01	6.30	1.76	30.04	111.56
		std	0.00	210.95	310.96	1043.81	3.75	3.11	7.69	128.12	0.04	0.01	0.71	0.10	4.88	26.35
	ini_{half}	avg	8000.00	2794.50	4143.69	6165.83	61.76	52.37	32.20	2615.17	2.97	0.01	6.30	1.78	18.95	120.25
		std	0.00	83.72	191.82	965.60	1.92	2.06	7.75	51.30	0.03	0.01	0.52	0.06	3.00	25.66
	ini_{rand}	avg	7999.32	2529.69	2715.87	-	65.71	69.34	-	2448.42	2.72	0.01	2.56	1.41	0.00	-
		std	3.72	53.36	68.46	-	1.43	0.98	-	35.88	0.05	0.01	0.76	0.20	0.00	-
Fixedbest	ini_{feas}	avg	19721.94	3112.72	4184.38	5660.87	57.52	51.95	34.07	2790.49	8.37	0.01	5.64	1.73	22.57	97.56
		std	4298.49	211.80	378.71	1044.43	3.47	4.64	8.95	114.76	2.22	0.01	0.84	0.11	5.55	31.22
	ini_{half}	avg	21059.50	2735.42	3901.45	5806.79	62.48	55.65	32.90	2574.09	9.02	0.01	5.79	1.77	16.01	110.20
		std	4869.91	73.46	242.39	902.10	1.82	2.82	9.03	38.90	2.49	0.01	0.64	0.08	4.06	29.77
	ini_{rand}	avg	21292.78	2545.34	2718.28	-	65.76	69.61	-	2442.90	8.03	0.01	1.72	1.25	0.00	-
		std	329.93	65.81	103.67	-	1.43	1.04	-	37.23	0.29	0.01	0.67	0.24	0.00	-

4.5.6 Comparison of Strategies

Strategy-1 and Strategy-5 turned out to be impractical to implement and excluded from the overall comparison consideration. The overall results for the main performance measures for the remaining strategies are provided in Table 4.21. It can be seen that REPAIR produces better solutions in general. However, its time requirement is also more than the others. CONSTRUCT can give solutions that are only 0.2% worse than the solutions of REPAIR in a time, which is less than the one third of the time required for REPAIR.

The statistical analysis that was carried out for TSPPD was repeated for TSPB. In fact, the ordering of the strategies were the same for every stopping condition. The differences among the algorithms were significant. The best algorithm was REPAIR regarding solution quality. It was followed by CONSTRUCT while PEN_REPAIR was the worst one among the three. However, the CPU of REPAIR was worst. For all tests, the overall p value was less than 0.05 although the worst overall p value obtained was 0.03. The results are provided in Table 4.22. The related tables are provided in Appendix I.

4.5.7 Comparison with Nearest Neighbor with Repair Heuristic

Although there are examples of conventional heuristics for TSPB in literature, we could not find any metaheuristic applications to this problem. Therefore, it would be convenient to analyze the relative improvement of this method over the conventional methods due to metaheuristic notion. In Table 4.23, the results of the NN with repair heuristic are provided. As it is realized in TSPPD case, the same repair algorithm of EA is utilized in this heuristic. All of the starting nodes are tried for generating a solution. The results for TSPB were worse than TSPPD. The average deviation is 18% which is far more than 3% deviation that can be attained in the earlier stops in our algorithms. The results for both TSPPD and TSPB do not leave any doubts and questions about the benefit of metaheuristics. For both cases, the solution quality improves drastically when the algorithms run a few seconds more.

Table 4.21 Overall results for TSPB[*]

			CONSTRUCT S-2		REPAIR S-3		PEN_REPAIR S-4	
			CPU	DEV$_b$	CPU	DEV$_b$	CPU	DEV$_b$
2000	ini$_{feas}$	avg	0.54	2.72	2.36	2.22	0.82	5.16
		std	0.01	0.74	0.06	0.75	0.01	1.28
	ini$_{half}$	avg	0.54	2.77	2.52	2.16	0.82	5.19
		std	0.01	0.82	0.05	0.8	0.01	1.23
	ini$_{rand}$	avg	0.54	2.81	2.69	2.22	0.82	5.15
		std	0.01	0.83	0.06	0.8	0.01	1.21
5000	ini$_{feas}$	avg	1.34	2.21	5.91	1.87	2.01	4.6
		std	0.01	0.61	0.13	0.71	0.01	1.22
	ini$_{half}$	avg	1.34	2.28	6.3	1.83	2.01	4.57
		std	0.01	0.69	0.14	0.7	0.01	1.11
	ini$_{rand}$	avg	1.34	2.35	6.73	1.92	2.01	4.58
		std	0.01	0.73	0.14	0.75	0.01	1.1
8000	ini$_{feas}$	avg	2.13	2.1	9.47	1.84	3.19	4.49
		std	0.01	0.56	0.21	0.7	0.01	1.21
	ini$_{half}$	avg	2.13	2.15	10.09	1.81	3.19	4.47
		std	0.01	0.69	0.23	0.7	0.01	1.11
	ini$_{rand}$	avg	2.13	2.24	10.77	1.87	3.19	4.49
		std	0.01	0.66	0.24	0.75	0.02	1.09
Fixedbest	ini$_{feas}$	avg	6.45	1.93	17.3	1.79	7.58	4.41
		std	1.84	0.63	3.22	0.71	1.52	1.18
	ini$_{half}$	avg	6.28	2.01	20.85	1.79	7.65	4.3
		std	1.73	0.68	3.86	0.71	1.36	1.16
	ini$_{rand}$	avg	6.23	2.01	23.99	1.82	7.7	4.34
		std	1.66	0.64	4.03	0.74	1.56	1.16

[*]S-i: strategy i, where i=2,3,4

Table 4.22 Indifference groups constructed by the result of the Tamhane's T2 test

		$ini_{feas} + ini_{half} + ini_{rand}$			ini_{feas}		
		1st group	2nd group	3rd group	1st group	2nd group	3rd group
Stopping Condition 2000	DEV_b	3	2	4	3	2	4
	CPU	2	4	3	2	4	3
Stopping Condition 5000	DEV_b	3	2	4	3	2	4
	CPU	2	4	3	2	4	3
Stopping Condition 8000	DEV_b	3	2	4	3	2	4
	CPU	2	4	3	2	4	3
Stopping Condition Fixedbest	DEV_b	3	2	4	3-2	4	-
	CPU	2	4	3	2	4	3

Table 4.23 Results of the NN with repair heuristic for TSPB

	Best	CPU	DEV_b
p00	467.00	0.00	21.30
p01	644.00	0.02	11.03
p02	787.00	0.00	33.62
p03	924.00	0.02	14.36
p04	1103.00	0.00	22.01
p05	1043.00	0.03	19.89
p06	141.00	0.00	14.63
p07	79.00	0.00	16.18
p08	147.00	0.00	14.84
p09	7012.00	0.00	12.30
p10	732.00	0.00	12.27
p11	585.00	0.00	10.38
p12	1402.00	0.05	31.40
p13	993.00	0.00	27.47
p14	343.00	0.00	6.85
p15	1259.00	0.03	9.38
p16	1186.00	0.02	38.07
p17	57579.00	0.02	26.00
p18	700.00	0.00	6.22
p19	567.00	0.00	18.87
	avg	0.01	18.35
	std	0.01	9.05

4.5.8 Comparison with Solving Corresponding ATSP Instance

As stated before, every TSPB instance can be transformed to a TSP instance by modifying the cost matrix. Considering this fact, an option for solving TSPB problems can be transforming these instances to TSP and solving these instances. We are interested in experimenting this option also, as it will reveal the benefit of dealing with TSPB instances instead of the corresponding TSP instances. Therefore, we experimented the performance of this strategy which solves transformed TSPB instances utilizing TSP solving EA. This strategy is called TRANS. The results of this strategy and our previous strategies are provided in Table 4.24.

Table 4.24 Overall results and results of TRANS for TSPB

		CONSTRUCT		REPAIR		PEN_REPAIR		TRANS	
		CPU	DEV_b	CPU	DEV_b	CPU	DEV_b	CPU	DEV_b
2000	avg	0.54	2.81	2.69	2.22	0.82	5.15	0.34	4.38
	std	0.01	0.83	0.06	0.8	0.01	1.21	0.01	1.39
5000	avg	1.34	2.35	6.73	1.92	2.01	4.58	0.84	3.64
	std	0.01	0.73	0.14	0.75	0.01	1.1	0.01	1.19
8000	avg	2.13	2.24	10.77	1.87	3.19	4.49	1.33	3.44
	std	0.01	0.66	0.24	0.75	0.02	1.09	0.02	1.14
Fixedbest	avg	6.23	2.01	23.99	1.82	7.7	4.34	3.91	3.18
	std	1.66	0.64	4.03	0.74	1.56	1.16	1.23	1.03

TRANS gives worse results than constructing feasible solutions from scratch or repairing infeasible solutions. As we are penalizing the edges going from pickups to deliveries to realize the transformation, this option can be considered as a static penalizing scheme where the penalty costs are reflected to edge costs. Looking through this respect, the results of this experiment coincides with the results for the previous ones. Again, working with only feasible solutions gives better results than a scheme that permits infeasible solutions in the population. However, TRANS finds better solutions than PEN_REPAIR. Actually, this reveals the benefit of reflecting penalty coefficients to edges. As expected, this strategy works faster than other strategies due to absence of additional constraint handling effort.

4.6 Concluding Remarks

The main issue in this study is to answer the question "Should we keep infeasible solutions in the population for constrained routing problems?". Looking at the performance results of the specific algorithms' performances we can discuss the value of the infeasible solution.

Firstly, it should be mentioned that, the answer to the question varies according to the side constraints added. In our case, the two side constraints are deemed different in hardness. Hardness can be defined by the effort required to find a feasible solution with respect to the side constraints when realizing the search in the solution space of the unconstrained problem. From this respect, TSPB looks "harder" in our case. For TSPPD, one of the schemes working with infeasible solutions, namely, PEN_REPAIR, provides fairly good results. For this algorithm, 70-80% of the population is composed of the infeasible solutions on the average. However, as the side constraint gets harder, the value of the infeasible solutions decreases. PEN_REPAIR algorithm, for TSPB, does not produce results as good as REPAIR algorithm. In fact, the structures of algorithms resemble to each other. However, the repaired version of the infeasible solution replaces the infeasible solution in REPAIR, which turns out to give results 3% better than PEN_REPAIR while 90% of the population is infeasible on the average. Therefore, for this problem, we can say that proceeding with infeasible solutions is not advisable. In fact, the fairly good results for the softer constraints may be conceivable. In PEN_REPAIR, all of the infeasible solutions are repaired, and the reason for this algorithm to give good results may be solely this repairing operation. The influence of keeping infeasible solutions on obtaining good solutions may be negligible. In almost all experiments, replacing the infeasible solution with its repaired version instead of keeping this infeasible solution and penalizing it gave better results. The only exception occurred in the test bed of Gendreau et al. (1999), when the problems are generated with $\boldsymbol{\beta} = 0.20$.

Nevertheless, proceeding with infeasible solutions may be considered for softer side constraints as long as penalizing is performed appropriately. The results of PEN_ADAPT were unacceptably bad, whereas PEN_REPAIR provides the best results in some stopping conditions for TSPPD. The difference arises due to the feasibility distance metric used. The constraint violation used in PEN_ADAPT, ceases to be an appropriate measure for defining the distance of the infeasible solution to the feasible solution space, especially, when the crossover operator is based on a heuristic. In fact, the adaptive penalizing scheme proposed by Coit et. al (1996) is experimented with a uniform crossover operator, and to our knowledge, no work studying the effects of adaptive penalizing used with heuristic crossover methods exist in the literature. The results obtained here implies that simply penalizing the solution value does not forces the algorithm to find good feasible solutions, as the crossover operator mainly deals with the edges, instead of the value of the solution. Actually, it may be a good idea to reflect the penalized value to the edges that causes infeasibility for our purposes. However, this may be a more complex and time consuming penalizing scheme.

Another comparison can be realized between the strategies using feasible solutions. REJECT strategy has been found insufficient after experiments due to its enormous computation time requirement. The remaining two strategies, CONSTRUCT and REPAIR, are good strategies. Superiority of one over the other has not been observed for the softer side constraint. For the harder side constraint, REPAIR yielded significantly better results. However, no persuasive reasoning can be stated.

For both TSPPD and TSPB, computation time required for generation of initial population did not differ with respect to population types. However utilizing a random population is easier to implement relative to other types. Another general outcome is the improvement capabilities of the heuristics proposed. Improvement figures do not provide a basis for comparing the strategies, however it surely gives an opinion about the value added by the heuristics. For good strategies, the average improvement of the best solution relative to the best solution in the initial population is around 70% for TSPPD and 60% for TSPB. These figures show only slight differences among the initial population types.

CHAPTER 5

CONCLUSION

In this research, we propose evolutionary algorithms for the traveling salesman problem with side constraints. Specifically, we try to adapt an EA that is proved to work well for TSP, to TSPPD and TSPB. The algorithm was not a traditional Genetic Algorithm but a more sophisticated on utilizing the conventional, well known TSP heuristic, called Nearest Neighbor Heuristic as the crossover operator. The main difficulty in this adaptation is to ensure the feasibility of the solution with respect to the side constraints. The literature proposes several constraint handling techniques for EAs. The individual results for these techniques are available, however, to our knowledge, a comparison for these techniques has not been realized before. Impressed by the versatility of TSP, we intended to make a comparison of constraint handling techniques for EAs in the domain of TSP.

From the reviewed constraint handling techniques, the most basic ones are selected to be compared. The first one is simply rejecting the infeasible solutions. Prior to our experimentation, this strategy is already known to be inappropriate for the cases in which the solution spaces of the constrained and the unconstrained problems rarely coincide. However, in order to quantify the "rareness" for the problems of question and to provide a thorough comparison, this strategy is included in the study.

