İSTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

THE ROLE OF REPRESENTATIONS IN DYNAMIC ENVIRONMENTS

M. Sc. Thesis by
Merve ORBAYI, Eng.

Department : **Advanced Technologies**

Programme: **Computer Sciences**

JUNE 2006

İSTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

THE ROLE OF REPRESENTATIONS IN DYNAMIC ENVIRONMENTS

M.Sc. Thesis by
Merve ORBAYI, Eng.

704031012

Date of submission : 5 May 2006
Date of defence examination: 29 June 2006

Supervisor (Chairman):	**Assoc. Prof. Dr. A. Şima ETANER-UYAR (İTU)**
Supervisor:	**Dr. Juergen BRANKE (UK)**
Members of the Examining Committee	**Assoc. Prof. Dr. Borahan TÜMER (MU)**
	Assoc. Prof. Dr. Şule GÜNDÜZ-ÖĞÜDÜCÜ (İTU)
	Assoc. Prof.Dr. Zehra ÇATALTEPE (İTU)

JUNE 2006

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

GÖSTERİLİMLERİN DİNAMİK ORTAMLARDAKİ ROLÜ

Yüksek Lisans Tezi

Bilg. Müh. Merve ORBAYI

704031012

Sunulma tarihi : 5 Mayıs 2006
Savunma tarihi : 29 Haziran 2006

Danışman (Başkan): Yrd. Doç. Dr. A. Şima ETANER-UYAR (İTU)
Danışman: Dr. Juergen BRANKE (UK)
Savunma jürisi üyeleri Yrd. Doç. Dr. Borahan TÜMER (MU)
Yrd. Doç. Dr. Şule GÜNDÜZ-ÖĞÜDÜCÜ (İTU)
Yrd. Doç. Dr. Zehra ÇATALTEPE (İTU)

HAZİRAN 2006

ACKNOWLEDGMENT

First of all, I am thankful to my both advisors, Assoc. Prof. Dr. Şima Etaner - Uyar and Dr. Juergen Branke for their efforts and guidance. Then grateful to my family for their supports through all my life. I would also like to thank Erdem Salihoğlu and Michael Stein for their help.

MAY 2006 Merve ORBAYI

TABLE OF CONTENTS

ABBREVIATIONS

EA	: Evolutionary Algorithm
dMKP	: Dynamic Multi Dymansional Knapsack Problem
dTSP	: Dynamic Travelling Salesman Problem
TSP	: Travelling Salesman Problem
KP	: Knapsack Problem
MKP	: Multi Dimensional Knapsack Problem
PMX	: Partially Mixed Crossover
UOBX	: Uniform Order Based Crossover
WC	: Weight Coding
GA	: Genetic Algorithm

LIST OF TABLES

LIST OF FIGURES

Page Numbers

LIST OF SYMBOLS

For Multi- dimensional Knapsack Problem

x_j	: Decision variable showing if item j is added to knapsack
n	: Number of items
m	: Number of resources
p_j	: Profit of item j
r_{ij}	: Resource consumption of item j for resource i
c_i	: Capacity constraint of resource i
σ	: Standard deviation of the normally distributed random variable
lb	: Multiplier used to define lower boundaries for change severity
ub	: Multiplier used to define upper boundaries for change severity

For Traveling Salesman Problem

n	: Number of cities
P_n	: Collection of all permutations of the set $\{1,2,\ldots,n\}$
c_{ij}	: Cost of the edge joining node i to node j
$p_{\max}$	: The largest profit value
$r_{\min}$	: The minimum resource consumption
CV(x, i)	: The maximum constraint violation for the i-th constraint c_i

GÖSTERİLİMLERİN DİNAMİK ORTAMLARDAKİ ROLÜ

ÖZET

Gösterilimlerin evrimsel algoritmalar üzerindeki etkisi durağan ortamlar için bugüne kadar birçok çalışmada incelenmiş, bununla beraber dinamik ortamlardaki etkisi ihmal edilmiştir. Bu çalışmada, farklı gösterilimlerin dinamik ortamlardaki etkisini deneysel olarak inceledik.

Probleme ara dönüşüm olmaksızın çözüm olabilen gösterilimlere, doğrudan gösterilimler denir. Dolaylı gösterilimlerde ise, arama farklı bir uzayda yapıldığından dolayı, çözümün doğrudan gösterilim haline ulaşmak için bir dönüşüm gerekir. Genelde, doğrudan gösterilimler tüm uzayda arama yaptıklarından dolayı geçersiz çözümlerle de evrim sırasında baş etmelidir. Dolaylı gösterilimler genetik arama uzayında arama yaptığı ve aday çözümler genelde geçerli çözüm uzayına izdüştüğü için, geçersiz çözüm problemi yoktur.

Testlerde çoğunluk tarafından bilinen çok boyutlu sırt çantası (multi-dimensional knapsack problem) ve gezgin satıcı (traveling salesman) optimizasyon problemleri için doğrudan ve dolaylı yöntemleri inceledik, ve karşılaştırdık.

Çok boyutlu sırt çantası problemi için seçtiğimiz dolaylı ağırlık kodlama (weight coding) yöntemi ile travelling salesman problemi için kullandığımız dolaylı ötelenmis koordinatlar (perturbed coordinates) yöntemi, çözümü bulmak için temelde aynı mantığı paylaşıyor. Her iki yöntemde de aday çözümler, orjinal problemin bazı değerlerini değiştirerek biraz farklı bir problem elde ediyor. Daha sonra elde edilen yeni probleme hızlı bir sezgisel yöntemle çözüm buluyor. Bulunan bu çözümü de orjinal problemin çözümüymüş gibi kullanıyor.

Dolaylı gösterilimlerde, her değişim anında, toplumu oluşturan çözümlerin genetik arama uzayındaki bileşenleri, çözüm uzayına izdüşürülerek, toplumun yeni probleme adapte olması sağlanıyor.

Sonuçlar dolaylı gösterilimlerin değişim anlarında, sahip oldukları sezgisel adaptasyon mekanizmasıyla var olan çözümleri yeni probleme adapte etmelerinden dolayı dinamik problemler için daha uygun olduğunu gösterdi. Ek olarak, gösterilimlerin dinamik ortamlardaki etkisinin, statik ortamlardakinden daha büyük olduğunu gördük. Bu nedenle dinamik ortamlarda gösterilimler seçilirken seçici olunmalı, adaptif yöntemler tercih edilmelidir.

THE ROLE OF REPRESENTATIONS IN DYNAMIC ENVIRONMENTS

SUMMARY

The effect of different representations has been thoroughly analyzed for evolutionary algorithms in stationary environments. However, the role of representations in dynamic environments has been largely neglected so far. In this study, we empirically analyze the effects of different representations in dynamic environments.

A representation is called direct, if it can be interpreted directly as a solution to the problem. In indirect representations, search is done in a different space, and a mapping is required to get the direct representation of the solution. Direct representations search in the entire search space, so generally the method should deal with the infeasible solutions through evolution. In indirect representations, search is done in genetic search space and generally all elements map to feasible solutions of the search space. Thus indirect representations do not have a problem of infeasible solutions.

In this thesis, we analyze and compare direct and indirect representations for two generally known optimization problems, the multi-dimensional knapsack problem and the traveling salesman problem.

Two indirect representations selected, weight coding for multi-dimensional knapsack problem and perturbed coordinates for traveling salesman problem, follow the same track while finding the solution. In both approaches, candidate solutions first create a slightly changed problem, by changing some values of the original problem. Then they find a solution to the changed problem using a fast heuristic.

In indirect representations, at each change point, the existing population is adapted to the new problem by mapping the solutions' components of the genetic search space to the solution space.

Our results indicate that indirect representations are particularly suitable for dynamic problems, because they implicitly provide a heuristic adaptation mechanism that improves the current solutions after a change. In addition, we saw that the choice of representation in dynamic environments is even more important than in static environments. For this reason, one should be careful when selecting a representation to be used in a dynamic environment, and should prefer adaptive representations.

1 INTRODUCTION

Many real-world problems are dynamic in nature. The interest in applying evolutionary algorithms (EAs) in dynamic environments has been increasing over the past years, which is reflected in the increasing number of papers on the topic. For an in-depth overview on the topic, see e.g. [1-4].

Most of the literature attempts to modify the algorithm to allow a better tracking of the optimum over time, e.g. by increasing diversity after a change, maintaining diversity throughout the run, or incorporating a memory. In this thesis, we focus on the representation's influence on an EA's performance in dynamic environments. Instead of searching the solution space directly, usually EAs search in a transformed space defined by the genetic encoding. This mapping between solution space and genetic search space is generally called "representation", or "genotype-phenotype mapping". The representation together with the genetic operators and the fitness function define the fitness landscape, and it is generally agreed upon that a proper choice of representation and operators is crucial for the success of an EA, see e.g.[5-6].

Depending on the representation, the fitness landscape can change from being unimodal to being highly multimodal and complex, and thus the representation strongly influences the EA's ability to approach the optimum. In a dynamic environment, in addition to the (static) characteristics of the fitness landscape, the representation influences the characteristics of the fitness landscape dynamics, as has been recently demonstrated in [7]. Consequently, depending on the representation, the tracking of the optimum over time may be more or less difficult.

This thesis examines the performance of different genetic representations for the dynamic multi-dimensional knapsack problem (dMKP) and for the dynamic travelling salesman problem (dTSP). Both problems are well studied. Different representations have been proposed and compared e.g. in [5, 8-10] and in [11-14] for the multi-dimensional knapsack problem (MKP) and the traveling salesman problem (TSP) respectively. For our study, for dMKP, the binary representation with a

penalty for constraint handling is selected as an example of a direct representation. As indirect representations, we consider a permutation representation and a weight-coding. In the latter, the items' profits are modified and a simple deterministic heuristic is used to construct the solution. For dTSP, we selected permutation representation as direct and perturbed city coordinates as indirect representation. In the latter, city coordinates are modified and a minimum distance connecting heuristic is used to construct the solution. Intuitively, indirect representations for both problems seem particularly promising for dynamic environments, as they naturally incorporate heuristic knowledge that would immediately improve a solution's phenotype after a change of the problem instance.

Organization of the thesis is as follows. Section 2 presents the test problems used and gives details about how we made them dynamic. In Section 3, properties of the evolutionary algorihtm, representations and genetic operators selected are explained. Section 4 describes the experiments and discusses emprical results. Finally, the document finishes with a conclusion in Section 5.

2 DYNAMIC PROBLEMS

TSP and Knapsack problems (KP) are well known and popular problems that has become a standard for testing computational algorithms. Hence, we used these problems in our study. Problem definitions, how we implemented dynamic versions and the original problems used are given in the following sections.

2.1 Dynamic Multi-dimensional Knapsack Problem

Knapsack problems [15] are commonly used combinatorial benchmark problems. MKP belongs to the class of NP-complete problems. MKP has a wide range of real world applications such as cargo loading, selecting projects to fund, budget management, cutting stock, etc. It can be formalized as follows.

maximize $$\sum_{j=1}^{n} p_j . x_j \quad \textbf{(2.1)}$$

subject to $$\sum_{j=1}^{n} r_{ij} . x_j \leq c_i, i = 1,2,...,m \quad \textbf{(2.2)}$$

where n is the number of items, m is the number of resources, $x_j \in \{0,1\}$ shows whether item j is included in the subset or not, p_j shows the profit of item j, r_{ij} shows the resource consumption of item j for resource i and c_i is the capacity constraint of resource i. In words, objective is to select a subset of items such that the total profit of the selected items is maximized and consumed resources do not exceed the corresponding resource capacity.

For the MKP, several different genetic representations and genetic operators have been proposed. A detailed analysis and comparison for static environments can be found in [8] and more recently in [5]. In Section 3, we describe the representations selected for our tests in dynamic environments.

In our study, we use a dynamic version of MKP as proposed in [7] and described below. Basis is the first instance given in the file mknapcb4.txt which can be downloaded from [16]. It has 100 items, 10 knapsacks and a tightness ratio of 0.25. For every change, the profits, resource consumptions and the constraints are multiplied by a normally distributed random variable as follows:

$$\begin{aligned} p_j &\leftarrow p_j * (1 + \mathrm{N}(0, \sigma_p)) \\ r_{ij} &\leftarrow r_{ij} * (1 + \mathrm{N}(0, \sigma_r)) \\ c_i &\leftarrow c_i * (1 + \mathrm{N}(0, \sigma_c)) \end{aligned} \quad \textbf{(2.3)}$$

Unless specified otherwise, the standard deviation of the normally distributed random variable used for the changes has been set to $\sigma_p = \sigma_r = \sigma_c = 0.05$ which requires on average 11 out of the 100 possible items to be added or removed from one optimal solution to the next. Each profit p_j, resource consumption r_{ij} and constraint c_i is restricted to an interval as determined in Equation (2.4).

$$\begin{aligned} lb_p * p_j \le p_j \le ub_p * p_j \\ lb_r * r_{ij} \le r_{ij} \le ub_p * r_{ij} \\ lb_c * c_i \le c_i \le ub_p * c_i \end{aligned} \quad \textbf{(2.4)}$$

where $lb_p = lb_r = lb_c = 0.8$ and $ub_p = ub_r = ub_c = 1.2$. If any of the changes causes any of the lower or upper bounds to be exceeded, the value is bounced back from the bounds and set to a corresponding value within the allowed boundaries.

