MOBILE ROBOTS PATH PLANNING OPTIMIZATION IN STATIC AND DYNAMIC ENVIRONMENTS

A Thesis

Presented to

The Faculty of Graduate Studies

of

The University of Guelph

by

AHMED ELSHAMLI

In partial fulfilment of requirements

for the degree of

Master of Science

August, 2004

ABSTRACT

MOBILE ROBOTS PATH PLANNING OPTIMIZATION IN STATIC AND DYNAMIC ENVIRONMENTS

Ahmed Elshamli
University of Guelph, 2004

Advisor:
Professor: Hussein A. Abdullah
Professor: Shawki Areibi

Path planning for mobile robots is a complex problem. The solution should not only guarantee a collision-free path with minimum traveling distance, but also provide a smooth and clear path. In this dissertation, a Genetic Algorithm Planner (GAP) is proposed for solving the path planning problem in static and dynamic mobile robot environments. The GAP is based on a variable-length representation. A generic fitness function is used to combine the objectives of the problem. Different evolutionary operators are applied some are random-based, and others use problem-specific domain knowledge. Various techniques are investigated to ensure that the GAP is appropriate for dynamic environments.

To further increase the efficiency of the GAP, an Island-based GA (IGA) is developed on a ring topology and Message Passing Interface (MPI) library is utilized to implement the IGA.

A new Local Search (LS) is also developed in this thesis and different approaches are examined for combining the LS algorithm with the GAP to obtain superior solutions.

Acknowledgments

The foremost one to be thanked is *Allah* (God). I thank God for the help and guidance and ask His forgiveness. Then, I extend my deepest gratitude to all people who have shared and offered me their knowledge, help, support, experience, care, and prayers. At the top of the list are my parents, the greatest and most important people in my life. "Dad and Mom" I ask *Allah* to forgive you, save you, and reward you with the best for your abundant work, amen. Second is my wife, "I thank you for your patience and support". Third are my other family members especially my brothers, sister and uncle for their encouragement.

I want to express my gratitude to my advisors, Dr. Hussein Abdullah and Dr. Shawki Areibi. The opportunity to work with them has been extremely rewarding. Throughout my term as a graduate student, they have been a source of trusted advice. "Thank you, Dr. Abdullah and, thank you, Dr. Areibi."

Finally, all brothers, sisters, friends, colleagues, faculty, and staff at the University of Guelph, thanks a lot! You have all very supportive and helpful. I would like also to thank Jeff Bueckert for his help in the implementation. "Thank you Jeff, you were of great help."

Contents

1 Introduction 1
1.1 Motivations 1
1.2 Objectives 2
1.3 Contributions 4
1.4 Thesis Organization 4
2 Background 6
2.1 Mobile Robots 6
2.2 Autonomous Robots 8
2.3 Path Planning 9
2.3.1 Path Planning Problems Classifications 10
2.3.2 Path Planning Algorithms 11
2.4 Optimization 11
2.4.1 Components of Optimization Problems 12
2.4.2 Optimization Problem Classification 14
2.4.3 Dynamic Optimization Problems 14
2.4.4 Combinatorial Optimization Problems 15
2.4.5 Algorithm Classifications 16
2.5 Genetic Algorithms 19
2.5.1 Solution Encoding 20
2.5.2 Population 21
2.5.3 Fitness 22
2.5.4 The Evolution Process 22
2.5.5 Replacement Strategy and Stopping Criteria 25
2.6 Parallel Genetic Algorithms 26
2.6.1 Master-Slave Scheme 27
2.6.2 Fine-Grained 28
2.6.3 Coarse-Grained 28
2.7 Summary 29
3 Literature Review 31
3.1 Environment representation 31
3.2 Path Planning Approaches 32
3.2.1 Roadmap Approach 33
3.2.2 Cell Decomposition Approach 35
3.2.3 Artificial Potential field approaches 37
3.3 GA Based Path Planning 37
3.4 Summary 42

4 Genetic Algorithm Planner 43
4.1 Problem Definition 43
4.2 The Algorithm 44
4.2.1 Environment Representation 44
4.2.2 Chromosome Representation 45
4.2.3 Initial Population 47
4.2.4 Path Evaluation 47
4.2.5 Genetic Operators 51
4.2.6 Replacement Strategy and Stopping Criteria 55
4.3 Dynamic planning 56
4.4 Experimental setup 58
4.5 Sensitivity of GA Parameters 59
4.5.1 Fitness Parameters 60
4.5.2 Population Size 61
4.5.3 Operators probability 63
4.6 Results 73
4.6.1 Static Environments 76
4.6.2 Dynamic Environments 81
4.7 Parallel Implementation 93
4.7.1 Results 94
4.8 Summary 103
5 Local Search and Memetic Algorithms 104
5.1 Local Search 104
5.1.1 LS Results 106
5.1.2 GA vs. LS in Static Environments 113
5.1.3 GA vs. LS in Dynamic Environments 116
5.2 Memetic Algorithms 118
5.2.1 Results 119
5.3 Summary 121
6 Conclusions 123
6.1 Conclusions 124
6.2 Future work 125
Appendix A 127
References 130

List of Tables

Table 2.1: Combinatorial explosion for the TSP problems 16
Table 2.2: Main differences between exact and approximate algorithms 17
Table 2.3: Optimization solver (GA) analogy with Real biology..... 19
Table 4.1: Algorithm terminology. 46
Table 4.2: Best, average and CPU time vs. population size (map2)..... 62
Table 4.3: Repair operator effect on map3 67
Table 4.4: Tasks attributes 74
Table 4.5: Task 1 results 90
Table 4.6: Task 2 results 90
Table 4.7: Task 3 results 91
Table 4.8: Task 4 results 91
Table 4.9: Task 5 results 92
Table 4.10: IGA results: migration frequency = 10, migration percentage = 10% 95
Table 4.11: IGA results (BSF): frequency = 10, percentage = 10%..... 95
Table 4.12: DGA results (BSF): frequency = 10, percentage = 20% 97
Table 4.13: DGA results (BSF): frequency = 10, percentage = 20% 98
Table 4.14: DGA results (BSF): frequency = 10, percentage = 30% 98
Table 4.15: DGA results (BSF): frequency = 10, percentage = 30% 98
Table 4.16: DGA results (BSF): frequency = 20, percentage = 20% 99
Table 4.17: DGA results (BSF): frequency = 20, percentage = 20% 99
Table 4.18: DGA results (BSF): frequency = 20, percentage = 30% 99
Table 4.19: DGA results (BSF): frequency = 20, percentage = 30% 100
Table 5.1: LS results for Task 1..... 107
Table 5.2: LS results for Task 2..... 108
Table 5.3: LS results for Task 3..... 108
Table 5.4: LS results for Task 4..... 109
Table 5.5: LS results for Task 5..... 109
Table 5.6: LS results for Task 6..... 110
Table 5.7: LS results for Task 7..... 110
Table 5.8: LS results for Task 8..... 111
Table 5.9: Best path cost..... 119
Table 5.10: Average path cost 119
Table 5.11: Average CPU time..... 119

List of Figures

Figure 1.1: Path planning (Planner) as a part of full mobile robot system 3
Figure 2.1: Robots applications 7
Figure 2.2: Simple LP model 13
Figure 2.3: Solution space for LP problem in Figure 2.2 13
Figure 2.4: Optimization problems classification 14
Figure 2.5: Combinatorial explosion 16
Figure 2.6: Heuristics (LS) vs. meta-heuristics 18
Figure 2.7: SGA structure. 20
Figure 2.8 : Path encoding (a) schema of encoding 8 possible movements of the robot. (b) encoding of a complete path 21
Figure 2.9: Roulette wheel 23
Figure 2.10: One point crossover operation 24
Figure 2.11: Mutation operation 25
Figure 2.12: Master-slave parallel GA 27
Figure 2.13: Fine-grained structure 28
Figure 2.14: Migration structures, DGA with six islands 29
Figure 3.1: Environment representations approaches 32
Figure 3.2: Roadmap approach. (a) initial robot environment, (b) nodes as generated using trapezoidal map, (c) connectivity graphs connecting each adjacent nodes, and (d) search algorithm is applied to find a free path 34
Figure 3.3: Exact cell decomposition. (a) initial environment (b) composing the Cfree into trapezoidal and triangular cells (c) construction of the connectivity graph (d) path in the connectivity graph determines the channel in the Cfree. 35
Figure 3.4: Approximate cell decomposition. (a) Cspace, (b) approximate decomposition and the free path 36
Figure 3.5: MAKLINK Graph: (a) environment with obstacle and free convex area, and (b) a graph representation of the mobile environment 39
Figure 4.1: Obstacle representation 44
Figure 4.2: Chromosome representation 45
Figure 4.3: Two paths generated for the same task. 46
Figure 4.4: Randomly generated paths. 47
Figure 4.5: Path cost calculations 48
Figure 4.6: Two infeasible paths 51
Figure 4.7: Crossover operation. (a) Crossover positions, (b) Divided paths, (c) New combined paths. 52
Figure 4.8: Mutation operator 53
Figure 4.9: Random repair operator 54
Figure 4.10: Exact repair operator 54
Figure 4.11: Shortcut operator 55
Figure 4.12: Smooth operator 55
Figure 4.13: Path similarity 57
Figure 4.14: Selected Test Set 59

Figure 4.15: Influence the preferred clearance parameter (τ): (a) τ = 0, (b) τ =1,(c) τ = 2, and (d) τ = 4 60
Figure 4.16: Influence of the preferred steering angle parameter (α), (a) α = 90°, (b) α= 45°, (c) α= 20°, and (d) α = 5° 61
Figure 4.17: Population size effect in the solution quality 63
Figure 4.18: Population size effect in the algorithm complexity 63
Figure 4.19: Crossover rate vs. average path cost (mutation fixed at 10%) 64
Figure 4.20: Mutation rate vs. average path cost (crossover fixed at 50%) 65
Figure 4.21: Population snapshots, mutation = 10% 66
Figure 4.22: Population snapshots, mutation = 90% 66
Figure 4.23: Effect of the repair operator (map3) 68
Figure 4.24: Average of best so far for the shown configuration (map3) 68
Figure 4.25: Repair effect on the gain of the average cost and CPU time (map3) 69
Figure 4.26: Repair operator vs. average cost 69
Figure 4.27: Shortcut rate vs. CPU time 70
Figure 4.28: Shortcut rate vs. solution quality 71
Figure 4.29: Smooth rate vs. solution quality 72
Figure 4.30: Replacement strategy: parent replacement vs. worst replacement 72
Figure 4.31: Replacement strategy complexity 73
Figure 4.32: Benchmarks 75
Figure 4.33: Task 1 results 76
Figure 4.34: Task 2 results 77
Figure 4.35: Task 3 results 77
Figure 4.36: Task4 results 78
Figure 4.37: Task 5 results 78
Figure 4.38: Task 6 results 79
Figure 4.39: Task7 results 79
Figure 4.40: Task 8 results 80
Figure 4.41: Algorithm complexity with respect to the environment attributes 81
Figure 4.42: Population snapshots task 1 using the basic algorithm 83
Figure 4.43: Best path (GAP) 84
Figure 4.44: Best paths: GAP +Memory 84
Figure 4.45: GAP vs. GAP +Memory 85
Figure 4.46: Population snapshots task 1 using GAP + LC 86
Figure 4.47: Best paths (GAP+ LC) 87
Figure 4.48: GAP vs. GAP+ LC 87
Figure 4.49: GAP vs. GAP + RI 88
Figure 4.50 : GAP vs. GAP +MRI 89
Figure 4.51: Task 1 Dynamic mode 90
Figure 4.52: Task 2 Dynamic mode 90
Figure 4.53: Task 3 Dynamic mode 91
Figure 4.54: Task 4 Dynamic mode 91
Figure 4.55: Task 5 Dynamic mode 92
Figure 4.56: Staic vs. dynamic (Task 2) 92
Figure 4.57: Staic vs. dynamic (Task 3) 93
Figure 4.58: (BSF) Speed-up: frequency =10 and percentage = 10% 96

Figure 4.59: Relative fitness (BSF), with frequency =10 and percentage = 10% 96
Figure 4.60: Relative fitness (BSF): frequency =10 and percentage = 10% 97
Figure 4.61: Solution quality deviation against different migration parameters 100
Figure 4.62: (BSF) Sample outputs for benchmark 7 101
Figure 4.63: (GB) Speed-up: frequency =10 and percentage = 10%..... 102
Figure 4.64: Relative fitness (GB): frequency =10 and percentage = 10%..... 102
Figure 4.65: Relative fitness (GB): frequency =10 and percentage = 10%..... 103
Figure 5.1: LS algorithm..... 105
Figure 5.2: Path improvements: (a) initial repaired path, (b) nodes search window, and (c) improved path 106
Figure 5.3: Accept first vs. accept best strategy (Task 1)..... 107
Figure 5.4: Accept first vs. accept best strategy (Task 2)..... 108
Figure 5.5: Accept first vs. accept best strategy (Task3)..... 108
Figure 5.6: Accept first vs. accept best strategy (Task 4)..... 109
Figure 5.7: Accept first vs. accept best strategy (Task 5)..... 109
Figure 5.8: Accept first vs. accept best strategy (Task 6)..... 110
Figure 5.9: Accept first vs. accept best strategy (Task 7)..... 110
Figure 5.10: Accept first vs. accept best strategy (Task 8)..... 111
Figure 5.11: Solution quality (accept first vs. accept best)..... 111
Figure 5.12: Computational time (accept first vs. accept best)..... 111
Figure 5.13: LS Dynamic result (Task 1) 112
Figure 5.14: LS Dynamic result (Task 2) 112
Figure 5.15: LS Dynamic result (Task 3) 112
Figure 5.16: LS Dynamic result (Task 4) 113
Figure 5.17: LS Dynamic result (Task 5) 113
Figure 5.18: Best solution GA vs. LS (Static) 114
Figure 5.19: Average path cost GA vs. LS (Static) 115
Figure 5.20: Average CPU time GA vs. LS (Static)..... 115
Figure 5.21: Best solution GA vs. LS (Dynamic)..... 116
Figure 5.22: Average path cost GA vs. LS (Dynamic)..... 117
Figure 5.23: Average CPU time GA vs. LS (Dynamic) 117
Figure 5.24: Best Paths for the implemented techniques..... 120
Figure 5.25: Average Cost for the implemented techniques..... 120
Figure 5.26: Average CPU Time for the implemented techniques..... 121

Chapter 1

Introduction

The field of robotics has attracted a great deal of attention in research and industrial communities. What was considered to be science fiction 25 years ago has now become a reality. Currently robots are used in manufacturing, medicine, services, exploration and transportation. However, future robotic systems will need to be more autonomous and intelligent than the present ones so that robots can execute various types of tasks with minimum or no human intervention. One of the challenges for such an intelligent robot is determining its fastest and safest route to its destination. This is what is known as the path planning problem.

The path planning problem is an ordering problem, where a sequence of configurations is sought, beginning at the initial location and ending at the goal location. The robot searches for an optimal or near optimal path with respect to the problem objectives, whose criteria include distance, time, energy, safety and smoothness.

1.1 Motivations

Mobile robot path planning is one of the problems that need to be solved to achieve full robot autonomy; therefore, the need for a robust, adaptive, intelligent planner has become

essential. Many approaches have been proposed, but so far, no robotic system can navigate efficiently in the real world without human supervision or guidance.

The driving forces behind this research can be summarized as follows:

- The need for an autonomous path planner, so a mobile robot can plan its actions in real world environments with minimum or no human supervision or guidance.
- The drawbacks of the existing techniques, including adaptability, computational complexity, a poor response to uncertainty, a lack of robustness in the optimization goals and the lack of alternative paths.
- Carry out a feasible study of the potential of Genetic Algorithms (GAs) to effectively solve complex problems.
- The Complexity of GA and to enhance its performance led to an efficient parallel implementation.
- The fact that GA offers more than one solution for the same problem; in a dynamic environment, this is useful so that one of the alternative solutions can be used when the current path becomes infeasible.

1.2 Objectives

The goal of this research is to design and implement a robust planner for solving mobile robot path planning problems in static and dynamic environments. This planner should be easily integrated with a mobile robot system which includes a sensory system (to attain the necessary information about the environment) and a control system (to control the movements of the robot) as illustrated in *Figure 1.1*. By introducing this planner, the robot

should be able to work autonomously in its environment with minimum or no human supervision.

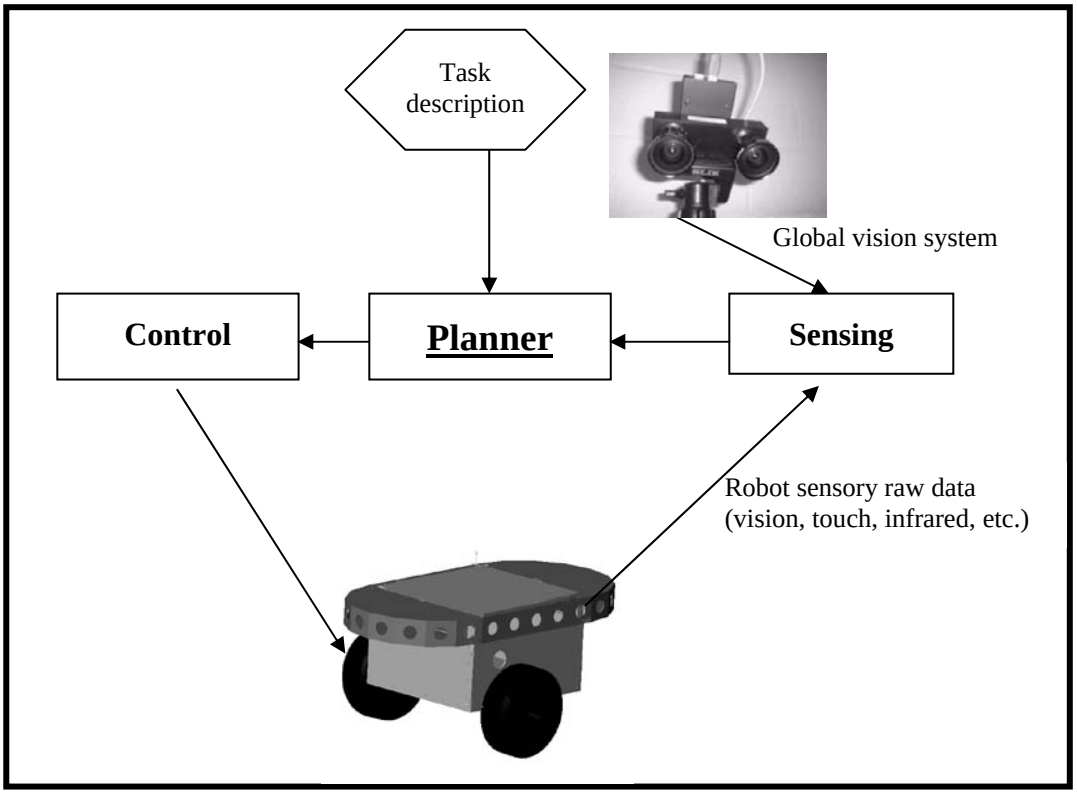


Figure 1.1: Path planning (Planner) as a part of full mobile robot system

To solve the path planning problem, a Genetic Algorithm Planner (GAP) based on a variable length representation is implemented in this thesis. A generic fitness function is used to combine all the objectives of the problem. Then, different evolutionary operators are also applied: some depend on randomness and others employ problem-specific domain knowledge. Different benchmarks are developed to test the system performance in both static and dynamic environments. Furthermore, the GAP is parallelized using the Message Passing Interface (MPI) library for a speed-up. An Island-based GA (IGA) is implemented by using a ring topology with different migrations approaches techniques. Novel Local Search (LS)

algorithm is further proposed in this work and different approaches are examined for combining the LS algorithm with the GAP to obtain superior solutions.

1.3 Contributions

The contributions of this dissertation are summarized as follows:

- The Exploration of the feasibility of applying a GA to solve the path planning problem in static and dynamic environments and fine tuning critical parameters.
- Design of a new heuristic technique (LS) to solve path planning.
- Investigation of an Island-based parallel GA was carried out to enhance the system performance.
- The examination of GA and LS hybridization (Memetic Algorithms) to obtain high quality solutions.

1.4 Thesis Organization

The thesis consists of six chapters. Chapter 2 provides the necessary background to define the path planning problem, optimization and GAs. Conventional approaches and GA approaches to solve the path planning problem are reviewed in Chapter 3. In Chapter 4, a detailed implementation of the developed GAP along with the algorithm analysis and results for the static and the dynamic environments is presented. The parallel implementation details and the results of the IGA are also presented in Chapter 4. Chapter 5 introduces the developed Local Search (LS) approach to solve the path planning problem and its

hybridization with the GAP. Finally, chapter 6 provides a conclusion and suggestions for future work.

Chapter 2

Background

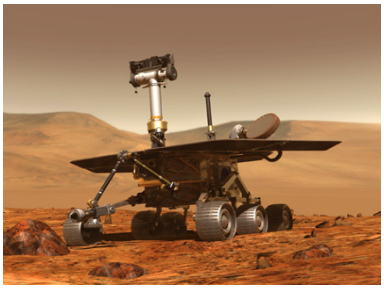
This chapter reviews several topics that are related to path planning. The introduction of the path planning problem is followed by a discussion of optimization, complexity of the combinatorial optimization problems and the Genetic Algorithms. By the end of this chapter, the reader should have acquired the necessary background information to appreciate the contributions of this thesis.

2.1 Mobile Robots

The Czechoslovakian playwright, Karel Capek introduced the word robot in the play, *R.U.R.* (Rossum's Universal Robots). The word, *robot*, is derived from the Czech "*Robota*" which refers to servitude, forced labour, or slave. In the dictionary, a robot is defined as *a mechanical device that sometimes resembles a human and is capable of performing a variety of often complex human tasks on command or by being programmed in advance*. Russell et al. [1] defines a robot as an *active, artificial agent whose environment is the physical world*. This definition includes all the artificial creatures that exist physically in the real world and interact with it in some manner.

Over the past 30 years, an increasing interest in robotic systems has been expressed. Robotic systems have proven to be crucial in such fields as manufacturing automation, space and deep sea exploration, dangerous and hazardous missions (e.g. rescue, police and military missions), and finally life-like new toys that talk and respond similar to real creatures. Example of such systems are reflected in Figure 2.1

(a) General Material Handling



(b) Exploration

(c) Rescue

(d) AIBO Toy (SONY Corporation)

Figure 2.1: Robots applications

One class of robots that has attracted special attention is that of mobile robot. It is “a robot vehicle capable of self-propulsion and (pre)programmed locomotion under automatic control in order to perform a certain task” [2].

Mobile robots have a wide range of potential applications, such as the transportation of objects in buildings, factories, warehouses, airports, and libraries, service robots that can vacuum apartments, and inspection robots that operate in hazardous environments and space exploration. Although the demand for these applications is high, the limitations of the

existing robots in the real world, as well as their high cost, have disallowed broad practical utilization. The bottleneck in this effort is the problem of the planning and navigation, and the lack of the required flexibility and adaptation in different environments and setups.

2.2 Autonomous Robots

Future robots are required to be more autonomous than present robot systems. For a robot to be autonomous, it must answer the following questions: where am I, where should I go, and how can I travel there?

A robot requires several capabilities to answer these questions and be able to operate in an intelligent and autonomous manner. These capabilities fall into three categories [3]:

(1) Sensing: allows the robot to gather information about its surrounding environment by using different sensing devices. The raw sensory data needs to be analyzed and transformed into a realistic model that represents the environment.

(2) Motion Planning: is performed to plan given tasks such as robot navigation, robot assembly on an assembly line, and machining.

(3) Control: A low-level control is required to execute each planned task.

A robot that is equipped with such capabilities can autonomously plan and execute different tasks successfully. For example, a user may ask the robot to bring him/her a coffee mug from a certain location. The robot has to break this task into subtasks: how to obtain the coordinates of the mug, how to "gently" grasp the mug, how to return to the location of the user and, how to hand him/her the requested mug.

The sensing capability allows the robot to sense the objects in its working space and repetitious to answer the first question: "Where am I?". The motion planning capability allows the robot to answer the second question "Where should I go?". This capability is the brain by which the robot plans how to reach the location of the mug, when and how to grasp the mug, and how to return it to the point of origin. All the planned motions must be as efficient as possible and as safe as possible. Finally, the control capability would be responsible for executing and monitoring the execution of the planned subtasks.

Motion planning allows the robot to decide how to achieve a given task. It is sufficient for the user to supply the robot with an activity, then, the robot determines on its own how to achieve it. The motion planning problem is as old as robots are; however, most of the revolutionary work in this field was conducted during the 1980s [4]. The motion planning problem is divided into two problems: path planning and trajectory planning [5]. Path planning refers to the design of the geometric specifications (positions and orientations) wherein the dynamics of the robots are neglected, whereas trajectory planning includes the design of the linear and angular velocities to track the found path and reach the goal. In this thesis the path planning is the main focus due to the potential of the applications.

2.3 Path Planning

For path planning, the collision-free routes (paths) must be identified to move a robot from an initial position "A" to a final destination "B". The path should also include the robot mobility constraints and map boundaries. This type of path planning is exercised in several robotic applications, including: finding routes for autonomous robots, planning the

manipulator's movement of a stationary robot, and moving entities to different locations on a map to accomplish certain goals in manufacturing and services applications.

The path planning problem is an ordering problem, where a sequence of configurations is sought, beginning from the location and ending at the goal location. The path planning problem is also called the collision-free path planning problem, where a robot attempts to search for an optimal or near optimal path with respect to the problem criteria. The latter includes distance, time, energy safety and smoothness. The distance is the most typical criterion. Shorter paths are executed faster and require less energy; however, conventional path planning approaches do not take into consideration path safety and path smoothness. The safety constraint is important to both the robot and its surrounding objects. Path smoothness, on the other hand, enhances the energy consumption and the execution time. Smoothness is also a constraint and affects most mobile robots because of the bounded turning radius. For example, a car like robot has this constraint due to the mechanical limitations of its steering angle.

In real-life applications, the robot navigates in dynamic environments adjusting and modifying its planned paths according to the changes that occur in the environment.

2.3.1 Path Planning Problems Classifications

The classifications of path planning problems depend on the problem framework [5]. If all the environment information is known a priori with no ongoing changes, this classification is called *static path planning.* However, if only partial information about the environment is available, the classification is known as *dynamic path planning.* Motion planning can be

either constrained or unconstrained, depending on the restrictions of the motions of the robot, which include velocity boundaries, acceleration boundaries, and curvature constraints.

2.3.2 Path Planning Algorithms

Path planning algorithms are categorized according to their completeness and scope [5]. Complete algorithms are used for finding optimal solutions. They either find an exact solution or prove that there is no solution at all. Non-complete (or heuristic) algorithms are adopted for finding a near-optimal solution in a short period of time. There is strong evidence that a complete planning requires time that is proportional to the number of the degrees of freedom (DOF) of the robot. Therefore, Canny [6] classified the path planning problem algorithms as NP-Complete.

Depending on the scope, path planning algorithms are divided into two categories: *global* and *local*. In global algorithms all the environment information is considered, and the path is planned form start to finish. This type of path planning is also known as *off-line* path planning. Local algorithms are designed to avoid obstacles near the robot and to improve the path safety and smoothness; therefore, only the information that is close to the robot is employed. Local path planning is also known as *on-line* path planning.

2.4 Optimization

Optimization concerns decision-making. Optimization or *mathematical programming* is the study of maximizing and/or minimizing the functions that satisfy the predefined criteria called boundary conditions or *constraints*. Optimization forms an integral part of many applications in engineering, management, and economics.

2.4.1 Components of Optimization Problems

The elements that constitute an optimization model include design variable, objective function and constraints

Design Variables

The design variables represent the variables that are required to quantify or describe the system. The design variables consist of design parameters and decision variables. The design parameters are the data that defines the problem and the decision variables are the quantities whose numerical values are sought to obtain the optimal solution.

Objective Function

Objective function or cost function is the prescribed criterion by which the solutions are evaluated. It is a mathematical equation that embodies the design variables. The ultimate goal is to minimize the cost or maximize the profit by minimizing or maximizing the objective function.

Constraints

The optimization constraints, are the conditions that must be satisfied, while the optimal solution is being sought which are applied to the design variables. A constraint can be written mathematically, either in an equality format such as $x_1 = 0$ or in the form of an inequality such as $x_1 \geq 0$. The *design space* (or the *solution space*) is the total region or domain, defined by the design variables in the objective function. Usually, the design space is divided into two regions: *feasible* and *infeasible*. The feasible region satisfies the problem constraints, and the infeasible region does not satisfy the problem constraints.

By using the basic optimization components the optimization problem is defined as follows: *Find the values of the variables that minimize or maximize the objective function where the constraints are satisfied*

This discussion is illustrated by considering the linear programming optimization model in Figure 2.2.

$$
\begin{aligned}
& \textit{Max} \quad x_1 + 2x_2 \\
\text{Subject to} \ & : \\
& x_1 + x_2 \le 24 \\
& 2x_1 + x_2 \le 30 \\
& x_1 \ge 0; x_2 \ge 0
\end{aligned}
$$

Figure 2.2: Simple LP model

In this simple model, the objective function is $x_1 + 2x_2$, the decision variables are x_1 and x_2, and the constraints are given in the inequality form. Figure 2.3 reflects how the solution space is divided into a feasible region and an infeasible region.

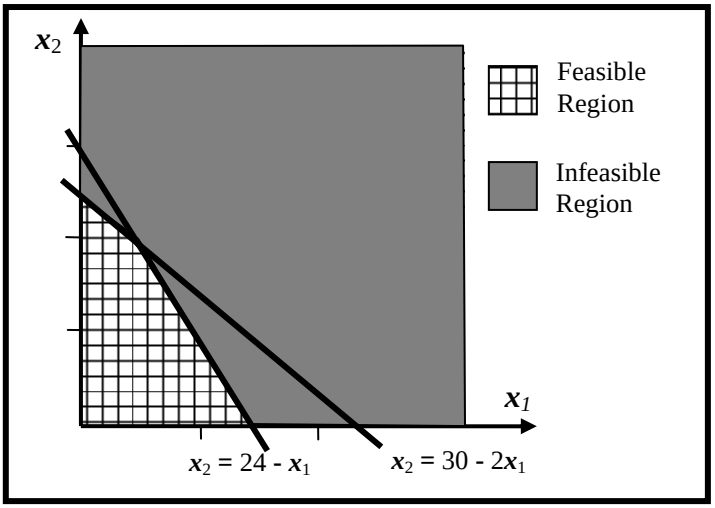


Figure 2.3: Solution space for LP problem in Figure 2.2

2.4.2 Optimization Problem Classification

Naturally optimization problems are divided into two categories: continuous and discrete. Continuous optimization problems seek to solve variables that are defined in the real space. Discrete optimization problems refer to problems where the variables can take on discrete values. Within this context, the classes of optimization problems are shown in Figure 2.4.

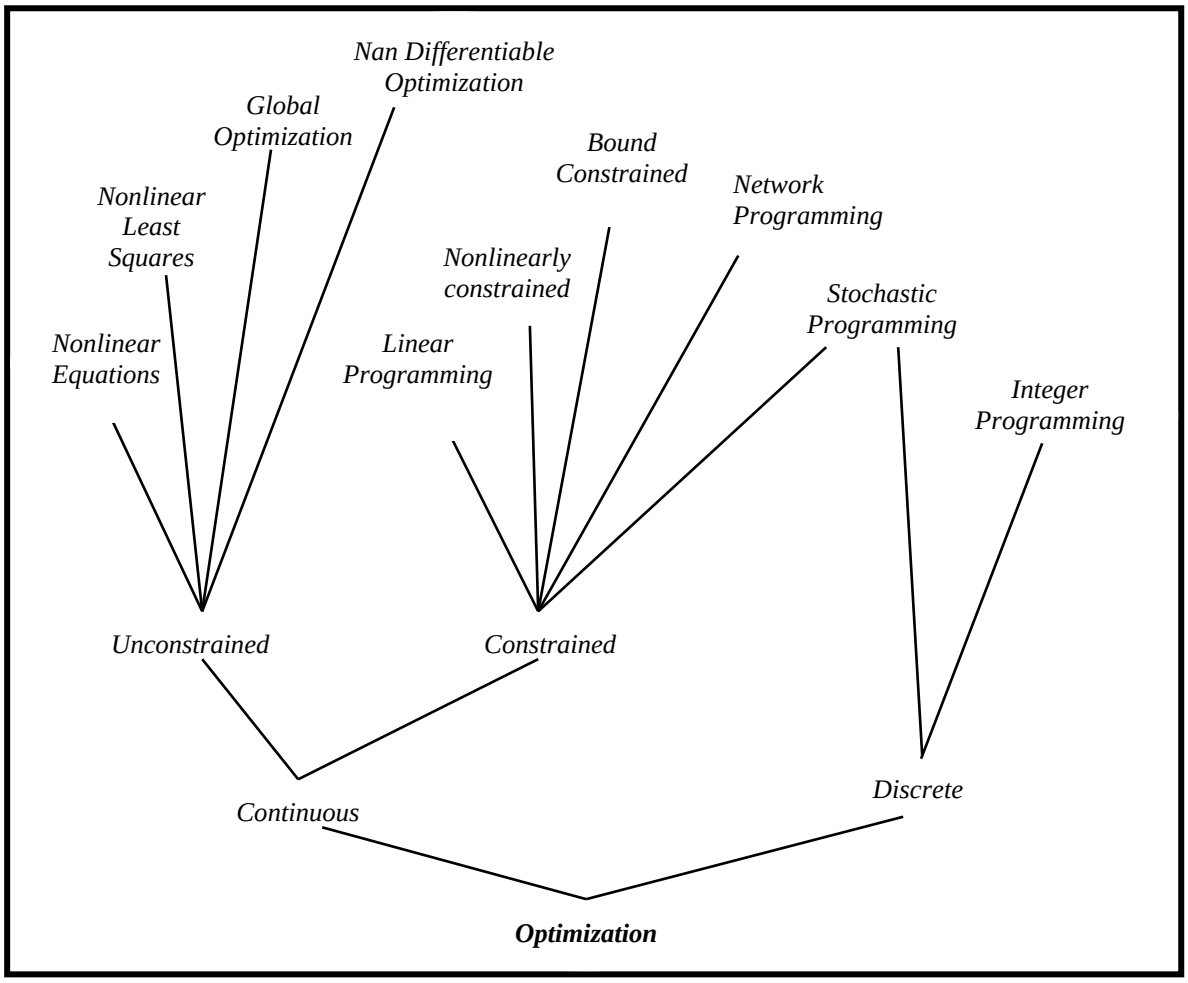


Figure 2.4: Optimization problems classification

2.4.3 Dynamic Optimization Problems

When the optimization problem is time dependent, it is said to be dynamic. Most real world applications are time dependent. When a dynamic optimization problem is being solved the

objective is no longer to find the optimal solution but to track the progression of the optimal solution throughout the solution space. In general, a dynamic problem is more complex than a static problem, due to the following:

- Changes in the problem size: The solution space is time dependent, and therefore, the complexity of the problem is time dependent.
- Feasibility changes: As time progresses, feasible solutions can become infeasible and vice versa.

2.4.4 Combinatorial Optimization Problems

Combinatorial Optimization Problems (COPs) are decision problems that have a countable or finite number of solutions. These types of problems are encountered in everyday situations, particularly in engineering design. It may seem trivial to obtain the optimal solution for combinatorial problems simply by checking all the feasible solutions. However, it turns out that finding this optimal solution becomes intractable, when the number of variables increases. One of the challenges in combinatorial optimization is to deal effectively with what is known as a *combinatorial explosion*, where the number of feasible solution grows exponentially as the size of the problem increases.

For example, consider the Traveling Salesman Problem (TSP) which is defined as follows. A salesman needs to visit a finite number of cities. A cost is associated with each path that is travelled between the two cities. The objective is to find the least costly solution for the salesman to visit each city only once and return to the starting city. For a problem with n cities, the possible routes are $(n!)/2$. Table 2.1 shows how the number of possible

routes exponentially increases as the number of cities increases. The running time is calculated, based on assumption that one could possibly enumerate 10^9 tours per second.

Number of cities	Possible routes (*n!/2*)	Running time
10	1.8144×10^{06}	0.0018 sec
15	6.5384×10^{11}	10.89 min
20	1.2165×10^{18}	~ 38 years
25	7.75561×10^{24}	~ 0.245×10^{09} years
30	1.32626×10^{32}	~ 4.20556×10^{15} years

Table 2.1: Combinatorial explosion for the TSP problems

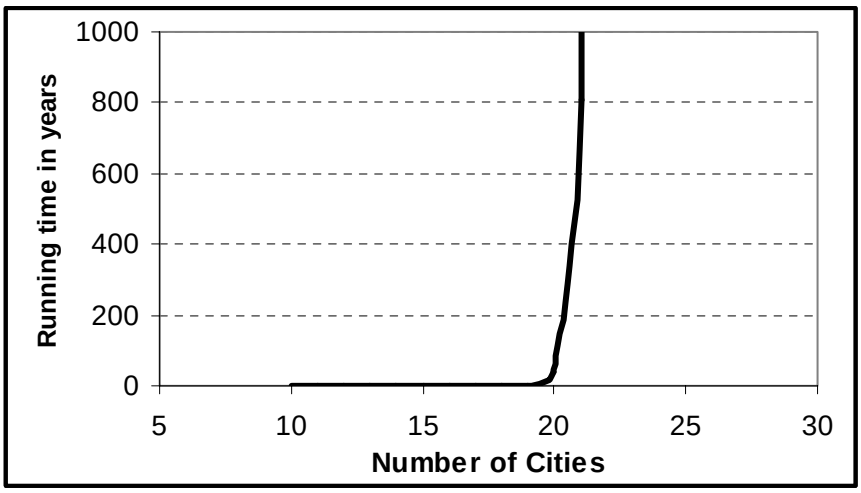


Figure 2.5: Combinatorial explosion

2.4.5 Algorithm Classifications

Combinatorial optimization problems can be solved using search algorithms. These algorithms are classified as either exact or approximate approaches. Exact algorithms are used to produce solutions that are optimal. Approximate algorithms take a different approach; instead of seeking the optimal solution, they aim to produce a near-optimal solution by using a reasonable amount of computational resources. Approximate algorithms

are used to solve large complex problems that the exact algorithm fails to solve in a reasonable time. The basic differences between the two classes are presented in Table 2.2.