The second and third ones are modifying the crossover operator to ensure feasibility and repairing the infeasible solutions. In the literature, there are successful examples for both strategies in dealing with side constraints. However, their relative performance is not measured before.

The last strategies are penalizing strategies in which the infeasible solutions are permitted in the population but the chance for them to pass their genetic material to the following generations is reduced by penalizing the fitness value of these solutions. There are various penalizing schemes that can be utilized. We, again, selected the most basic ones. The first of them is taking the repaired value of infeasible solutions as the penalized value, whereas the second one, which is an adaptive penalizing strategy, utilizes a distance metric based on the constraint violation.

After determining a framework for our comparison, the algorithms for constrained single vehicle routing problems are designed. These algorithms are then implemented for TSPPD and TSPB, and analyzed by computational experiments on a test bed taken from the literature.

Firstly, the convergence plots of the algorithms are drawn for both TSPPD and TSPB. However, no single bound on the stopping condition can be determined. Bounds on stopping conditions are considered as experimental factors. The specific values of these bounds are determined after analyzing the convergence plots. In addition to stopping conditions, three initial population types differing in number of feasible solutions in population are experimented in the study.

The influence of the factors is analyzed by ANOVA. In our experimentation, the algorithms are found to be statistically insensitive to the initial population type for most of the cases. However, in some cases, the random population gives slightly significant, or close to significant results for TSPPD regarding the solution quality. In general, initial population types do not influence solution quality for TSPB. However, due to significant differences regarding the computation times, all feasible initial population is reported to be better than the others.

The results for stopping conditions are analyzed for every strategy individually. In almost all cases, the significance regarding the solution quality and computation time is obtained between 8000th and 5000th generations. Speaking with the average figures, the least difference occurred between these stopping bounds. Therefore, the differences between other pairs are assumed to be significant.

The results of the algorithms are compared by Tamhane's T2 test. Repairing, constructing from scratch, and penalizing by repairing turned out to give better results relatively to rejecting and penalizing adaptively strategies for TSPPD. High computation time requirement of rejecting infeasible solutions for TSPPD gets higher for TSPB, which makes this strategy almost infeasible for TSPB. The adaptive penalizing strategy provides inferior solutions regarding the solution quality for TSPPD. For TSPB, the differentiation is more clear. The relative order of five strategies from best to worst regarding the solution quality is repairing, constructing from scratch and penalizing by repair. However, repairing consumes much more time than it consumes for TSPPD case. The solutions produced by penalizing adaptively were worse than two times of the optimal solutions, which is totally unacceptable. At the end utilizing this scheme with a heuristic crossover is found inappropriate.

In general, keeping infeasible solutions in the population may not be a good alternative as the side constraints restrict the solution space more. However, by using a penalizing scheme incorporating the penalty values to force production of feasible children, better results can be found. Penalizing the fitness values of solutions alone may not be a good idea to use with a NN crossover operator.

In the overall, strategy-2, 3 and 4 give solutions deviating from the optimal TSP values between 4.2 - 4.5% in about 2 seconds for TSPPD. The solutions for TSPB deviate around 2% from the optimal values for strategy-2 and 3. The result of strategy-4 is worse than the optimal values by 4.5% on the average. The time required for strategy-2 and 4 are 2 - 3 seconds, while strategy-3 requires a time around 10 seconds. At the end, we can say that using a modified crossover operator that produces feasible solutions and repairing infeasible solutions are better alternatives for constrained routing problems. If the side constraint is a milder one, allowing the infeasible solutions and penalizing them by using the value of their repaired versions is a viable option also. Actually these conclusions are valid for the EA structure selected. In order to come up with more general conclusions, other EAs working well for TSP should be used in the experimentations.

The three constraint handling strategies were better than the other two for TSPPD. We compared them with the heuristics of Gendreau et al. (1999). These heuristics are one conventional heuristic and two tabu search procedures. Gendreau et al. can find better solutions for TSP, but our algorithms provide better results as the hardness of the constraints increases. Penalizing by repair gives an average deviation 2.8%, and the deviation of the other two are around 3-4%, whereas the best heuristic of Gendreau et al., deviates by 6.3%. Although our algorithms may require more time than those of Gendreau et al., the improvement in solution quality makes our algorithms a better choice.

Designing a mutation operator performing local search is the first item on the list of our future research. From Sönmez (2003), we know that the deviation figures can be improved by 2% after NN crossover is used with a random initial population for TSP. This figure motivates us to analyze the effect of such a mutation operator for the constrained cases. An alternative local search may incorporate feasible two exchange moves.

For future research, another immediate study can be adaptation of these algorithms to the problems with optional pickups. In this problem, the necessity of visiting every pickup customer is removed, and visiting a pickup customer returns a profit. The objective of the problem is to minimize the net cost of the tour visiting all delivery customers and selected pickup customers. This adaptation can be realized very easily by updating the crossover operator utilized in this study. If the costs of edges going to pickup customers are decreased by considering the revenue to be collected from that customer, and if the tour is constructed by an optional NN heuristic with the updated costs, then the optional problems can be solved by the algorithm.

Another research topic in future is to adapt these algorithms for the multi-vehicle cases. Actually, by utilizing a cluster first route second approach, the algorithms proposed here can be directly applicable.

In future research, it might be interesting to design a penalty scheme that can work well with the NN crossover operator. Here, the penalty figures can be reflected to the cost of edges that cause infeasibility. The children are produced using the penalized cost matrix.

As our algorithm gives fairly good results for TSP, transforming TSPB to STSP and solving this instance with our standard algorithm may give good results. The experimentation of this method of solving TSPB's will be considered in further research.

A last research topic that can be worked in future may be proposing an algorithm that utilizes the better constraint handling techniques in a combined fashion. Here, the infeasible solution is treated by one of these techniques according to a probability.

REFERENCES

Anily, S. and G. Mosheiov (1994), The Traveling Salesman Problem with Delivery and Backhauls, Operations Research Letters 16, 11-18.

Anily, S. and J. Bramel (1999), Approximation Algorithms for the Capacitated Traveling Salesman Problem with Pickups and Delivers, Naval Research Logistics 46, 654-670.

Applegate, D., R. Bixby, V. Chavatal, and W. Cook (1998), On the Solution of Traveling Salesman Problems, Documenta Mathematica, Journal der Deutschen Mathematiker-Vereinigung, 645-656.

Baldacci R., E. Hadjiconstantinou and A Mingozzi (2003), An Exact Algorithm for the Traveling Salesman Problem with Deliveries and Collections, Networks 42 (1), 26-41.

Beasley, D., D. R. Bull and R. R. Martin (1993), An Overview of Genetic Algorithms: Part 1, Fundamentals, University of Computing 15 (2), 58-69.

Beasley, J. E., and P. C. Chu (1996), A Genetic Algorithm for the Set Covering Problem, European Journal of Operational Research 94, 392-404.

Bodin, L. and B. Golden (1981), Classification in Vehicle Routing and Scheduling, Networks 11, 97-108.

Bodin L., B. Golden, A. Assad and M. Ball (1983), Routing and Scheduling of Vehicles and Crews – State of the Art, Computers & Operations Research 10 (2), 63-211.

Chen, S., S. Smith and C. Guerra-Salcedo (1999), The Genie is Out! (Who Needs Fitness to Evolve?) in Proceedings of the Congress on Evolutionary Computation 3, 2102-2108.

Coit D. W., A. E. Smith and D. M. Tate (1996), Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems, Journal of Computing 8 (2), 173-182.

Daganzo, C. F. and R. W. Hall (1993), A Routing Model for Pickups and Delivers: No Capacity Restrictions on the Secondary Items, Transportation Science 27 (4), 315-329.

Davis, L. (1991), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York.

Demirel, Ö. (2001), Heuristic Algorithms for the Routing Problems with Backhauls, MSc Thesis, Ankara, METU.

Dethloff, J. (2002), Relation between Vehicle Routing Problems: An Insertion Heuristic for the Vehicle Routing Problem with Simultaneous Delivery and Pick-Up Applied to the Vehicle Routing Problem with Backhauls, Journal of the Operational Research Society 53, 13-118.

Erdem, E., N. E. Özdemirel (2003), An Evolutionary Approach for the Target Allocation Problem, Journal of the Operational Research Society 54, 958-969.

Fisher, M. (1994), Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees, Operations Research 42, 626-642.

Gendreau, M., A. Hertz and G. Laporte (1996), The Traveling Salesman Problem with Backhauls, Computers & Operations Research 23 (5), 501-508.

Gendreau, M., G. Laporte and A. Hertz (1997), An Approximation Algorithm for the Traveling Salesman Problem with Backhauls, Operations Research 45 (4), 639-641.

Gendreau, M., G. Laporte and D. Vigo (1999), Heuristics for the Traveling Salesman Problem with Pickup and Delivery, Computers & Operations Research 26, 699-714.

Ghaziri, H. and I. H. Osman (2003), A Neural Network Algorithms for the Traveling Salesman Problem with Backhauls, Computers & Industrial Engineering 44, 267-281.

Goetschalckx, M. and C. Jacobs-Blecha (1989), The Vehicle Routing Problem with Backhauls, European Journal of Operational Research 42, 39-51.

Golden, B. L., E. A. Wasil, J. P. Kelly and I Chao (1998), The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results in Crainic T. and G. Laporte (eds.) Fleet Management and Logistics, Kluwer Academic Publishers, Boston MA, 33-56.

Holland, J. H. (1975), Adaptation in Natural and Artificial Systems, Ann Arbor, MI, University of Michigan Press.

Jog, P., J. Y. Suh and D. V. Gucht (1989), The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem in Schaffer, D. J. (ed.), Proceedings 36 of the Third International Conference of Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA, 110-115.

Johnson, D. S. and L. A. McGeoch (1997), The Traveling Salesman: A Case Study in Aarts, E. and J. K. Lenstra (eds.), Local Search in Combinatorial Optimization, John Wiley & Sons, New York, 215-310.

Jongens, K. and T. Volgenant (1985), The Symmetric Clustered Traveling Salesman Problem, European Journal of Operational Research 19, 68-75.

Jürgen, M., G. Reinelt and G. Rinaldi (1995), The Traveling Salesman Problem, in Ball M. O. et al. (eds.) Handbooks in OR & MS 7, Elsevier Science, 235-329.

Kilby P., P. Prosser and P. Shaw (2000), A Comparison of Traditional and Constraint-based Heuristic Methods on Vehicle Routing Problems with Side Constraints, Constraints 5, 389-414.

Michalewicz Z. and D. B. Fogel (2000), How to Solve It: Modern Heuristics, Springer, Berlin.

Mingozzi, A., S. Giorgi and R. Baldacci (1999), An Exact Method for the Vehicle Routing Problem with Backhauls, Transportation Science 33 (3), 315 329.

Mosheiov, G. (1994), The Traveling Salesman Problem with Pick-up and Delivery, European Journal of Operational Research 79, 299-310.

Nagata, Y. and S. Kobayashi (1997), Edge Assembly Crossover: A High-Power Genetic Algorithm for the Traveling Salesman Problem, in Proceedings of the 7th International Conference on Genetic Algorithms, 450-457.

Nagy, G. and S. Salhi (2004), Heuristic Algorithms for Single and Multiple Depot Vehicle Routing Problems with Pickups and Deliveries, European Journal of Operational Research, in press.

Orvosh, D. and L. Davis (1993), Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints, in Forrest, S. (ed.) Proceedings of 5th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, p. 650.

Osman, I. H. and N. A. Wassan (2002), A Reactive Tabu Search Meta-heuristic for the Vehicle Routing Problem with Backhauls, Journal of Scheduling 5, 263-285.

Reeves, R. C. (1995), A Genetic Algorithm for Flowshop Sequencing, Computers & Operations Research 22, 5-13.

Reeves, R. C. (1997), Genetic Algorithms for the Operations Researchers, Journal of Computing, 9 (3), 231-250.

Rego, C. and F. Glover (2002), Local Search and Metaheuristics in Gutin, G. and A. P. Punnen (eds.), The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Dordrecht, Netherlands, 309-368.

Savelsbergh M. W. P. and M. Sol (1995), The General Pickup and Delivery Problem, Transportation Science 29 (1), 17-29.

Sönmez, M. (2003), An Evolutionary Approach to TSP: Crossover with Conventional Heuristics, MSc Thesis, Ankara, METU.

Süral, H. and J. H. Bookbinder (2003), The Single-Vehicle Routing Problem with Unrestricted Backhauls, Networks 41 (13), 127-136.

Toothacker, L. E. (1993), Multiple comparisons procedures, CA: Sage Publications, Inc, Newbury Park.

Toth P. and D. Vigo (1997), An Exact Algorithm for the Vehicle Routing Problem with Backhauls, Transportation Science 31 (4), 372-385.

Toth, P. and D. Vigo (1999), A Heuristic Algorithm for the Symmetric and Asymmetric Vehicle Routing Problems with Backhauls, European Journal of Operational Research 113, 528-543.

Tsiligirides, T. (1984) Heuristic Methods Applied to Orienteering, Journal of the Operational Research Society 35, 797-809.

Tzoreff, T. E., D. Granot, F. Granot and G. Sosic (2002), The Vehicle Routing Problem with Pickups and Deliveries on Some Special Graphs, Discrete Applied Mathematics 116, 193-229.

Ulusoy, G., F. Sivrikaya-Şerifoğlu, Ü. Bilge (1997), A Genetic Algorithm Approach to Simultaneous Scheduling of the Machines and Automated Guided Vehicles, Computers & Operations Research 24 (4), 335-351.

Van Breedam, A. (2001), Comparing Descent Heuristics and Metaheuristics for the Vehicle Routing Problem, Computers & Operations Research 28, 289-315.

Wade, A. C. and S. Salhi (2002), An Investigation into a New Class of Vehicle Routing Problem with Backhauls, Omega 30, 479-487.

Whitley, D. (1989), The Genitor Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best, in Proceedings of Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 116-121.

Yano, C. A., T. J. Chan, L. K. Richter, T. Cutler, K. G. Mutry and D. Mcgettigan (1987), Vehicle Routing at Quality Stores, Interfaces 17, 52-63.

