A COMPARATIVE STUDY OF TREE ENCODINGS FOR EVOLUTIONARY COMPUTING

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

BY

ESİN SAKA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

JULY 2005

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. Ayşe Kiper
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.

Assoc. Prof. Dr. İsmail Hakkı Toroslu
Co-Supervisor

Assoc. Prof. Dr. Göktürk Üçoluk
Supervisor

Examining Committee Members

Assoc. Prof. Dr. İsmail Hakkı Toroslu(ODTU-CENG) _____

Assoc. Prof. Dr. Göktürk Üçoluk (ODTU-CENG) _____

Dr. Onur Tolga Şehitoğlu (ODTU-CENG) _____

Dr. Meltem Turhan Yöndem (ODTU-CENG) _____

Y. Müh. Arslan Arslan (LOGO) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ESİN SAKA

Signature :

ABSTRACT

A COMPARATIVE STUDY OF TREE ENCODINGS FOR EVOLUTIONARY COMPUTING

SAKA, ESİN

M.Sc., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Göktürk Üçoluk

Co-Supervisor: Assoc. Prof. Dr. İsmail Hakkı Toroslu

JULY 2005, 82 pages

One of the most important factors on the success of evolutionary algorithms (EAs) about trees is the representation of them. The representation should exhibit efficiency, locality and heritability to enable effective evolutionary computing. Neville proposed three different methods for encoding labeled trees. The first one is similar with Prüfer's encoding. In 2001, it is reported that, the use of Prüfer numbers is a poor representation of spanning trees for evolutionary search, since it has low locality for random trees. In the thesis Neville's other two encodings, namely *Neville branch numbers* and *Neville leaf numbers*, are studied. For their performance in EA their properties and algorithms for encoding and decoding them are also examined. Optimal algorithms with time and space complexities of $O(n)$, where n is the number of nodes, for encoding and decoding Neville branch numbers are given. The localities of Neville's encodings are investigated. It is shown that, although the localities of Neville branch and leaf numbers are perfect for star type trees, they are low for random trees. Neville branch and Neville leaf numbers are compared with other codings in EAs and SA for four problems: 'onemax tree problem', 'degree-constrained minimum spanning tree problem', 'all spanning trees problem' and 'all degree constrained spanning trees problem'. It is shown that, neither Neville nor Prüfer encodings are suitable for EAs. These encodings are suitable for

only tree enumeration and degree computation. Algorithms which are timewise and space-wise optimal for 'all spanning trees problem' (ASTP) for complete graphs, are given by using Neville branch encoding. Computed time and space complexities for solving ASTP of complete graphs are $O(n^{n-2})$ and $O(n)$ if trees are only enumerated and $O(n^{n-1})$ and $O(n)$ if all spanning trees are printed , respectively, where n is the number of nodes. Similarly, 'all degree constrained spanning trees problem' of a complete graph is solvable in $O(n^{n-1})$ time and $O(n)$ space.

Keywords: evolutionary algorithms, genetic algorithms, tree representation, one-max tree problem, degree constrained minimum spanning tree problem, all spanning trees problem

ÖZ

EVRİMSEL ALGORİTMALAR İÇİN AĞAÇ YAPILARININ KARŞILAŞTIRMALI ÇALIŞMASI

SAKA, ESİN

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Assoc. Prof. Dr. Göktürk Üçoluk

Ortak Tez Yöneticisi: Assoc. Prof. Dr. İsmail Hakkı Toroslu

TEMMUZ 2005, 82 sayfa

Ağaçlarla ilgili evrimsel algoritmaların başarısıdaki en önemli faktörlerden birisi ağaç yapılarının gösterimidir. Etkin bir evrimsel hesaplama için, gösterim verimlilik, bölgesellik ve atasallık özelliklerini barındırmalıdır. Neville, etiketli ağaç yapılarının kodlanması için üç çeşit metod sunmuştur. Bunlardan birincisi, Prüfer kodlaması ile benzerdir. 2001 yılında, ağaç yapılarının gösterimi için Prüfer sayılarının kullanımının, Prüfer sayılarının rastgele ağaçlar için düşük bölgeselliği nedeniyle, evrimsel aramada yetersiz bir yöntem olduğu önerilmiştir. Bu tezde, Neville'in diğer iki kodlama yöntemi, yani *Neville dal numaraları* ve *Neville yaprak numaraları* çalışıldı. Evrimsel algoritmalardaki performansları için özellikleri, kodlama ve kodlanmış yapıyı oluşturma algoritmaları incelendi. Neville dal numaralarının kodlanması ve kodlanmış yapının oluşturulması için, n düğüm sayısı iken, $O(n)$'lik zaman ve yer karmaşıklığına sahip optimal algoritmalar verildi. Neville'in kodlanmışlarının bölgeselliği araştırıldı. Neville dal ve yaprak numaralarının bölgeselliğinin yıldız tipi ağaçlar için mükemmel olmasına rağmen, rastgele ağaçlar için düşük olduğu gösterildi. Neville dal ve Neville yaprak numaraları evrimsel algoritmalardaki diğer kodlamalarla dört problem üzerinde karşılaştırıldı: 'bir fazla ağaç problemi', 'derece kısıtlı en küçük tüm ağaç problemi', 'bütün tüm ağaçlar

problemi' ve 'derece kısıtlı bütün tüm ağaçlar problemi'. Ne Neville ne de Prüfer kodlamalarının evrimsel algoritmalar için uygun olduğu gösterildi. Bu kodlamalar sadece derece hesaplamalarında ve ağaçların birer birer sayımında uygundur. Bütün tüm ağaçlar problemi (ASTP) için zaman ve yer bakımından tam ağaçlarda optimal algoritmalar, Neville'in dal kodlaması kullanılarak verildi. n düğüm sayısını gösterirken, tam ağaçlarda ASTP'yi çözmek için hesaplanan zaman ve yer karmaşıklıkları, sırasıyla, ağaçlar sadece kod olarak basılırsa $O(n^{n-2})$ ve $O(n)$; ağaçların kendisi basılırsa $O(n^{n-1})$ ve $O(n)$'dir. Benzer şekilde, tam ağaçlarda 'derece kısıtlı bütün tüm ağaçlar problemi'. $O(n^{n-1})$'lik zaman ve $O(n)$'lik yerde çözülebilir.

Anahtar Kelimeler: evrimsel algoritmalar,genetik algoritmalar, ağaç yapılarının gösterimi, bir fazla ağaç problemi, derece kısıtlı en küçük tüm ağaç problemi, bütün tüm ağaçlar problemi, derece kısıtlı bütün tüm ağaçlar problemi

To science...

ACKNOWLEDGEMENTS

I would like to thank Göktürk ÜÇOLUK and Hakkı TOROSLU for their precious supervision and Onur Tolga ŞEHİTOĞLU for his valuable patience to my questions. You were very important for the very early steps of my academic life.

That was the smiling faces and infinite support of my friends that made me not to give up at the difficult times.

Without love, support, encouragement and trust of my family, I wouldn't do MSc, and this study would not exist at all.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURES

List of Algorithms

CHAPTER 1

INTRODUCTION

An evolutionary algorithm (EA) works on populations of candidate solutions to a given problem. If an encoding/decoding based EA is used, the way you represent a genome deeply affects the performance. It is expected that the representation exhibits efficiency, locality, heritability and closure.

Labeled trees are used in several practical and theoretical areas of computer science. For example, in some networks, like Ethernet, each terminal must be connected to any other terminal and no cycles can exist. Furthermore, since every terminal is uniquely identified, labeling the nodes is necessary. Many algorithms require generating spanning trees of labeled graphs. In 1918, Prüfer showed a mapping between any n-node labeled tree structure and (n-2)-tuples of node labels. This mapping is one-to-one and onto. In other words, there is a unique code for each labeled tree and each code represents a unique labeled tree. This mapping is also used for the proof of Cayley's theorem ([4]). In 1953, Neville proposed three different labeled tree encodings[15]. Neville's first encoding style is similar to Prüfer's encoding. The most studied one is Prüfer coding. In 2001, it is reported that, Prüfer technique is a poor representation of spanning trees for evolutionary search [8].

This thesis studies Neville's uninvestigated encodings, namely *Neville branch numbers* and *Neville leaf numbers*. It analyzes the properties of Neville's encodings, examines algorithms for encoding and decoding them, investigates the localities by random walks, inspects suitability of these techniques for EAs, and compares Neville numbers with other codings in EAs and simulated annealing (SA) for four problems:

- One-max tree problem (OMTP).

- Degree-constrained minimum spanning tree problem (DCMSTP).
- All spanning trees problem (ASTP).
- All degree constrained spanning tree problem (ADCSTP).

Neville's techniques have some benefits. Firstly, there is a unique bijective mapping between the sets of Neville branch numbers and labeled trees and also between the sets of Neville leaf numbers and labeled trees. Thus, Neville numbers are closed under classical EA operations. In other words, after classical mutation and crossover, what you get is again a tree. But if graph is not complete, then encoded tree may not be a valid tree. Secondly, computation of degrees of nodes are simple. Degree of a node of encoded tree is one more than the number of times the node appears in Neville's encodings. Therefore, Neville's encodings has an advantage in degree constraint computation. This computation is valid for all prüfer-like encodings.

However, Neville's encoding have some problems: low locality and heritability: *Locality* of an encoding is the relatedness of the phenotype (the code) and the genotype (the tree). If a small change in genotype results in a small change in phenotype, a coding has high locality. High locality means, coding is suitable for evolutionary search. *Heritability* is the relatedness of parent phenotypes and children phenotypes. If children are similar to parents, a code is said to have high heritability with respect to crossover operator. Heritability is low where locality is low. Unfortunately, experiments showed that, Neville numbers have low locality. Locality analysis for Neville branch numbers and Neville leaf numbers are done by random walks. Firstly random walks through the search space of Neville's encodings of random trees are performed. Low locality is observed. Then the search space is explored whether the locality changes according to the structure of goal tree or not. This research showed that although both Neville branch numbers and Neville leaf numbers have perfect locality with stars, their locality is low for random trees. Therefore Neither Neville nor Prüfer numbers are suitable for EAs.

In the last years there is an increase in the interest to the Prüfer-like encodings. However, the success of encoding depends on the problem. To figure out the negative effect of low locality properties and show when these encodings can be useful, experiments were done on OMTP, DCSTP, ASTP and ADCSTP. With OMTP Neville's numbers performed perfect with stars. However, Neville's numbers are not successful when problem size is increased and goal

is not a star. Thus, Neville numbers are not suitable for OMTP.

As mentioned above, although degree constraint computation is simple with Neville's encodings, low locality caused Neville numbers be unsuccessful with solving DCMSTP with EAs and SA. Since the degree constraint in DCMSTP decreases the probability of having a star as best solution, Neville's encodings are not suitable for DCMSTP, also.

The bijection property of Neville's encodings makes them good choices for tree enumeration. Generating a code has $O(n)$ time and $O(n)$ space complexities, where n is the number of nodes of complete graph G. Since Neville numbers are closed, when all numbers of size $n-2$ are generated, it means that, all spanning trees of G are enumerated with $O(n^{n-2})$ time and $O(n)$ space complexities. The best known algorithm has $O(s+n+e)$ time and $O(n+e)$ space complexities [25], where s is the number of spanning trees and e is the number of edges. For a complete graph, since $s = n^{n-2}$, time and space complexities become $O(n^{n-2})$ and $O(n+e)$, respectively. Both algorithms have the same time complexity but our algorithm is better at space complexity. If all spanning trees are printed, each code is decoded. There were no reported algorithms for encoding and decoding Neville branch numbers and Neville leaf numbers. At Section 3.1.1, the pseudo-code given at Algorithm 12 is an algorithm to decode a Neville branch number and at Section 3.1.1 the pseudo-code given at Algorithm 11 is an algorithm to encode a Neville branch number with $O(n)$ time and space complexities. These are the best encoding and decoding algorithms for Neville branch numbers. When Neville branch numbers are used, time and space complexities of our solution to ASTP and printing all spanning trees are $O(n^{n-1})$ and $O(n)$, respectively. The optimal algorithm which enumerates all spanning trees by outputting all edges of each spanning tree was reported to have $O(sn+n+e)$ time and $O(n+e)$ space complexities [5]. For a complete graph, the time and space complexities becomes: $O(n^{n-1}+n+e)$ and $O(n+e)$, respectively. Thus our algorithm is better for complete graphs. When degree constraint is added, ADCSTP is solved with $O(n^{n-1})$ and $O(n)$ time and space complexities and resulting trees are printed or ADCSTP is solved with $O(n^{n-2})$ and $O(n)$ time and space complexities and only codes are given. So, time and space optimal algorithms for ASTP and ADCSTP of weighted or unweighted complete graphs are presented by using Neville branch numbers at Sections 4.3 and 4.4. But when the graph is an incomplete sparse graph, they are not the optimal solutions.

The thesis is structured as follows: Chapter 2 includes the background information on EAs and SA, existing encodings, history of Prüfer's and Neville's encodings and definitions

of problems studied. Existing encodings for labeled trees are given at section 2.3. Chapter 3 contains properties, encoding and decoding algorithms and locality analysis of Neville's encodings. Applications and results are given at Chapter 4. Finally, Chapter 5 concludes the thesis and presents possible future directions. In the Appendices A and B, data set and parameters used are given, respectively.

CHAPTER 2

BACKGROUND

2.1 EVOLUTIONARY ALGORITHMS

Evolutionary algorithms (EAs) are search methods. They take inspiration from evolution in the biological world. EAs simulate the evolution of individual by processes of selection and reproduction. EAs are used in both search and learning. Instead of one potential solution, they involve search from a group of solutions. This is why they are different from traditional optimization techniques. *Individual* is a candidate solution to a problem. *Population* is the set of individuals.

An EA starts with an initial population. At each time step, new individuals are generated by crossover and mutation. New generation is selected according to a fitness function and selection mechanism. In summary, a typical EA runs the pseudo-code given in Algorithm 1.

There are a variety of EAs ([26], [11]):

- **Genetic programming (GP):** Evolves programs.
- **Evolutionary programming(EP):** Focuses on optimizing continuous functions without crossover.
- **Evolutionary strategies(ES):** Focuses on optimizing continuous functions with crossover.
- **Genetic algorithms(GAs):** Focuses on optimizing general combinatorial problems.

Algorithm 1 The evolutionary algorithm.

inputs: *a problem*

outputs:*a solution state*

1: generation no = 0
2: randomly generate initial population
3: **while** generation no $<$ MAXIMUM GENERATION NO **do**
4: generation no = generation no + 1
5: generate children *(includes crossover and mutation)*
6: select next generation
7: statistics
8: **end while**

2.1.1 FITNESS FUNCTION

Fitness function is one of the major parameters which affects the success of EAs deeply. It plays the role of the environment, computes a fitness value for a given individual and quantifies the optimality of the solution. It measures the (potential for) reproductive success of the individual in a given environment and the individual is ranked against all the other individuals depending on the fitness value. Also, selection is basically based on the fitness.

The fitness function may additionally depend on different side conditions/constraints and stochastic influences (fitness noise/noisy fitness). The hope is that fitness function correlates closely with the algorithm's goal and there is a correlation between the fitness of the parents and offspring. Otherwise the search would be essentially a random walk search. Also, fitness function should be computed quickly. Because, speed of execution is very important, as a typical EA iterates several times in order to produce a usable result for a non-trivial problem.

2.1.2 CROSSOVER

Crossover is an information exchange between individuals[1]. In a crossover routine, two individuals are selected. They are called *parents*. Two parents generates individuals, which are called *offsprings*.

2.1.2.1 K-POINT CROSSOVER

Simple one-point crossover proceeds in two steps. Firstly, two parents in the mating pool are mated. Secondly, these parents exchange genes as follows: randomly a position p, where $1 \leq p \leq chromosome_{length} - 1$, is selected. Two new individuals are created by swapping all genes between position $p + 1$ and $chromosome_{length}$. For example, let $I_1 = 1\ 3\ 2\ 4\ 5$ and $I_2 = 5\ 2\ 1\ 3\ 3$ be two mated parents. and let $p = 3$ be the cross point, which is indicated by the separator symbol $|$. Then the parents are:

$$I_1 = 1\ 3\ 2 \mid 4\ 5$$

$$I_2 = 5\ 2\ 1 \mid 3\ 3$$

The children are:

$$C_1 = 1\ 3\ 2 \mid 3\ 3$$

$$C_2 = 5\ 2\ 1 \mid 4\ 5$$

If it is a *multi-point crossover*, multiple random points are used, instead of one. For example for the above parents I_1 and I_2, a 3 point crossover with $p_1 = 1$, $p_2 = 3$ and $p_3 = 4$ is:

$$I_1 = 1 \mid 3\ 2 \mid 4 \mid 5$$

$$I_2 = 5 \mid 2\ 1 \mid 3 \mid 3$$

The children are:

$$C_1 = 1 \mid 2\ 1 \mid 4 \mid 3$$

$$C_2 = 5 \mid 3\ 2 \mid 3 \mid 5$$

2.1.2.2 UNIFORM CROSSOVER

Uniform crossover combines genes sampled uniformly from the two parents. The crossover mask is generated as a random bit string with each bit chosen at random from either parent and independent of other genes.

Parents for crossover can be selected by different selections mechanisms. Two of them are *random selection* and *roulette wheel selection*.

2.1.2.3 RANDOM SELECTION

Randomly two individuals are selected as parents.

Algorithm 2 Random Selection (for crossing)

input: *population size and population*

output: *the id of individual selected as parent*

return a random integer i such that $1 \leq i \leq populationsize$

2.1.2.4 ROULETTE WHEEL SELECTION

Parents are selected according to their fitness values. Fitter individuals have higher probability for being chosen as a parent.

In the Algorithm 3, sum of the fitnesses of all individuals are computed. *Partial sum* is a value depending on fitnesses of individuals. Partial sum of first individual is, it's fitness value. If partial sum of n^{th} individual is P_n, then partial sum of $n + 1^{th}$ individual is the sum of partial sum of n^{th} individual and fitness value of $n + 1^{th}$ individual. According to the given procedure, partial sums are evaluated for each individual. A random rational number is generated between 0 and the sum of the fitnesses. The individual which has corresponding partial sum is chosen as a parent.