Approximate algorithms can be further classified as heuristic or meta-heuristic. Simple heuristic techniques, also known as L*ocal Search* (LS for short) and *Hill-Climbing*, operate as iterative improvement techniques. The improvement process is either done deterministically or randomly. In the LS, only the moves that result in an immediate improvement in the objective function are accepted. Therefore, iterative improvement techniques usually become trapped in a *local minimum*, which can be far from the *global minimum* as shown in Figure 2.6(a).

	Exact	**Approximate**
Computation Time	high	low
Solution Quality	optimal	sub-optimal
Performance	guaranteed	not guaranteed
Implementations	modelling	heuristic implementation

Table 2.2: Main differences between exact and approximate algorithms

A meta-heuristic is an iterative master process that guides the operations of the subordinate heuristics (usually a local search). The main characteristic of meta-heuristic techniques is the strategy it uses to escape the local minimum. In contrast to local search, which only accepts downhill moves, the meta-heuristic algorithms allows for uphill moves to avoid being trapped in a local minimum as illustrated in Figure 2.6(b)

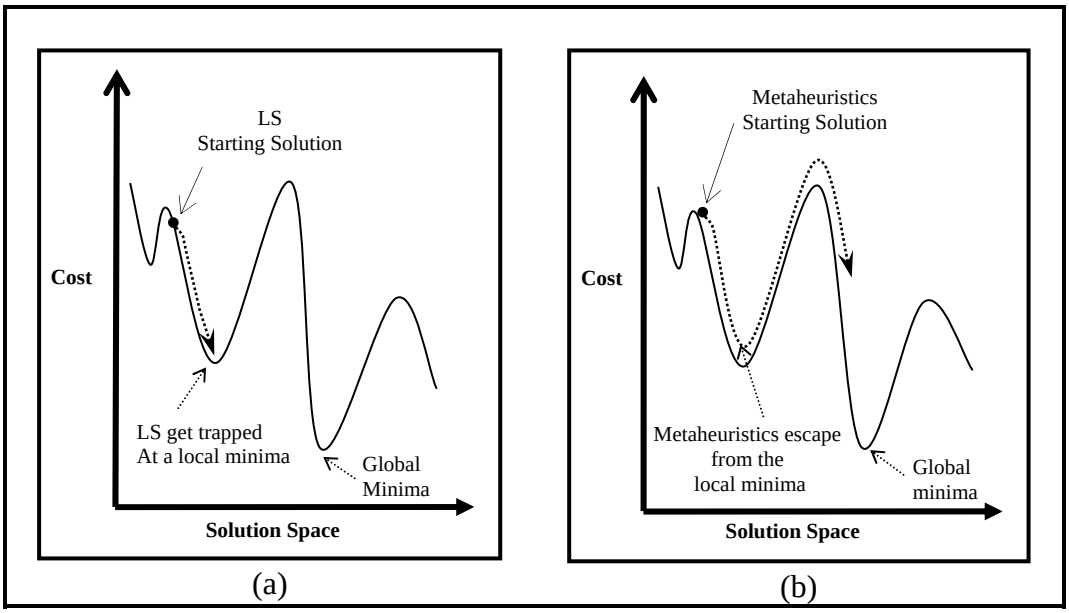


Figure 2.6: Heuristics (LS) vs. meta-heuristics

Various meta-heuristic search principles have been developed. Some of them have been inspired by nature and are modelled on processes such as annealing and evolution [7]. The Simulated Annealing (SA), Greedy Randomized Adaptive Search Procedure (GRASP), Tabu Search (TS), Genetic Algorithms (GA) and Ant Colony Optimization (ACO) [8,9], are the most widely applied meta-heuristics techniques for large combinatorial problems. A meta-heuristic search process can manipulate a single solution (e.g. SA and TS) or a collection of solutions per iteration (e.g. ACO and GA). Each meta-heuristic technique uses a different strategy to guide the search, and escape from a local minimum. The goal of the meta-heuristic technique is to efficiently explore the search space in order to find an optimal or near-optimal solution. Thus, the element that characterizes the meta-heuristic techniques is the balance between *diversification* and *intensification*. Diversification, also known as *exploitation*, is the component that allows the search process to explore the solution space. Intensification is the component that allows the search process to focus more on the promising regions. A meta-heuristic with such a balance can quickly identify the regions in

the search space that have a high quality solutions, since no time is wasted in the regions of the search space which are either already explored or do not exhibit high quality features.

2.5 Genetic Algorithms

In the 1970's, John Holland introduced Genetic Algorithms (GAs) as an optimization based technique [10]. The continuing performance improvements of computational systems have made GAs attractive for some types of problems. In particular, genetic algorithms work very well on continuous, discrete, and combinatorial problems [11]. A GA is a search strategy that uses a mechanism that is analogous to the evolution of life in nature, where a set of *individuals* (solutions) go through a process of evolution. However, the process of evolution is not a directed process. When different individuals compete for the resources in the environment, those that are fitter are more likely to survive and propagate their genes to the next generation. Holland's GA [10] is a population- based algorithm, where individuals propagate themselves and their genes, based on the mechanisms of *natural selection* and genetically-inspired operators. Table 2.3 lists the analogy of the optimization problem solver with the natural evolution of biology.

Genetics	**Optimization (GA)**
Gene	Bit
Chromosome	Individual (candidate solution)
Fitness	Objective function
Population	Set of solutions
Generation	Iteration
Evolution	Operators

Table 2.3: Optimization solver (GA) analogy with Real biology

Holland's GA is commonly called the Simple Genetic Algorithm (SGA). Crucial to the SGA's proper functionality is a population of binary strings. Each string of 0s and 1s represent the encoded version of a solution to the optimization problem. By using the genetic operators, *crossover* and *mutation*, the algorithm creates, in subsequent generations, new individuals from the current population. This generational cycle is repeated until a desired termination criterion is achieved. Figure 2.7 introduces the SGA in pseudo-code. In the following sections, a detailed description of the components of GA is presented.

```
Simple Genetic Algorithm ()
 {
  Initialize population;
  Evaluate population;
  While termination condition not met
   {
     Select solutions for next population;
     Perform crossover and mutation;
     Evaluate population;
   }
 }
```

Figure 2.7: SGA structure.

2.5.1 Solution Encoding

In nature, the genetic code describes a genotype which is translated into an organism, a chromosome, by the process of cell division. This chromosome represents the solution of the problem. Different mapping strategies are used to map the chromosome. This mapping is called *encoding or chromosome representation*.

Binary encoding is the original chromosome representation; i.e., the chromosomes consist of a binary string of 0s and 1s. However, there is no restriction on the encoding as long as a good method for the encoding and decoding exists. The encoding should include all the design variables.

For example one of the encoding strategies for mobile robot path planning is performed by encoding the moving directions of the mobile robot [12]. As depicted in Figure 2.8 this encoding moves the robot from its current position to one of the eight possible directions.

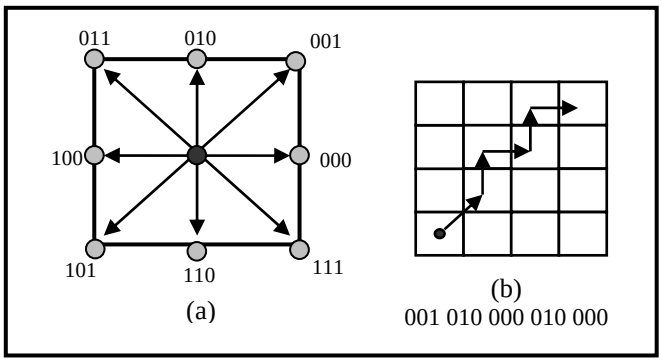


Figure 2.8 : Path encoding (a) schema of encoding 8 possible movements of the robot. (b) encoding of a complete path

2.5.2 Population

The evolved solutions in the GA are called population. The population at a given generation ***t*** is called ***P(t)***. Usually, the initial population (at ***t = 0)*** is generated randomly to give the GA the diversity it needs to explore the solution space. However, combinations of random and constructive solutions are also used to produce the initial population. All the subsequent populations are generated according to the initial population. The *population size*, the number of individuals in the population, is a vital parameter. A large population size allows the GA to explore the entire solution space, but at the expense of high computational time. Generally, the population size should be a function of the problem type, the problem size, and the problem instance. However, finding a reasonable population size is not a trivial task.

2.5.3 Fitness

In nature, it is the fit individuals who are most likely to survive. The mechanism by which an individual's fitness is measured in nature is still unknown. In the GA the fitness is calculated based on the objective function of the problem. Computing the fitness of the population is a time consuming process, since the population is evaluated by calculating the fitness of all the individuals. Furthermore, the evaluation process must deal with the feasibility of the solutions (i.e., some individuals in the population are infeasible). There are many approaches to deal with the feasibility problem. The *penalty* approach [13] attempts to assign fitness to an infeasible chromosome that is worse than the fitness of any feasible chromosome. The *repair* technique seeks to fix infeasible chromosomes to maintain a fully feasible population at all times.

2.5.4 The Evolution Process

Evolution by natural selection is driven, in part, by changes in the gene structure. These changes are usually random; for example during sexual reproduction, radioactivity or cosmic rays can damage the DNA molecule. In the GA s*election*, *crossover* and *mutation* are the basic operators that form the evolution process.

Selection Operator

The selection operator determines which individuals will be chosen for recombination. Although selection is based on fitness, the selection process is random. The most used selection methods are the *roulette wheel selection* and *tournament selection [11]* In the roulette wheel selection, also known as the *fitness proportionate selection,* the individual fitness is used to associate a selection probability; individuals with less fitness are less likely

to be selected. However, there is still a chance that they can be selected. For the analogy to a roulette wheel, imagine a roulette wheel in which each individual represents a slice on the wheel, proportional to the probability of selecting the individual. Figure 2.9 depicts an example of a roulette wheel for a population of five individuals. It is clear that individual number three has a higher probability of being selected, than individual number four which has a lower probability of being selected.

Individual	Fitness
1	150.00
2	212.00
3	740.00
4	95.00
5	420.00

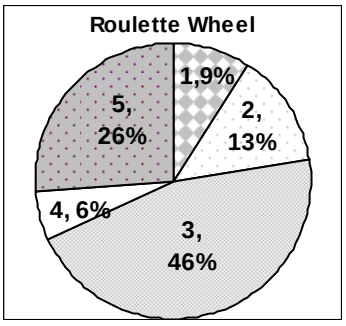


Figure 2.9: Roulette wheel

In the tournament selection, a number of individuals are picked randomly. And the best among these individuals are permitted to reproduce. The fitness function does not really matter, as long as it discriminates well between the individuals. There are several types of tournament selection that are based on the *tournament size*, which is the number of the selected individuals for the comparison. The most common tournament size is two, which is called the *binary tournament selection*.

After the selection, each selected individual is called a *parent*, and all the selected individuals make up what is known as the *mating* pool.

Crossover Operator

Following the selection, a crossover operator is performed on the selected individuals. The crossover, or *recombination*, is the process of combining the genes of one parent with those of another to create offspring that inherit characteristics of both parents. The *crossover probability*, or *crossover rate*, is the probability of performing crossover. The chosen parent is paired with a mate also pre-selected for crossover. From each pair (P_1, P_2) of parents, two offspring (C_1, C_2) will be created that might replace their parents.

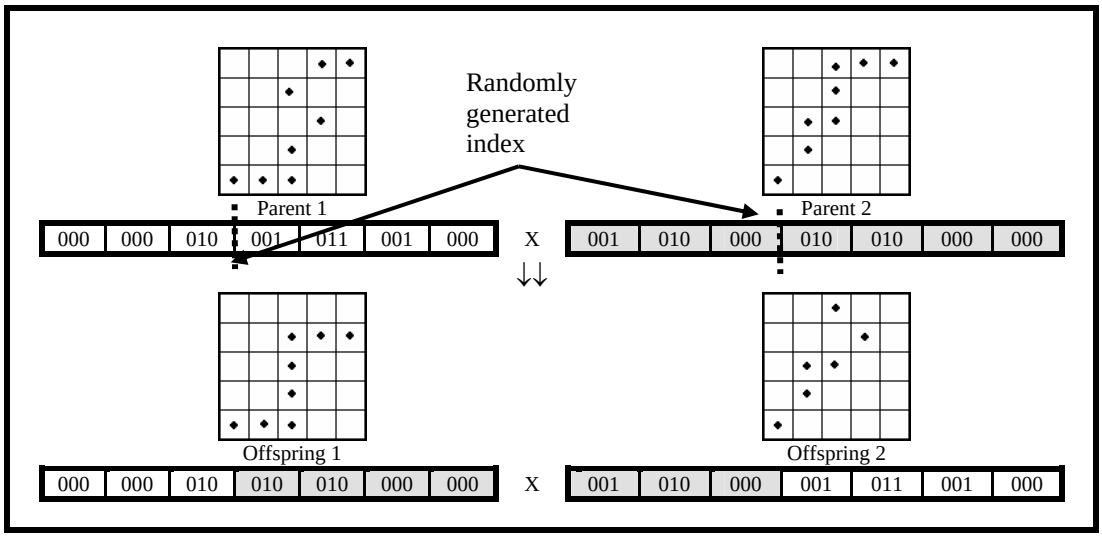


Figure 2.10: One point crossover operation

Figure 2.10 presents a simple illustration of a one point crossover operation. For offspring 1, the genes between the initial position of P_1 and a randomly generated index are inherited from P_1, whereas the remaining genes are inherited from P_2. Also, multi point crossover is employed where more than one part of P_1 is copied and the remaining parts are copied in the same order from P_2. Note that both offspring (C1, C2) in Figure 2.10 are infeasible, and additional operators have to be applied following crossover.

The crossover operator is simple when binary representation is used to encode the fixed length chromosomes. However, when the chromosome encoding is not binary or the length

is variable (such as the path encoding in Figure 2.10) this operator becomes complicated; usually, a special operator need to be designed to overcome the side effects of this operation.

Mutation Operator

Mutation is preformed to give the GA the diversity it needs to explore the entire solution space and help prevent the population from entering a state of stagnation (i.e., the GA stuck in a local minimum). The GA has a *mutation probability*, or *mutation rate*, which dictates the frequency at which the mutation occurs.

For each gene in each individual, the GA randomly changes the gene value at a frequency governed by the mutation rate. In binary strings, 1s are changed to 0s and vice versa. The mutation probability should be kept very low so that good building blocks are not destroyed. Figure 2.11 shows how the mutation operation is performed on a binary encoded chromosome where the bits at the positions, 6, 11, and 18 are altered.

Before mutation	000	000	010	001	011	001	000
Mutation bits	000	001	000	010	000	001	000
After mutation	000	001	010	011	011	000	000

Figure 2.11: Mutation operation

2.5.5 Replacement Strategy and Stopping Criteria

The replacement strategy controls how the newly generated offspring are inserted in the new population. The common strategies are:

- Parent replacement: The new offspring replaces one of its parents.
- Random replacement: The new offspring replaces an individual, randomly selected from the old population.
- Worst replacement: the new offspring replaces the worst individual.

Replacement is often combined with *elitism.* An elitism strategy ensures that the best generated solution(s), so far, are retained in each population. Elitism is performed so that the best generated so far, either replaces the worst individual or a randomly selected individual.

The Stopping criteria, or termination condition, refers to the condition at which the GA terminates. The most commonly used termination condition is based on a predefined number of generations, of which the GA terminates. An alternative stopping criteria is based on the population convergence, where the GA terminates once it does not generate better solutions during the last x generations, where x is a predefined number. Another popular termination condition is based on the population diversity, or *stagnation*, at which the GA terminates after all the chromosomes have become the same (or almost the same).

The previous discussion outlines the basics of the GA. The most important aspects, when a GA is designed for a specific problem, are: the solution encoding, the definition of the objective function, and the definition of each genetic operator. Once these aspects have been well defined, the GA should work fairly well. Beyond that, several approaches can be used to further improve the performance such as incorporating the LS, adaptively fine-tuning the GA parameters or even speeding up the convergence using parallel computing as will be discussed in the next section.

2.6 Parallel Genetic Algorithms

As GAs become more popular, they are applied to complex problems that can require a long computation time. In such cases, parallel implementations of GAs can be used to attain high-quality solutions in a reasonable amount of time.

There are several techniques for implementing parallel GAs [14,15] including the global single-population Master-slave, the single-population fine-grained, and the multiple-population coarse-grained.

2.6.1 Master-Slave Scheme

In this type the population is centrally maintained on a single processor with slave processors that are used to execute only some of the GA operations in parallel as portrayed by Figure 2.12. The most common operation that is parallelized is the evaluation of the individual, where the master executes GA operators and distributes individuals to the slaves. The slaves evaluate the fitness of the individuals. This is highly desirable area for parallelization because the evaluation function is independent of the rest of the population and the evaluation function is the costly component of a GA.

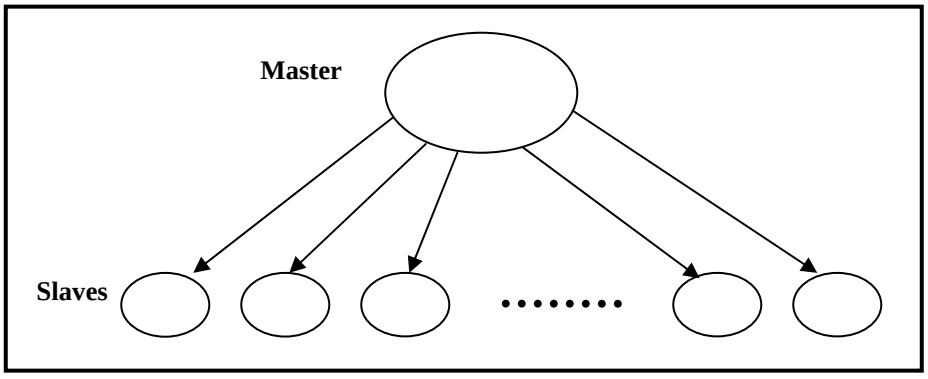


Figure 2.12: Master-slave parallel GA

The advantage of this parallelization type is that there is no need for communication between the processes during the fitness evaluations. However, it requires a significant amount of communication between the master processor and its slaves, since the entire population must be transferred from the master to the slaves at each generation.

2.6.2 Fine-Grained

In this scheme, the population is divided, and the GA operators are restricted to a local neighbourhood, as shown in Figure 2.13. This type of parallel GA is suitable for massively parallel computing. The scheme consists of one spatially-structured population and selection, and mating is restricted to a local neighborhood. However, some interaction among the individuals is allowed. The ideal case is to have only one individual for each existing processor.

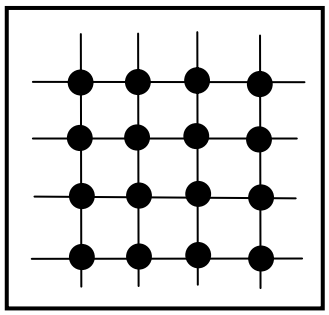


Figure 2.13: Fine-grained structure

2.6.3 Coarse-Grained

This approach is also known as Distributed Genetic Algorithms (DGA) and the Island-based Genetic Algorithms (IGA). The main characteristic of this scheme is that the population itself is divided into subpopulations across the multiple processors. Each island maintains its own population and performs the evolution process locally (i.e., all the genetic operators and the fitness evaluations are performed on the local population). At predetermined times, individuals *migrate* between the islands; some individual(s) are selected from one island and exchanged with individual(s) from another island. The advantage of this scheme is that it eliminates nearly all the communication overhead that is imposed by the master-slave parallelization. Although this approach is faster, it is less

effective in obtaining good solutions due to the fact that smaller populations are generally maintained on each processor and migration is infrequent. There are several techniques by which the DGA is implemented. The differences among these techniques concern how the migrations occur and how the migrated individuals are selected. There are many possibilities for the structure of the migration among the subpopulations. The complete net topology and the ring topology, as represented in Figure 2.14 are the most typical structures.

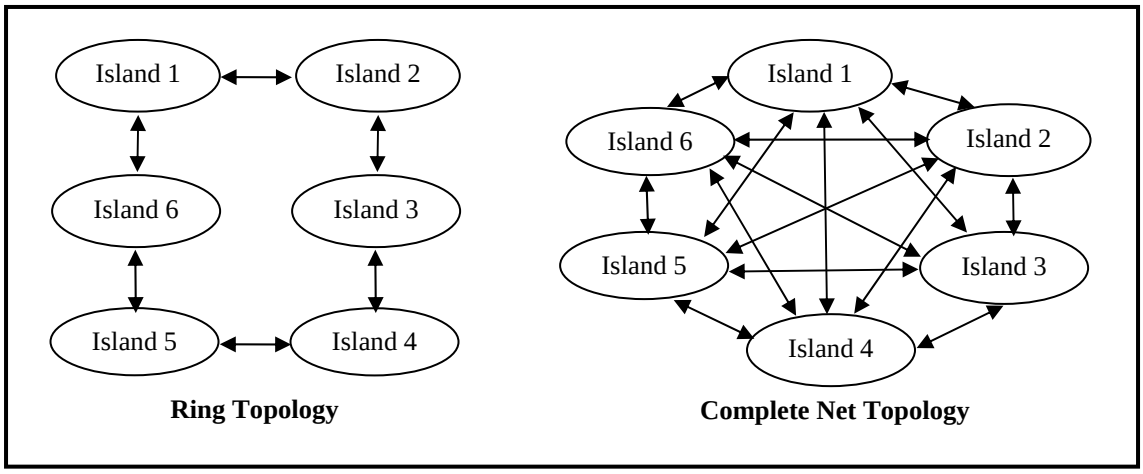


Figure 2.14: Migration structures, DGA with six islands

It is important to notice that only the master-slave method does not affect the behavior of the algorithm, while other methods change the way the GA works. For example, in master-slave, selection takes into account all the population, but in the other two methods, selection only considers a subset of individuals.

2.7 Summary

This Chapter introduces several topics including, the path planning problem, optimizations techniques and the complexity issues faced when solving hard optimization problems. The dynamic path planning problem is a difficult optimization problem and it is

faced in most mobile robots applications. This chapter also introduces Genetic Algorithms (GAs) as a tool to solve hard combinatorial optimization problems. The GA is robust and adaptive because it is a problem-independent search method. Furthermore, The GA provides multiple solutions because it is a population based approach; therefore, it is suitable for solving static and dynamic optimization problems. In the next chapter a complete review of the most common approaches to solve the path planning problem and pervious approaches utilized GA will be introduced.

Chapter 3

Literature review

In Chapter 2 the path planning problem and the GA as a potential optimization tool were introduced. This Chapter presents the most common approaches to solve the path planning problem and describe pervious approaches utilized GA.

3.1 Environment representation

Before a robot can plan a collision-free path, the robot needs a model of the objects in its environment. There are different ways for object representation in robotic environments, [4, 5] including the grid, the *cell tree*, and the *polyhedral*. Figure 3.1 reflects a simple environment representation by using these approaches. In the grid representation shown in Figure 3.1(b) an array of identical cells is setup, and the cells are marked according to the occupancy (usually 1 (dark), if occupied; 0 (white) otherwise). This type of representation simplifies the computation, but requires a large amount of memory [5]. The cell tree method overcomes this disadvantage by using a smaller number of cells. Cells that are completely inside or outside an object(s) are marked as such, and the cells which are partially occupied by object(s) are further divided into smaller cells. The process is repeated until all cells are completely inside or outside the objects or the maximum resolution is reached. The 2D *quadtree* (Figure 3.1(c)) is the most widely used representation of the Cell tree class. This

class of representation is particularly efficient in environments that contain large objects; however, when the environment is occupied by small objects, this representation is wasteful due to the overhead of computing the adjacency of the cells.

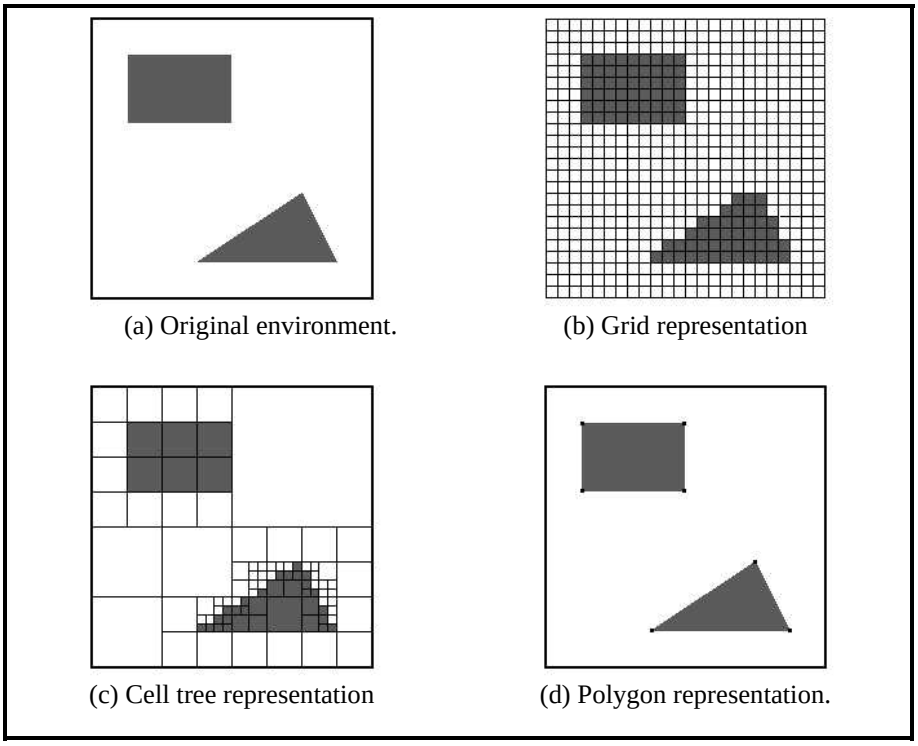


Figure 3.1: Environment representations approaches

In the polyhedral representation in Figure 3.1(d) a description of each object is given by its set of vertices. This representation is popular, since it allows many environments to be closely approximated. Furthermore, this representation has the advantage that many efficient algorithms exist for computing the distance and line segment intersections which are the most important issues in path planning.

3.2 Path Planning Approaches

The most commonly used approaches for solving the path planning problem include the roadmap approach, the cell decomposition approach, and the artificial potential field

approach [4]. However, Most conventional approaches for solving the path planning problem are not performed in the physical workspace (i.e., the space in which the robot and the obstacles are physically present) but in the *configuration space*, denoted as the *Cspace*, which was first introduced by Lozano-Perez and Wesley in 1979 [16]. The Cspace is a topological space that is generated by the set of the all possible configurations. Each configuration corresponds to a transformation that can be applied to the robot. Complicated problems such as determining how to move a piano from one room to another in a house can be reduced by using the Cspace concepts to determine a path for a point in the Cspace. In other words, the piano (3D rigid body) becomes a moving point in the Cspace.

The Cspace consists of two sub spaces; namely the *obstacle space* (*Cobstacle*) and *free space* (*Cfree*). The Cobstacle is a set of infeasible configurations whereas the Cfree is a set of feasible configurations. All motion planning problems become equivalent once the Cspace is formulated. The key difference between the conventional path planning approaches is the methodology by which the Cspace is searched in order to find the global path. The most commonly used search strategy is the graph search strategy. The Cspace is formulated using different techniques; however, computing the Cspace itself is computationally expensive [17,18].

3.2.1 Roadmap Approach

The roadmap approach, also known as the *skeleton*, or the *freeway* approach, is one of the earliest path planning methods [19] that has been widely employed to solve the shortest path problem. The approach is based on capturing the connectivity of the robot's free space in the

form of a network of 1-D curves, as denoted in Figure 3.2. In this approach, the Cspace is used and the key feature of this approach is the construction of a roadmap or a freeway.

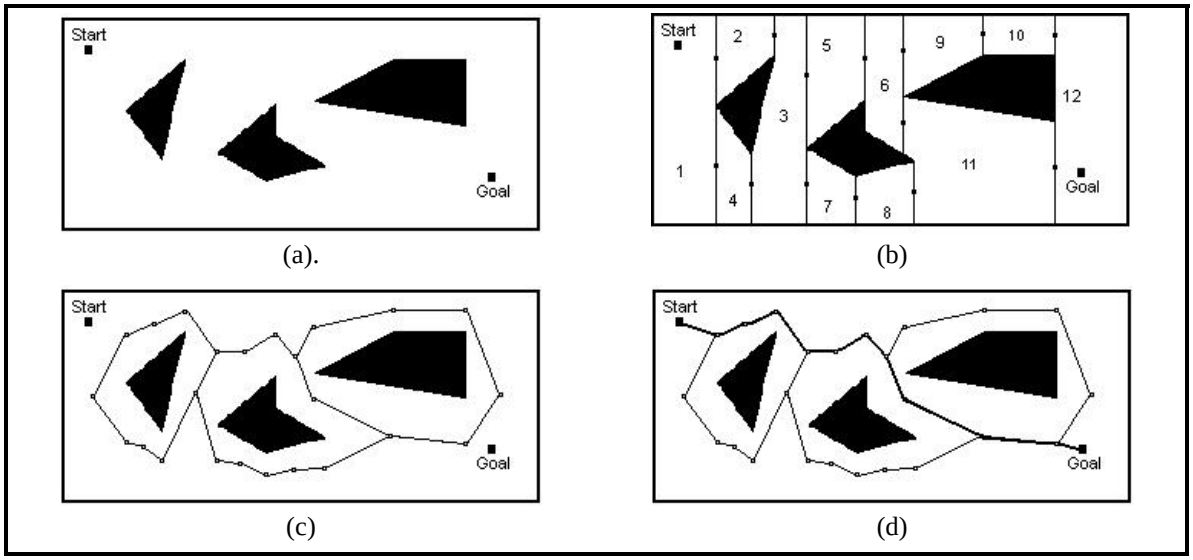


Figure 3.2: Roadmap approach. (a) initial robot environment, (b) nodes as generated using trapezoidal map, (c) connectivity graphs connecting each adjacent nodes, and (d) search algorithm is applied to find a free path

Two phases are involved in the roadmap approach: the pre-processing phase and the query phase. The construction of the roadmap is performed in the pre-processing phase, where a graph whose set of vertices includes the source point and the goal point, and an edge is formed between the two vertices, if the edge is completely in the Cfree. There are different approaches to construct the roadmap [4, 5]. The visibility graph, voronoi diagram and trapezoidal map [20] as illustrated in Figure 3.2 are the most popular techniques.

Searching the roadmap for a free path is performed in the query phase. In this phase a search operator is used to connect the source vertex with the goal vertex. The roadmap is classified as a complete approach, (i.e., it finds a free path, if one exists.) however, other non-complete (probabilistic) variations exist for constructing and searching the roadmap [21]. Probabilistic roadmaps in general improve the speed of the algorithm. However, the principal

disadvantages of the roadmap approaches are: (i) the roadmap goal is to find a free path (not an optimal path or near-optimal), (ii) it is complex and not suitable for dynamic environments due to the need for reconstructing the roadmap whenever a change occur.

3.2.2 Cell Decomposition Approach

In this approach, the Cfree is decomposed into cells, and a connectivity graph is constructed whose vertices and edges represent these cells and their adjacencies. Again, a search operator is used to connect the source point with the goal point in the constructed graph. The decomposition is either exact or approximate.

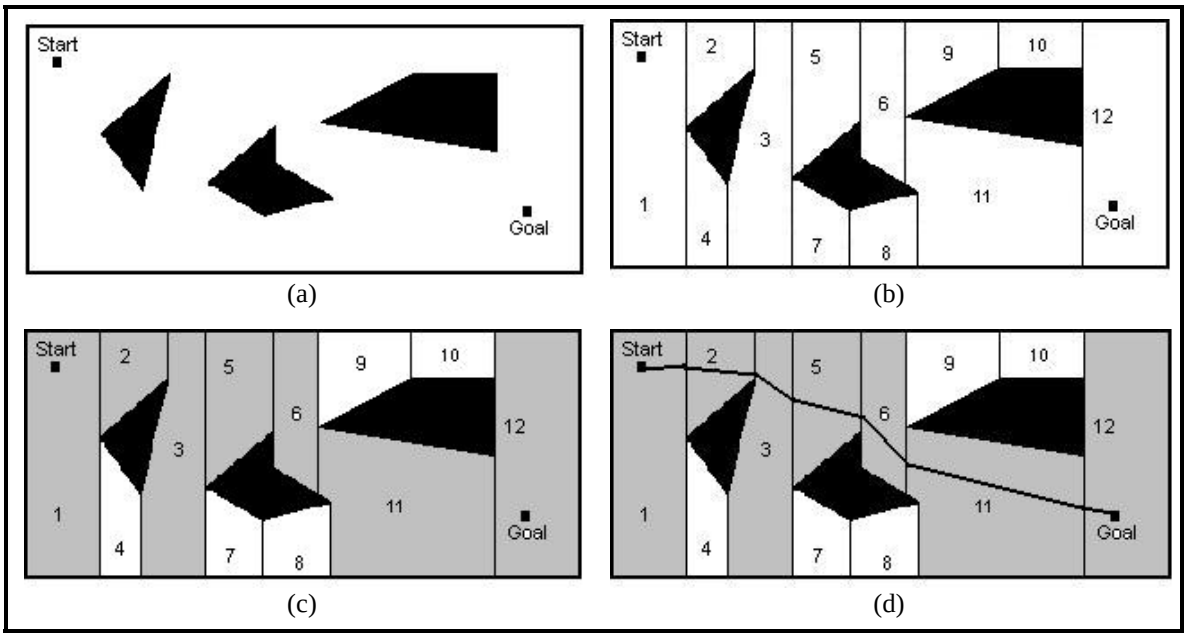


Figure 3.3: Exact cell decomposition. (a) initial environment (b) composing the Cfree into trapezoidal and triangular cells (c) construction of the connectivity graph (d) path in the connectivity graph determines the channel in the Cfree.

Exact cell decomposition generates a set of cells that completely fills the Cfree. The generated cells are complicated due to their irregular boundaries. Figure 3.3 illustrates the concept of the exact cell decomposition in a 2D Cspace. The exact cell decomposition is

considered complete, but this accuracy is a more difficult mathematical process for which the computational time is high, especially in crowded environments.

To effectively reduce the computational complexity, approximate cell decomposition (also called the quadtree (see Section 3.1)) is utilized. Approximate cell decomposition is performed by recursively decomposing the Cspace into smaller cells in steps, each generating four identical new cells. This decomposition continuously subdivides the cells until an arbitrary resolution limit is reached or each cell lies completely in either the Cfree region or in the Cobstacle region. Following decomposition, a free path can then be easily found by following the adjacent, as illustrated in Figure 3.4.

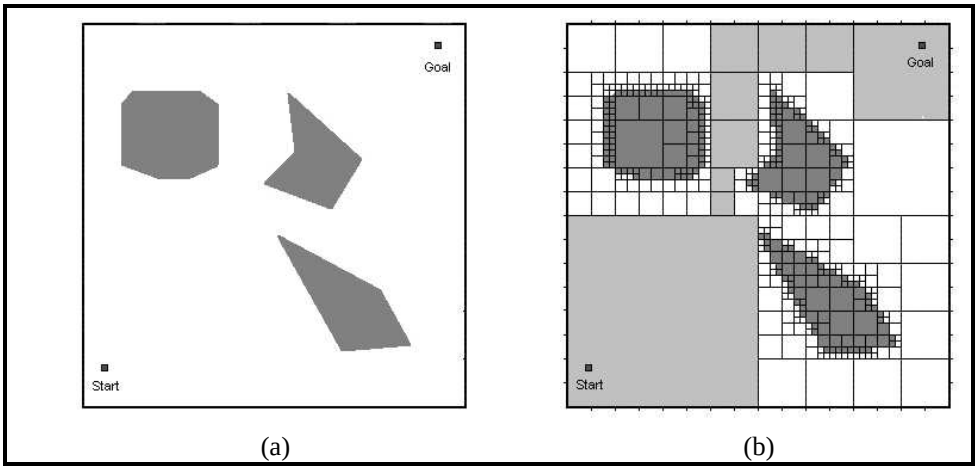


Figure 3.4: Approximate cell decomposition. (a) Cspace, (b) approximate decomposition and the free path

Approximate cell decomposition is not as expensive as exact cell decomposition, and can yield similar, if not exactly, the same, results as those of the exact cell decomposition. However, cell decomposition approaches are not suitable for dynamic environments due to the fact that a new decomposition must be created whenever changes are reported.

3.2.3 Artificial Potential field approaches

The artificial potential field approach is based on a grid representation, by discretizing the space into a fine regular grid of configurations. This approach involves modeling the robot as a particle, moving under the influence of an artificial potential field whose local variations are expected to reflect the structure of the environment. The potential field method is based on the idea of *attraction/repulsion* forces. The attraction force tends to pull the robot toward the goal configuration, whereas the repulsion force pushes the robot away from the obstacles. At each step, the force, generated by the potential function at the current configuration, changes the direction and moves incrementally to the next configuration. The artificial potential field concept was first introduced by Khatib [22] as a local collision avoidance approach, which is applicable when the robot has no prior knowledge about the environment, but the robot can sense the surrounding environment during motion execution.

The major disadvantages of this approach is that the robot can become stuck in a local minimum, since this approach is local rather than a global (i.e., the immediate best course of action is considered). Escaping the local minimum is enabled by constructing potential field functions that contain no local minimum or by coupling this method with some other heuristics technique that can escape the local minimum [23]. The artificial potential field approach can be turned into a systematic motion planning approach by combining it with graph search techniques [4].