APPENDIX A

TRANSFORMATION OF TSPB TO STSP

In this appendix, an example for illustrating the transformation of TSPB instances to STSP instances is provided. First TSPB instances are transformed to asymmetric TSP instances as discussed in Chapter 2. In Figure A.1, a trivial instance of TSPB is provided. The square represents the depot (node c), the white node (a) is the only delivery customer and the black node (b) is the only pickup customer. This instance is transformed into the ATSP instance given in Figure A.2.

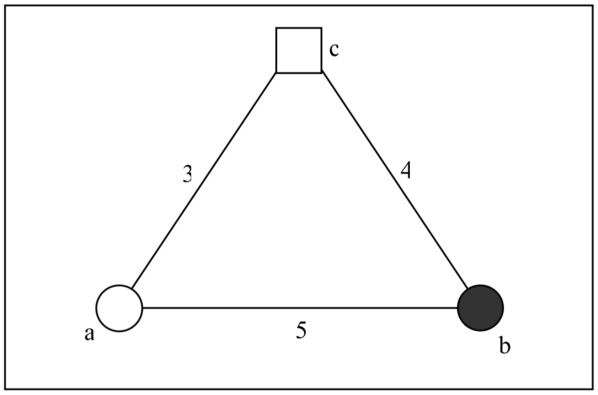


Figure A.1 An instance of TSPB

After obtaining ATSP instances, they are converted to symmetric instances by doubling all nodes (Jürgen et al. 1995). A single node **i** is represented by two nodes in this new version, namely, $\mathbf{i}_{\text{arrival}}$ and $\mathbf{i}_{\text{departure}}$. The arc costs of the updated instance should be designed so that the optimal tour in this version gives the optimal tour for the asymmetric case. The cost of an edge directed from node **i** to node **j** at the original instance equals to the cost of the edge between $\mathbf{i}_{\text{arrival}}$ and $\mathbf{i}_{\text{departure}}$ in the updated instance.

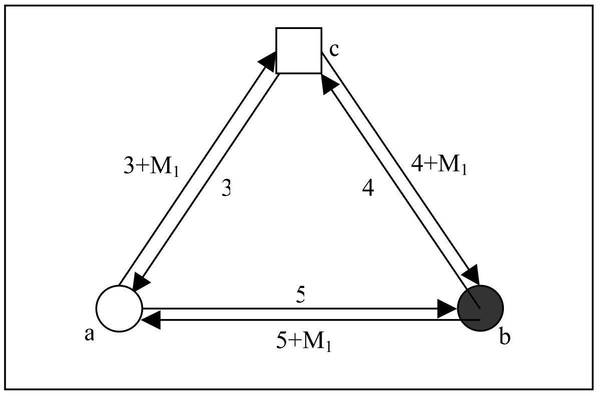


Figure A.2 ATSP instance corresponding to the TSPB instance given

The solution for the updated version should not include any edge between the nodes that have the same subscript, i.e., $\mathbf{i}_{\text{arrival}}$ and $\mathbf{j}_{\text{arrival}}$, or $\mathbf{i}_{\text{departure}}$ and $\mathbf{j}_{\text{departure}}$. Therefore, their cost**s** are set to sufficiently large values of M. Once the tour reaches to node $\mathbf{i}_{\text{coming}}$, the tour should continue with visiting $\mathbf{i}_{\text{going}}$. Note that, the cost of the edges between the duplicates of nodes of the original problem are irresistibly small, i.e., 0. The STSP instance corresponding to the initial example is provided in Figure A.3. Note that, M_1, M_2, and M_3 should be set far larger than the cost of the original edges. For this case, an M value of 100 will be appropriate for obtaining a valid TSPB tour from this instance.

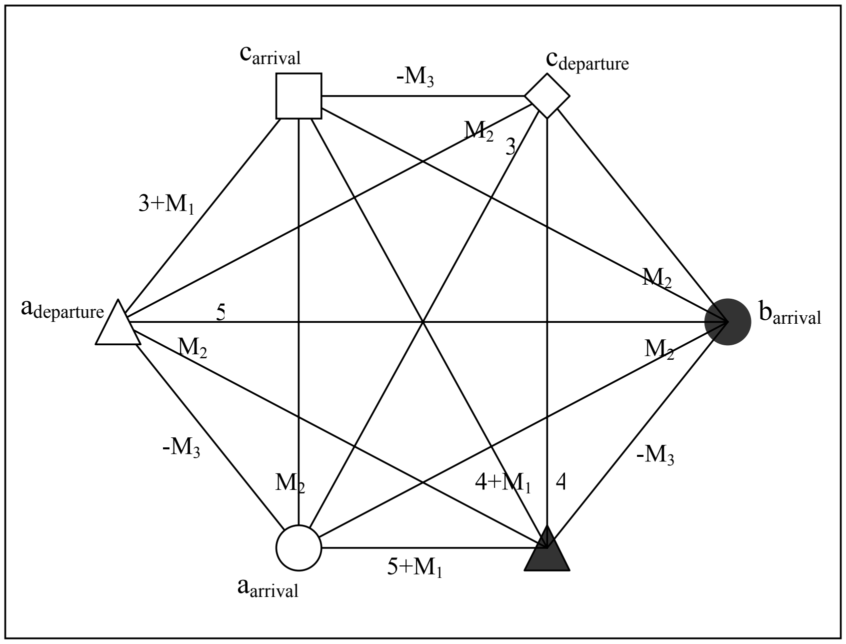


Figure A.3 STSP instance corresponding to the TSPB instance given

APPENDIX B

OPTIMAL TOUR VALUES

In Table B.1, the optimal tour values are provided for the test bed. TSP_{opt}, $TSPPD_{opt}$, and $TSPB_{opt}$ represent the optimal solution values for TSP, TSPPD and TSPB, respectively.

Table B.1 Optimal values for different problem types

Name	TSP_{opt}	$TSPPD_{opt}$	$TSPB_{opt}$
P00	294	310	385
P01	462	479	580
P02	463	-	589
P03	585	-	808
P04	651	-	904
P05	631	-	870
P06	97	100	123
P07	55	55	68
P08	106	106	128
P09	4431	4488	6244
P10	484	502	652
P11	402	408	530
P12	810	-	1067
P13	602	-	779
P14	252	-	321
P15	796	-	1151
P16	709	-	859
P17	33551	-	45697
P18	511	529	659
P19	377	381	477

APPENDIX C

CONVERGENCE PLOTS

In this appendix, the convergence plots of five strategies for problems p00 and p15 are provided. Firstly, the plots for TSPPD algorithms and then the plots for TSPB algorithms are provided. For TSPPD plots, the initial population type utilized is ini_{rand} whereas ini_{feas} is utilized for TSPB. In the figures, "bestog" stands for the best solution averages for 30 replications. Similarly, "avgog" keeps the average of population averages and "cpu" keeps the average computation time. For penalizing strategies, "bestfeas" keeps the average of best feasible solution.

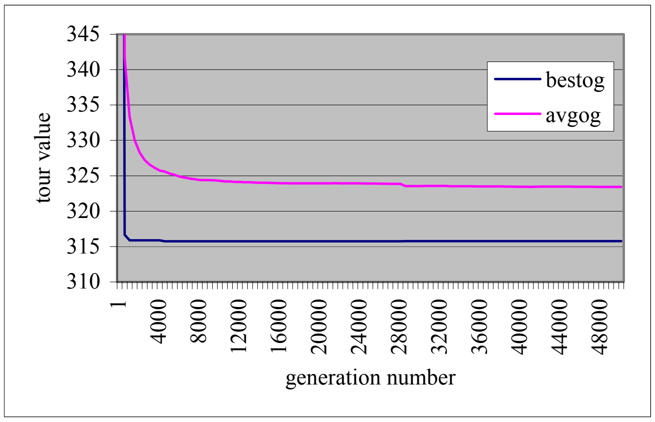


Figure C.1 Bestog and Avgog vs. number of generations for p00 with strategy-1 for TSPPD

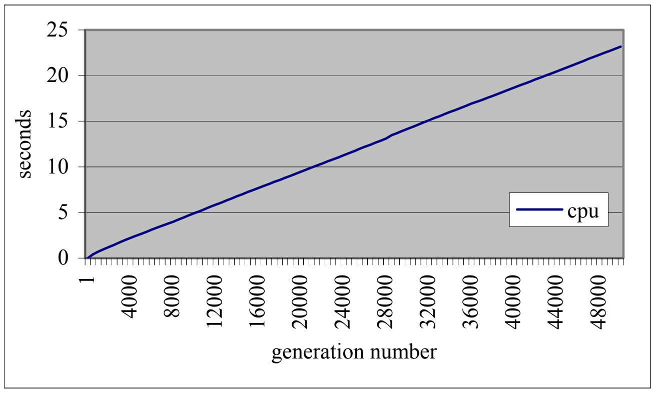


Figure C.2 CPU vs. number of generations for p00 with strategy-1 for TSPPD

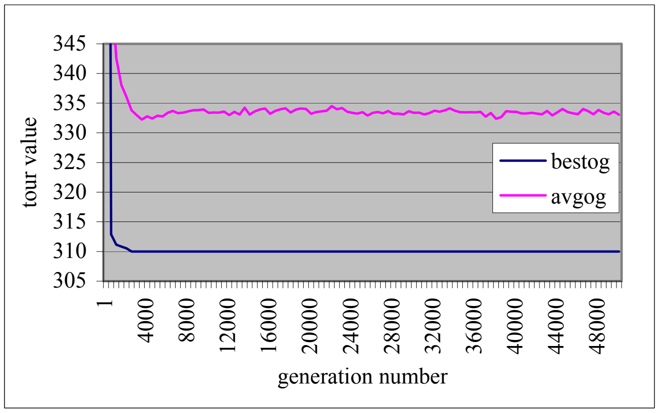


Figure C.3 Bestog and Avgog vs. number of generations for p00 with strategy-2 for TSPPD

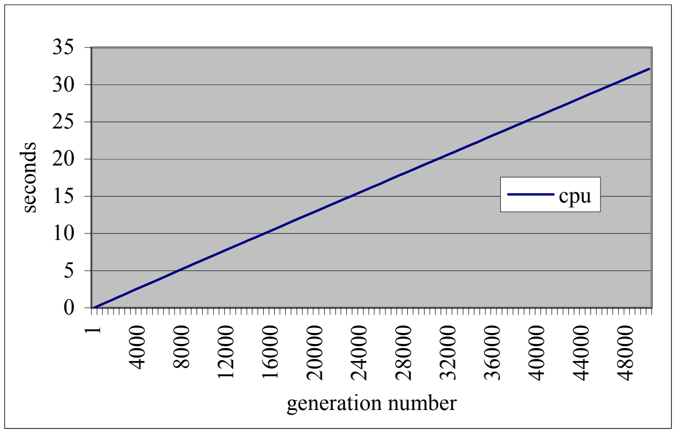


Figure C.4 CPU vs. number of generations for p00 with strategy-2 for TSPPD

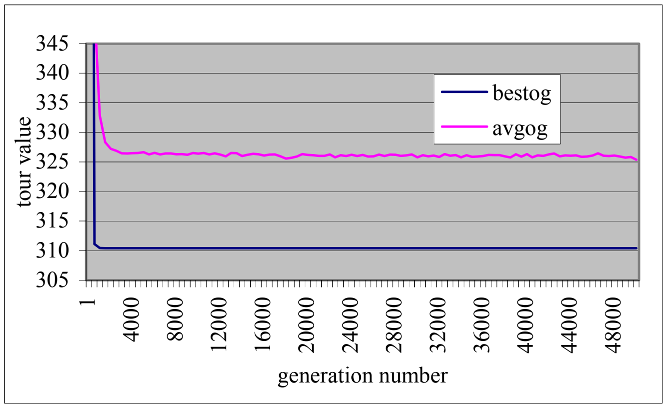


Figure C.5 Bestog and Avgog vs. number of generations for p00 with strategy-3 for TSPPD

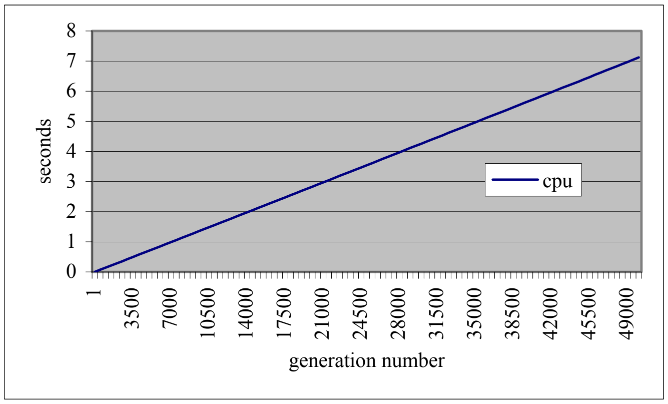


Figure C.6 CPU vs. number of generations for p00 with strategy-3 for TSPPD

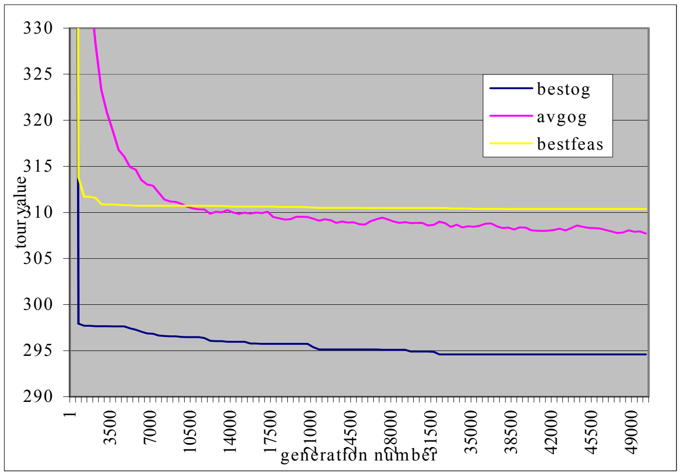


Figure C.7 Bestog, Avgog, and Bestfeas vs. number of generations for p00 with strategy-4 for TSPPD