2.2 Dynamic Traveling Salesman Problem

The traveling salesman problem [17] represents a typical ‘hard’ combinatorial optimization problem. The problem has been shown to be NP-hard.

Applications of the TSP and its variations are not used only for planning the route of a traveling salesman but also for several areas of knowledge such as mathematics, computer science, operations research, genetics, engineering, and electronics. Machine sequencing and scheduling, route planning, job scheduling, electronic circuit board drilling, integrated circuit fabrication are just some of the examples for real world applications. Problem can be formalized as follows.

subject to $\pi = (\pi(1), \pi(2),\ldots, \pi(n)) \in P_n$ **(2.5)**

minimize $c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$ **(2.6)**

where P_n is the collection of all permutations of the set $\{1,2,\ldots,n\}$, n is the number of cities, $(\pi(1), \pi(2), \ldots,\pi(n))$ gives the order in which cities are visited starting with city $\pi(1)$, c_{ij} corresponds to the cost of the edge joining node i to node j. In words, TSP is to find a route, visiting each city exactly once and returning to the starting city in such a way that the total distance traveled is minimum.

For the TSP, several different genetic representations and genetic operators have been proposed. In Section 3, we describe the representations selected for our study in dynamic environments.

We generated 50 and 100 uniform random points separately in a unit square region of the Euclidean plane, which form the basis problems for our tests. For every change, a predetermined number of cities, according to the change severity wanted, are removed and same number of new cities are added to problem. We changed 2, 4, and 6 cities to implement different change severities.

3 THE EVOLUTIONARY ALGORITHM

For this study, we used a more or less standard steady-state EA. Pseudo code for the EA is as seen in Figure 3.1. However binary tournament selection is used for all representations; other genetic operators, crossover and mutation depend on the representation and have been implemented as described in the following sections. The new child replaces the current worst individual in the population if its fitness is better than or equal to the worst. The EA uses phenotypic duplicate avoidance, i.e. a child is re-generated if a phenotypically identical individual already exists in the population. This feature seems important in particular for indirect representations with high redundancy, i.e. where many genotypes are decoded to the same phenotype.

Unless stated otherwise, after a change, the whole population is re-evaluated before the algorithm is presumed. The genotypes are kept unless the change creates phenotypically identical individuals, in which case duplicates are randomized.

```
generate n random individuals (n is the population size)
calculate fitness of the initial population
repeat
        select two indiviuals
        apply crossover
        apply mutation
        calculate fittness of the offspring
        replace worst with offspring, if worst is worse than or equal to offspring
until end condition
```

Figure 3.1 : Pseudo code of the EA

3.1 Direct Representations

A representation is called direct if it can be interpreted directly as a solution to the problem. One drawback of direct representations is often the difficulty to maintain feasibility, as the search space contains many infeasible solutions. Infeasible solutions can be eliminated from search by large penalties, or they can be included to the search if graded penalty terms are used. We give details of how we dealt with infeasible solutions for the direct representations we used, in related sections.

Although several genetic representations have been used for TSP and MKP, we selected binary representation for MKP and path representation for TSP, which are both common and simple. You can find explanations and details about these methods in the following sections.

3.1.1 Binary Representation with Penalty For MKP

In binary representation, a candidate solution for MKP is a bit string where each bit correponds to an item, indicating whether an item should be included in the knapsack or not. To drive the search towards feasible regions of the search space, we use a simple penalty-based method. We apply the penalty mechanism recommended in [18], which guarantees that feasible solutions are always preferred over infeasible ones.

$$fitness(x) = f(x) - penalty(x) \quad \textbf{(3.1)}$$

$$penalty(x) = \frac{p_{\max} + 1}{r_{\min}} * \max\{CV(x,i) \mid i = 1...m\} \quad \textbf{(3.2)}$$

$$p_{\max} = \max\{p_i \mid i = 1...m\} \quad \textbf{(3.3)}$$

$$r_{\min} = \min\{r_{ij} \mid i = 1...m, j = 1...n\} \quad \textbf{(3.4)}$$

$$CV(x,i) = \max(0, \sum_{j=1}^{n} r_{ij}.x_j - c_i) \quad \textbf{(3.5)}$$

where $p_{\max}$ is the largest profit value calculated as in Equation (3.3), $r_{\min}$ is the minimum resource consumption calculated as in Equation (3.4) and CV(x, i) is the

maximum constraint violation for the *i*-th constraint c_i calculated as in Equation (3.5). It should be noted that $r_{\min} \neq 0$ must be ensured. In a dynamic environment, fitness of the solutions should be "re-calculated" after a change; as some of the profits, resource consumptions and constraints are different in the new environment. As genetic operators bit flip mutation and uniform crossover are used.

3.1.1.1 Bit Flip Mutation

Flipping random bits of the genotype.

For example;

00001111 may become 00001110

if the 8^{th} bit is flipped.

3.1.1.2 Uniform Crossover

Each gene of the offspring is selected randomly from the corresponding genes of the parents. Uniform crossover produces one offspring.

For example;

Parent 1 : 00000000

Parent 2 : 11111111

then offspring could be

Offspring : 01010101

3.1.2 Path Representation For TSP

In Path Representation for TSP, a tour is represented as a list of *n* cities in a meaningful order. *i*-th city of the list shows the *i*-th city to be visited. For a problem consists of five cities, the tour 3-1-0-2-4 is represented by (31024). As each city is used only once in the permutation, all search space is (tour possibilities are) feasible in this representation. In a dynamic environment, fitness of the solutions should be "re-calculated" after a change; as some new cities are added and some are removed from the environment. Inversion mutation and partially mixed crossover (PMX) are used as variation operators.

3.1.2.1 Inversion Mutation

In inversion mutation two points are selected randomly, and section between these two points are reversed.

For example;

12 | 3456 | 78 may become 12 | 6543 | 78

if the 3 and 6 are the two randomly selected points that determines the borders of section to be reversed.

3.1.2.2 Partially Mixed Crossover

In PMX, two points are selected randomly and sections between points are swapped between offsprings.

For example;

Parent 1 : 12 | 345 | 678

Parent 2 : 13 | 572 | 468

then offsprings are

Offspring 1 : 13 | 572 | 648

Offspring 2 : 12 | 345 | 768

if the 3 and 6 are the two randomly selected points that determines the borders of sections to be swapped.

3.2 Indirect Representations

Indirect representations require to run a decoder to generate the solution based on the genotype. There are many possible indirect representations for both problems. Usually, a representation is preferred that decodes all elements of the search space into feasible solutions. Thus, it is not necessary to design complicated repair mechanisms or to use a penalty to ensure feasibility. In this thesis, we implemented two indirect representations for MKP and one for TSP discussed below.

3.2.1 Permutation Representation For MKP

A popular indirect representation for MKP is the permutation representation [8,19], where the search space consists of all possible permutations of the items. To obtain the phenotype (actual solution), a decoder starts with an empty set, then considers the items one at a time in the order specified by the permutation. If an item can be added without violating any constraint, it is included in the solution, otherwise not.

The decoder used by the permutation representation guarantees that only feasible solutions are generated. Furthermore, these solutions lie on the boundary of the feasible region in the sense that no additional items could be included without violating at least one constraint, which is a necessary condition for optimality. Thus, the decoder generates solutions that are of significantly higher quality than randomly generated solutions. In a dynamic environment, solutions are immediately "repaired" after a change such that they are again at the boundary of the feasibility in the new environment.

In [9], a good setup for the permutation representations recommended, including uniform order based crossover (UOBX) and insert mutation as variation operators.

3.2.1.1 Insert Mutation

In insert mutation, a new position for an element is selected randomly. The mutated element is inserted into its new position and the other elements are re-positioned accordingly.

For example;

Case 1: Element position > new position

123**4**5678 may become **4**1235678

if the 4 and 1 are the two randomly selected points that determines the position of element selected and its new position respectively.

Case 2: Element position < new position

123**4**5678 may become 234**1**5678

if the 1 and 4 are the two randomly selected points that determines the position of element selected and its new position respectively.

3.2.1.2 Uniform Order Based Crossover

In UOBX, some positions are transferred directly to the offspring from the first parent with probability $p_1 = 0.45$. Then, starting from the first position, undetermined positions are filled with missing items in the order of the second parent.

For example;

Parent 1 : 12345678

Parent 2 : 13572468

then offsprings are

Offspring 1 : 1*3*5***

Offspring 1 : **1**7**3**2**5**468

Offspring 2 : 1*5*2***

Offspring 2 : **1**3**5**4**2**678

if 1, 3 and 5 are the randomly selected points which are positions for direct transfer.

3.2.2 Real Valued Representation with Weight Coding For MKP

A more complex example for indirect representations for MKP is the weight-coding (WC) approach [10]. In the weight-coding technique, a candidate solution for the MKP consists of a vector of real-valued genes (biases) associated with each item. To obtain the corresponding phenotype, first the original problem P is transformed (biased) into a modified problem P' by multiplying the original profits of each item with the corresponding bias. Then, a fast heuristic is used to find a solution to P', and finally, the resulting solution (items to be placed in the knapsack) is evaluated based on the original problem. Raidl [10]discusses two possible decoding heuristics. The one using the surrogate relaxation method is preferred due to its lower computational requirements. The surrogate relaxation method [20] simplifies the original problem by transforming all constraints into a single one as follows:

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{m} a_i . r_{ij}\right) x_j \le \sum_{i=1}^{m} c_i \qquad \textbf{(3.6)}$$

where a_i is the surrogate multiplier for the *i*-th constraint, and r_{ij} is the resource coefficient.

Surrogate multipliers are determined by solving the relaxed MKP (i.e., variables x_i can take any value $\in [0,1]$) by linear programming, and using the values of the dual variables as surrogate multipliers. Then, to obtain a heuristic solution to the MKP, the profit/pseudo-resource consumption ratios denoted as u_j are calculated as given in Equation (3.7).

$$u_j = \frac{p_j}{\sum_{i=1}^{m} a_i r_{ij}} \quad \textbf{(3.7)}$$

The items are then sorted in decreasing order based on their u_j values, and this order is used to construct solutions just as for the permutation representation, i.e. items are considered one at a time, and is none of the constraints are violated, added to the solution. To keep computation costs low, in [10] the surrogate multiplier values a_i are determined only once for the original problem at the beginning. As a result, the decoding step starts with the computation of the u_j values based on the biased profits. Note that in a dynamic environment, the WC representation requires to re-compute the pseudo-resource consumption values a_i after every change of the environment.

In [10], several biasing techniques are discussed and compared. We initialize the biases according to $w_j = 2^R$, where R is a uniformly generated variable in the range [-1, 1]. This leads to a distribution with many small and few larger values. For mutation, we deviate from the re-initialization of biases used in [10] and instead use Gaussian mutation with σ = 1. To generate the modified profits, the original profits are simply multiplied with the biases, i.e. $p'_j = p_j * w_j$. Uniform crossover is used as second genetic operator.

3.2.2.1 Gaussian Mutation

Adding a random value from a Gaussian distribution to each gene of the parent to create a new offspring.

3.2.2.2 Uniform Crossover

Definition is given in Section 3.1.1.2 for binary representation with penalty for MKP.

3.2.3 Permutation vs. Weight Coding

Since the permutation representation and the WC representation share similar construction mechanisms, they both benefit from the resulting heuristic bias. However, by calculating the pseudo-resource consumption values, the influence of heuristic knowledge for WC is even larger.

In dynamic environments, the WC representation appears to be particularly advantageous, for two reasons:

1. Because of the integration of heuristic knowledge, good solutions are generated right from the beginning, i.e., the algorithm improves more quickly. In dynamic environments, time is scarce (otherwise one could just regard the problem as a sequence of stationary problems), and the heuristic bias gives this representation a head start.
2. Changes of the environment are immediately taken into account by the underlying heuristic, which means that the existing solutions are heuristically adjusted after a change of the problem instance.