3.3 GA Based Path Planning

Although most conventional and graph search approaches provide a good solution (optimal with respect to the traveling distance criterion), they have several disadvantages, the

approach lack the capability to cover all the mobility constraints, and can not be directly utilized in dynamic environment [5].

Since GAs are powerful for searching large and complex spaces [11], a number of researchers have employed them to solve the path planning problems [24]. Yuval Davidor [25] has used a variable-length chromosome representation to solve the path planning problem for an arm manipulator. Davidor has introduced a specialized genetic operator, which he called the *analogous crossover*, to deal with the problems of the order based and variable-length chromosomes. In contrast to the standard crossover operator which determines the corresponding cross points according to their respective position in the genome, the analogous crossover uses the candidate chromosome structure to find the cross point position. The disadvantages of this approach are: (i) it does not operate in the entire working space, (ii) it is not applicable in dynamic environments

Shibata et al. [26] developed a path planner for multiple autonomous robots by using two GAs. Initially, a GA is applied to each robot. Here, the robots explore the solution space, and attempt to find a feasible near-optimal path. This technique is referred to as *selfish planning*. The second GA selects the most explored path(s) by coordinating the robots to avoid collisions and dead-lock, and to maintain the minimum cost path. This is referred to as *coordinative planning*. Shibata et al. have adopted MAKLINK graph, proposed by Habib and Asama [27], to model the environment of the robot. This graph is based on the free-link concept to construct the free space within the robot environment in terms of the free convex areas as shown in Figure 3.5.

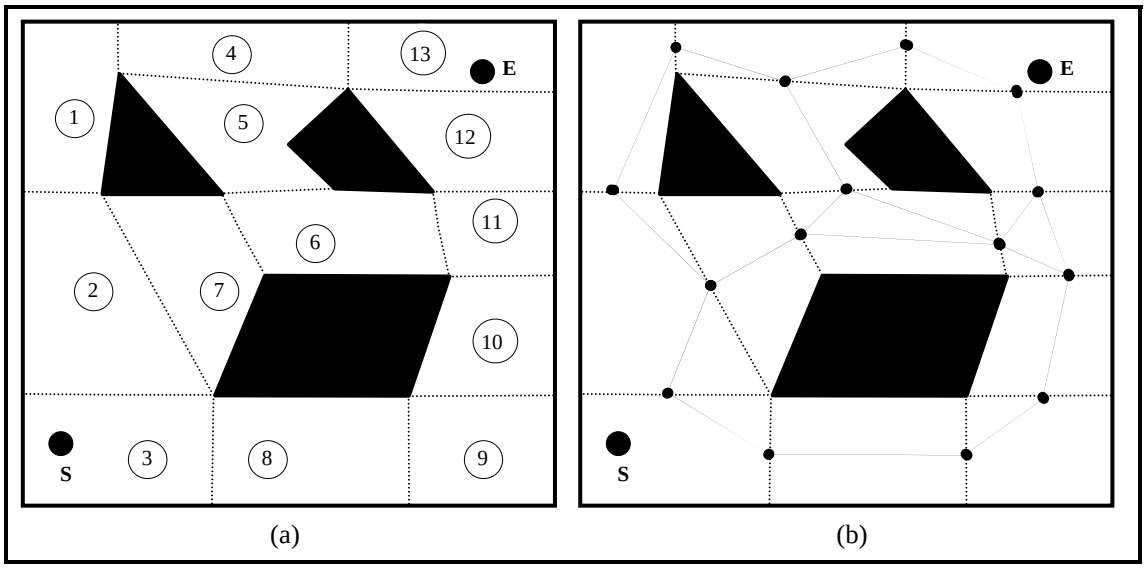


Figure 3.5: MAKLINK Graph: (a) environment with obstacle and free convex area, and (b) a graph representation of the mobile environment

Following the construction of the graph as seen in Figure 3.5 (b), the edge lengths of the graph are used to calculate the fitness. The path selfish planning is encoded according to an order of points so that a path passes each point in the graph only once, and the point is selected randomly. Specialized crossover and mutation operators are implemented to deal with the variable-length encoding. The fitness evaluation is based solely on the minimum path length (i.e., energy, smoothness, and safety are not considered.) Although the GA can obtain an optimal path 22% of 50 runs for the same problem instance, the work has not been further investigated for complex tasks nor in dynamic environments. The same planner was further improved in [28] by adding fuzzy logic for the path evaluation which is referred to as a *Fuzzy Critic*. However this approach does not operate in the entire working space and can not be directly applied in dynamic environments.

In [29], a mobile robot path planning in a 3D grid with dynamic obstacles was investigated. A genetic string of variable-length is used; the robot path is coded as a string of n points, represented by their Cartesian coordinates. The point values are stored in binary

form. This approach uses grids and limits the robot to move in one of eight possible directions, as shown in Figure 2.8a. The individual fitness is computed based on the path length, traversing energy, and traversing time. But this approach does not operate in the entire free space, and does not take into account mobility constraints.

Hoi-Shan Lin et al. [30] propounded the evolutionary navigator (EN) which combines offline and online planning, this was the first publication of this project which was later known as EP/N (Evolutionary Planner/Navigator) as presented in [31]. In contrast with the previous attempts the EP/N is empowered more by domain-specific knowledge and a continuous search map. The EP/N, based on the knowledge that is acquired from sensing the local environment, represents a candidate path with a chromosome representing a path of nodes that are connected by line segments.

The evaluation function is designed to accommodate path length, smoothness, and safety, and a penalty for the infeasible paths; the worst feasible path is fitter than the best infeasible path. Six GA operators are introduced in the EP/N (crossover, mutation 1, mutation 2, insertion, deletion, and swap). Some of these are completely random-based, but others incorporate domain-specific knowledge. Each operator has a performance index [32] that is based on the operator's effectiveness with respect to: the number of times the operator results in improvement per the number of times the operator applied, the required time to apply the operator, and the average time of the increase or decrease gained for all operators on the average change in the number of nodes. This performance index affects the operator's probability, and therefore, the algorithm self-tunes its performance.

EP/N performance on dynamic environments is enhanced by adding memory to the algorithm. Trojanowski et al. [33] used a local memory where each individual has its own

memory buffer. The buffer size is constant during the evolutionary process and initially the buffers are empty. The new generated individual is appended to the buffer if it improves upon the previous solution quality. The memory is recalled in the online process, whenever the robot encounters a new obstacle. Each chromosome and its memory(s) are re-evaluated; if any of the remembered chromosomes are better than the currently active chromosome then they are swapped with the current one to become active. The main drawbacks of the EP/N are that it does not operate in the entire space, and it does not include all the mobility constraints.

Chen and Zalzala [34] applied GAs to solve the path planning problem for a mobile robot with safety criteria. They use the grid method by cell decomposition for the environment representation. Two numerical potential fields in the work space are used: one for the goal which represents the minimum distance value from the goal to a node, and the other is the value from the boundary of the closest obstacle to the node. A chromosome is encoded as a string of variable nodes, represented by their Cartesian coordinates. The numerical potential fields' values are used for evaluating the paths, where safety and traveling distance are considered. A special crossover operator, which is identical to other approaches, is designed to deal with the variable-length coding, however, the mutation differs. A node in a given path is selected randomly with a very small probability then, all the nodes, after the selected node, are deleted and new nodes are randomly generated. The drawbacks of this approach are: it does not operate in the entire space, since it uses a grid by cell decomposition, and it does not utilize domain-specific knowledge.

Gemeinder and Gerek in [35] employed a map that involves binary assignment of the topography's quality. The environments is represented as a grid, where each entry in the

environment is a real value, indicating various amount of energy, which is needed if the robot moved through a specific region. This energy factor must lie in the interval between 0 and 1, for which 1 indicates that this entry cannot be used. The chromosome is encoded as a sequence of movements in these environment grids. The initial population is filled with sinusoidal shaped paths with random amplitudes and bent sides (left and right). After the paths are evaluated according to length and energy consumptions the detour operator is introduced for an active search. The worst feasible path is considered fitter than the best infeasible path. In addition of the disadvantages of the grid representation, this approach does not consider dynamic environments and the mobility constraint are not considered. In addition, the authors have not demonstrated the performance of the algorithm on large benchmark problems.

3.4 Summary

This chapter reviews the common approaches to solve the path planning problem; these approaches have several disadvantages which include their computational complexity and their disability to solve the problem in dynamic environments. The chapter also reviews previous work that utilizes the GA for solving the path planning problem. However, most of these approaches are not scalable and also not suitable for dynamic environments. This thesis will attempt to introduce a methodology that is capable of solving the problem effectively and efficiently.

Chapter 4

Genetic Algorithm Planner (GAP)

In the previous chapters, several techniques to solve the path planning problem were introduced and Genetic Algorithm was a candidate technique to solve the problem efficiently. In this chapter, a description of the proposed Genetic Algorithm Planner (GAP) is given. The analysis of the results in both static and dynamic environments is presented. Finally, the details and the results of the implemented Island-based GA (IGA) are presented in this chapter.

4.1 Problem Definition

Given a point mobile robot in a two-dimensional environment with stationary obstacles find a path for the mobile robot to move from its current position to its final destination. The path should be:

- **Collision-free:** there should be no collision with any obstacles.
- **Short:** i.e., the traveling distance is the minimum.
- **Safe:** a maximum clearance distance should be adhered to.
- **Smooth**: there should be a minimum curvature.

4.2 The Algorithm

In this section, a complete description of the algorithm is presented, including the presentation strategy, fitness evaluation, evolutionary operators, replacement strategy and termination criteria.

4.2.1 Environment Representation

In this thesis, the polygonal representation is selected for the following reasons: (i) it provides a good approximation of the environments [5], (ii) It requires less space (memory) with respect to the grid representation, (iii) it provides flexibility for generating the shorter, and smoother paths, and (iv) path feasibility can be checked by using efficient and simple geometric algorithms [36,37].

The environment is rectangular (usually referred to as the map boundary) with polygonal obstacles that are represented by the ordered list of its vertices. As displayed in Figure 4.1 Obstacle segments are constructed by connecting these vertices, starting with the first vertex and ending with the last vertex; Thus, the number of line segments and the number of vertices are equal for any given obstacle.

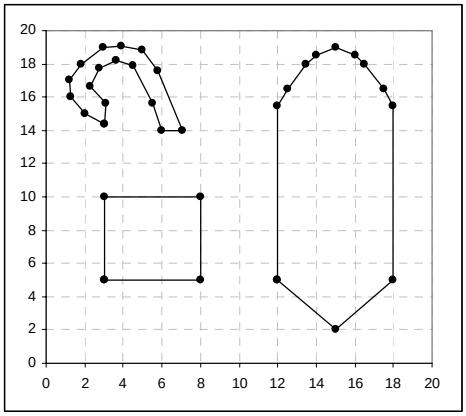


Figure 4.1: Obstacle representation

4.2.2 Chromosome Representation

As illustrated in Chapter 2, the chromosome representation or encoding is one of the most critical issues when a GA technique is used. The solution, represented by a chromosome, is a sequence of ordered positions, starting at the initial point and ending at the goal point. Consequently, different solutions may have a different number of sequences, (therefore, a variable length chromosome, representing a solution, was selected). A linked list data structure is used, as illustrated in Figure 4.2.

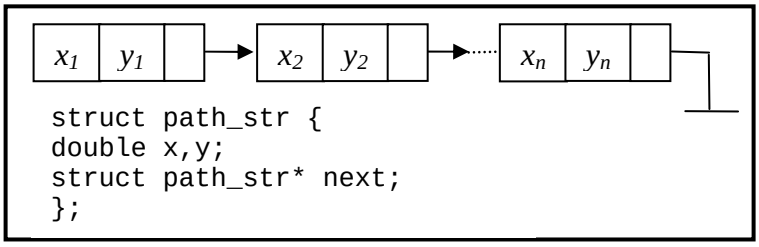


Figure 4.2: Chromosome representation

Each gene in the chromosome contains the x and y coordinates of the path node. The first node contains the starting point coordinates, or the robot's current location, and the last node contains the goal point coordinates. The number of intermediate nodes (knot nodes) is variable. Figure 4.3 shows an example of the different paths for the same task.

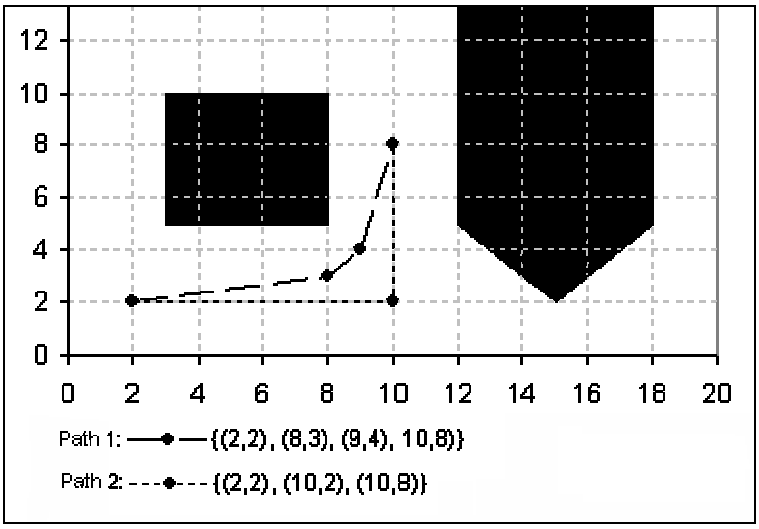


Figure 4.3: Two paths generated for the same task.

In this dissertation, a floating point data type is used for the x, y coordinates. Although floating point operations are more complex than integer and binary ones, the former gives more flexibility for searching the entire space and accommodating the changes in the robot location. As a result, when navigating errors in the robot position occur the algorithm can easily adapt to the current location with no additional map adjustments.

At this stage, it is useful to introduce a terminology chart to familiarize the reader with the terms that are used for this thesis.

Technical Term	Description
Free space	The space in the environment with no obstacles
Occupied space	The space in the environment occupied by an obstacle
Infeasible point	A point inside the occupied space
Feasible point	A point outside the occupied space
Infeasible segment	A line segment intersecting with an obstacle
Feasible segment	A line segment not intersecting with any obstacle
Infeasible path	A path has at least one infeasible segment or one infeasible node
Feasibility ratio	The ratio between the number of feasible paths and infeasible paths
Knot nodes	Intermediate nodes in the path
Collision number	The number of obstacles intersecting with the path

Table 4.1: Algorithm terminology.

4.2.3 Initial Population

The initial population is generated randomly. The number of nodes in any given path is assigned arbitrarily, and bounded between three and the average number of vertices in the environment. The knot nodes coordinates are also generated randomly and only feasible nodes are chosen. Figure 4.4 shows a sample initial population.

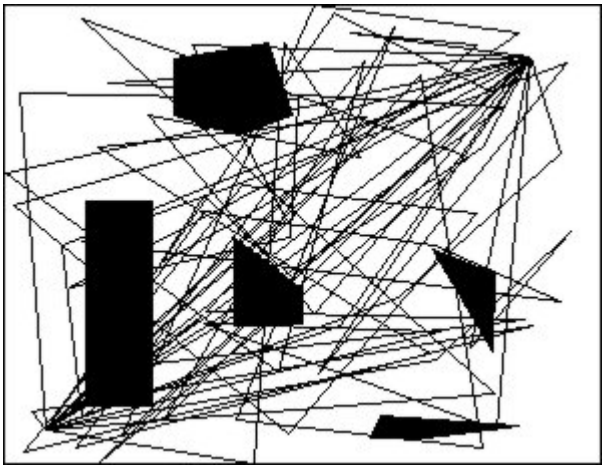


Figure 4.4: Randomly generated paths.

4.2.4 Path Evaluation

The value of the objective function determines the path cost of a chromosome; therefore, all the objectives of the problem must be taken into account. Since a chromosome can be either feasible or infeasible, two evaluation functions are designed. The evaluation functions, introduced by [31], are utilized here with modifications.

Feasible Path Evaluation

To achieve a good approximation of the feasible path cost, safety, time and energy are considered to be the primary factors. A path that is located away from the surrounding obstacles is considered to be safe. Shorter paths are executed faster and require less energy.

Most mobile robots need to consider smoothness as a constraint because of the bounded turning radius. Smoothness can also be included as a factor in the time and energy objectives, because smoother paths are executed in less time and consume less energy. A feasible *P* with n nodes is evaluated by a linear combination of these factors as follows:

$$\cos t_f(p) = w_d \,.\, dist(p) + w_s \,.\, smooth(p) + w_c \,.\, clear(p)\,. \qquad (4\text{-}1)$$

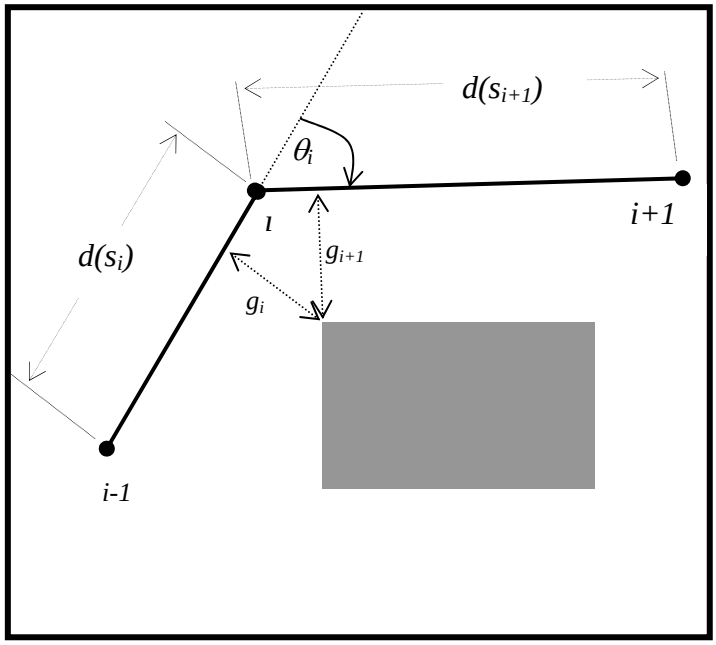


Figure 4.5: Path cost calculations

This is a multi-objective function, where w_d, w_s and w_c represent the weight of each objective to the total cost. The remaining parameters are defined as follows:

1. *dist(p)* is the total path length, and is computed by the following equation:

$$dist(p) = \sum_{i=1}^{n-1} d(m_i, m_{i+1}) \ , \qquad (4\text{-}2)$$

where $d(m_i, m_{i+1})$ is the distance between node m_i and m_{i+1}

2. *smooth(p)* is the path smoothness and it is calculated as follows:

$$smooth(p) = \sum_{i=2}^{n-1} e^{a(\theta_i - \alpha)} \quad , \tag{4-3}$$

where $\theta_i \in [0, \pi]$ represents the angle between the two line segments, connecting the i[th] node. The parameter α is the desired steering angle and *"a*" is referred to as the *map coefficient* which is a problem dependent and is defined as follows:

$$a = MAX\left(\frac{A}{2 * O_A}, 2\right) \tag{4-4}$$

Where A is the total map area, and O_A is total obstacle area. The smoothness and clearance factors increase for simple environments, whereas this coefficient becomes small in crowded environments.

3. *clear(p)* is the path clearance and is calculated as follows:

$$clear(p) = \sum_{i=1}^{n-2} e^{a(g_i - \tau)} \ , \tag{4-5}$$

where g_i is the shortest distance between the i[th] segment and all obstacles, and τ is the desired clearance distance. Figure 4.5 illustrates all the components of the feasible path evaluation function.

Infeasible Path Evaluation

Different strategies can be used to compute the cost of infeasible paths. The evaluation criteria of the infeasible path can include the following:

- the number of intersections with obstacles,
- the ratio between the number of feasible segments and the number of infeasible segments,
- the depth of the intersections,

- the length of the infeasible segments.

Since the fitness evaluation is the most expensive component in GA, the evaluation function for the infeasible paths is designed so that it is easy to compute and give a reasonable ranking to the infeasible paths. The infeasible path is evaluated as follows:

$$\mathrm{cost}_u(p_k) = \mu(p_k) + \eta(p_k) + \max_{p_j \in F} \mathrm{cost}_f(p_j), \qquad \textit{(4-6)}$$

where $\mu(p_k)$ is the total number of intersections with obstacles, and $\eta(p_k)$ is the average number of intersections for each infeasible segment. The last factor in equation (4-6) represents the worst feasible paths, and is included to make the latter better than the best infeasible path. By adding this factor the feasible path are fitter than any infeasible path and therefore, feasible paths are more likely to be selected for recombination. Figure 4.6 signifies two different infeasible paths whose cost values is computed as follows:

$\mu(p_1) = 4$, $\eta(p_1) = 2$; i.e., $Cost(p_1) = 6$ and

$\mu(p_2) = 2$, $\eta(p_2) = 2$; i.e., $Cost(p_2) = 4$.

Therefore, P_2 is fitter (it has less cost) than P_1, in other words, P_2 has a lower level of infeasibility, and it is more likely to generate a feasible offspring. This example is introduced to illustrate the feasibility/infeasibility concept; however, there might be scenarios where a more infeasible path is more likely to be feasible.

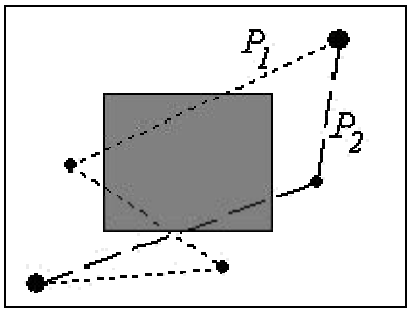


Figure 4.6: Two infeasible paths

4.2.5 Genetic Operators

This section describes the six operators used in the GAP: selection, crossover, mutation, repair, shortcut, and smooth.

Selection Operator

Since the population can contain both feasible and infeasible paths whose cost is always greater than all the feasible paths, the tournament selection method is considered as the selection strategy. With this method, an infeasible path still has a good chance to be chosen for recombination and is less expensive than other selection schemes. The tournament size in this implementation is two (i.e. the binary tournament selection).

Following the selection process, five operators are used to evolve the selected paths. The application of each operator is controlled by a certain probability.

Crossover Operator

The crossover operator is primarily responsible for the improvements in fitness by recombining two solutions into two new paths. Since the chromosome length for this problem is variable, there can be a different number of nodes in the two parents. Therefore,

traditional crossover operations cannot be used. The crossover operator is designed as follows. Each of the two parents (P_1, P_2) is divided into two parts. The first part of P_1 is combined with the second part of P_2, and the first part of P_2 is combined with the second part of P_1. This operation is illustrated in Figure 4.7.

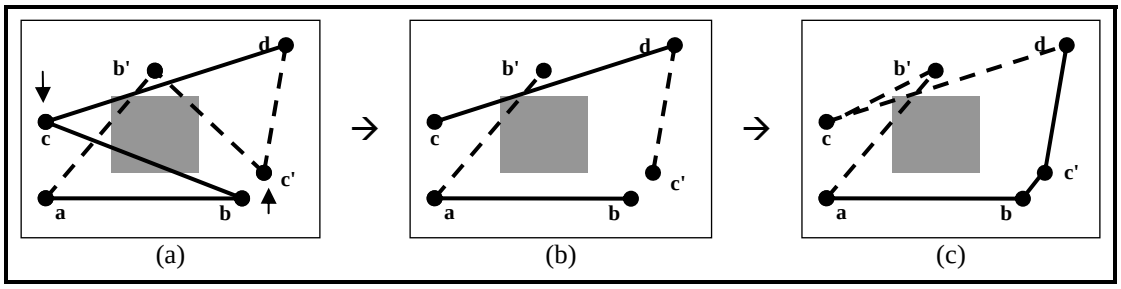


Figure 4.7: Crossover operation. (a) Crossover positions, (b) Divided paths, (c) New combined paths.

Mutation Operator

Mutation, viewed as a background operator, has a secondary role to recover genes lost by the crossover operator [10]. However, mutation plays a more important role to fine tune and further explores the solution space simultaneously [38].

This operator is applied to both feasible and infeasible paths and is implemented as follows. The coordinates of the selected node (x, y) for this operation are changed by (Δx, Δy), as illustrated in Figure 4.8. The change range Δx and Δy is dynamically varied according to the number of feasible paths in the population. If feasible paths do not exist in the population, the new coordinates can be anywhere in the workspace; otherwise, they are changed to the nearby coordinates, and as the feasibility ratio increases, the value of Δx and Δy decreases.

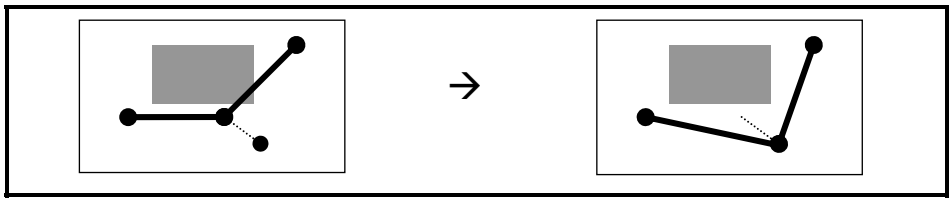


Figure 4.8: Mutation operator

Repair Operator

The application of crossover and mutation operators tends to create improved feasible and infeasible solutions. However, for complex problems, crossover and mutation alone may not be enough to generate a feasible solution. Therefore, a repair operator is introduced to ensure problem feasibility by acquiring problem specific-domain knowledge. The repair operator introduces new knot nodes that render the newly generated segments feasible. Two repair operators are designed namely *random repair* and *exact repair*.

Random Repair:

This operator generates and insets a random feasible point near an obstacle based on the following criteria:

1. At least one of the two new line segments is feasible;
2. The number of infeasible segments is reduced; and
3. the number of collisions is reduced.

This operation is repeated, until all the segments become feasible, as illustrated in Figure 4.9, or the number of the inserted nodes exceeds the maximum allowable number set. The maximum allowable number is relative to the problem size, and is set to the average number of vertices per obstacle.

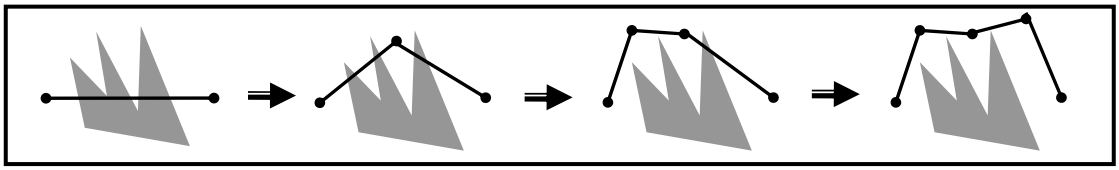


Figure 4.9: Random repair operator

Exact Repair:

Although the random repair operator can be effective in providing feasible solutions, it fails to repair paths for complex problems (e.g., a maze). Consequently, an exact repair operator is introduced to manoeuvre around an obstacle by tracing the obstacle vertices, as depicted in Figure 4.10.

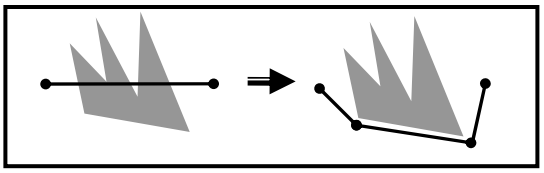


Figure 4.10: Exact repair operator

Shortcut Operator

This operator, illustrated in Figure 4.11, is applied to both feasible and infeasible paths. The objective is to delete the intermediate node (or nodes) between a selected pair if the line segment that is connecting that pair is feasible. When this operation is applied to feasible paths, the deletion is accepted if the new line segment curvature is better than the worst curvature in the initial path. This condition is set so that the shortcut operator does not override solutions obtained by the smooth operator.

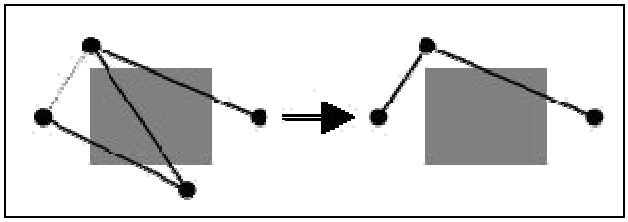


Figure 4.11: Shortcut operator

Smooth Operator

This operator smooths the feasible path by cutting the corners of sharp turns. The operator inserts two new nodes on the path segments, connected to the selected node, and the latter node is deleted, as indicated in Figure 4.12.

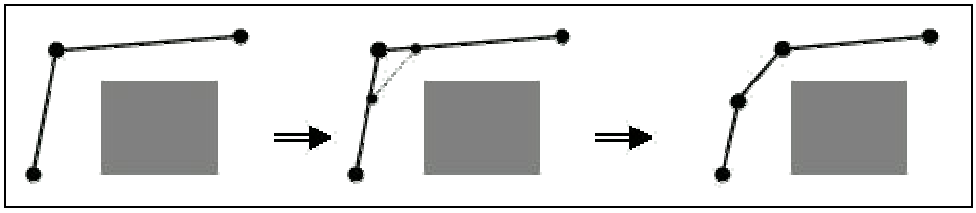


Figure 4.12: Smooth operator

4.2.6 Replacement Strategy and Stopping Criteria

For the replacement strategy, two techniques are investigated: the parent replacement and the worst replacement. In the parent replacement, the new offspring replace their parents, whereas in the worst replacement the new offspring replace the parent if the new offspring is fitter.

For the stopping criteria, the algorithm is terminated either after a given maximum number of generations, or if the algorithm fails to converge to better solutions during a given number of generations.

4.3 Dynamic planning

Optimization in a dynamic changing environment is a difficult problem. The change in the environment is more likely to alter the solution landscape, and therefore, reposition the global optimum. The key to successful GA implementation for dynamic problems is ensuring population diversity during the search process [39]. Since GA might converge to a single sub optimal solution and therefore, losses its explorations capability. This diversity can be guaranteed by using one of these techniques: introduce new random solutions during the search, run the algorithm with a low convergence rate, and use a memory to recall previously visited solution(s). These techniques are utilized in this research and four approaches are implemented.

1. Random Immigrants (RI)

In this approach, diversity is ensured by injecting randomly generated individuals to the population at a fixed rate. The Random Immigrants (RI) approach may seem impractical, because newly generated individuals cannot survive in a population that has some super fit individuals. However, the binary tournament selection is used in this implementation so that unfit chromosomes still have a good chance of being selected for recombination.

2. Low Convergence (LC)

If the GA converges to a sub optimal solution quickly, the algorithm loses its exploration capability. A Low Convergence (LC) refers to running the GA with low crossover rate and high mutation rate respectively. However, the strategy used in this approach is based on having a high mutation range (Δx *and* Δy) independent of the population feasibility.

3. Memory (M)

In general, the Genetic Algorithm retains a set of individuals in a certain generation. After each evolution process the GA replaces the current individuals with newly evolved individuals. As time proceeds the GA loses all previously visited solutions and may become stuck in a local minimum. Memory (M) forces the GA to recall the visited solutions during the search process. The main problem with this approach is that the information that the GA needs to memorize is not identified, nor is the size of the memory determined. For many types of combinatorial optimization problems, a special memory needs to be designed to efficiently memorize the visited solutions.

For the path planning problem, the topological diversity introduced in [40] is utilized here. The concept of topological diversity among paths is simple. Two feasible paths are considered similar, if and only if the polygon, formed by these two paths, is a collision-free polygon (i.e., no obstacles lie inside the polygon) as illustrated in Figure 4.13. The similarity is checked by forming a polygon ψ using the two paths being tested, and verifying the collision of ψ with all the obstacles in the environment. Each obstacle in the environment is tested by checking only one of its vertices for a collision with ψ. If the selected vertex is inside ψ, then a collision with this obstacle exists, and these two paths are considered diverse. However, if none of the obstacles collides with ψ, then the two paths are similar.

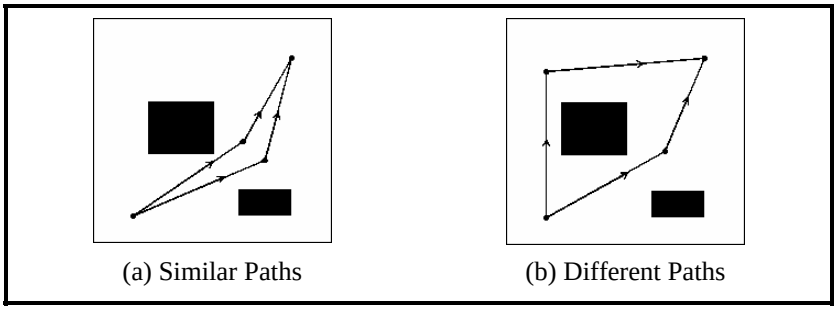


Figure 4.13: Path similarity

After grouping the paths, only the best path of each group is memorized, and those memorized paths are re-evaluated and injected, whenever a change occurs in the environment.

4. Memory with Random Immigrants (MRI)

Combining Memory with Random Immigrants (MRI) gives the GA more balance between exploration and exploitation. This technique is investigated by adopting different frequency rates for the memory recalling and the random immigrant's injection.

4.4 Experimental setup

The GAP is implemented in the C programming language and compiled under Solaris/Windows operating systems. Unless it is indicated, all the runs were conducted on HP workstation x2100, which has an Intel Pentium processor running at 2.4 GHz with 1 GBytes of main memory. This workstation is running Microsoft Windows 2000 Professional. Two ASCII input files are used: the benchmark file, which has the all problem attributes, and the task file which has the task attributes and the algorithm input parameters. The benchmark file structure is described in Appendix A.

Since the GA depends heavily on randomness, a robust random number generator is essential. The *Mersenne Twister*[1] is recognized as one of more robust pseudo-random sequence generators; accordingly, it is used in this dissertation.

1 *http://www.math.keio.ac.jp/home2/matumoto/public_html/emt.html*

The random number generator depends on a number called the *seed* (used to set the random starting point for generating a series of uniform random numbers).

In order to obtain a good analysis of the algorithm, a testing set must be defined. The testing set is composed of both simple tasks and difficult tasks. The decision is made to have the same boundary size for the test set, but with different obstacle numbers, sizes and arrangements. The designed set in Figure 4.14 is used to analyze the algorithm. The dynamic environments are simulated by introducing new obstacles during the search process.

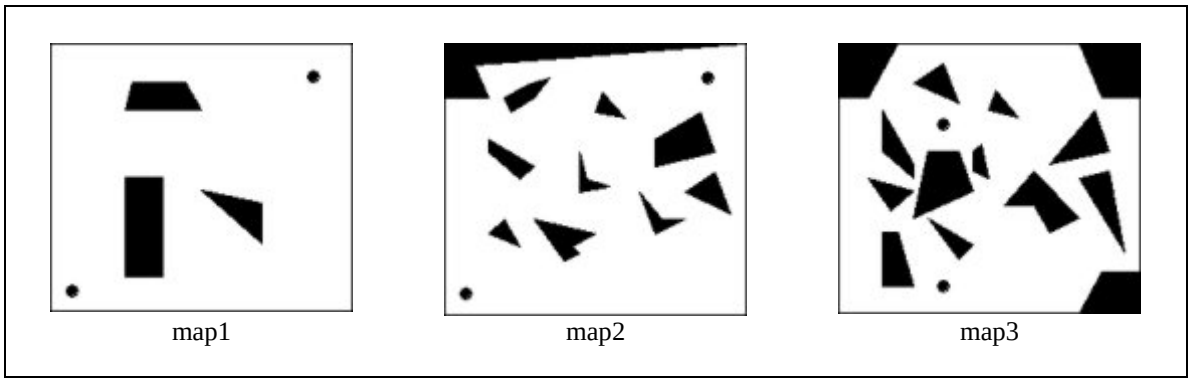


Figure 4.14: Selected Test Set

4.5 Sensitivity of GA Parameters

The objective of this section is to analyze the algorithm behaviour, and fine-tune the GA parameters for further testing. Since the GA is a stochastic technique like all other meta-heuristic techniques, conclusions can not be drawn from a single run. All the results are based on the following: for each configuration, a number of runs (ten unless otherwise stated) are performed for the same configuration, but each run is supplied with a different seed number. The best cost, average cost, worst cost, and the average CPU time are registered for these runs (all the CPU times are in seconds).

4.5.1 Fitness Parameters

In any generic Genetic Algorithm implementation the fitness function is the only part of the algorithm that determines the quality of the search. In this implementation, a multi-objective fitness function that consists of the length, clearance and smoothness of the path are the objectives that form the fitness of the path. The parameters that determine the preferred paths are the weight of each objective (w_d, w_s and w_c), preferred clearance (τ), and the preferred steering angle (α). Since these parameters affect the fitness function (path cost), displaying the path cost as a number will not aid in analyzing the algorithm; as a result, visualization has been selected to show the influence of these parameters at this stage. For each run, the best path is displayed for the selected set of parameters.

Figure 4.15 conveys the influence of the preferred clearance on the solution. It is evident form the figure that as the clearance parameter increases, more safe paths are generated as, shown in Figure 4.15(d).

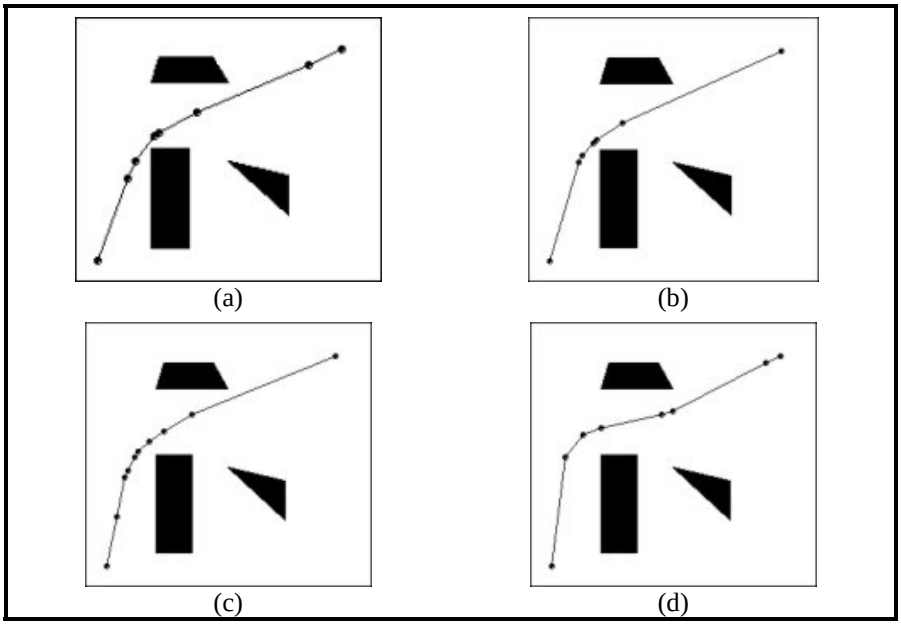


Figure 4.15: Influence the preferred clearance parameter (τ): (a) τ = 0, (b) τ =1,(c) τ = 2, and (d) τ = 4

Figure 4.16 exhibits the influence of the preferred steering angle on the solutions. It is clear from Figure 4.16(d) how a smaller steering angle generates smoother paths.