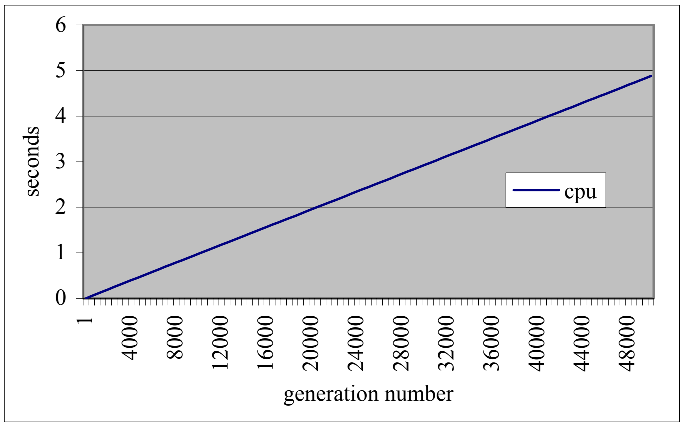


Figure C.8 CPU vs. number of generations for p00 with strategy-4 for TSPPD

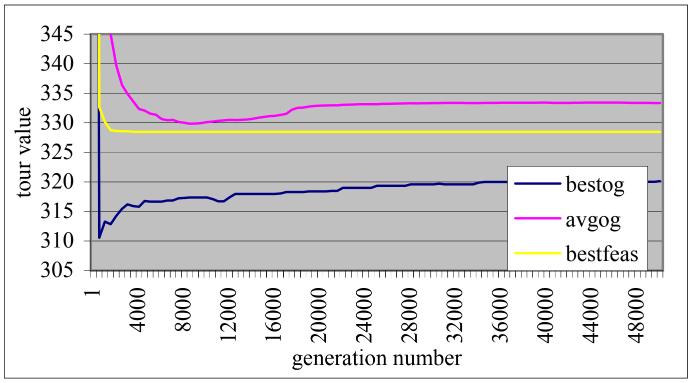


Figure C.9 Bestog, Avgog, and Bestfeas vs. number of generations for p00 with strategy-5 for TSPPD

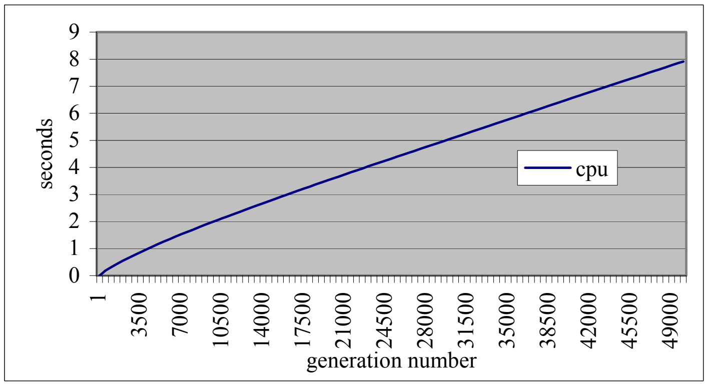


Figure C.10 CPU vs. number of generations for p00 with strategy-5 for TSPPD

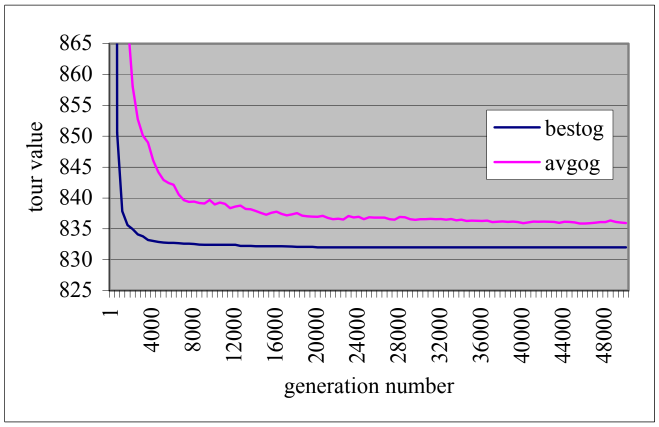


Figure C.11 Bestog and Avgog vs. number of generations for p15 with strategy-1 for TSPPD

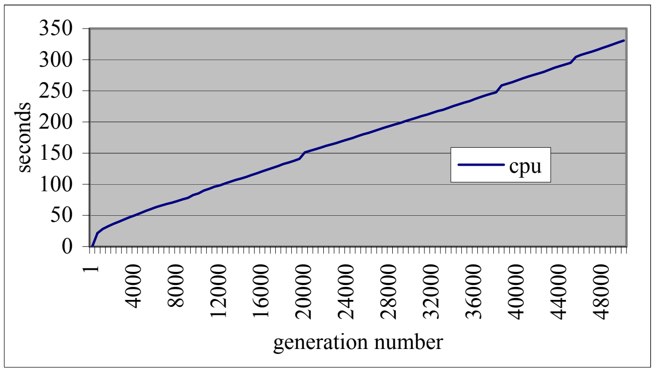


Figure C.12 CPU vs. number of generations for p15 with strategy-1 for TSPPD

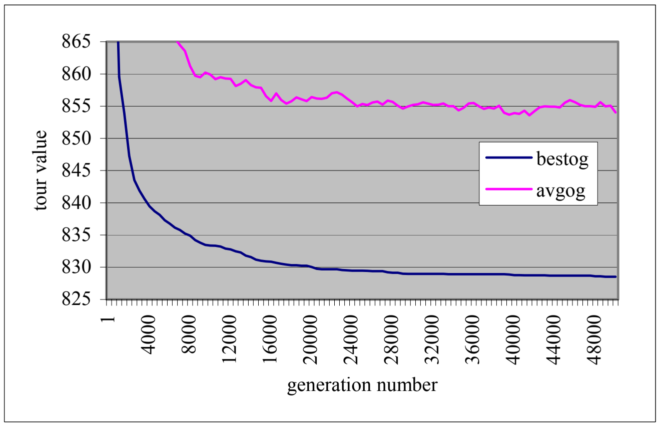


Figure C.13 Bestog and Avgog vs. number of generations for p15 with strategy-2 for TSPPD

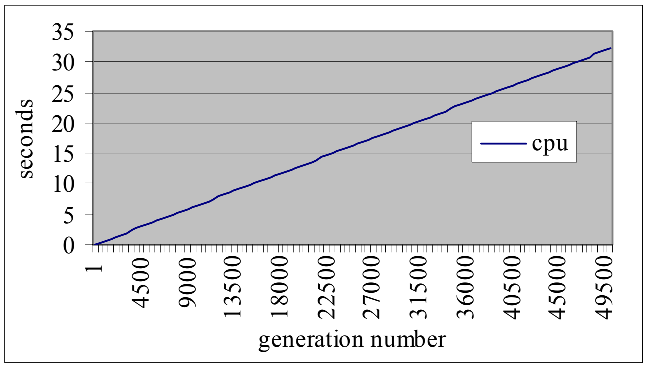


Figure C.14 CPU vs. number of generations for p15 with strategy-2 for TSPPD

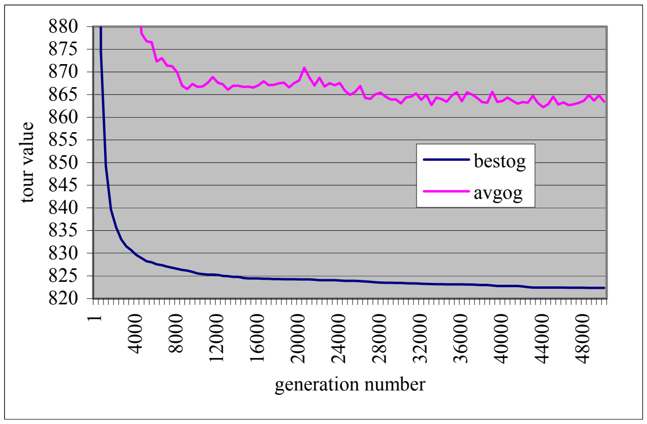


Figure C.15 Bestog and Avgog vs. number of generations for p15 with strategy-3 for TSPPD

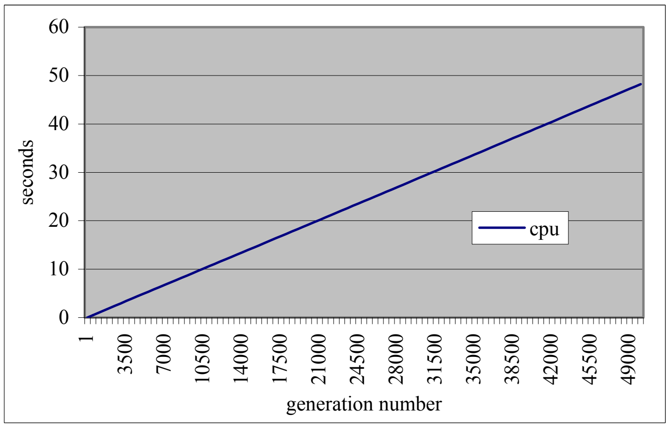


Figure C.16 CPU vs. number of generations for p15 with strategy-3 for TSPPD

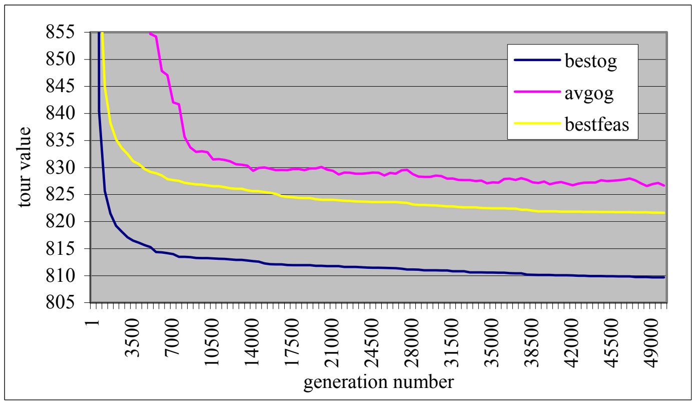


Figure C.17 Bestog, Avgog, and Bestfeas vs. number of generations for p15 with strategy-4 for TSPPD

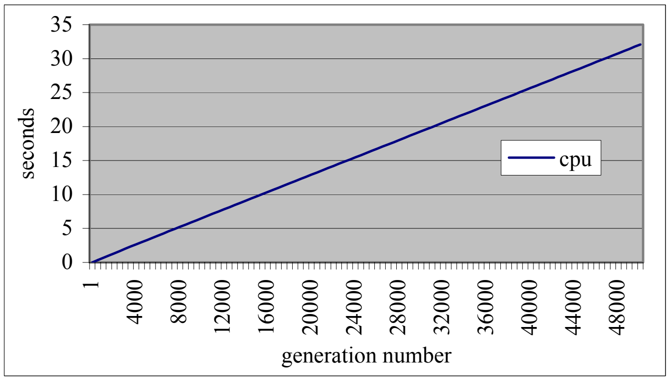


Figure C.18 CPU vs. number of generations for p15 with strategy-4 for TSPPD

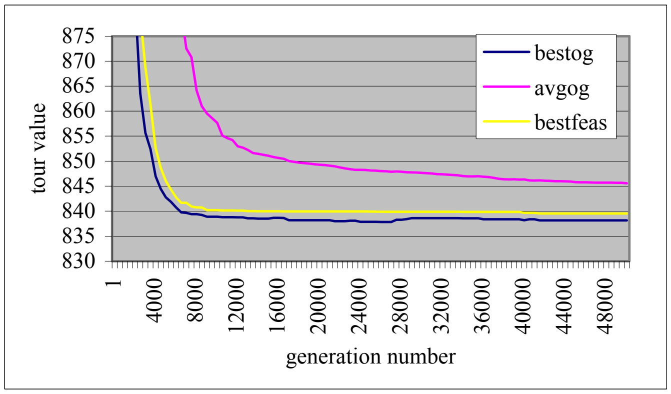


Figure C.19 Bestog, Avgog, and Bestfeas vs. number of generations for p15 with strategy-5 for TSPPD

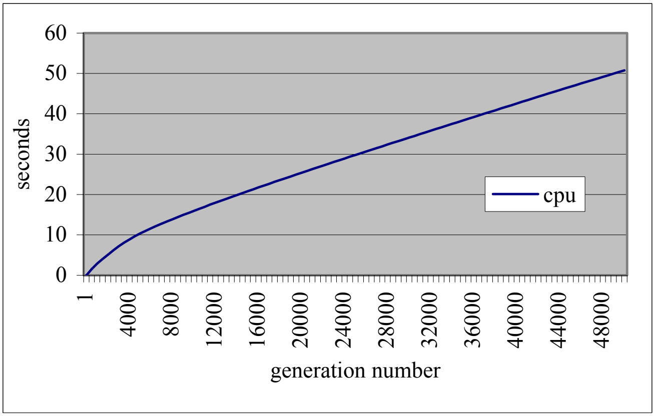


Figure C.20 CPU vs. number of generations for p15 with strategy-5 for TSPPD

Due to large computational requirements of REJECT for TSPB, the convergence experiment could not be realized for problem p15. For the smaller problem p00, we stop earlier, at 10000th generation, because of the same reason.

