**UNIVERSITY OF OSLO**
**Department of Informatics**

# Multi-objective Evolutionary Path Planning with Neutrality

Master Thesis

Eivind Samuelsen

**May 8, 2012**

# Multi-objective Evolutionary Path Planning with Neutrality

Eivind Samuelsen

May 8, 2012

# Abstract

One of the main challenges when developing mobile robots is path planning. Efficient and robust algorithms are needed to produce plans for the movements of the robot. Many classical path planning algorithms depend on geometrically simple environments to achieve good performance, otherwise the paths produced tend to be far from ideal - especially when the paths are to be optimized for multiple objectives.

Evolutionary algorithms have proved to be able to optimize paths in complex environments in a way that is easily adapted to solving multiple objectives. However, the solution space in path planning problems is very complex, marred by infeasible regions and local optima. This makes finding true optimal solutions difficult.

In the last decade or so, neutrality - the ability to generate the same solution in multiple ways - has gained attention in evolutionary computing. Some work indicate that neutrality improves optimization in problems with difficult solution spaces.

In this thesis an evolutionary algorithm for path planning with a neutral chromosome encoding is proposed. The chromosomes are encoded as sets of points, which are translated into roadmap graphs, which are then traversed to find one or more optimal solutions within the graph. To best represent the various strenghts of each chromosome, selection methods are proposed that let a number of solutions compete collectively for their chromosome.

The algorithm has been implemented and tested thoroughly on four different environments, first for single-objective optimization, and then for multi-objective optimization problems. A comparison has been done to a reference algorithm that is similar but without a neutral solution representation.

The proposed algorithm is not very efficient when optimizing distance only, but shows promising performance in multi-objective problems where other objectives are involved. The performance is significantly more robust than the reference algorithm in an environment that has many routes that separate and cross multiple times, finding a near-optimal solution up to 27% of the time, while the reference algorithm finds solutions of the same quality only 7% of the time.

# Contents

# List of Figures

# List of Tables



# List of Algorithms

# Chapter 1

# Preface

Thanks go to my two supervisors, Kazi Shah Nawaz Ripon and Kyrre Harald Glette for all the support and advise you have given me during the process of making this thesis and for encouraging me to try to write a paper along the way. The resulting trip to the AROB conference in Japan was very inspirational, not to mention loads of fun.

I also want to thank my parents and familty, and especially my wife, who has been very loving and supportive through all of this, including inspiration and our continued heated discussions. I might not be where I am today if it were not for your support.

Also, in no particular order: Excess, Outracks, Portal Process, Gammel Opland af 1891, BitFlavour, Odd, PlayPsyCo, Youth Uprising, Indigo, Panda Prowess, Darklite, Loonies, SystemK & nonoil et al: pants off!

# Chapter 2

# Introduction

This chapter will give a short explanation of the motivation and goals of this thesis, followed by a brief outline of the rest of the text.

## 2.1 Motivation

For several decades, robots have been used in the industry to automatize physical labor such as assembly or logistics. These robots are highly specialized machines, contained in their own separate environment so that they can be programmed on an offline basis.

However, there is also a demand for automation outside of the contained environments of the factory floor. Robots could perform construction tasks, such as painting or laying tiles on large surfaces. They could do heavy lifting or other physical labor in the health care industry. Or they could be used for logistic tasks in office buildings. These are just some of many concieveable applications. All these tasks require safe and robust ways of planning paths and motions in complex environments. Most of these environments would have people or other autonomous agents present, so the robot must be able to do the planning in real time.

As robots move into less controlled environments, the need to handle multiple objectives simultaneously also increases. Moving objects and inaccurate measurements of the environment requires that trade-offs between path length and safety are considered. Mobile robots need a mobile power source, so it becomes necessary to consider the physical work needed to move along different paths. When the robot shares the environment with other agents, especially people, visibility and predictability becomes important factors as well.

Evolutionary algorithms have shown good results in real-time path planning in dynamic environments [51]. They are also easily adapted to solving multi-objective problems, and there are examples of multi-objective evolutionary path planners [25, 1]. However, path planning problems often have solutions spaces that can be very hard for evolutionary algorithms to navigate in. Obstacles can create barriers between different local optima that can be close to impossible to traverse.

Neutrality is present in an evolutionary algorithm if it is able to search

along networks of equally good solutions. If implemented correctly, these networks may improve the chances of finding solutions that would otherwise be very difficult to find for an evolutionary algorithm. However, little or no neutrality is present in existing evolutionary path planners.

## 2.2 Research Goals

The primary goal of this thesis has been to design and implement a multi-objective evolutionary path planner with neutrality, and then study the performance this algorithm.

Based on general work on neutrality [16] it is hypothesized that a neutral path planner should be able to tunnel across these barriers through neutral networks, thereby being less reliant on having solutions on each side of the barriers initially. To examine whether this is the case was therefore an important goal.

Since the environments can be expected to change over time, run time performance is important. A neutral solution coding will in the path planning case necessarily require additional computation compared to a conventional representation. Therefore, advantage in the quality of solutions produced by the algorithm, if any, must be seen in close relation to the run time used.

## 2.3 Outline

The thesis is separated into five chapters, comprising the background, methods, experiments, results and conclusion of the thesis. Chapter 3 gives a introductory summary of the theory and works related to this thesis in the fields of path planning, evolutionary algorithms and multi-objective optimization problems.

Chapter 4 proposes a new, hybrid evolutionary path planning algorithm, designed to increase robustness and quality through neutrality. The algorithm is first described as a single-objective path planner and then expanded into a more general multi-objective approach.

Chapter 5 describes the experimental setup and tests used to compare the performance of the proposed algorithm to a more standard evolutionary path planner, and in chapter 6 results from these tests are presented. Finally, chapter 7 concludes the thesis with discussion of the results and some final concluding sections.

# Chapter 3

# Background

This chapter attempts to give an overview of the fields of path planning, evolutionary algorithms and multi-objective optimization, including pointers to recent work in these fields, especially where they intersect. In tradition with other literature relating to evolutionary algorithms, algorithms which are not population-based or otherwise based on principles similar to evolutionary algorithms are called 'classical' to separate them from evolutionary variants.

## 3.1 Path Planning

In this section we will first introduce path planning viewed as an optimization problem. Then the concept of a configuration space will be introduced. Finally, a short summary of popular classical path planning algorithms is given.

### 3.1.1 Definition

In essence, path planning consists of finding a path for some moving object though an environment. Ideally, we want to find the optimal path for the given circumstances. The moving object - the subject of the path planning problem - could be a robot or autonomous vehicle, or maybe some computer game character or other simulated agent. The indtroductory part of [32] can be referred to for more detailed examples.

One can define path planning as the problem of optimizing

$$F(\gamma), \qquad \gamma \in C$$

Where $C$ is the set of feasible paths in the given environment. $F(\gamma)$ is some function which quantifies how good the curve $\gamma$ is according to the problem circumstances. A path is here mathematically the same as a curve - a set of points that can be parametrized as a function from $\mathbb{R}$ to some space convenient for describing relevant information about the subject.

A path is feasible if it obeys all constraints upon the subject. For example, the subject should usually not collide with any obstacles in the environment, and the path should obey the physical limitations of the

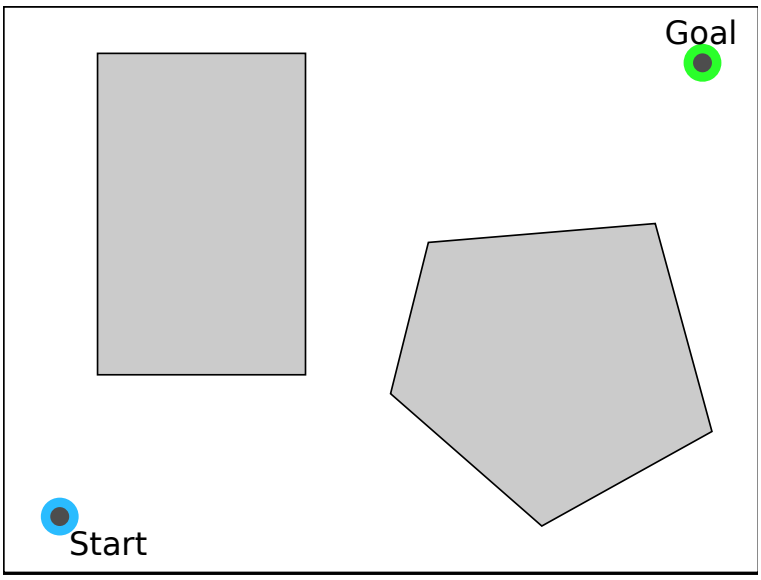



**Figure 3.1:** An example of a path planning problem. Find a path from start to goal. A path $\gamma$ is in $C$ if and only if it is inside the environment boundaries and do not intersect any obstacles (the gray areas).

subject such as rates of acceleration and turning. An example of a simple environment for a path planning problem is illustrated in Figure 3.1.

Commonly, $F(\gamma)$ is just the curve length of $\gamma$, but it could also be an estimate of the work needed to traverse the path, or maybe some result of a detailed simulation of the subject trying its best to follow the path. If there are more than one quality we want to optimize, $F(\gamma)$ can be a vector function. It then becomes a multi-objective optimization problem, which can have any number of optimal solutions. How to handle this will be discussed in Section 3.3.

### 3.1.2 Configuration space

When considering the movements of a path planning subject it is useful to do so in terms of its mechanics and degrees of freedom. In robotics, the base position of the a robot, its rotation and the rotation or translation of all its links is called the configuration of the robot. The set of all possible configurations constitute the configuration space, or C-space [35].

Working in C-space simplifies the path planning problem, because the subject here consists of a single point. All obstacles in the environment then becomes regions of C-space that are infeasible robot configurations. If the subject is a simple translational robot, i.e. a robot that does not rotate, have no links and can move in the same speed in any direction at any time, then C-space is identical to world space except obstacles in C-space are "padded" with the shape of the robot. For robots with many complex parts making an precise translation into C-space can become time-consuming. But it is often relatively easy to at least find a worst-case estimate, which is sufficient for most purposes.

A curve through C-space would constitute a transition between configurations. A configuration is feasible if it is in unobstructed C-space, that is, it is not inside any obstacles. However, in addition to obstacles, there might be dynamics constraints on the subject that limit its movements. These constraints are not captured by C-space, so a configuration transition might not be feasible even if every point on the curve is feasible. When the dynamic constraints are not too limiting then one can assume that any line segment not intersecting any obstacle is a feasible configuration transition for path planning purposes. It is then left up to some kind of low-level control system to make the subject follow the path to its best ability.

When dynamic constraints cannot be safely ignored during path planning one can expand C-space into state-time space [14], created by adding a time dimension plus one dimension for each constrained dynamic variable to C-space. Then the dynamic constraints can be translated into infeasible areas of state-time space, in effect representing the constraints as state-time obstacles.

### 3.1.3 Overview of Classical Algorithms

This section will give a brief overview of the main families of algorithms that have been developed to solve path planning problems. A more detailed overview of the algorithms can be found in [31] and [32].

The most basic distinction between the different algorithms is between local and global planners. Global planners find a complete path based on information on the entire environment. Local planners only proposes a way to get onward from the current state given the immediate surroundings. By applying a local planning algorithm repeatedly one may then plan a full path. Only one local planner - the artificial potential fields algorithm - will be discussed here.

### 3.1.4 Artificial Potential Fields

Artificial potential fields (APF) is a local path planning algorithm based on the idea of guiding the subject around using attractive and repulsive artificial field functions. One regards the goal as the focus of an attractive potential field and the obstacles as having a repulsive potential field associated to them. A total potential field function made from summing up all the individual field functions can then be calculated and a path is generated by doing a gradient descent of this field, as illustrated in Figure 3.2.

First formulated in [27], it was soon discovered that it was impossible to guarantee that there were no local minima in the APF except for the goal, or even that the goal is a local minimum when the goal lies near an obstacle.

Since gradient descent terminates once a local minimum is reached, one cannot guarantee that this algorithm will find a path to the goal in all situations. A number of workarounds for this has been proposed, improving the performance of the algorithm by making local minima more

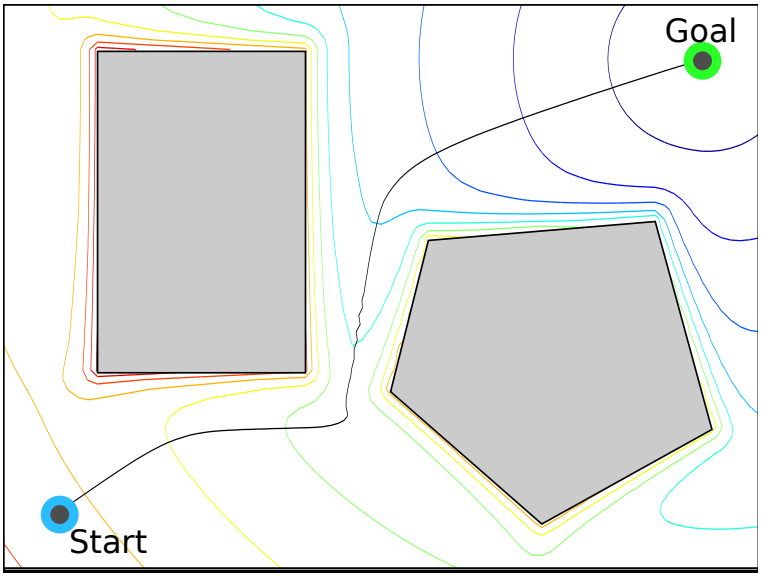



**Figure 3.2:** An artificial potential field. The black curve is the path obtained by a gradient descent from the start point.

unlikely [28] and using techniques for helping the gradient descent escape local minima [31], but it is still an unreliable algorithm in many situations.

While unreliable for complex environments, it is very useful as a local planner since it has low complexity - for each point along the path one only needs to evaluate a simple gradient function of the goal and each nearby obstacle once. Thus it can be useful for local obstacle avoidance between waypoints laid out by a global path planner, or as an alternative connectivity check between points in the environment in some of the algorithms discussed below.

### 3.1.5 Global Planners

Global planners find solutions by creating a suitable representation of the environment and then searching for solutions in that representation. Usually the representation is some kind of graph with feasible configurations as nodes and feasible configuration transitions as edges. Finding the shortest path from start to goal is then a matter of running a graph traversal algorithm such as A* [20] from the start node to the goal node. This kind of graph is often referred to as a roadmap.