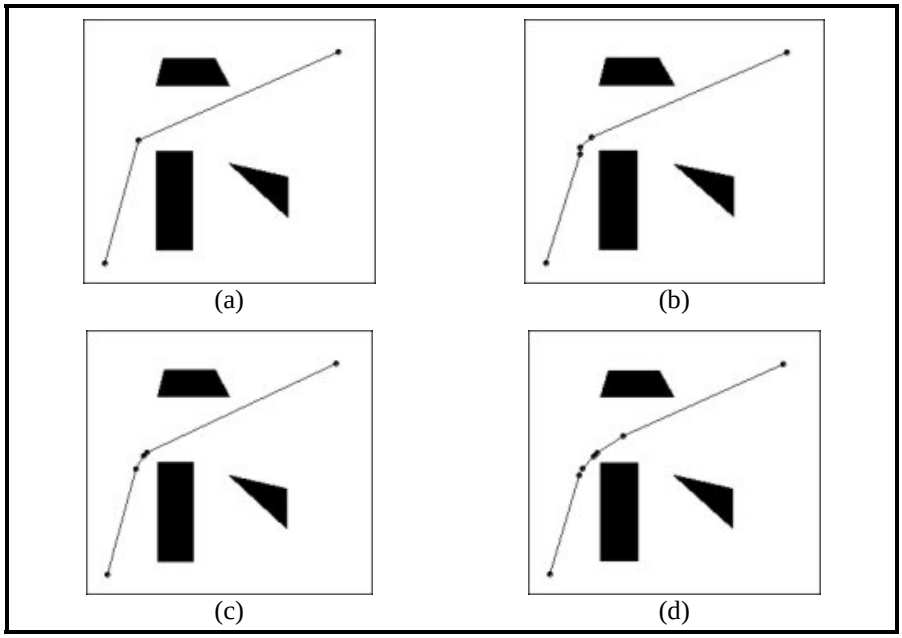


Figure 4.16: Influence of the preferred steering angle parameter (α), (a) $\alpha = 90^{o}$, (b) $\alpha = 45^{o}$, (c) $\alpha = 20^{o}$, and (d) $\alpha = 5^{o}$

4.5.2 Population Size

The population size is one of the main critical factors that determine the performance of any GA implementation. Obviously, when the population size is large, the possibility of obtaining a better solution is increased, However, GA will need additional time and space, thereby increasing the algorithm complexity. Thus a trade off between the solution quality and the solution cost is necessary. There is no rule of thumb that helps in setting the population size in a GA (the population size varies with each type of problem). In most cases though, the population size should be relative to the problem size; as the problem size increases, the population size should increase accordingly (i.e., more individuals are needed to search the large solution space).

Preliminary tests indicate that a small population size (between 10 and 50) provides a high quality solution in a reasonable time. To illustrate how the population size affects the computational time, the selected test set is solved by using different population sizes (4 to 52). The best and the average of 20 runs is computed for each population size. Table 4.2 lists these results for map2. It can be seen that the population size does not affect the solution quality in the same way that it affects the computational time. For all tested problems, it is discovered that after a certain population size, there is no significant gain in the solution quality. However, there is a significant increase in the computation time. This observation is supported by the charts of the population effect for the three benchmarks in Figure 4.17 and Figure 4.18 respectively. The relation between the population size and computation time can be approximately fitted to a linear relation in Figure 4.18. It is clear from the charts that a small number of individuals (from 10 to 30) is reasonable for solving the different benchmarks.

Population Size	Best	Average	CPU Time (Sec)
4	99.20	12526.68	1.39
8	98.38	156.64	1.61
12	80.15	119.51	1.94
16	87.72	117.20	2.42
20	80.79	105.13	2.76
24	86.13	111.31	2.97
28	87.62	107.15	3.11
32	80.07	103.99	3.37
36	80.15	101.09	3.66
40	80.79	92.06	3.77
44	80.07	95.08	4.09
48	80.79	99.33	4.43
52	80.15	97.87	4.95

Table 4.2: Best, average and CPU time vs. population size (map2)

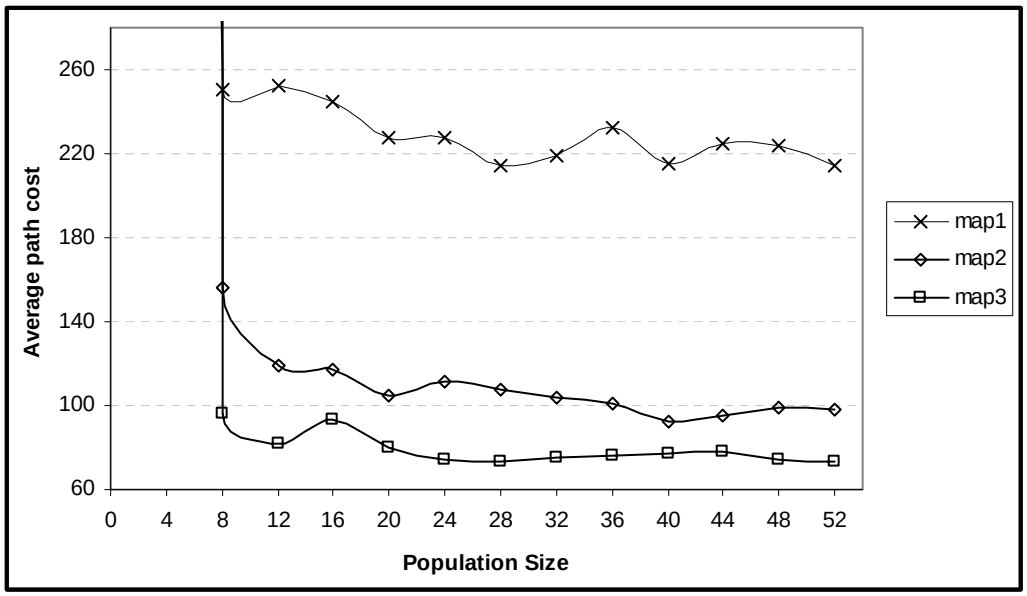


Figure 4.17: Population size effect on solution quality

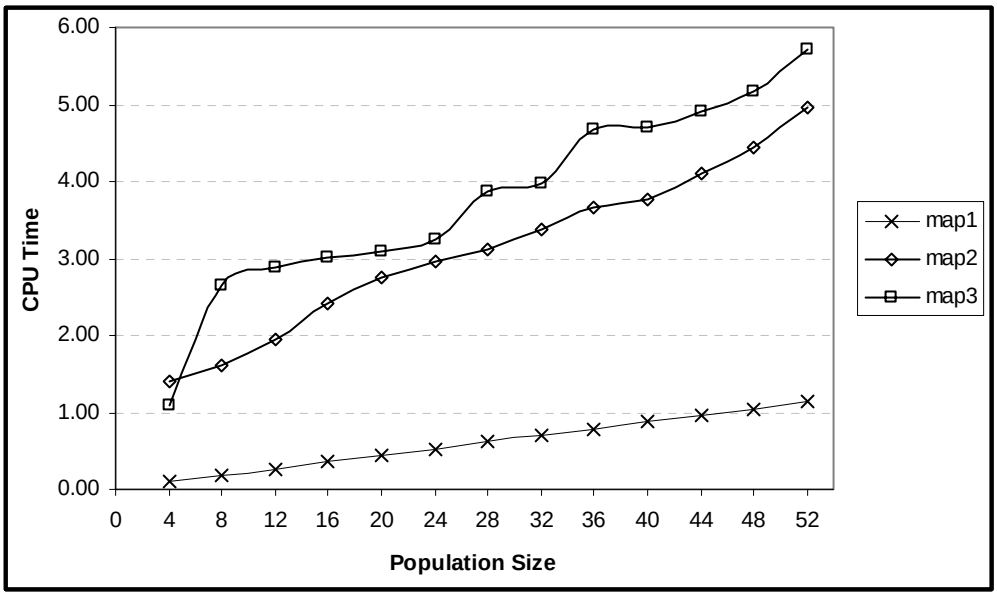


Figure 4.18: Population size effect on algorithm complexity

4.5.3 Operators probability

In any GA implementation it is important to show the influence of each operator on the algorithm performance. The operator's effect on the system is determined by changing the operator's rate while fixing other operators' rates. In this procedure, for each setting the best,

the average, and the worst are calculated for 20 runs by using a fixed termination condition in all the runs.

Crossover and Mutation Operators

Crossover and mutation are standard operators for any classic GA. The effect of these two operators is determined, while all the knowledge-based operators (repair, shortcut, and smooth) are disabled (i.e., their rate value is set to be zero). Each operator's rate is changed from 0.1 to 0.9.

Figure 4.19 plots the crossover rate versus the average path cost of the conducted runs for the three maps; note that in this chart, the average path costs are normalized for each benchmark. Figure 4.19 illustrates how the crossover affects the algorithm performance. In the traditional GA, the general rule is to set the crossover rate at a high value (e.g. 95%). In contrast an intermediate value (50%) yields a better result for most benchmarks.

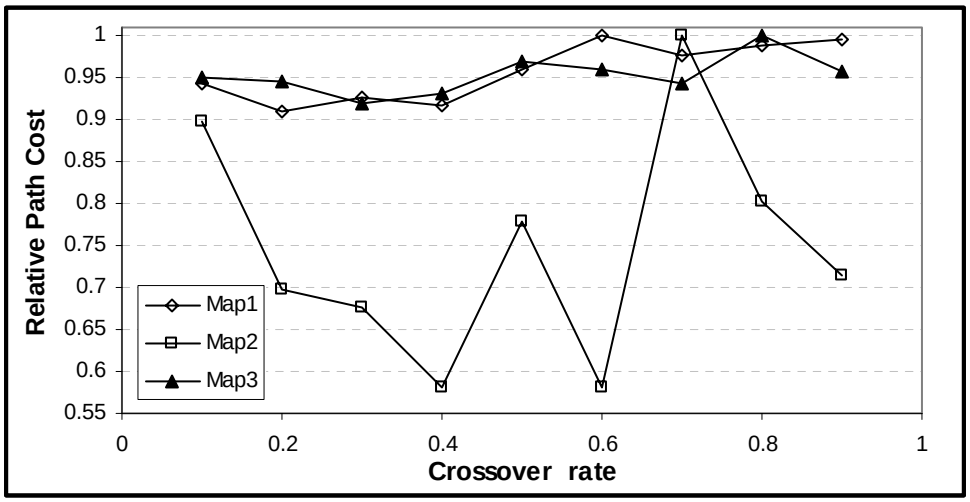


Figure 4.19: Crossover rate vs. average path cost (mutation fixed at 10%)

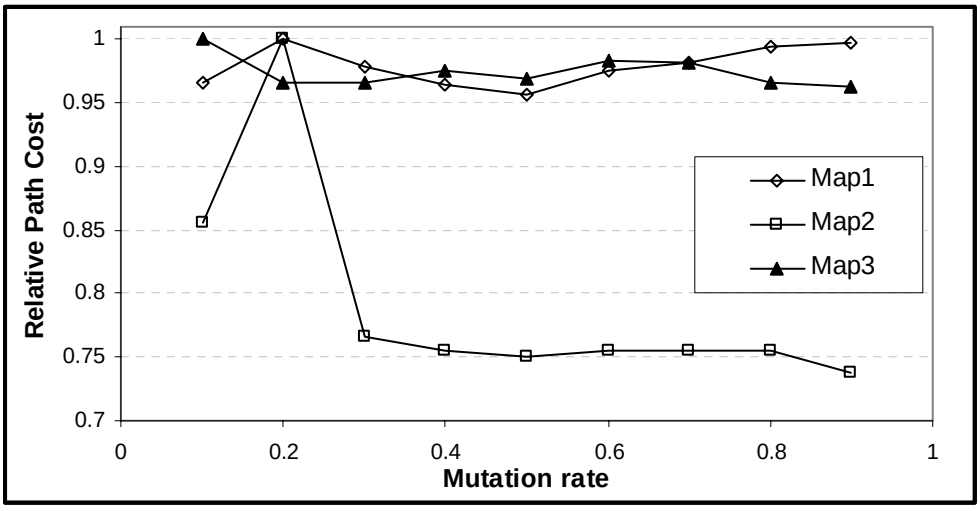


Figure 4.20: Mutation rate vs. average path cost (crossover fixed at 50%)

The effect of the mutation rate on the solution quality is shown in Figure 4.20. It is obvious that a higher mutation rate is more likely to produce better solutions. This can be explained by picturing the population status during the evolution process. Figure 4.21 and Figure 4.22 show the population at different generations for different mutation values, where T refers to the generation number and the crossover rate is fixed at 50% for the two runs. The population diversity is lost at the low mutation rate at which most of the population converges to a sub-optimal path as shown in Figure 4.21(d). On the other hand, at a high mutation rate there is always diversity in the population, as shown in Figure 4.22, which gives the GA more exploration capability. However, the nature of this operator in this implementation is also to fine-tune solutions, as the population feasibility increases. Since a balance between exploration and exploitation is always needed to obtain good quality solutions, the mutation rate was set at 50%.

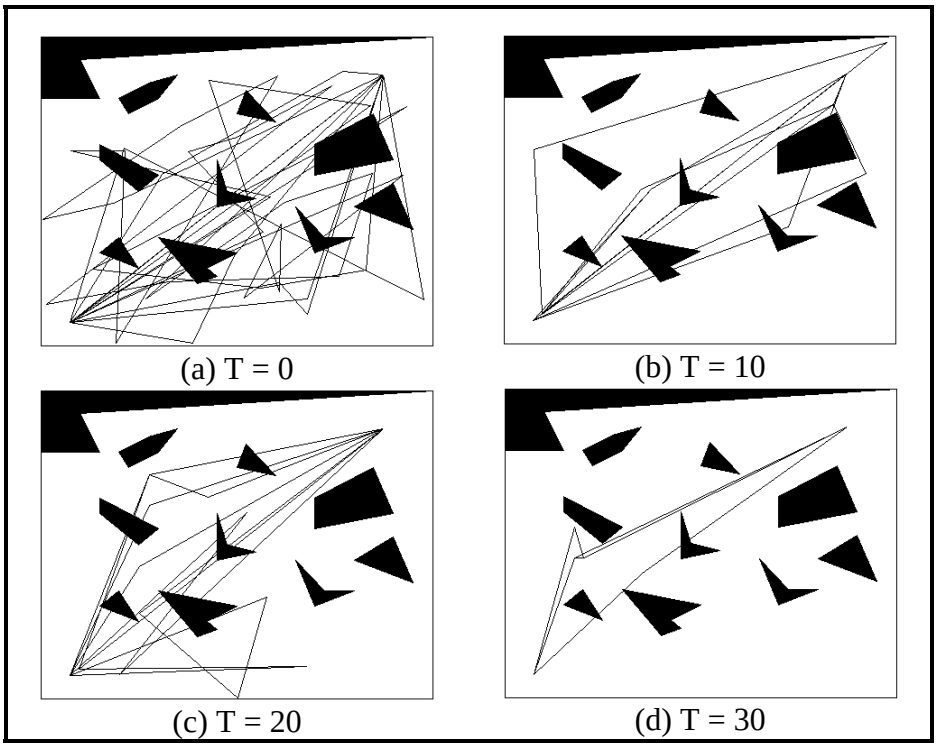


Figure 4.21: Population snapshots, mutation = 10%

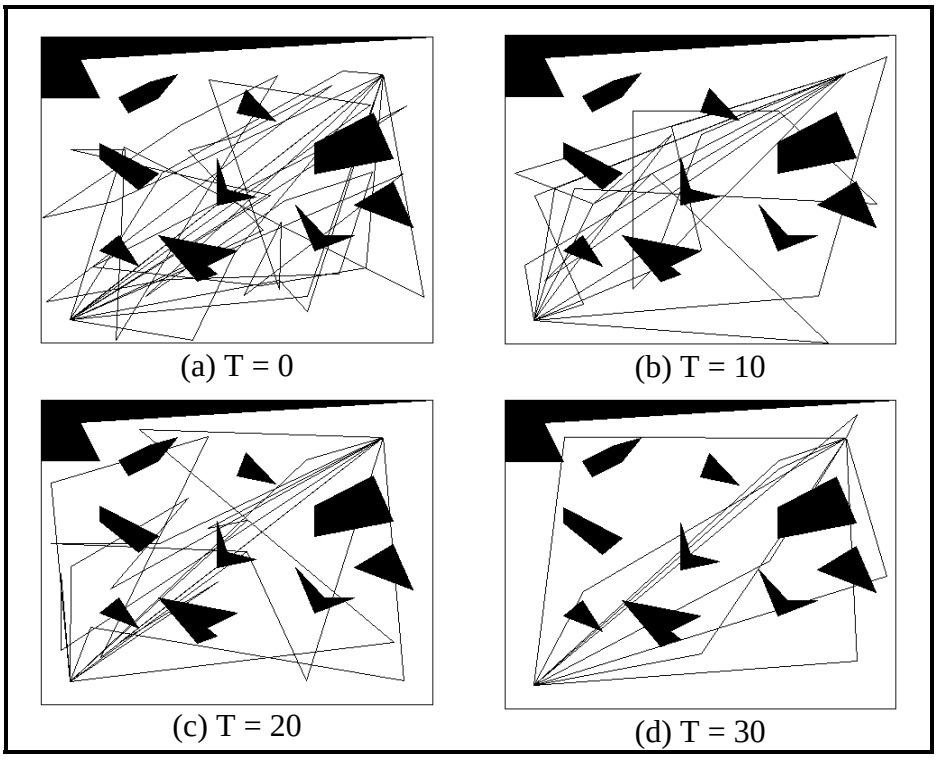


Figure 4.22: Population snapshots, mutation = 90%

Repair Operator

In some complex and structured environments, the GA traditional operators (crossover and mutation) tend to produce infeasible solutions. Therefore, the repair operator is introduced to ensure feasibility. Since the repair operator uses the obstacle vertices to repair any given segments, it can be said that this operator complexity is O(n) where n is the number of vertices. The effect of the repair operator in map3 is recorded in Table 4.3. The average and the best are plotted in Figure 4.23. A convergence curve for the average path cost of the 20 runs for each configuration for map3 is shown in Figure 4.24. The convergence for any of the runs with the repair operator is much faster than the convergence without the repair operator. Figure 4.25 shows the repair rate versus the average gain in the fitness and the CPU time increase. Finally the effect of this operator on the solution quality for the selected benchmarks is highlighted in Figure 4.26.

Repair Rate	Best	Worst	Average	Std	Time
0%	78.27	172.62	113.22	23.60	1.37
10%	68.05	151.88	81.93	26.61	4.23
20%	59.49	108.29	70.10	12.63	14.20
30%	59.49	112.05	73.37	14.59	23.40
40%	61.59	105.42	68.99	8.79	42.00
50%	60.07	68.05	66.01	2.75	43.00
60%	59.71	136.73	69.53	16.05	58.80
70%	59.69	68.05	66.44	2.43	67.30
80%	63.87	96.93	68.82	6.85	71.50
90%	64.13	68.05	66.49	1.95	83.50
100%	63.87	83.58	67.25	4.31	86.10

Table 4.3: Repair operator effect on map3

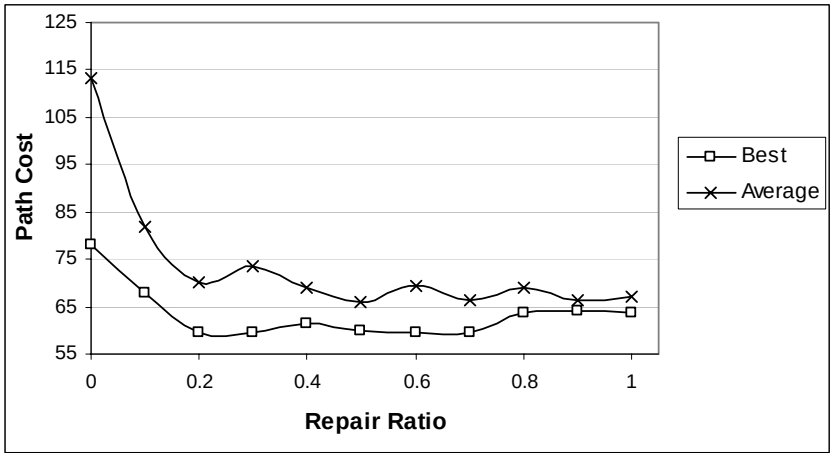


Figure 4.23: Effect of the repair operator (map3)

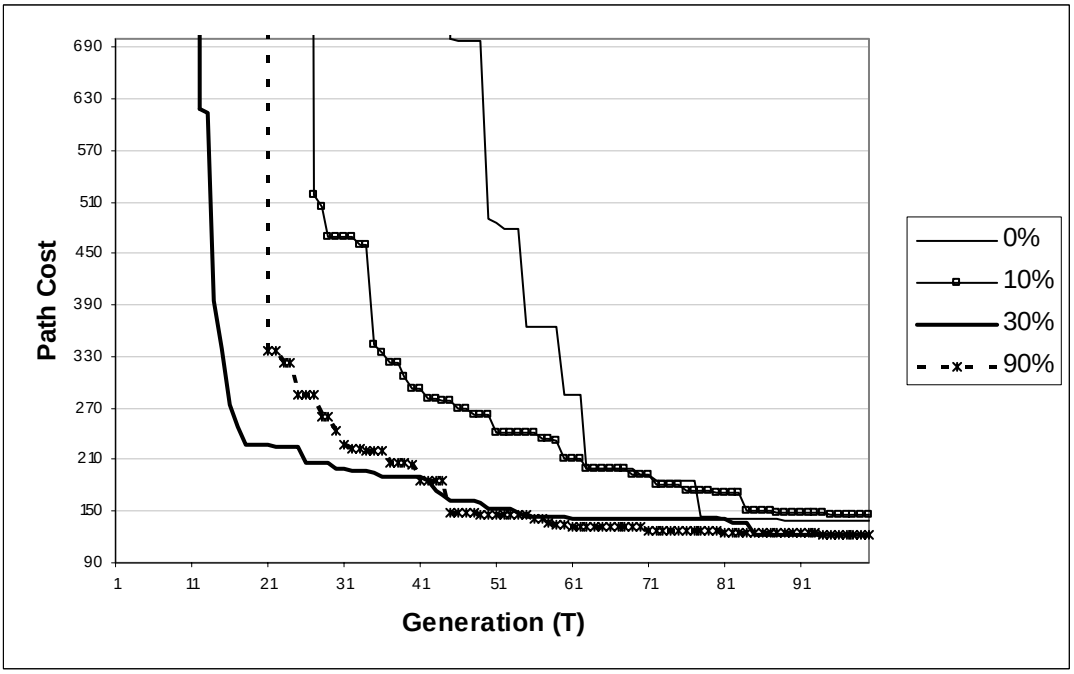


Figure 4.24: Average of best so far for the shown configuration (map3)

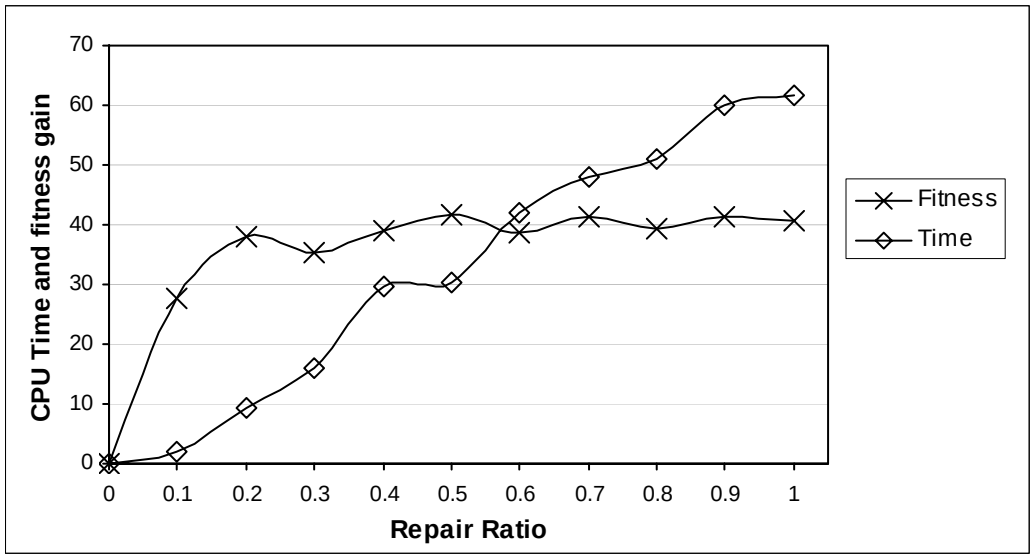


Figure 4.25: Repair effect on the gain of the average cost and CPU time (map3)

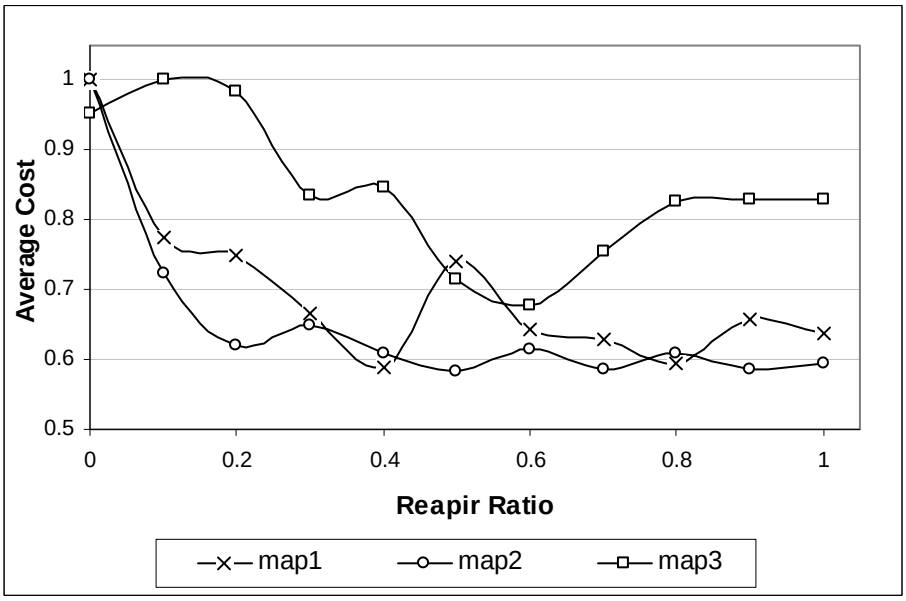


Figure 4.26: Repair operator vs. average cost

From the given data it is obvious that the repair operator affects both solution quality and CPU time. Setting the repair rate at a high value guarantees a higher CPU time and does not guarantee a high quality solution, as can be seen in Figure 4.25. This is attributed to the following: as the population feasibility increases, the exploration mechanism decreases, and

the exploitation mechanism is increased by the mutation operator. Therefore, a repair ratio of 30% is selected as the default value for the algorithm.

Shortcut Operator

This operator deletes unnecessary knot points as illustrated in Section 4.2.5. Figure 4.27 reflects the effect of this operator on the algorithm's complexity, and Figure 4.28 denotes the effect of the operator on the solution quality. The solution quality is measured as the average of the best paths of the 20 runs for each benchmark. With respect to all other operators, the shortcut operator is the only operator that reduces the CPU time, as shown in Figure 4.28. This is possible since the paths that have a small number of knot nodes require less computational time. However, setting the operator rate to a 100% will ultimately effect the applications of the other operators, and the algorithm will tend to converge prematurely and get stuck in a local minimum. Therefore, a value of 70% is selected as the default value for this operator.

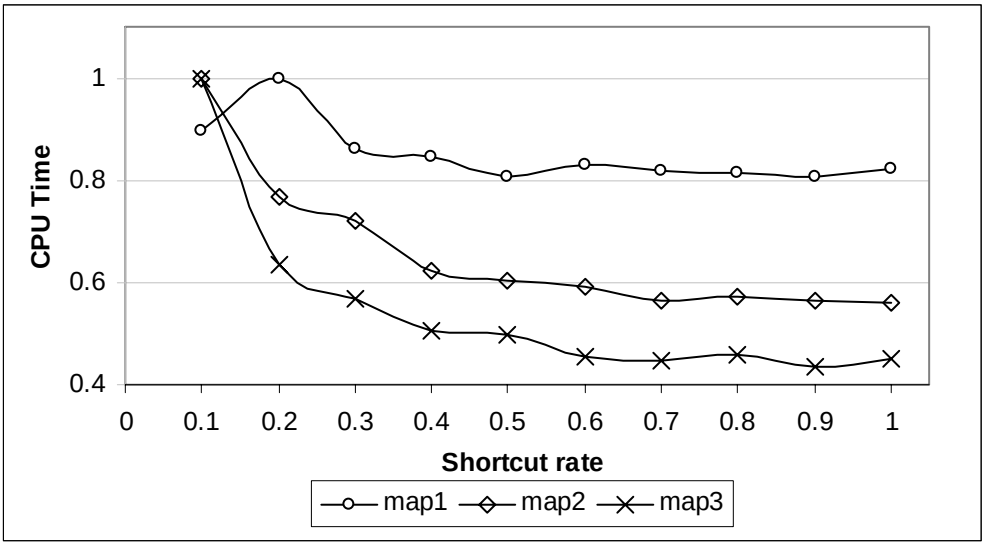


Figure 4.27: Shortcut rate vs. CPU time

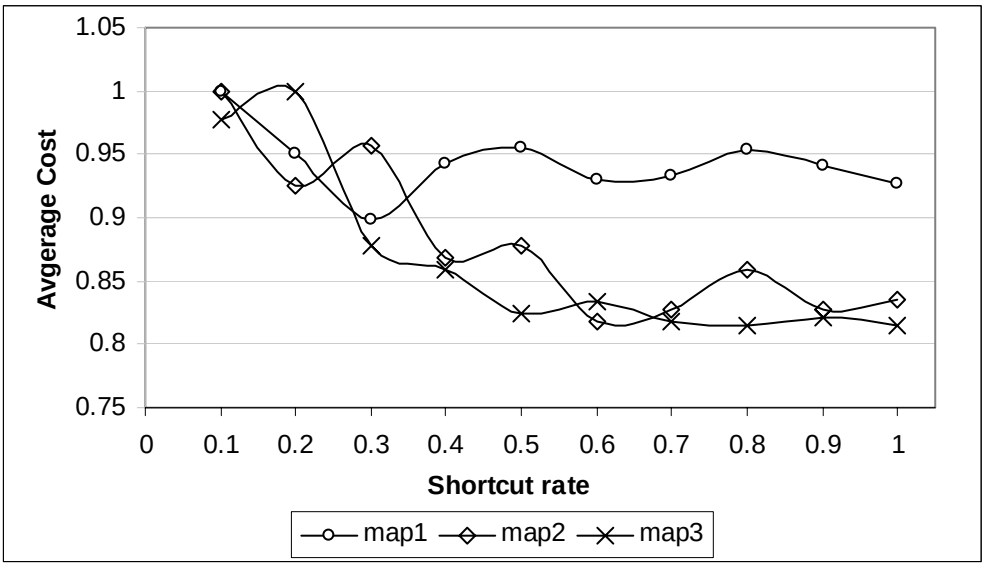


Figure 4.28: Shortcut rate vs. solution quality

Smooth Operator

The smooth operator is responsible for the smoothness of the paths. The complexity of the smooth operator is close to the complexity of the repair operator. Figure 4.29 reveals the effect of the smooth operator on the path cost. It is evident that the effect of this operator decreases as the problem complexity increases. This is due to the fact that in crowded environments, a smooth path is less likely to occur. However, a trade off between the speed and solution quality must be established; and therefore, a default rate for the smooth operator is set at 50%.

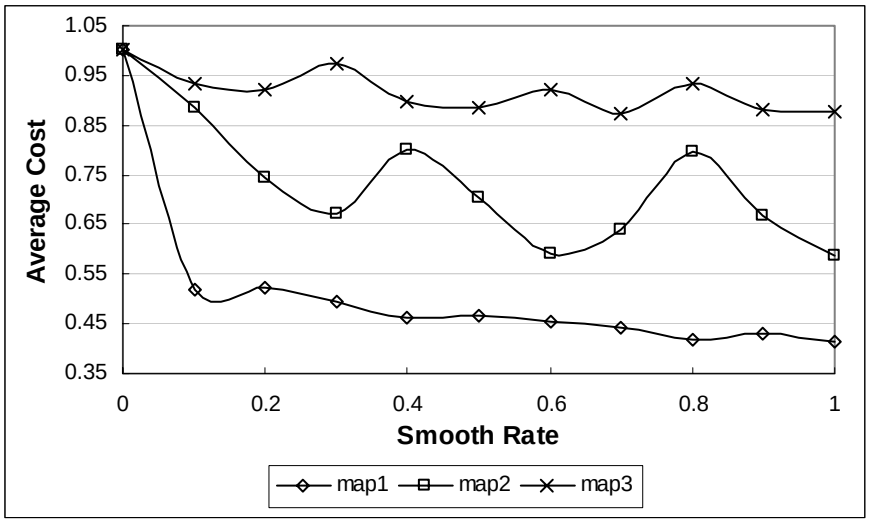


Figure 4.29: Smooth rate vs. solution quality

Replacement Strategy

The results for the two replacement strategies that are implemented are shown in Figure 4.30 and Figure 4.31. The difference between the two techniques is negligible; however, the parent replacement demonstrates the best solutions, and for this reason, this strategy is selected as the replacement technique for the algorithm.

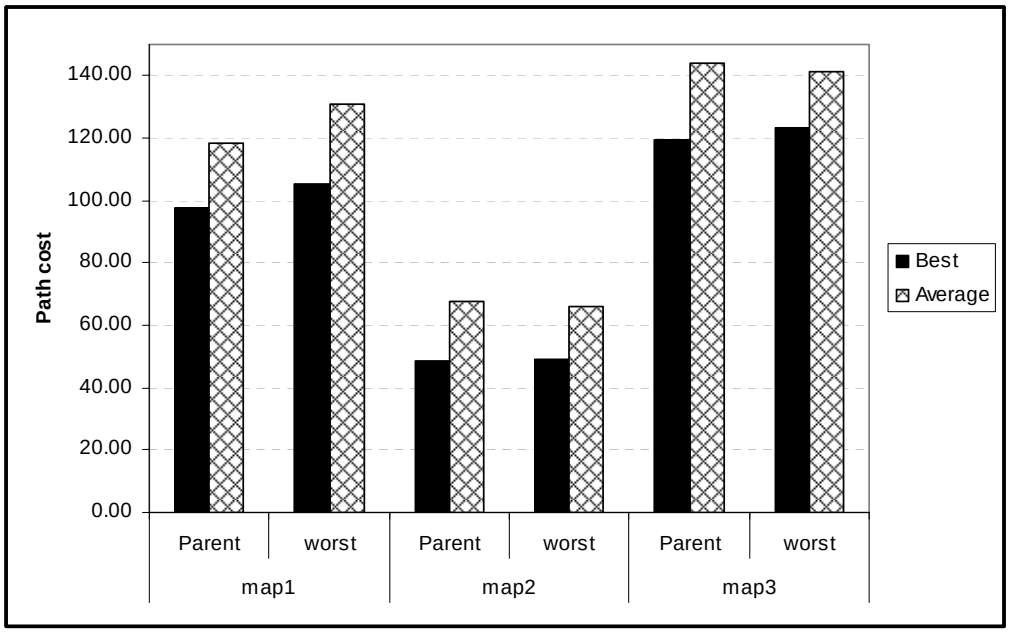


Figure 4.30: Replacement strategy: parent replacement vs. worst replacement

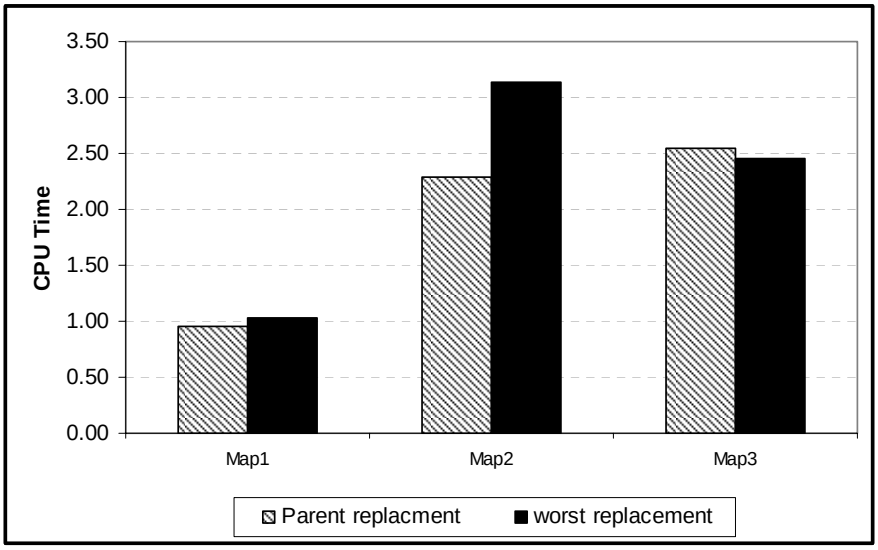


Figure 4.31: Replacement strategy complexity

4.6 Results

In this section, the results of the GAP are obtained for both static and dynamic environments. The algorithm is tested with various benchmarks for a fair judgement. The benchmarks vary from those embedded in simple environments to those embedded in crowded environments. In addition structured and non-structured environments are utilized. Table 4.4 and Figure 4.32 present the benchmarks and their attributes.