Algorithm 3 Roulette Wheel Selection

input: *population size, sum of fitnesses and population*

output: *the id of individual selected as parent*

partial sum is a real number with initial value 0.0

rand out is a real number

i is an integer

rand out = a random real number greater than or equal to 0 and less than sum of the fitnesses

for i=1, partsum=0; i$\leq$ population size and partial sum $<$ rand out; i++ **do**

partsum += pop[i].fitness;

end for

i = i - 1

return i

2.1.3 MUTATION

Mutation is an operator which slightly changes the genotype. Randomly choosing a gene and randomly assigning a new value is basic mutation. Mutation is mostly important for local

search. Mutation probability defines the occurrence frequency.

2.1.4 SELECTION

Selection is making a decision for the individual for the next generation. It relies on the fitness function used and selection mechanism applied.

2.1.4.1 $(\lambda + \mu)$ SELECTION

$(\lambda + \mu)$ selection is a deterministic selection, in which only the fittest individuals survive. Deterministic selections leads to a fast convergence. The population size is represented as μ and the number of children generated as λ. In $(\lambda + \mu)$ selection, the best μ individual, which have higher fitnesses among the $(\lambda + \mu)$ individuals, are selected to continue to the next generation by a competition, between both parents and children.

2.1.4.2 (λ, μ) SELECTION

(λ, μ) selection is similar to $(\lambda + \mu)$ selection. Similarly, λ children is generated, (λ, μ) selection again chooses the best μ, however the competition is only among the children. Parents are thrown away.

2.2 SIMULATED ANNEALING

Simulated annealing (SA) is a kind of search algorithm. Initially a solution is generated. At each step, randomly a solution candidate is generated. If the new one has higher fitness value, then it is *better* and it is assigned as current solution. If it is not better, it is assigned as the current solution with a probability depending on the difference between the solutions and the time step. This probability depending on temperature T is the metropolis probability:

$$P(T) = e^{\frac{next\_fitness - current\_fitness}{T}}$$

In summary, a typical simulated-annealing algorithm runs the following pseudo-code:

Algorithm 4 The simulated annealing search algorithm.

inputs: *a problem* and *schedule* which is a mapping from time to temperature T

outputs: *a solution state*

step_count = 0

initialize initial population

while goal_not_found or maximum_step_count_is_not_reached **do**

step_count = step_count + 1

temperature = schedule(step_count)

if temperature is 0 **then**

return current_solution

end if

randomly_generate_next_solution

if next_fitness>current_fitness **then**

current_solution=next_solution

else

current_solution=next_solution with probability

$P(T) = e^{\frac{next\_fitness - current\_fitness}{T}}$

end if

statistics

end while

2.3 TREE ENCODINGS

A *tree* is an undirected fully connected graph which does not contain any closed cycles. There are many problems in terms of finding the optimal tree satisfying some given constrains within a graph. EAs can be used for these problems provided that a suitable representation is chosen, since in EAs, the way a tree is represented strongly affects the success and efficiency of finding the solution.

Suppose that a tree has n *nodes* (vertices). If these nodes are named from 1 to n, this tree is a *labeled tree*. In other words, suppose $T = (N, E)$ is a tree, where $N = 1, ..., n$ is the set of n nodes (vertices) and $E = 1, ..., n-1$ is the set of edges (arcs). Since the nodes are named from 1 to n, the tree is called *labeled tree*. Many algorithms require generating spanning trees of labeled graphs. Since every tree can be represented as a labeled tree, from now on, the word tree is used instead of labeled tree.

A tree can be represented as a list of edges, as a vector of edges, according to predecessors and/or brothers or by encoding based on neighborhood. Representing tree as a list of numbers, ie. representing a tree as a code, makes it suitable for evolutionary algorithms, since, this code code can be used for the *genotype*, in other words as the *chromosome*, for the evolutionary algorithm. Choosing such a linear representation enables the use of a GA. So we can make use of classical versions of evolutionary operators like cross-over and mutation.

This section firstly provides some criteria to evaluate tree representations, especially for EA, continues with descriptions of some possible representations and lastly includes some prüfer-like encodings, which are studied in this thesis, in more detail.

2.3.1 CRITERIA TO SELECT TREE REPRESENTATIONS IN EVOLUTIONARY ALGORITHMS

The representation deeply affects the performance of a solution algorithm. If a representation is being searched for an EA, efficiency, locality, heritability and closure properties are considered.

2.3.1.1 EFFICIENCY

Efficiency is the cost compared to what is gained. It this section, efficiency expectations are analyzed according to evolutionary computation.

If an encoding is used, computations about it should be with minimum cost.

When considering an encoding efficient or not, three complexity analysis should be done:

- The cost of computing fitness.
- The cost of computing the genotype from the phenotype.
- The cost of computing the phenotype from the genotype.

The evaluation of fitness depends on the problem. This is why, some encodings are efficient on some problems. The cost of computing genotype and phenotype are two characteristics of encoding scheme.

2.3.1.2 LOCALITY

Locality of an encoding can be described as the relatedness of the phenotype (the code) and the genotype (the tree). If a small change in genotype results in a small change in phenotype, a coding has high locality. High locality means, coding is suitable for evolutionary search. Because with each mutation, a new individual similar to mutant comes out and search advances step by step.

2.3.1.3 HERITABILITY

If offspring phenotypes consist mostly of parents' phenotypes, a code is said to be have high heritability with respect to crossover operator.

2.3.1.4 CLOSURE

If after crossover and mutation, children are still valid individual representations, representation is said to be *closed* according to the problem. For example, let T_1 and T_2 be two trees. After applying crossover operator on them, if children are again trees, the representation is closed. But if the children are graphs which may contain cycles, then representation is not closed. If a representation is not closed under defined crossover and mutation, at least the cost of defining suitability and removing or deleting the individual should be small.

Before describing the encodings studied in the thesis, let us provide descriptions of some other possible tree encodings.

2.3.2 TREE REPRESENTATIONS

First let us give four different representations [17] and [13], which are not closed under classical GA operators.

2.3.2.1 ARC LIST REPRESENTATION

In this representation, a list $E = 1, ..., n-1$ keeps the id's of edges in the tree. Suppose, E is used as a genotype and crossover operation is applied on it. It is not guaranteed that, resulting children will be trees, they may include cycles.

2.3.2.2 ARC VECTOR REPRESENTATION

This is another simple representation. A vector E has an entry for each edge in the graph G. If spanning tree T of G includes an edge, its entry is 1, 0 otherwise. Clearly, $n-1$ elements of E need to have value of 1.

2.3.2.3 PREDECESSOR REPRESENTATION

It is also possible to designate a node r as the *root node* of the tree. Then encode the tree according to the immediate predecessors. Let i be a node in the tree. For each i keep $p(i)$, which is the immediate predecessor in the path from r to i.

2.3.2.4 ORIENTED TREE REPRESENTATION

In this representation, two predecessor pointers are associated with each node i, $LSon(i)$ and $RBroth(i)$. $LSon(i)$ is the left most son of i and $RBroth(i)$ is the rightmost son of i. A leaf node has an $LSon(i) = -i$ and A particular node which is the last node in one level of this structure tree has its $RBroth(i)$ as the negative of its parent (predecessor) in the rooted tree.

2.3.3 PRÜFER'S ENCODING

One mostly known encoding of a tree is Prüfer's encoding. Suppose $T = (N, E)$ is a tree where $N = 1, ..., n$ is the set of n nodes and $E = 1, ..., n-1$ is the set of $n-1$ edges. The *Prüfer number*, $P(T)$, is the $n-2$ digit number which encodes the tree T with Prüfer's encoding scheme [18]. Digits of Prüfer number are between 1 and n.

If there is an edge between all nodes, the graph is called *complete*. In 1889, Cayley proved that the number of spanning trees of an undirected complete graph with n vertices is n^{n-2} ([4], pp.103-104). In 1918, Prüfer introduced a one-to-one mapping between spanning trees and Prüfer numbers. This mapping proved the Cayley's theorem more elegantly ([4], pp.104-106).

If a tree-code is a sequence of (n-2) node labels and the code is computed by iteratively deleting the leaves of the tree in some deterministic order, the tree-code is called Prüfer-like.

2.3.3.1 ENCODING A TREE BY USING PRÜFER NUMBERS

The *degree* of a node x is the number of edges connected to x. Any node which has degree 1 is called *leaf*. A tree can be encoded as a Prüfer number by the Algorithm 5. If there exist less than 3 nodes, the algorithm is trivial. At step 3 of the algorithm, since i is a leaf, there exists exactly one j. An algorithm, by Kilingsberg, for constructing a tree with n nodes from its Prüfer number can also be adopted to encode the tree by its Prüfer numbers [3], [16] (p. 271).

Algorithm 5 The Construction of the Prüfer Number from a Tree

input: *the labeled tree T of order n.*

output: *the $n-2$ digit Prüfer number $P(T)$ of T*, where $1 \leq digit \leq n$.

1: **while** There exists more than 2 nodes **do**
2: Let i be the lowest numbered leaf in T.
3: Let j be the node connected to i. Add label of j as the rightmost digit of $P(T)$.
4: Remove i and the edge connecting i and j from T and from further considerations.
5: **end while**

There exists an efficient implementation of Algorithm 5 with time complexity $O(nlogn)$. It uses a priority queue implemented in a heap to hold the leaf nodes [19].

Let us demonstrate Algorithm 5 with a brief example. Let us generate Prüfer number of tree T given at Figure 2.1. The smallest labeled leaf is 2. Remove 2 and record 8, $P(T) = 8$. Then the smallest node is 3 with neighbor 4, so, $P(T) = 8\ 4$ Similarly, remove 4 5 7 6 one by one and record 1 6 6 1 respectively. Thus, $P(T) = 8\ 4\ 1\ 6\ 6\ 1$.

2.3.3.2 DECODING PRÜFER NUMBERS

It is possible to decode Prüfer number and get the unique tree by the Algorithm 6.

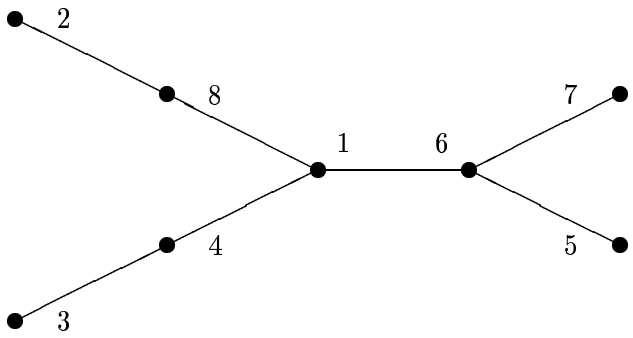


Figure 2.1: Example tree

Let us demonstrate the construction for the example tree in Figure2.1. Given $P(T) =$ 8 4 1 6 6 1, The lowest eligible no is 2. So $T = (2, 8)$ 8 becomes eligible. Then the next lowest eligible node is 3, so add $(3, 4)$ to T. $T = (2, 8), (3, 4)$. 4 becomes eligible. Then take 4. $T = (2, 8), (3, 4), (4, 1)$ but 1 is still not eligible. Similarly add edges $(5, 6)$, $(7, 6)$ and $(6, 1)$. Then 1 and 8 remains eligible, add $(1, 8)$. So $T = (2, 8), (3, 4), (4, 1), (5, 6), (7, 6), (6, 1), (1, 8)$

Algorithm 6 The Construction of the Tree from Prüfer Number

input: *a Prüfer number $P(T)$ with $n - 2$ digits*, where $1 \leq digit \leq n$

output: *the labeled tree T of order n.*

1: Let all node labels not part of $P(T)$ be designated as eligible for consideration.
2: **while** There exists more than 0 digits in $P(T)$ **do**
3: Let i be the lowest eligible node label. Let j be the leftmost digit of $P(T)$.
4: Add the edge (i, j) to T.
5: Designate i as no longer eligible and remove left most digit j from $P(T)$.
6: **if** j does not occur anywhere in $P(T)$ **then**
7: Designate j as eligible.
8: **end if**
9: **end while**
10: There are exactly 2 node labels ,k and l, which are eligible. Add the edge k and l to T.

2.3.3.3 PROPERTIES OF PRÜFER NUMBERS

As mentioned, a tree of n nodes is encoded as an $n - 2$ digit Prüfer number. Each digit can get values between 1 and n. So, there exist n^{n-2} Prüfer numbers for a graph of n nodes.

Prüfer numbers have some benefits. Every tree can be represented by a unique Prüfer

number. Every Prüfer number represents exactly one tree. Only trees are represented by Prüfer numbers. All trees are equally represented. Degree of a node is one more than the number of times the node appears at Prüfer number.

However, they also have some disadvantages. They have low locality and low heritability.

The definition of locality is given in Section 2.3.1.2. A coding has high locality if small changes in genotype (code) -like mutation- changes the corresponding phenotype (tree) slightly. Several researches like [17], [20], [21], [19] pointed out that Prüfer numbers has poor locality with respect to conventional position-by-position mutation.But in fact it is seen that Prüfer numbers only have high locality if they encode stars [19].

2.3.4 NEVILLE'S ENCODINGS

Neville proposed three different methods for encoding trees, in 1952 [15]. These methods also prove Cayley's theorem. The first method is similar to Prüfer's method. Other two methods differ in the order of removing leaves. Some studies call these encodings as Neville's second or third encoding[3]. However, these numbers can be confused. Thus, they are named as Neville branch encoding and Neville leaf encoding, according to the approaches they use when removing leaves.

2.3.4.1 NEVILLE'S BRANCH ENCODING

This method is the second one given in the Neville's study [15] and named as third encoding at paper [3]. In this method, firstly the leaf with smallest label is chosen. It is removed and it's neighbor is recorded. If the neighbor becomes a leaf, same procedure is applied. Otherwise, the smallest leaf on the remaining tree is searched and the above process is applied on it. This methods deletes the leaves branch by branch. So it is called *Neville branch encoding*. Codes generated are *Neville branch numbers*.

Encoding Neville Branch A tree can be encoded as a Neville branch number by the Algorithm 7. If there exist less than 3 nodes, the algorithm is trivial. At step 3 of the algorithm, since i is a leaf, there exists exactly one j.

When the Algorithm 7 is applied in the example tree of Figure 2.1, The nodes are removed in the order of 2, 8, 3, 4, 1, 5 and 8, 1, 4, 1, 6, 6, is recorded respectively. So $NB(T) =$ 8 1 4 1 6 6.

Algorithm 7 The Construction of the Neville Branch Number from a Tree

input: *the labeled tree T of order n.*

output: *the $n-2$ digit Neville branch number $P(T)$ of T*, where $1 \leq digit \leq n$.

1: Let $node\_to\_be\_removed$ be the lowest numbered leaf in T.
2: **while** There exists more than 2 nodes **do**
3: Let j be the node connected to $node\_to\_be\_removed$. Add label of j as the rightmost digit of $P(T)$.
4: Remove $node\_to\_be\_removed$ and the edge connecting $node\_to\_be\_removed$ and j from T and from further considerations.
5: **if** j becomes a leaf **then**
6: Let $node\_to\_be\_removed$ be j.
7: **else**
8: Let $node\_to\_be\_removed$ be the lowest numbered leaf in T.
9: **end if**
10: **end while**

Decoding Neville Branch The Algorithm 8 decodes Neville branch numbers.

2.3.4.2 NEVILLE'S LEAF ENCODING

This encoding technique is given at Neville's paper as third method [15] and named as Neville's second method by Deo and Micikevicius [3] This technique sorts the available leaves according to their labels. After removing all of the them, again sorts the available leaves and continues this procedure until one edge remains. Since the technique deletes nodes leaves by leaves, it is called *Neville leaf encoding*. Code generated by Neville leaf encoding technique is called *Neville leaf numbers*.

Encoding Neville Leaf A tree can be encoded as a Neville leaf number by the Algorithm 9. If there exist less than 3 nodes, the algorithm is trivial. At step 5 of the algorithm, since i is a leaf, there exists exactly one j.

When the Algorithm 7 is applied in the example tree of Figure 2.1, The nodes are removed in the order of 2, 3, 5, 7, 4, 6 and 8, 4, 6, 6, 1, 1, is recorded respectively. So $NL(T) =$ 8 4 6 6 1 1.

Algorithm 8 The Construction of the Tree from Neville Branch Number

input: *a Neville leaf number $NB(T)$ with $n-2$ digits, where $1 \leq digit \leq n$.*

output: *the labeled tree T of order n.*

1: Let all node labels not part of $NB(T)$ be designated as eligible for consideration.
2: Let $node\_removed$ be the lowest eligible node label.
3: **while** There exists more than 0 digits in $NB(T)$ **do**
4: Let j be the leftmost digit of $NB(T)$.
5: Add the edge $(node\_removed, j)$ to T.
6: Designate $node_removed$ as no longer eligible and remove left most digit j from $NB(T)$.
7: **if** j does not occur anywhere in $NB(T)$ **then**
8: Let $node\_removed$ be j.
9: **else**
10: Let $node\_removed$ be the lowest eligible node label.
11: **end if**
12: **end while**
13: There are exactly 2 node labels ,k and l, which are eligible. Add the edge k and l to T.

Algorithm 9 The Construction of the Neville Leaf Number from a Tree

input: *the labeled tree T of order n.*

output: *the $n-2$ digit Neville leaf number $NL(T)$ of T, where $1 \leq digit \leq n$.*

1: **while** There exists more than 2 nodes **do**
2: Let L be the set of leaves of T.
3: **while** L is not empty and there exists more than 2 nodes in T **do**
4: Let $node\_to\_be\_removed$ be the lowest numbered leaf in L.
5: Let j be the node connected to $node\_to\_be\_removed$. Add label of j as the rightmost digit of $P(T)$.
6: Remove $node\_to\_be\_removed$ and the edge connecting $node\_to\_be\_removed$ and j from T, L and from further considerations.
7: **end while**
8: **end while**

Decoding Neville Leaf Algorithm 10 can be used to construct a tree from a Neville leaf number.

Algorithm 10 The Construction of the Tree from Neville Leaf Number

input: *a Neville leaf number* $NL(T)$ *with* $n-2$ *digits*, where $1 \leq digit \leq n$

output: *the labeled tree* T *of order* n.