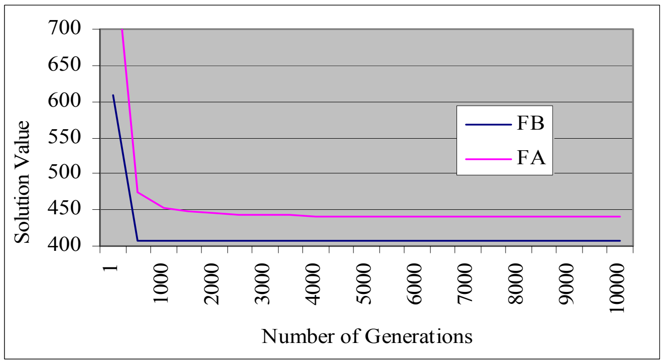


Figure C.21 Bestog and Avgog vs. number of generations for p00 with strategy-1 for TSPB

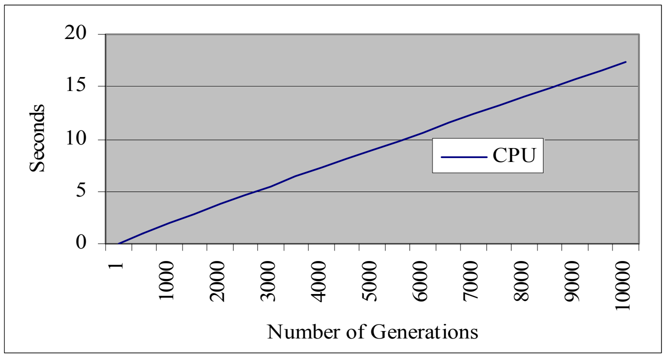


Figure C.22 CPU vs. number of generations for p00 with strategy-1 for TSPB

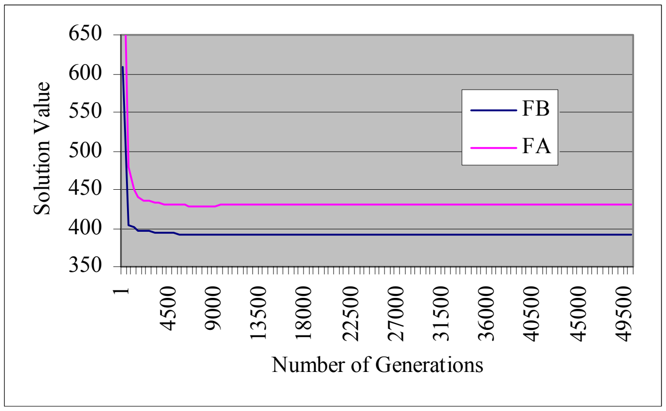


Figure C.23 Bestog and Avgog vs. number of generations for p00 with strategy-2 for TSPB

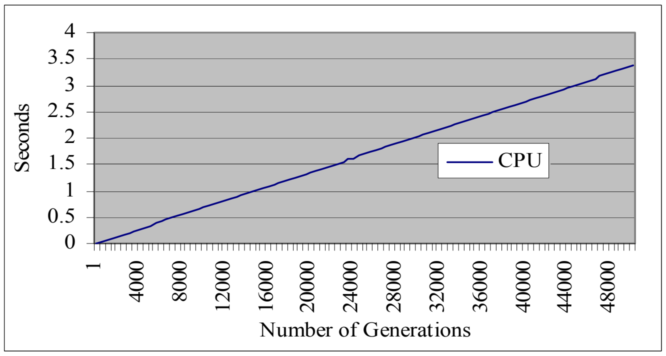


Figure C.24 CPU vs. number of generations for p00 with strategy-2 for TSPB

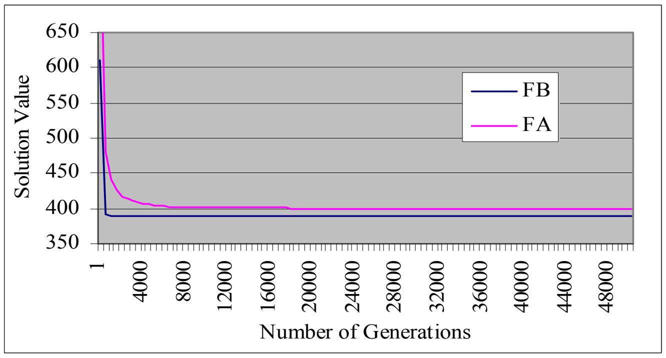


Figure C.25 Bestog and Avgog vs. number of generations for p00 with strategy-3 for TSPB

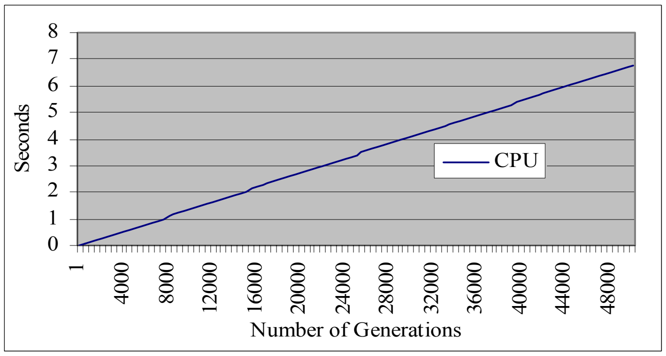


Figure C.26 CPU vs. number of generations for p00 with strategy-3 for TSPB

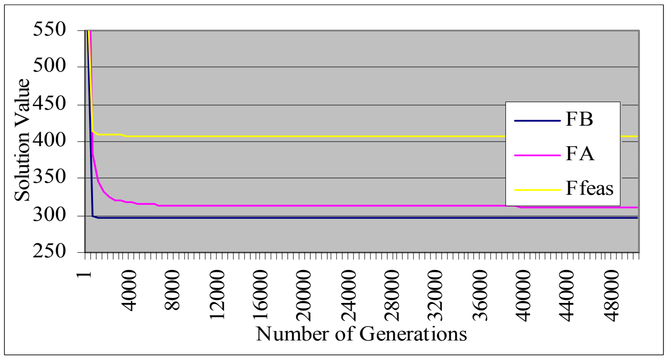


Figure C.27 Bestog, Avgog, and Bestfeas vs. number of generations for p00 with strategy-4 for TSPB

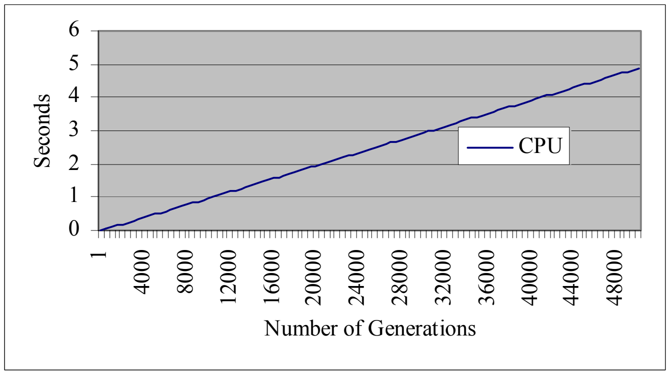


Figure C.28 CPU vs. number of generations for p00 with strategy-4 for TSPB

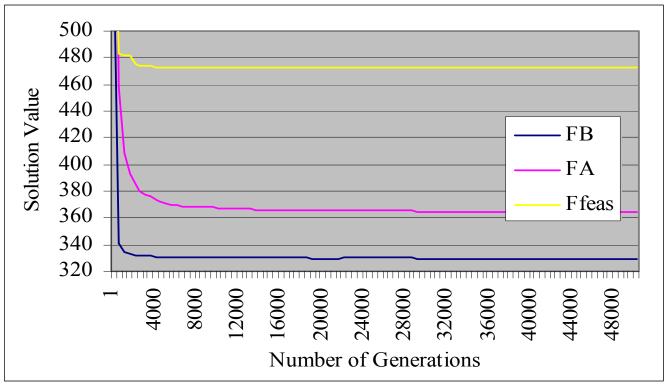


Figure C.29 Bestog, Avgog, and Bestfeas vs. number of generations for p00 with strategy-5 for TSPB

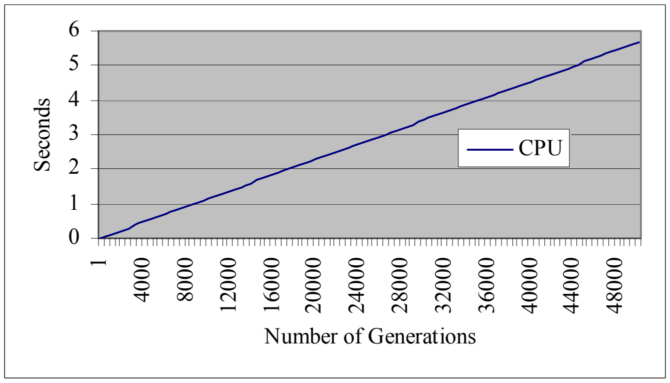


Figure C.30 CPU vs. number of generations for p00 with strategy-5 for TSPB

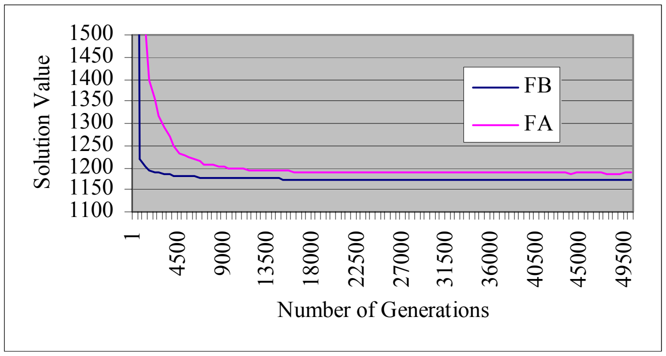


Figure C.31 Bestog and Avgog vs. number of generations for p15 with strategy-2 for TSPB

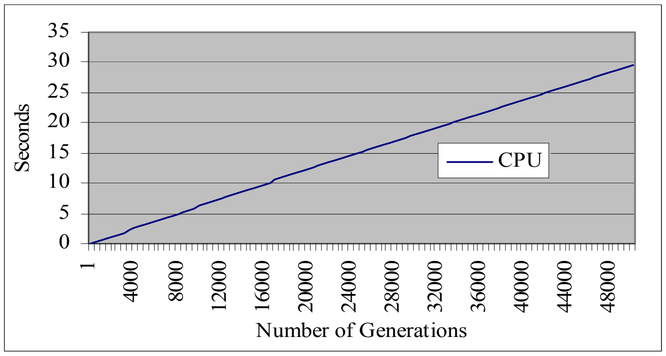


Figure C.32 CPU vs. number of generations for p15 with strategy-2 for TSPB

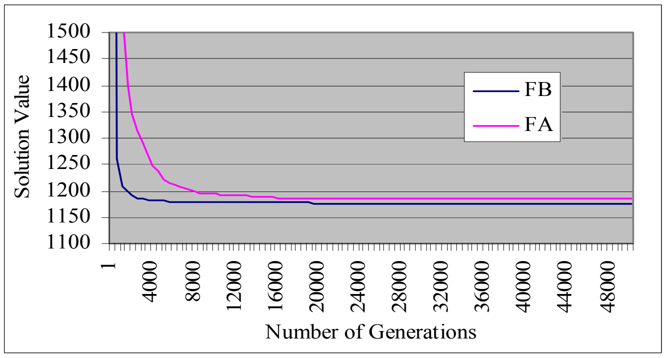


Figure C.33 Bestog and Avgog vs. number of generations for p15 with strategy-3 for TSPB

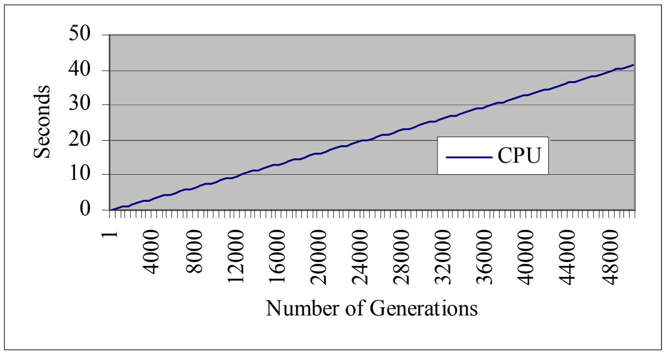


Figure C.34 CPU vs. number of generations for p15 with strategy-3 for TSPB

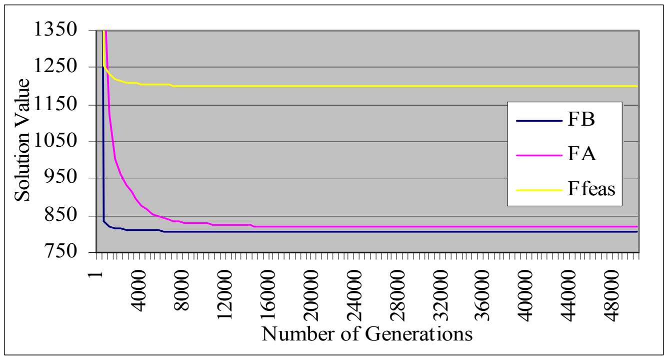


Figure C.35 Bestog, Avgog, and Bestfeas vs. number of generations for p15 with strategy-4 for TSPB

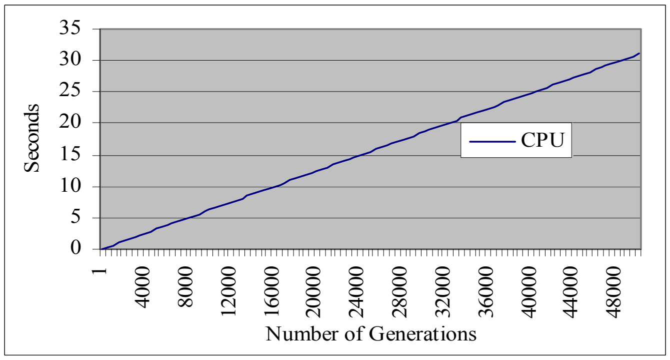


Figure C.36 CPU vs. number of generations for p15 with strategy-4 for TSPB

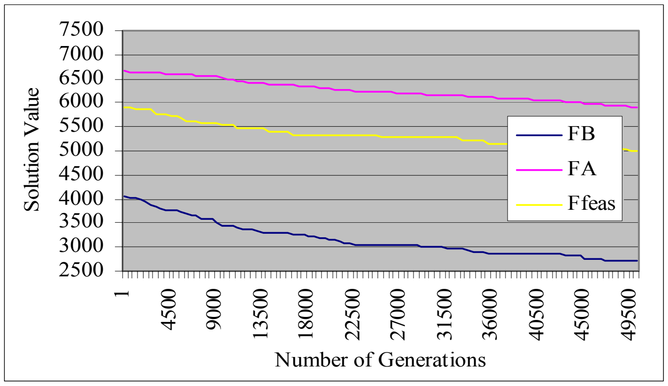


Figure C.37 Bestog, Avgog, and Bestfeas vs. number of generations for p15 with strategy-5 for TSPB

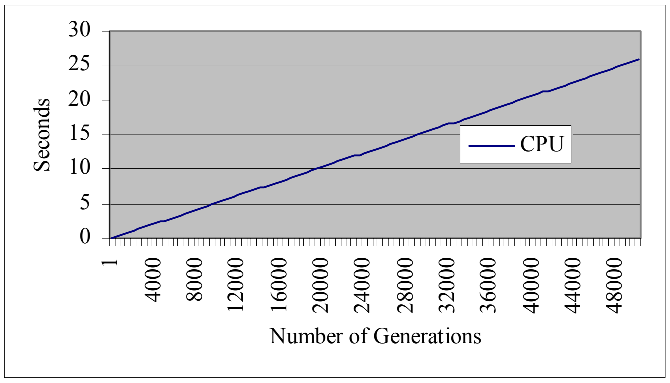


Figure C.38 CPU vs. number of generations for p15 with strategy-5 for TSPB

APPENDIX D

STATISTICAL ANALYSES REGARDING THE EFFECT OF INITIAL POPULATION FOR TSPPD

In this appendix, ANOVA tables and related plots for determining the effect of initial population type on solution quality and computation time are provided for each strategy when the stopping conditions are 2000 and Fixedbest.