As shown in Figure 4.32 Task 1 represents a simple environment, Task 5 is a crowded environment and it is the most complicated benchmark. Task 2, Task 3, and Task 4 are mid range in terms of complexity. Task 6, Task 7 and Task 8 represents structured environments; however, Task 8 is the most complex environment.

The results are obtained by running the algorithm with fixed GA parameters for all the benchmarks. For each benchmark ten runs are conducted; the best path and the worst path are

displayed for comparison. Also the average path cost and the average CPU time per run are reported.

Benchmark Name	Map Boundary	O_N	O_V	A/O_A	S	D	τ	α
Task 1	40x40	3	11	8.31%	3, 3	35, 35	2.0	5.0
Task 2	40x40	10	40	13.5%	3, 3	35, 35	1.0	15.0
Task 3	40x40	14	51	22.66%	14, 4	14, 28	2.5	15.0
Task 4	100x100	6	43	17.72%	20, 50	80, 50	1.0	10.0
Task 5	160x160	24	95	30.44%	150, 5	5, 150	5.0	25.0
Task 6	100x80	5	20	11.00%	10, 40	90, 40	1.5	15.0
Task 7	40x40	3	20	9.50%	14, 33	25, 7	2.0	25.0
Task 8	100x100	1	20	17.25%	45, 50	95, 20	2.5	10.0

Table 4.4: Tasks attributes

Note that the terms used in the above table are defined as follows: $\boldsymbol{O_N}$ is the number of obstacles, $\boldsymbol{O_V}$ is the total number of obstacle vertices, $\boldsymbol{A/O_A}$ is the total map area / total obstacle area, $\boldsymbol{S}$ is the initial robot coordinates, $\boldsymbol{D}$ is the robot final destination coordinates, τ is the preferred clearance distance, and α is the preferred steering angle.

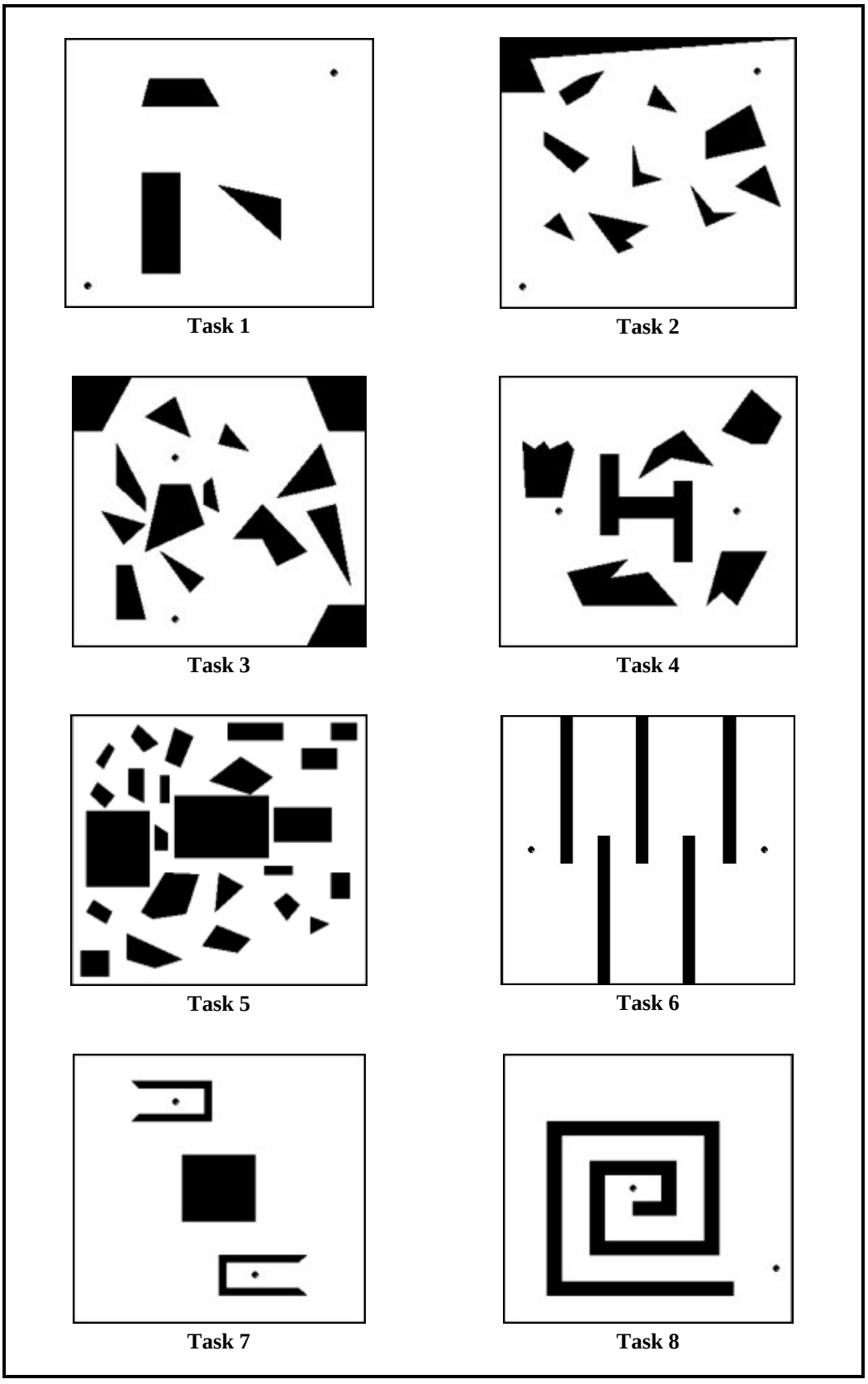
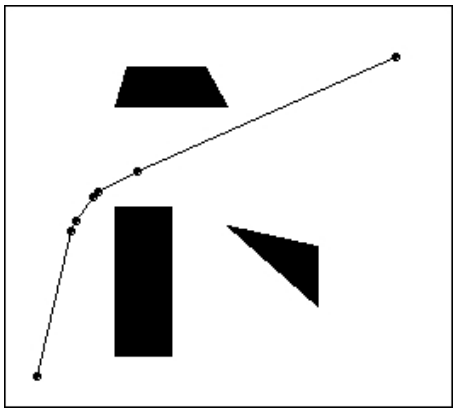


Figure 4.32: Benchmarks

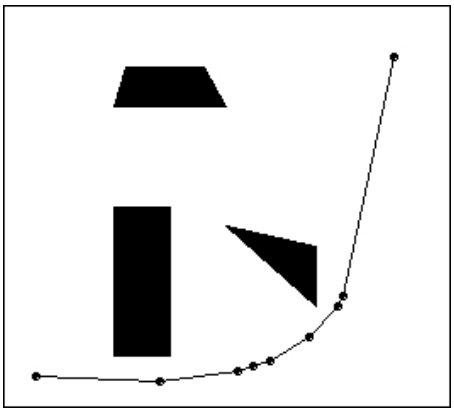
4.6.1 Static Environments

For each benchmark, the best path and worst path are shown along with their path attributes and average CPU time in seconds. These results are depicted in Figure 4.33 to Figure 4.40.



Best Path

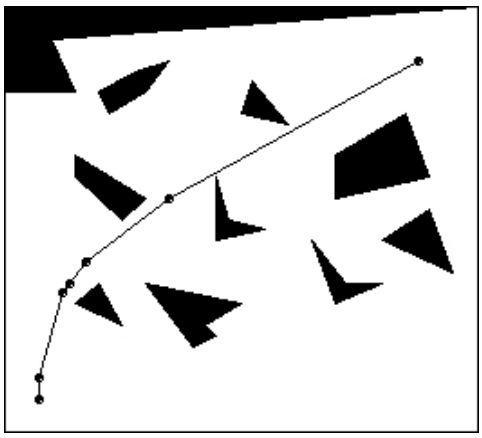
Distance	49.29
Minimum clearance	2.05
Maximum steering	19.44
Clearance value	0
Smoothness value	56.36
Path cost	105.65
CPU Time	0.69



Worst Path

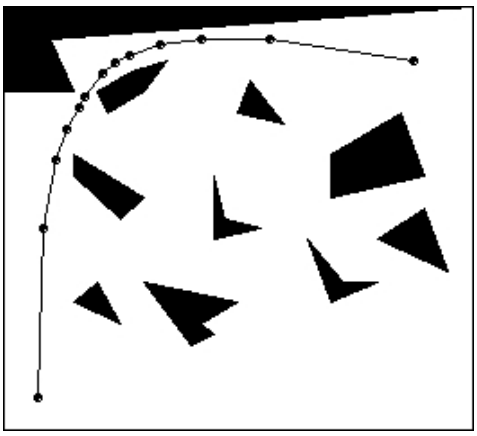
Distance	54.99
Minimum clearance	1.54
Maximum steering	21.80
Clearance value	20.16
Smoothness value	94.92
Path cost	170.06
CPU Time	0.87

Figure 4.33: Task 1 results



Best path

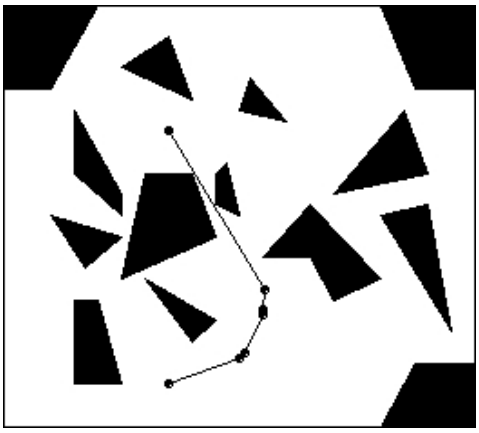
Distance	47.78
Minimum clearance	0.40
Maximum steering	13.06
Clearance value	15.05
Smoothness value	0
Path cost	62.83
CPU Time	2.27



Worst path

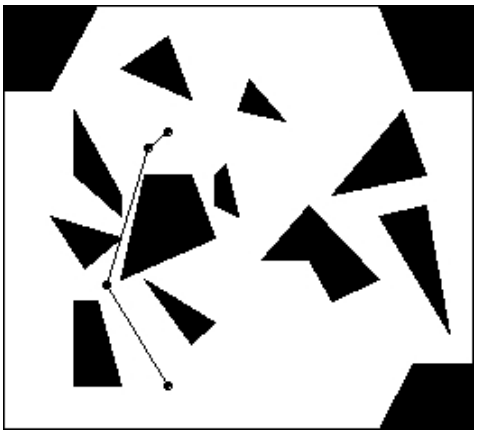
Distance	59.26
Minimum clearance	0.54
Maximum steering	33.69
Clearance value	11.81
Smoothness value	11.36
Path cost	82.42
CPU Time	3.38

Figure 4.34: Task 2 results



Best path

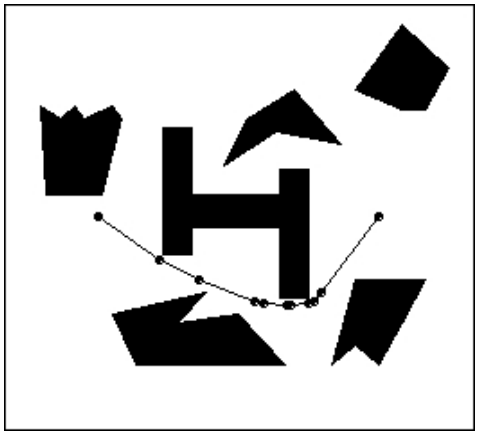
Distance	30.61
Minimum clearance	015
Maximum steering	53.94
Clearance value	66.59
Smoothness value	40.56
Path cost	137.77
CPU Time	2.39



Worst path

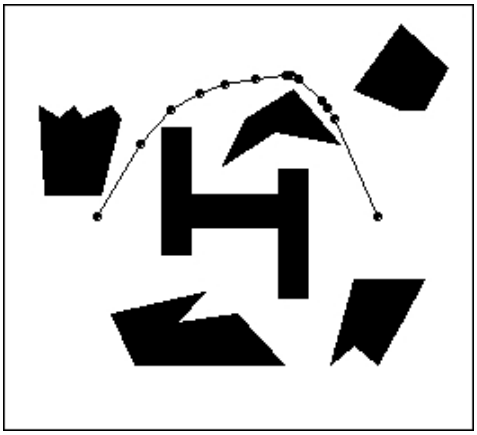
Distance	26.54
Minimum clearance	0.03
Maximum steering	33.44
Clearance value	100.03
Smoothness value	40.17
Path cost	166.73
CPU Time	2.13

Figure 4.35: Task 3 results



Best Path

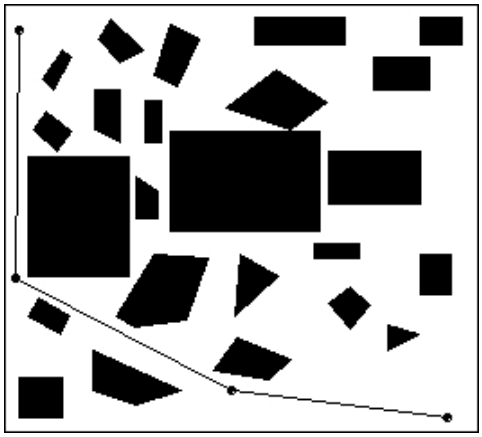
Distance	76.43
Minimum clearance	1.34
Maximum steering	26.57
Clearance value	0
Smoothness value	25.41
Path cost	101.84
CPU Time	4.20



Worst Path

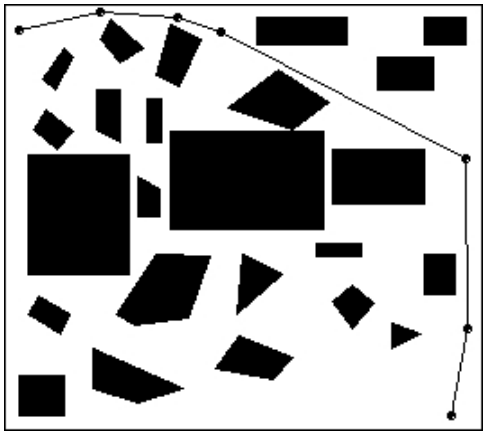
Distance	59.70
Minimum clearance	0.97
Maximum steering	24.78
Clearance value	5.22
Smoothness value	45.22
Path cost	146.13
CPU Time	4.89

Figure 4.36: Task4 results



Best Path

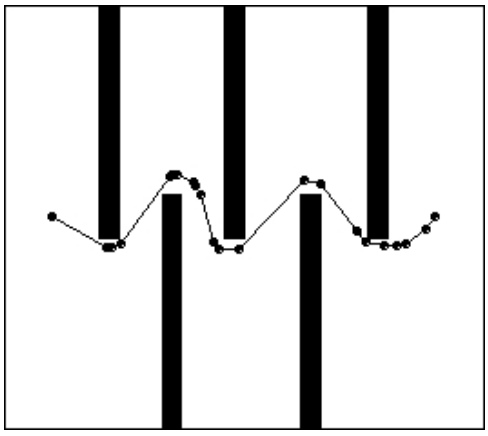
Distance	250.66
Minimum clearance	2.44
Maximum steering	60.36
Clearance value	86.61
Smoothness value	34.42
Path cost	371.69
CPU Time	25.40



Worst Path

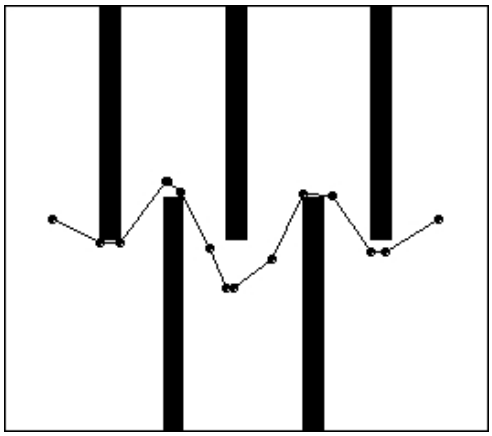
Distance	261.95
Minimum clearance	2.33
Maximum steering	59.31
Clearance value	283.95
Smoothness value	33.48
Path cost	579.25
CPU Time	35.38

Figure 4.37: Task 5 results



Best Path

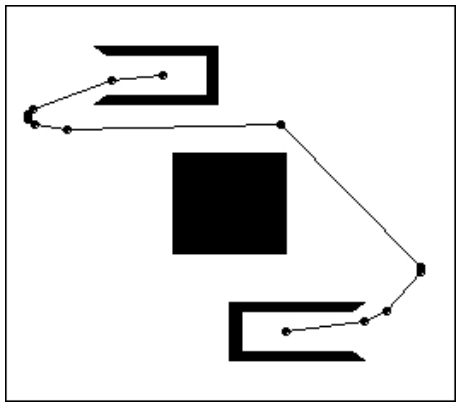
Distance	109.16
Minimum clearance	0.77
Maximum steering	64.80
Clearance value	114.76
Smoothness value	500.00
Path cost	723.92
CPU Time	3.02



Worst Path

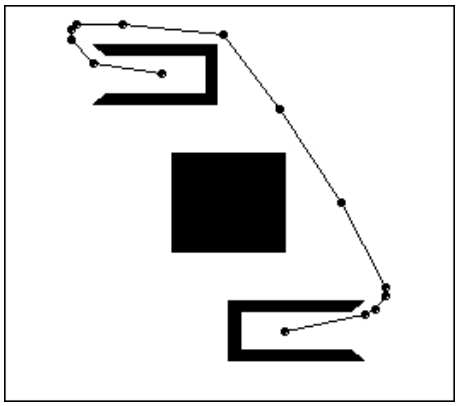
Distance	114.11
Minimum clearance	0.06
Maximum steering	74.62
Clearance value	502.02
Smoothness value	1256.03
Path cost	1872.15
CPU Time	3.10

Figure 4.38: Task 6 results



Best Path

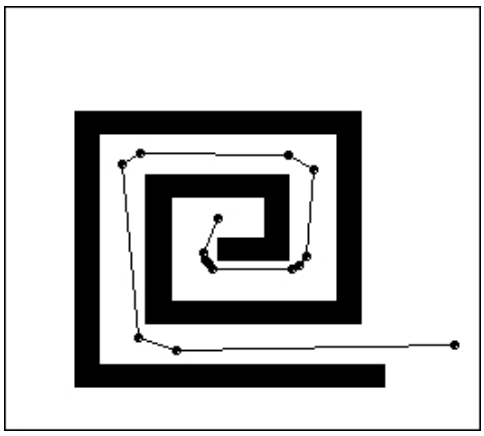
Distance	69.89
Minimum clearance	1.13
Maximum steering	63.43
Clearance value	149.33
Smoothness value	216.73
Path cost	453.95
CPU Time	3.54



Worst Path

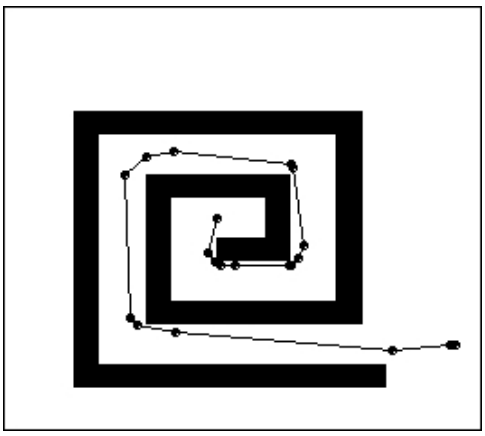
Distance	64.49
Minimum clearance	0.63
Maximum steering	73.88
Clearance value	658.57
Smoothness value	226.72
Path cost	949.78
CPU Time	2.78

Figure 4.39: Task7 results



Best Path

Distance	205.28
Minimum clearance	1.66
Maximum steering	64.80
Clearance value	62.90
Smoothness value	435.64
Path cost	703.82
CPU Time	2.49



worst Path

Distance	192.30
Minimum clearance	0.67
Maximum steering	53.13
Clearance value	456.45
Smoothness value	329.29
Path cost	978.04
CPU Time	2.40

Figure 4.40: Task 8 results

Results obtained indicate that the algorithm is 100% successful to find a reasonable solution for all the tested benchmarks. Also, the results clearly show how the algorithm is able to find a high quality solution in a reasonable computation time (CPU time). However, the computation times for some benchmarks are excessive. It has been found that the complexity of the algorithm depends on the following attributes: the number of obstacles, the total number of vertices, the total area that is occupied by obstacles, and obstacles arrangement. Figure 4.41 displays the relation between the first three factors and the CPU time. It can be observed that the main controlling factor is the total number of vertices in the environment followed, by the total obstacle area with respect to the total environment area.

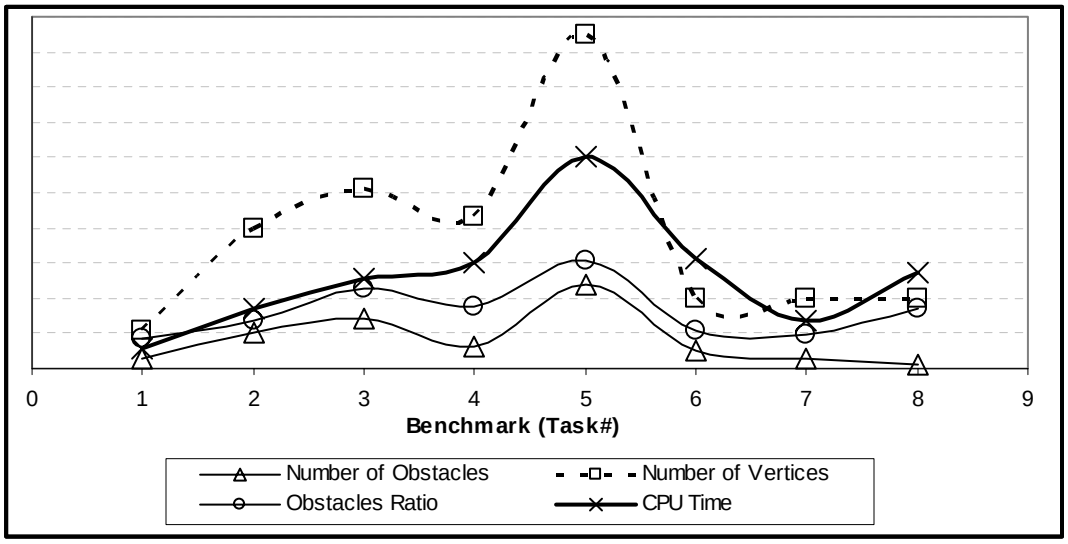


Figure 4.41: Algorithm complexity with respect to the environment attributes

4.6.2 Dynamic Environments

For the dynamic environments, a benchmark is selected to test the four proposed approaches presented in section 4.3. Two scenarios for the dynamic environment are implemented: (i) predefined dynamic obstacles, and (ii) random dynamic obstacles. In the predefined dynamic obstacles, an object is introduced during the run so that the obstacle has an effect on solutions previously obtained by the static mode. In this mode, the dynamic obstacle size and shape are user defined and can assume any form. Each dynamic obstacle is associated with a generation number at which the obstacle can be activated. For the random mode, a newly squared shape generated obstacle is introduced at a fixed number of generations. The randomly generated obstacle's attributes have the following restrictions: the obstacle centroid does not lie inside an active obstacle, and the new obstacle size is approximately the same size as the average size of the existing obstacles. These restrictions are imposed to decrease the chances of generating unsolvable environments.

In order to gain a good understanding of the different investigated techniques, the methodology used here in displaying the results are as follows: first a simple case is taken as a case study, where the population snapshots before and after the environment is changed (turning point) are displayed and discussed for this task. The new obstacle, are displayed at each turning point is filled with a grey colour, the convergence curves are used to determine how each technique deals with the changes in the environments

Case Study:

In this case, Task 1 is considered; new obstacles are introduced at generation 25, 40, and 50. In addition to the basic GAP, this benchmark is solved using the four proposed techniques: (i) The Memory (M), (ii) Random Immigrants (RI), (iii) Memory and Random Immigrants (MRI), and (iv) Low Convergence (LC).

GAP vs. GAP with Memory (M)

Figure 4.42 presents the population for the GAP algorithm, and Figure 4.43 shows the best path after each turning point. It is evident from the figures that the GA gets stuck in a local minimum, and thereby loses its exploration capability.

In Figure 4.44, the best path snapshots for the same problem are presented when the memory component is added. The figures indicate how the memory component affects the solution quality, as the GA recalls the previously visited solutions after the new obstacle is introduced (generation 26).

The convergence curve for the GAP is plotted in Figure 4.45 versus the convergence curve for the GAP with the memory. The plot shows how the added memory renders the algorithm more applicable in dynamic environments.

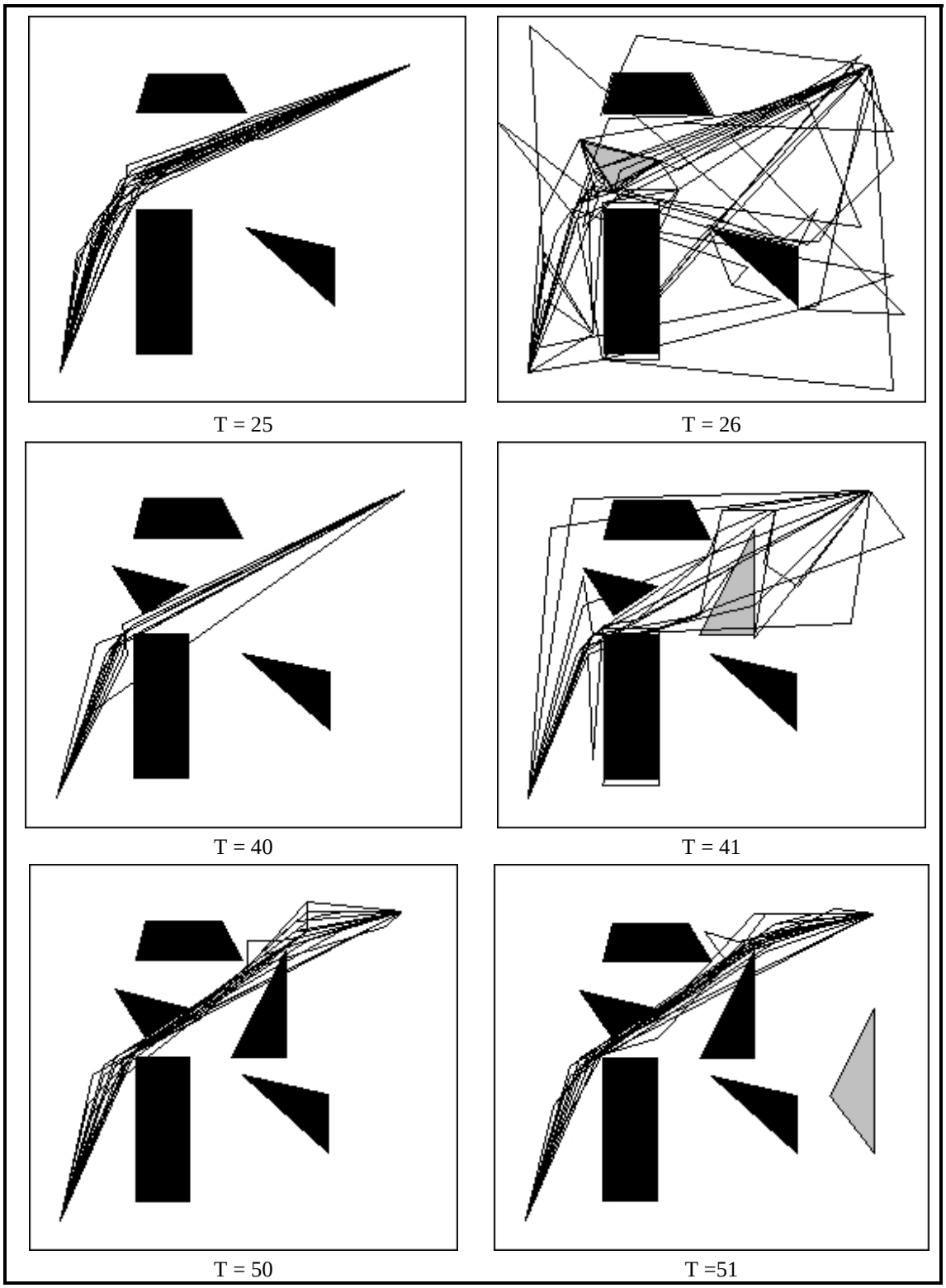


Figure 4.42: Population snapshots task 1 using the basic algorithm

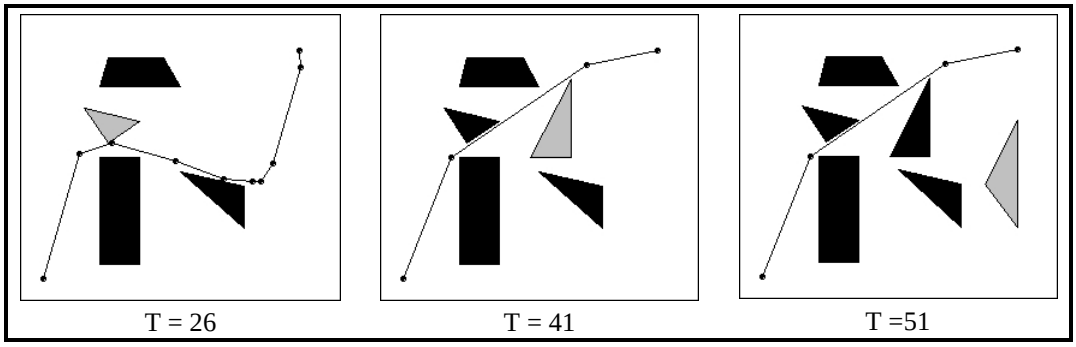


Figure 4.43: Best path (GAP)

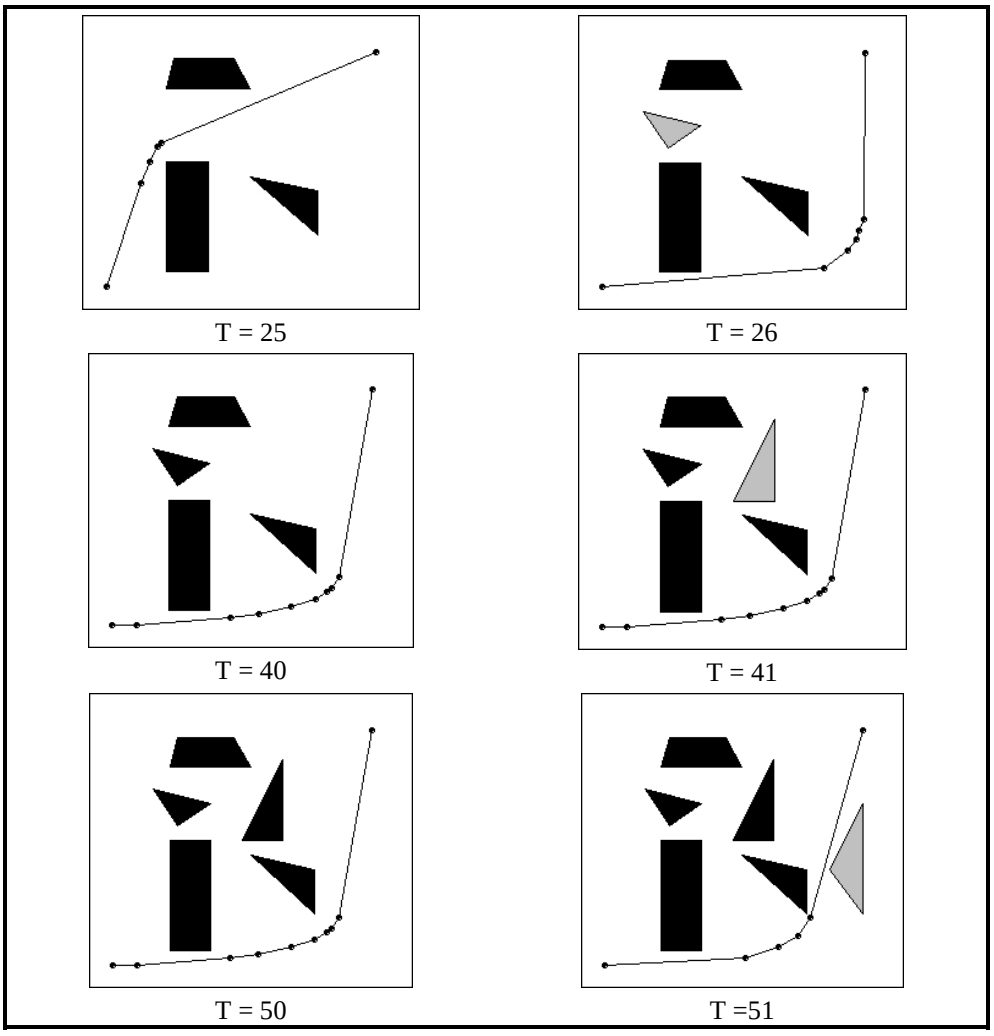


Figure 4.44: Best paths: GAP +Memory

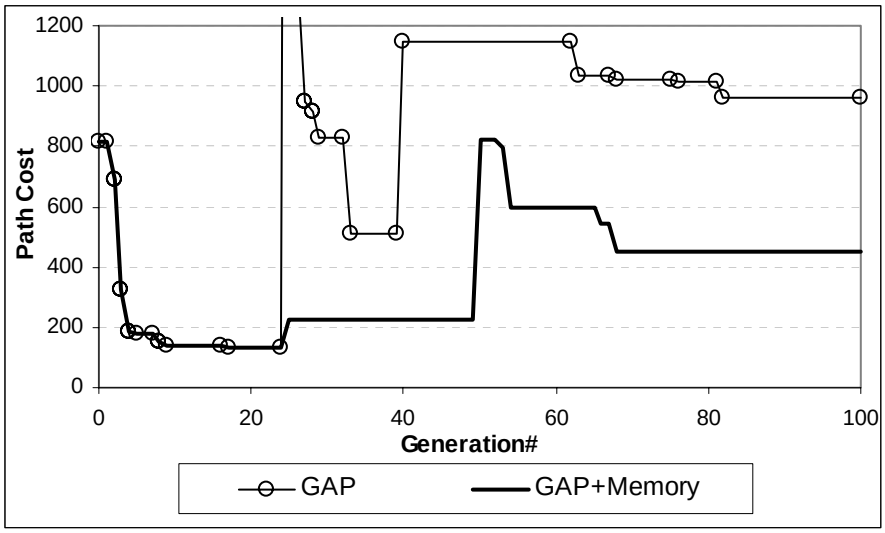


Figure 4.45: GAP vs. GAP +Memory

GAP vs. GAP with Low convergence (LC)

In the LC mode, the evolution process is running in a slow mode. The main factor that enables the LC to work effectively is the mutation that is set at 65%. The mutation range Δx, Δy is maintained high all the time rather than controlled by the feasibility ratio in the population.

Figure 4.46 displays the population for the GAP with the LC, and Figure 4.47 portrays the best path after each change in the environment. Figure 4.46 shows how the diversity in the populations is consistently ensured. The convergence curve for the GAP is plotted in Figure 4.48 versus the convergence curve for the GAP with the LC. In this plot, it is interesting to conclude that this approach converges slowly in the early generations and is unable to obtain enhanced solution with respect to pure GAP approach. However, when the environment changes, it recovers very quickly and leads to good solutions.