The different roadmap-based algorithms differ in how the nodes are distributed in the environment or C-space, and how they are connected. A main distinction here is whether the graph construction method can guarantee that the graph will contain a path from start to goal if such a path exists, or rather, under what conditions such a guarantee can be given.

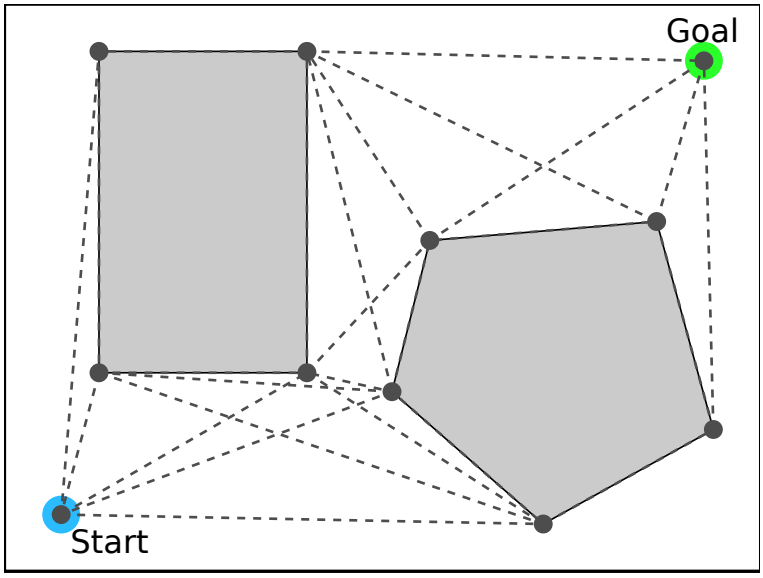



**Figure 3.3:** A visibility graph. Each corner in the environment generates a node. One can visually confirm that the graph contains the shortest path from start to goal.

### 3.1.6 Combinatorial Path Planners

Combinatorial path planners create a roadmap by analyzing the geometry of the entire environment and all obstacles. This includes the visibility graph, Voronoi diagram, and other algorithms that divide or analyze the environment in some exact way.

The visibility graph has a node for each corner on each obstacle, and an edge for each pair of nodes that are mutually "visible", e.g. the straight line between them is a feasible transition [36]. The visibility graph for the environment in Figure 3.1 is shown in Figure 3.3.

This algorithm is often used as a reference algorithm because the shortest path in the visibility graph is guaranteed to be the shortest path possible if the environment is two-dimensional. For higher-dimensional spaces that property no longer holds, and the number of nodes will often become very large in high-dimensional geometry.

Variants of the visibility graph, such as the art gallery algorithm described in [37] amend the performance problems of the visibility graph by reducing the number of nodes and edges used, while still keeping the main concept of making sure that all parts of unobstructed space are "visible". However, these variants will usually not contain the shortest possible path.

A Voronoi diagram algorithm finds all points in the environment where there are two or more obstacles that are equally close, as shown in Figure 3.4. This set of points forms the Voronoi diagram of the environment. The Voronoi diagram will consist of curves and intersections, forming corridors throughout C-space. The resulting roadmap has a node for each intersection in the diagram and an edge for each curve [3, 5]. This roadmap

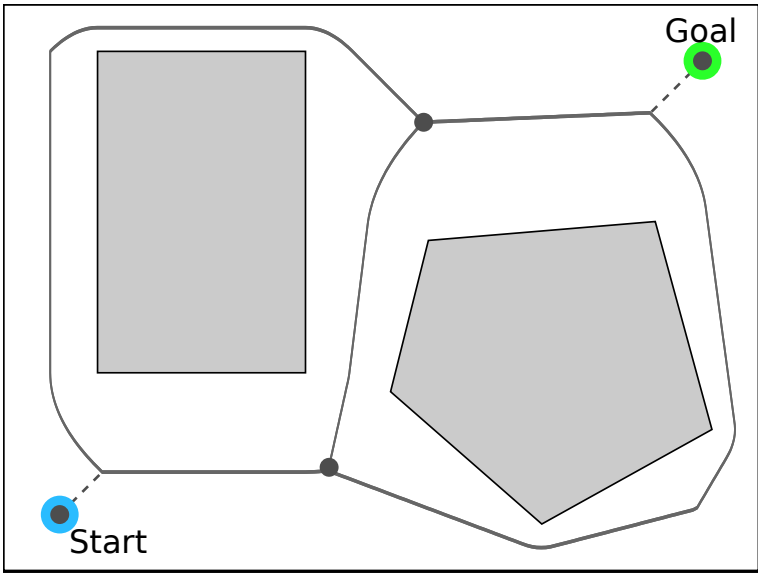



**Figure 3.4:** A Voronoi diagram. The gray curves are equally far from the two or more nearest obstacles. By following these curves one can get from start to goal while staying as far as possible from any obstacles on the way.

will not contain the shortest path in most cases, but it has the property of generating maximally safe paths - paths which stay as far from any obstacle as possible at all times.

All the combinatorial algorithms have in common that the way they create the graph guarantees that you can find a feasible solution if one exists. Thus they are said to be *complete*. However, for many-dimensional spaces or environments with high geometrical complexity they can become impractically slow, because of the complexity of creating the graph, or because the graph becomes too large to search efficiently.

### 3.1.7 Sampling-Based Planners

Sampling-based planners do not analyze the environment geometrically in the way combinatorial planners do, but rather sample them. Thus they cannot guarantee to always find paths in the same way the complete planners do. However, they do guarantee that if enough samples are taken they can find a path from one point to another if any such paths exists. They are said to be *resolution-complete* or *probabilistically complete*, depending on whether the sampling is done in a systematic fashion or by some probabilistic method.

Though they are less reliable than the combinatorial algorithms, they are less dependent upon the complexity and accuracy of the obstacles geometry, because they only need to be able to find out whether the points they sample are obstructed or not and some way of determining whether a straight-forward path can be taken between them. Combinatorial algorithms, on the other hand, need the surface of the obstacles to be defined in a manner they can process. Furthermore, real-world geometry is

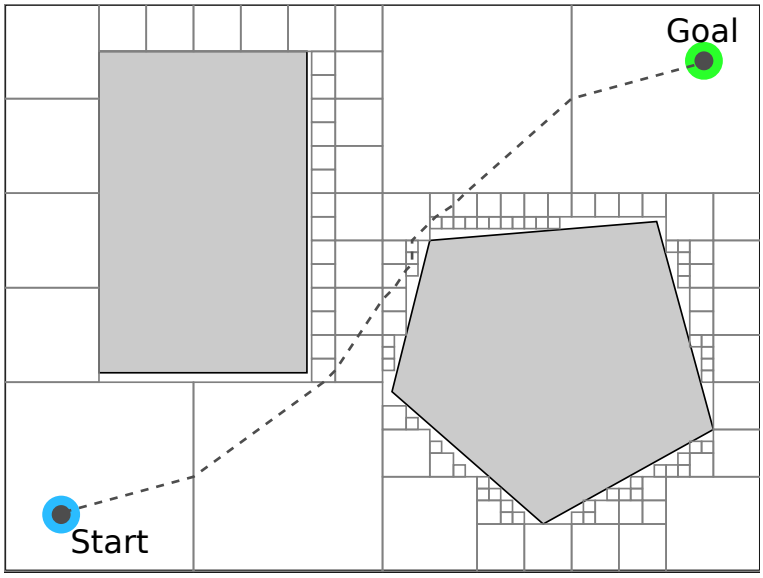



**Figure 3.5:** An adaptively sampled environment. Progressively smaller squares are tested for intersection with the environment until unobstructed squares are found. One near-optimal solution through the data structure is shown.

usually so complex you have to simplify it in order to make combinatorial algorithms run with decent performance, and then the completeness guarantee given by those algorithms is weakened.

The simplest way to sample a space is to sample evenly along every axis, creating a grid. The grid will have obstructed and unobstructed nodes, and one would assume that all straight lines between two neighboring unobstructed nodes are feasible transitions. This is a popular approach in many situations, because it enables efficient memory storage and makes the underlying traversal algorithm easy to implement.

The resolution necessary to find paths through tight passages will often give rise to an enormous amount of nodes, though, and memory usage quickly becomes high when C-space has high dimensionality. These problems can be amended by using an adaptive resolution sampling scheme like quad-trees, effectively taking more samples near the borders of obstacles [22]. An example of this is shown in Figure 3.5.

A different approach to sampling C-space is to take random samples [26, 41]. One usually discards obstructed samples and keep sampling until a certain number of unobstructed samples are found, or until a path has been found. These samples are then connected to other samples nearby to form a roadmap, as shown in Figure 3.6. Because of their probabilistic nature these algorithms are called probabilistic roadmaps (PRM).

Uniformly random sampling will result in many of the same trade-offs between graph size and completeness as the grid-based algorithms, and so different methods have been designed to amend this. In [4] two points are generated at a time at a normally distributed distance from each other. A point is added to the graph only if it is feasible and the other is not, thus

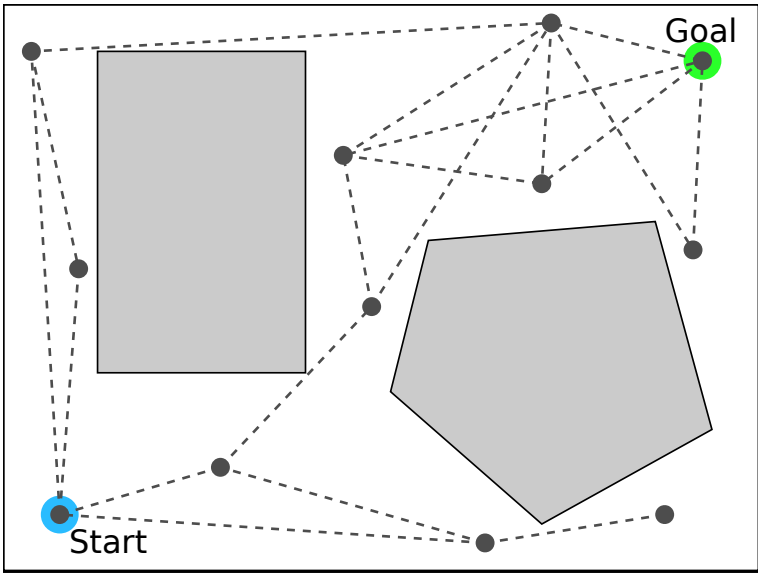



**Figure 3.6:** A probabilistic roadmap. Nodes are generated randomly. All non-obstructed nodes are connected by visibility.

sampling mostly near edges.

In [40] a new sample is only added to the graph if it is not visible from any node already in the graph or if it connects two previously disconnected parts of the graph. This ensures that large, open areas in C-space do not get more samples than necessary. An example is given in Figure 3.7.

## 3.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) give a general framework for population-based optimization algorithms, inspired by evolution in nature and adopting terminology from evolutionary biology and genetics. The general idea is to cross a set of solutions to a problem, letting information about which solutions are better and which are worse guide the creation of iteratively better sets of solutions [11].

Besides good evidence that the evolutionary processes that has inspired these algorithms have worked out well in nature, there has been done some work to explain why evolutionary algorithms can optimize efficiently even when little is known of the problem to be optimized except for a fitness function and a good way to represent a solution. A summary of that explanation can be found in [30].

The following sections will give a short overview of evolutionary algorithms, with focus on the variant most relevant for path planning, genetic algorithms.

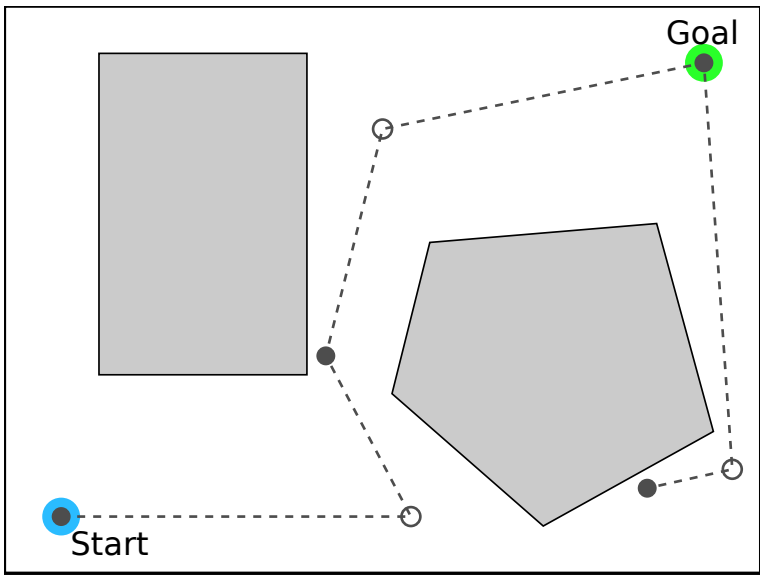



**Figure 3.7:** Visibility-based PRM. Nodes are generated randomly, but are only inserted if they cover some "unseen" area (visible by zero existing nodes) or connect two or more "seen" areas (visible by nodes in at least two unconnected sub-graphs). Solution quality can be a lot worse than with a simple PRM, but run time is greatly reduced because much fewer connectivity checks are needed.

### 3.2.1 General Principles

Since the first EAs were first suggested in [24] and became popular through genetic algorithms like [17] the terminology of evolutionary algorithms have slowly developed into what it is today. This section will give a quick guide to the terminology and then explain the basic principles of EAs. A more in-depth explanation of the terminology and principles can be found in [11] and also in the chapters introducing single-objective evolutionary algorithms in [7] or [6]. Algorithm 1 outlines the general flow of evolutionary algorithms.

An object that can be used to generate a solution is called a chromosome. The chromosome directly or indirectly represents a solution. The information stored in the chromosome is called the genotype. The solution it encodes is called the phenotype. Often both chromosome and solution is referred to as the individual, which is a kind of catch-all for both genotype and phenotype. It fits very well with the biological analogy: Each individual starts out out as just the genotype and then grows up into a genotype-phenotype pair.

A set of individuals is called a population. A generation refers to an iteration in the algorithm, and also to the population present at the beginning of that iteration. Each new generation partially or completely replaces the old population, generated from the old generation by evolutionary operators.

These operators take one or two parent individuals and creates new

**Algorithm 1** Rough sketch of an evolutionary algorithm

```
P ← generate-initial-population()
for all I ∈ P do
    fitness(I) ← calculate-fitness(chromosome(I))
end for
while ¬satisfied(P) do
    P′ ← ∅
    while |P′| < new individuals per generation do
        P′ ← P′ ∪ select-parents-and-generate-offspring(P)
    end while
    for all I ∈ P′ do
        fitness(I) ← calculate-fitness(chromosome(I))
    end for
    P ← select-survivors(P, P′)
end while
```

child individuals until a certain number of offspring (often denoted as $\lambda$) have been created. Which individuals that are used as parents is determined by a parent selection method.