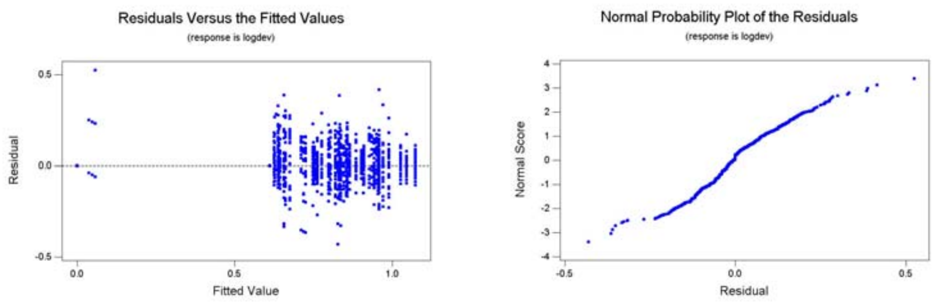


Figure D.1 Normality and residual plots for strategy-1, when stopping condition is 2000 with response log(DEV_b)

Table D.1 ANOVA table for strategy-1, when stopping condition 2000 with response log(DEV_b)

```
Analysis of Variance for logdev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0105     0.0105     0.0053    0.95  0.395
problem           19   124.9644   124.9644     6.5771 1189.96  0.000
initype*problem   38     0.2100     0.2100     0.0055    0.64  0.959
Error           1740    15.1215    15.1215     0.0087
Total           1799   140.3065
```

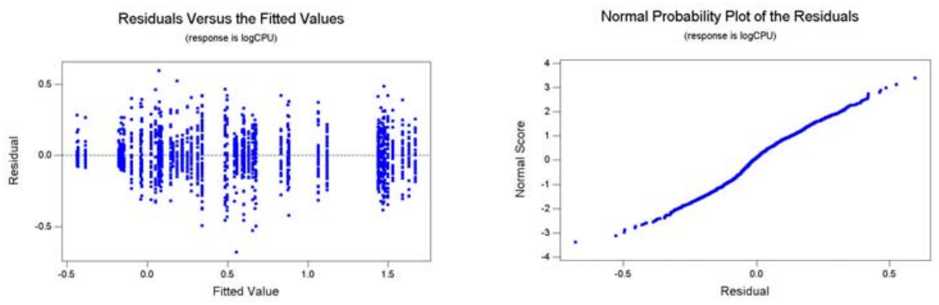


Figure D.2 Normality and residual plots for strategy-1, when stopping condition is 2000 with response log(CPU)

Table D.2 ANOVA table for strategy-1, when stopping condition 2000 with response log(CPU)

```
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS        F       P
initype            2      0.139      0.139      0.069     2.72   0.079
problem           19    663.173    663.173     34.904  1369.80   0.000
initype*problem   38      0.968      0.968      0.025     1.26   0.132
Error           1740     35.119     35.119      0.020
Total           1799    699.399
```

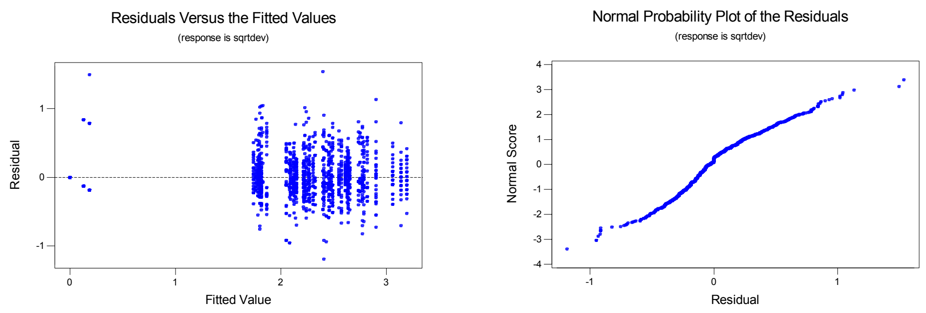


Figure D.3 Normality and residual plots for strategy-1, when stopping condition is 8000 with response sqrt(DEV_b)

Table D.3 ANOVA table for strategy-1, when stopping condition 8000 with response sqrt(DEV_b)

```
Analysis of Variance for sqrtdev, using Adjusted SS for Tests

Source              DF     Seq SS     Adj SS     Adj MS       F      P
initype              2      0.069      0.069      0.035    0.49  0.619
problem             19   1052.986   1052.986     55.420  777.03  0.000
initype*problem     38      2.710      2.710      0.071    0.86  0.719
Error             1740    145.061    145.061      0.083
Total             1799   1200.826
```

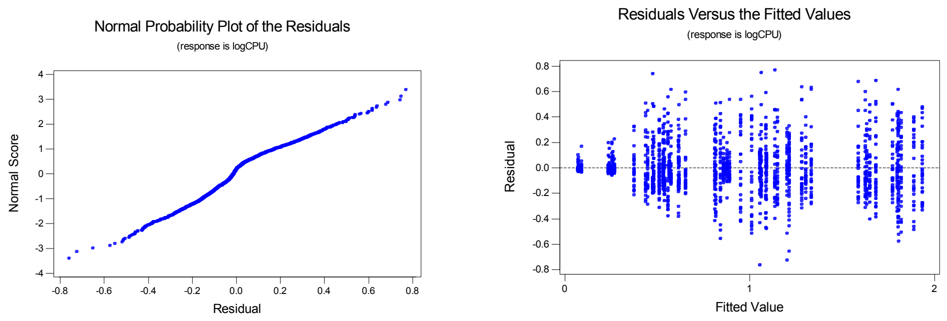


Figure D.4 Normality and residual plots for strategy-1, when stopping condition is 8000 with response log(CPU)

Table D.4 ANOVA table for strategy-1, when stopping condition 8000 with response is log(CPU)

```
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source              DF     Seq SS     Adj SS     Adj MS       F      P
initype              2     0.0548     0.0548     0.0274    0.66  0.523
problem             19   508.0761   508.0761    26.7408  644.12  0.000
initype*problem     38     1.5776     1.5776     0.0415    1.10  0.317
Error             1740    65.8683    65.8683     0.0379
Total             1799   575.5768
```

Table D.5 ANOVA tables for strategy-2

```
Stopping condition 2000
Analysis of Variance for dev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2       3.97       3.97       1.99    1.06  0.356
problem           19   18218.63   18218.63     958.88  512.04  0.000
initype*problem   38      71.16      71.16       1.87    1.32  0.090
Error           1740    2459.64    2459.64       1.41
Total           1799   20753.40
```

```
Stopping condition 2000
Analysis of Variance for CPU, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0001     0.0001     0.0001    2.45  0.100
problem           19   311.6065   311.6065    16.4003 7.0E+05  0.000
initype*problem   38     0.0009     0.0009     0.0000    0.52  0.993
Error           1740     0.0781     0.0781     0.0000
Total           1799   311.6856
```

```
Stopping Condition Fixedbest
Analysis of Variance for dev, using Adjusted SS for Tests


Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2       2.57       2.57       1.29    3.03  0.060
problem           19    6671.18    6671.18     351.11  827.05  0.000
initype*problem   38      16.13      16.13       0.42    0.70  0.918
Error           1740    1059.70    1059.70       0.61
Total           1799    7749.59
```

```
Stopping Condition Fixedbest
Analysis of Variance for logCPU, using Adjusted SS for Tests



Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0121     0.0121     0.0060    0.37  0.696
problem           19   302.7415   302.7415    15.9338  965.42  0.000
initype*problem   38     0.6272     0.6272     0.0165    1.41  0.049
Error           1740    20.3176    20.3176     0.0117
Total           1799   323.6984
```

Table D.6 ANOVA tables for strategy-3

```
Stopping Condition 2000
Analysis of Variance for logdev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0043     0.0043     0.0021    0.38  0.683
problem           19   142.6518   142.6518     7.5080 1352.88  0.000
initype*problem   38     0.2109     0.2109     0.0055    1.21  0.182
Error           1740     8.0012     8.0012     0.0046
Total           1799   150.8681
```

```
Stopping Condition 2000
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0006     0.0006     0.0003    1.74  0.189
problem           19   150.7840   150.7840     7.9360 4.6E+04  0.000
initype*problem   38     0.0066     0.0066     0.0002    1.64  0.008
Error           1740     0.1845     0.1845     0.0001
Total           1799   150.9756
```

```
Stopping Condition Fixedbest
Analysis of Variance for logdev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0019     0.0019     0.0009    0.19  0.826
problem           19   113.9628   113.9628     5.9980 1236.19  0.000
initype*problem   38     0.1844     0.1844     0.0049    1.05  0.394
Error           1740     8.0715     8.0715     0.0046
Total           1799   122.2206
```

```
Stopping Condition Fixedbest
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0006     0.0006     0.0003    1.74  0.189
problem           19   150.7840   150.7840     7.9360 4.6E+04  0.000
initype*problem   38     0.0066     0.0066     0.0002    1.64  0.008
Error           1740     0.1845     0.1845     0.0001
Total           1799   150.9756
```

Table D.7 ANOVA tables for strategy-4

```
Stopping Condition 2000
Analysis of Variance for logdev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0080     0.0080     0.0040    0.45  0.639
problem           19   143.8101   143.8101     7.5690  859.66  0.000
initype*problem   38     0.3346     0.3346     0.0088    1.28  0.120
Error           1740    11.9813    11.9813     0.0069
Total           1799   156.1340
```

```
Stopping Condition 2000
Analysis of Variance for sqrtcpu, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0001     0.0001     0.0001    2.22  0.122
problem           19   115.4462   115.4462     6.0761 2.2E+05  0.000
initype*problem   38     0.0010     0.0010     0.0000    0.84  0.751
Error           1740     0.0569     0.0569     0.0000
Total           1799   115.5043
```

```
Stopping Condition Fixedbest
Analysis of Variance for sqrtdev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2      0.281      0.281      0.141    2.42  0.103
problem           19   1038.689   1038.689     54.668  940.46  0.000
initype*problem   38      2.209      2.209      0.058    0.87  0.695
Error           1740    116.156    116.156      0.067
Total           1799   1157.335
```

```
Stopping Condition Fixedbest
Analysis of Variance for logcpu, using Adjusted SS for Tests


Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0289     0.0289     0.0144    1.39  0.262
problem           19   258.1284   258.1284    13.5857 1307.49  0.000
initype*problem   38     0.3948     0.3948     0.0104    0.93  0.587
Error           1740    19.3732    19.3732     0.0111
Total           1799   277.9252
```

Table D.8 ANOVA tables for strategy-5

```
Stopping Condition 2000
Analysis of Variance for logdev, using Adjusted SS for Tests

Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2     1.6686     1.6686     0.8343    4.36  0.020
problem            19   308.7941   308.7941    16.2523   84.99  0.000
initype*problem    38     7.2663     7.2663     0.1912   10.43  0.000
Error            1740    31.9035    31.9035     0.0183
Total            1799   349.6325
```

```
Stopping Condition 2000
Analysis of Variance for cpu, using Adjusted SS for Tests

Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2    205.487    205.487    102.744   16.13  0.000
problem            19   2391.308   2391.308    125.858   19.76  0.000
initype*problem    38    242.059    242.059      6.370  278.13  0.000
Error            1740     39.852     39.852      0.023
Total            1799   2878.705
```

```
Stopping Condition Fixedbest
Analysis of Variance for logdev, using Adjusted SS for Tests

Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2     0.0148     0.0148     0.0074    0.49  0.616
problem            19   142.2806   142.2806     7.4885  495.90  0.000
initype*problem    38     0.5738     0.5738     0.0151    1.07  0.351
Error            1740    24.4778    24.4778     0.0141
Total            1799   167.3471
```

```
Stopping Condition Fixedbest
Analysis of Variance for logcpu, using Adjusted SS for Tests


Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2     0.1341     0.1341     0.0670    4.56  0.017
problem            19   269.1023   269.1023    14.1633  962.54  0.000
initype*problem    38     0.5591     0.5591     0.0147    1.06  0.379
Error            1740    24.2570    24.2570     0.0139
Total            1799   294.0525
```

APPENDIX E

STATISTICAL ANALYSES REGARDING THE EFFECT OF STOPPING CONDITION FOR TSPPD

In this appendix, ANOVA tables and related plots for determining the effect of stopping condition on solution quality and computation time are provided for each strategy when data from the stated initial population types is considered.