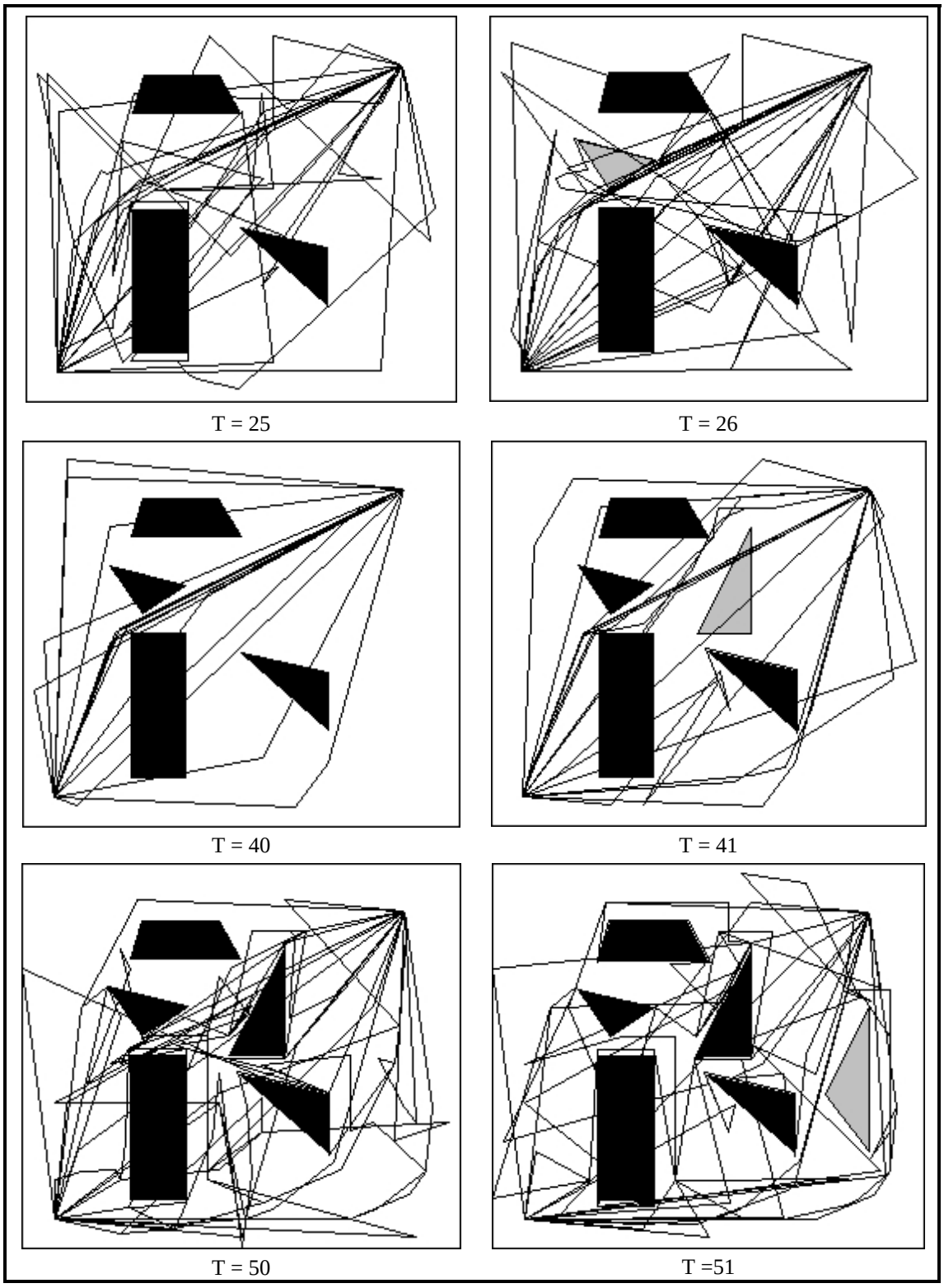


Figure 4.46: Population snapshots task 1 using GAP + LC

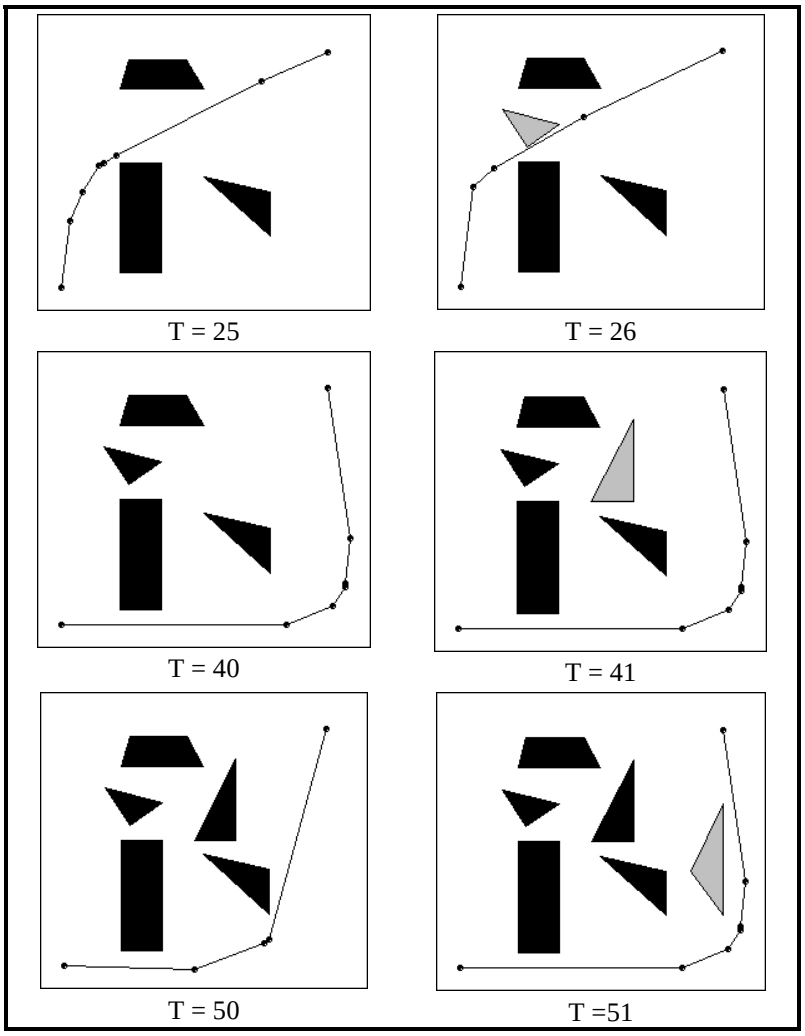


Figure 4.47: Best paths (GAP+ LC)

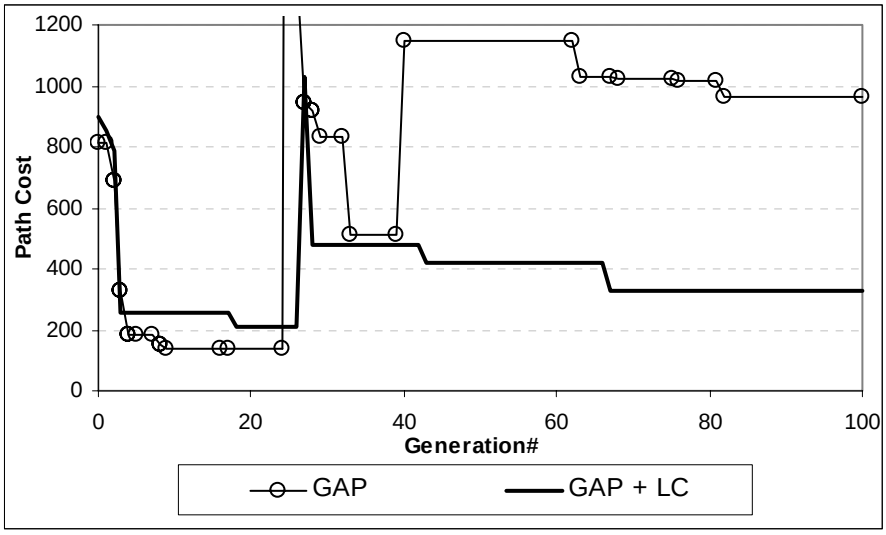


Figure 4.48: GAP vs. GAP+ LC

GAP vs. GAP with Random Immigrants (RI)

In this mode, randomly generated individuals replace the worst individuals in the population. The number of RI individuals is generated randomly, and bounded by half of the population size. The frequency of immigration is set at every five generations. Similar behaviour is expected from this technique with respect to the previous technique (LC). The convergence curve in Figure 4.49 is close to the previous technique. For this run, the solution quality obtained by this technique is better than all the other techniques.

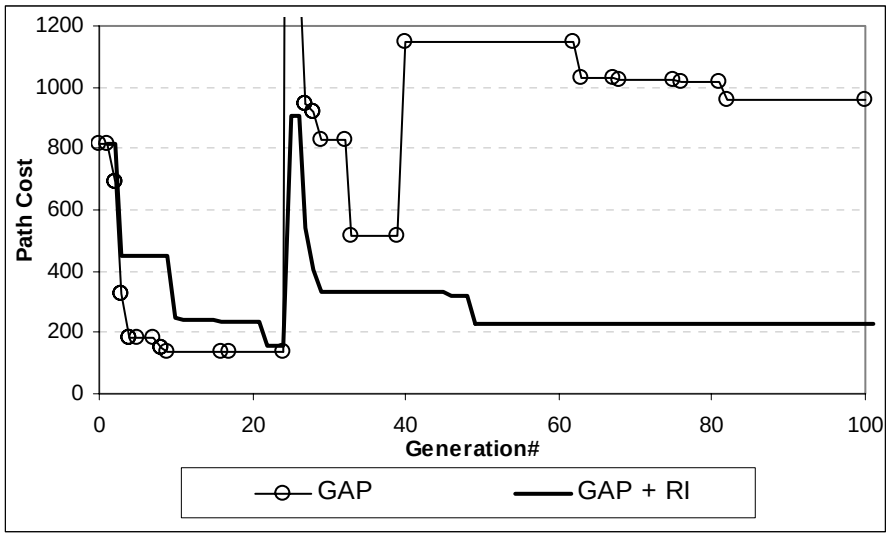


Figure 4.49: GAP vs. GAP + RI

GAP vs. GAP with Memory and Random Immigrants (MRI)

In this mode of operation, memory is enabled when changes in the environment occur and as a result random immigrants are enabled. Figure 4.50 shows the convergence of this technique which is almost identical to that based on the RI approach.

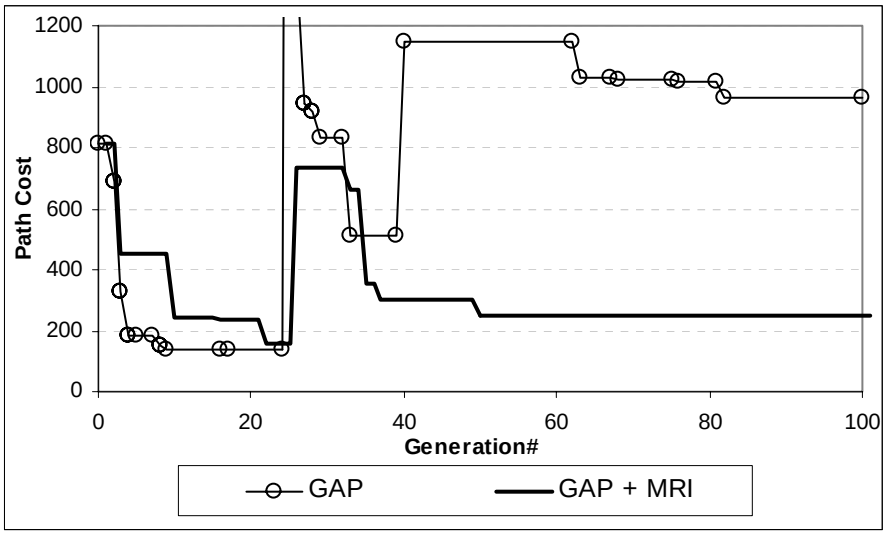


Figure 4.50 : GAP vs. GAP +MRI

The represented case study helps to understand the behaviour of each technique but does not provide any insight about the general performance of each technique. Therefore, multiple runs are conducted to compare the various techniques. The best path, average path, and CPU time are used for such comparisons. Figure 4.51 to Figure 4.55 present the best and the worst paths found in the five selected dynamic tasks. The dynamic obstacles are characterized by the grey colour where as the static obstacles black in colour. Table 4.5 to Table 4.9 display the gathered data from the ten runs for these tasks. It is clear from the results obtained that the most appropriate technique is the one which adopt the memory. The strength of this technique is in its ability to remember the promising solutions, and as the solution landscape changes, the recalled solutions quickly facilitate the recovery of the algorithm. The other observation that can be made is that the LC mode is executed in a longer time, especially in crowded environments, because of the high mutation range. Consequently, the path infeasibility is increased as more path nodes are changed to new arbitrary coordinates within the environment boundary. Therefore, the application rate of the repair operator is increased dramatically.

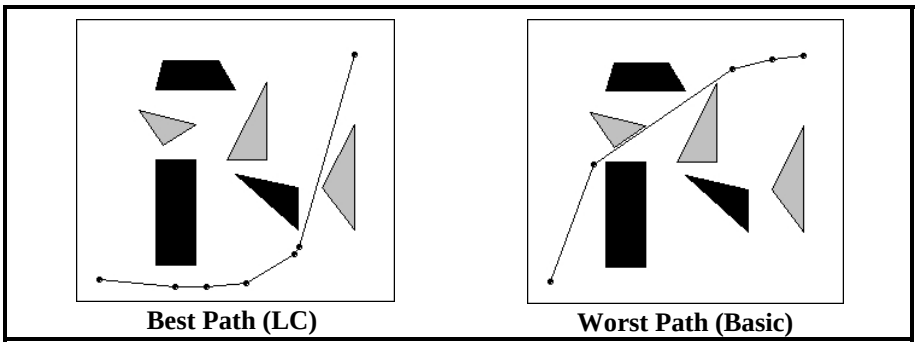


Figure 4.51: Task 1 Dynamic mode

	Pure GAP	M	MRI	LC	RI
Best	198.84	178.23	182.68	**173.88**	222.76
Average	572.88	388.46	**316.22**	373.34	444.68
Time	2.29	2.43	2.48	2.38	**2.01**

Table 4.5: Task 1 results

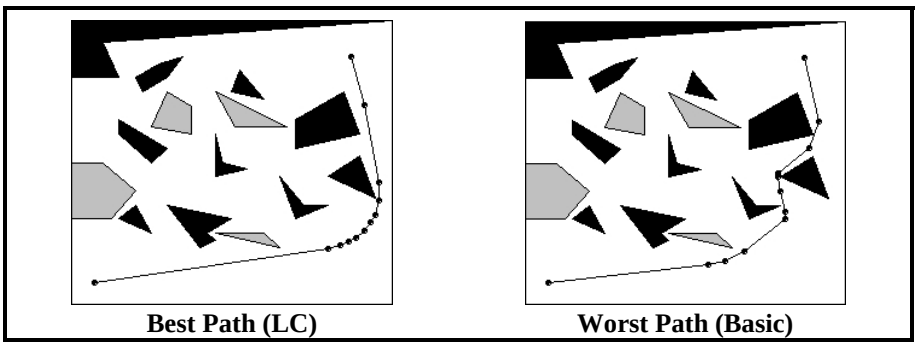


Figure 4.52: Task 2 Dynamic mode

	Pure GAP	M	MRI	LC	RI
Best	115.68	112.50	112.04	**109.28**	114.05
Average	179.92	178.10	**125.93**	131.13	132.45
Time	8.10	**7.14**	7.54	7.50	8.25

Table 4.6: Task 2 results

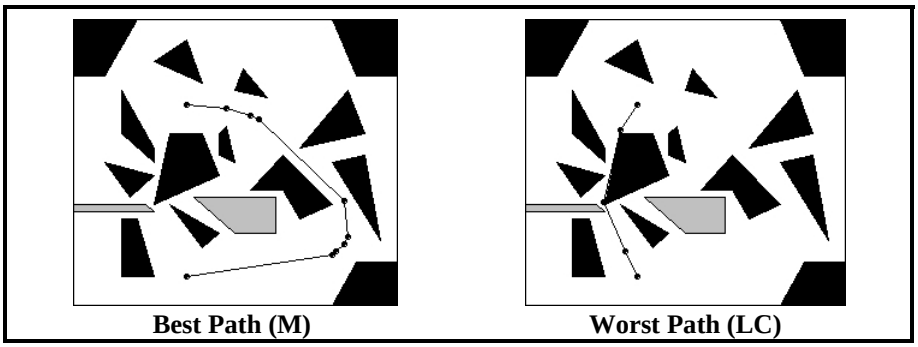


Figure 4.53: Task 3 Dynamic mode

	Pure GAP	M	MRI	LC	RI
Best	183.80	**157.73**	163.26	183.29	184.79
Average	195.25	**184.27**	187.97	208.57	195.56
Time	**5.97**	7.92	7.40	10.07	7.41

Table 4.7: Task 3 results

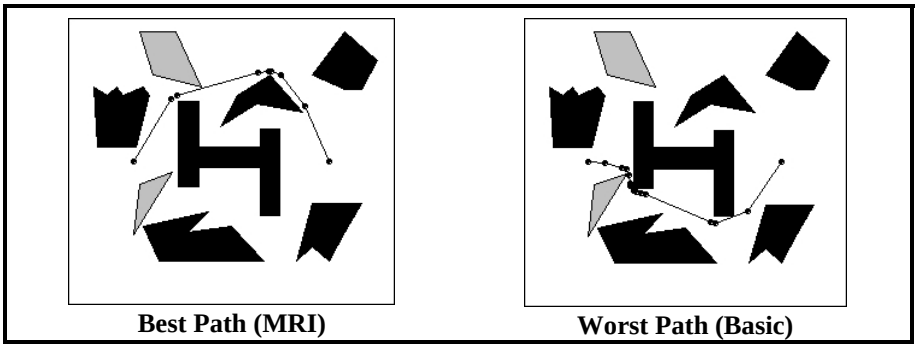


Figure 4.54: Task 4 Dynamic mode

	Pure GAP	M	MRI	LC	RI
Best	171.09	165.31	**164.76**	175.19	164.76
Average	228.35	222.00	220.32	222.64	**212.35**
Time	**6.43**	10.57	9.52	16.86	7.10

Table 4.8: Task 4 results

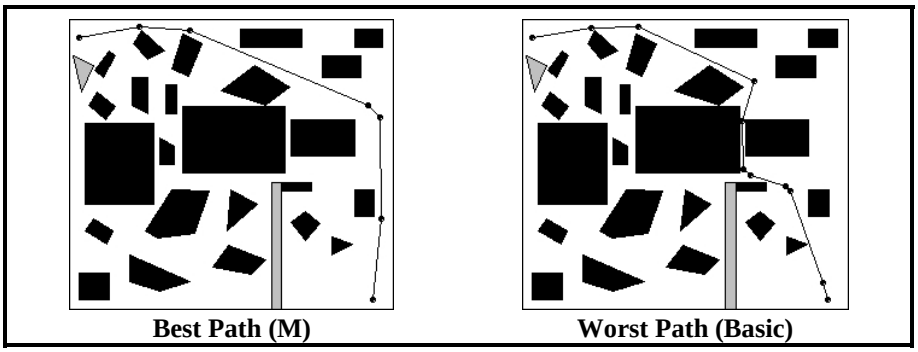


Figure 4.55: Task 5 Dynamic mode

	Pure GAP	M	MRI	LC	RI
Best	551.62	**515.39**	531.09	570.23	541.67
Average	906.29	**613.51**	650.47	661.31	727.45
Time	19.93	**19.90**	19.67	23.74	20.89

Table 4.9: Task 5 results

As illustrated in section 2.4.3, in dynamic optimization problems, the objective is no longer to find the optimal solution but to track the progression of the optimal solution throughout the solution space. To verify that the algorithm is tracking the progression of the optimal solution, the same dynamic benchmarks are solved in static mode, i.e., all the dynamic obstacles are known a priori. Figure 4.56 and Figure 4.57 display solutions obtained by the static solver (static mode) and the dynamic solver, for two different tasks. In the dynamic mode, initially several objects are removed from the environment (grey shaded objects) and then suddenly introduced again.

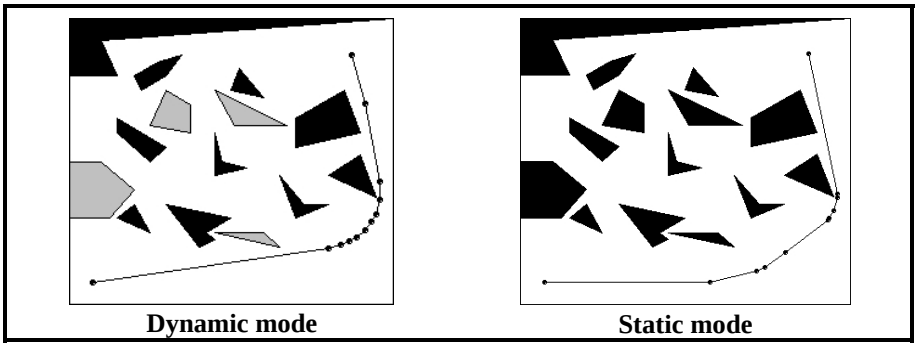


Figure 4.56: Staic vs. dynamic (Task 2)

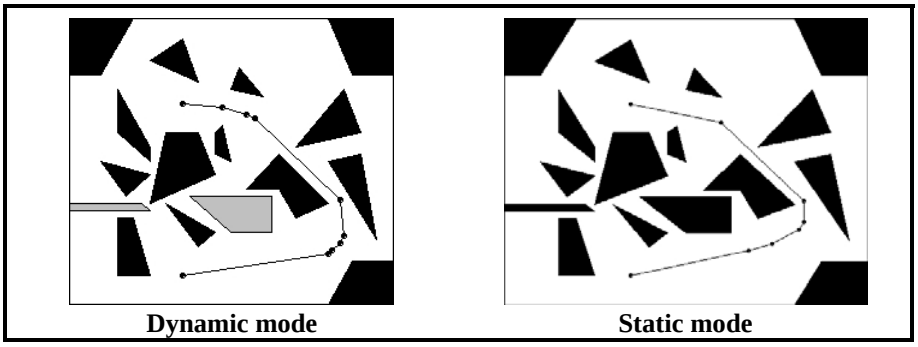


Figure 4.57: Staic vs. dynamic (Task 3)

The algorithm performance in the dynamic mode was as expected and provides good solutions similar to those provided by the static solver when the entire environment is known. The results clearly indicate that the algorithm is capable of tracking the progress of the solution space peaks and adapts to new changes in the environment.

4.7 Parallel Implementation

The GAP is capable of finding feasible and high quality solutions for all the tested benchmarks. However, as the complexity of the problem increases the GAP efficiency decreases. Therefore, a parallel implementation of the algorithm is adopted to increase the algorithm's performance.

The parallel algorithm is implemented by using the Message Passing interface (MPI) library [41]. The MPI is a library specification for message-passing, and is proposed as a standard by a broadly based committee of vendors, implementers, and users (www.mpi-forum.org). The goal of the forum is to develop practical, portable and efficient standard for writing message-passing programs.

The GAP is parallelized by using the Island-based Genetic Algorithm (IGA) approach. Islands are arranged in a ring topology as shown in Figure 2.14. Island individuals are allowed to migrate to adjacent neighbours. Migration occurs every "x" generations (i.e., x is the frequency of the migration). The number of individuals that migrate is the same for each islands, and is governed by the percentage of migration parameter. The percentage of migration dictates the percentage of the island population immigrates to neighbouring islands. All immigrants are accepted, and replace the islands' worst individuals.

Two techniques are investigated for the selection strategy, The *Best So Far Migration* (**BSF**) and The *Generation Best Migration* (**GB**). In The **BSF** the best solution(s) so far immigrate to the neighbouring islands, while in the **GB,** the best within the current generation emigrate.

4.7.1 Results

The DGA results are conducted on *SunUltra* workstations, running 900Mhz UltraSPARC III CPU with 1 Gbytes of main memory. The workstations which run UNIX are interconnected with 100Mbyte/sec Ethernet local area network.

The same benchmarks are used for testing the IGA; with each of these benchmarks, number of islands, migration frequency, and the percentage of the migrator are altered to compare the results. Six different island systems are tested: a single island, a two-island ring, a four-island ring, an eight-island ring and a ten-island ring. The entire population size is constant at 96, except for the case of the ten-islands, where the total population size is 100. A fixed number of generations is used as a termination condition, and is set to 100 for all the benchmarks.

Best So Far Migration (BSF)

Table 4.10 and Table 4.11 exhibit the obtained results with the BSF migration style, where the migration frequency = 10, and migration percentage = 10%. Note that the number of islands is denoted as "N" in the tables, the time is in seconds and the cost represents the best path cost obtained from each run. Figure 4.58 displays the speed-ups for the different benchmarks with the same configurations. Figure 4.59 and Figure 4.60 display the relative fitness for different benchmarks where a score that is greater than one denotes an improvement in the solution quality over that found in the serial algorithm. (the one-island based algorithm).

N	Benchmark 1		Benchmark 2		Benchmark 3		Benchmark 4	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	12.83	111.67	32.41	58.68	36.07	139.36	63.15	98.81
2	6.47	110.55	18.73	58.68	27.32	126.10	39.26	100.01
4	3.65	108.03	10.83	58.59	12.20	121.37	21.31	98.66
6	2.49	108.31	10.73	54.70	10.03	132.43	14.87	106.25
8	2.02	106.81	8.23	65.83	7.23	132.06	11.52	111.71
10	1.67	110.99	4.95	59.28	6.76	129.93	8.28	105.45

Table 4.10: IGA results: migration frequency = 10, migration percentage = 10%

N	Benchmark 5		Benchmark 6		Benchmark 7		Benchmark 8	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	68.58	371.97	56.71	983.28	40.97	514.39	42.53	687.63
2	38.13	373.56	37.07	1077.41	27.59	258.14	29.90	584.05
4	34.88	364.93	26.48	903.56	15.39	281.18	13.58	665.68
6	27.74	373.99	15.79	1034.49	9.12	368.93	8.15	655.69
8	23.86	375.94	13.50	1282.68	6.58	525.25	6.32	767.61
10	12.61	367.30	12.36	977.57	6.32	767.61	5.35	659.53

Table 4.11: IGA results (BSF): frequency = 10, percentage = 10%

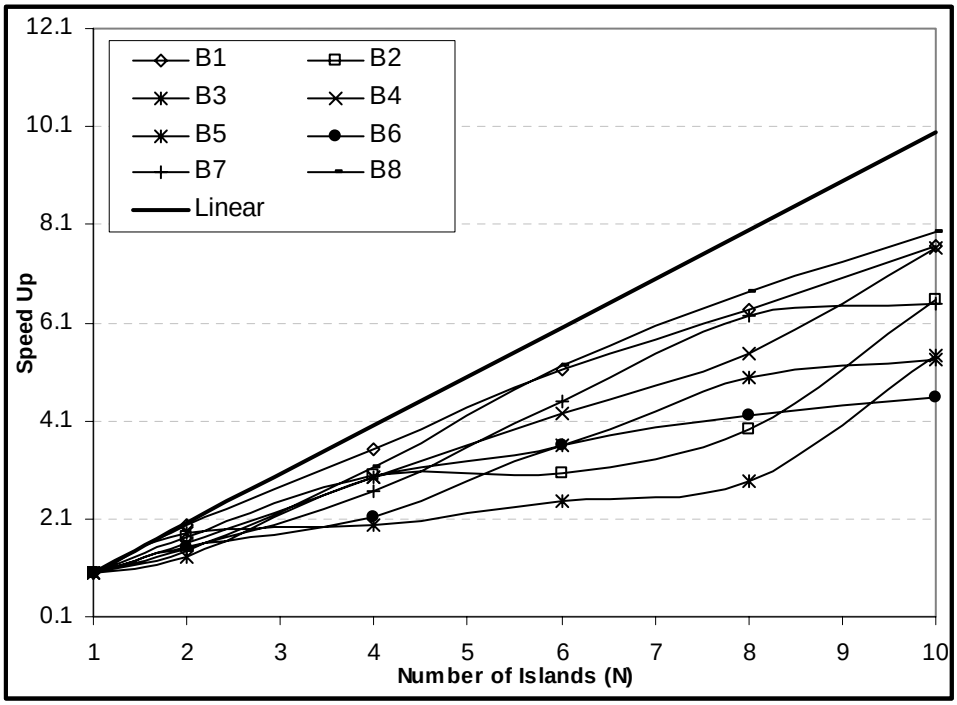


Figure 4.58: (BSF) Speed-up: frequency =10 and percentage = 10%

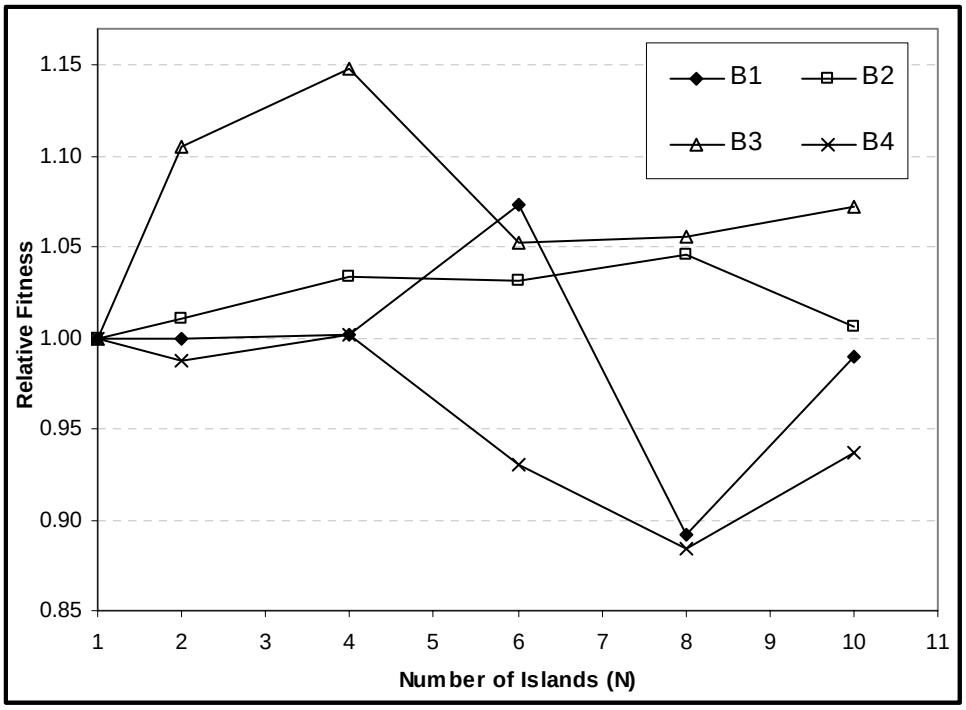


Figure 4.59: Relative fitness (BSF), with frequency =10 and percentage = 10%

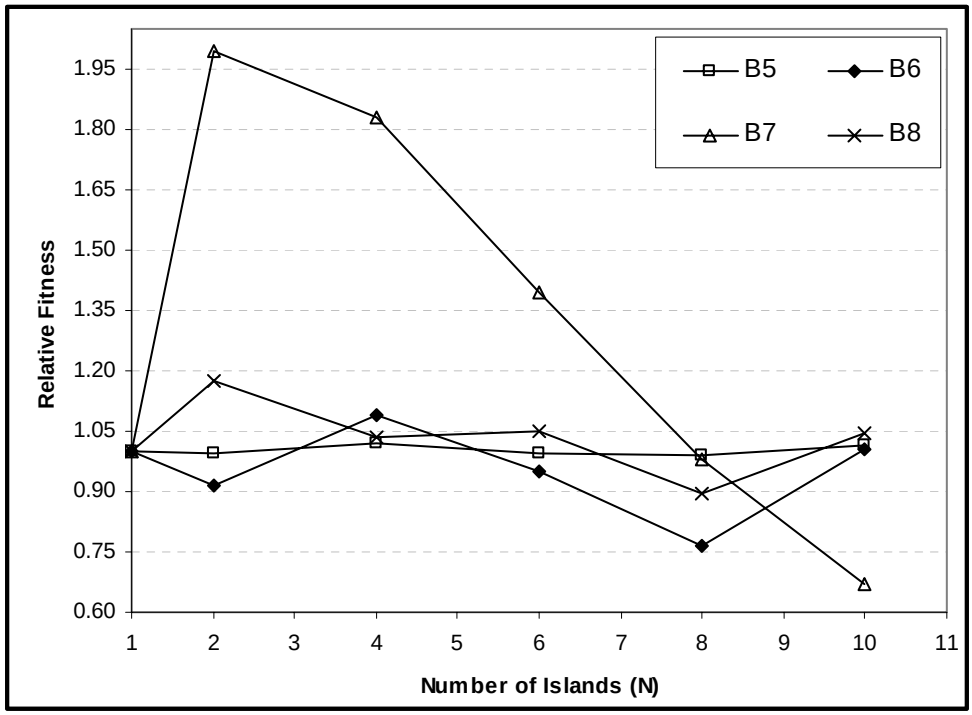


Figure 4.60: Relative fitness (BSF): frequency =10 and percentage = 10%

It is obvious that a speed-up is gained for each of the benchmarks. However, there is no clear relation between the fitness gain or loss and the number of islands utilized. This is due to the fact that as the size of the population decreases, the efficiency of the algorithm to obtain good quality solution decreases. As mentioned in Section 2.6.3 this is the main disadvantage of this type of parallel implementation. The results obtained from the other parameters are provided in Table 4.12 to Table 4.19.

N	Benchmark 1		Benchmark 2		Benchmark 3		Benchmark 4	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	12.16	111.67	32.41	58.68	36.07	139.36	63.15	98.81
2	7.09	101.21	20.77	59.39	29.71	133.81	39.63	102.89
4	3.51	108.41	11.26	58.59	11.02	128.42	22.13	104.11
6	2.47	101.22	10.56	48.74	9.14	123.50	14.39	108.53
8	2.01	109.27	8.74	56.07	7.58	122.45	10.4	106.85
10	1.76	107.32	6.98	60.03	6.71	136.46	8.99	108.47

Table 4.12: DGA results (BSF): frequency = 10, percentage = 20%

N	Benchmark 5		Benchmark 6		Benchmark 7		Benchmark 8	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	68.58	371.97	56.71	983.28	40.97	514.39	42.53	687.63
2	33.53	371.40	38.48	864.94	30.05	274.29	33.42	632.40
4	28.23	367.30	20.92	1016.51	15.42	325.91	18.81	585.04
6	26.62	371.81	19.88	857.83	8.78	482.32	9.33	668.90
8	21.22	356.61	13.27	1464.51	6.33	382.45	7.49	574.00
10	12.90	367.30	14.88	1168.29	5.61	513.40	4.86	686.32

Table 4.13: DGA results (BSF): frequency = 10, percentage = 20%

N	Benchmark 1		Benchmark 2		Benchmark 3		Benchmark 4	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	12.83	111.67	32.41	58.68	36.07	139.36	63.15	98.81
2	7.18	105.17	18.26	58.71	26.68	127.26	37.29	106.47
4	4.35	102.90	10.16	58.59	14.45	123.87	19.61	105.84
6	2.57	111.66	10.80	48.95	8.81	132.97	13.98	105.04
8	2.04	109.27	8.02	54.51	7.48	119.53	10.06	110.51
10	1.65	113.49	5.12	59.27	6.52	136.46	8.66	112.75

Table 4.14: DGA results (BSF): frequency = 10, percentage = 30%

N	Benchmark 5		Benchmark 6		Benchmark 7		Benchmark 8	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	68.58	371.97	56.71	983.28	40.97	514.39	42.53	687.63
2	34.28	378.62	36.96	848.81	29.11	277.12	28.57	624.81
4	27.24	369.53	23.98	914.66	14.43	354.61	15.24	538.90
6	35.23	397.84	16.12	1153.77	10.28	285.26	9.69	607.31
8	24.21	339.19	13.07	1181.86	6.91	327.31	6.30	624.50
10	15.08	352.09	13.25	1659.35	5.74	631.80	5.02	734.41

Table 4.15: DGA results (BSF): frequency = 10, percentage = 30%

N	Benchmark 1		Benchmark 2		Benchmark 3		Benchmark 4	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	12.83	111.67	32.41	58.68	36.07	139.36	63.15	98.81
2	6.94	105.16	18.81	59.63	24.80	137.85	36.70	97.78
4	3.57	108.72	11.73	58.59	14.23	126.92	20.52	103.78
6	2.49	107.28	10.45	54.10	9.14	131.14	13.79	107.53
8	1.87	102.10	7.79	54.96	7.26	126.96	10.30	107.94
10	1.79	106.22	5.14	64.03	5.96	125.18	8.05	105.24

Table 4.16: DGA results (BSF): frequency = 20, percentage = 20%

N	Benchmark 5		Benchmark 6		Benchmark 7		Benchmark 8	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	68.58	371.97	56.71	983.28	40.97	514.39	42.53	687.63
2	45.82	380.15	37.90	839.64	24.52	377.40	31.21	561.08
4	33.56	374.91	22.08	903.98	14.94	417.71	16.44	529.09
6	28.87	376.27	19.87	908.32	8.89	445.60	8.60	629.02
8	22.86	370.77	13.24	1101.85	6.35	535.16	6.83	625.61
10	14.68	371.58	13.81	1306.63	5.41	243.88	4.87	743.49

Table 4.17: DGA results (BSF): frequency = 20, percentage = 20%

N	Benchmark 1		Benchmark 2		Benchmark 3		Benchmark 4	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	12.83	111.67	32.41	58.68	36.07	139.36	63.15	98.81
2	6.91	102.56	20.09	58.71	27.63	129.59	37.27	108.35
4	3.70	107.72	11.30	58.59	13.06	133.78	19.60	104.04
6	2.65	109.27	9.09	59.06	9.25	138.11	14.87	108.09
8	2.01	103.90	7.44	61.92	6.78	133.05	10.35	110.35
10	1.66	113.69	6.02	59.28	6.33	136.63	8.58	106.87

Table 4.18: DGA results (BSF): frequency = 20, percentage = 30%

N	Benchmark 5		Benchmark 6		Benchmark 7		Benchmark 8	
	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	68.58	371.97	56.71	983.28	40.97	514.39	42.53	687.63
2	43.56	386.45	36.86	1041.27	25.29	362.98	28.19	613.28
4	51.71	359.02	19.23	909.72	14.35	312.51	15.07	607.20
6	28.54	373.99	18.44	1082.20	9.17	339.12	9.55	652.53
8	21.06	362.42	12.51	1121.10	5.99	479.45	6.62	625.11
10	27.08	359.33	15.37	1303.87	5.16	536.79	4.38	855.85

Table 4.19: DGA results (BSF): frequency = 20, percentage = 30%

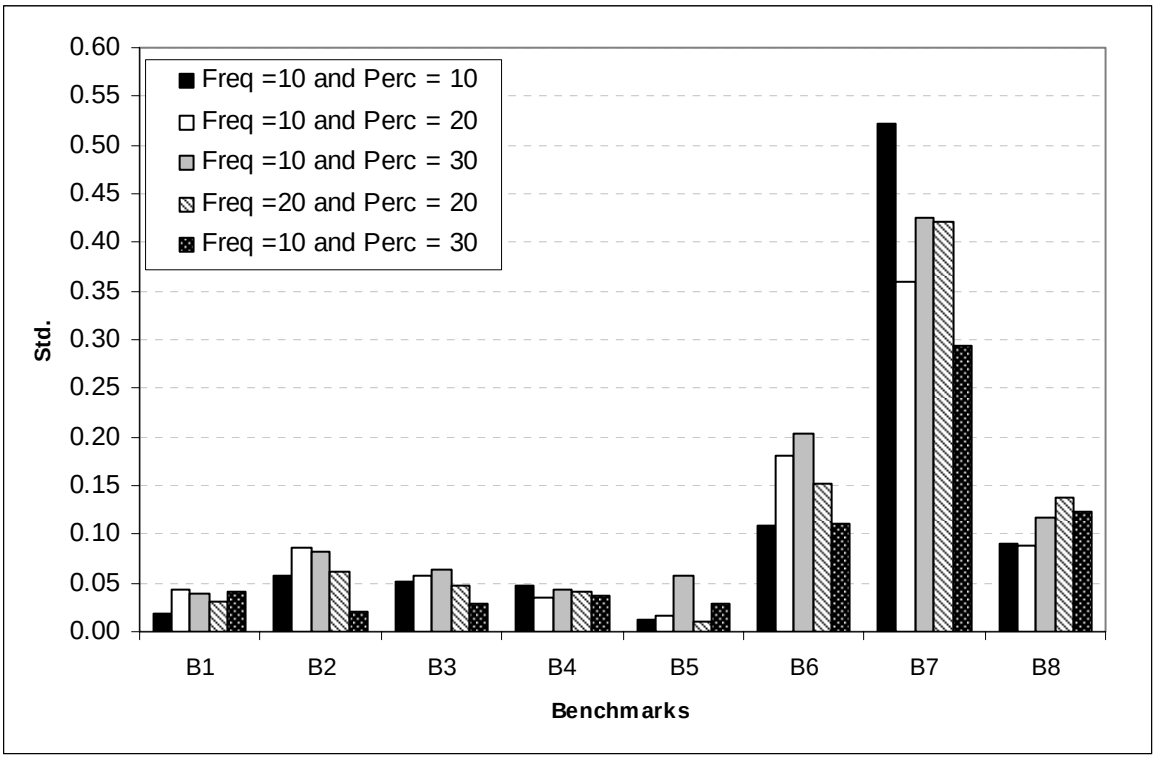
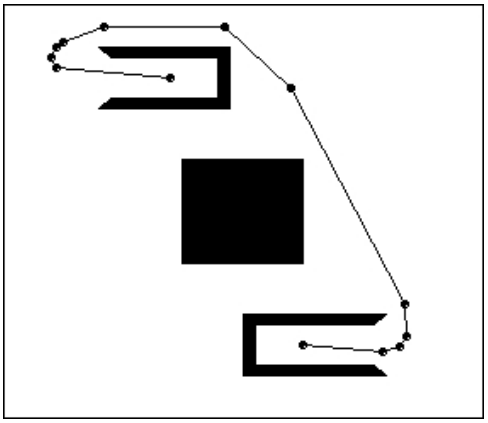


Figure 4.61: Solution quality deviation against different migration parameters

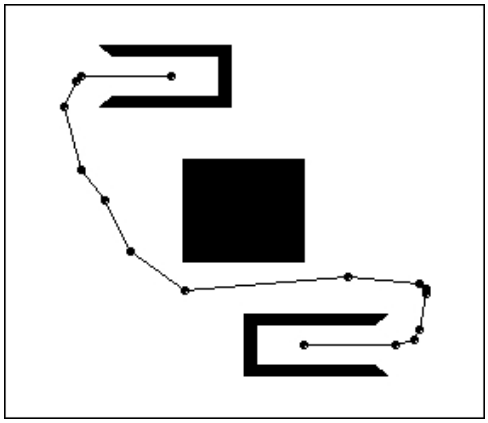
In terms of the speed-up there are no deviations in the results, but in terms of the fitness a deviation is observed. However, this difference exists as a result of the ill-behaved fitness function. A small difference in the knot node coordinates can result in big difference in the path cost. For most of the benchmarks, it is still noticed that the deviation is very small, which is reflected in Figure 4.61 (the Y axis in Figure 4.61 represents the standard deviation

of the cost for each configuration). However, a distinction exists in benchmarks 6, 7, and 8. This variation is due to the nature of these benchmarks; since these benchmarks are structured environments (see Figure 4.32). For example, benchmark 7 paths are shown in Figure 4.62. Recall that the obstacles in this case are occupying only 9.50% of the environment. As a result, '*a*' coefficient (the expositional coefficient for the clearance and the smoothness in the fitness function) is large (equal to 5.26). Therefore, small changes in the node coordinates lead to a tremendous change in the fitness function.