After the offspring have been created, a surviving population is produced for the next iteration of the algorithm. The surviving population can be based either on just the new generation or on a combination of old and new individuals. Which individuals survive and which are replaced is then determined by some survival selection method.

At least one of the two selection stages are based on the fitness of the individuals. The fitness of an individual is a function $f(\boldsymbol{x})$ of its phenotype, and should measure of how well this solution solves the problem. Usually this function is simply the function we want to optimize.

Initially, few or maybe none of the solutions generated will be especially good, and in some cases even solutions known to be invalid are included. But, taken together the individuals give an indication as to what combination of parameters makes a solution better or worse.

By relating the fitness - which ideally is a measure of distance from optimality - of each individual to its chances to propagate its parameter values to the next generation, the population as a whole is gradually steered towards more optimal solutions. If part of the surviving population is reserved for the best individuals among both the old and the new generation the EA is elitist - a selection of the "elite" individual are kept through generations.

### 3.2.2 Evolutionary Operators

There are two basic kinds of evolutionary operators: The first, crossover, combines chromosomes from two individuals into one or more new ones. The second, mutation, changes one existing chromosome in some random fashion. Often these two are used in sequence or with some probability each to create one new individual, and some times several different variants of

crossover and mutation is used in the same algorithm.

Crossover creates a mix of two solutions so the new solution will be somewhere between or near its parents in the search space. Some selection procedure ensures that good solutions are selected for parenthood more often than bad solutions, and new solutions created this way are mostly somewhere around or between known good solutions.

Mutation, on the other hand, displaces a solution in a random manner in the search space. Mutation operators are traditionally intended to explore randomly, and introduce new variation to the population. This stands in contrast to the crossover operator which is meant to exploit information from earlier generations by searching near solutions that are known to be good, a process that generally decreases variation.

Some EAs have no crossover operator, and creates the next generation from mutation only. In that case the algorithm relies on which individuals are selected for mutation to do the exploitation and steer the population towards optimality.

Regardless of how the old population is exploited, exploitation is essential to the evolutionary process. Without some measure of exploitation evolutionary algorithms would essentially be the same as a random search. Therefore, it is also important that the initial population is diverse enough to cover all parts of the solution space, so that it is not dependent on a long string of random mutations before exploitation can begin to act towards the global optimum.

### 3.2.3 Genetic Algorithms

Genetic algorithms (GAs) was the first variant of population-based optimization to appear, with early works including [24]. In its original form the solution representation was a fixed-length binary string, inspired by the strings of DNA that make up genetic material, which also inspired the crossover and mutation operators used.

In binary coded GAs crossover is usually done by taking two individuals and creating offspring where different segments of its binary string is taken from different parents. There are many variations on how to do this, but basically each bit in the resulting string is inherited from either one parent or the other. The mutation operator takes a solution and flips one or more random bits in the binary string.

Real-valued GAs have also been created, using operators that operate value-by-value instead bit-by-bit. These operators change values with operations like randomly weighted averaging, interpolation or addition of some kind of random values.

There are many different schemes used for parent selection in GAs. During parent selection fitness proportional, rank-based and tournament selection are the most common. In fitness proportional selection each individual has a probability of selection proportional to its fitness value. In rank-based selection the probability of selection is proportional to the rank of the individual, i.e. individuals are sorted by fitness and assigned a probability based on their place on the list. In tournament selection two or

more individuals are picked at random and the most fit is selected. Survival selection is usually done by truncation, that is, the population size is set to some constant $\mu$, and the $\mu$ fittest individuals always survive.

### 3.2.4 Other Evolutionary Algorithms

In addition to GAs there are three main classes of evolutionary algorithms: evolution strategies (ES), evolutionary programming (EP) and genetic programming (GP).

Evolution strategies is similar to GAs on many points, but concerns itself mainly with continuous parameter optimization. Central to ES is the addition of strategy parameters to the chromosomes. These are parameters that do not affect the phenotype directly, but rather control how large changes are made to the solution parameters during mutation. Usually, parameters are mutated by adding a normally distributed variable to them and the strategy parameters then set the deviation used for those distributions. Much of ES then revolves around getting the strategy parameters to work as efficiently as possible. [11]

Parents are usually selected completely at random, while survival is determined by the fitness value alone: the new generation is made much larger than the old one and then the least fit are removed so that the new generation is the same size as the old.

Evolutionary programming, on the other hand, generates exactly one offspring, from mutation alone, from each individual in the previous generation. Survival is then determined by several rounds of pairwise tournaments between any two parent of offspring individuals. Each individual takes part in some fixed number of tournaments, and then the half of the individuals that have the most wins survive. [11]

Other than the difference on selection methods and the lack of crossover EP is very similar to ES - they both usually use a real-valued vector representation and a strategy variable-based normally distributed mutation operator.

All the evolutionary algorithms mentioned so far represent their solutions as some sort of fixed or variable length string of parameters, which is then used by a pre-defined formula or algorithm to give a solution to the problem. However, in some cases there might be so little known about the problem that no such algorithm is known, or there might be several candidate algorithms.

Genetic programming, introduced in [30], is an EA that represents solutions as computer programs. The claim is that that most optimization problems can be reformulated as a problem of finding an optimal program for solving a certain task, and that such a program can be found efficiently by EA methods.

The programs are represented in a tree structure where each leaf node is a constant and each other node represents some operation or conditional. The child nodes act as operands for the parent nodes. Here, the crossover operator usually takes two parents and exchanges random subtrees of the

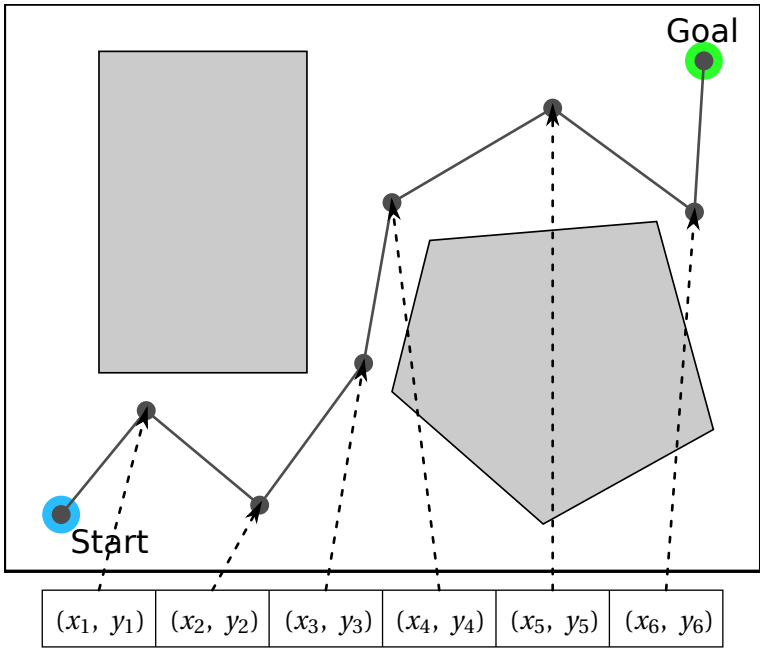



**Figure 3.8:** A Path Represented by a Sequence of Points. The chromosome that produced the path is shown, with arrows to the corresponding points on the path.

solutions, creating two offspring, while the mutation operator takes a leaf node or subtree and replaces it with a new random leaf or subtree.

### 3.2.5 Evolutionary Path Planning

Evolutionary algorithms have been successfully applied to the path planning problem as a replacement to classical path planning algorithms in many environments. As will be seen below, the real valued, variable-length genetic algorithm is a popular choice in path planning, but there exist other variant, such as the genetic programming algorithm in [19].

Several representation schemes have been used in evolutionary path planning. In [2] paths are represented as iterative displacements along a grid. Several spline representations are explored in [1], and [25] uses cubic spiral segments. In [23] the path is represented as piecewise linear curves using recursively split and displaced line segments, coded as a tree structure.

The most intuitive representation is perhaps a sequence of points. The path is then formed by using these points as milestones that are to be followed in order. One algorithm using this representation that have been studied extensively is the evolutionary planner / navigator (EP/N). It was first presented in [34] and extended in [52], [12] and others. In order to evolve efficiently the EP/N uses several specialized mutation operators that smooth out corners or repair infeasible solutions by stretching them around obstacles they intersect.

The easiest choice of initial population in algorithms like EP/N is to generate a set of random paths, feasible or unfeasible. This leads to a long initial planning time as it may take a while for the algorithm to evolve good solutions in complex environments. [39] suggests using probabilistic roadmaps to generate a set of feasible initial solutions instead. In this way the initial population will contain a random sample of the feasible solution space, with more individuals distributed near fit parts of the solution space.

## 3.3 Multi-Objective Optimization

When one wants to find problem solutions that are optimal with relation to not only one, but several objectives one is presented with a multi-objective optimization problem (MOOP). A good introduction to the topic, as seen from a EA perspective, can be found in [7] and [6]. In mathematical terms, instead of a scalar objective function $f(\mathbf{x})$ we have a $m$-dimensional vector objective function

$$F(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) & \dots & f_{m-1}(\boldsymbol{x}) & f_m(\boldsymbol{x}) \end{bmatrix}^T$$

comprised of $m$ scalar objective functions. In contrast to single-objective optimization problems where there is usually sufficient to consider only one optimal solution, there may be any number of different solutions in an MOOP that are better according to one objective function and worse according to another in relation to each other and usually there won't be a single solution that is best in every way.

A common example of a MOOP is that of cost versus quality. In a store we may choose between four models of computers. Let's imagine that we can choose between five models, with performance and price as plotted in 3.9. Model A is more powerful than model B, which again is more powerful than model C and E, which finally is more powerful than the severely outdated model D. Based only on how powerful the models are, the obvious choice would be model A. But, it so happens that the models range from expensive to cheap in the same order, except for model E which is both slower and more expensive than model C. Then we would have to make a careful consideration of how much computing power we can afford.

### 3.3.1 Reduction to a Single Objective

If we had set up a strict budget before going to to the computer store, then our choice would be simple - just buy the most powerful computer below a certain price. Setting up a worst-acceptable value for each objective function except one and the optimizing that single objective within those limits is called the $\epsilon$-constraint method.

When there is no such budget one might think up some measure of benefit - how much value computing power has relative to the money spent. Based on those priorities one can set up a numerical weight for each factor, multiply the cheapness and the performance with their respective weights

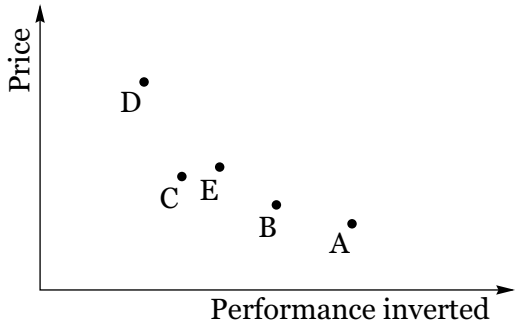



**Figure 3.9:** A price-performance problem. Notice that the performance axis is inverted, so that both axis are to be minimized.

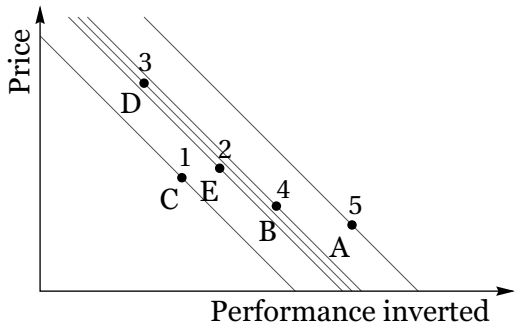



**Figure 3.10:** Weighted price-performance sum. For these particular weights solution E is ranked as the next-best solution.

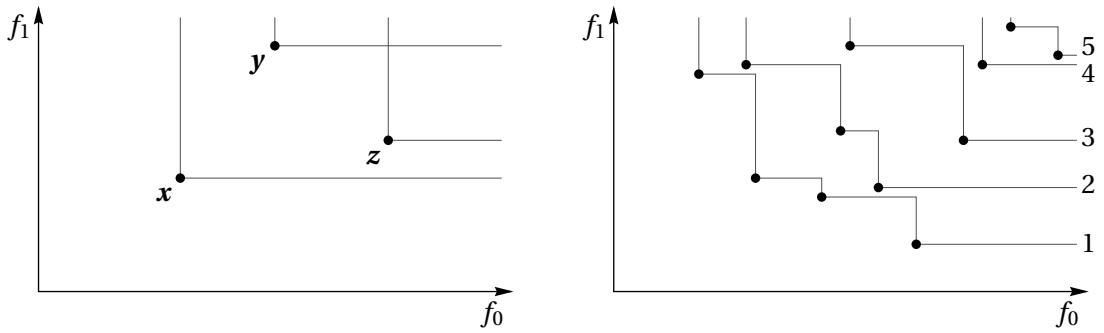
and sum them up. This gives a single number signifying how much value each computer has and we can choose the one that is best.

This is called the weighted sum approach, and an application of it to the computer purchase example above is given in Figure 3.10. A number of other methods also exist aiming to reduce multiple objectives into one so that we can find a single best solution using a conventional single-objective optimizing algorithm. A overview of the most common ones can be found for example in chapter 3 of [7].

### 3.3.2 Pareto-Optimality

The reduction methods mentioned above will find a single best computer for us, but ideally we would like to know what the selection is before planning a budget or deciding our preferences weight-wise. Furthermore, we want to avoid taking model E, which is both more expensive and slower than model C into consideration. With the weighted sum approach model E might seem a better choice than either of models A,B or D. Worse, there might not be any pair of weights that make model B the best solution at all.

To make an informed choice among a large number of trade-offs we




**(a)** Dominance. Here, $\boldsymbol{x}$ is not dominated by anything and dominates both $\boldsymbol{y}$ and $\boldsymbol{z}$, while neither $\boldsymbol{y}$ nor $\boldsymbol{z}$ dominates any other point. If $\boldsymbol{x}$ is removed, both $\boldsymbol{y}$ and $\boldsymbol{z}$ will be non-dominated.

**(b)** Non-dominated Sorting: Non-dominated fronts are found iteratively by finding the non-dominated front of the current set of points, removing that front from the set and repeating, each time assigning a higher rank to the front removed.