3.2.4 Perturbed Coordinates For TSP

This technique [21] uses a genetic algorithm (GA) to breed perturbed city coordinates in the corresponding perturbation zone. The perturbed coordinate sets are the chromosomes and genetic operators are applied to perturbed city coordinates. The aim is to fool the simple heuristic algorithms for producing much better solutions. With the chosen heuristic algorithm, a tour is produced according to the perturbed city coordinates. The original coordinates are then used to calculate intercity distances for computing the tour length. Scale of the perturbation zone for random uniform points is calculated with the formula as given in Equation (3.8).

$$l = k\frac{R^p}{n^q} \tag{3.8}$$

In Equation (3.8), R is the area of the (square) region, *n* is the problem size. We used p=1/2, q=3/2 and k=60 as suggested in the paper. We used minimum distance connecting heuristic, as it is described in [21]. Cities closest to each other are joined in order while no city is allowed to join more than 2 cities and no tour is permitted of length less than *n*, the number of cities in the problem. In a dynamic environment, fitness of the solutions should be "re-calculated" after a change; as some new cities are added and some are removed from the environment. Uniform crossover and random reset mutation is used.

3.2.4.1 Random Reset Mutation

Resetting each gene randomly according to a predetermined mutation probability.

3.2.4.2 Uniform Crossover

Definition is given in Section 3.1.1.2 for binary representation with penalty for MKP.

4 EXPERIMENTS

All results are averages over 50 runs with different random seeds but on the same series of environment changes. As a performance measure, we use the error to the optimum. Note that the following analysis assumes that the evaluation is by far the most time-consuming operation (as is usual for many practical optimization problems), allowing us to ignore the computational overhead caused by the decoders.

4.1 Dynamic Multi-dimensional Knapsack Problem

For MKP tests**,** we used an EA with a population size of 100, crossover probability of 1.0, and mutation probability of 0.01 per gene. We use glpk [22] for calculating the surrogate multipliers for the WC and CPLEX for calculating the true optimum for each environment.

4.1.1 Relative Performance in Stationary Environments

Figure 4.1 compares the three representations on a stationary environment, which will serve as a baseline for the comparison in dynamic environments.

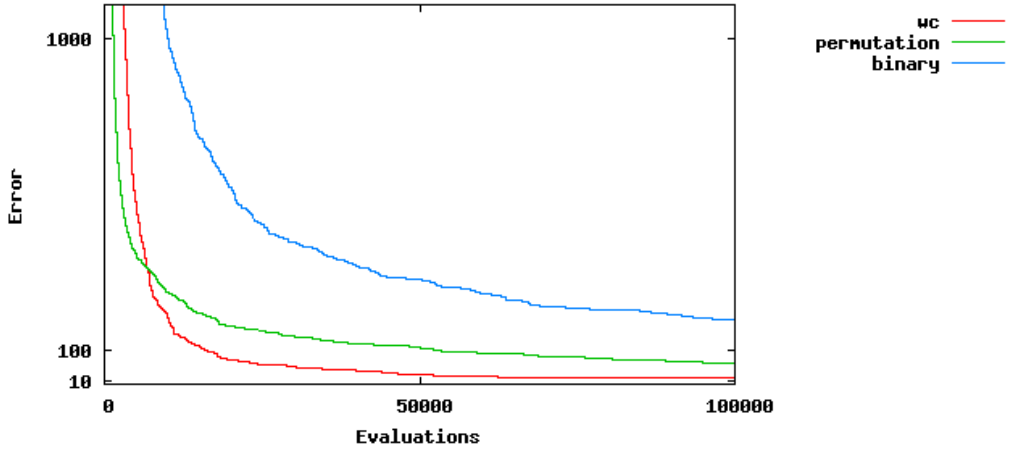

Figure 4.1 : Error over time for different representations in a stationary environment

As can be seen, the permutation representation is fastest to converge, WC is somewhat slower but then takes over, and the binary representation with penalty is

very slow, and remains worst throughout the run. The first (random) solution generated by the WC, permutation, and binary approaches has an error of approximately 6366, 7381, and 16374922, respectively. This shows that the WC representation has a higher heuristic bias than the permutation representation, while the binary approach starts with infeasible (more infeasible solutions with lower tightness ratios) and highly penalized solutions.

4.1.2 Dynamic Environment

The relative performance of the different representations in a dynamic environment is shown in Figure 4.2. In the plot, the fitness of the first individual after a change is indicated with (x) for the permutation approach and (+) for the WC approach. For further details see also Table 4.1.

Table 4.1: Average error or initial solution, the solution right before change, and the solution right after a change, ± standard error

	WC	Permutation	Binary
Initial solution	6307 ± 133	7471 ± 140	15764226 ± 418421
Before change	197 ± 10	260 ± 15	834 ± 33
After change	1482 ± 67	2201 ± 94	577226 ± 47984

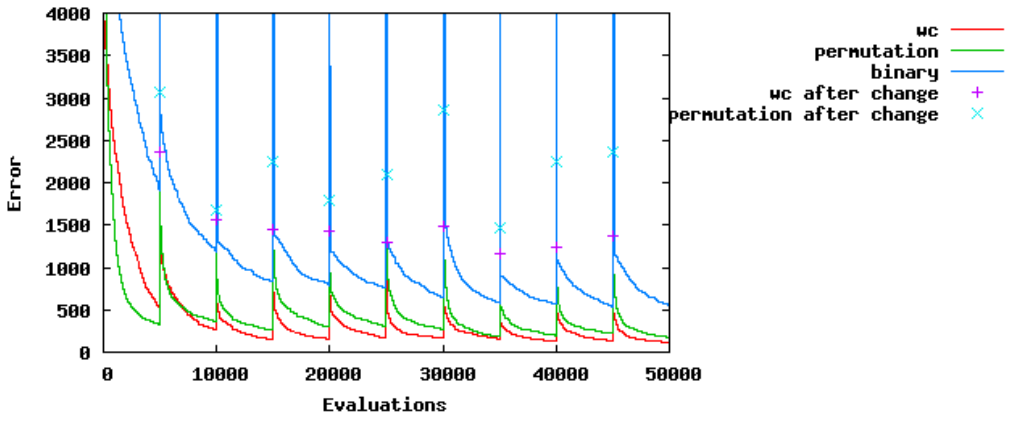

Figure 4.2 : Error over time for different representations in a dynamic environment

Several interesting observations can be made. First, as expected, there is a significant increase in error right after a change. Nevertheless, the error after a change is much smaller than the error at the beginning of the run. Compared to the first environment, the average error of the starting solution in environments 2-10 is reduced by approximately 75% for WC, 71% for permutation and 96% for the binary representation with penalty. This means that all representations benefit dramatically

from transferring solutions from one environment to the next. WC starts better than the permutation representation, and both indirect representations are much better than the binary one. The binary representation with penalty can not prevent the solutions to become infeasible, but recovers quickly. It clearly benefits most from re-using information, and it can improve its performance over several environmental periods (not only from the first to the second environment).

Second, starting from better solutions, the algorithms are able to find better solutions throughout the stationary periods. The benefit seems highest for the binary representation, while the permutation approach can improve performance only a little bit. At the end of the 10^{th} environmental period (evaluation 50000), the solution quality reached by the indirect representations is close to the error found after 50000 evaluations in a stationary environment. This means that the algorithms do not get stuck at a previously good but now inferior solution.

Third, as in the stationary case, the WC representation outperforms the permutation representation after a while and performs best overall.

4.1.3 Restart

Instead of continuing the EA run after a change, one might also re-start the EA with a new, randomized population, which is a common strategy to avoid premature convergence of the population. In this case, improving the solution quality fast would be even more important. As Figure 4.3**,** Figure 4.4**,** and Figure 4.5 show, re-initializing the population after a change is worse than simply continuing for all three representations.

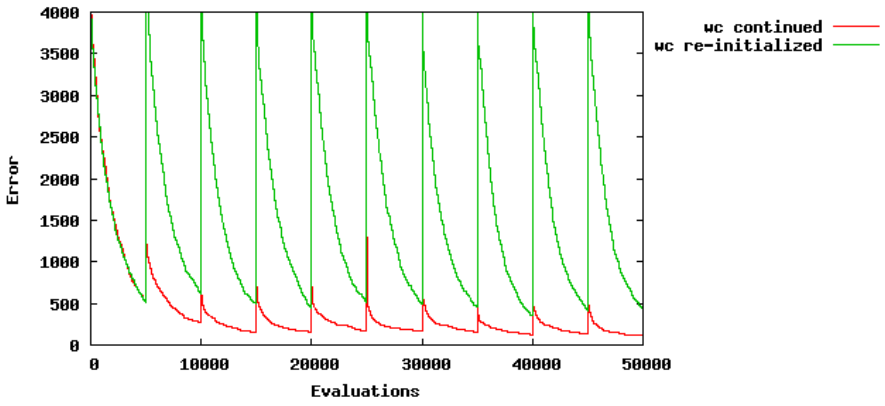

Figure 4.3 : Comparison of keeping the population or re-start after a change for the WC representation.

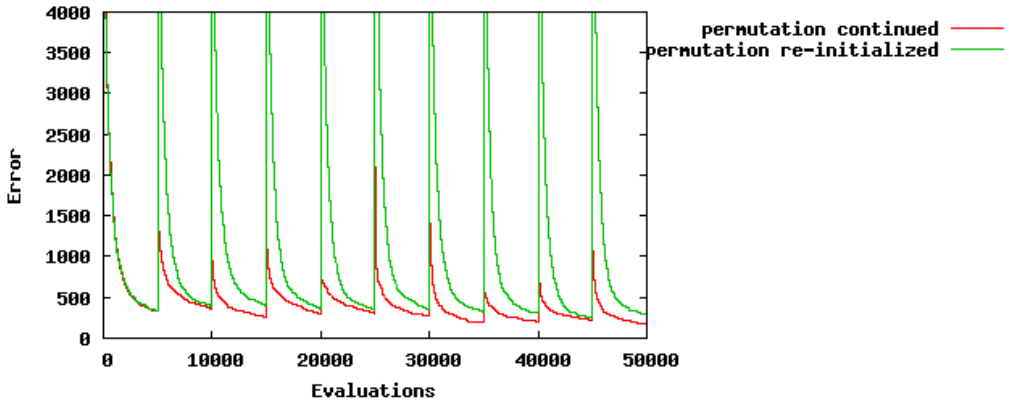

Figure 4.4 : Comparison of keeping the population or re-start after a change for the permutation representation.

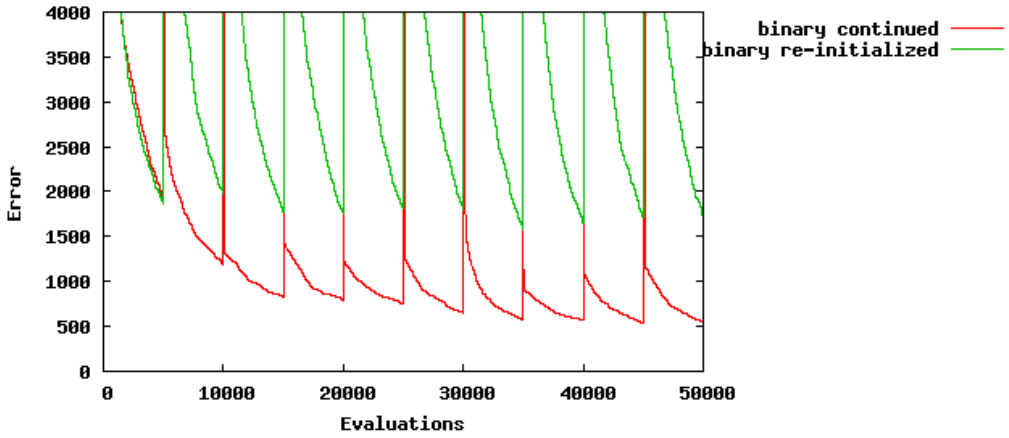

Figure 4.5 : Comparison of keeping the population or re-start after a change for the binary representation.

For the permutation representation, the difference is the smallest, for binary representation it is largest. The results suggest that premature convergence is not so much an issue in the settings considered, either because the duplicate avoidance used is sufficient to prevent premature convergence, or because the population does not converge within the 5000 evaluations per period anyway.