Single island

Distance	70.10
Minimum clearance	0.98
Maximum steering	57.43
Clearance value	303.85
Smoothness value	341.41
Path cost	715.37



2 islands

Distance	66.98
Minimum clearance	1.85
Maximum steering	45.40
Clearance value	7.41
Smoothness value	199.86
Path cost	274.29

Figure 4.62: (BSF) Sample outputs for benchmark 7

Generation Best Migration (GB)

The results of the GB migration style are similar to those of the BSF migration style. Figure 4.63 reveals the gained speed-up, while Figure 4.64 and Figure 4.65 display the relative fitness for the different benchmarks. The figures clearly show the speed-up factors obtained and the gain/loss in solution quality with respect to the number of islands. The

speed-up is less in crowded environments, however, the speed-up is guaranteed but the solution quality rises and falls as expected, when the entire population is divided.

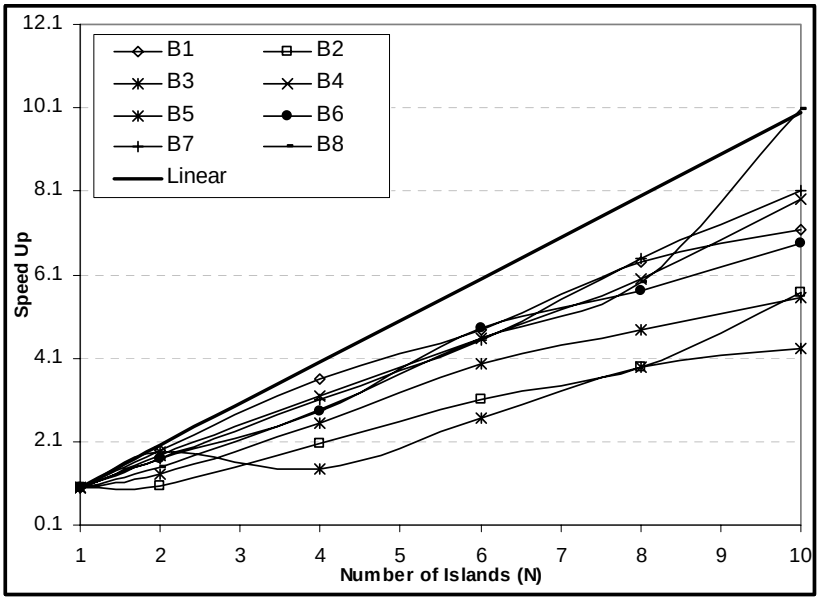


Figure 4.63: (GB) Speed-up: frequency =10 and percentage = 10%

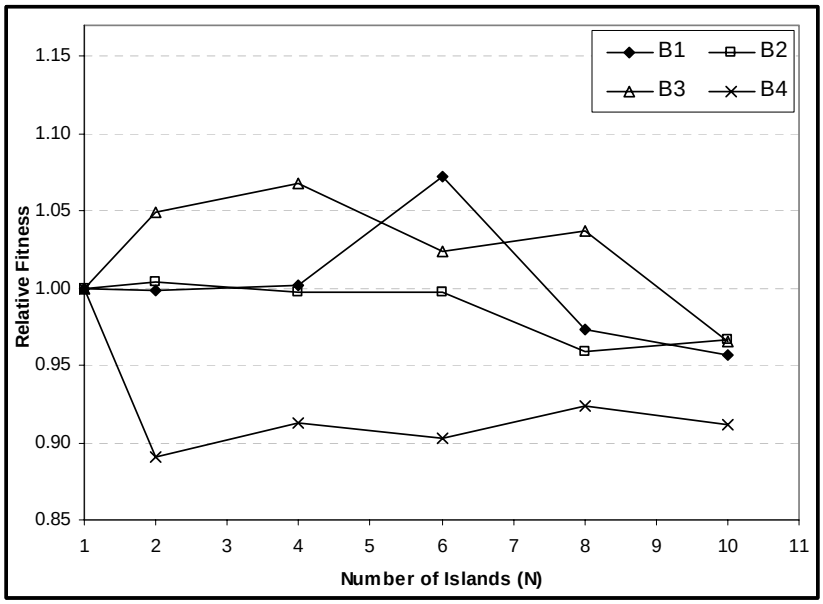


Figure 4.64: Relative fitness (GB): frequency =10 and percentage = 10%

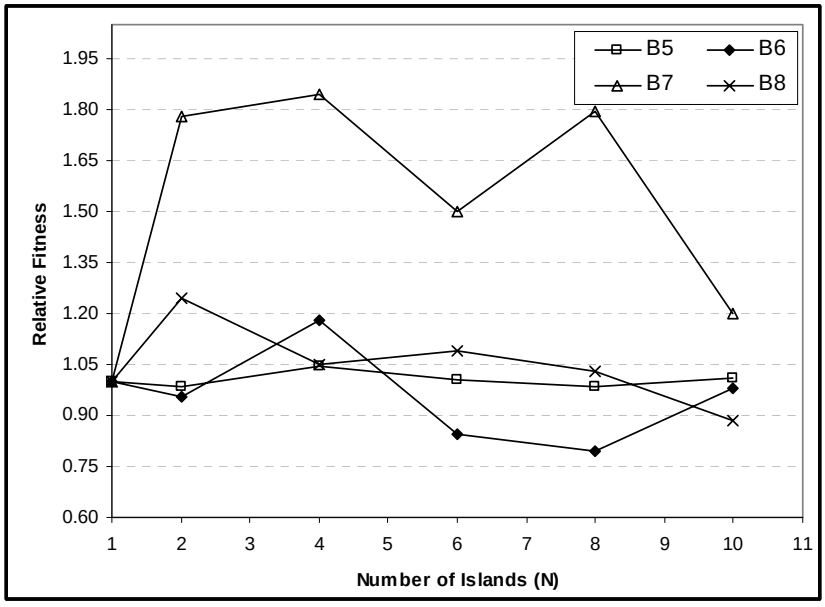


Figure 4.65: Relative fitness (GB): frequency =10 and percentage = 10%

4.8 Summary

In this Chapter, a Genetic Algorithm Planner (GAP) for solving the path planning problem is presented. Results show clearly that the GAP was 100% successful in obtaining good solutions in both static and dynamic environments. In addition, different approaches were investigated to enable the algorithm to function correctly in both static and dynamic environments. Experimental results indicate that the memory based technique is the best among other approaches investigated in this dissertation. Furthermore, The GAP is parallelized by using an island-based GA approach (IGA). In this model of computation each processor executes the GAP based on its own subpopulation with occasional exchange of high quality individuals with it's neighbour processors. Results show a near linear speed-up in the computational time. Since the entire population is divided into subpopulations the solution quality is also affected.

Chapter 5

Local Search and Memetic Algorithms

GAs are appropriate for exploring the solution space but fail to fine-tune the search. As discussed in Section 2.4.5 Local Search (LS) algorithms use iterative improvement techniques which are applied to a single solution. The new neighbourhood solution is generated based on the current solution. If the newly generated solution provides an improvement, it becomes the current solution else a new neighbourhood solution is generated and evaluated. This process is repeated until no further improvement is achieved. The LS method provides local optimum only, and the solution quality depends tremendously on the starting initial point. To increase the chances of obtaining a global optimum, the LS is usually executed from several starting solutions.

Combining global and local search is used for many global optimization approaches. Memetic Algorithms (MAs) [42] - also known as hybrid EAs and genetic local searches are recognized as a powerful search paradigm for evolutionary computing.

5.1 Local Search

The LS is designed and implemented with the goal of integrating it with the GA to form a Memetic Algorithm (MA). Figure 5.1 shows an overview of the developed LS algorithm.

The main component is to define the solution neighborhood. For this problem, the neighbour search (neighbour operator) is conducted by changing the node coordinates of a given solution. The neighbour search process is illustrated in Figure 5.2. For each node, the neighbouring nodes are generated within the desired clearance distance of the original node, denoted as the search window. The number of possible nodes within the window changes randomly from four to N, where N is the maximum search resolution (denoted here as the maximum zoom). The maximum zoom resolution increases, whenever the search fails to improve the path. For each knot node in the path, the algorithm either accepts the best improvement or the first found improvement. At the beginning of each improvement process, the path segments are divided to give the improvement process more nodes to improve. After each improvement process, the path is lined up to delete the unneeded nodes.

The path improvement process terminates if no improvements are reported during a given number of iterations. It is found that the algorithm is trapped in a local minimum, if the process cannot make further improvements during two consecutive iterations.

```
Generate random path P;
If (P is not feasible) Repair (P);
While termination condition not met
{
   Split segments (P);
   Improve (P);
   Line up (P);
   If (P' < P) P = P';
}
```

Figure 5.1: LS algorithm

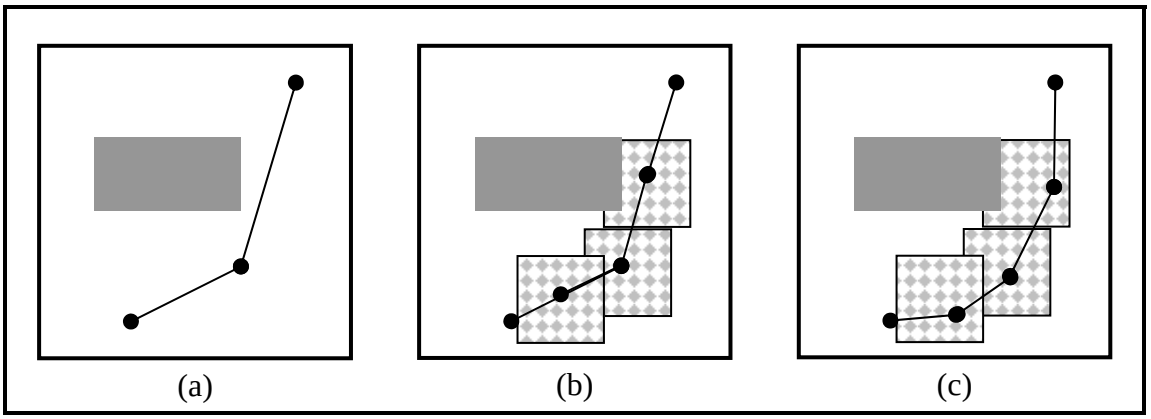


Figure 5.2: Path improvements: (a) initial repaired path, (b) nodes search window, and (c) improved path

5.1.1 LS Results

This section presents the results of the implemented LS in static and dynamic environments for the selected benchmarks.

Static Environment

As discussed in the previous section, two techniques are implemented for the improvement process. Either the "first improvement" or the "best improvement" is accepted. In utilizing the "first improvement" approach, the algorithm accepts the first determined move for each node. On the other hand, accepting the "best improvement" forces the algorithm to check all the neighbours and allows only the best move found so far to be used. The LS algorithm is applied to all benchmarks. For each benchmark, ten runs are carried out, and the resultant best path is displayed graphically for each LS technique along with the best cost, average cost, and the average CPU time.

It is expected that "accept first" takes less time and the accept best generates high quality solutions, however, the results show that this is not always the case for this implementation. It can be seen from the obtained results in Tables 5.1 through 5.8 that there is not much

difference in the performance of either technique. However, as the problem complexity increases accept best technique is more likely to perform better as shown in Figure 5.11. It is also notices that for a problem that has less alternative solutions topology like the structured environments in task 6 and task 8 the accept best provides a high quality solution, however, these solutions are obtained at a the cost of the computational time. The best obtained result for each benchmark is presented in Figure 5.3 through Figure 5.10.

As far as the computational time is concerned, the accept first requires slightly less time to generate a similar results, as signified in Figure 5.12. Therefore, the accept first technique is chosen for further development and comparison in this work.

The strength of the LS is its ability to find the global minimum, if the starting solution is in the global minimum region. Therefore, it can find the optimal solution in problems with a small number of local minima.

	Best	Average	Time
Accept Best	93.07	168.39	13.35
Accept First	99.56	205.61	10.88

Table 5.1: LS results for Task 1

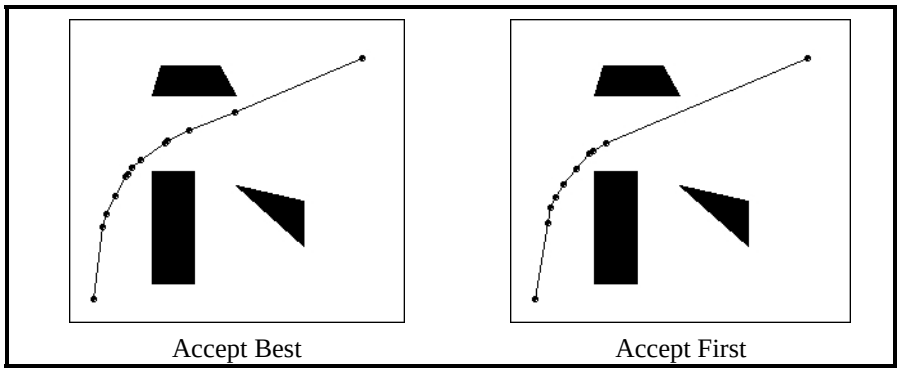


Figure 5.3: Accept first vs. accept best strategy (Task 1)

	Best	Average	Time
Accept Best	48.69	119.20	15.13
Accept First	59.60	116.88	14.96

Table 5.2: LS results for Task 2

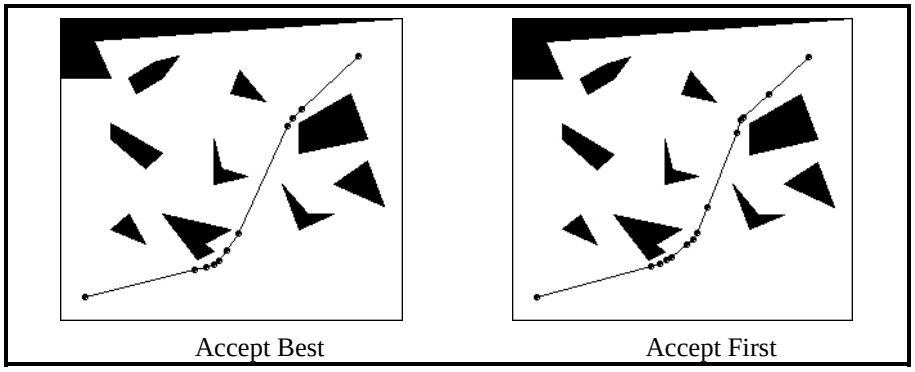


Figure 5.4: Accept first vs. accept best strategy (Task 2)

	Best	Average	Time
Accept Best	140.49	265.12	9.69
Accept First	139.35	299.24	9.66

Table 5.3: LS results for Task 3

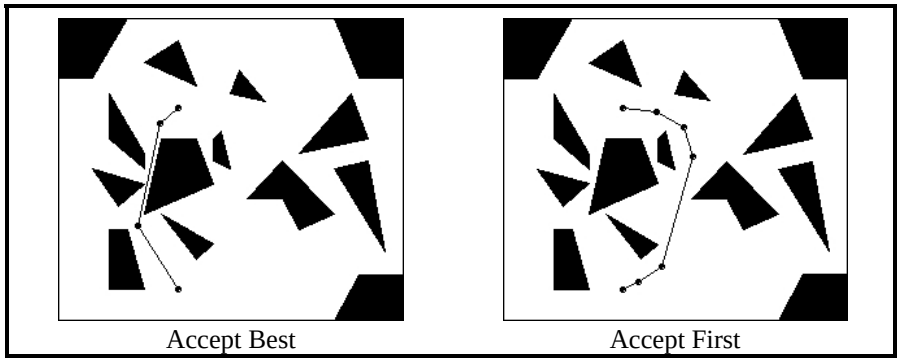


Figure 5.5: Accept first vs. accept best strategy (Task3)

	Best	Average	Time
Accept Best	94.60	124.45	32.04
Accept First	77.46	141.47	30.82

Table 5.4: LS results for Task 4

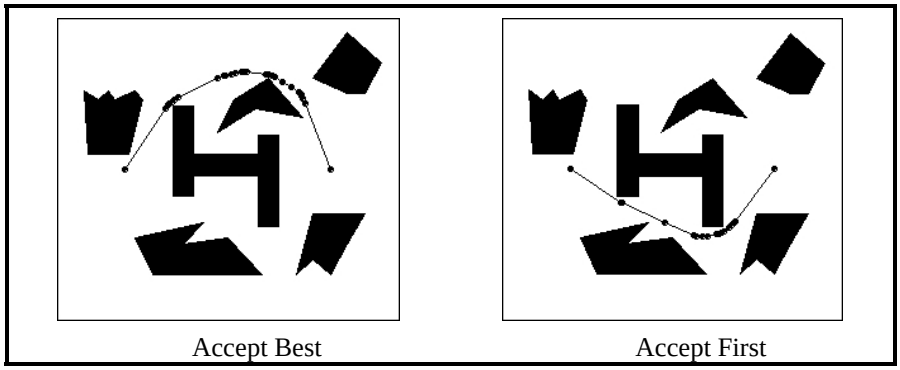


Figure 5.6: Accept first vs. accept best strategy (Task 4)

	Best	Average	Time
Accept Best	408.47	1720.71	64.52
Accept First	720.55	1923.66	94.53

Table 5.5: LS results for Task 5

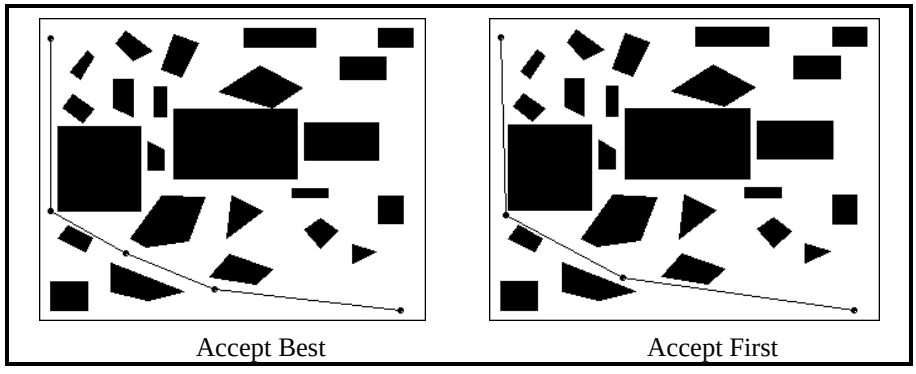


Figure 5.7: Accept first vs. accept best strategy (Task 5)

	Best	Average	Time
Accept Best	228.92	316.46	120.22
Accept First	389.90	505.37	70.81

Table 5.6: LS results for Task 6

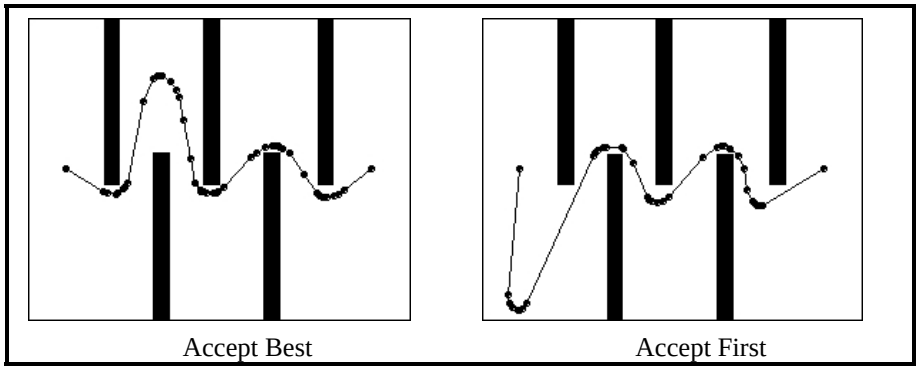


Figure 5.8: Accept first vs. accept best strategy (Task 6)

	Best	Average	Time
Accept Best	126.92	666.77	25.20
Accept First	283.41	1938.13	18.06

Table 5.7: LS results for Task 7

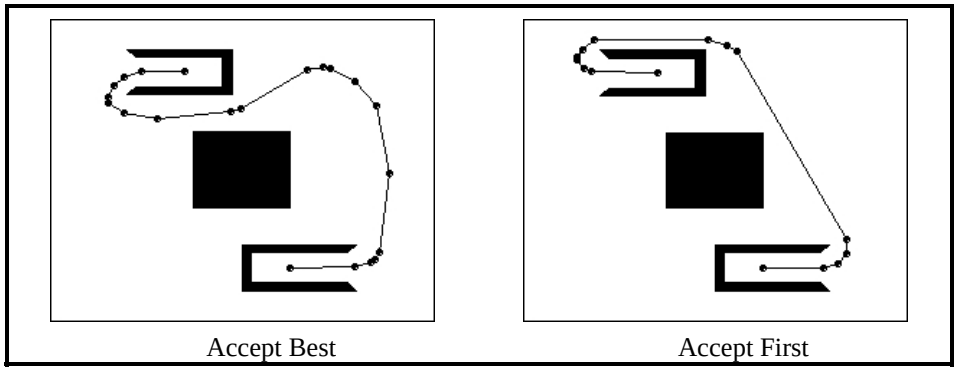


Figure 5.9: Accept first vs. accept best strategy (Task 7)

	Best	Average	Time
Accept Best	259.22	301.68	299.51
Accept First	274.33	348.63	181.78

Table 5.8: LS results for Task 8

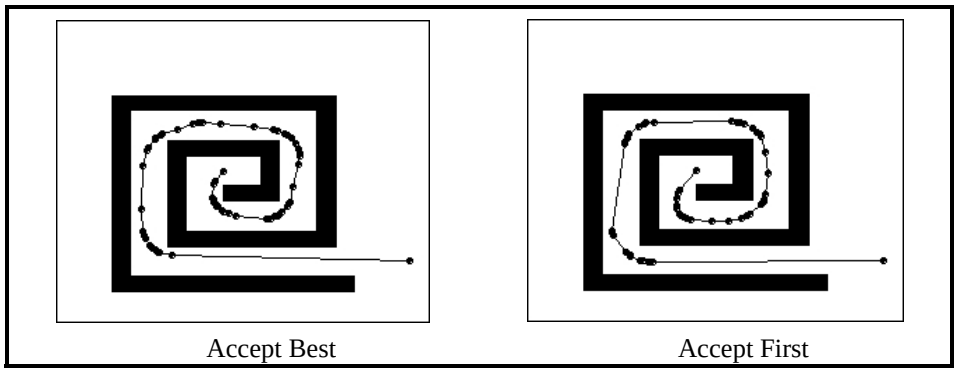


Figure 5.10: Accept first vs. accept best strategy (Task 8)

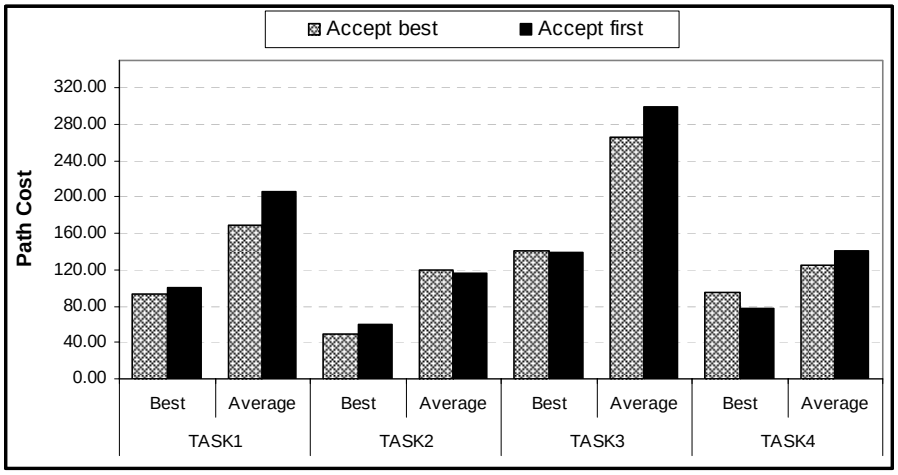


Figure 5.11: Solution quality (accept first vs. accept best)

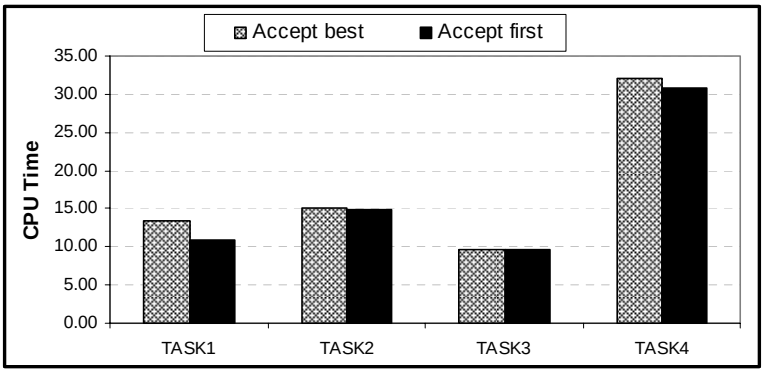


Figure 5.12: Computational time (accept first vs. accept best)

Dynamic Environments

In the dynamic Genetic Algorithms environment obstacles were introduced at a given generation number, however, in LS the iteration number are considered. Ten runs are conducted for each dynamic benchmark. Results obtained are presented in Figures 5.13 through 5.17.

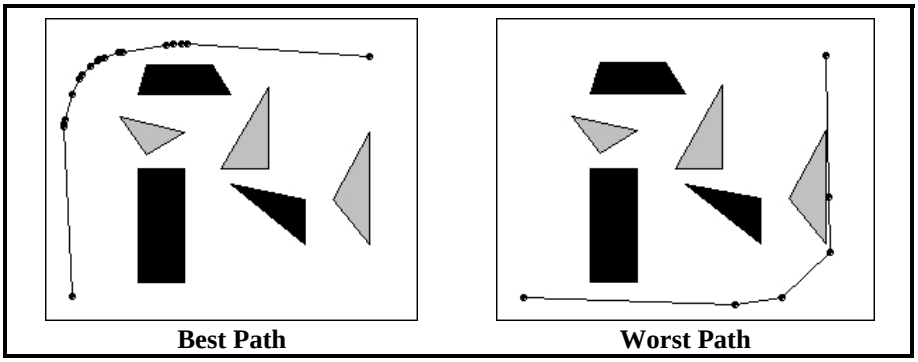


Figure 5.13: LS Dynamic result (Task 1)

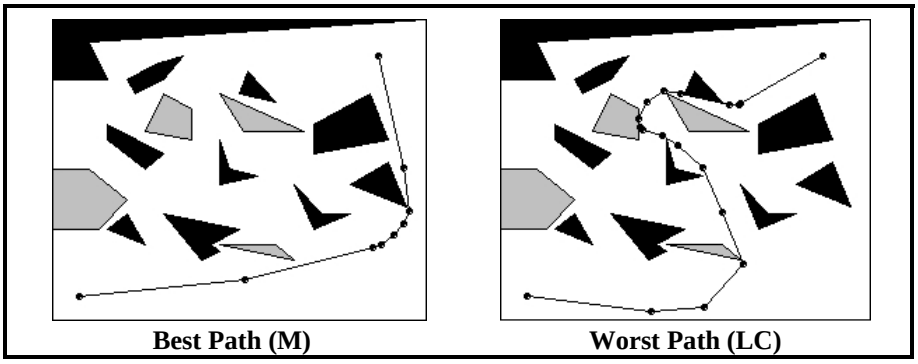


Figure 5.14: LS Dynamic result (Task 2)

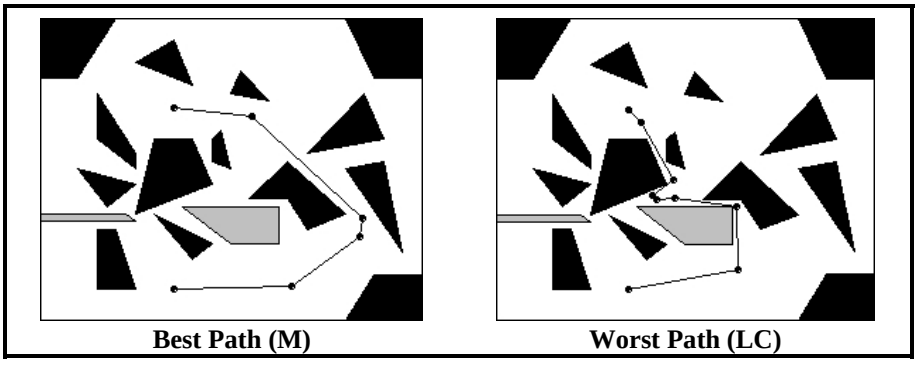


Figure 5.15: LS Dynamic result (Task 3)

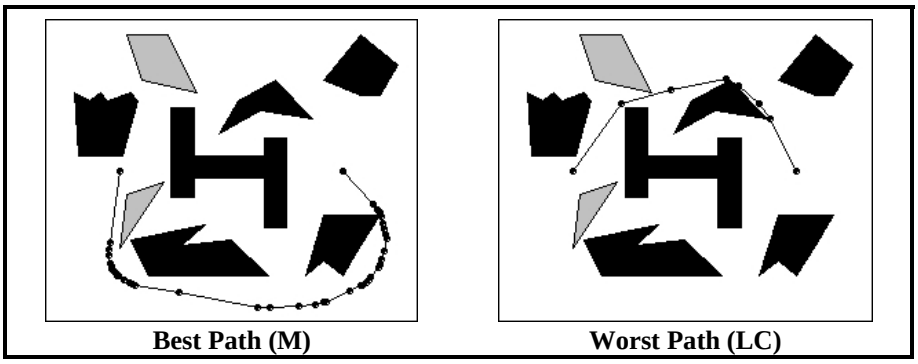


Figure 5.16: LS Dynamic result (Task 4)

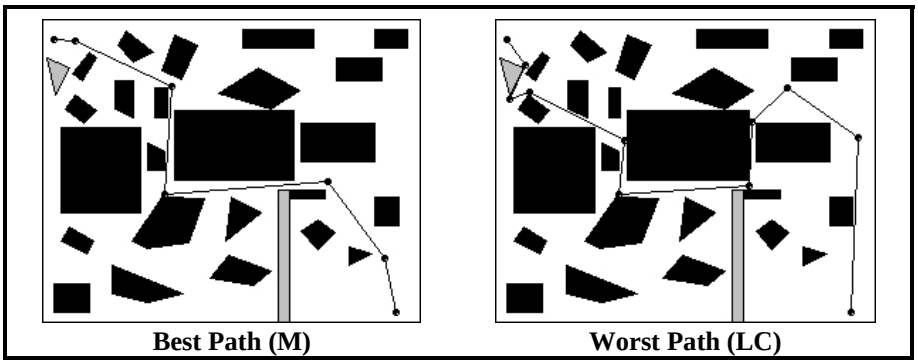


Figure 5.17: LS Dynamic result (Task 5)

As can be seen from the results, the LS get stuck in a local minimum and depend highly on the starting initial solution.

5.1.2 GA vs. LS in Static Environments

The results from the previous section and the results from Section 4.6.1 are combined and plotted for comparison. Figure 5.18 demonstrates that for most of the tested benchmarks the LS obtains slightly better solutions than those of the GAP. It is also observed that for Task 6 and Task 7 the performance of the LS in terms of the solution quality is much better than the GA. This is expected since these two benchmarks have only one possible topology and therefore, the LS were able to obtain superior solutions. However, these high quality solutions have a price and are obtained using high computation time.

In terms of the average of success to obtain a high quality solution the GAP performs better than the LS, as shown in Figure 5.19. Furthermore, the GAP also requires much less time to obtain the solutions as can be observed from Figure 5.20.