1: Let all node labels be designated as unused.
2: **while** There exists more than 0 digits in $NL(T)$ **do**
3: Let all node labels not part of $NL(T)$ and unused be designated as eligible for consideration.
4: Let L be the set of eligible node labels.
5: **while** L is not empty and there exists more than 0 digits in $NL(T)$ **do**
6: Let $node\_removed$ be the lowest eligible node label in L.
7: Let j be the leftmost digit of $NL(T)$.
8: Add the edge $(node_removed, j)$ to T.
9: Designate $node\_removed$ as used.
10: Remove $node\_removed$ from L.
11: Remove left most digit j from $NL(T)$ and from further considerations.
12: **end while**
13: **end while**
14: There are exactly 2 node labels ,k and l, which are unused. Add the edge k and l to T.

2.4 PROBLEMS

Four different kinds of problems are studied:

- One-max tree problem.
- Degree constrained minimum spanning tree problem.
- All minimum spanning trees problem.
- All degree constrained minimum spanning trees problem.

2.4.1 ONE-MAX TREE PROBLEM

Suppose T is a given goal tree. Trying to reach T is defined as one-max tree problem (OMTP), by Routlauf [22]. One-max tree problem (OMTP) is defined by Rothlauf as one-min problem at [20] and one-max tree problem at [22]. Both of them tries to reach any determined tree. There are differences between defining the goal tree and fitness. Both of them uses a distance between two trees to compute fitness. The definition of distance, so fitness, changes according to the representation of tree. Both has fitness as 0 for the best solution.

In this thesis, our problem definition is similar: One-max tree problem is defined as trying to reach to a given optimum solution. The goal is given. Initially a solution or solution set is generated. Iteratively, goal is tried to be reached. There is a difference in defining the fitness with previous definitions [20] and [22]. The better a solution, the more fitness it has. In other word, one solution with higher fitness value is better. Goal has the maximum fitness.

OMTP is especially good for locality analysis. Being successful in OMTP means higher locality.

2.4.2 DEGREE CONSTRAINED MINIMUM SPANNING TREE PROBLEM

A $minimum\ spanning\ tree$ of a weighted graph is a set of edges (arcs) connecting all nodes (vertices) such that the sum of weights of the edges are smaller than or equal to any other set of edges connecting all nodes.

The *degree* of a node x is the number of edges incident to x.

With the definitions above and the definition of complete graph at given at Section 2.3.3, suppose $G = (N, E)$ is an undirected complete graph, where $N = 1, ..., n$ is the set of n nodes and $E = 1, ..., m$ is the set of edges. The $degree\ constrained\ minimum\ spanning$

$tree\ problem$ (DCMSTP) is finding the spanning tree of G with minimum total cost and all nodes with degree less than or equal to a given degree bound d.

DCMSTP is NP-hard [6]. So, as the number of nodes and edges increases, any exact solution approach becomes inefficient. So there are a variety of applications on getting better solutions.

Encodings given at section 2.3 has an advantage at computing the degrees of nodes. So, they may be successful at DCMSTP.

2.4.3 ALL SPANNING TREES PROBLEM

Given the definition of complete graph at Section 2.4.2, suppose $G = (N, E)$ is an undirected complete graph, where $N = 1, ..., n$ is the set of n nodes and $E = 1, ..., m$ is the set of edges. *All spanning trees problem*, ASTP, is finding S, such that $S = the\ set\ of\ all\ the\ spanning\ trees\ of\ G$.

If G is also a weighted graph and all spanning trees with minimum total cost is searched, then the problem is called *all minimum spanning trees problem*, AMSTP.

Also, some additional constraints often makes the problem harder. For example, a bound on the degree: Finding all the spanning trees of G with minimum total cost and all nodes with degree less than or equal to a given degree bound is called *all degree constrained minimum spanning trees problem*, ADCMSTP.

Since 1965, the spanning tree enumeration problem have been studied and many algorithms have been proposed [14]. One approach , which is used by many early algorithms, is backtracking. It is used mostly when listing the subgraphs. By backtracking approach, an algorithm with $O(|N| + |E| + |S||N|)$ time and $O(|N| + |E|)$ space outputs all spanning trees [14]. Recently, another technique have been developed by Kapoor and Ramesh [12], Matsui [14], and Shioura and Tamura [24]. These algorithms find a new spanning tree by changing one edge of current one. Instead of outputting the sequence of entire spanning tree, they output a list of spanning trees in *compact* form. They just output the relative difference of each consecutive two spanning trees. Their optimal one has $O(|N| + |E| + |S|)$ time and $O(|N| + |E|)$ space complexities [25].

CHAPTER 3

NEVILLE'S TREE REPRESENTATIONS

As mentioned at 2.3, representing tree as a list of numbers, makes it suitable for evolutionary algorithms. And also, as mentioned in section 2.3, there are several alternatives for representing a tree as a chromosome. Prüfer's encoding is introduced in Section 2.3.3. In this chapter, the remaining two representations are analyzed; namely, Neville's branch encoding and Neville's leaf encoding, (will be called Neville branch, and Neville leaf ,respectively, for simplicity).

3.1 NEVILLE'S BRANCH ENCODING

No algorithm with complexity analysis for encoding and decoding Neville branch numbers are reported. This section presents algorithms for encoding and decoding Neville branch numbers with $O(n)$ time and space complexity. Section continues with locality analysis.

3.1.1 ENCODING NEVILLE BRANCH

A tree can be encoded as a Neville branch number by the Algorithm 11 with $O(n)$ time and space complexity.

The loop on lines 1 through 5 adds the leaves of the given tree to the stack in descending order of their labels. The loop on lines 6 through 13 records the code of the tree. The node to be deleted next is selected on line 7. Node v adjacent to the selected node u, is appended to the tree code on line 8. After node u is deleted from the tree on line 9, the degree of v is checked on line 10. If v becomes a leaf, it is added to the top of stack.

It is assumed that all node degrees are computed and stored in an array when the nodes are examined in the loop on lines 1-5. This requires $O(n)$ time, since each edge has to be examined twice when adjacency list data structure is used to store a tree. Each call to degree function can therefore execute in $O(1)$ time. Since only nodes with degree 1 are removed on line 9, the degree of one other node has to be updated each time. Thus, removing a node requires $O(1)$ time. So, tree encoding requires $O(n)$ time.

Algorithm 11 Algorithm to Construct Neville Branch Number of a Tree with $O(n)$ Time and Space Complexity

uses a stack S, tree is an adjacency list T.

time and space complexity: O(n) input: E(T), the edge set of the encoded tree

output: C, code of length (n-2)

```
1:  for i from n to 1 do
2:      if degree of i is 1 then
3:          add i to S
4:      end if
5:  end for
6:  for i from 1 to n-2 do
7:      remove u from top of S
8:      C[i]←v, where v is the node adjacent to u
9:      remove u from T
10:     if degree of v is 1 then
11:         add v to S
12:     end if
13: end for
```

3.1.2 DECODING NEVILLE BRANCH

A tree can be decoded from a Neville branch number by the Algorithm 12 with $O(n)$ time and space complexity. An array $last\_used$ keeps the last appearance of a node on code. It is assigned on lines 1 through 6. This requires $O(n)$ time. On lines 7 through 11, leaves are pushed into the stack in descending order of their labels. On lines 12 through 18, edges are added by joining the top of stack with the leftmost digit of code. If it is the last appearance of leftmost digit on the code, digit $C[i]$ is pushed onto the top of stack. When all digits of C is

traced, there exits two nodes in the stack, representing the last edge. The last edge is formed by joining the two last edges in the stack. Operations in loops can be implemented in constant time. Loops requires $O(n)$ time. Thus, Algorithm 12 needs $O(n)$ time.

Two arrays of size n and one array of size $n-2$ is used. Therefore, Algorithm 12 needs $O(n)$ space.

3.1.3 LOCALITY OF NEVILLE BRANCH NUMBERS

The locality of Neville branch numbers are examined by random walks through the search space and neighborhood analysis.

3.1.3.1 RANDOM WALKS

Given a labeled tree T and its Neville branch number $Nb(T)$ with $n-2$ digits, random walk through $Nb(T)$ means, that one digit of the $Nb(T)$ is changed by a randomly chosen digit d, such that $1 \leq d \leq n$, and the change of edges in T is examined.

For evaluating the locality of Neville branch numbers, computer simulations are presented. Figure 3.1 presents the results of computer simulations for random walks on the search space of Neville branch numbers. For this purpose, a random code is generated, which is the initial individual. To gain statistically significant information independent of initial individual, 400 steps (mutations) were performed on the individual. This procedure is applied for 100 independent runs. So, 40000 steps are carried out in the search space. At the end of these steps the count of edge changes for one digit change is evaluated and percentages are computed. These experiments are done for 16 node trees and 32 node trees.

When walking through the genotypic search space, the plots in Figure 3.1 shows that, only about 20% of all one digit changes result in a change of one edge in the phenotype. Interestingly, the percentage of two edge changes per one digit change is apparently extremely less than the percentages of 1 change and 3 changes per digit.

The random walks through genotypic search space shown that the locality of the Neville branch number representation is low. Small changes in the genotype result in high changes in the phenotype. In the following, whether the locality is uniformly low everywhere in the search space, or there exist some areas of high locality is investigated.

Algorithm 12 Algorithm to Construct Tree from Neville Branch Number with $O(n)$ Time and Space Complexity

uses a stack S and a last_used array

time complexity: O(n)

space complexity: O(n)

input: C, code of length (n-2)

output: E(T), the edge set of the encoded tree

```
1: for i from 1 to n do
2:     last_used[i] ← 0
3: end for
4: for i from 1 to n-2 do
5:     last_used[C[i]] ← i
6: end for
7: for i from n to 1 do
8:     if last_used[i] == 0 then
9:         push i onto the top of S
10:    end if
11: end for
12: for i from 1 to n-2 do
13:    pop v off the top of S
14:    E(T) ← E(T) U v, C[i]
15:    if last_used[C[i]] == i then
16:        push C[i] onto the top of S
17:    end if
18: end for
19: pop v off the top of S
20: pop u off the top of S
21: E(T) ← E(T) U v,u
```

3.1.3.2 ANALYSIS OF NEIGHBORHOOD

The results of random walks showed that the locality of Neville branch encoding is low. Therefore, whether the structure of the tree affects the locality or not is further investigated.

To explore whether the locality of encoding is different for different areas of the search space, an individual t_i with specific properties is chosen and mutation is applied on it. The number of edges that got changed is recorded. This procedure is run for 1000 independent experiments. A frequency analysis is carried out on the results.

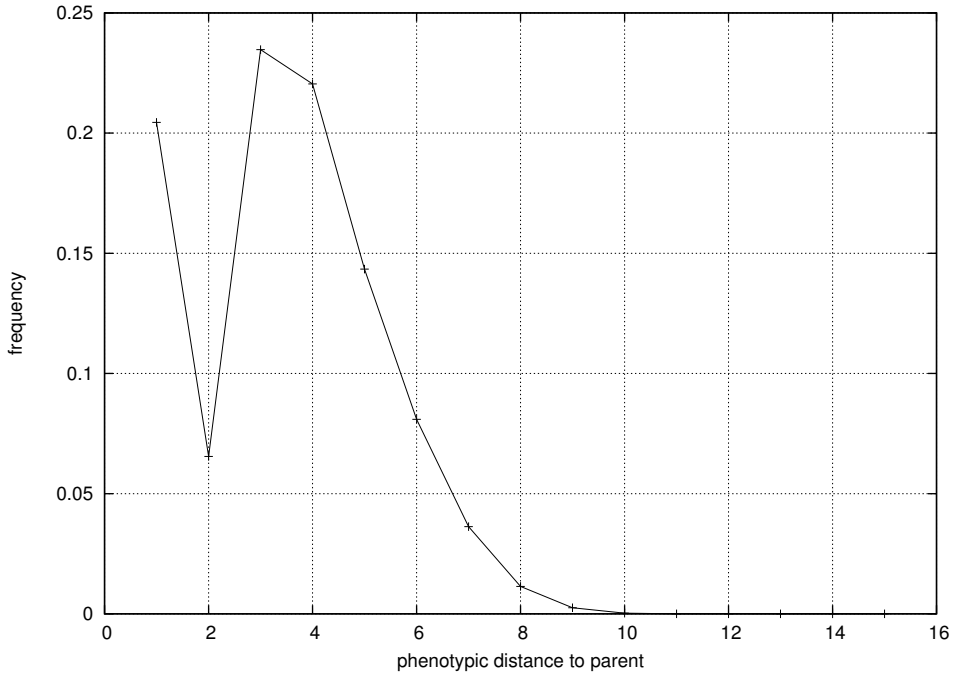
To examine the neighborhood of an individual, three different types of trees are used:

- *Star:* A tree with one node of degree $n-1$ and $n-1$ nodes of degree 1.
- *List:* A tree with two nodes of degree 1 (the first and last nodes of list) and $n-2$ nodes of degree 2.
- *Random tree:* An arbitrary tree.

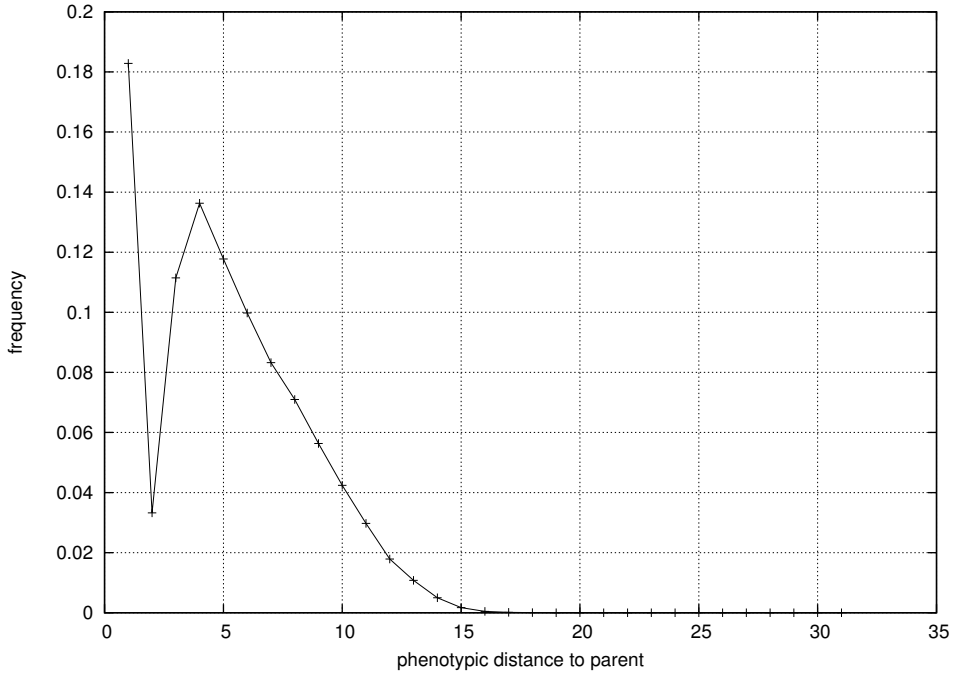
Figure 3.2 examines the neighborhood of star, list and random tree on 16 and 32 node trees. Figure shows the percentage of changed edges versus total edges when one digit is changed on the encoding.

Figure 3.2 shows that, the neighborhood of genotype depends on the structure of the encoded tree. If a star is encoded, exactly one edge changes for every digit change. This means that, locality of Neville branch number is perfect for stars. If a list is encoded, the possibility of less than 5 edge changes per one digit change is nearly zero. But the possibility of one edge change per one digit change is also low, it is about 20%. Independent of number of nodes, mostly probable number of edge changes per one digit change is 3. Nearly half of the simulations resulted in 3 different edges for list type trees. When the tree structure is random, there is no dominant number for number of changes. But for a tree of n nodes, the possibility of more than $n/2$ edge changes is near to zero. In all cases, the percentage of 2 changes is interestingly low compared to the percentages of 1 and 3.

The results show that, the locality of Neville branch number is dependent on the phenotypic structure of the encoded tree and is highly irregular. If the encoded tree is a list or a random tree, the locality is very low. Most genotypic neighbors of Neville branch number are phenotypically very different. However, if a star type tree is encoded, the locality of Neville branch number is perfect. A genotypic neighbor of a star is also a phenotypic neighbor.