4.1.4 Hypermutation

Hypermutation [23] has been suggested as a compromise between complete restart and simply continuing evolution. With hypermutation, the mutation probability is increased for a few iterations after a change to re-introduce diversity. For our experiments, we tried to increase mutation in such a way that it would have similar

effects for all representations. For the binary representation, we increased the probability of mutation to $p_m = 0.05$ per gene. For WC, we increased the standard deviation for the Gaussian mutation to $\sigma = 5$. For the permutation representation, we applied insert mutation 5 times to each newly generated individual. Figure 4.6, Figure 4.7, Figure 4.8 show the performances of 3 methods with hypermutation applied through 3 generations at the beginning of each environment to diversify population.

In our tests, hypermutation had little effect except if the WC representation is used. Only for WC, hypermutation helped to speed up fitness convergence significantly in the first period, which indicates that either the mutation step size or the area for initialization have been chosen too small in the first environment.

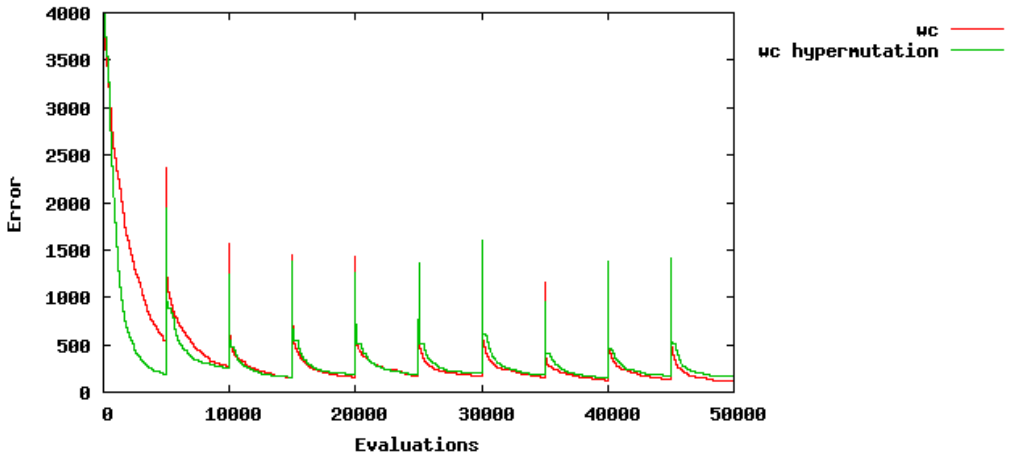

Figure 4.6 : Comparison of applying hypermutation through 3 generations for the wc representation

Neither in permutation representation nor in binary representation hypermutation has any noticable effect. We also applied hypermutation through 5 generations and again no change was observed in the performances of representations just as in the case of 3 generations hypermutation.

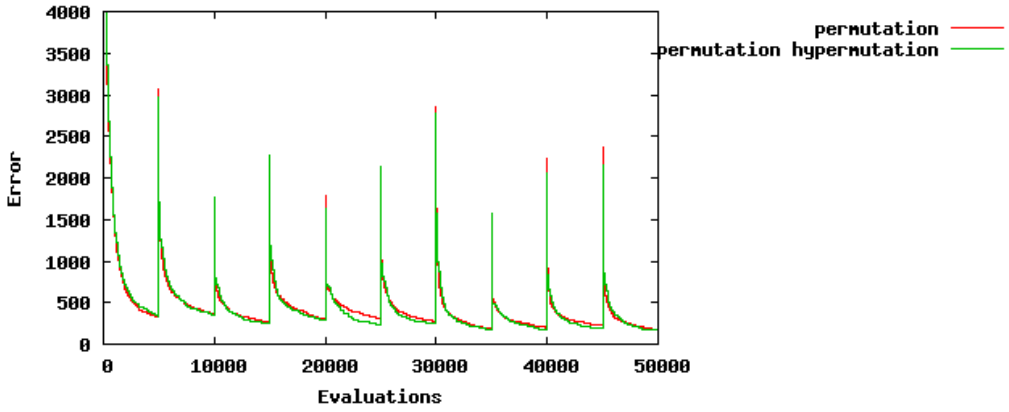

Figure 4.7 : Comparison of applying hypermutation through 3 generations for the permutation representation

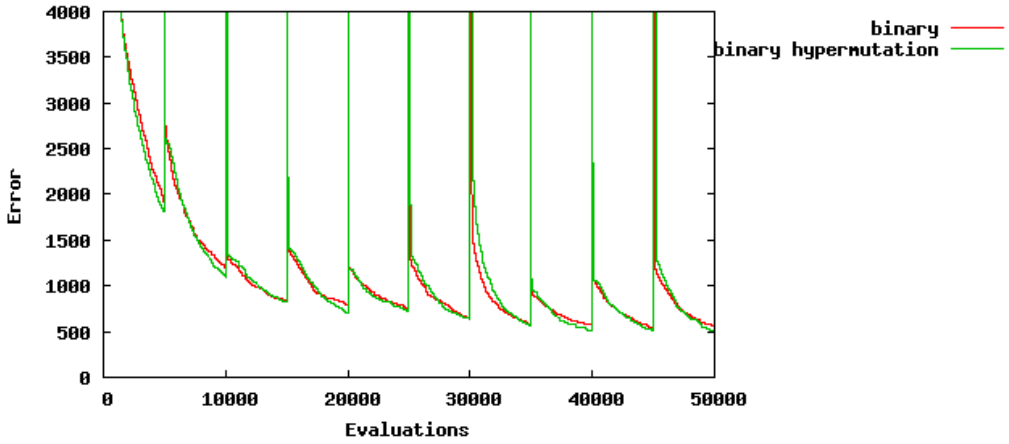

Figure 4.8 : Comparison of applying hypermutation through 3 generations for the binary representation

4.1.5 Higher Change Frequency

Obviously, the higher the change frequency, the more important it is to produce good solutions quickly, and thus the higher should be the advantage of indirect representations. Figure 4.9 shows the same performance plots as in the previous subsection, but with a change every 2000 evaluations.

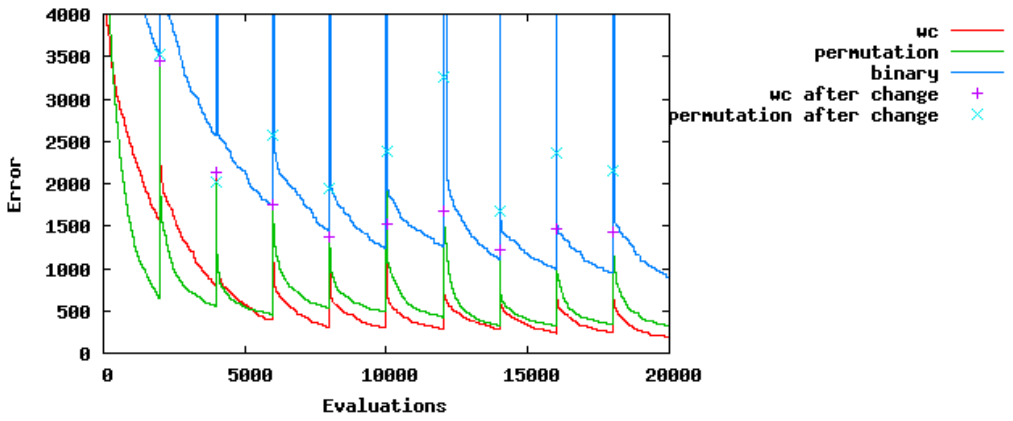

Figure 4.9 : Error over time for different representations in a dynamic environment with a change every 2000 evaluations

With the higher change frequency, the WC representation improves over the permutation representation only in the third environmental period, although according to the number of evaluations, the switch is actually earlier than in the previous case with lower change frequency. The binary representation keeps improving over all 10 environmental periods.

4.1.6 Effect of Change Severity

The aim of this experiment is to see the effect of change severity. To implement a more severe change, in Equation (2.3), we set the standard deviation of the normally distributed random variable used for the changes to $\sigma_p = \sigma_r = \sigma_c = 0.1$ and each profit p_j, resource consumption r_{ij} and constraint c_i is restricted to an interval as determined in Equation (2.4) with $lb_p = lb_r = lb_c = 0.5$ and $ub_p = ub_r = ub_c = 1.5$.

The results for this more severe environment, shown in Figure 4.10, looks very similar to the standard case we have looked at in the above subsections. The error immediately after a change is significantly higher, but all algorithms adapt rather quickly and the solutions found later on are comparable to those in the standard case. As the analysis of the offline error below will show, in particular the binary encoding suffers from the increased severity. The indirect encodings are barely affected.

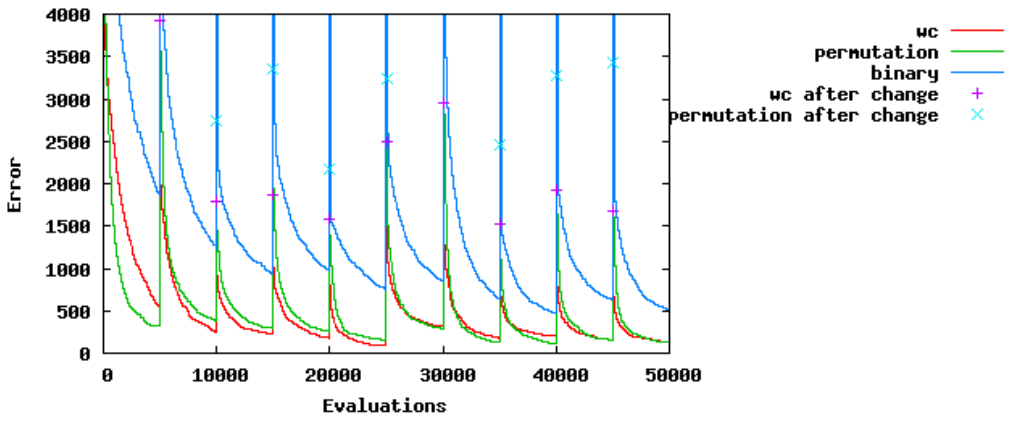

Figure 4.10 : Error over time for different representations in a highly severe dynamic environment

4.2 Dynamic Traveling Salesman Problem

For TSP tests**,** we used an EA with a population size of 25, crossover probability of 1.0, and mutation probability of 0.1 per gene. We use Concorde [24] for calculating the true optimum for each environment. As a note, in our code, we used real numbers for calculation of fitness values. However, Concorde uses rounded distances between cities. As a result, for the same solution, Concorde and our code generate different fitness values. To solve this problem, while finding the optimum solutions for all environments with Concorde, we blew up the problem instances. We multiplied all coordinates by 10000, to decrease the rounding effect. Concorde finds the optimal solution then, although we still need to re-calculate the exact difference without rounding.

4.2.1 Relative Performance in Stationary Environments

Figure 4.11 shows performances of both methods, perturbed coordinates and path representation in a stationary environment for a 50 cities problem. Methods are run for 10000 fitness evaluations.

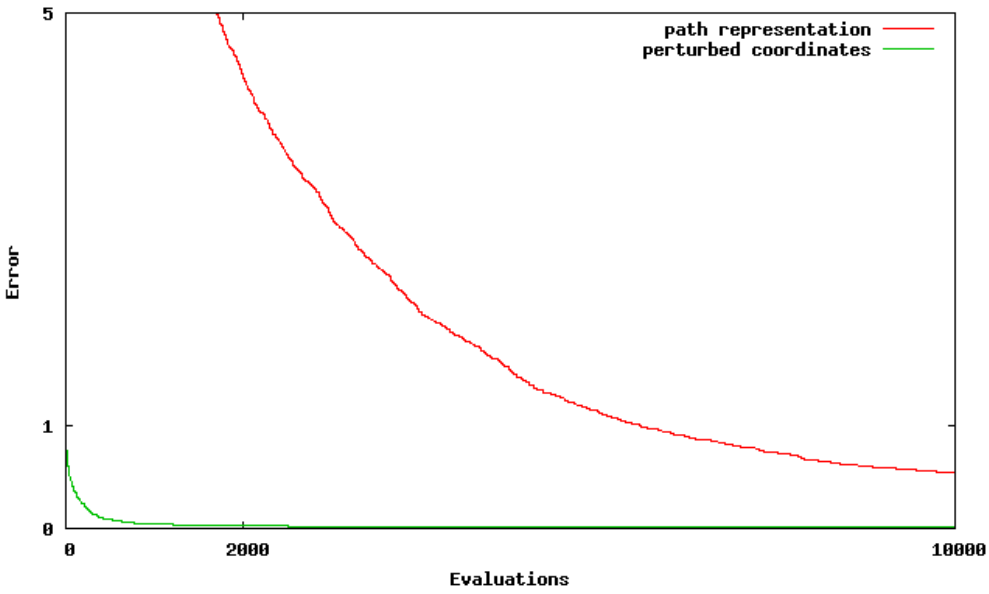

Figure 4.11 : Error over time for different representations in a stationary environment for a 50 cities problem

Perturbed coordinates method converges faster and also performs better than path representation throughout the run. It generally finds the optimum solution (solution with average error 0.025) after about 2000 fitness evaluations. However, path representation finds a good solution (solution with average error 0.54), rarely the optimum one after 10000 fitness evaluations. Figure 4.12 shows the same case for a 100 cities problem.