**Figure 3.11:** Dominance and non-dominated sorting.

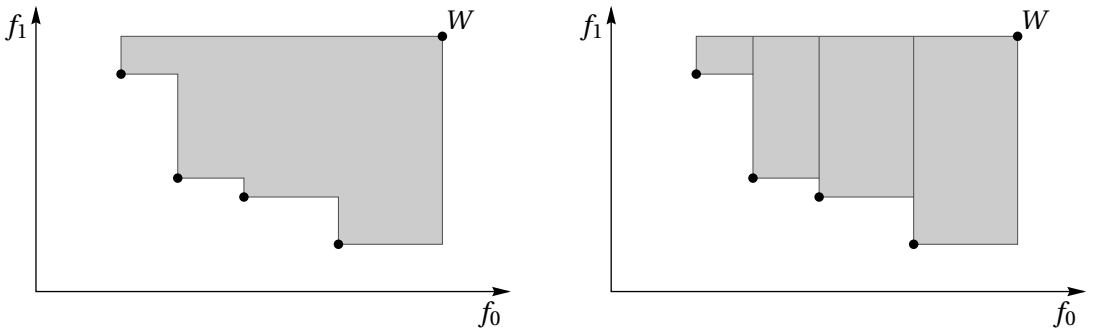
need the concepts of domination and Pareto-optimality. A solution $\boldsymbol{x}$ is said to dominate a solution $\boldsymbol{y}$, written $\boldsymbol{x} \preceq \boldsymbol{y}$, if and only if it is as good as or better than $\boldsymbol{y}$ for all objective functions and strictly better for at least one objective function. When all objective functions are to be minimized this can be expressed as:

$$\boldsymbol{x} \preceq \boldsymbol{y} \equiv \forall i \left(\mathrm{f}_i(\boldsymbol{x}) \leq \mathrm{f}_i(\boldsymbol{y})\right) \wedge \exists j \left(\mathrm{f}_j(\boldsymbol{x}) < \mathrm{f}_j(\boldsymbol{y})\right)$$

A simple example of dominance i shown in Figure 3.11a. A solution $\boldsymbol{x}$ is said to be Pareto-optimal if and only if there is no other solution in the entire solution space $\mathcal{S}$ that dominates it. Notice that if a solution $\boldsymbol{x}$ is better in one way and solution $\boldsymbol{y}$ in another, then neither solution dominates the other, so there can be any number of Pareto-optimal solutions. The set of all Pareto-optimal solutions is called the Pareto set or the Pareto front (since the Pareto set tends to form a surface in objective space).

### 3.3.3 Non-dominated Sets and Sorting

Using the definition of dominance, we can device several ways of ranking the solutions in a given set. One simple way would be to count the number of solutions that dominate each given solution and the assign lower ranks to solutions dominated by fewer solutions. A slightly more involved approach would be to iteratively "peel" off layers of non-dominated solutions from the set, assigning each layer equal rank. This method is called non-dominated sorting. An illustration is given in Figure 3.11b. In each iteration the non-dominated front of the set is removed. The non-dominated front is the subset of solutions that are not dominated by any solutions in the current




**(a)** The hypervolume. One can see that it measures how much of $[0, W_0] \times [0, W_1]$ is dominated by the set of points.

**(b)** How a 2D hypervolume can be divided for easy calculation.

**Figure 3.12:** An 2D Hypervolume.

set. The set of solutions removed in the first iteration is usually given rank 1, the second rank 2, and so on.

### 3.3.4 The Hypervolume Indicator

Summarising the performance of multi-objective optimization algorithms (for example for comparison with other algorithms) can be difficult, especially when using stochastic algorithms like EAs that need to be run multiple times and treated statistically. Interpreting a set of multi-dimensional non-dominated fronts in comparison to another directly will usually be too complex. Therefore, a number of methods has been developed to interpret and compare such results, some of which are summarized in chapter 8 of [7]. If one knows the ideal solution set beforehand one can create some sort of measure of distance to that, but in complex problems this is often not available.

The hypervolume indicator is one of the common metrics for multi-objective optimization performance. It measures the volume of the union of the hyper-rectangles created with a non-dominated solution and some reference point W as opposite corners, as shown in Figure 3.12. This gives an indication of how well a non-dominated front covers some relevant part of objective space.

The hypervolume is not a perfect measure of the performance of the quality of a solution set (for example, it is not independent of scaling of the different objective functions), but it is often used, as it can produce a scalar value for each run of an algorithm which then can be used for analysis later.

### 3.3.5 Multi-Objective Evolutionary Algorithms

For MOOPs there is one big advantage with using an evolutionary algorithm: it inherently optimizes a large number of solutions. Most other kinds of algorithms work on finding only one solution at a time and can

only optimize for a single objective function, forcing the use of some sort of reduction method.

Sometimes optimization algorithms with specialized reduction methods can work out pretty well, as in [47]. But finding a good reduction method might be difficult for many problems, and as always with reduction methods you have to know the priority of the different objectives beforehand. And while specifying priorities beforehand may not always be a disadvantage - a reduction method that optimizes multiple objectives in turn according to priority is utilized in [33], for example - it can be a disadvantage in many situations.

The main difficulty with how to adapt evolutionary algorithms to multi-objective problems is how to select individuals for reproduction and to keep good diversity. One simple solution to this, as presented in [44] (VEGA), is to choose one of the different objective functions for different sub-populations, selecting a number of solutions from each sub-population for crossover, and then let all the selected solutions cross over at random.

However, even with a mutation operator added this can lead to "niching" - that solutions cluster up at the optimums for the different objective functions. As a sort of generalization on this one could use a weighted sum approach, but with different weights for different individuals. In [18] (WBGA) this is done with weights contained in each chromosome and in [38] (RWGA) random weights are used for each fitness evaluation.

Another way of solving this problem is to rank solutions by dominance, either by counting how many solutions that dominate each solution [13] (MOGA) or by sorting the solutions into non-dominated sets [48] (NSGA). The first non-dominated set contains all the solutions that are not dominated by any other solution in the entire population. The next non-dominated set contains all solutions that are only dominated by solutions in the first non-dominated set, the third set contains the solutions only dominated by solutions int the first and second sets and so on. The disadvantage to this is that you need to keep track of who dominates who in the entire population, which has high time complexity.

Several more advanced algorithms also exist. Two of these, SPEA and NSGA-II, will be covered in some detail below.

### 3.3.6 SPEA and Hierarchical Clustering

The strength Pareto evolutionary algorithm, or SPEA, was proposed in [53]. It keeps an elite population as an external record containing a selection of the non-dominated solutions so far alongside a non-elitist population. For each generation these non-dominated solutions are ranked according to how many of the current population they dominate. This measure is termed the strength of those solutions. Each individual in the population is then assigned a fitness based on the sum of strengths of the non-dominated solutions that dominate them.

The method used in SPEA to reduce the size of the elite population is called hierarchical clustering and is of interest in this text. It solves the following problem: We want to reduce the number of solutions to some

**Algorithm 2** Average Linkage Hierarchical Clustering

**function** `reduce-size`$(P)$
  $C \leftarrow \{ \{ p \} \mid p \in P \}$
  **while** $|C| > \overline{N}$ **do**
    $c_a, c_b \leftarrow \underset{c_i, c_j}{\operatorname{argmin}}(\texttt{distance}(\texttt{centroid}(c_i), \texttt{centroid}(c_j)))$
    $C \leftarrow C \setminus \{ c_b \}$
    $c_a \leftarrow c_a \cup c_b$
  **end while**
  **return** $\{ \texttt{centroid}(c) \mid c \in C \}$
**end function**

maximum number of elite solutions, $\overline{N}$, while maintaining a good spread of solutions. The algorithm solves this by iteratively merging clusters of solutions: Initially, each solution is assigned its own cluster. Then, the two clusters with the shortest distance between them are merged. This is done iteratively until only $\overline{N}$ clusters remain (see Algorithm 2).

The centroid of each cluster - the point with shortest average distance to all points in the cluster - is used to calculate the distances. When the number of clusters have been reduced, the elite population is made out of those centroids. In SPEA it is proposed to use objective space distance for the clustering, but in theory any distance measure can be used by the clustering algorithm.

### 3.3.7 Non-dominated Sorting Genetic Algorithm-II

Ranking solutions by which non-dominance set it is in helps keep variation near the Pareto front since solutions that are close to being Pareto-optimal are more likely to survive, no matter where they are along the front. If the algorithm also makes sure solutions are more likely to survive when they are far from other solutions in objective space, then a good and varied selection of solutions along the Pareto front can be achieved.

In [8] an elitist non-dominated sorting GA is suggested. The algorithm was termed NSGA-II to reflect that it was based on experiences from NSGA. NSGA-II selects parents by binary tournaments and survivors by truncation. The same comparison operator is used both for determining the winner in binary tournaments and for sorting the population for truncation.

The comparison operator works in the following manner: All individuals have a rank given by non-dominated sorting. If the two individuals compared have different rank, the one with lowest rank is best. If their ranks are equal then the individual with the highest crowding distance is best.

Crowding distance is a measure of objective space distance between individuals on the same non-dominated front. It is calculated as shown in Algorithm 3: The individuals are sorted for each dimension of objective space. The crowding distance of an individual is the sum of the distances between the two neighboring solutions in each dimension. A solution that

**Algorithm 3** Crowding Distance Calculation

```
function assign-crowding-distance(S)
    for all s ∈ S do
        crowding-dist(s) ← 0
    end for
    for m = 1 → M do
        sort-by-component(S, m)
        crowding-dist(S_1) ← ∞
        crowding-dist(S_N) ← ∞
        for i = 2 → N − 1 do
            crowding-dist(S_i) ← crowding-dist(S_i) + (S_{i−1,m} − S_{i+1,m})
        end for
    end for
end function
```

is best or worst in a dimension is assigned infinite crowing distance.

### 3.3.8 Multi-Objective Graph Traversal

In order to solve multi-objective path planning problems using classical graph-based algorithms like visibility graphs or probabilistic roadmaps one must either use some function to reduce to a single objective in order to use the usual graph traversal algorithms the way they are, as is done in for example [47], or one can adapt some traversal algorithm to multi-objective problems, for example using concept domination and non-dominated sets. An example of this can be found in [15].

A* - perhaps the most common graph traversal algorithm - has been proved in [49] to be adaptable to solve multi-objective problems, while keeping the same theoretical properties as single-objective A*. Basically, for each node in the graph one records a front of non-dominated partial solutions reaching that node instead just one best partial solution. Then, when the search can no longer improve the front of any node, the loop ends and the non-dominated set of solutions at the goal node(s) is returned.

## 3.4 Neutrality

Traditionally chromosome encoding is chosen so that any change to the genotype would result in some change to the phenotype and the fitness. In some optimization problems it might happen that “barriers” are formed between local optima: As the population settles into one or more local optima it gets increasingly difficult to find new, better local optima.

This can happen when the sequence of evolutionary operators needed to get to a better local optima would be several generations long, and some of the generations are so unfit that they cannot survive. Then the other local optima can no longer be reached in that run of the algorithm.

Some have suggested that to remedy this, we can again look to

evolutionary biology, this time to what is called neutral theory. In the following sections neutral theory and neutrality will be presented, first by as brief overview of its biological origins and then in the context of evolutionary algorithms. A more detailed summary can be found in [16].

### 3.4.1 Neutral Theory in Biology

In evolutionary biology it is common that the genetic material contains a high degree of redundancy, with multiple chromosome encodings lead to the same result. When two different genotypes lead to the same fitness it can be possible for a mutation to lead from one to the other. A mutation that does this is called a neutral mutation.

Since it was suggested in [29] that neutral mutations play a big role in evolutionary changes at the molecular level, much work has been done studying the effects of neutral mutations. These studies seem to indicate that the neutrality helps the genetic material stay robust at high rates of mutation while adapting efficiently in both flat landscapes with difficult-to-find optima, and rugged landscapes with many scattered local optima.

### 3.4.2 Neutrality in Evolutionary Algorithms

Inspired by neutral theory, much work has been done on exploring the effects of neutrality in evolutionary algorithms. The term neutral networks was introduced in [21] to describe sets of equally-fit points in the search space that can be traversed through neutral mutations.

Several ways of artificially introduce neutrality with the use of genotype-phenotype mappings were described in [46] and [45]. These mappings were later investigated further in [9, 10] and [42].

The latter claims that in order to have an effect beyond that of a random search, any redundancy should be synonymously redundant. That is, genotypes that lead to the same phenotype should be close to each other in the sense that they belong to the same neutral network.

It was pointed out in [50] that self-adapting methods such as those used in evolution strategies add neutrality to the chromosome, since strategy parameters are not used in fitness evaluation, but rather affect the evolutionary process indirectly.

The exact effects and usefulness of neutrality in EAs are still undetermined. The summary in [16] seems to indicate that neutrality in itself is no guarantee for improved performance and that the right kind of neutrality is needed to improve upon a non-neutral EA or even a random search . However, as to exactly what makes the right kind of neutrality is the opinion is more divided.

# Chapter 4

# The Evolving Roadmaps Algorithm

In this chapter an evolutionary path planning algorithm is described which uses a neutral, roadmap-like representation. The motivation for designing such a method was to explore the possibilities for an evolutionary algorithm with neutrality in path planning problems. It was first proposed and tested for single-objective path-planning in [43]. The single-objective approach described below is essentially identical to the algorithm presented in that paper.

## 4.1 Single-Objective Approach

The following sections will detail the evolving roadmaps (ERM) algorithm. First, the reasons for choosing this approach are outlined. Then chromosome representation and evaluation is explained. The evolutionary operators used are described, and finally the method for population initialization and the rest of the evolutionary process will be described.

### 4.1.1 Choice of Methods

In order to have neutrality in an evolutionary path planner one would have to design a chromosome encoding which would make it probable that two different chromosomes result in the same fitness. While two different paths may have the same fitness, the simples way to guarantee the presence of neutral networks is to make sure that equal paths can be generated by chromosomes few mutations apart. One way of doing this is by using some representation where not every gene in the chromosome affects the resulting path. Ideally, the non-contributing genes would only come into effect if they are mutated in a manner that enhances the phenotype.

Looking to classical path planning for inspiration, it is clear that the graphs used in PRMs and similar algorithms has this property. All nodes in the graph that are not part of the best path through it can be move quite liberally around or even duplicated, added or deleted without affecting the result - that is, unless they enable a better path to be found.