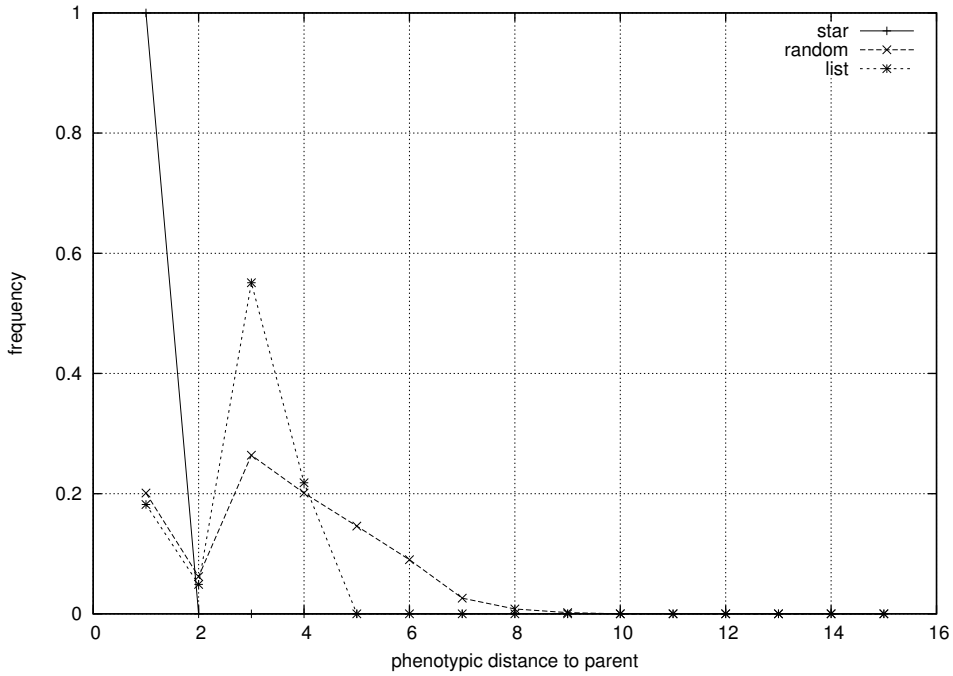


(a) 16 nodes

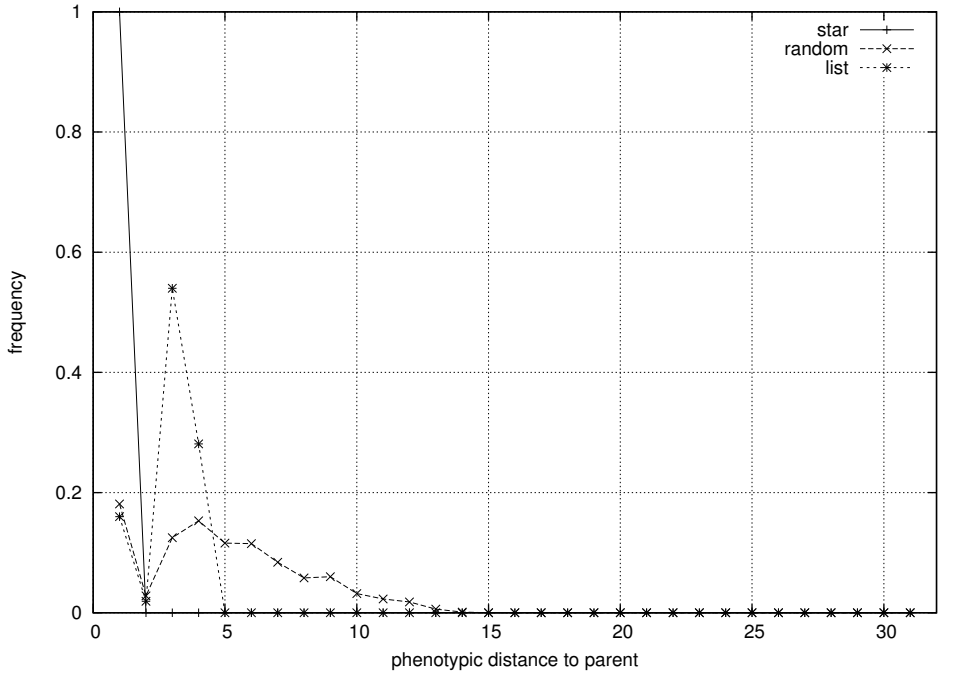


(b) 32 nodes

Figure 3.1: Distribution of phenotypic distances for neighboring Neville branch numbers on 16 and 32 node trees.
Random walks through Neville branch numbers are performed and graphs show how many links are different in the tree if one digit of the Neville branch number is changed.



(a) 16 nodes



(b) 32 nodes

Figure 3.2: Distribution of phenotypic distances for neighboring Neville branch numbers on 16 and 32 node trees for specific tree types.
Random walks through Neville branch numbers are performed and graphs show how many edges are different in a tree if one digit of Neville leaf number is changed for star, list and random type trees.

3.2 NEVILLE'S LEAF ENCODING

This section presents the locality analysis for Neville leaf encoding.

3.2.1 LOCALITY OF NEVILLE LEAF NUMBERS

The locality of Neville leaf numbers are examined in a similar way to the locality analysis of Neville branch numbers. Again, random walks through the search space of Neville leaf numbers are performed and neighbors are analyzed according to the tree structure.

3.2.1.1 RANDOM WALKS

Figure 3.3 plots the results of computer simulations for locality analysis of Neville leaf numbers.

The simulation results given at Figure 3.3 present that, only 20% of all one digit differences in genotype result in a phenotypic distance of one. The results are similar to the results of random walks through Neville branch numbers. Besides, again, there exists a strict fall down at the percentage of 2 different edges per 1 difference in Neville leaf number.

So, the random walks through genotypic search space of Neville leaf numbers revealed that the locality of the Neville leaf number representation is also low. Small changes in the genotype cause high changes in the phenotype. Neighborhood analysis of Neville branch number showed that, locality of Neville branch number changes according to the structure of the tree encoded. Following, the relation between locality of Neville leaf numbers and the structure of encoded tree is examined.

3.2.1.2 ANALYSIS OF NEIGHBORHOOD

The results of random walks indicate that the locality of Neville leaf numbers is low. Besides, neighborhood analysis of Neville branch encoding show that, locality is affected by the structure of the encoded tree. Thus, whether the structure of the tree affects the locality of Neville leaf numbers or not is investigated.

The method used for analyzing neighborhood of Neville leaf numbers is the same with the method used in Section 3.1.3. Figure 3.4 plots the percentage of the number of the changed edges are for one digit is change on the encoding. These simulations are performed for 14 and 30 digit Neville leaf numbers.

Figure 3.4 gives the results of neighborhood analysis for Neville leaf number. Locality is best for encodings of star like trees. One difference in the Neville leaf encoding means one difference in the tree structure. But if the tree structure is list, nearly 50% of the Neville leaf numbers have neigbors of phenotypic distance 3. Only 20% of list encodings have neighbors with phenotypic distance of 1. Locality for list encoding Neville leaf numbers is very low. But the possibility of getting a totally different phenotype with a mutation on genotype is zero. For random trees, a wider range for phenotypic distance is observed. But the percentages of observed distances are closer and low. Again only 20% of genomes have phenotypic distance of 1. For all tree types, percentage of 2 changes is interestingly low compared to percentages of 1 and 3 and for any tree of n nodes, the possibility of more than $n/2$ edge changes is near to zero.

3.2.2 PROPERTIES OF NEVILLE NUMBERS

The properties of Neville numbers are similar to the properties of Prüfer numbers. Similarly, benefits and disadvantages can be listed as:

- Every tree can be represented by a unique Neville number.
- Every Neville number represents exactly one tree.
- Only trees are represented by Neville numbers.
- All trees are equally represented.
- Neville numbers are closed under classical EA operations.
- Degree of a node is one more than the number of times the node appears at Neville number.

Disadvantages:

- Low locality.
- Low heritability.

Although these properties are similar, Neville branch number has an advantage. Since it does not include sorting of leaves, encoding and decoding functions are simpler and needs less time and storage.

3.3 COMPARISON OF PROPERTIES OF PRÜFER'S AND NEVILLE'S ENCODINGS

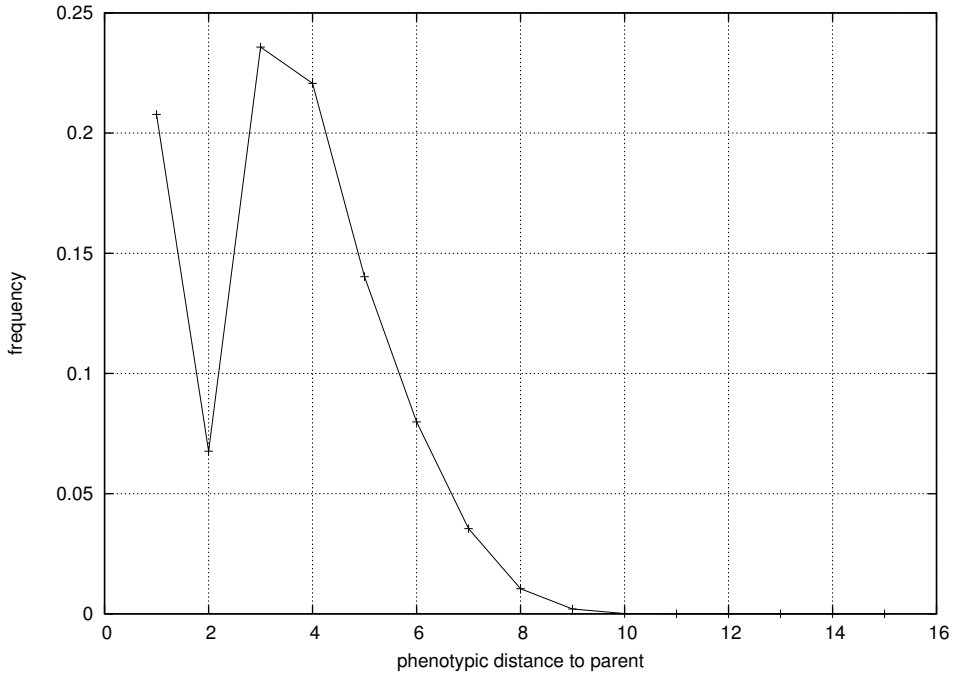
As given at Section 2.3.3, Prüfer's encoding has low locality. To make a comparison between encoding styles, similar experiments are done on Prüfer's encoding. Figure 3.5 presents the results for random walks. A starting chromosome is chosen randomly and one digit is changed. The count of changed edges is recorded. Each randomly generated chromosome is mutated 400 times. Each run is performed 100 times, independently.

Figure 3.6 presents results of random walks according to tree structure. Prüfer encoding also performs best with star type networks.

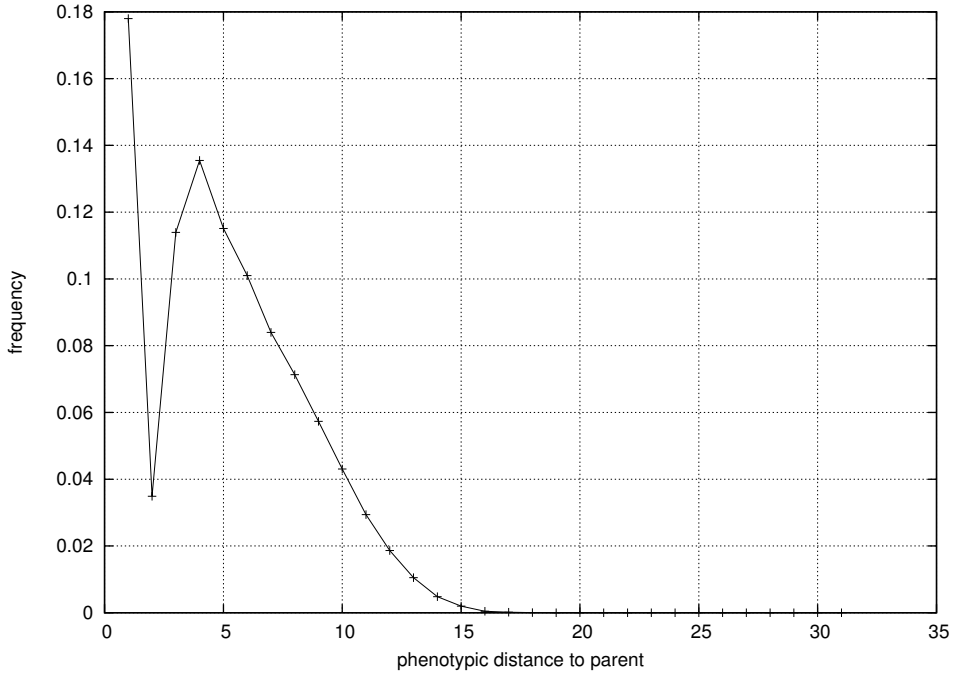
All encodings encode star network as same code. For an n node tree T, there exist n different stars and n different encodings. In a star, one node has degree $n-1$, let us call it root r and others have degree 1. If one node is connected to a node except root on phenotype, one digit in genotype will change: Let us prove this: Let S be a star type tree. Let the edge to be removed be (i, r) and edge to be inserted (i, j). If i is the smallest labeled leaf, then the first digit of encoding will change from r to j. Else, if i is not the smallest labeled leaf: There exist x leaves with smaller label than i, where $1 \leq x \leq (n-3)$ in the mutated tree. Remove x leaves. The first x digit of code will be r, which is same for the code of S. If $x < (n-3)$, it means that, degrees of r and j are greater than 1 and i is the leaf with smallest label. Remove i and $(x+1)^t h$ digit of code will be j. The remaining tree will be coded same with S. If $x = (n-3)$, then a list like tree remains, with edges (i, j) and (j, r). If $i < r$, r will be removed and last digit will be j. Else, i will be removed and last digit will be again j. Therefore, in all cases, only one digit of code will change if one edge changes for a star. This is why all encodings have perfect locality with stars. Neville branch and leaf encodings have nearly the same results for locality analysis. Prüfer numbers performs better for list type trees. For random trees, Prüfer numbers percentage of one different edge per one different digit is nearly two times Neville numbers. But, mutation on Prüfer numbers can result in much more different trees in phenotype. Neville's encodings does not produce so many different edges with one digit change. When EAs are considered, Prüfer numbers locality properties seems better. Because, Prüfer numbers generate much more different phenotypes with mutation. This makes a wider search area. Besides, when the optimum solution is near, small changes in the phenotype are not to miss the optimum solution. Neville's branch and leaf encodings do not produce very different mutants but may miss the optimum solution since percentage of

phenotypic distance of 1 is less compared to Prüfer's encoding. If Neville's encoding is used in EAs, less fit individuals should also be kept. Because one less fit individual may become fitter with one mutation.

In overall, simulations show that both Prüfer numbers and Neville numbers have low locality.

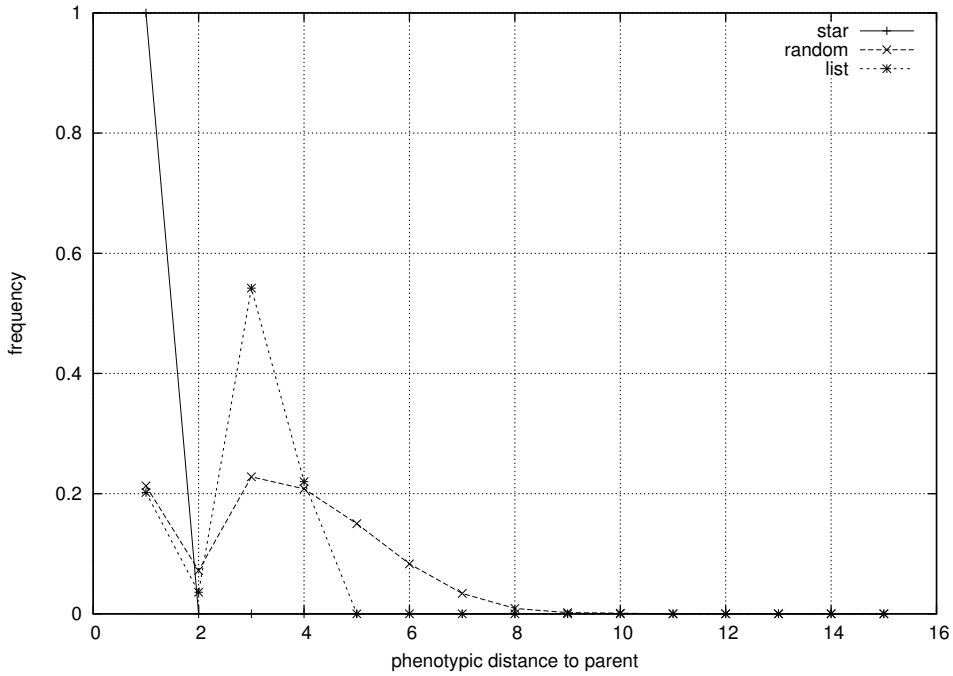


(a) 16 nodes

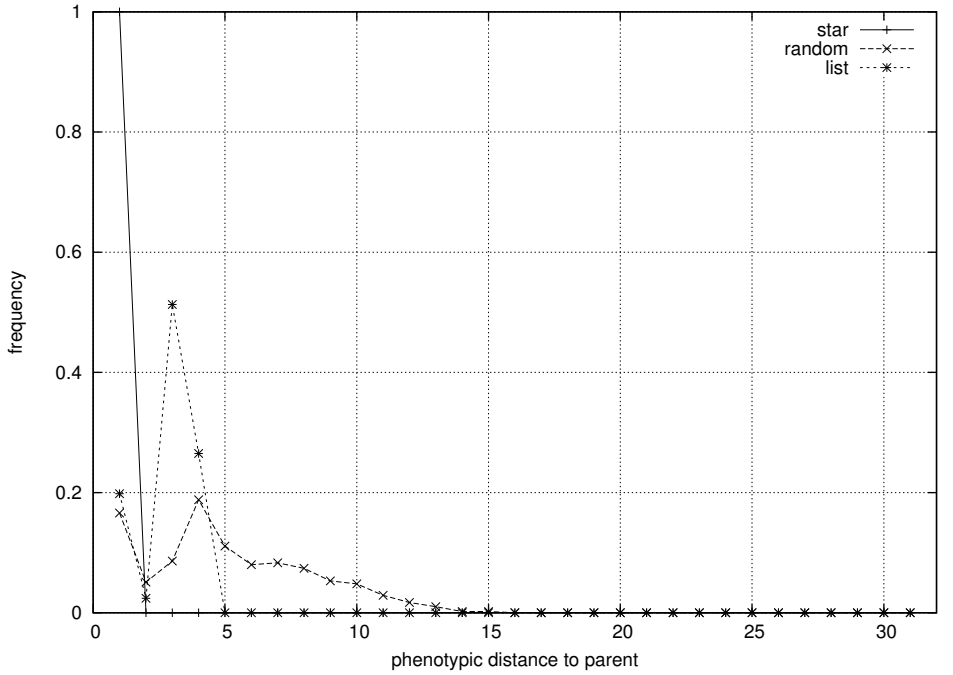


(b) 32 nodes

Figure 3.3: Distribution of phenotypic distances for neighboring Neville leaf numbers on 16 and 32 node trees.
Random walks through Neville leaf numbers are performed and graphs show how many links are different in a tree if one digit of Neville leaf number is changed.