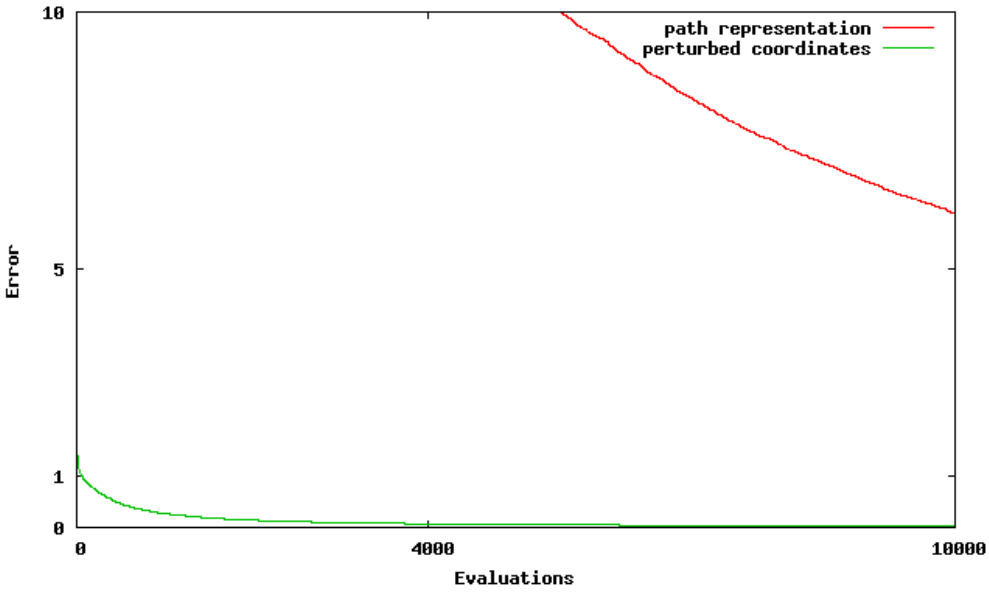

Figure 4.12 : Error over time for different representations in a stationary environment for a 100 cities problem

As the number of cities increases, problems get harder. Perturbed coordinates again generally finds the optimum (solution with average error 0.677), but after about 4000 fitness evaluations. As the number of cities doubled, fitness evaluations required to find the optimum is doubled for the method. Path representation is more influenced in a bad manner. The quality of the solution found at the end of the run gets worse.

We also run the path representation for 100000 fitness evaluations to see if it approaches to the solution quality of perturbed coordinates. Figure 4.13 and Figure 4.14 show the performance of the method for 50 cities and 100 cities problems respectively. For both problems, method converges and does not improve after some time. Last points of improvement are nearly 40000 and 85000 fitness evaluations for 50 and 100 cities problems respectively. For 50 cities problem, after 10000 fitness evaluations error of the solution does not decrease much.

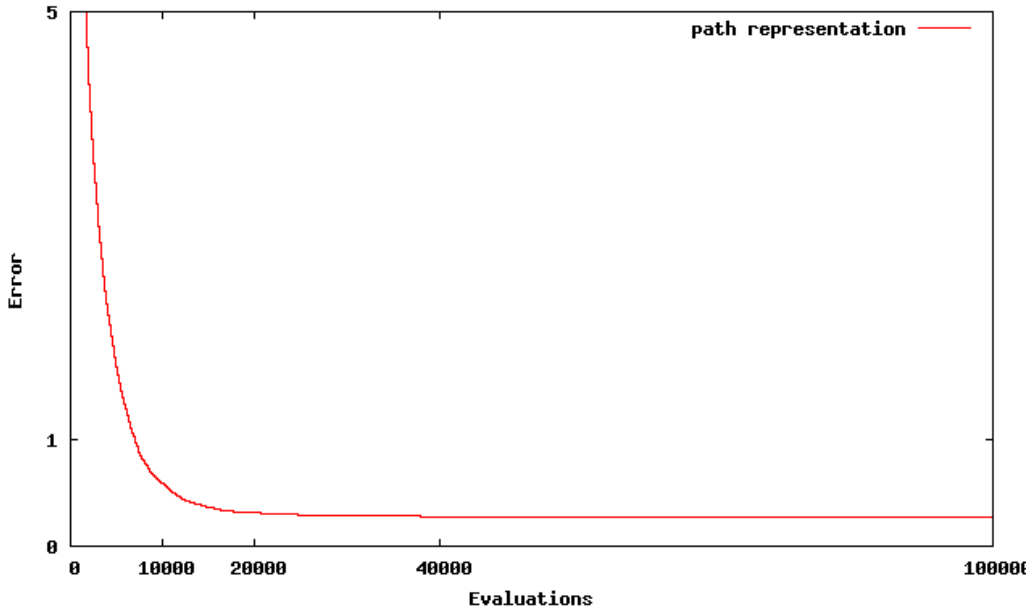

Figure 4.13 : Error over time for path representations in a stationary environment for a 50 cities problem, run for 100000 fitness evaluations

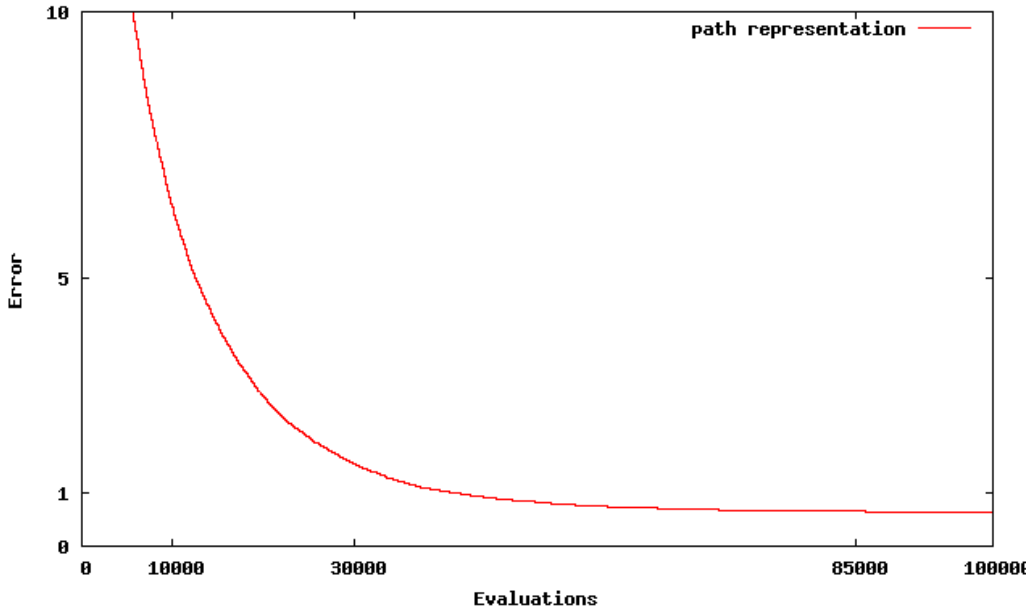

Figure 4.14 : Error over time for path representations in a stationary environment for a 100 cities problem, run for 100000 fitness evaluations

In Figure 4.14, for the 100 cities problem, the algorithm keeps improving till about 30000 fitness evaluations. Although the method is run for 100000 fitness evaluations for both problems, it can not reach the error of perturbed coordinates at 10000 fitness evaluations.

4.2.2 Dynamic Environment

Figure 4.15 shows the relative performance of the representations in a dynamic environment for a 50 cities problem. Methods are run for 20000 fitness evaluations, 10 changes at each 2000 fitness evaluations.

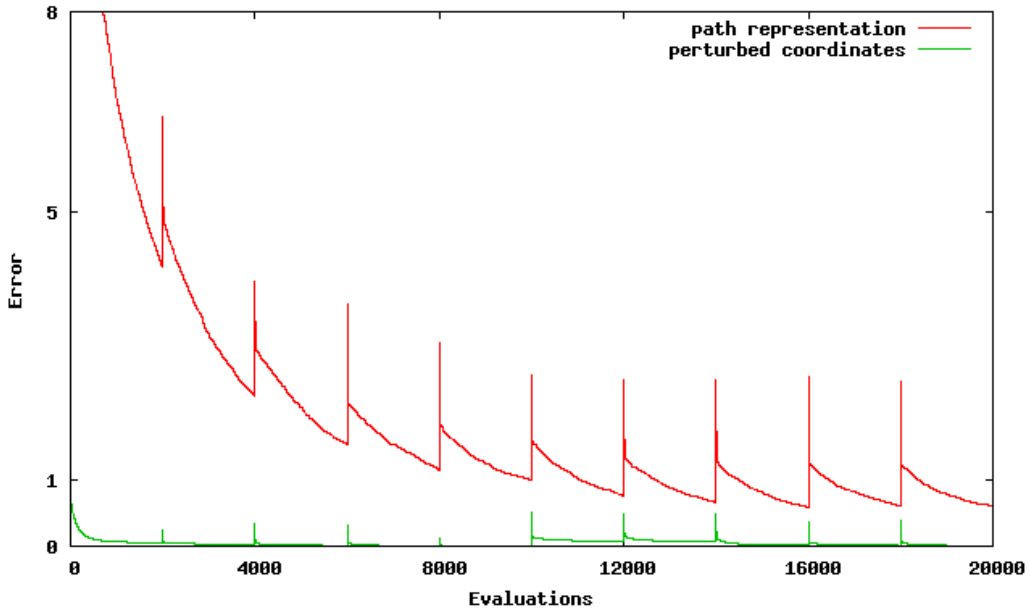

Figure 4.15 : Error over time for different representations in a dynamic environment for a 50 cities problem

There is an increase in error at change points, which is the result of change in the environment and also observed in the dynamic environment of MKP. For path representation, the error after a change is much smaller than the error at the beginning of the run, which means that the method benefits from transferring solutions from one environment to the next. For perturbed coordinates, the problem is not hard and the method starts from good solutions at each environment, thus this error decrease at change points can not be observed. However that does not mean that the method does not benefit from transferring solutions. Let us examine the performance of perturbed coordinates in two steps. First, performance between 0 and 10000 fitness evaluations, then the remaining part. In the stationary case, the method finds the optimum after 2000 fitness evaluations. In the first environment again it nearly takes 2000 fitness evaluations to find the optimum. But in the next 4 environments, fitness evaluations required to find the optimum significantly decreases. However, 2 cities are changed at each environment, the impact of the change at 10000^{th} fitness evaluation is somehow strong and made the solutions at hand non-effective. But again in the following 4 environments, fitness evaluations required to find optimum keep decreasing.

Table 4.2 shows the number of fitness evaluations required to find a solution, having the same quality as the solution found in the first environment. At the end of 2000

fitness evaluations in the first environment, the error (average of 50 runs) of the solution found by perturbed coordinates method is 0.045. Between 2000 and 10000 fitness evaluations, at each environment, the number of fitness evaluations required to reach an error of 0.045 decreases. At the beginning of the 6^{th} environment, the starting solutions become ineffective and the EA starts with a bad population, relative to previous environments. Thus, the EA can not find a solution with error of 0.045 for 2 environments. Then it recovers, and again finds solution with error 0.045 in decreasing time.

Table 4.2 The number of fitness evaluations required to find the solution, having same quality with the solution found in the first environment, and error of the best solution found at the end of each interval

Interval	Number of fitness evaluations to reach the error of 0.045	Error of the best solution found at the end of each interval
0-2000	2000	0.045
2000-4000	158	0.023
4000-6000	34	0.010
6000-8000	23	0.005
8000-10000	4	0.007
10000-12000	Na	0.064
12000-14000	Na	0.069
14000-16000	327	0.015
16000-18000	115	0.011
18000-20000	105	0.009

Perturbed coordinates method performs better than path representation in each environment as in stationary case.

Same observations can be made for the Figure 4.16, showing methods for a 100 cities problem. As the problem gets harder, the error at the start of run in the first environment is increased. In this case, a decrease in error after a change according to the error at the beginning of the run is now slightly seen.