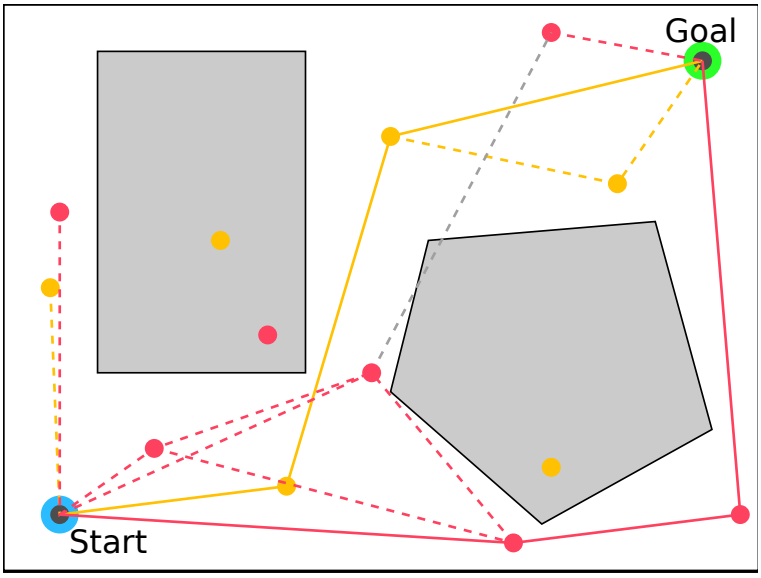



**Figure 4.1:** Two different roadmap chromosomes. The two chromosomes are shown in different hues. Notice that the grey edge can become feasible with only a slight mutation in the red chromosome. Then the best path in the graph will "tunnel" to the other side of the pentagon.

Like most evolutionary path planners, PRMs handles high-dimensional spaces well and only require that pairs of points can be tested for connectivity. However, since the nodes are generated more or less blindly the results are rarely close to optimal. The method presented here can be seen as a way of iteratively improving the nodes in "PRMs" using an evolutionary process.

### 4.1.2 Representation and Evaluation Function

The chromosome encoding chosen to code the graphs is simply a string of points in the environment or C-space. Instead of connecting the points by sequence, they represent the set of nodes in a graph. This is illustrated in Figure 4.1. These nodes are connected by some local path planner. Each node is connected to all nodes that can be reached from it by the local algorithm. Like in classical graph-based algorithms, any traversal algorithm can then be used to find the best path in the graph.

The path produced by the graph traversal will then be the phenotype of the individual, and can be given a fitness evaluation like any other path. However, the graph traversal will most likely have to calculate the fitness of the path in order to find it, and the translation from genotype to phenotype becomes heavily entangled with fitness evaluation.

### 4.1.3 Evolutionary Operators

In essence, the chromosomes are sets of points, so the evolutionary operators should be adapted to that. Crossover is implemented so a pair

**Algorithm 4** Single-objective phenotype translation and fitness evaluation

```
function assign-fitness(I)
    g ← empty-graph()
    nodes(g) ← { start, goal }
    if local-planner-find-path(start, goal) then
        edges(g) ← edges(g) ∪ { { start, goal } }
    end if
    for all p ∈ chromosome(I) do
        if ¬obstructed(p) then
            for all q ∈ nodes(g) do
                if local-planner-find-path(p, q) then
                    edges(g) ← edges(g) ∪ { { p, q } }
                end if
            end for
            nodes(g) ← nodes(g) ∪ { p }
        end if
    end for
    path(I) ← graph-traversal(g)
    fitness(I) ← evaluate(path(I))
end function
```

of parents generate a pair of offspring and each point in each parent chromosome is transferred to one and only one of the offspring. This is done by taking each point of both parents' chomosomes and copying them one at a time to one of the offspring, selected at random.

The chromosome size of each offspring will then be binomially distributed, which ensures that chromosome sizes get averaged out a bit and do not fluctuate too much. Notice that if both parents are actually the same individual or if the parents have common ancestry some points are likely to appear more than once in the offspring.

This is not considered a problem here, as long as the graph generated later ensures that duplicate points do not create duplicate entries in the roadmap constructed during evaluation (otherwise some graph traversal algorithms would return duplicate solutions). That is because point duplication can be beneficial later if one of the points mutate.

Three mutation algorithms are used in the algorithm: Change, insert and delete. Change is closest to the standard GA mutation operator in both operation and intent: to randomly explore close solutions. For each point in the chromosome it makes a normally distributed addition with a certain probability $P_{\mathrm{m}}$. The normal distribution has a constant standard deviation.

The insert and delete mutations are meant to keep variation in the population and restrain chromosome size. The insert mutation inserts a random point, using a uniform distribution over the entire environment both obstructed and unobstructed.

The delete mutation takes each point in the chromosome and erases it with a probability $P_{\mathrm{d}}$. The insert mutation provides variation through

adding a fixed number of completely random points, while the delete mutation on average deletes $n \cdot P_d$ random points from each chromosome, where $n$ is the size of the chromosome.

The insert mutation is applied to each offspring $N_i$ times and the delete mutation once. Together they are likely to spread the points in the chromosome a little more evenly through the environment. And since the delete mutation is an order "stronger" than the insert mutation, they put some pressure on the evolutionary process to stabilize chromosome size, which has a tendency to grow over time in EAs with variable-size chromosomes (if only the offspring survive in each iteration and no other factor had affected chromosome size, it will average out to $N_i \cdot (1 - P_d) / P_d$).

### 4.1.4 Initial Population and Evolutionary Process

The population is initialized to size $N$, and the same amount of offspring are generated in each new generation (rounded up to the nearest even number since the crossover operation always generate two individuals). Parents are selected by binary tournament selection. After crossover each offspring is mutated once by the change operator, a constant number of times by the insert operator and then once by the delete operator. Binary tournaments are then used to select $N$ different individuals for survival.

Naively, since the chromosomes are just collections of points, one could initialize the population with a random amount of random points in each chromosome. As mentioned above, each chromosome acts as a roadmap so that would in effect initialize the population with simple PRMs.

However, without knowing how many points are needed on average in a PRM to find a path in the give environment one could end up with too few feasible solutions. Therefore it is best to initialize each chromosome in some way that at least guarantees a high probability of giving feasible solutions in the graph traversal algorithm used in the fitness evaluation.

Actually, almost all the ways of constructing graphs mentioned in classical path planning literature should be applicable for initializing the population, as long as it generates graphs that are feasible most of the time. As the important thing during initialization is to find as varied a distribution of initial solutions as possible, it might even be beneficial to initialize different individuals using different graph construction techniques.

## 4.2 Multi-Objective Approach

In this section the single-objective algorithm presented above is adapted to multi-objective problems. Basically, the steps in the evolutionary process is generalized to vector fitness values using NSGA-II as a reference framework, but since the algorithm uses a graph traversal algorithm during fitness evaluation, and graph traversal may result in any number of non-dominated solutions it was decided to adapt the methods used in NSGA-II to multiple solutions per individual.

### 4.2.1 Adaption to Multiple Objectives

In the single-objective ERM the graph traversal algorithm is assumed to only return one solution. This is a relatively reasonable assumption - two or more equally good solutions could appear, but since their fitness will be exactly the same choosing just one of them should not affect the survival chances of the individual anyway.

In the multi-objective case however, a graph traversal algorithm could return any number of solutions representing different trade-offs present within the graph. Indeed this is desirable, as it can not be known at this stage which these are good or bad among the population as a whole.

One could use a weighted sum approach like WBGA and use single-objective graph-traversal based on weights decided by the chromosome. This would, admittedly remove the single genotype - multiple phenotype problem and retain the neutrality property well.

However, the idea of having multiple phenotypes generated from a single genotype might not be as bad an idea as one might think. To make an analogy, consider the case of identical twins. While they will usually grow up to look more or less the same, one might argue that in order to maximize the chances of procreation it might be wise for the twins to learn different skill-sets and proficiencies.

If they always make the same choices in life chances will be higher that none of them will find a mate, since there is (in this analogy, at least) no way for them to know if those choices make them fit relative to the rest of the population. Based on this thought the algorithm proposed here lets each chromosome produce a number of phenotypes. These solutions sometimes act as a single entity and sometimes act as separate individuals.

In order to stop large families from overpowering smaller families a limit is set to the number of solutions that can be generated per chromosome. This is done by taking the complete set of solutions returned by the graph-traversal algorithm and then reducing to the maximum number of solutions using the hierarchical clustering algorithm used in SPEA.

### 4.2.2 Adapting the Evolutionary Process to NSGA-II

Having several solutions per chromosome leads to some challenges in the evolutionary process. The perhaps simplest way to solve this would be to just let each solution be a completely separate individual. This would increase the selection pressure a lot, and potentially give a disadvantage to chromosomes that result in only one or two very good solutions.

Instead, a method was developed that threats each group of solutions as one individual (or maybe sibling group would be a better term) during parent and survival selection and as separate individuals during the non-dominated sorting and crowding distance calculation phase.

For parent selection a binary tournament style operator was used: A given number of rounds are done between two sibling groups, each round being a binary tournament between a random solution in each of the two

groups. The group with the most wins is selected as a parent. (In the case of a draw one of the groups are selected at random.)

For survival selection each individual of each group is put individually in a large solution set and sorted by non-domination and crowding distance as usual in NSGA-II. However, when the first individual of a group is given a rank, that rank is assigned to the entire group, and the rest of the individuals of the group are ignored. In this way, the survival of a group is only tied to the best solution in it globally.

# Chapter 5

# Benchmark

This chapter will present the method used to evaluate the performance of the proposed algorithm. First, a reference algorithm used for comparison is described. Then, the objective functions used in the tests are explained. Finally, the environments and parameters used are presented.

## 5.1 Reference Algorithm

In order to examine the effects of the roadmap chromosome encoding in the ERM, an algorithm that is as similar as possible in other aspects was constructed. NSGA-II is used for the evolutionary process, with initialization, chromosome encoding and evolutionary operators based on [39].

The initial population is generated using graph construction and traversal, so that the populations of the two algorithms can be initialized in the exact same way. The evolutionary operators used are single point crossover and the three mutation operators change, shortcut and smoothen.

The change mutation moves a random point in the chromosome a normally distributed distance in some direction. The shortcut mutation picks two random points in the chromosome and removes all points between them, creating a shortcut. The smoothen mutation takes a random point in the chromosome and replaces it with two points, uniformly randomly distributed the each of the adjacent edges.

All three mutation operators will check if the affected areas of the chromosome is feasible or not after the mutation. The operation will be retried up to 7 times if the feasibility tests fail. On the 8th try the mutation will be applied regardless of feasibility. After being generated by the crossover operator, each offspring is either modified by one of operators or left as it is with probabilities $P_{\text{ch}}$ for the change mutation, $P_{\text{sc}}$ for shortcut, $P_{\text{sm}}$ for smoothen and $(1 - P_{\text{ch}} - P_{\text{sc}} - P_{sm})$ for no change.

## 5.2 Objective Functions

To test the multi-objective algorithms two different objective functions will be defined. The first is a normalized version of the standard curve length function used to solve shortest distance problems. The second is derived from a simple model of the work that the subject must exert to follow the path at a fixed speed.

### 5.2.1 Distance

As a measure of distance a normalized curve length is used. This is done to lessen effect the size of the environments has on the algorithm and the results. Since the solutions are piecewise linear paths the curve length can be expressed as the sum of the Euclidean norms of each linear piece:

$$\|\gamma\| = \sum_{i=1}^{n-1} \|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|$$

However, this metric will be dependent on the scale of the environment, which is not of importance in a benchmarking context. Therefore, the curve distance is divided by the Euclidean distance from start to goal, forming a normalized distance function:

$$\mathrm{f}_1(\gamma) = \frac{1}{\|\boldsymbol{p}_n - \boldsymbol{p}_1\|} \sum_{i=1}^{n-1} \|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|$$

### 5.2.2 Estimated Work

As a secondary objective, an estimation of the work that needs to be exerted by a subject to follow a given path has been used. It is assumed that the subject proceeds at a constant speed. The estimation used is based on the following expression for work:

$$W = \int_{t_0}^{t_1} \boldsymbol{F}\boldsymbol{v}\mathrm{d}t + \int_{t_0}^{t_1} \tau\omega\mathrm{d}t$$

Here, the first term is the translational work and the second term is the rotational work. The speed $v$ is constant, and it is assumed that the force translational force parallel with the velocity is also constant also constant. In that case the translational work is proportional with $v \cdot (t_1 - t_0) = v \cdot \Delta t$ which is equal to the curve length of the path, $\|\gamma\|$.