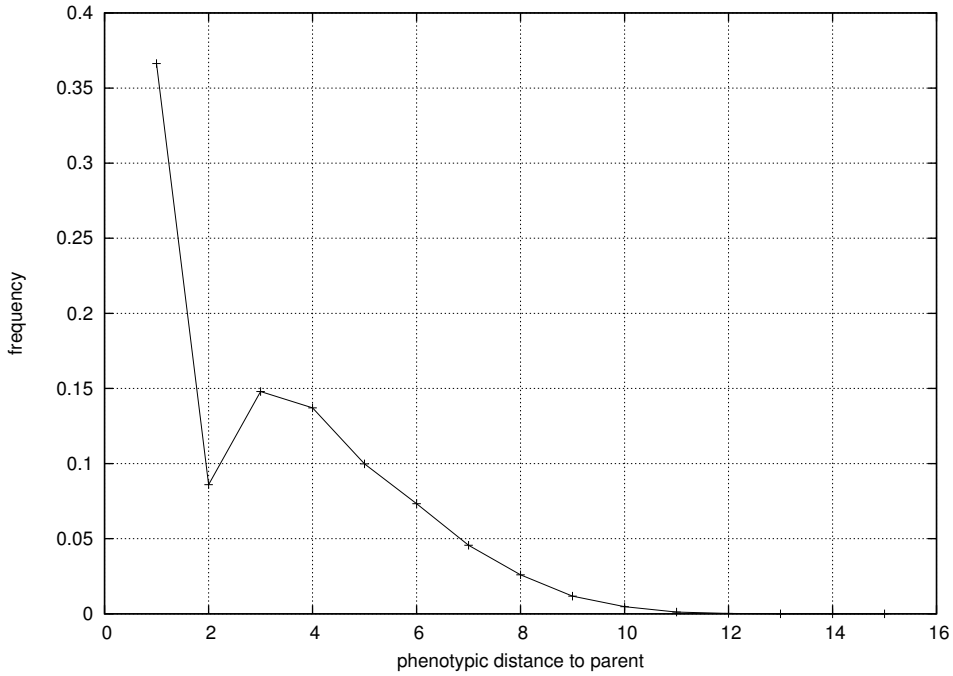


(a) 16 nodes

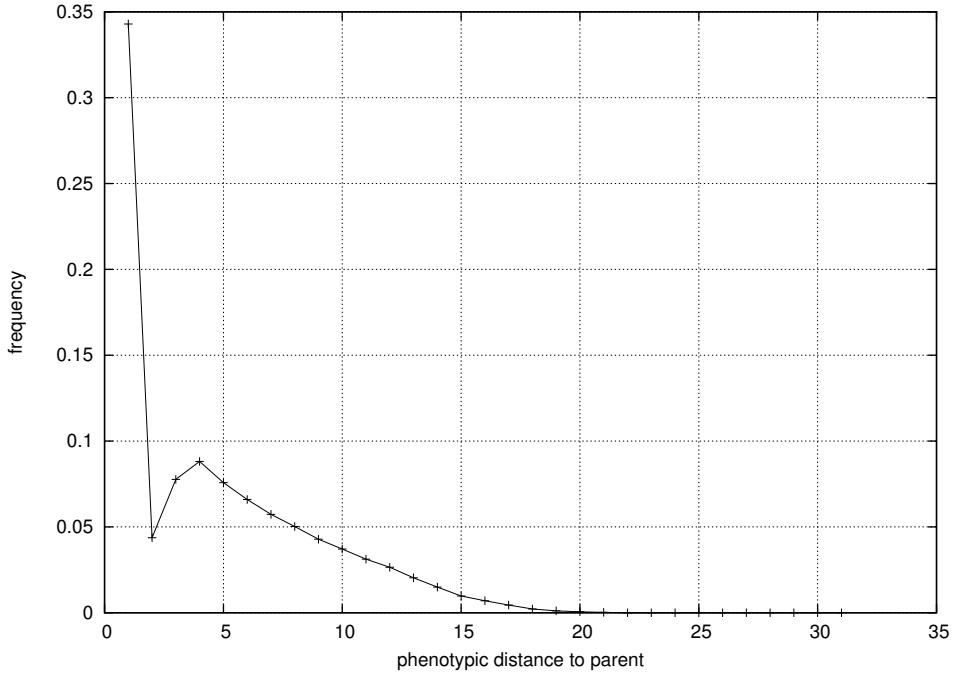


(b) 32 nodes

Figure 3.4: Distribution of phenotypic distances for neighboring Neville leaf numbers on 16 and 32 nodes for specific tree types.
Random walks through Neville leaf numbers are performed and graphs show how many edges are different in a tree if one digit of Prüfer number is changed for different tree structures.

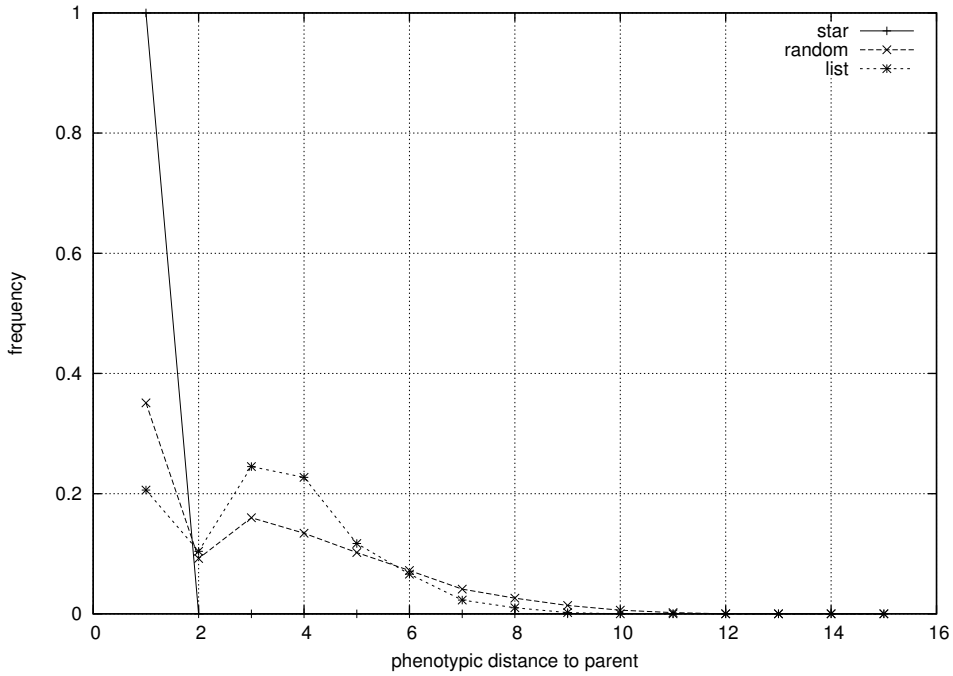


(a) 16 nodes

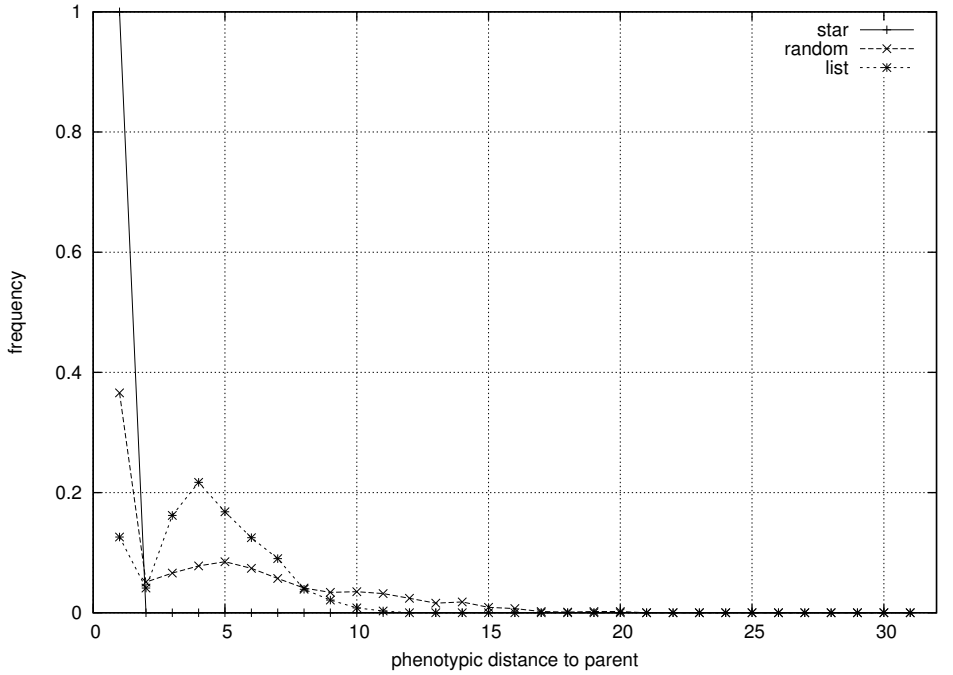


(b) 32 nodes

Figure 3.5: Distribution of phenotypic distances for neighboring Prüfer numbers on 16 and 32 nodes.
Random walks through Prüfer numbers are performed and graphs show how many edges are different in tree if one digit of Prüfer number is changed.



(a) 16 nodes



(b) 32 nodes

Figure 3.6: Distribution of phenotypic distances for neighboring Prüfer numbers on 16 and 32 node trees for specific tree types.
Random walks through Prüfer numbers are performed and graphs show how many edges are different in tree if one digit of Prüfer number is changed for star, list and random type trees.

CHAPTER 4

APPLICATIONS

In this chapter, performances of Prüfer's and Neville's encodings on EAs and SAs are investigated empirically with the following four problems:

- OMTP
- DCSTP
- ASTP
- ADCSTP

All EAs used are based on Goldberg's simple genetic algorithm [7], given at Section 2.1.

Experiments on OMTP show that, locality affects the performance of encodings on EA and SA. In addition, heritability is also low if locality is low. Thus, Neville's and Prüfer's encodings are very successful if the tree to be encoded is star type. Although degree computation is very easy with these encodings, since locality of the encodings is low for random trees, these encodings are not successful in simple EA operations. However it is observed that, Neville branch number representation gives the best solution for both ASTP and ADCSTP.

4.1 ONE-MAX TREE PROBLEM

Experiments performed in OMTP show that, neither Prüfer nor Neville encodings perform well for the search spaces with low locality.

OMTP is suitable for locality and heritability analysis. As mentioned previously, if locality and heritability is high, this means that encoding is suitable for evolutionary algorithms.

4.1.1 METHODOLOGY

For OMTP, 2 different algorithms are tested, namely EA and SA. In order to compare the success of the encoding, a greedy solution is used.

At the initial state, for a given number of nodes n, goal is generated randomly. Since any permutation can be decoded as a tree with the encodings in Section 2.3, a permutation of length $n-2$ is randomly generated. This is used as the goal chromosome. Locality analysis in Chapter 3 showed that, encodings have different locality properties for different tree structures. Because of this, experiments are done for 3 different tree types: star, list and random.

For the greedy solution, *distance* represents the number of differences in permutations. For the encodings, *distance* defined as the number of edges different in the decoded tree.

4.1.2 EXPERIMENTS

4.1.2.1 EA

In this subsection, results for EA on OMTP are presented. For each test, 100 independent runs were performed and each run is terminated after the population is fully converged. In all simulations one-point crossover with crossover probability 1 and mutation probability 0.5 is used. Each of the figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 include 4 sub-figures. The only difference between the algorithms used in sub-figures are the fitness functions used. Each sub-figure shows the result of a different fitness function. The first sub-figure, called genotypic distance comparison, is the results of an EA which uses genotypic distance as fitness function. For an n node OMTP, an individual is a $n-2$ digit number. In order to compute the fitness value of an individual, the goal chromosome is compared with the individual chromosome. The fitness value is the number of same digits in two encodings. So, the maximum of fitness value of genotypic distance comparison for a n node OMTP is $n-2$. When $n-2$ is reached as the fitness value, the problem is solved. Other three sub-figures use Neville's and Prüfer's encodings for fitness function. The code is decoded and phenotypic distance is used to evaluate fitness. The number of the same edges with the goal tree is assigned as the fitness value. An individual is a $n-2$ digit number, for an n node OMTP. Fitness value of an individual is the number of same edges between goal chromosome and the individual.

All the sub-figures include 3 plots, *MIN, MAX, AVARAGE*. MAX plots the maximum

fitness value available for the current generation. Similarly, MIN plots the minimum fitness value found in the given generation, and finally AVARAGE plots the average fitness value of the generation. Each plot is the average of 100 independent runs.

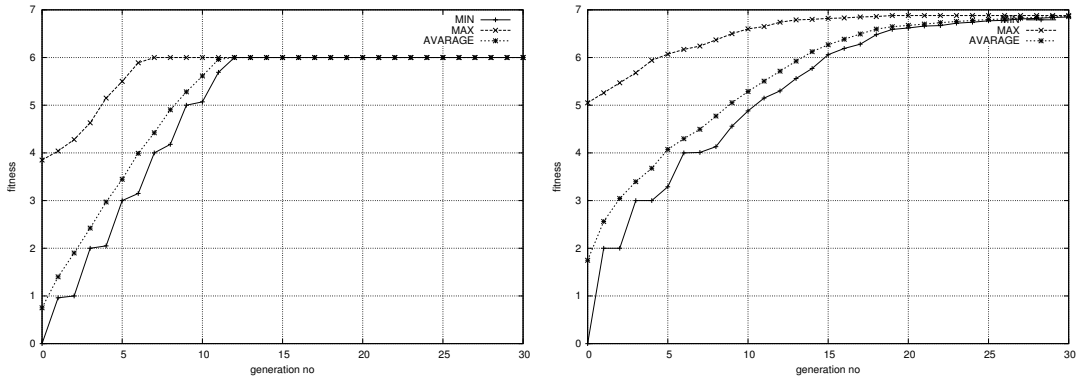
Figure 4.1 shows the performances of Prüfer's and Neville's encodings with EA for 8 node OMTP with random trees. $(\lambda + \mu)$ selection is used with $\lambda = \mu = 500$. Parents are chosen for crossover by roulette wheel selection. When the value 6 is reached as the fitness value for genotypic distance comparison, it means that, the goal is reached. Similarly, 7 is the goal fitness value if Prüfer's and Neville's encodings are used. The plots show minimum, maximum and average fitness values for each $generation$. Experiments show that, population size is important in finding the optimum solution. For example, optimum solution cannot be reached if the population size is 100. Figure 4.1 show that, Prüfer numbers and Neville numbers can solve only small size OMTPs. The locality and heritability properties of Prüfer numbers are also similar to the locality and heritability properties of Neville branch numbers

Figure 4.2 shows fitness value of individual for each $step\_count$ for 16 node OMTP with random trees. $(\lambda + \mu)$ selection is used with $\lambda = \mu = 500$. Roulette wheel selection is used for mating parents. When the problem size get larger, EAs suffer from sticking on a local maximum. Figure 4.2 shows that, none of the encodings can solve a 16 node OMTP for random trees. Prüfer numbers perform better than Neville numbers.

Figure 4.3 shows the effect of selecting mates for crossover. In a roulette wheel selection, fitter parents are chosen more often than the less fit ones. In addition to roulette wheel selection, random selection is also tested. It is seen that, when random selection is used, encodings converge faster.

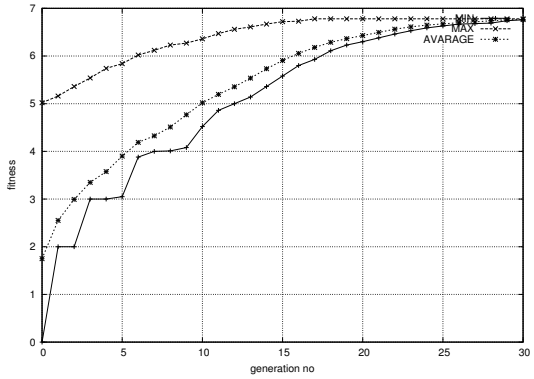
On the simulations of 4.4, a limit on the duplicate members selected for the next generation is applied. Let i be an individual chosen for the next generation. Let j be a candidate to be chosen for the next generation, such that it has the highest fitness among the remaining candidates. If i and j are the same chromosomes, then j is omitted and a new individual is searched for selection. So, an individual can fill at most half of the population. From this technique, no improvement is observed.

In figures 4.5 and 4.6, simulations are the same except the input types. $(\lambda+\mu)$ selection is used with $\lambda = \mu = 1000$ in both of them. Roulette wheel selection is used to select parents. Genotypic distance comparison finds the solution when fitness is 30. Neville and Prüfer numbers find the solution when fitness is 31. However, Figure 4.5 uses random type trees

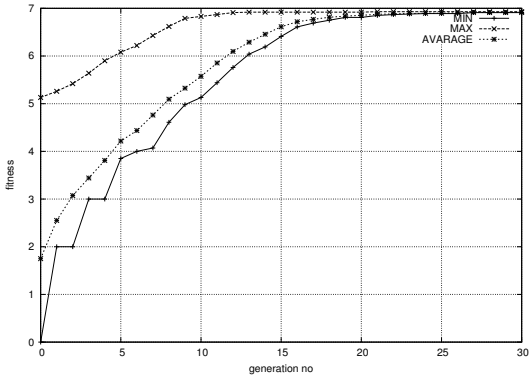


(a) Genotypic distance comparison

(b) Neville branch numbers



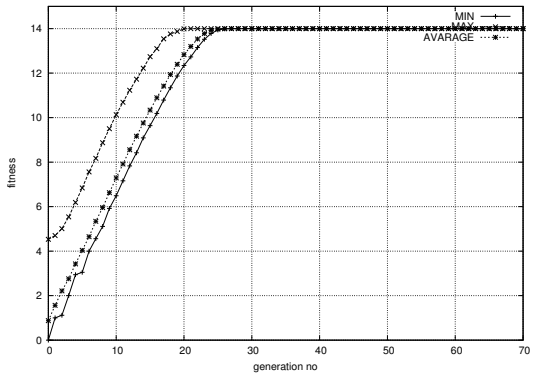
(c) Neville leaf numbers



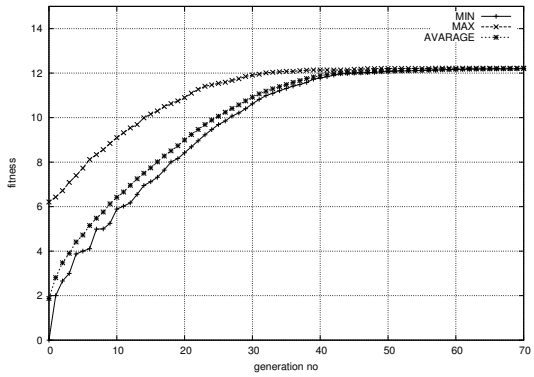
(d) Prüfer numbers

Figure 4.1: Performances of Prüfer's and Neville's encodings with EA for 8 node OMTP with random trees.