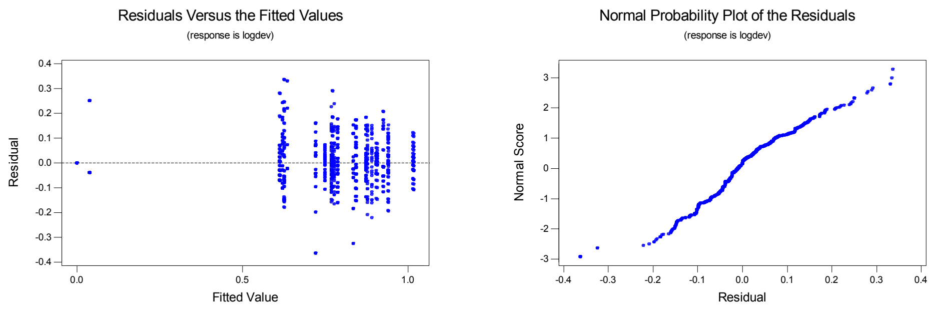


Figure E.1 Normality and residual plots for strategy-1, with response log(DEV$_b$)

Table E.1 ANOVA table for strategy-1 (5000-8000), when initial population type used is ini$_{feas}$, with response log(DEV$_b$)

```
Analysis of Variance for logdev, using Adjusted SS for Tests

Source         DF      Seq SS     Adj SS     Adj MS       F       P
gen             1      0.0013     0.0013     0.0013    8.03   0.011
problem        19     79.5134    79.5134     4.1849 2.7E+04   0.000
gen*problem    19      0.0030     0.0030     0.0002    0.02   1.000
Error        1160      9.1058     9.1058     0.0078
Total        1199     88.6234
```

Table E.2 ANOVA table for strategy-2 (5000-8000), when initial population type used is ini_{rand}, with response $\log(DEV_b)$

```
Analysis of Variance for logdev, using Adjusted SS for Tests


Source         DF     Seq SS     Adj SS     Adj MS       F      P
gen             1      28.08      28.08      28.08   20.98  0.000
problem        19    6147.72    6147.72     323.56  241.77  0.000
gen*problem    19      25.43      25.43       1.34    1.41  0.111
Error        1160    1097.93    1097.93       0.95
Total        1199    7299.16
```

Table E.3 ANOVA table for strategy-3 (5000-8000), when initial population type used is ini_{rand}, with response $\log(DEV_b)$

```
Source         DF     Seq SS     Adj SS     Adj MS       F      P
gen             1     0.0037     0.0037     0.0037    1.18  0.292
problem        19    83.1659    83.1659     4.3772 1398.59  0.000
gen*problem    19     0.0595     0.0595     0.0031    0.73  0.790
Error        1160     4.9705     4.9705     0.0043
Total        1199    88.199
```

Table E.4 ANOVA table for strategy-4 (5000-8000), when initial population type used is ini_{rand}, with response $\log(DEV_b)$

```
Analysis of Variance for logdev, using Adjusted SS for Tests

Source         DF     Seq SS     Adj SS     Adj MS       F      P
gen             1     0.0334     0.0334     0.0334   24.40  0.000
problem        19    84.3398    84.3398     4.4389 3243.41  0.000
gen*problem    19     0.0260     0.0260     0.0014    0.19  1.000
Error        1160     8.1489     8.1489     0.0070
Total        1199    92.5481
```

Table E.5 ANOVA table for strategy-5 (5000-8000), when initial population type used is ini_{rand}, with response $\log(\text{DEV}_b)$

```
Analysis of Variance for logdev, using Adjusted SS for Tests

Source         DF     Seq SS     Adj SS     Adj MS       F      P
gen             1     0.1084     0.1084     0.1084    5.32  0.033
problem        19   105.6677   105.6677     5.5615  272.91  0.000
gen*problem    19     0.3872     0.3872     0.0204    1.52  0.071
Error        1160    15.5668    15.5668     0.0134
Total        1199   121.7301
```

APPENDIX F

STATISTICAL ANALYSES FOR COMPARING STRATEGIES FOR TSPPD

In this appendix, the resulting tables of the stattistical tests comducted for comparing the five strategy are given. The results of the Levene's homogeneity test and Tamhane's multiple comparison test are provided.

Table F.1 Levene's test results

Stopping Condition	Initial Population	Response	F	df1	df2	Sig.
2000	all	Log(DEV_b)	43.627	99	8900	0.000
2000	all	Log(CPU)	108.556	99	8900	0.000
2000	ini_{rand}	Log(DEV_b)	13.109	99	2900	0.000
2000	ini_{rand}	Log(CPU)	37.394	99	2900	0.000
5000	all	Log(DEV_b)	35.409	99	8900	0.000
5000	all	Log(CPU)	112.835	99	8900	0.000
5000	ini_{rand}	Log(DEV_b)	11.660	99	2900	0.000
5000	ini_{rand}	Log(CPU)	41.323	99	2900	0.000
8000	all	Log(DEV_b)	31.466	99	8900	0.000
8000	all	Log(CPU)	121.425	99	8900	0.000
8000	ini_{rand}	Log(DEV_b)	12.443	99	2900	0.000
8000	ini_{rand}	Log(CPU)	43.795	99	2900	0.000
Fixedbest	all	Log(DEV_b)	40.760	79	7120	0.000
Fixedbest	all	Log(CPU)	37.268	79	7120	0.000
Fixedbest	ini_{rand}	Log(DEV_b)	16.526	79	2320	0.000
Fixedbest	ini_{rand}	Log(CPU)	14.017	79	2320	0.000

Table F.2 Tamhane's test results for stopping condition 2000, when all initial population types are considered, with responses log(DEV_b) and log(CPU)

log(DEV_b)					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
1	2	-0.020	0.010	0.329	1	2	0.870	0.016	0.000
	3	0.039	0.009	0.000		3	0.889	0.016	0.000
	4	0.018	0.010	0.450		4	0.882	0.016	0.000
	5	-0.224	0.012	0.000		5	0.482	0.017	0.000
2	1	0.020	0.010	0.329	2	1	-0.870	0.016	0.000
	3	0.059	0.010	0.000		3	0.019	0.010	0.385
	4	0.038	0.010	0.001		4	0.012	0.010	0.917
	5	-0.204	0.013	0.000		5	-0.388	0.010	0.000
3	1	-0.039	0.009	0.000	3	1	-0.889	0.016	0.000
	2	-0.059	0.010	0.000		2	-0.019	0.010	0.385
	4	-0.021	0.010	0.287		4	-0.007	0.010	0.998
	5	-0.263	0.012	0.000		5	-0.407	0.010	0.000
4	1	-0.018	0.010	0.450	4	1	-0.882	0.016	0.000
	2	-0.038	0.010	0.001		2	-0.012	0.010	0.917
	3	0.021	0.010	0.287		3	0.007	0.010	0.998
	5	-0.242	0.012	0.000		5	-0.400	0.010	0.000
5	1	0.224	0.012	0.000	5	1	-0.482	0.017	0.000
	2	0.204	0.013	0.000		2	0.388	0.010	0.000
	3	0.263	0.012	0.000		3	0.407	0.010	0.000
	4	0.242	0.012	0.000		4	0.400	0.010	0.000

Table F.3 Tamhane's test results for stopping condition 5000, when all initial population types are considered, with responses log(DEV_b) and log(CPU)

log(DEV_b)					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
1	2	0.028	0.009	0.020	1	2	0.721	0.015	0.000
	3	0.058	0.009	0.000		3	0.747	0.015	0.000
	4	0.038	0.009	0.001		4	0.738	0.015	0.000
	5	-0.096	0.010	0.000		5	0.394	0.016	0.000
2	1	-0.028	0.009	0.020	2	1	-0.721	0.015	0.000
	3	0.029	0.009	0.014		3	0.026	0.010	0.076
	4	0.009	0.009	0.982		4	0.017	0.010	0.586
	5	-0.125	0.010	0.000		5	-0.327	0.010	0.000
3	1	-0.058	0.009	0.000	3	1	-0.747	0.015	0.000
	2	-0.029	0.009	0.014		2	-0.026	0.010	0.076
	4	-0.020	0.009	0.239		4	-0.009	0.010	0.988
	5	-0.154	0.010	0.000		5	-0.352	0.010	0.000
4	1	-0.038	0.009	0.001	4	1	-0.738	0.015	0.000
	2	-0.009	0.009	0.982		2	-0.017	0.010	0.586
	3	0.020	0.009	0.239		3	0.009	0.010	0.988
	5	-0.134	0.010	0.000		5	-0.343	0.010	0.000
5	1	0.096	0.010	0.000	5	1	-0.394	0.016	0.000
	2	0.125	0.010	0.000		2	0.327	0.010	0.000
	3	0.154	0.010	0.000		3	0.352	0.010	0.000
	4	0.134	0.010	0.000		4	0.343	0.010	0.000

Table F.4 Tamhane's test results for stopping condition 8000, when all initial population types are considered, with responses log(DEV_b) and log(CPU)

log(DEV_b)					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
1	2	0.046	0.009	0.000	1	2	0.660	0.015	0.000
	3	0.066	0.009	0.000		3	0.685	0.015	0.000
	4	0.045	0.009	0.000		4	0.676	0.015	0.000
	5	-0.079	0.010	0.000		5	0.378	0.015	0.000
2	1	-0.046	0.009	0.000	2	1	-0.660	0.015	0.000
	3	0.019	0.009	0.279		3	0.024	0.010	0.108
	4	-0.001	0.009	1.000		4	0.016	0.010	0.655
	5	-0.125	0.010	0.000		5	-0.282	0.010	0.000
3	1	-0.066	0.009	0.000	3	1	-0.685	0.015	0.000
	2	-0.019	0.009	0.279		2	-0.024	0.010	0.108
	4	-0.020	0.009	0.213		4	-0.008	0.010	0.992
	5	-0.144	0.010	0.000		5	-0.306	0.010	0.000
4	1	-0.045	0.009	0.000	4	1	-0.676	0.015	0.000
	2	0.001	0.009	1.000		2	-0.016	0.010	0.655
	3	0.020	0.009	0.213		3	0.008	0.010	0.992
	5	-0.124	0.010	0.000		5	-0.298	0.010	0.000
5	1	0.079	0.010	0.000	5	1	-0.378	0.015	0.000
	2	0.125	0.010	0.000		2	0.282	0.010	0.000
	3	0.144	0.010	0.000		3	0.306	0.010	0.000
	4	0.124	0.010	0.000		4	0.298	0.010	0.000

Table F.5 Tamhane's test results for stopping condition Fixedbest, when all initial population types are considered,with responses log(DEV_b) and log(CPU)

log(DEV_b)					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
2	3	-0.003	0.009	0.999	2	3	0.082	0.013	0.000
	4	-0.026	0.009	0.020		4	0.090	0.014	0.000
	5	-0.160	0.009	0.000		5	-0.093	0.014	0.000
3	2	0.003	0.009	0.999	3	2	-0.082	0.013	0.000
	4	-0.022	0.009	0.066		4	0.007	0.013	0.993
	5	-0.156	0.009	0.000		5	-0.175	0.013	0.000
4	2	0.026	0.009	0.020	4	2	-0.090	0.014	0.000
	3	0.022	0.009	0.066		3	-0.007	0.013	0.993
	5	-0.134	0.010	0.000		5	-0.183	0.013	0.000
5	2	0.160	0.009	0.000	5	2	0.093	0.014	0.000
	3	0.156	0.009	0.000		3	0.175	0.013	0.000
	4	0.134	0.010	0.000		4	0.183	0.013	0.000

APPENDIX G

STATISTICAL ANALYSES REGARDING THE EFFECT OF INITIAL POPULATION FOR TSPB

In this appendix, ANOVA tables and related plots for determining the effect of initial population type on solution quality and computation time are provided for each strategy when the stopping conditions are 2000 and Fixedbest.

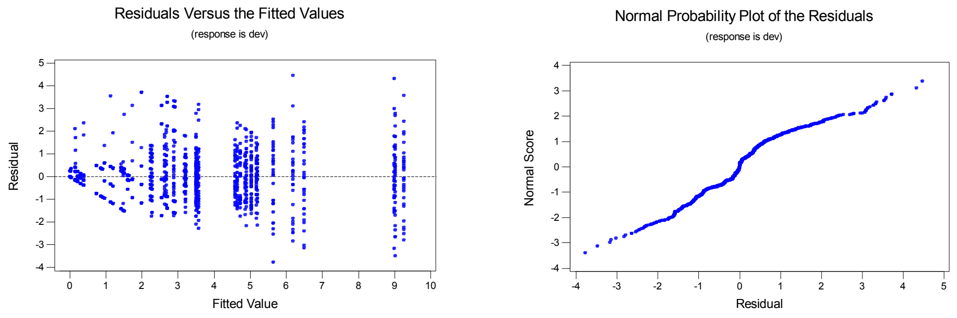


Figure G.1 Normality and residual plots for strategy-2, when stopping condition is 2000 with response DEV$_{opt}$

Table G.1 ANOVA table for strategy-2, when stopping condition 2000 with response DEV$_{opt}$

```
Analysis of Variance for dev, using Adjusted SS for Tests

Source            DF     Seq SS    Adj SS    Adj MS       F      P
initype            2       2.77      2.77      1.39    1.48  0.241
problem           19    9414.81   9414.81    495.52  528.03  0.000
initype*problem   38      35.66     35.66      0.94    1.03  0.416
Error           1740    1581.34   1581.34      0.91
Total           1799   11034.58
```