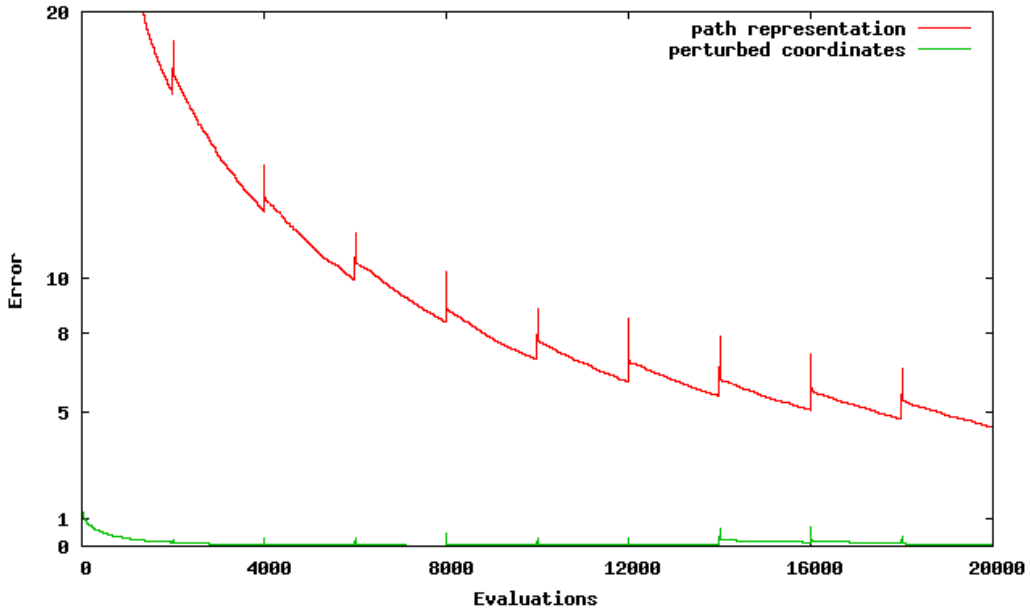

Figure 4.16 : Error over time for different representations in a dynamic environment for a 100 cities problem

4.2.3 Restart

Figure 4.17**,** Figure 4.18**,** Figure 4.19**,** and Figure 4.20 show the performances of both representations for both problems when the EA is re-started. For all cases, continuing with the population at changes is better than re-generating a new population.

In Figure 4.17 and Figure 4.18**,** it can be seen that **the** perturbed coordinates method, benefits from its fast convergence property and finds the optimum, although the population is re-generated at each change.

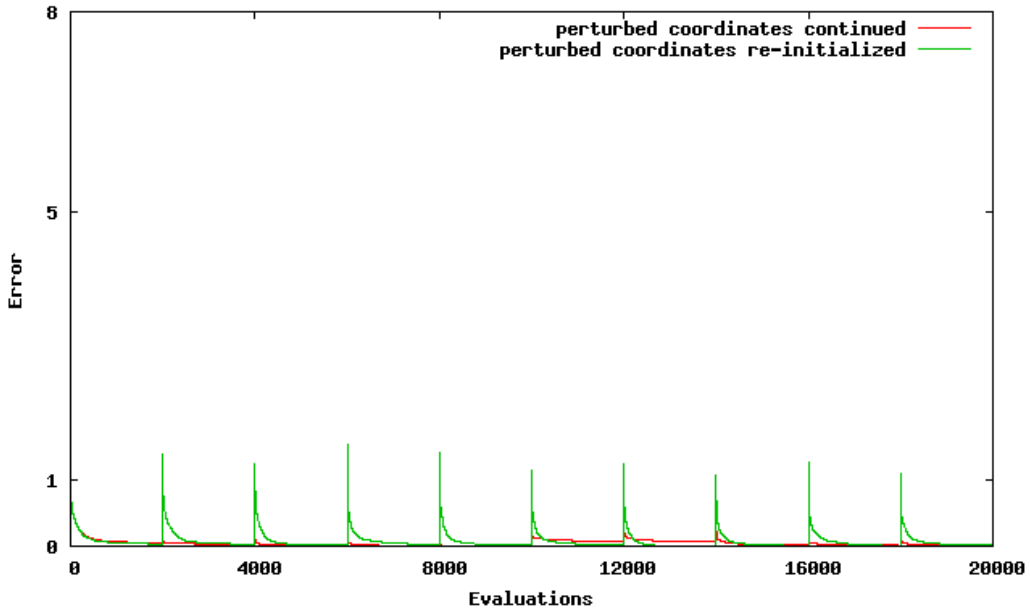

Figure 4.17 : Comparison of keeping the population or re-start after a change for the Valenzuela representation for a 50 cities problem.

In Figure 4.18, for problem with 100 cities, it takes 2000 fitness evaluations to find nearly optimum solution in each environment as in the stationary case.

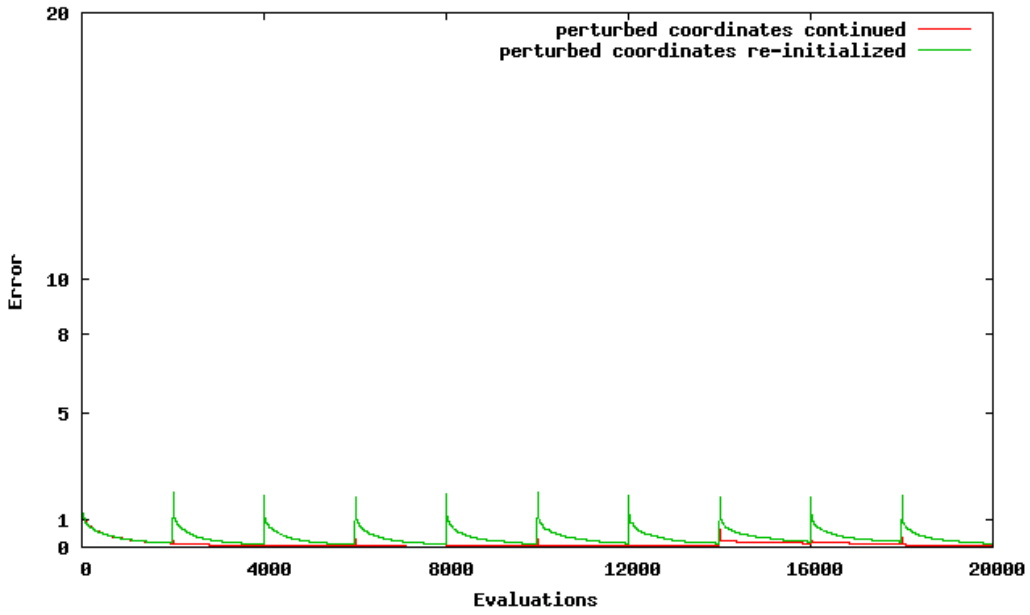

Figure 4.18 : Comparison of keeping the population or re-start after a change for the Valenzuela representation for a 100 cities problem.

Path representation, also converges fast in 2000 fitness evaluations, but as being a direct representation and not using a heuristic starting population is not good enough to let method find optimum in such a short time.

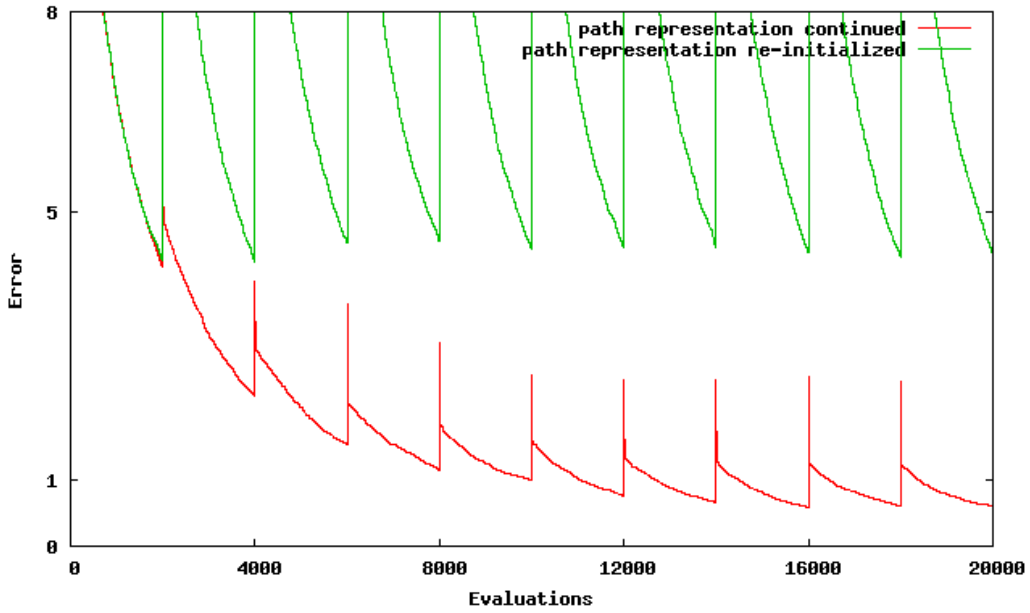

Figure 4.19 : Comparison of keeping the population or re-start after a change for the Permutation representation for a 50 cities problem.

For the harder problem, as the starting population gets worse, performance of the representation suffers.

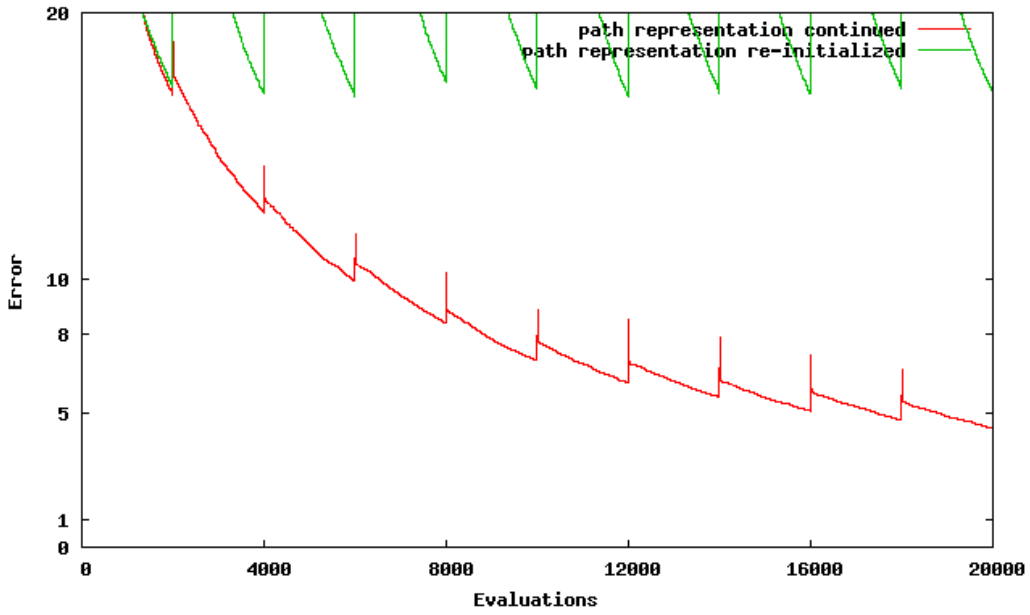

Figure 4.20 : Comparison of keeping the population or re-start after a change for the Permutation representation for a 100 cities problem.

4.2.4 Higher Change Frequency

For the path representation method, for 50 cities problem, 2000 fitness evaluations are not sufficient to find a good solution at preliminary environments. The method keeps improving for 3 environments in Figure 4.15, EA with change at each 2000 fitness evaluations. When the environment period is decreased to 1000 fitness evaluations, in Figure 4.21, it can be seen that method improvement duration increases to 6 environments.

For perturbed coordinates, it seems in Figure 4.15 that the method converges fast and finds the optimum earlier than 2000 fitness evaluations in most environments. Thus, higher change frequency does not have a strong effect on the method. It can be easily seen in Figure 4.21 that in most environments, the error of the perturbed coordinates converges to 0 before 1000 fitness evaluations.