In order to approximate the second term we will assume that the subject will make a small deviation from the piecewise linear path in order to keep constant speed at the corners. The subject is assumed to always make a turn of constant curvature at each corner that deviates at most $\varepsilon$ from the path. Further, it is assumed that the subject requires no torque to be applied to keep a constant rotational speed (i.e. air resistance, friction and so on does not affect rotation). Then torque is only applied in a short moment before and after the turn. The torque for each corner can then be approximated

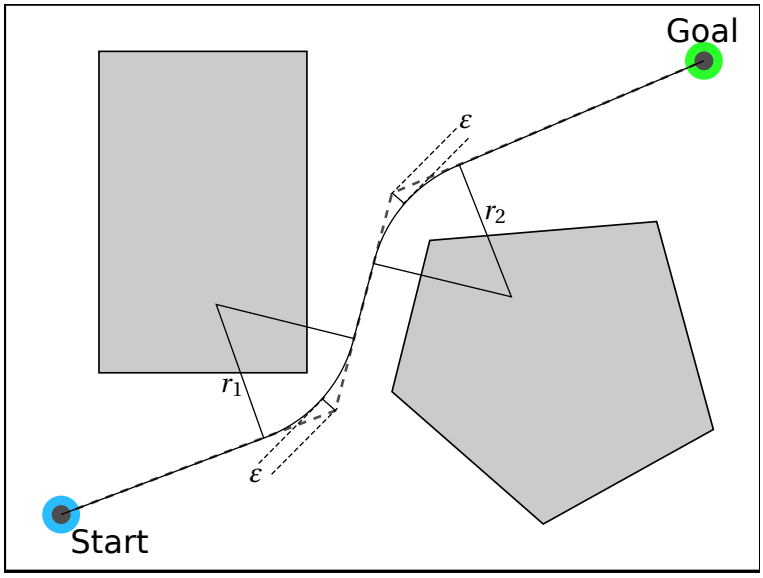



**Figure 5.1:** An example of the curve used to estimate work. The original curve $\gamma$ produced by the path planner is in gray dashes. The "rounded" curve $\gamma'$ is in black, together with some of the geometry used to construct it.

in terms of the times $t_a$ and $t_b$ of entering and leaving the curve and some short duration $h$:

$$\tau(t) = \begin{cases} \omega_c / h & t \in [t_a, t_a + h) \\ -\omega_c / h & t \in [t_b - h, t_b) \\ 0 & \text{else} \end{cases}$$

The rotational speed will then be

$$\omega(t) = \int_{t_a}^{t} I\tau(t')\,\mathrm{d}t = \begin{cases} I\frac{w_c}{h}(t - t_a) & t \in [t_a, t_a + h) \\ I w_c & t \in [t_a + h, t_b - h) \\ I\frac{w_c}{h}(t_b - t) & t \in [t_b - h, t_b) \\ 0 & \text{else} \end{cases}$$

Which in turn lets us calculate the rotational work expended during that turn:

$$\begin{aligned} \int_{t_a}^{t_b} \tau(t)\omega(t)\,\mathrm{d}t &= \int_{t_a}^{t_a+h} I\frac{\omega_c^2}{h^2}(t - t_a)\,\mathrm{d}t + \int_{t_b+h}^{t_b} I\frac{\omega_c^2}{h^2}(t_b - t)\,\mathrm{d}t \\ &= I\frac{\omega_c^2}{h^2}\left(\left[\frac{1}{2}t^2\right]_0^h + \left[\frac{1}{2}t^2\right]_0^h\right) = I\frac{\omega_c^2}{h^2}h^2 \\ &= I\omega_c^2 \end{aligned}$$

All that is left now is to find an expression for the constant rotational speed, $\omega_c$. For small $h$ it can be approximated by a circular arc tangential to

both of the line segments that meet at the corner. The rotational speed of a body rotating along with the tangent of a circle with radius $r$ while moving at a speed $v$ has the rotational speed

$$\omega = \frac{v}{r}$$

Thus we can approximate the rotational work with a value proportional to the square reciprocal of the turn radius, and the total work exerted can be approximated as

$$W = c_T \cdot \|\gamma\| + c_R \cdot \sum_{i=1}^{n-2} r_i^{-2}$$

$c_R / c_T$ then determines the relative importance of a short path versus a smooth path. If we set $c_T = \|\boldsymbol{p}_n - \boldsymbol{p}_1\|^{-1}$ then the first term equals the distance function $f_1(\gamma)$ defined earlier. After initial testing, $c_R / c_T$ was set to 4, giving the following expression for the second objective function:

$$f_2(\gamma) = f_1(\gamma) + \frac{4}{\|\boldsymbol{p}_n - \boldsymbol{p}_1\|} \sum_{i=1}^{n-2} r_i^{-2}$$

To find an expression for $r$ we need to apply the restriction given above that the "rounded" curve $\gamma'$ that the subject will actually follow does not deviate more than $\varepsilon$ from the piecewise linear curve $\gamma$ given by the path-planning algorithm. The maximal deviation around a corner will happen between the exact corner of $\gamma$ and the middle of the corresponding circle arc in $\gamma'$. With some straightforward trigonometry one can find that the radius is related to the deviation and the angle $\alpha$ between the adjacent line segments by

$$r = (r + \varepsilon) \sin\frac{\alpha}{2} \Rightarrow r = \varepsilon \frac{\sin\frac{\alpha}{2}}{1 - \sin\frac{\alpha}{2}}$$

However, this constraint sets no limits to how far into each line segment the circle arc starts, and may give values for $r$ that lead to unconstructable curves. To avoid this, a second criteria is used that will override the deviation criteria when it leads to values for $r$ such that the circle arc extends more than $\frac{d}{2}$ into each line segment, where $d$ is the length of the shortest segment. The expression for $r$ such that the arc extends exactly $\frac{d}{2}$ into each segment is

$$r = \frac{d}{2} \tan\frac{\alpha}{2}$$

In order for the second criteria to hold we can simply set r to be the minimum of these two expressions:

$$r = \min\left\{\varepsilon \frac{\sin\frac{\alpha}{2}}{1 - \sin\frac{\alpha}{2}}, \quad \frac{d}{2} \tan\frac{\alpha}{2}\right\}$$

## 5.3 Benchmark Environments

Four two-dimensional environments, shown in Figure 5.2, have been selected. The four maps are topologically similar to test environments used in the literature. The first two maps are identical to maps used in [43]. For future reference, the environments are defined in a simple text-based format and enclosed as Appendix A.

The first map (Figure 5.2a) has many smaller, convex obstacles scattered around the environment. This gives rise to many possible routes between the obstacles, and many of the short paths are not very straight.

The second map (Figure 5.2b) contains two large spiral structures. The start point is at the center of one of them and the goal at the other, so the path planner has to successfully optimize several turns while finding a way into the goal spiral.

The obstacles in the third map (Figure 5.2c) create a thick barrier between start and goal, with one narrow zigzagging passage between them and a wide open way around one of them. The narrow passage is designed so that it is much shorter than the way around, but requires three times as many turns.

Finally, the fourth map (Figure 5.2d) is a relatively simple, open environment, except for two "hooks" blocking the straight path between start and goal. This map should not pose any big problems for either algorithm tested, but is included here to test the efficiency of the algorithms in optimising simpler problems.

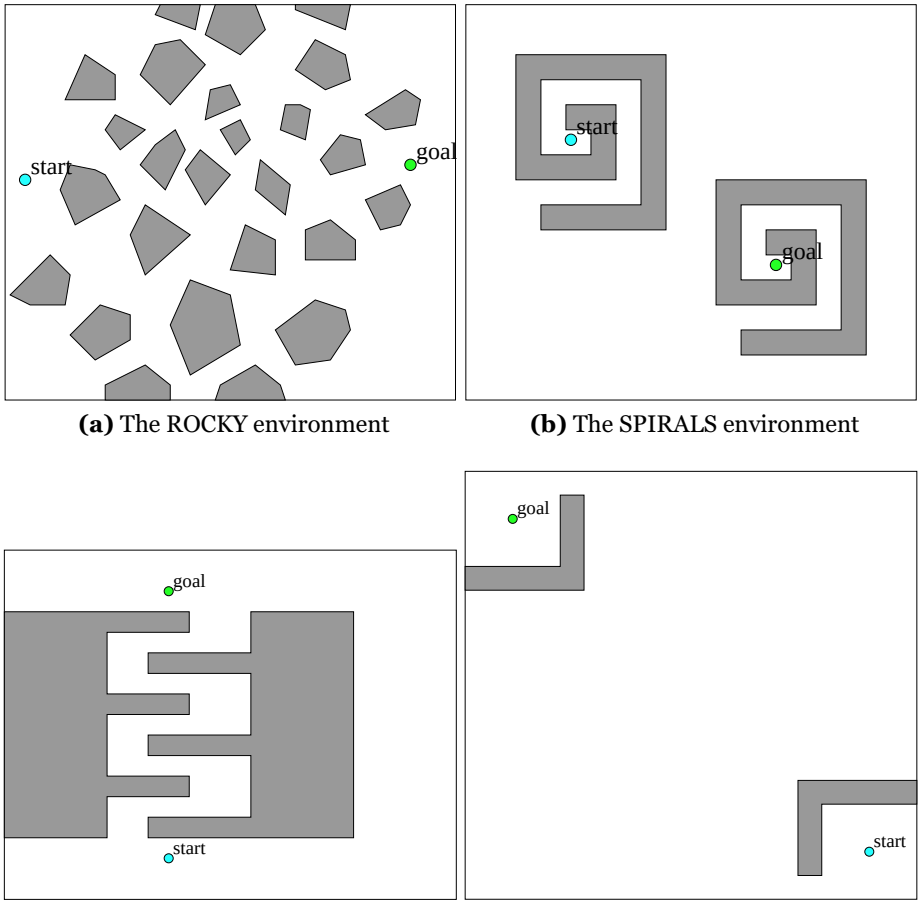
## 5.4 Benchmark Setup

The two algorithms that were compared are designed to be as similar as possible, except for chromosome encoding and differences in selection and evolutionary operators as a consequence of that. The parameters used for the two algorithms are summarized in Table 5.1. Since the encoding in ERM requires a graph traversal for each chromosome evaluation it is more complex computationally than the reference algorithm.

However, initial experiments indicated that in several of the environments ERM could produce solutions that match the reference algorithm in quality (as measured by the hypervolume indicator) at lower population sizes. To examine the relation between run time and solution quality both algorithms were tested at four population sizes in each environment. The population sizes used are shown in Table 5.1. Each of these 8 algorithm variants were run 200 times per map.

Additionally, this benchmark was run twice, once with one objective (shortest distance) and once with two objectives (shortest distance and estimated work). Notice that although only a single objective is used in the first run of the benchmark, the multi-objective algorithms are used in both runs.

For each of run of each of the algorithm variants the front of non-dominated solutions in the population after the final iteration is recorded




**(a)** The ROCKY environment

**(b)** The SPIRALS environment

**(c)** The DETOUR environment

**(d)** The STAR environment

**Figure 5.2:** The four environments tested in the benchmark. The ROCKY and SPIRALS environments are based on [52], while DETOUR and STAR are taken from [22].

| Algorithm | Reference algorithm | Evolving roadmaps |
|---|---|---|
| Population sizes | $N = 30,\ 50,\ 70,\ 90$ | $N = 5,\ 15,\ 25,\ 35$ |
| Offspring/iteration | $\lambda = N$ | $\lambda = N$ |
| Termination | $s_0 = 2$<br>$s_{\min} = 0.01$<br>$\alpha = 0.002$ | $s_0 = 2$<br>$s_{\min} = 0.01$<br>$\alpha = 0.02$ |
| Operators | $P_{\text{ch}} = 0.25$<br>$P_{\text{sc}} = 0.2$<br>$P_{\text{sm}} = 0.3$ | $P_{\text{m}} = 0.1$<br>$N_{\text{i}} = 2$<br>$P_{\text{d}} = 0.07$ |

**Table 5.1:** Parameters Used in the Tests

along with the iteration number and the total CPU time of the run. The hypervolume of the population after each iteration is also recorded. The tests were run on 8-core Intel i7 870 (2.93GHz) computers, using a single thread for each run and registering the CPU time used by that thread only.

### 5.4.1 Initialization and Evaluation Algorithms

For generality, no specific method of initialization, local planning or graph traversal is mentioned in the description of ERM in section 4.1. In these tests, both ERM and the reference algorithm uses a version of the visibility-based PRM described in section 3.1.7 for initialization.

This algorithm is much faster than an ordinary PRM while creating more varied solutions. As a stand-alone path planner the large variation in solutions is a drawback, but as a method of population initialization this is an advantage, since a diverse initial population is favorable for an evolutionary algorithm.

The local planner is implemented as a visibility test, simply checking whether the straight line between the two nodes is obstructed or not. The graph traversal algorithm used is the multi-objective A* algorithm described in section 3.3.8.

### 5.4.2 Termination Criteria

Termination is determined by exponentially-weighted moving average of the improvements to the hypervolume indicator each iteration. The average is given some initial value $s_0$ and then calculated as:

$$s_i = (1 - \alpha) s_{i-1} + \alpha \Delta \mathrm{HV}_i$$

Where $\Delta HV_i$ is the difference in hypervolume between this iteration and the previous one. Negative $\Delta \mathrm{HV}_i$ are truncated to zero. When $s_i$ reaches below a constant $s_{\min}$ the run is terminated. This ensures that the algorithm is given extra iterations when progress is made by running for at least $\ln(s_{\min}/s_i) / \ln(1-\alpha)$ additional iterations waiting for subsequent improvements after iteration $i$. The values used in the test is shown in Table 5.1. These values were set based on initial experiments that indicated that the reference algorithm showed significantly less improvement per iteration than ERM. The settings used force ERM to do at least 263 iterations, while the reference algorithm has to do ten times as much.

# Chapter 6

# Results

In this chapter the results of the benchmark described in the previous chapter will be presented. Results from the single-objective benchmark and the multi-objective one will be presented in turn, with a brief introduction, a summary of each environment and then a general summary of the tendencies observed.

Since each of the two algorithms has been tested with four different settings, each of which has been run 200 times, there are 1600 data points for each environment. The results have different characteristics for each of the four environments and can not easily be merged or averaged between them.

Furthermore, the results from each class do not vfit the normal distribution very well, and are sometimes even multi-modal. There is often correlation between run time and solution quality.

All these factors make it difficult to get a good overview of the results using boxplots or simple scatterplots. In order to visualize the distribution of the results better, each dataset of 200 points is first clustered by distance into 15 clusters, which are then merged further when neccesary to reduce overlap on the plots. These clusters are then plotted as circles with area proportional to the number of points in the cluster. The circles are centered at the mean coordinates of the points it represent.

| | Reference algorithm ($N = 50$) | | Evolving roadmaps ($N = 15$) | |
|---|---|---|---|---|
| | Mean | Deviation | Mean | Deviation |
| ROCKY | 3.4797ms | 0.2475ms | 9.3554ms | 1.4893ms |
| SPIRALS | 2.2796ms | 0.0452ms | 9.6658ms | 1.2471ms |
| DETOUR | 1.5686ms | 0.0550ms | 4.3798ms | 0.6157ms |
| STAR | 1.7216ms | 0.0565ms | 9.0248ms | 1.1176ms |

**Table 6.1:** Single-objective run time per iteration

| | ROCKY | SPIRALS | DETOUR | STAR |
|---|---|---|---|---|
| Visibility graph | 1.0406 | 3.0002 | 1.5400 | 1.2101 |
| Reference algorithm | 1.0476 | 3.0002 | 2.2393 | 1.2101 |
| Evolving roadmap | 1.0433 | 3.0658 | 2.2398 | 1.2105 |

**Table 6.2:** Shortest distances found

## 6.1 Single-Objective Benchmark

Figures 6.1, 6.2, 6.3 and 6.4 show plots of run time versus shortest distance for all eight tests on each of the four maps. The shortest distance for each map, as found by the visibility graph algorithm, compared to the minimum distance found by each algorithm on any run is summarized in Table 6.2.

The plots show that the run time of the two algorithms are within the same order of magnitude with the parameters used. Considering that the termination criteria are set differently, this does not say anything about the run time needed per iteration. Table 6.1 summarizes the per iteration run time for the algorithms. For both algorithms the mean and standard deviation increase nearly linearly with population size, therefore only one population size is shown for each of them.

On the first map ERM performs as good as the reference algorithm even at low population sizes. The algorithms are able to find equally good solutions, but ERM produces a much smaller tail of sub-optimal solutions. The solutions produced by the reference algorithm seem to fall into discrete levels, possibly indicating local optima.