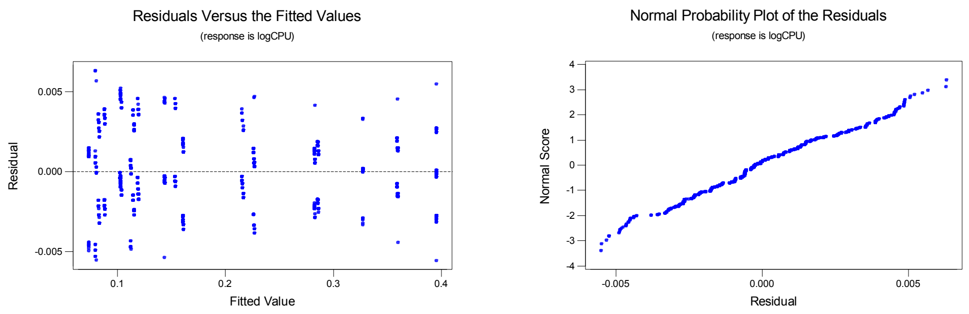


Figure G.2 Normality and residual plots for strategy-2, when stopping condition is 2000 with response log(CPU)

Table G.2 ANOVA table for strategy-2, when stopping condition 2000 with response log(CPU)

```
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source            DF      Seq SS      Adj SS      Adj MS       F       P
initype            2     0.00000     0.00000     0.00000    0.16   0.852
problem           19    17.74282    17.74282     0.93383 2.4E+05   0.000
initype*problem   38     0.00015     0.00015     0.00000    1.00   0.466
Error           1740     0.00685     0.00685     0.00000
Total           1799    17.74982
```

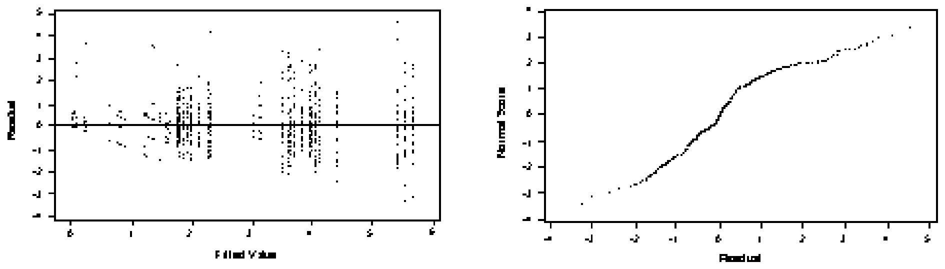


Figure G.3 Normality and residual plots for strategy-2, when stopping condition is Fixedbest with response DEV_{opt}

Table G.3 ANOVA table for strategy-2, when stopping condition Fixedbest with response DEV_{opt}

```
Analysis of Variance for dev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2      2.688      2.688      1.344    3.88  0.029
problem           19   4216.431   4216.431    221.917  640.06  0.000
initype*problem   38     13.175     13.175      0.347    0.62  0.969
Error           1740    979.526    979.526      0.563
Total           1799   5211.820
```

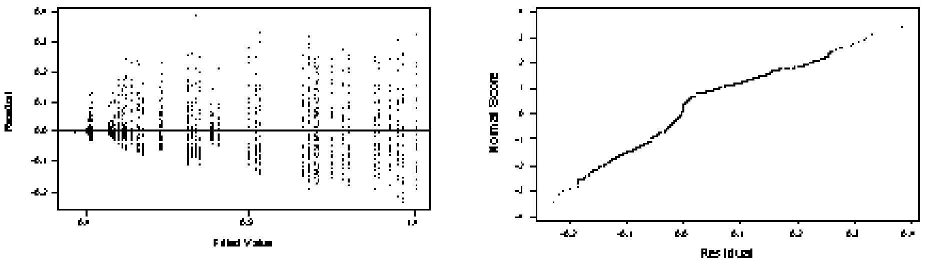


Figure G.4 Normality and residual plots for strategy-2, when stopping condition is Fixedbest with response log(CPU)

Table G.4 ANOVA table for strategy-2, when stopping condition Fixedbest with response log(CPU)

```
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0160     0.0160     0.0080    1.82  0.176
problem           19   173.5177   173.5177     9.1325 2073.89  0.000
initype*problem   38     0.1673     0.1673     0.0044    0.71  0.910
Error           1740    10.8451    10.8451     0.0062
Total           1799   184.5462
```

Table G.5 ANOVA tables for strategy-3

```
Stopping Condition 2000
Analysis of Variance for dev, using Adjusted SS for Tests

Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2       1.25       1.25       0.63    0.68  0.514
problem            19    6253.58    6253.58     329.14  356.36  0.000
initype*problem    38      35.10      35.10       0.92    1.06  0.370
Error            1740    1514.76    1514.76       0.87
Total            1799    7804.69
```

```
Stopping Condition 2000
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2     0.3963     0.3963     0.1981   96.89  0.000
problem            19    88.9540    88.9540     4.6818 2289.59  0.000
initype*problem    38     0.0777     0.0777     0.0020   43.22  0.000
Error            1740     0.0823     0.0823     0.0000
Total            1799    89.5103
```

```
Stopping Condition Fixedbest
Analysis of Variance for logdev, using Adjusted SS for Tests

Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2     0.0001     0.0001     0.0001    0.00  0.997
problem            19   104.5358   104.5358     5.5019  288.83  0.000
initype*problem    38     0.7239     0.7239     0.0190    1.48  0.031
Error            1740    22.4291    22.4291     0.0129
Total            1799   127.6888
```

```
Stopping Condition Fixedbest
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source             DF     Seq SS     Adj SS     Adj MS       F      P
initype             2     5.2165     5.2165     2.6083   75.35  0.000
problem            19   246.8849   246.8849    12.9939  375.40  0.000
initype*problem    38     1.3153     1.3153     0.0346    7.79  0.000
Error            1740     7.7325     7.7325     0.0044
Total            1799   261.1492
```

Table G.6 ANOVA tables for strategy-4

```
Stopping Condition 2000
Analysis of Variance for dev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2       0.50       0.50       0.25    0.12  0.887
problem           19   26520.46   26520.46    1395.81  676.13  0.000
initype*problem   38      78.45      78.45       2.06    0.97  0.530
Error           1740    3718.57    3718.57       2.14
Total           1799   30317.97
```

```
Stopping Condition 2000
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2    0.00005    0.00005    0.00003    9.08  0.001
problem           19   28.30425   28.30425    1.48970 5.1E+05  0.000
initype*problem   38    0.00011    0.00011    0.00000    0.80  0.803
Error           1740    0.00639    0.00639    0.00000
Total           1799   28.31081
```

```
Stopping Condition Fixedbest
Analysis of Variance for logdev, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0465     0.0465     0.0232    1.37  0.266
problem           19   152.6431   152.6431     8.0338  474.53  0.000
initype*problem   38     0.6433     0.6433     0.0169    1.07  0.351
Error           1740    27.4444    27.4444     0.0158
Total           1799   180.7774
```

```
Stopping Condition Fixedbest
Analysis of Variance for logCPU, using Adjusted SS for Tests

Source            DF     Seq SS     Adj SS     Adj MS       F      P
initype            2     0.0068     0.0068     0.0034    1.06  0.356
problem           19   142.8089   142.8089     7.5163 2350.12  0.000
initype*problem   38     0.1215     0.1215     0.0032    0.91  0.625
Error           1740     6.1046     6.1046     0.0035
                      Total           1799   149.0419
```

APPENDIX H

STATISTICAL ANALYSES REGARDING THE EFFECT OF STOPPING CONDITION FOR TSPB

In this appendix, ANOVA tables and related plots for determining the effect of stopping condition on solution quality and computation time are provided for each strategy when data from the stated initial population types is considered.

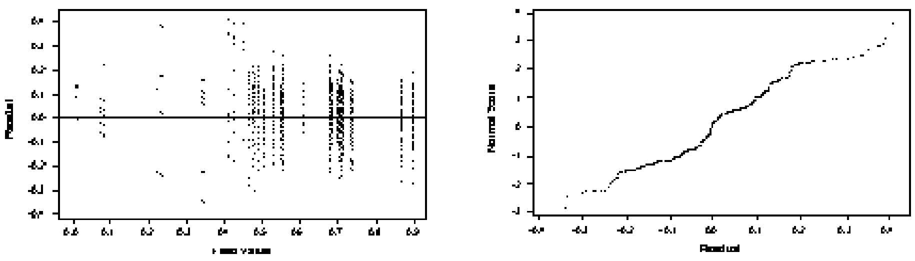


Figure H.1 Normality and residual plots for strategy-2, with response log(DEV_b)

Table H.1 ANOVA table for strategy-2 (5000-8000), when initial population type used is ini_{feas}, with response log(DEV_b)

```
Analysis of Variance for logdev, using Adjusted SS for Tests

Source         DF     Seq SS     Adj SS     Adj MS        F       P
gen             1     0.1407     0.1407     0.1407    23.34   0.000
problem        19   241.2442   241.2442    12.6971  2107.17   0.000
gen*problem    19     0.1145     0.1145     0.0060     0.55   0.938
Error        3560    38.7211    38.7211     0.0109
Total        3599   280.2204
```

Table H.2 ANOVA table for strategy-3 (5000-8000), when initial population type used is ini_{feas}, with response $\log(DEV_b)$

```
Analysis of Variance for logdev, using Adjusted SS for Tests

Source         DF     Seq SS     Adj SS     Adj MS       F      P
gen             1     0.0148     0.0148     0.0148   23.70  0.000
problem        19   215.8703   215.8703    11.3616 1.8E+04  0.000
gen*problem    19     0.0119     0.0119     0.0006    0.05  1.000
Error        3560    46.4143    46.4143     0.0130
Total        3599   262.3112
```

Table H.3 ANOVA table for strategy-4 (5000-8000), when initial population type used is ini_{feas}, with response $\log(DEV_b)$

```
Analysis of Variance for logdev, using Adjusted SS for Tests

Source         DF     Seq SS     Adj SS     Adj MS       F      P
gen             1     0.0342     0.0342     0.0342   24.27  0.000
problem        19   311.9561   311.9561    16.4187 1.2E+04  0.000
gen*problem    19     0.0268     0.0268     0.0014    0.09  1.000
Error        3560    54.0542    54.0542     0.0152
Total        3599   366.0713
```

APPENDIX I

STATISTICAL ANALYSES FOR COMPARING STRATEGIES FOR TSPB

In this appendix, the resulting tables of the stattistical tests comducted for comparing the five strategy are given. The results of the Levene's homogeneity test and Tamhane's multiple comparison test are provided.

Table I.1 Levene's test results

Stopping Condition	Initial Population	Response	F	df1	df2	Sig.
2000	all	DEV_{opt}	57.150	59	5340	.000
2000	all	Log(CPU)	152.275	59	5340	.000
2000	ini_{feas}	DEV_{opt}	23.907	59	1740	.000
2000	ini_{feas}	Log(CPU)	27.486	59	1740	.000
5000	all	DEV_{opt}	50.272	59	5340	.000
5000	all	Log(CPU)	155.339	59	5340	.000
5000	ini_{feas}	DEV_{opt}	22.010	59	1740	.000
5000	ini_{feas}	Log(CPU)	34.449	59	1740	.000
8000	all	DEV_{opt}	51.656	59	5340	.000
8000	all	Log(CPU)	122.764	59	5340	.000
8000	ini_{feas}	DEV_{opt}	20.743	59	1740	.000
8000	ini_{feas}	Log(CPU)	37.413	59	1740	.000
Fixedbest	all	DEV_{opt}	50.909	59	5340	.000
Fixedbest	all	Log(CPU)	43.114	59	5340	.000
Fixedbest	ini_{feas}	DEV_{opt}	18.533	59	1740	.000
Fixedbest	ini_{feas}	Log(CPU)	15.066	59	1740	.000

Table I.2 Tamhane's test results for stopping condition 2000, when all initial population types are considered, with responses DEV_{opt} and log(CPU)

DEV_b					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
2	3	0.568	0.076	0.000	2	3	0.066	0.010	0.000
	4	-2.397	0.113	0.000		4	-0.201	0.011	0.000
3	2	-0.568	0.076	0.000	3	2	-0.066	0.010	0.000
	4	-2.965	0.109	0.000		4	-0.267	0.010	0.000
4	2	2.397	0.113	0.000	4	2	0.201	0.011	0.000
	3	2.965	0.109	0.000		3	0.267	0.010	0.000

Table I.3 Tamhane's test results for stopping condition 5000, when all initial population types are considered, with responses DEV_{opt} and log(CPU)

DEV_b					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
2	3	0.409	0.062	0.000	2	3	-0.438	0.008	0.000
	4	-2.300	0.095	0.000		4	-0.099	0.006	0.000
3	2	-0.409	0.062	0.000	3	2	0.438	0.008	0.000
	4	-2.709	0.092	0.000		4	0.339	0.008	0.000
4	2	2.300	0.095	0.000	4	2	0.099	0.006	0.000
	3	2.709	0.092	0.000		3	-0.339	0.008	0.000

Table I.4 Tamhane's test results for stopping condition 8000, when all initial population types are considered, with responses DEV_{opt} and log(CPU)

DEV_b					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
2	3	0.325	0.060	0.000	2	3	-0.492	0.009	0.000
	4	-2.319	0.092	0.000		4	-0.116	0.007	0.000
3	2	-0.325	0.060	0.000	3	2	0.492	0.009	0.000
	4	-2.644	0.090	0.000		4	0.376	0.009	0.000
4	2	2.319	0.092	0.000	4	2	0.116	0.007	0.000
	3	2.644	0.090	0.000		3	-0.376	0.009	0.000

Table I.5 Tamhane's test results for stopping condition Fixedbest, when all initial population types are considered, with responses DEV_{opt} and log(CPU)

DEV_b					log(CPU)				
(I)	(J)	Mean (I-J)	Std. Error	Sig.	(I)	(J)	Mean (I-J)	Std. Error	Sig.
2	3	0.183	0.056	0.003	2	3	-0.447	0.012	0.000
	4	-2.368	0.088	0.000		4	-0.106	0.010	0.000
3	2	-0.183	0.056	0.003	3	2	0.447	0.012	0.000
	4	-2.551	0.088	0.000		4	0.341	0.011	0.000
4	2	2.368	0.088	0.000	4	2	0.106	0.010	0.000
	3	2.551	0.088	0.000		3	-0.341	0.011	0.000