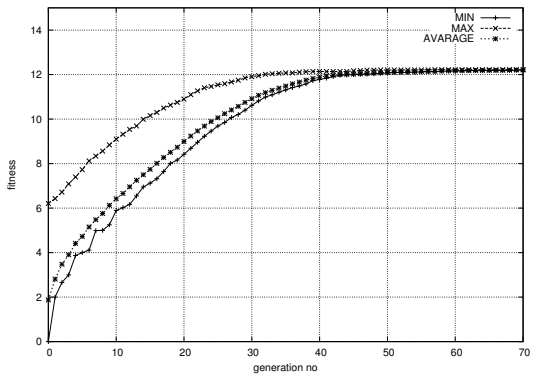
and Figure 4.6 uses star type trees. The comparison shows that, tree structure determines the success. If tree is star type, solution is found as efficiently as in the genotypic fitness approach, because Neville and Prüfer numbers have perfect locality with star type trees.



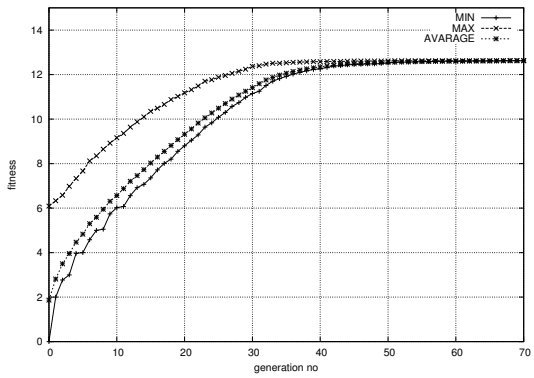
(a) Genotypic distance comparison



(b) Neville branch numbers

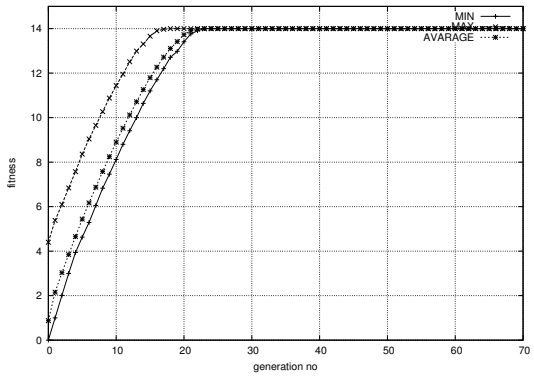


(c) Neville leaf numbers

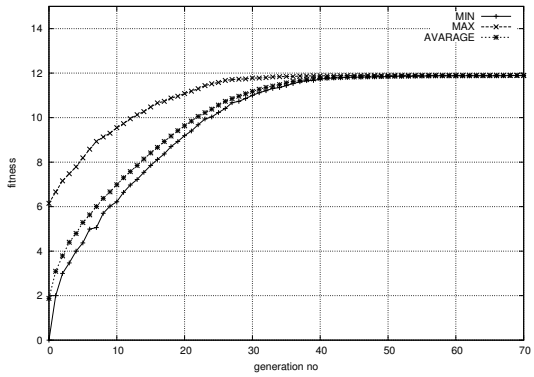


(d) Prüfer numbers

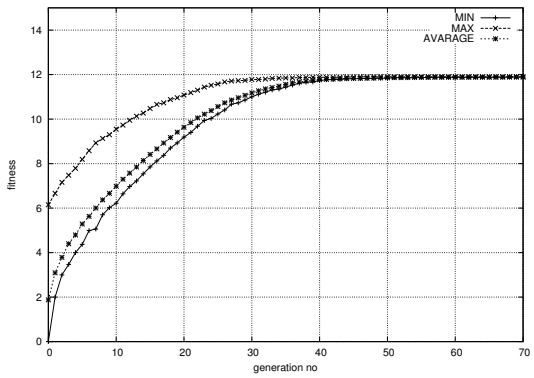
Figure 4.2: Performances of Prüfer's and Neville's encodings with EA for 16 node OMTP with random trees.



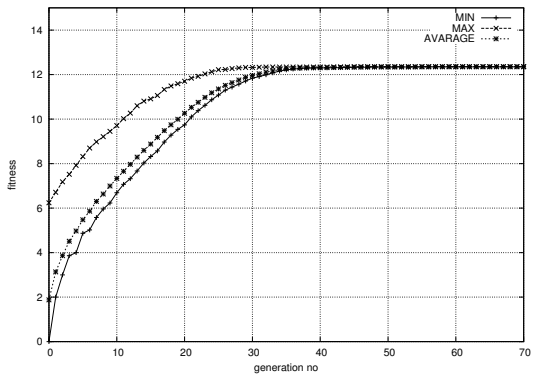
(a) Genotypic distance comparison



(b) Neville branch numbers

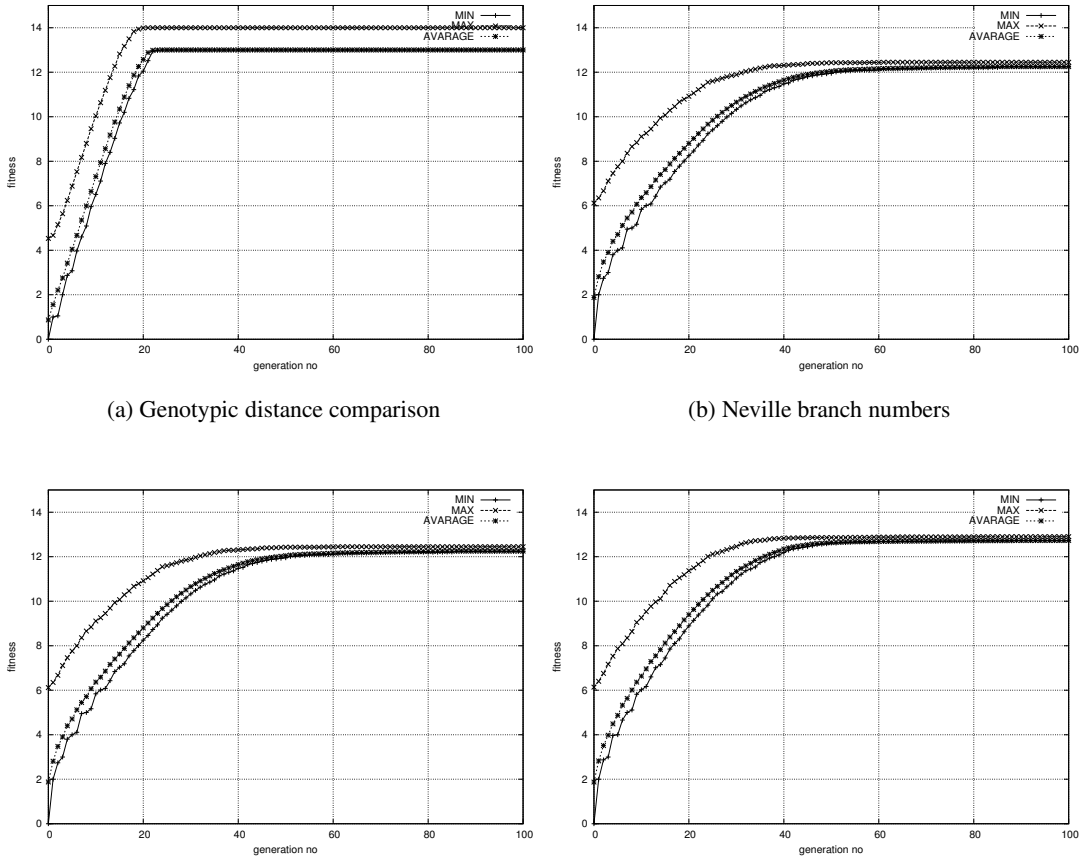


(c) Neville leaf numbers



(d) Prüfer numbers

Figure 4.3: Performances of Prüfer's and Neville's encodings with EA for 16 node OMTP with random trees with random selection for crossover.



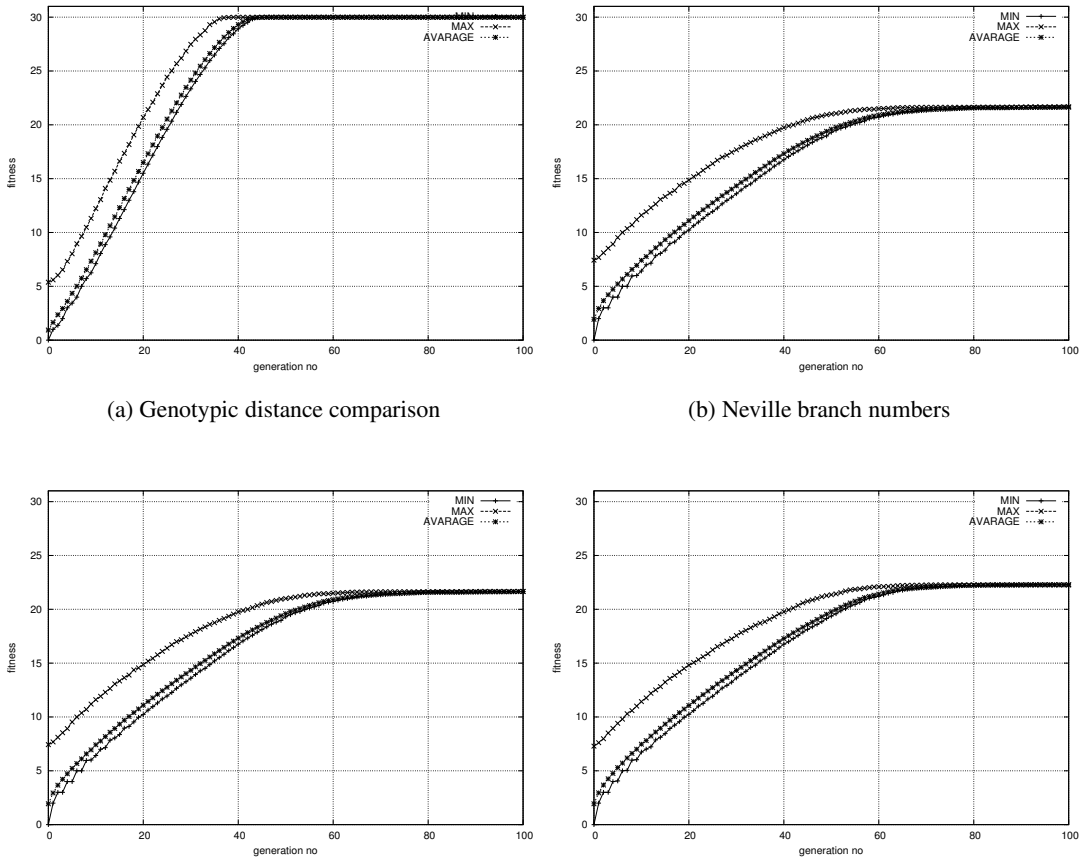
(a) Genotypic distance comparison

(b) Neville branch numbers

(c) Neville leaf numbers

(d) Prüfer numbers

Figure 4.4: Performances of Prüfer's and Neville's encodings with EA for 16 node OMTP with random trees with limit on duplication.



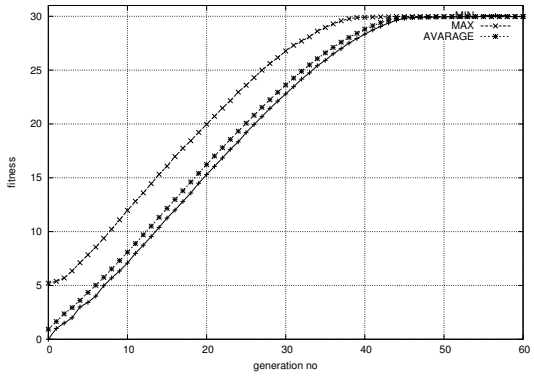
(a) Genotypic distance comparison

(b) Neville branch numbers

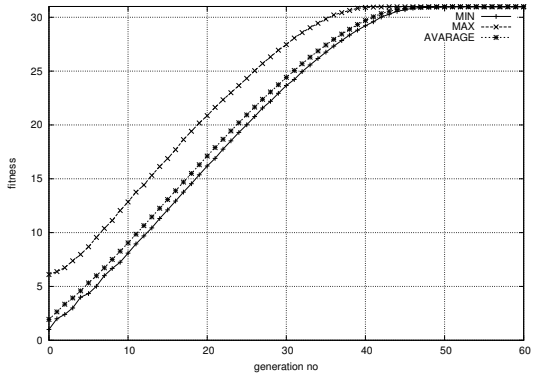
(c) Neville leaf numbers

(d) Prüfer numbers

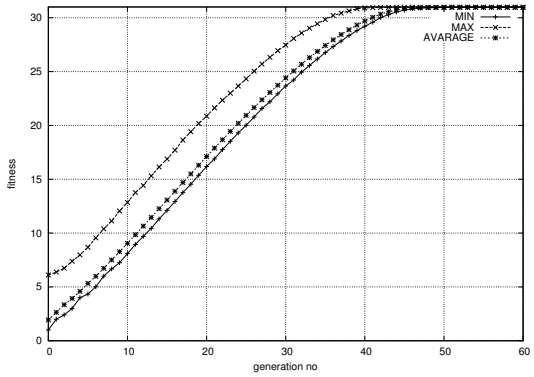
Figure 4.5: Performances of Prüfer's and Neville's encodings with EA for 32 node OMTP with random trees.



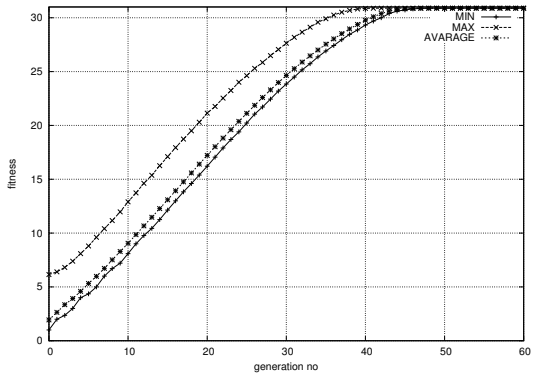
(a) Genotypic distance comparison



(b) Neville branch numbers



(c) Neville leaf numbers



(d) Prüfer numbers

Figure 4.6: Performances of Prüfer's and Neville's encodings with EA for 32 node OMTP with star type trees.

4.1.2.2 SA

SA given in Section 2.2 is used for the SA tests. Initially a random solution is generated, which is called as *current*. Then, mutation is applied to current and *next* is obtained. If next is better that current, next is assigned to the current. Otherwise, next is assigned to the current with a probability depending on temperature T as follows:

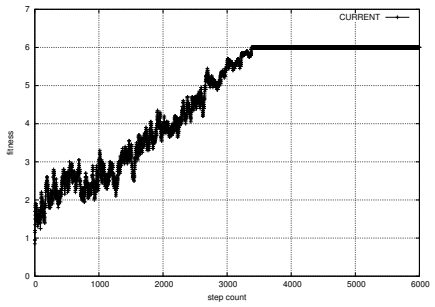
$$P(T) = e^{\frac{next\_fitness - current\_fitness}{T}}$$

where T is initially maximum and decreases with step count. T is computed as:

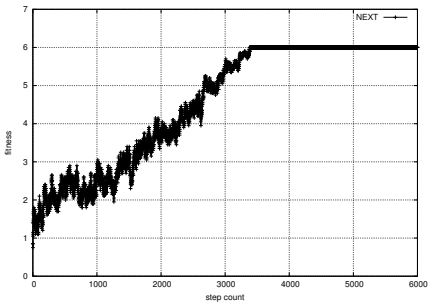
$$T = T_0 - step\frac{T_0 - T_{final\_step\_count}}{final\_step\_count} where T_0 = 1 and T_{final\_step\_count} = 0$$

Each figure consists of 8 sub-figures. Genotypic fitness approach obtains the best solution when fitness $n-2$ is reached, where n is the number of nodes. Other three approaches, which use Neville's and Prüfer's encodings, need to obtain $n-1$ as fitness to reach the goal. For each fitness approach, plots show the fitness values of the current individual and the potential current individual, ie. next individual, versus step. For each problem 20 independent runs were performed.

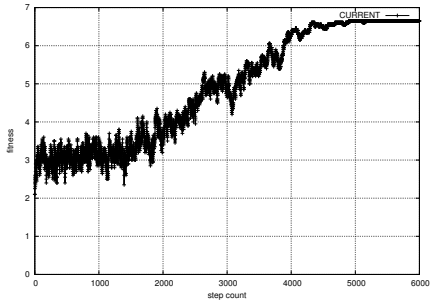
In the experiments of Figure 4.7, 6000 steps were run. When the number of nodes increases to 32 in Figure 4.8, steps are increased to 12000. Figures 4.7 and 4.8 show that, if the optimal solution is star, SA can find it. However, if the optimal tree is not a star, even if it is a small sized tree, it is not guaranteed that the optimal tree will be reached.