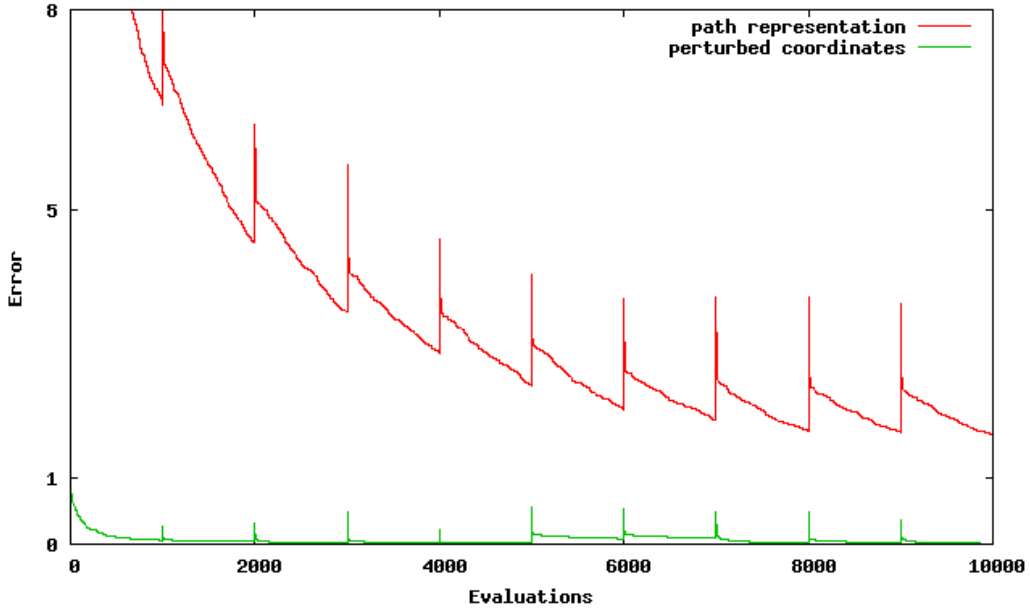

Figure 4.21 : Error over time for different representations in a dynamic environment with a change every 1000 evaluations for a 50 cities problem.

Same observations can be made for the harder problem.

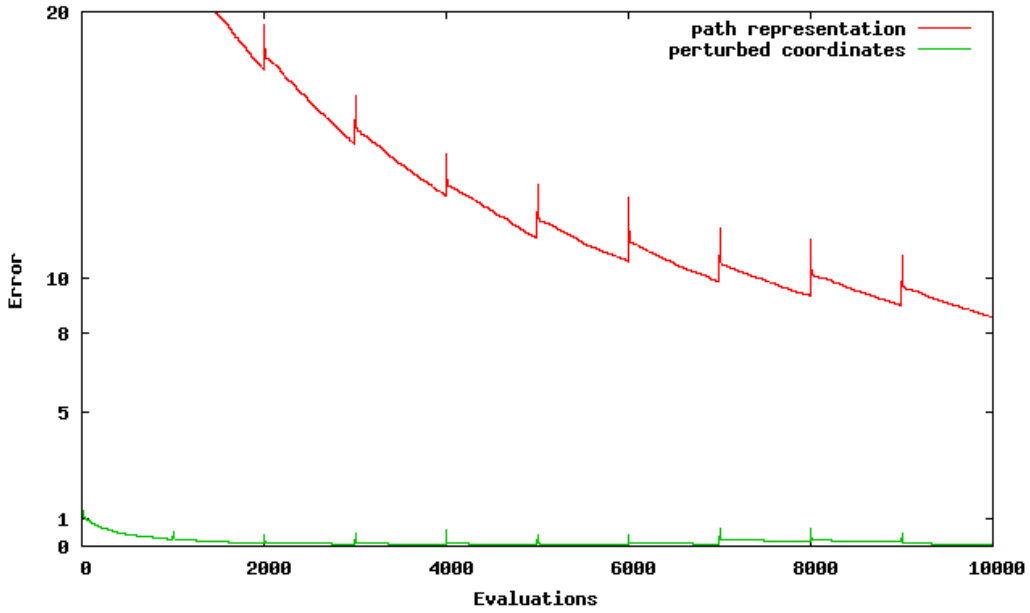

Figure 4.22 : Error over time for different representations in a dynamic environment with a change every 1000 evaluations for a 100 cities problem.

4.2.5 Effect of Change Severity

To implement a more severe change, we increased the number of cities to be changed for each environment. We take 2 cities change as base, and try 4 and 6 cities change to see the effect of change severity. For both number of cities and for both city changes, Figure 4.23, Figure 4.24, Figure 4.25, Figure 4.26 show performances of representations.

As expected and seen, the error right after a change is significantly higher than the increase in error in the base change. Solutions found at the end of 2000 fitness evaluations in higher change severity get worse.

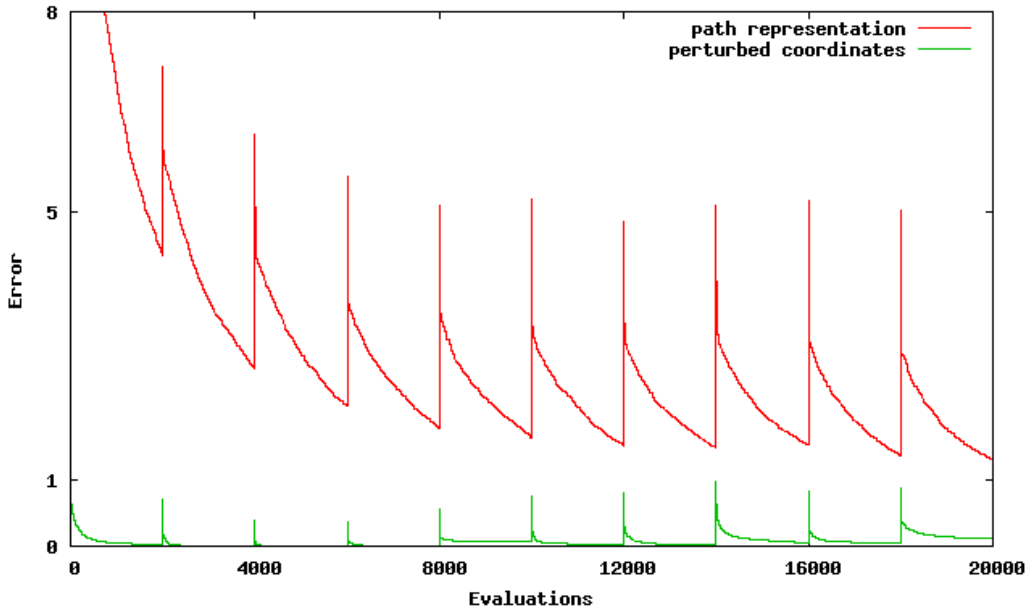

Figure 4.23 : Error over time for different representations in a highly severe dynamic environment for 50 cities problem with 4 cities change at each environment

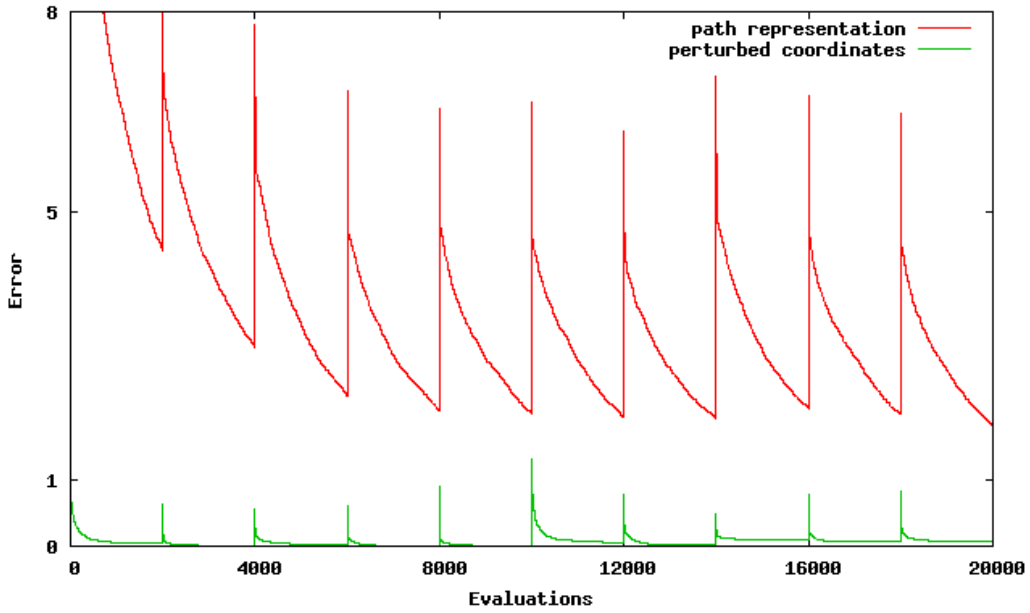

Figure 4.24 : Error over time for different representations in a highly severe dynamic environment for 50 cities problem with 6 cities change at each environment

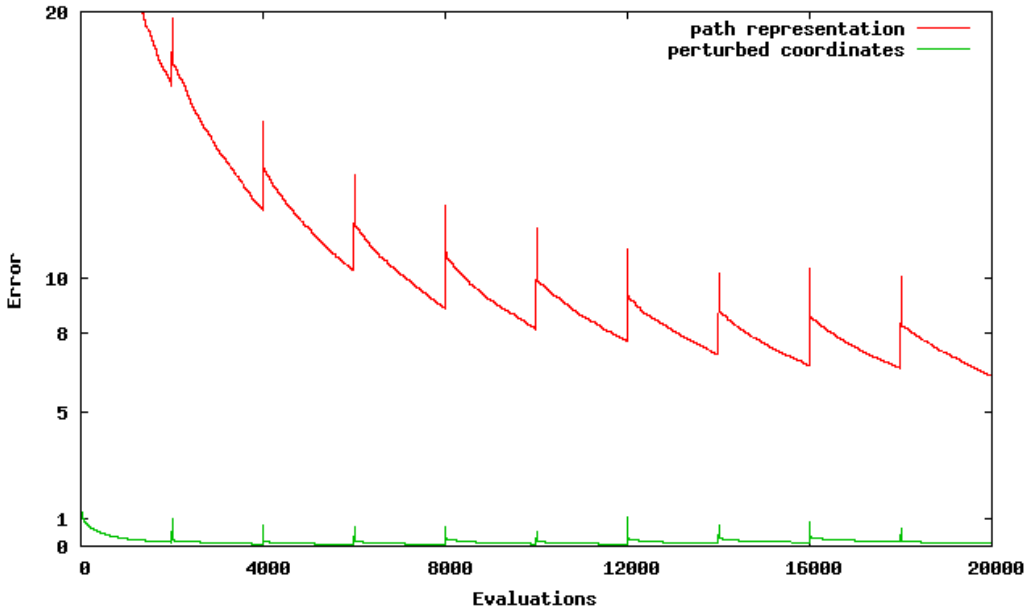

Figure 4.25 : Error over time for different representations in a highly severe dynamic environment for 100 cities problem with 4 cities change at each environment

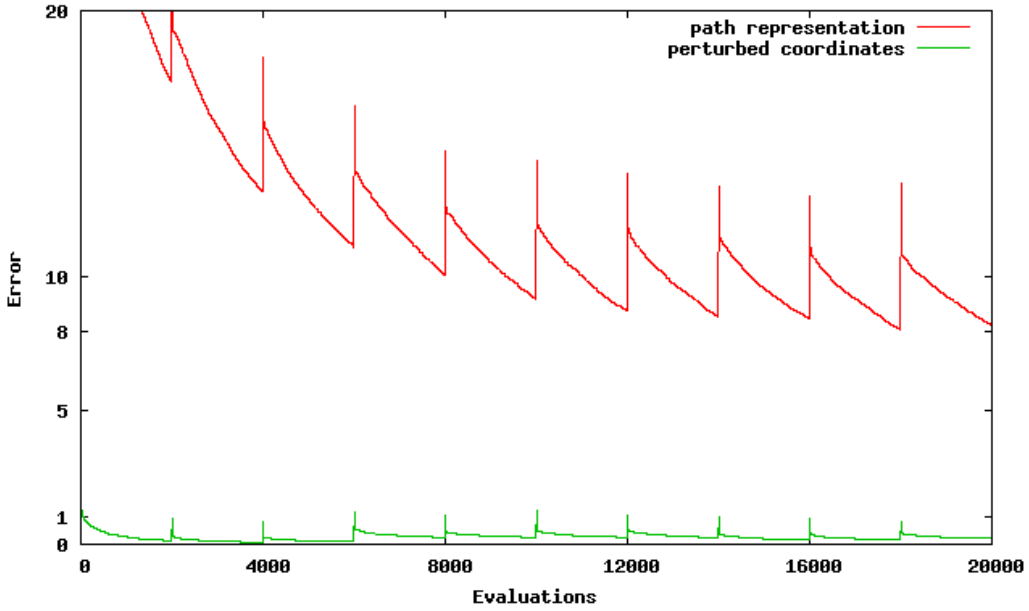

Figure 4.26 : Error over time for different representations in a highly severe dynamic environment for 100 cities problem with 6 cities change at each environment

4.3 Summary of Results

The results have demonstrated that the representation can have a tremendous effect on an EA's performance in dynamic environments.