On the second map, SPIRALS, the solution quality produced by the reference algorithm is excellent, while the ERM solutions are much farther from the optimal value. ERM also shows large variation in solution quality for the smaller population sizes and in run time for the larger population sizes.

On the DETOUR map the reference algorithm has the best performance, with ERM showing more variation in solution quality, especially at lower population sizes. However, even the result range of the five-individual ERM is within half a percentage point of the total distance. Both algorithms fail to come close to the shortest path, however, indicating that neither of them have been able to find the narrow passage between the obstacles.

On the last map the reference algorithm is again able to find better solutions than ERM, with very little variation in solution quality, as in both SPIRALS and DETOUR. As in the SPIRALS environment, ERM has comparatively large variation in solution quality for the small populations, which decreases with population size, while run time variation increases.

Overall, the performance is clearly in favor of the reference algorithm in the single-objective case. The exception to this is the ROCKY environment, where ERM performs as good or perhaps better than in the other environments, while the reference algorithm performs substantially worse and has a different distribution of solution quality that could indicate that it has some difficulties escaping the local minima of this map.

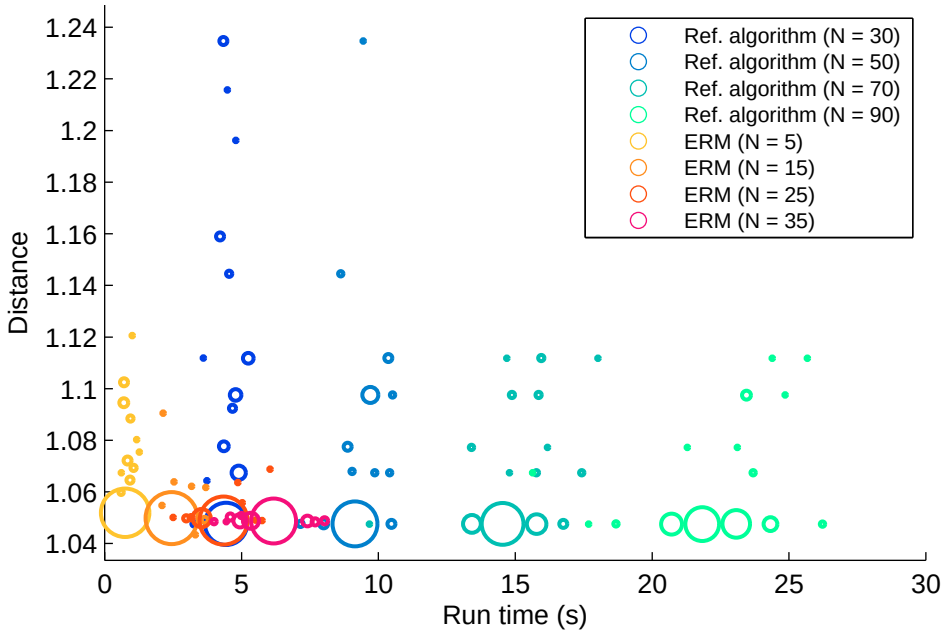



**Figure 6.1:** Single-objective results for the ROCKY environment

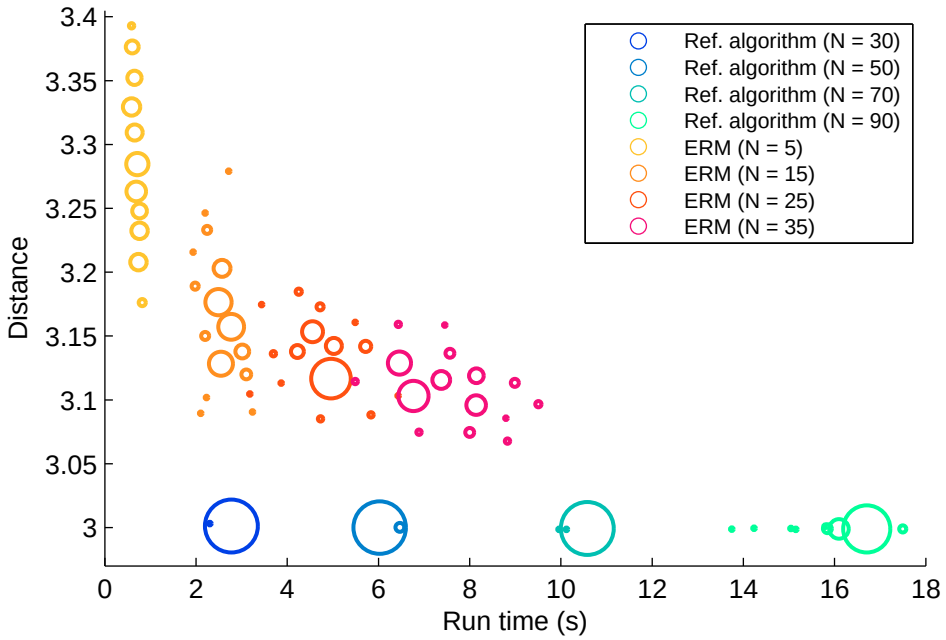



**Figure 6.2:** Single-objective results for the SPIRALS environment

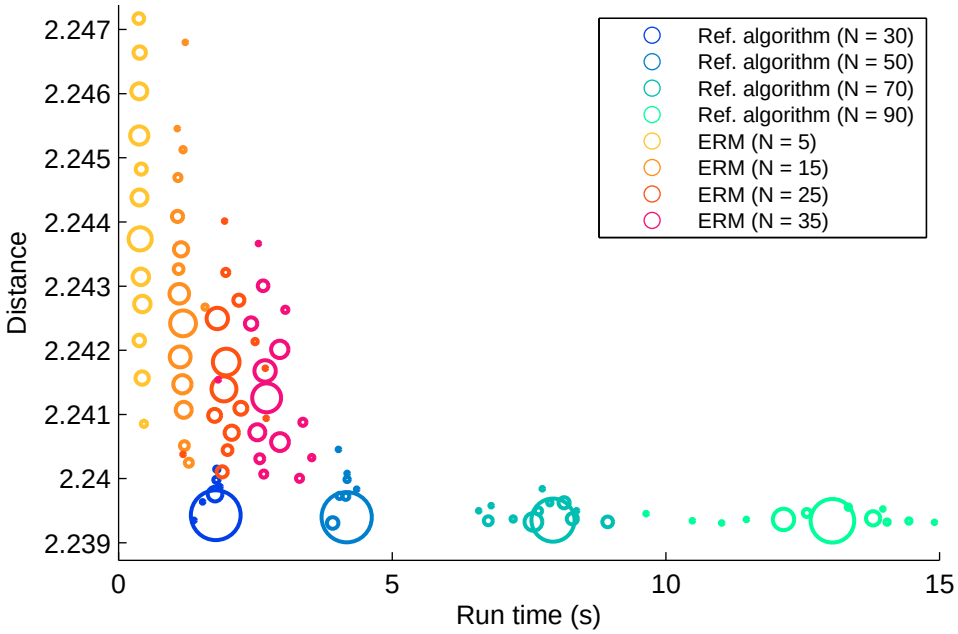



**Figure 6.3:** Single-objective results for the DETOUR environment

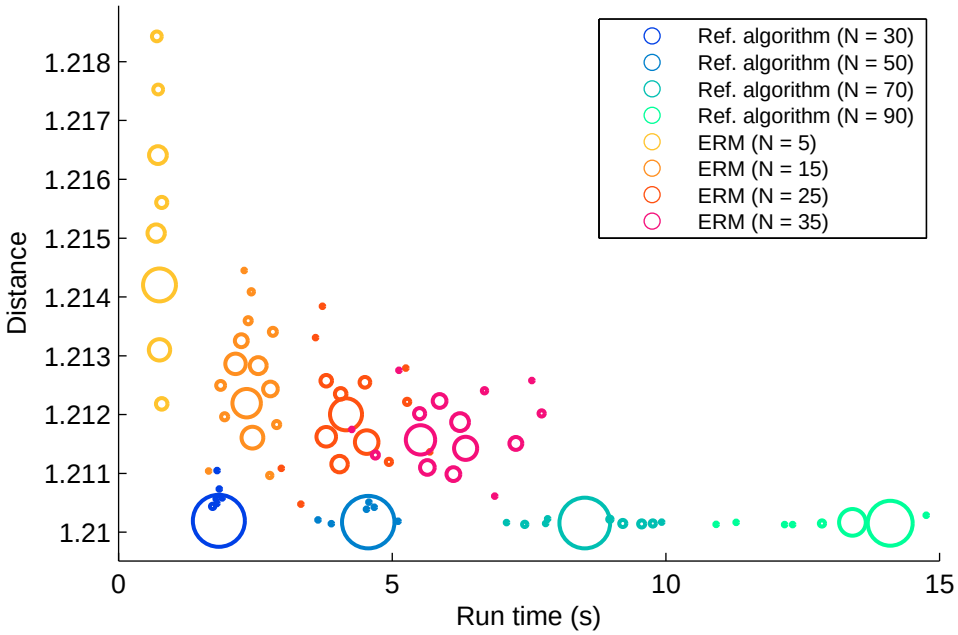



**Figure 6.4:** Single-objective results for the STAR environment

## 6.2 Multi-Objective Benchmark

Figures 6.6, 6.7, 6.8 and 6.9 plot run time versus hypervolume for the multi-objective tests for each of the four environments. The per iteration run time statistics are shown in Table 6.4. The shortest distances found in any run in the multi-objective test are shown in table 6.3. These results do not deviate significantly from the ones found in the single-objective case.

The reference point for all hypervolume calculations were $[200, 200]$. Because of the way the fitness functions are normalized, the straight line between start and goal would have had fitness $[1, 1]$ if it was feasible. This is the optimal path for both objectives, so the maximum hypervolume obtainable is less than $199 \times 199 \approx 3.96 \times 10^4$.

| | ROCKY | SPIRALS | DETOUR | STAR |
|---|---|---|---|---|
| Visibility graph | 1.0406 | 3.0002 | 1.5400 | 1.2101 |
| Reference algorithm | 1.0476 | 3.0002 | 2.2393 | 1.2102 |
| Evolving roadmap | 1.0480 | 3.0301 | 2.2396 | 1.2106 |

**Table 6.3:** Shortest distances found during multi-objective optimization

In the results for the ROCKY environment the hypervolume shows a clear tendency to separate into distinct levels. The highest level, around $3.95 \times 10^4$, is reached infrequently by both algorithms, while most runs settle around $3.94 \times 10^4$.

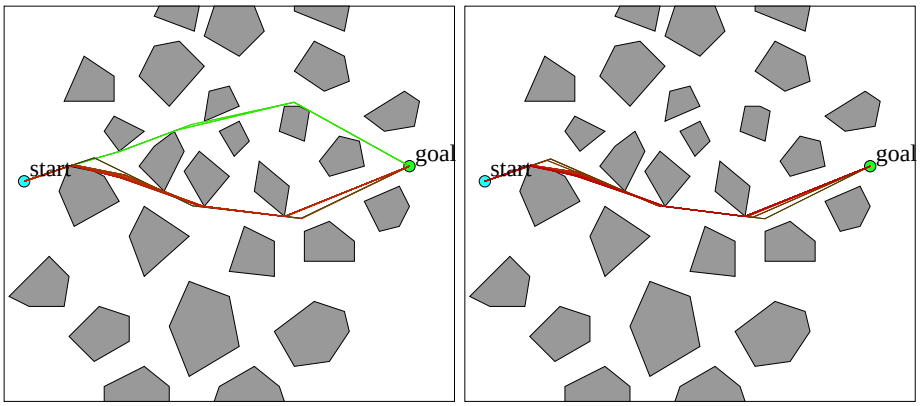
Figure 6.5 shows typical results from two of the hypervolume levels. Both results were generated by ERM with a population size of 15. Finding the green path in Figure 6.5a results in a significant improvement in the best estimated work and a corresponding improvement in hypervolume.

While the results in the upper level clearly forms a separate group, the results below are not spread out evenly, either. There is one level at approximately $3.943 \times 10^4$, in addition to the main group near $3.941 \times 10^4$. In addition, the results far below the main group probably belong to a separate distribution as well. In Table 6.5 the data has been divided into four groups by threshold the hypervolume values at $3.948 \times 10^4$, $3.942 \times 10^4$ and $3.939 \times 10^4$.

From the table one can see that the reference algorithm has produced solutions in group 1 more rarely than ERM, and actually has a decreasing number of results in group 1 and 2 when the population size is increased. In both algorithms the number of results that fall into group 4 decreases

| | Reference algorithm ($N = 50$) | | Evolving roadmaps ($N = 15$) | |
|---|---|---|---|---|
| | Mean | Deviation | Mean | Deviation |
| ROCKY | 3.6682ms | 0.2866ms | 14.0524ms | 1.8033ms |
| SPIRALS | 2.1132ms | 0.0872ms | 34.2536ms | 5.9525ms |
| DETOUR | 1.4510ms | 0.0544ms | 16.1827ms | 2.1694ms |
| STAR | 1.5402ms | 0.0480ms | 34.4735ms | 5.5661ms |

**Table 6.4:** Multi-objective run time per iteration




**(a)** Example 1: Two distinct groups of solutions have formed, one with with lower estimated work and one with slightly shorter distance. The hypervolume is 39517, with distances within $[1.0494, 1.0729]$ and estimated work within $[1.3726, 2.0307]$.

**(b)** Example 2: Only one of the groups have been found. The hypervolume is 39410, with distances within $[1.0491, 1.0682]$ and estimated work within $[1.9109, 2.1579]$.

**Figure 6.5:** Example runs for the ROCKY environment, showing all solutions from each run. Each solution is color-coded according to its fitness so that solutions close to the minimum distance are red and those close to the observed minimum work are green.

with population size, but the reference algorithm has a slightly higher percentage of its results in this category.

In the SPIRALS environment both algorithm show the same tendency: variants with few individuals have large variation in solution quality and low variation in solution quality, while for large populations the opposite is the case. Overall, the reference algorithm runs faster while ERM gives less varied quality. Especially with ERM the variance in run time is very high and there is much overlap between the different population sizes.