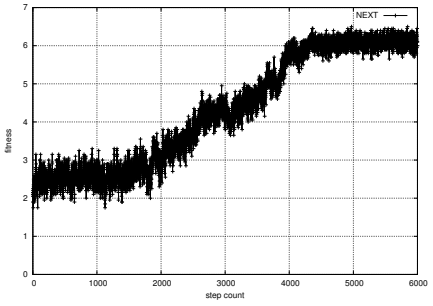
(a) Genotypic distance comparison - current



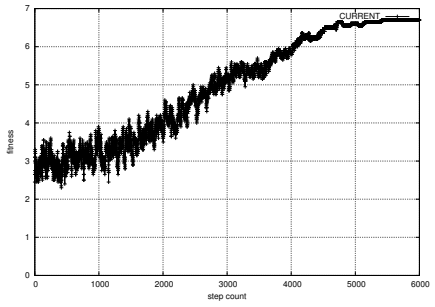
(b) Genotypic distance comparison - next



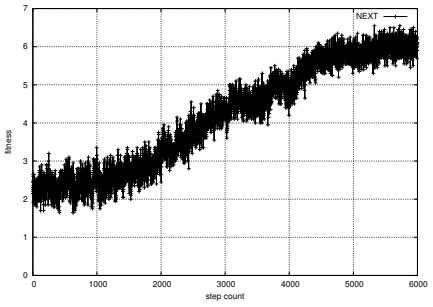
(c) Neville branch numbers - current



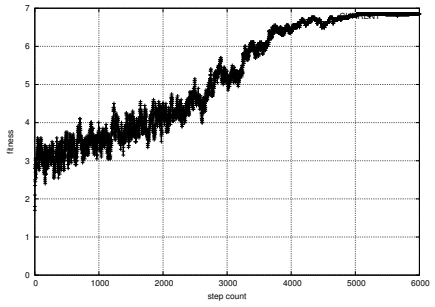
(d) Neville branch numbers - next



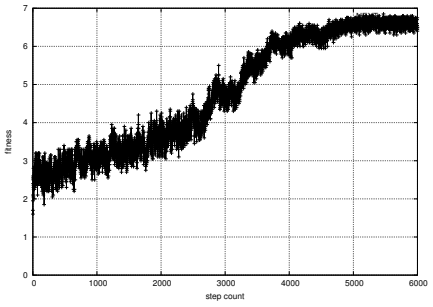
(e) Neville leaf numbers - current



(f) Neville leaf numbers - next

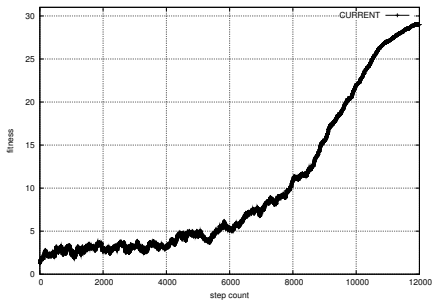


(g) Prüfer numbers - current

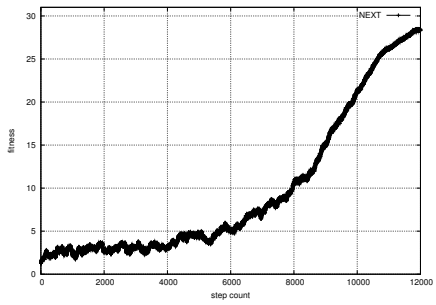


(h) Prüfer numbers - next

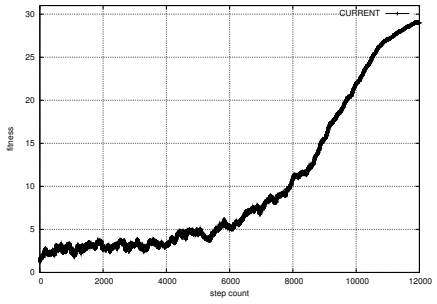
Figure 4.7: Performances of Prüfer's and Neville's encodings with SA for 8 node OMTP with random trees.



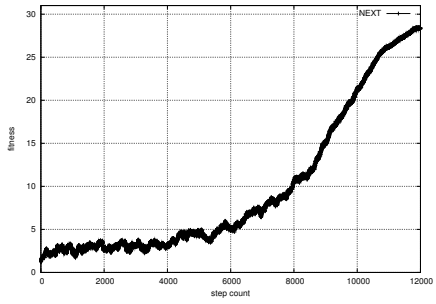
(a) Genotypic distance comparison - current



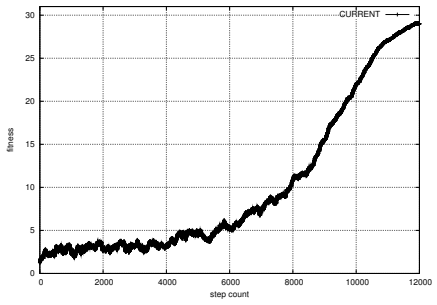
(b) Genotypic distance comparison - next



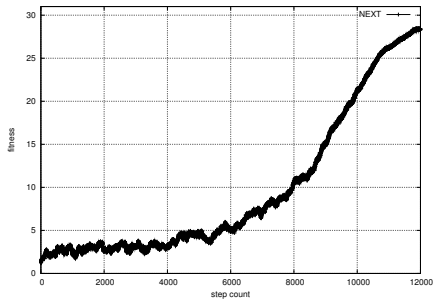
(c) Neville branch numbers - current



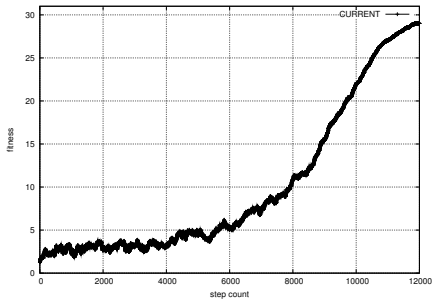
(d) Neville branch numbers - next



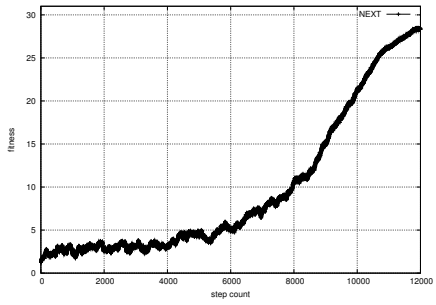
(e) Neville leaf numbers - current



(f) Neville leaf numbers - next



(g) Prüfer numbers - current



(h) Prüfer numbers - next

Figure 4.8: Performances of Prüfer's and Neville's encodings with SA for 32 node OMTP with star type trees.

4.2 DEGREE CONSTRAINED MINIMUM SPANNING TREE PROBLEM

Since degree of a node of a tree, represented by prüfer-like codes is one more than the number of times the node appears in the code, and there is a degree constraint at the problem, prüfer-like codes might perform well.

4.2.1 METHODOLOGY

Two different algorithms are used for degree constrained minimum spanning tree problem:

- Evolutionary algorithm.
- Simulated annealing.

Problem is tested by a 8 node random tree with degree bound 4. Data set is given in the Appendices A.

Each experiment is performed for each of Neville's and Prüfer's encoding styles. For a graph G, *cost* is the sum of weights of edges used. Mutation and crossover operators are applied on genomes and fitness is calculated according to the phenotypes: Let dp_i be 0 if $degreeofnodei \leq degreebound$ and $degreeofnodei - degreebound$ otherwise. Also, let dp_{total} be the sum of d_is for all nodes. Then *degree_penalty* is defined as

$$dp_{total} * (n-2) * DEGREE\_PENALTY\_CONSTANT$$

where n is the number of nodes DEGREE_PENALTY_CONSTANT is determined according to the size of the problem. For an 8 node DCMSTP, it is assigned as 100. Then, the fitness function is defined as:

$$totalcostofgraph - totalcostoftree + (MAX\_DEGREE\_PENALTY - degree\_penalty)$$

where MAX_DEGREE_PENALTY is assigned as 5000.

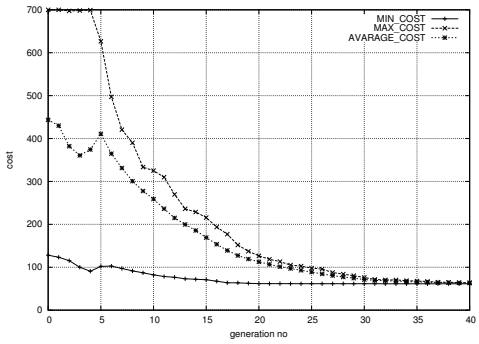
4.2.2 EXPERIMENTS

4.2.2.1 EA

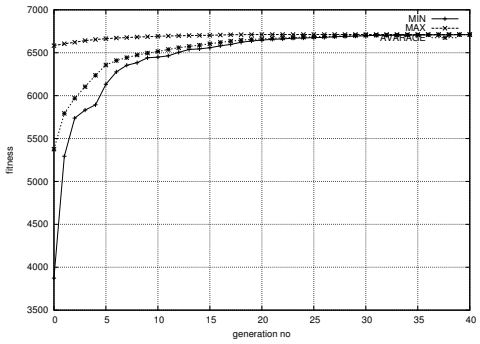
In EA, initial population is randomly generated. A selection is done according to the fitness value, defined at Section 4.2.1. Individuals with nodes having degrees more than degree bound are possible but since degree penalty is applied, the possibility is small. The plots

in Figure 4.9 show minimum, maximum and average fitness values for each $generation$. Neville and Prüfer numbers find the solution when fitness is 7. $(\lambda + \mu)$ selection is used with $\lambda = \mu = 500$. Roulette wheel selection is used for crossover. Crossover probability is 1. Mutation probability is 0.5. 100 independent runs were performed and each run was stopped after the population was fully converged. Optimum cost for given sample is 43.

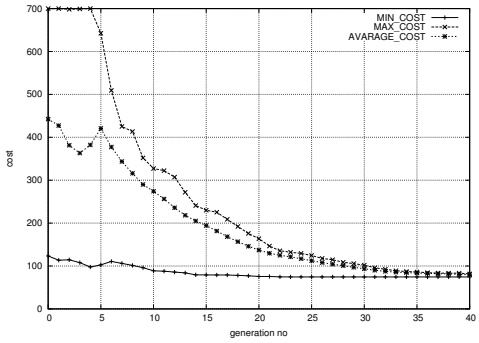
Experiments showed that all encodings can hardly solve even simple DCMSTP problems. When the problem gets bigger, they become far away from finding the optimal solution.



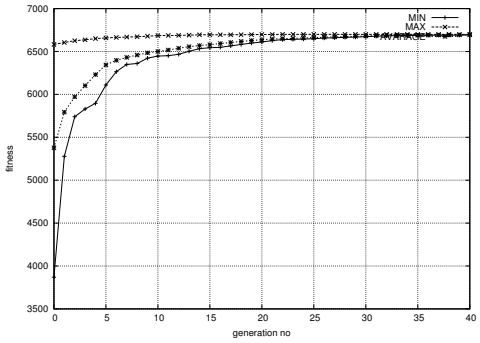
(a) Neville branch numbers - cost



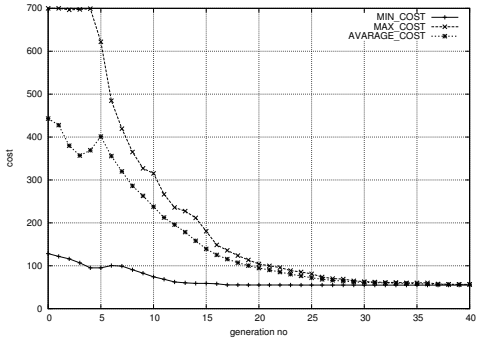
(b) Neville branch numbers - fitness



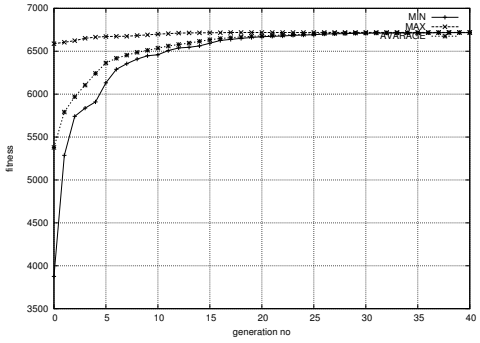
(c) Neville leaf numbers - cost



(d) Neville leaf numbers - fitness



(e) Prüfer numbers - cost

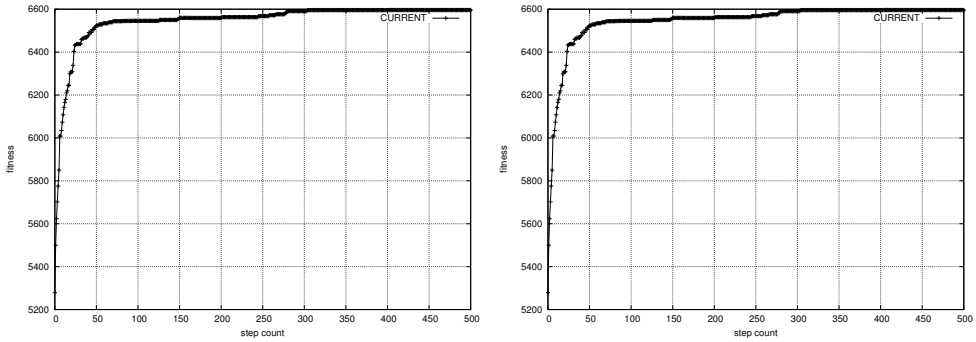


(f) Prüfer numbers - fitness

Figure 4.9: Performances of Prüfer's and Neville's encodings with EA for 8 node DC-MST.

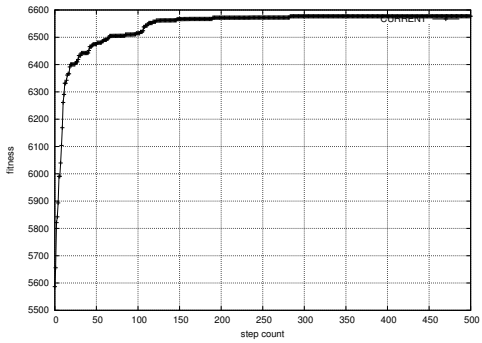
4.2.2.2 SA

Figure 4.11 plots fitness values of the current individual and the next individual for each $step$. and Figure 4.10 plots the cost values of the current individual and the next individual for each $step$, for the same experiment. The same sample input with EA on DCMSTP is used. Neville and Prüfer numbers find the solution when fitness is 7. 6000 steps are run. For each problem 20 independent runs were performed. Figures 4.11 and 4.10 show that, node of the encodings can solve the problem.

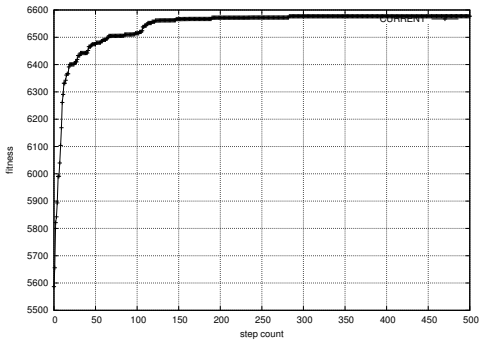


(a) Neville branch numbers - current

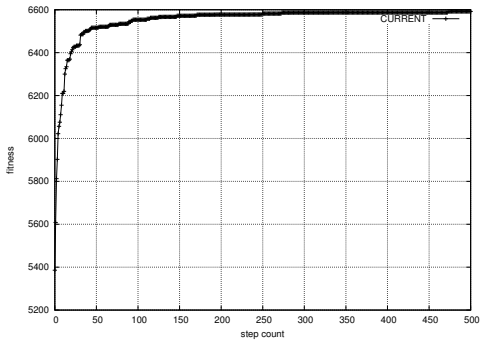
(b) Neville branch numbers - next



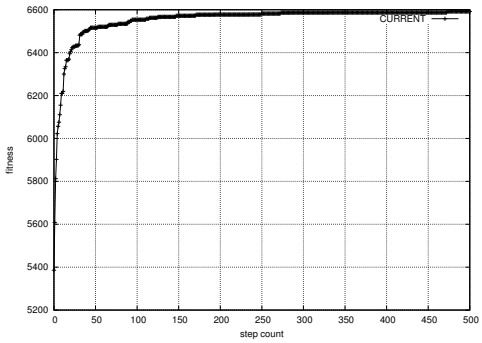
(c) Neville leaf numbers - current



(d) Neville leaf numbers - next

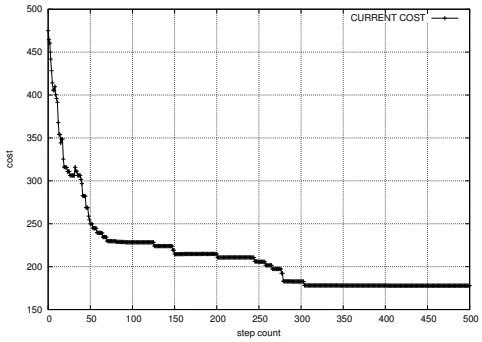


(e) Prüfer numbers - current

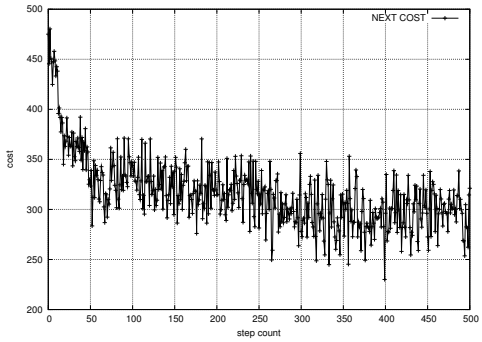


(f) Prüfer numbers - next

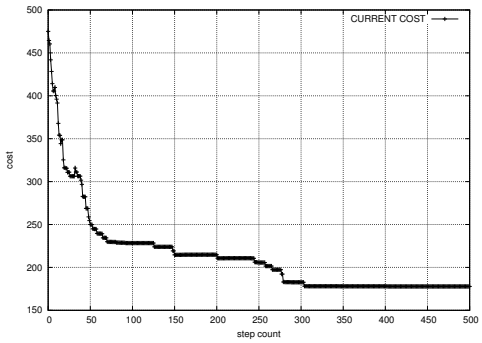
Figure 4.10: Performances of Prüfer's and Neville's encodings with SA for 8 node DC-MST (fitness).



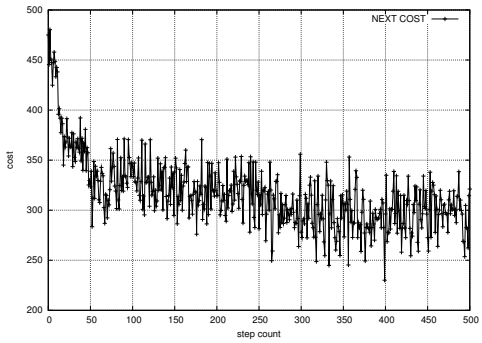
(a) Neville branch numbers - current



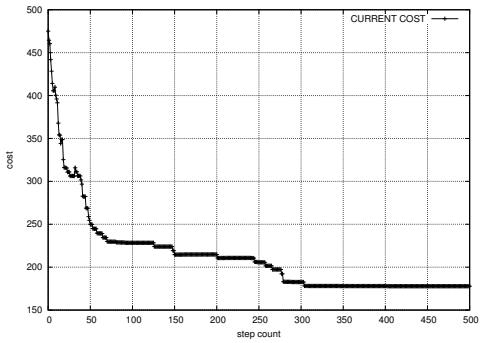
(b) Neville branch numbers - next



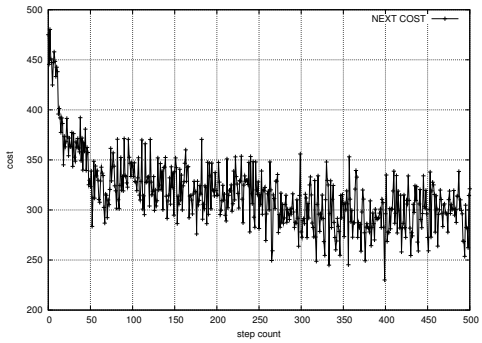
(c) Neville leaf numbers - current



(d) Neville leaf numbers - next



(e) Prüfer numbers - current



(f) Prüfer numbers - next

Figure 4.11: Performances of Prüfer's and Neville's encodings with SA for 8 node DC-MST (cost).