For dMKP, while the permutation representation was fastest to converge in terms of solution quality, the WC representation was best in coping with the dynamics of the problem. The binary representation with penalty performed extremely poor, as it improves slowly and is not even able to maintain feasibility after a change.

In a stationary environment, what usually counts is the best solution found at the end. After 20000 evaluations, the obtained errors of the approaches are 73 for WC, 170 for permutation, and 570 for binary reprsentation with penalty. However, in a dynamic environment usually the optimization quality over time is important. Table 4.3 summarizes all the results by looking at the offline error, i.e. the average error over evaluations 5000-20000. This interval has been chosen because it was covered in all experiments, and removes the influence from the initial "warm-up" phase.

Table 4.3: Offline error of different representations in different environments for dMKP, Evaluations 5000-20000

	WC	Permutation	Binary
Stationary	179.1	248.1	947.8
Restart	1581.6	1115.2	625746.0
Dynamic base	342.2	470.4	1823.0
High frequency	397.1	561.5	3445.7
High severity	456.4	621.6	14756.9

In the stationary case, WC representation performs best, with permutation a close second (+39%), and binary representation with more than four times the offline error of the two indirect representations. In a dynamic environment, when the algorithm is restarted after every change, the permutation representation benefits from its fast fitness convergence properties and performs best, while the binary representation improves so slowly and generates so many infeasible solutions that it is basically unusable. If the algorithms are allowed to keep the individuals after a change, they all work much better than restart. With increasing change frequency or severity, the performance of all approaches suffers somewhat, but the gap between the best-performing WC representation and the direct binary representation increases from 532% in the dynamic baseline scenario to 867% in the high frequency scenario and 3233% in the high severity scenario.

For dTSP, starting with a good population and with its fast convergence properties, the perturbed coordinates method performes best throughout all environments in all cases. For this reason, it was hard for us to show that indirect methods are more

suitable for dynamic problems also for dTSP. Decrease in time, to find the same quality solution with respect to the solution quality found at the begining of the run when EA is not restarted, is the point that shows we are right in our hypothesis.

Table 4.4 and Table 4.5 summarize all the results as in Table 4.3, but now for dTSP with different number of cities, by looking at the offline error, i.e. the average error over evaluations 2000-10000. This interval has been chosen as it is valid for all experiments, and skips the initial "warm-up" phase. In both tables, the data shown in the first column in parenthesis gives information about the number of cities, number of cities changed to form a new problem, and the number fitness evaluations for each environment. For example, (50_2_2000) means 50 cities problem, change 2 cities at each 2000 fitness evaluations. In the last column, data in parenthesises show the absolute value and percentage of difference between methods.

Table 4.4 : Offline error of different representations in different environments for 50 cities dTSP, Evaluations 2000-10000. Absolute value and percentage of difference between methods are given in parenthesis respectively.

	Perturbed coordinates (indirect)	Path representation (direct)
Stationary (50_2_10000)	0.016	1.544 (+1.53) (+9550%)
Restart (50_2_2000)	0.080	7.980 (+7.90) (+9875%)
Dynamic base (50_2_2000)	0.018	2.081 (+2.06) (+11461%)
High frequency (50_2_1000)	0.050	2.661 (+2.61) (+5222%)
High severity (50_4_2000)	0.025	2.932 (+2.91) (+11628%)
High severity (50_6_2000)	0.024	3.498 (+3.47) (+14475%)

In all cases perturbed coordinates performs much more better than path representation approach. In stationary case, offline error of the path representation is 9550% more than the offline error of the perturbed coordinates. In dynamic environment with change frequency 2000, when the EA is restarted at each change, this ratio remains nearly the same with being 9875%, showing that both methods are similar relative to each other as in the stationary case, but now for 10 environments. Again in dynamic environments, but when the EA is not restarted, a decrease in offline errors for both methods is seen, showing that both methods benefit from reusing previous populations. Although a higher frequency affects perturbed

coordinates relatively more than path representation, perturbed coordinates still finds the optimum in most cases. While the percentage of difference in offline error decreases to 5222%, the absolute difference increases, which is the indicator to be considered for this comparison. For high severity, perturbed coordinates is not affected much, nevertheless the impact increases as the number of cities increases for path representation.

Table 4.5 shows the same information for a 100 cities problem. As the problem gets harder, balance changes between methods, and also all values in the table increases as expected.

Table 4.5 : Offline error of different representations in different environments for 100 cities dTSP, Evaluations 2000-10000. Absolute value and percentage of difference between methods are given in parenthesis respectively.

	Perturbed coordinates (indirect)	Path representation (direct)
Stationary (100_2_10000)	0.058	10.016 (+9.96) (+17169%)
Restart (100_2_2000)	0.319	23.799 (+23.48) (+7361%)
Dynamic base (100_2_2000)	0.054	10.884 (+10.8) (+20056%)
High frequency (100_2_1000)	0.117	11.787 (+11.67) (+9974%)
High severity (100_4_2000)	0.122	11.664 (+11.54) (+9461%)
High severity (100_6_2000)	0.237	12.886 (+12.65) (+5337%)

According to increasing number of cities, path representation starts with a worse solution group and can not converge to 0 swiftly, hence the difference between two methods significantly widens (+17169%) in the stationary environment. For 100 cities problem, it gets harder also for perturbed coordinates to find the optimum so the percentage of the difference between two methods decreases relative to 50 cities problem, in the restart case. In dynamic case, when same genotypes are used throughout all environments, again both methods benefit from the previous knowledge, and decrease in offline error values according to restart case is observed.

For higher frequency, again increase in absolute difference and decrease in percentage of the difference according to dynamic base are observed as in 50 cities problem. As the number of cities increases, perturbed coordinates method can no

longer handle the problem with higher change severity. For this reason, the percentage of the difference between two methods decreases relative to the 50 cities case.

5 CONCLUSION

In the literature, lots of studies are reported which analyze the effect of representations in stationary environments for EAs. Nevertheless, the role of representations in dynamic environments has been ignored so far. In this thesis, we empirically analyzed the effects of different representations in dynamic environments.

Direct representations can be considered directly as a solution to the problem. In indirect representations, as the search is done in a different space, a mapping is required to get the solution. Direct representations generally search in the entire search space, so the method has to deal with the infeasible solutions through evolution. In indirect representations, search is done in genetic search space and generally they all map to feasible solutions of the search space. For this reason, indirect representations do not have a problem of infeasible solutions. In this thesis, we empirically analyzed and compared direct and indirect representations for two generally known optimization problems, multi-dimensional knapsack problem and traveling salesman problem.

Weight coding for multi-dimensional knapsack problem and perturbed coordinates for traveling salesman problem are the two indirect representations we selected. In both approaches, each candidate solution first creates a slightly changed problem, by changing some values of the original problem. Then they all find a solution to the changed problem using a fast heuristic. In indirect representations, at each change point, the existing population is adapted to the new problem by mapping solutions' components of genetic search space to the solution space.

In experiments, for both problems, we first analyzed the performances of representations in stationary environments. Then, by using the same population (same genotypes) throughout all evolution, we run the EAs in dynamic environments. For indirect representations to avoid duplicates, at the beginning of each environment we regenerated all genotypes which map to a phenotype (actual solution) already in the population. We also run the EAs by restarting, to see the

effect of using a new group of genotypes at each environment. At the end of these tests, it is obviously seen that indirect representations work much better not only in stationary environments but also in dynamic environments. We tried hypermutation as a midway between restart and continuing with the same population. In our tests, duplicate avoidance prevent the population from converging and hypermutation had little effect. We also analyzed the effect of higher change frequency and higher change severity. Direct representations are affected by higher change frequency and higher change severity more than indirect representations obviously.

Our results indicate that indirect representations are particularly suitable for dynamic problems, because they implicitly provide a heuristic adaptation mechanism that improves the current solutions after a change. In addition, we saw that the choice of representation in dynamic environments is even more important than in static environments. For this reason, one should be selective while deciding representation to be used in a dynamic environment and one should prefer adaptive representations.

As future work, the effect of the representations on dynamic problems can be analyzed on different problems and also on the same problems with additional representations. Besides, the performance of representations should be compared with respect to running time also, an aspect that has been ignored in this thesis.

REFERENCES

[1] **Branke, J.,** 2001. Evolutionary Optimization in Dynamic Environments, Kluwer.

[2] **Jin, Y. and Branke J.**, 2005. Evolutionary optimization in uncertain environments – a survey, *IEEE Transactions on Evolutionary Computation*, **9(3)**, 303-317.

[3] **Morrison , R.,** 2004. Designing Evolutionary Algorithms for Dynamic Environments, Springer.

[4] **Weicker, K.,** 2003. Evolutionary Algorithms and Dynamic Optimization Problems, Der Andere Verlag,

[5] **Raidl, G.R. and Gottlieb, J.,** 2005. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem, *Evolutionary Computation*, **13(4)**, 441-75.

[6] **Rothlauf, F.,** 2002. Representations for Genetic and Evolutionary Algorithms, Physica.

[7] **Branke, J., Salihoglu, E. and Uyar, S.**, 2005. Towards an analysis of dynamic environments, *In Genetic and Evolutionary Computation Conference*, 1433-1439.

[8] **Gottlieb, J.**, 1999. Evolutionary Algorithms for Combinatorial Optimization Problems, *Phd*, Technical University Clausthal, Germany.

[9] **Raidl, G.R. and Gottlieb J.**, 2000. The effects of locality on the dynamics of decoder-based evolutionary search, *In Genetic and Evolutionary Computation Conference*, 283-290.

[10] **Raidl, G.R.**, 1999. Weight-codings in a genetic algorithm for the multiconstraint knapsack problem, *In Congress on Evolutionary Computation*, 596-603.

[11] **Bradwell, R.**, **Williams, L.P. and Valenzuela, C.L.,** 1997**.** Breeding Perturbed City Coordinates and Fooling Travelling Salesman Heuristic Algorithms, *Third International Conference on Artificial Neural Networks and Genetic Algorithms,* 241-249.

[12] **Valenzuela, C.L., Jones A.J.,** 1993. Evolutionary Divide and Conquer (I): a novel genetic approach to the TSP. *Evolutionary Computation,* **1(4)**, 313-333.

[13] **Julstrom B.A.,** 1998**.** Comparing Decoding Algorithms in a Weight-Coded GA for TSP. *In ACM Symposium on Applied Computing,* 313-317.

[14] **Julstrom B.A.,** 1999**.** Coding TSP Tours as Permutations via an Insertion Heuristic. *In ACM Symposium on Applied Computing,* 297-301.

[15] **Kellerer, H., Pferschy, U. and Pisinger, D.**, 2004. Knapsack Problems, Springer.

[16] **Beasley, J.E.,** Or-library. online, http://www.brunel.ac.uk/depts/ma/ research /jeb/orlib/mknapinfo.html.

[17] **Gutin, G. and Punnen, A.P.**, 2002. Traveling Salesman Problem and Its Variations, Kluwer.

[18] **Gottlieb, J.**, 2001. On the feasibility problem of penalty-based evolutionary algorithms for knapsack problems, *In E. J. W. Boers et al. , editors*, *Applications of Evolutionary Computing,* volume 2037 of Lecture Notes in Computer Science, pp. 50-60. Springer.

[19] **Gottlieb, J.**, 2000. Permutation-based evolutionary algorithms for multidimensional knapsack problems, *In ACM Symposium on Applied Computing*, **1**, 408-414.

[20] **Pirkul, H.**, 1987. A heuristic solution procedure for the multiconstraint zero-one knapsack problem, *Naval Research Logistics*, **34,** 161-172.

[21] **Valenzuela, C.L. and Williams, L.P.**, 1997. Improving Simple Heuristic Algorithms for the Traveling Salesman Problem using a Genetic Algorithm, *Proceedings of the Seventh International Conference on Genetic Algorithms*, 458 – 464.

[22] GLPK. GNU linear programming kit. online, http://www.gnu.org/ software/ glpk/glpk.html.

[23] **Cobb, H.G.**, 1990. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments, *Technical Report AIC-90-001*, Naval Research Laboratory, Washington, USA.

[24] Concorde. TSP solver. online, http://www.tsp.gatech.edu/concorde/index. html

[25] **Branke, J., Orbayi, M. and Uyar, S.**, 2006. The Role of Representations in Dynamic Knapsack Problems, *Applications of Evolutionary Computation,* Springer, to appear.

CV

She was born in Istanbul in 1980. She graduated from Istanbul Technical University in 2003 as a Computer Engineer. She has published two papers as a result of her studies both for undergraduate and graduate thesis. This study is her graduate thesis.