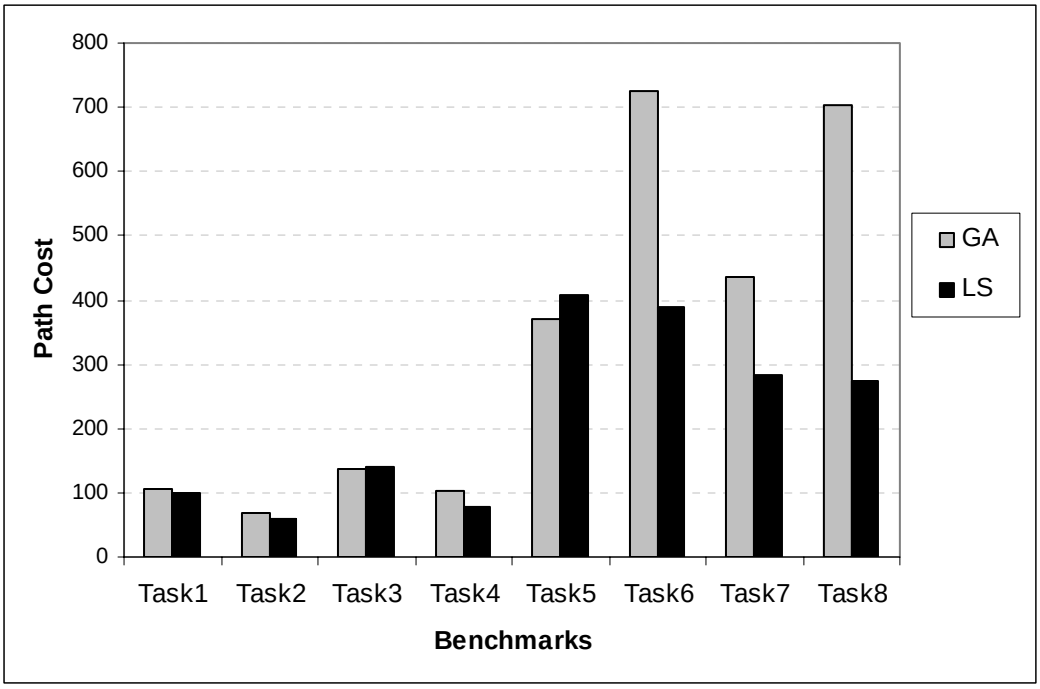


Figure 5.18: Best solution GA vs. LS (Static)

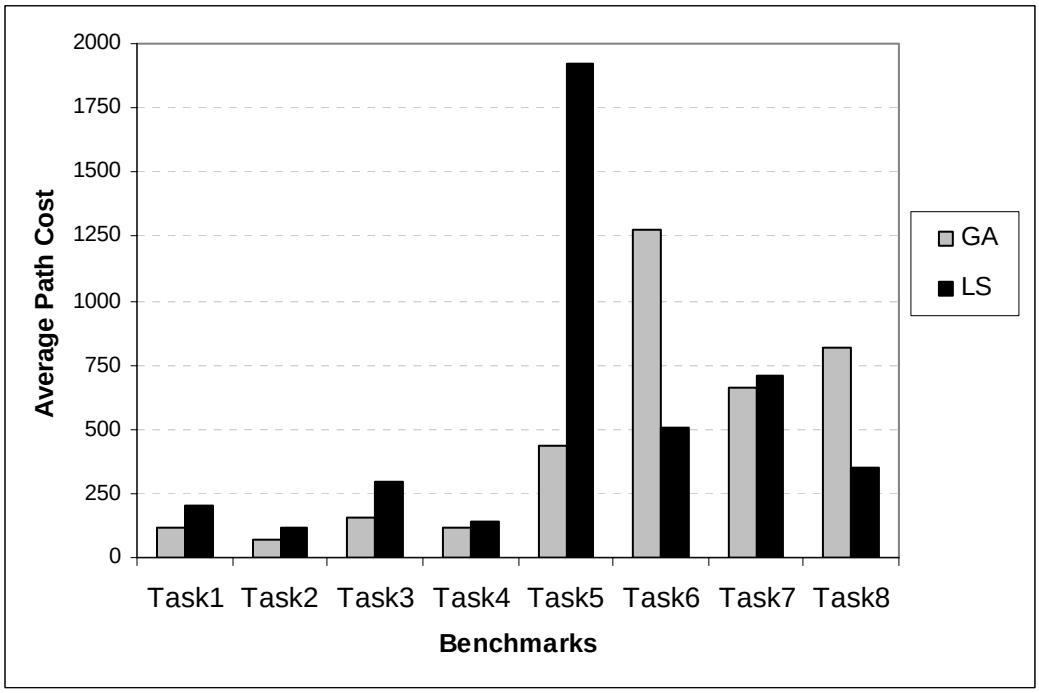


Figure 5.19: Average path cost GA vs. LS (Static)

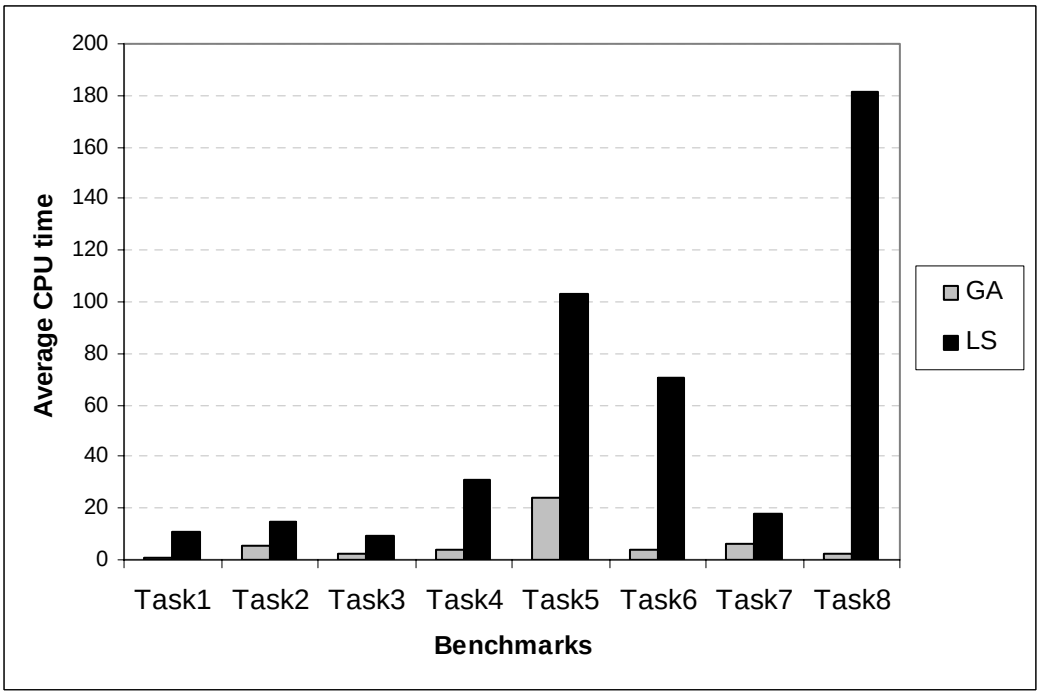


Figure 5.20: Average CPU time GA vs. LS (Static)

5.1.3 GA vs. LS in Dynamic Environments

The obtained LS and GA results from dynamic environments are also combined and plotted for comparison. As demonstrated in Figure 5.21, Figure 5.22 and Figure 5.23, the GA technique performs on average better than LS. However LS was able to provide better solutions for two benchmarks namely Task 1 and Task 4. This is attributed to the fact that in these two benchmarks newly generated obstacles don't affect all the solution space peaks and therefore, the LS chances to obtain better solution are higher. However, in terms on of the average of success to obtain high quality solutions the GAP performs better than the LS in all the tested benchmarks, and furthermore the GAP is much faster than LS.

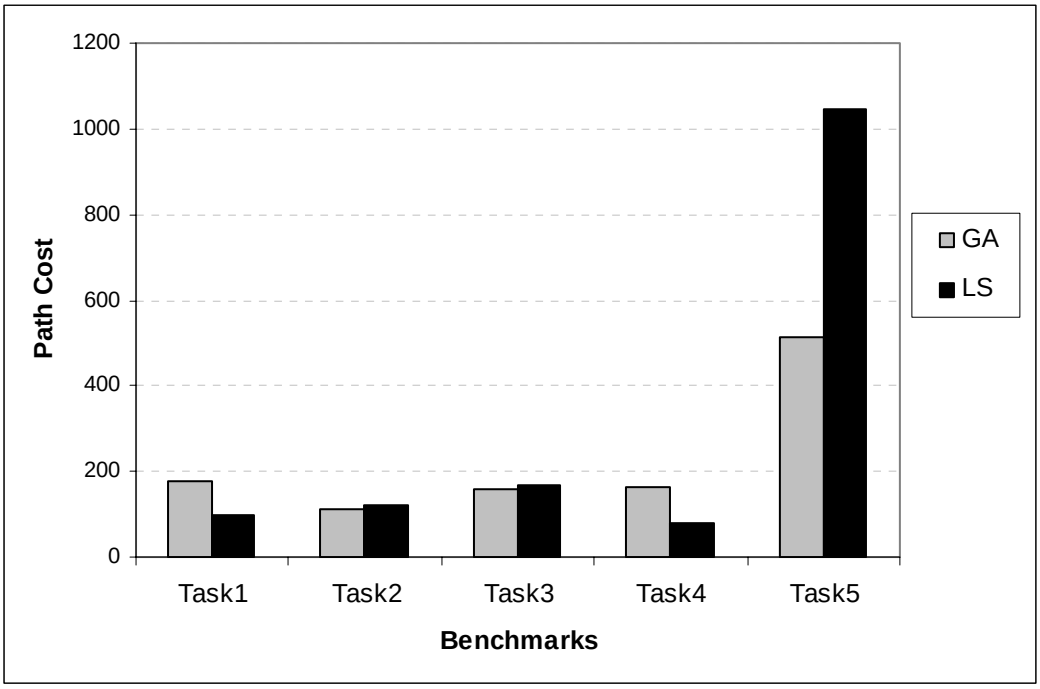


Figure 5.21: Best solution GA vs. LS (Dynamic)

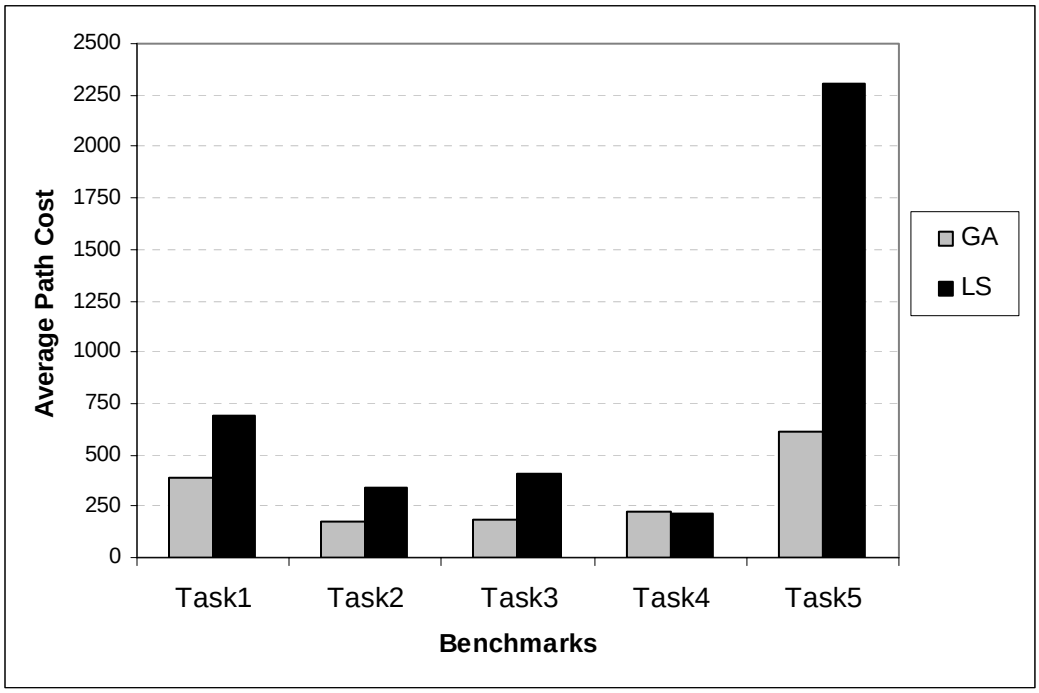


Figure 5.22: Average path cost GA vs. LS (Dynamic)

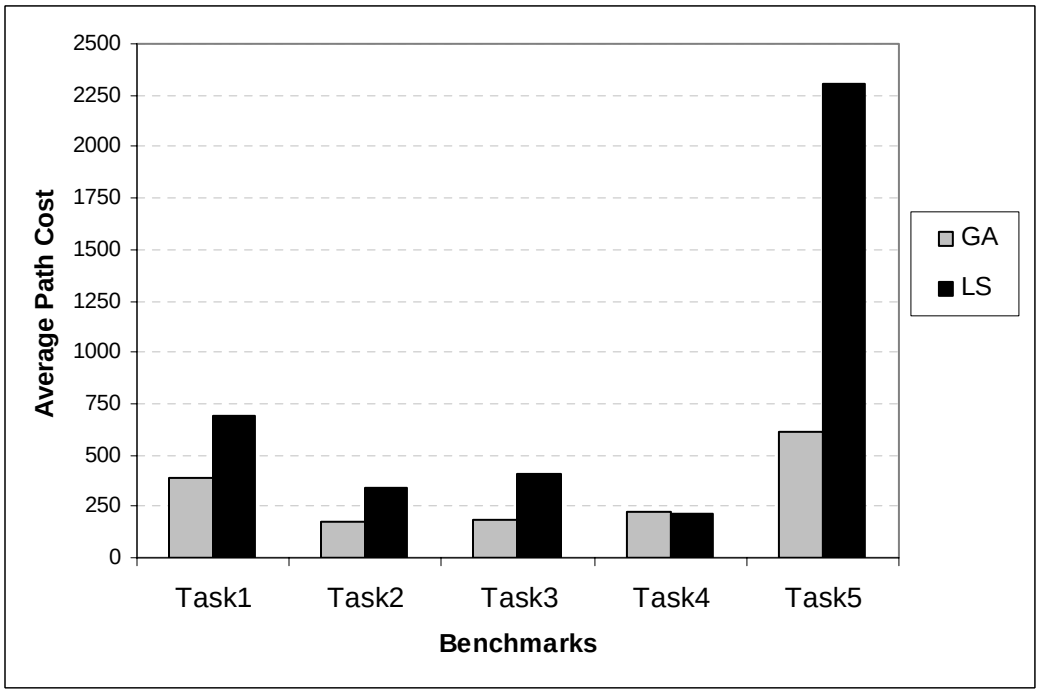


Figure 5.23: Average CPU time GA vs. LS (Dynamic)

It can be concluded from the results in the static/dynamic environments that the GA is more robust and scalable than the LS and requires much less time to compute final solutions.

5.2 Memetic Algorithms

GA is not suitable for fine-tuning solutions that are close to the sub-optimal solution. The absence of this feature (i.e., fine-tuning) can be recovered by using an LS approach. The incorporation of a local improvement operator within the evolution process of the GA will most likely generate improved solution quality. There are several approaches to implement MAs; however, in this thesis, three strategies are utilized to hybridize the GA and LS as follows:

1. Improve Best Solution

In this technique, after the GAP is used to solve the problem, the best found path is further tuned by using local search. A superior solution are expected from this technique but is expected to have more computational time.

2. LS Operator

In this technique, the LS is introduced as an operator within the GA; that is, an LS is applied to a percentage of the population. This LS operator is implemented such that only single improvement iteration is applied on each selected path (i.e., partial LS). This partial LS operator is applied to 10% of the population through experimentation, and the paths are selected randomly from the feasible paths within the current population.

3. Improve Initial

Here, complete LS is applied to 10% of the initial population. To accelerate the search process, the maximum number of unimproved iterations for the LS is reduced to 1, that is, the LS stops if no improvement is reported for one iteration.

5.2.1 Results

The same experimental setup is utilized here where ten runs are conducted for the three approaches. A comparison is then made between these approaches and the GAP in terms of the solution quality (the best path cost and the average cost) and computation time. Tables 5.9 through 5.11 present complete results for the implemented MAs, GA and LS. However, plots are created for the first four benchmarks are shown in Figures 5.24 through 5.26 respectively.

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Improve Best	90.38	48.46	118.87	89.85	379.26	198.20	143.31	248.54
LS operator	102.92	48.56	137.21	105.17	353.11	879.22	440.20	524.21
Improve Initial	106.53	59.44	128.67	108.81	408.69	338.56	219.03	325.55
GA	105.65	62.83	137.77	101.84	371.69	723.92	435.95	703.82
LS	99.56	59.6	139.35	77.45	408.25	389.90	283.41	274.33

Table 5.9: Best path cost

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Improve Best	106.48	58.87	140.09	102.79	460.68	289.20	254.37	315.29
LS operator	123.27	57.32	156.26	112.12	504.47	962.03	571.04	555.91
Improve Initial	124.59	69.20	150.96	111.97	488.74	962.03	571.04	336.56
GA	119.41	68.53	151.92	117.07	437.45	1274.95	663.20	817.06
LS	205.61	116.88	299.24	141.47	1923.6	505.37	707.45	348.63

Table 5.10: Average path cost

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Improve Best	9.77	9.66	13.00	31.51	113.61	479.21	91.25	1167.44
LS operator	12.67	16.43	34.61	30.86	1196.70	479.21	91.95	532.47
Improve Initial	3.58	7.31	7.14	9.52	168.68	268.95	58.43	430.62
GA	0.68	5.43	2.20	4.05	24.01	3.58	5.89	2.69
LS	10.88	14.96	9.66	30.82	103.40	70.81	18.06	181.78

Table 5.11: Average CPU time

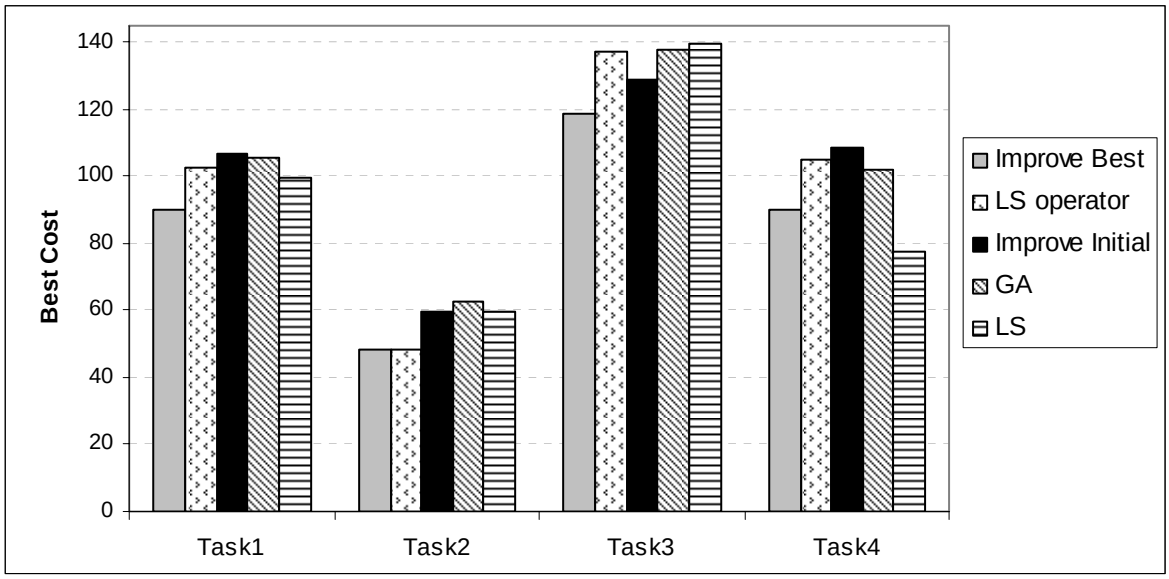


Figure 5.24: Best Paths for the implemented techniques.

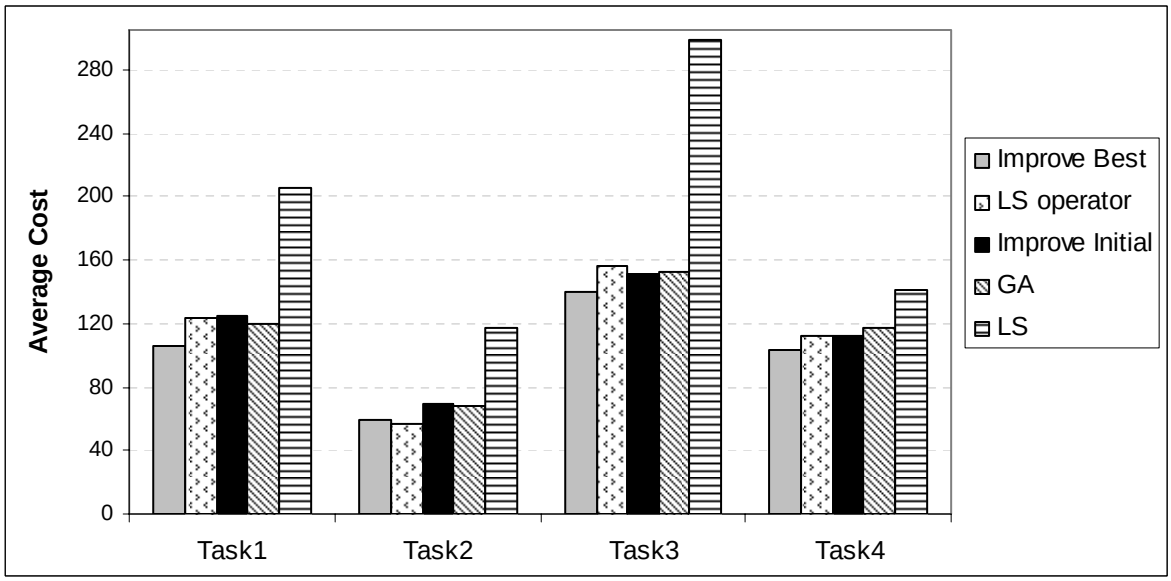


Figure 5.25: Average Cost for the implemented techniques

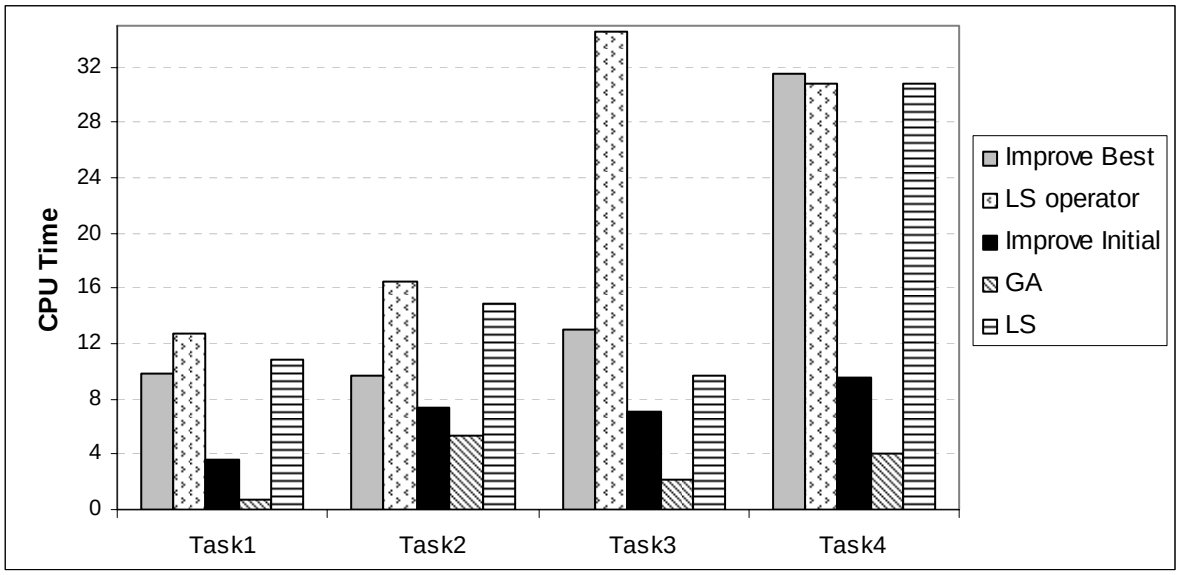


Figure 5.26: Average CPU Time for the implemented techniques

Results obtained show that the "improve best technique" gives the best results in terms of solution quality. The strength of this technique is that the GA is left without interruption (i.e., greedy search) to slowly search and explore the entire space. Once the GA terminates, the LS takes over and fine-tunes the best path found so far. Applying the LS as an operator is the worst in terms of the computational time and causes no improvement in the solution quality as it is expected. This can be attributed to the fact that following the application of the LS operator to individuals the later dominate the population and prevent any further exploration to take place. The same observation can be made to the "improve initial technique" where the initial enhanced population controls the search quality and leads to premature convergence.

5.3 Summary

A novel Local Search (LS) for solving the path planning problem is presented in this chapter. The LS algorithm uses an iterative improvement process by generating new

neighbourhood solutions by changing node coordinates. Results obtained indicate that the LS was successful in solving the path planning problem. Furthermore, in some of the tested benchmarks, LS obtained solutions superior to those obtained by the GAP. However, the LS can easily get stuck in a local minimum, and more starting points are required to obtain a high quality solutions. Consequently, a Memetic Algorithm (MA) is proposed and implemented by combining the GAP with LS. Different approaches are used in this research to combine the two algorithms. Results prove that applying the LS as a fine-tuner to the GAP final result, tends to improve the algorithm performance to obtain improved solutions.

Chapter 6

Conclusions

The path planning problem for mobile robots in dynamic environments is a difficult optimization problem. This type of problem is present in almost all mobile robots applications. The path planning problem is an ordering problem, where a sequence of configurations is sought, beginning from an initial location and ending at a specific destination. A robot searches for an optimal or near-optimal path with respect to the objectives of the problem.

The criteria of the path planning problem include distance, time, energy, smoothness and safety. The distance is the most common criterion. However, common path planning approaches do not take into consideration path safety and smoothness. Safety constraints are necessary for both the robot and its surrounding objects and can increase costs. Smoothness is also another important constraint, because of the bounded turning radius in most mobile robots. For example, car-like robots have this constraint due to the mechanical limitations of its steering angle.

6.1 Conclusions

This thesis introduced a Genetic Algorithm approach for solving mobile robot path planning problems in static and dynamic environments. The developed Genetic Algorithm Planner (GAP) utilizes variable-length chromosomes for the path encoding. A floating point solution encoding is utilized to provide the algorithm with the required flexibility to explore the solution space. A generic fitness function is used to combine the objectives of the problem. The paths are evolved by using specially designed random operators and specific-domain knowledge operators.

The GAP was implemented in C programming language and tested with a variety of benchmarks and tasks, which include simple, crowded, structured, and unstructured environments. The GAP was 100% successful in obtaining high quality solutions in both static and dynamic environments. In addition, different approaches were investigated to enable the algorithm to function correctly in both static and dynamic environments. Experimental results indicate that the "memory based technique" is the best among the other investigated approaches in this study.

Furthermore, to speed-up the algorithm, the GAP is parallelized by using island-based GA approach (IGA). In this model of computation each processor executes the GAP based on its own subpopulation with occasional exchange of high quality individuals with their neighbours' processors. The Message Passing Interface (MPI) library was utilized to implement the IGA. Results show a near linear speed-up in the computational time. Since the entire population is divided into subpopulations the solution quality was also affected. However, results reveal that if no improvement is made in the solution quality, the obtained

results are still of high quality (i.e., not far from the result that is obtained by using the serial GAP).

It can be concluded that, the GA is able to solve the path planning problem efficiently; however, the GA is not capable of fine-tuning the search. Therefore, a heuristic Local Search (LS) algorithm was proposed in this dissertation. The LS algorithm uses the iterative improvement process by generating new neighbour solutions which are generated by changing the node coordinates. LS was successful in solving the path planning problem, and in some of the tested benchmarks, LS obtained solutions superior to those obtained by the GAP. However, the LS can easily get stuck in a local minimum, and more starting points are required to obtain good solution. Consequently, a Memetic Algorithm (MA) was implemented by combining the GAP with the LS. Different approaches were used in this research to combine the two algorithms. Results obtained prove that applying the LS as a fine-tuner to the GAP final result, tends to improve the algorithm performance to obtain adequate solution quality in reasonable time.

6.2 Future work

One of the interesting directions for future work involves improving the adaptiveness of the GAP such that the GA parameters are automatically initialized and tuned according to the nature of the problem. The GA has various parameters that require proper tuning, and usually, this tuning process is achieved heuristically. Accordingly, developing an adaptive parameter tuning mechanism helps the GA to adjust the parameters to obtain better results. Other improvements can be made to dynamically change the GA operators and their

behaviour according to the system status. Also, a Graphical User Interface (GUI) can be developed to facilitate the communications between the user and the algorithm.

Although all the obtained results are promising, it is essential to test the proposed algorithm in a real stochastic mobile robot environment. One candidate system for such a test is *ROBOSOCCER*[1]. The role of the GAP in this system would be to find a good route for the various robots, *players*, according to the environment, *soccer field*, and to the given task.

Lastly it is recommended that the same approach be expanded to a higher degree of freedom (i.e., Arm Robots).

1- http://wolfman.eos.uoguelph.ca/~robosoccer/

Appendix A

Benchmarks Files

The benchmarks files are ASCII text files. The fields in the files are either space or linefeed delimited. The files are structured as follows:

The First two fields represent the map boundary (x_{max}, y_{max}), followed by the number obstacles in the environment. After that each obstacle record is presented, the first filed in the obstacle record is the number of vertices, followed by the vertices in order. Each vertex is presented by its x and y coordinates respectively.

All the benchmarks used in this study are given here

```
                                    ****************** Task 1********************
40    40                                                                  //Map boundary
3                                                                         // Number of obstacles
4     10  20    15  20    15  5     10  5             // Obstacle 1
4     20  30    18  34    11  34    10  30            // Obstacle 2
3     28  16    20  18    28  10                      // Obstacle 3

                                    ****************** Task 2 ********************
40    40
10
3     8   14    6   12    10  10
5     17  10    20  12    12  14    16  8     18  9
4     26  18    28  12    32  14    29  14
4     6   26    6   24    10  20    12  22
4     19  20    22  19    18  18    18  24
4     28  26    28  22    36  24    34  30
3     21  33    20  30    24  29
5     4   37    40  40    0   40    0   32    6   32
5     11  34    14  35    12  32    9   30    8   32
3     36  21    32  18    38  15
```

********** Task 3 **********

```
40  40
14
4   8   12  6   12  6   4   10  4
4   16  24  18  18  10  14  12  24
3   12  14  16  8   18  10
5   26  16  28  12  32  14  26  21  22  16
3   7   15  4   20  10  18
4   6   30  6   24  10  20  10  22
4   19  25  20  20  18  21  18  24
3   28  22  36  24  34  30
3   21  33  20  30  24  29
4   40  0   40  6   35  6   32  0
4   40  40  40  32  35  32  32  40
4   0   40  8   40  4   32  0   32
3   16  31  14  37  10  34
3   36  21  32  20  38  9
```

********** Task 4 **********

```
100 100
6
8   9   55  21  55  25  73  23  76  17  73  15  76  12  73  8   76
12  34  41  40  41  40  47.5 59  47.5 59  31  65  31  65  61  59  61
    59  55  40  55  40  71  34  71
5   58  70  72  67  62  80  52  73  47  62
6   70  15  75  20  80  15  85  25  90  35  75  35
7   28  15  44  15  60  15  50  27  37  25  43  32  23  27
5   75  80  85  95  95  85  90  75  85  75
```

********** Task 5 **********

```
160 160
24
3   140 36  130 30  130 40
3   80  66  93  58  78  43
4   155 155 155 145 141 145 141 155
4   125 140 144 140 144 128 125 128
4   141 66  151 66  151 51  141 51
4   117 54  124 47  117 38  110 48
4   36  154 47  143 39  138 32  147
4   107 75  56  75  56  112 107 112
4   97  113 75  121 92  135 109 123
4   115 155 115 145 85  145 85  155
4   56  152 66  147 59  129 51  133
4   39  128 39  108 31  113 31  128
4   23  140 17  128 13  132 20  143
4   14  120 23  112 18  105 10  113
```

```
4     8   103   42  103   42   58    8   58
4    12    50   22   43   19   36    8   43
4   110   105  141  105  141   85  110   85
4    79    35   97   27   90   19   71   23
4    45    95   52   90   52   80   45   80
4    48   124   53  124   53  108   48  108
4   105    70  120   70  120   65  105   65
4     5     5    5   20   20   20    20   5
4    30    30   60   15   45   10   30   15
5    69    65   62   42   44   39   38   43   51   66
6    38    27   38   20   62   20   62   11   30   11   31   27
```

********** Task 6 **********

```
100   80
5
4    20   80   24   80   24   36   20   36
4    33   44   37   44   37   00   33   00
4    46   80   50   80   50   36   46   36
4    62   44   66   44   66   00   62   00
4    76   80   80   80   80   36   76   36
```

********** Task 7 **********

```
40   40
3
8    20   4    20   10   32   10   31   9    21   9    21   5    31   5    32   4
4    25   25   25   15   15   15   15   25
8    19   36   19   30   8    30   9    31   18   31   18   35   9    35   8    36
```

********** Task 8 **********

```
100  100
1
20   80   15   20   15   20   70   70   70   70   30   35   30   35   55   55   55
     55   45   45   45   45   40   60   40   60   60   30   60   30   25   75   25
     75   75   15   75   15   10   80   10
```

References

[1] Stuart, Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, New Jersey: Prentice Hall, 1995

[2] G. Lazea, and A.E. Lupu, "Aspects on Path Planning for Mobile Robots," Intensive Course on Computer Aided Engineering in Flexible Manufacturing, Bucharest, Romania, May 19-23, 1997.

[3] M. Sharir, "Algorithmic Motion Planning in Robotics.", *IEEE Computer*, Vol. 22, No. P. 9-20, 1989.

[4] J.-C. Latombe, *Robot Motion Planning* , Kluwer Academic Publishers, Boston, MA, 1991.

[5] Y. K. Hwang and N. Ahuja, Gross Motion Planning - A Survey, *ACM Computing Surveys*, Vol. 24, No. 3, P. 219-291. 1992

[6] Canny J.F., *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1988.

[7] Wasserman, P.D., Neural Computing, Theory and Practice, Van Nostrand Reinhold , New York, 1989.

[8] M. Yagiura and T. Ibaraki, "On metaheuristic algorithms for combinatorial optimization problems", *Transactions of the Institute of Electronics, Information and Communication Engineers*., Vol. J83-D-1, No. 1, P. 3-25, 2000.

[9] C.Blum, A.Roli. "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison." *ACM Computing Surveys*, Vol.35, No. 3, 2003

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press. 2nd edition, 1992.

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

[12] Jianping Tu and Simon X. Yang. "Genetic algorithm based path planning for mobile robot", Proc. of the IEEE International conference on robotics & automation, Vol. 1, P. 1221-1226, 2003.

[13] D. Coit and A. Smith and D. Tate "Adaptive penalty methods for genetic optimization of constrained combinatorial problems," *INFORMS Journal on Computing*, vol. 8, No. 2, P. 173-182.

[14] E. Cantu-Paz, "A Survey of Parallel Genetic Algorithms," *Calculateurs Paralleles*, Vol. 10, No. 2, P. 141-171, 1998.

[15] E. Cantu-Paz, *Efficient and accurate parallel genetic algorithms*, Kluwer Academic Publishers, 2001.

[16] T. Lozano-Perez, and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", *Communications of the ACM*, Vol. 22, No. 10; P. 560-570, 1979.

[17] M. Sharir, "Efficient algorithms for planning purely translational collision-free motion in two and three dimensions", *Proc. Of the IEEE Internat. Conf. on Robotics and Automation*., Vol. 4, P. 1326-1331, 1987.

[18] J. Hopcroft, J. Schwartz, and M. Sharir. "Planning, geometry, and complexity of robot motion", Ablex Publishing, Norwood NJ, 1987.

[19] N.J. Nilsson. "A Mobile Automaton: An Application of Artificial Intelligence Techniques." *Proc. of the 1st International Joint Conference on Artificial Intelligence*, P. 509-520, 1969.

[20] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications.* Springer-verlag, Berlin, 1997.

[21] Kavraki L.E., Svestka P., J-C. Latombe, Overmars, M.H. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." *IEEE Transactions on Robotics and Automation*, Vol. 12, No. 4, P. 566 - 580, 1996

[22] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. of Robotic Research*, Vol. 5, No. 1, P. 90-98, 1986.

[23] F. Janabi-Sharifi, and D. Vinke, "Integration of the artificial potential field approach with simulated annealing for robot path planning," *Proc. of the IEEE International Symposium on Intelligent Control*, P. 536-541, 1993

[24] Yuval Davidor, *Genetic algorithms and robotics: a heuristic strategy for optimization*, World Scientific, 1991.

[25] Yuval Davidor, "A genetic algorithm applied to robot path-planning," Technical Report CS90-01, The Weizmann Institute of Science, 1990.

[26] T. Shibata, T. Fukuda, K. Kosuge, F. Arai, "Selfish and Coordinative Planning for Multiple Mobile Robots by Genetic Algorithm", *Proc. of the 31st Conference on Decision and Control*, Vol.3 , P. 2686-2691, 1992.

[27] M.K. Habib and H. Asama, "Efficient method to generate collision free paths for an autonomous mobile robot based on new free space structuring approach," *Proc. of IEEE/RSJ Int'l Workshop on Intelligent Robots and Systems IROS'91*, Vol. 2, P. 563-567, 1991.

[28] T. Shibata and T. Fukuda, "Intelligent Motion Planning by Genetic Algorithm with Fuzzy Critic", *Proc. of the 8th IEEE International Symposium on Intelligent Control*, P. 565- 570, 1993.

[29] C.H.Leung, A.M.S. Zalzala, "A genetic solution for the motion of wheeled robotic systems in dynamic environments." *Int. Conf. on Control*, Coventry, p.760-764, 1994.

[30] H.-S.Lin, J. Xiao, and Z. Michalewicz, "Evolutionary Navigator for a Mobile Robot," *Proc. IEEE Int. Conf. Robotics and Automation*, Vol. 3, P. 2199-2204, 1994.

[31] J. Xiao, Z. Michalewicz, L. Zhang, and Z. Lixin, "Adaptive Evolutionary Planner/Navigator for Mobile Robots," *IEEE Transactions on Evolutionary Computation*, Vol. 1, P. 18-28. 1997.

[32] J. Xiao, Z. Michalewicz, L. Zhang, and Z. Lixin, "Evolutionary Planner/Navigator: Operator Performance and Self-Tuning," *Proc. of the IEEE ICEC*, p. 366-371, 1996.

[33] K. Trojanowski, Z. Michalewicz , J. Xiao, "Adding Memory to the Evolutionary Planner/Navigator," *Proc. of the 4th IEEE ICEC*, P. 483-487, 1997.

[34] M. Chen and A. M. S. Zalzala, "A genetic approach to motion planning of redundant mobile manipulator systems considering safety and configuration," *Journal of Robotic Systems.*, Vol. 14, No. 7, P. 529–544, 1997.

[35] M. Gemeinder, M. Gerke, "GA-based path planning for mobile robot systems employing an active search algorithm," *Applied soft computing journal*. Vol. 3, No. 2, P. 149-158, 2003.

[36] T. Pavlidis. *Algorithms for Graphics and Image Processing*, Computer science press, 1982.

[37] J. O'Rourke, *Computational geometry in C*, Cambridge Univ. Press 2nd edition. 1998.

[38] R. Hinterding, H. Gielewski, and T.C. Peachey. "The nature of mutation in genetic algorithms." *In Proceedings of the Sixth International Conference on Genetic Algorithms*, L.J. Eshelman, ed. P. 65-72. 1995.

[39] J. Branke, Evolutionary approaches to dynamic optimization problems - Introduction and recent trends. *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, P. 2-4, 2003.

[40] J. Xiao, L. Zhang, and Z. Michalewicz, "On Topological Diversity and Multiple Path Planning," *in the 2nd International Conf. on Computational Intelligence and Neuralscience*, P. 10-13, 1997.

[41] W. Gropp, L. Ewing and A. Skjellum, *Using MPI, portable parallel programming with the Message Passing Interface*, The MIT Press, Cambridge, England, (1994).

[42] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms," Tech. Rep. No. 790, Caltech Concurrent Computation Program, California Institute of Technology, 1989.