4.3 ALL SPANNING TREES PROBLEM

In this section, optimum algorithms for all spanning trees problem (ASTP) of a (weighted or unweighted) complete graph is given. Note that, any weighted graph can be written as a weighted complete graph. If there is not an edge between two nodes, it is enough to add an edge with infinite weight, in order to obtain a weighted complete graph from a weighted graph.

4.3.1 METHODOLOGY

There are 2 different applications of ASTP:

- Do not output entire spanning tree, just enumerate trees.
- Output entire trees.

In the first application, entire trees are not printed. They stay printable when necessary. For example, suppose that a spanning tree with specific properties is searched. An encoding satisfies the constraints. Then the entire tree is printed. The second application outputs entire trees. This increases the time and space complexities.

4.3.2 ENUMERATE ALL SPANNING TREES

In 1889, Cayley proved that the number of spanning trees of an undirected complete graph G with n vertices is n^{n-2} , (see Section 2.3.3). Prüfer's and Neville's encodings span the tree space of G completely. Each encoding is a one-to-one and onto mapping between labeled tress and encodings. So, to enumerate all spanning trees, we only need to generate all codes one by one.

The algorithm 13 will generate n^{n-2} spanning trees. Time complexity is $O(n^{n-2})$. If S is the set of all spanning trees of G and $|S| = s$, time complexity can be written as s. A code is a number with $n-2$ digits. Thus, space complexity is $O(n)$. To the best of our knowledge, best known algorithm has $O(s+n+e)$ time and $O(n+e)$ [25] For a complete graph, since $s = n^{n-2}$, complexities become $O(n^{n-2})$ and $O(n+e)$, respectively. Both algorithms have the same time complexity but our algorithm is better at space complexity.

Any code is decodable by any of the algorithms 8, 10, 6. Due to the simplicity and low complexities, Neville's branch encoding is suggested.

Algorithm 13 Enumerate All Spanning Trees

input: A complete graph G with n nodes.

output: The set of all encodings which represents the set of all spanning trees of G.

1: len=$n-2$
2: code=empty
3: **if** len = 0 **then**
4: print code
5: **end if**
6: **for** $digit$=1 to n **do**
7: Add $digit$ as the leftmost digit of code.
8: len = len - 1
9: goto 3
10: **end for**

The Algorithm 13 can also be applied to incomplete graphs with a validity check. Since Neville's and Prüfer's encodings are mappings between codes and spanning trees of complete graphs, codes with unavailable edges will be generated. At the 4th step, validity can be checked. If code includes any edge not available, code is omitted. Checking if any unavailable edge exists in the code can be done by decoding the code. So, if Neville branch technique is used, time complexity of validity check will be $O(n)$. This will result in a time complexity of $O(n^{n-1})$. Also, the set of edges E is needed to check the existence of edges. This results in a space complexity of $O(n+e)$, where $e = |E|$. Thus, enumerating all spanning trees of an incomplete graph can be done in $O(n^{n-1})$ time and $O(n+e)$ space.

4.3.3 ENUMERATE AND OUTPUT ALL SPANNING TREES

The above algorithm 13 generates but does not print the spanning trees. If instead of printing the code, it is decoded, all spanning trees can be printed.

Complexity analysis of Algorithm 14 depends on Step 4, decoding operation. Neville branch decoding is suggested. Decoding of Neville branch number has time complexity $O(n)$ (see Algorithm 12) and space complexity $O(n)$. So, Algorithm 14 will output all spanning trees with a time complexity of $O(n^{n-1})$ and space complexity of $O(n)$.

The optimal algorithm which enumerates all spanning trees by outputting all edges of each spanning tree was reported to have $O(sn+n+e)$ time and $O(n+e)$ space complexities

Algorithm 14 Output All Spanning Trees

input: A complete graph G with n nodes.

output: The set of all spanning trees of G.

```
1: len=n − 2
2: code=empty
3: if len = 0 then
4:     Decode code
5: end if
6: for digit=1 to n do
7:     Add digit as the leftmost digit of code.
8:     len = len-1
9:     goto 3
10: end for
```

[5]. For a complete graph, the complexities becomes: $O(n^{n-1} + n + e)$. Thus our algorithm is better.

Similarly, Algorithm 14 can be applied to incomplete weighted or unweighted graphs with a validity check.

4.4 ALL DEGREE CONSTRAINED SPANNING TREES PROBLEM

Adopting the algorithms 13 and 14 for checking the degree constraint is quite easy. We only need to count the occurrences of digits. As presented at Sections 3.2.2, and 2.3.3.3 degree constrained is evaluated in $O(n)$. Moreover, degree constraints can be controlled before generating the code.

Degree constraint adds no complexity to ASTP. Similarly, if trees are given as output, time complexity is $O(n^{n-1})$ and space complexity is $O(n)$. If only codes are given as output, time complexity $O(n^{n-2})$ is and space complexity is $O(n)$.

4.4.1 DISCUSSIONS

The experiments on OMTP show that, the structure of the best solution has a large influence on the performance of EAs and SA. If locality is high, heritability is also high. So, if the best solution is a star, EAs and SA perform well. If the best solution is in a form of a list or a

Algorithm 15 All Degree Constrained Spanning Trees

input: A complete graph G with n nodes, a degree bound b.

output: The set of all degree constrained spanning trees of G.

1: len=$n-2$

2: code=empty

3: $degree\_count_i$ = 0, where $1 \leq i \leq n$.

4: **if** len = 0 **then**

5: Print or Decode code.

6: **end if**

7: **for** $digit$=1 to n **do**

8: **while** $degree\_count_{digit} <$ bound **do**

9: Add $digit$ as the leftmost digit of code.

10: $degree\_count_{digit}$++

11: len–

12: goto 4

13: **end while**

14: $degree\_count_{digit}$ = $degree\_count_{digit}$- 1

15: **end for**

random tree, neither EA nor SA can solve even simple large OMTP instances.

Although degree computation is easy for Neville's and Prüfer's encodings, because of low locality, neither Neville's nor Prüfer's encoding techniques can solve even simple large DCMSTPs.

When ASTP and ADCSTP are considered, given algorithms have optimum time and space complexities with Neville branch numbers only for complete graphs. So, our algorithms have best time and space complexities only for weighted and unweighted complete graphs.

CHAPTER 5

CONCLUSIONS

This thesis presents the study of the Neville's encoding techniques. There were no previously reported algorithms for encoding and decoding two of the Neville's encoding techniques. Both time-wise and space-wise optimal algorithms for Neville branch encoding technique are given in Algorithm 11 and Algorithm 12. with $O(n)$ time and space complexities. Properties of Neville's encodings are analyzed in detail using EAs and SA. Neville's tree encoding techniques have some benefits. Firstly, every tree can be represented by unique Neville branch and leaf numbers and every Neville branch and leaf number encodes exactly one tree. Classical crossover and mutation operations on Neville encodings produce new valid Neville encodings. Thus, Neville encodings are closed under classical EA operations. If Neville encodings are used to represent a tree in a graph which is not complete, then encoded tree may not be a valid tree. Secondly, computation of degrees of nodes is very simple. Degree of a node of encoded tree is one more than the number of times the node appears in Neville's encodings. Therefore, Neville's encodings can be used for degree constraint computations.

Despite the benefits of the encodings, they also have some disadvantages: Neville encodings have poor locality and heritability. Locality analysis are done by random walks and neighborhood analysis through the search spaces of Neville's encodings. Analysis showed that, although locality becomes maximal when the encoded tree is a star, locality of Neville's encodings is low for random type trees. Heritability is also low when locality is low, and encodings perform well on EAs and SA only when the locality is high. Thus, none of the Neville's encodings are suitable for EAs and SA.

It is shown that locality properties affect the performance of encodings on problems.

OMTP experiments are successful only when goal tree is star type or small. Otherwise, Neville's encodings are not suitable for OMTP. Also, although for small and simple inputs the degree constraint computation is simple for Neville's encodings, Neville's encodings cannot solve DCMSTP of random trees, because of their poor locality.

Neville's encodings are successful only for degree constrained computation and tree enumeration of complete graphs. Generating a code has $O(n)$ time and $O(n)$ space complexities, where n is the number of nodes of complete graph G. The best known algorithm for ASTP, which only enumerates spanning trees but does not prints them, has $O(s+n+e)$ time and $O(n+e)$ space complexities [25], where s is the number of spanning trees and e is the number of edges. For a complete graph, since $s = n^{n-2}$, time and space complexities become $O(n^{n-2})$ and $O(n+e)$, respectively. Though, we presented time and space optimal algorithm Algorithm 13, to enumerate all spanning trees of complete graph G with $O(n^{n-2})$ time and $O(n)$ space complexities. Both algorithms have the same time complexity but our algorithm is better for space complexity. If all spanning trees are printed, each code needs to be decoded. When Neville branch encoding is used, time and space complexities of our solution to ASTP and printing all spanning trees have $O(n^{n-1})$ and $O(n)$ complexities, respectively. The optimal algorithm which enumerates all spanning trees by outputting all edges of each spanning tree was reported to have $O(sn+n+e)$ time and $O(n+e)$ space complexities [5]. For a complete graph, the time and space complexities becomes: $O(n^{n-1}+n+e)$ and $O(n+e)$, respectively. Thus our algorithm Algorithm 14 is better for complete graphs. For ADCSTP, degree constraint adds no complexity to the ASTP algorithm. When degree constraint is added, ADCSTP is solved to output entire spanning trees with $O(n^{n-1})$ and $O(n)$ time and space complexities. If only codes are given as output, time complexity is $O(n^{n-2})$ and space complexity is $O(n)$. As a result, time and space optimal algorithms for ASTP and ADCSTP of weighted or unweighted complete graphs are presented by using Neville branch encodings. However, when the graph is an incomplete sparse graph, these solutions are not optimal.

In this thesis, locality properties of Neville's encodings are given. New problems, for which Neville's encodings are suitable, may be searched and performances of encodings can be analyzed on different problems. Moreover, locality analysis are done according to basic mutation. New mutation and crossover algorithms can be searched for better locality and heritability.

REFERENCES

[1] R. K. Ahuja, J. B. Orlin, A. Tiwari, A Greedy Genetic Algorthm for the Quadtatic Assignment Problem, *Computers and Operations Research*, vol.27, pp. 917-934, 2000.

[2] T. Bäck, H.-P. Schwefel, Evolution Strategies I: Variants and their computational implementation, *Genetic Algorithms in Engineering and Computer Science, Proceedings of the First Short Course EUROGEN'95*, Wiley, 1995

[3] N. Deo, P. Micikevicius, Prufer-like codes for labeled trees, *Congressus Numerantium*,151:65-73, 2001.

[4] S. Even, Algorithmic Combinatorics, New York: The Macmillan Company, pp. 104-106, 1973.

[5] H. N. Gabow, E. W. Myers, Finding All Spanning Trees of Directed and Undirected Graphs, *SIAM J. Comput.*, 24, pp.280-287, 1978.

[6] M.R. Garey, D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, *Freeman*, San Francisco, 1979.

[7] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, 1989.

[8] J. Gottlieb, B. A. Julstrom, G. R. Raidl, and F. Rothlauf, Prüfer numbers: A poor representation of spanning trees for evolutionary search, In L. Spector et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2001*, pp. 343-350, Morgan Kaufmann, 2001.

[9] J. Gottlieb, B. A. Julstrom, G. R. Raidl, and F. Rothlauf, Prüfer numbers: A poor representation of spanning trees for evolutionary search, *Technical Report 2001001*, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at UrbanaChampaign, 2001.

[10] J. Gottlieb, B. A. Julstrom, G. R. Raidl, and F. Rothlauf, Prüfer numbers: A poor representation of spanning trees for evolutionary search, *Working Paper 12/2000*, Department of Information Systems, University of Bayreuth, 2000.

[11] P. Gray, W. Hart, L. Painton, C. Phillips, M. Trahan, J. Wagner, A Survey of Global Optimization Methods, Sandia National Laboratories, 1997.

[12] S. Kapoor, H. Ramesh, Algorithms for Enumerating All Spanning Trees of Undirected and Weighted Graphs, SIAM J. Comput., 1995

[13] M. Krishnamoorthy, A.T. Ernst, Y.M. Sharaiha, Comparison of Algorithms for the Degree Constrained Minimum Spanning Tree, *Journal of Heuristics*, vol. 7, num 6, pp. 587-611, 2001.

[14] T. Matsui, A Flexible Algorithm for Generating All the Spanning Trees in Undirected Graphs, *Algorithmica*, vol. 18, pp. 530-544, 1997.

[15] E. H. Neville. The codifying of tree-structure. *Proceedings of Cambridge Philosophical Society*, vol. 49, pp. 381-385, 1953.

[16] A. Nijenhuis, H.S. Wilf, Combinatorial Algorithms for Computers and Calculators, New York, Academic Press, 1978.

[17] C.C. Palmer, A. Kershenbaum, Representing Trees in Genetic Algorithms, *Proceedings of the First IEEE International Conference on Evolutionary Computation*, vol. 1, pp. 379-384, 1994.

[18] H. Prüfer, Neuer Beweis eines Satzes ueber Permutationen, *Archiv für Methematik und Physik*, 27:742-744, 1918.

[19] F. Rothlauf, Representations for Genetic and Evolutionary Algorithms, Physica-Verlag, 2002.

[20] F. Rothlauf, D.E. Goldberg, Pruefernumbers and Genetic Algorithms: A Lesson How The Low Locality Of An Encoding Can Harm The Performance Of GAs, *Illegal Report No. 2000011, Illinois Genetic Algorithms Laboratory*, 2000.

[21] F. Rothlauf, D.E. Goldberg, Tree Network Design with Genetic Algorithms-An Investigation in the Locality of the Pruefernumber Encoding, In S. Brave and A. S. Wu eds., *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pp. 238-243, Orlando, FL, 1999.

[22] F. Rothlauf, D.E. Goldberg, A. Heinzl, Network Random Keys -A Tree Representation Scheme for Genetic nad Evolutionary Algorithms, *Evolutionary Computation*, vol 10, pp. 75-97, 2002.

[23] S. Russell, P. Norvig, Artificial Intelligence A Modern Approach, Prentice Hall International Editions, 1995.

[24] A. Shioura, A. Tamura, Efficiently Scanning All Spanning Trees of an Undirected Graph, *J. Oper. Res. Soc. Japan*, 38, pp. 331-334, 1995.

[25] A. Shioura, A. Tamura, T. Uno, An Optimal Algorithm for Scanning All Spanning Trees of an Undirected Graphs, *SIAM J COMPUT.*, vol. 26, no 3. pp. 678-692, 1997.

[26] W. M. Spears, K. A. De Jong, Th. Back, D. B. Fogel, H. de Garis, An overview of evolutionary computation, *In P. B. Brazdil, editor, Machine Learning: ECML-93*, volume 667 of Lecture Notes in Articial Intelligence, pp. 442-459, Springer, Berlin, 1993.

APPENDIX A

DATA SETS

First data set with best solution 43:

```
8 2
100 4 100 100 100 100 3 4
4 100 8 100 100 100 6 100
100 8 100 7 100 4 100 100
100 100 7 100 9 14 100 100
100 100 100 9 100 10 100 100
100 100 4 14 10 100 100 100
3 6 100 100 100 100 100 5
4 100 100 100 100 100 5 100
```

APPENDIX B

PARAMETERS

B.1 OMTP

Parameters of experiments on OMTP are given in two section, for EAs and SA.

B.1.1 EA

All EAs used are based on Goldberg's simple genetic algorithm [7], given at Section 2.1. For each test, 100 independent runs were performed and each run is terminated after the population is fully converged. In all simulations one-point crossover with crossover probability 1 and mutation probability 0.5 is used. As selection scheme, $(\lambda + \mu)$ selection is applied.

Table B.1: Parameters of OMTP with EA.

Figure	number of nodes	tree type	parent selection	λ	μ	limit on duplication
4.1	8	random	roulette wheel	500	500	no
4.2	16	random	roulette wheel	500	500	no
4.3	16	random	random	500	500	no
4.4	16	random	roulette wheel	500	500	yes
4.5	32	random	roulette wheel	1500	1500	no
4.6	32	star	roulette wheel	1500	1500	no

B.1.2 SA

SA given in Section 2.2 is used for the SA tests. T is computed as:

$$T = T_0 - step\frac{T_0 - T_{final\_step\_count}}{final\_step\_count} where T_0 = 1 and T_{final\_step\_count} = 0$$

For each problem 20 independent runs were performed.

Table B.2: Parameters of OMTP with SA.

Figure	number of nodes	tree type	step
4.7	8	random	6000
4.8	32	star	12000

B.2 DCMSTP

B.2.1 EA

Similar to OMTP experiments, all EAs used are based on Goldberg's simple genetic algorithm [7], given at Section 2.1. For each test, 100 independent runs were performed and each run is terminated after the population is fully converged. In all simulations one-point crossover with crossover probability 1 and mutation probability 0.5 is used. As selection scheme, $(\lambda + \mu)$ selection is applied.

Table B.3: Parameters of DCMSTP with EA.

Figure	number of nodes	tree type	parent selection	λ	μ	limit on duplication
4.9	8	random	roulette wheel	500	500	no

B.2.2 SA

SA given in Section 2.2 is used for the SA tests. same function given at Section B.1.2 is used to compute T. For each problem 20 independent runs were performed.

Table B.4: Parameters of DCMSTP with SA.

Figure	number of nodes	tree type	step
4.10 & 4.11	8	random	6000