The results from the DETOUR environment paint much of the same

| | Group 1 | Group 2 | Group 3 | Group 4 |
|---|---|---|---|---|
| From | $3.948 \times 10^4$ | $3.942 \times 10^4$ | $3.939 \times 10^4$ | |
| To | | $3.948 \times 10^4$ | $3.942 \times 10^4$ | $3.939 \times 10^4$ |
| Ref (30) | 7.0% | 17.0% | 58.5% | 17.5% |
| Ref (50) | 4.0% | 15.5% | 75.5% | 5.0% |
| Ref (70) | 3.0% | 12.0% | 81.5% | 3.0% |
| Ref (90) | 0.5% | 10.5% | 86.0% | 3.0% |
| ERM (5) | 6.5% | 19.0% | 69.5% | 5.0% |
| ERM (15) | 17.0% | 5.0% | 75.0% | 3.0% |
| ERM (25) | 24.5% | 1.5% | 72.0% | 2.0% |
| ERM (35) | 27.0% | 1.0% | 70.5% | 1.5% |

**Table 6.5:** Distribution between hypervolume groups in ROCKY

picture as those of the SPIRALS environment: Low population sizes give high variation in solution quality, high population sizes give high variation in run time. The variations are smaller along both axes, however, and the population variants do not overlap in run time nearly as much as in SPIRALS. As with the single objective, both algorithms fail to come close to the shortest distance.

In the STAR environment one can again see the same patterns as in DETOUR and SPIRALS, except here there seems to be a ceiling to the solution quality of ERM, that does not apply to the reference algorithm. Thus the average result produced by the reference algorithm end up being better than that of ERM. And, while the reference algorithm has about the same run time as in DETOUR, ERM is significantly slower here.

As in the single-objective case, the results on the ROCKY environment differ from the trend in the other three environments. Overall, ERM performed slightly better in ROCKY, while in the other three environments the two algorithms present different trade-offs between short, predictable run time and reliable solution quality. However, in the STAR environment the average solution quality of the reference algorithm is significantly better than ERM, making that trade-off of little relevance.

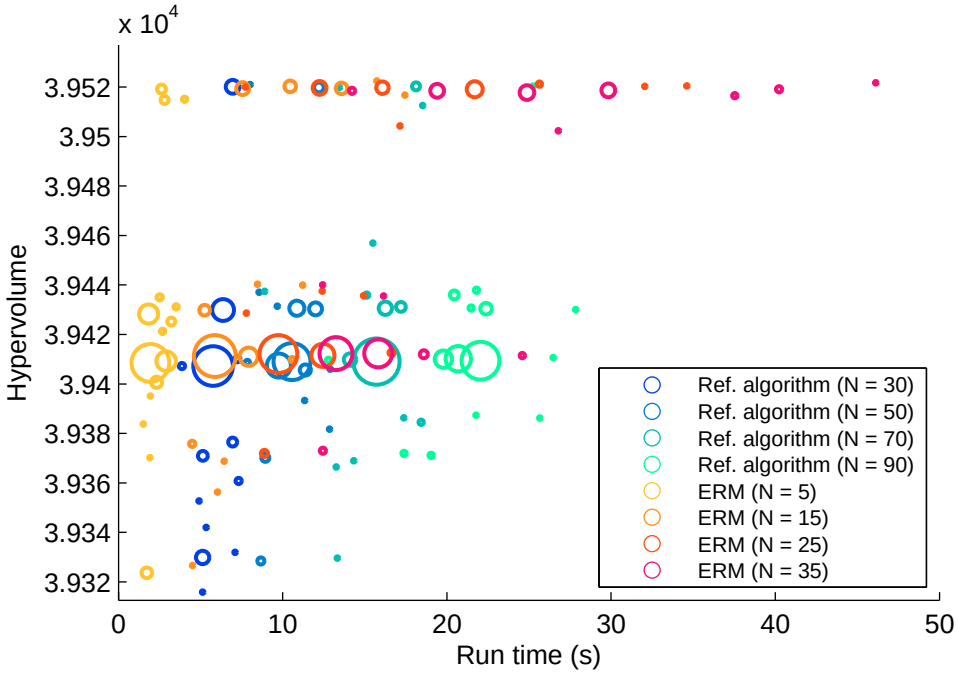



**Figure 6.6:** Multi-objective results for the ROCKY environment

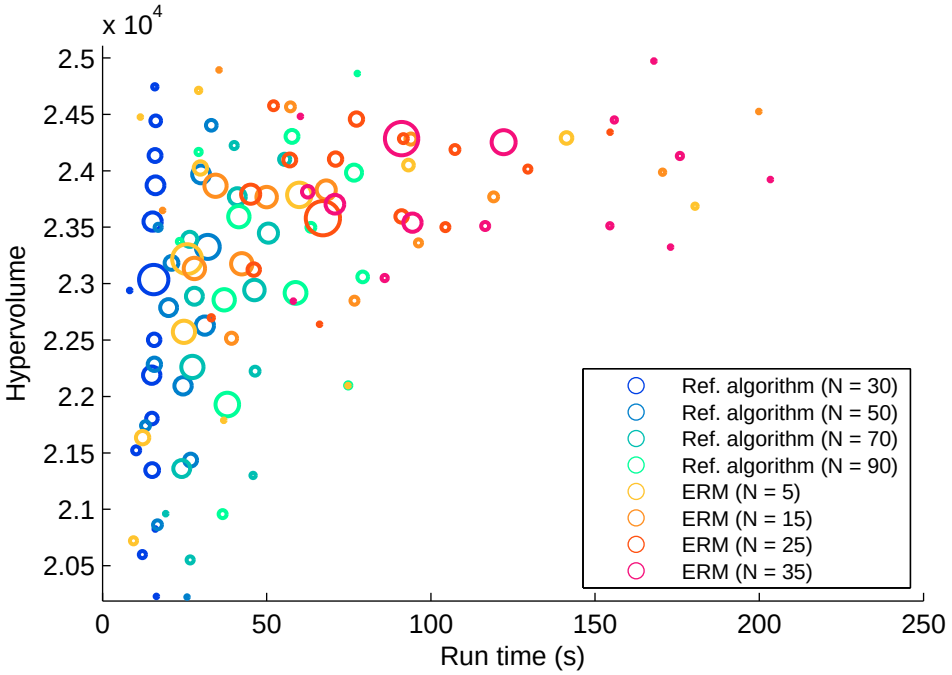



**Figure 6.7:** Multi-objective results for the SPIRALS environment

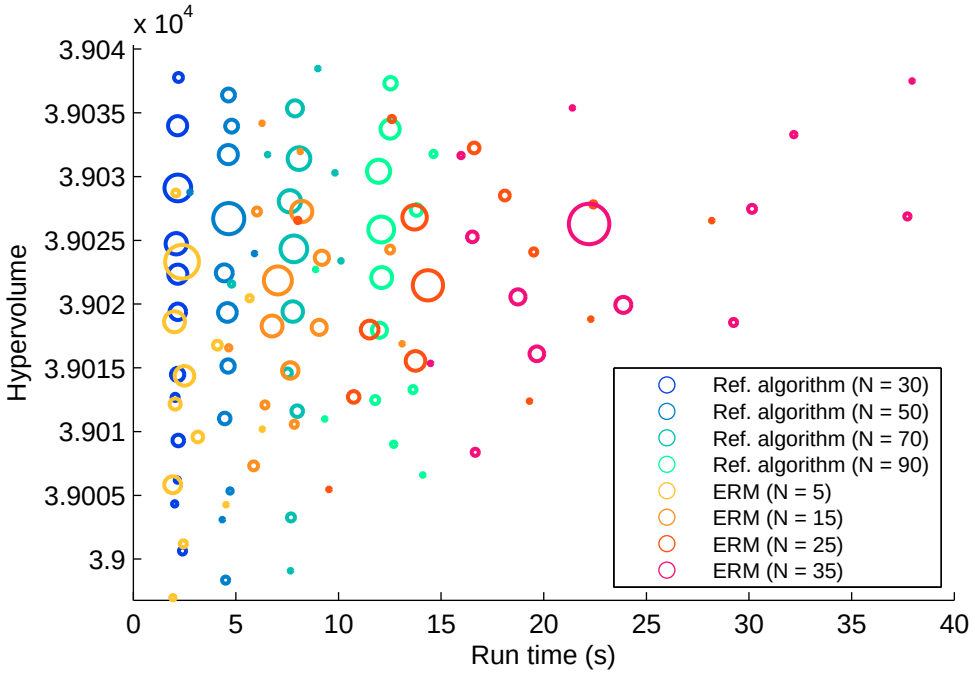



**Figure 6.8:** Multi-objective results for the DETOUR environment

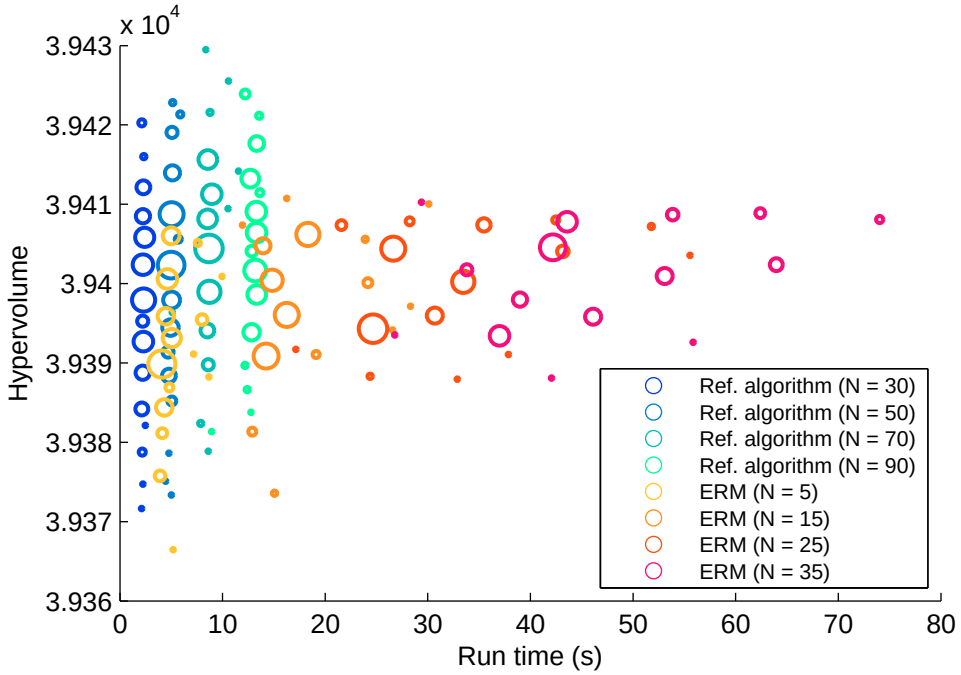



**Figure 6.9:** Multi-objective results for the STAR environment

# Chapter 7

# Conclusion

This chapter will conclude the thesis, beginning with a discussion of the results and some related issues. Then, possibilities for futher studies are suggested, before a final conclusive summary is given.

## 7.1 Discussion

The results show that the proposed algorithm does not perform better in general than a non-neutral reference algorithm, and often considerably worse in the single-objective case.

However, there is significant variation between environments in the performance of the two algorithms. Some of the environments are quite simple, and may not give a good indication of the performance in real-world scenarios.

In the most crowded and perhaps most realistic of the environments, ROCKY, ERM performed at least as well as the reference algorithm in the single-objective case, with a notably higher chance of reaching the best results in the multi-objective case. The performance in this environment will be discussed in more detail in section 7.1.2.

Neither algorithm was able to find the optimal but significantly more complex solution present in the DETOUR environment. This is likely a symptom of a larger issue related to population initialization, which is discussed in section 7.1.3.

### 7.1.1 Shortest Distance Performance

In the single-objective case, where minimization of distance was the lone objective, the reference algorithm showed much better performance than ERM, except for the ROCKY environment. While the reference algorithm performed at least as good as ERM in the same environments in the multi-objective case, it did not show nearly as big a difference in solution quality as with only a single objective.

One reason for this might lie in the evolutionary operators. While the evolutionary operators of ERM are about as general as they can be, the

mutation operators of the reference algorithm are designed with shortest distance optimization in mind.

Both shortcut and smoothen operators are guaranteed to shorten the path length if used. Only the change operator and crossover can create longer paths. With a different objective function longer paths might sometimes be better, and then the shortcut and smoothen mutations might not be as efficient as they are when optimizing for shortest distance.

This gives the reference algorithm an advantage over ERM, which does not make use of any domain knowledge beyond that it should try to connect points in some Cartesian space. If the estimated work had been used instead of the distance as the objective function the results might not have been as clearly in favor of the reference algorithm.

### 7.1.2 ROCKY Versus the Other Environments

In both the single- and multi-objective case there is a significant performance difference in the algorithms between ROCKY and the other three environments. The differences are most likely caused by differences in topology.

SPIRALS and STAR both have only one general route from start to goal, with a large open area in the middle of the route. The DETOUR environment has a second route in it, but the results show that this route was not found even once by the two algorithms in any of the tests.

The ROCKY environment, however, has a multitude of different routes, and no large open spaces between start and goal. The results indicate that either ERM performs better in this kind of environment, or the reference algorithm performs worse. It is here argued that there are at least two separate factors at work, one in favor of the reference algorithm in the other environments and one in favor of ERM in the ROCKY environment.

The first factor is related to one of the main reasons for developing ERM: The suspicion that algorithms like the reference algorithm are unable to reliably avoid the kind of local optima that are created when the solution space is split into separate routes. The single-objective results for the ROCKY environment seems to confirm these suspicions.

The fact that the results for ERM do not fall into fitness levels in the same way indicates that it is more robust in these situations, giving it an advantage over the reference algorithm that is not present in the other environments.

The other factor is the openness of the environment. When there are large open areas, the graphs created by the chromosomes in ERM will be more densely connected (there is a higher average number of outgoing edges per node) than in environments with narrow passages between evenly spread obstacles.

When the graphs are more densely connected more computations will be required to traverse these graphs. The best-first search of the A* algorithm minimizes the effect, but that is dependent on how good the heuristics of the objective functions are.

The distance heuristic estimates the remaining distance as the Euclidean distance between the current node and the goal, giving priority to edges that lead straight towards the goal. In the STAR and SPIRALS environments this is not of much use, since the only route available sometimes leads in a completely different direction. In these two environments ERM performs noticeably worse in run time in the single-objective tests.

No heuristic has been used for the rotational term of the estimated work function, while for the translational term the same heuristic is used as for the distance. Thus, the guesses made by the graph traversal are generally a bit worse in the multi-objective case, which coincides with smaller differences in run time variation for ERM between the different environments.

The ROCKY environment is important because it has similar topology to real-world indoor scenarios, with the obstacles representing equipment or furniture scattered around, possibly alongside autonomous agents or people. Good performance in this environment could suggest robustness outside of controlled environments as well.

### 7.1.3 Initialization in Evolutionary Path Planning

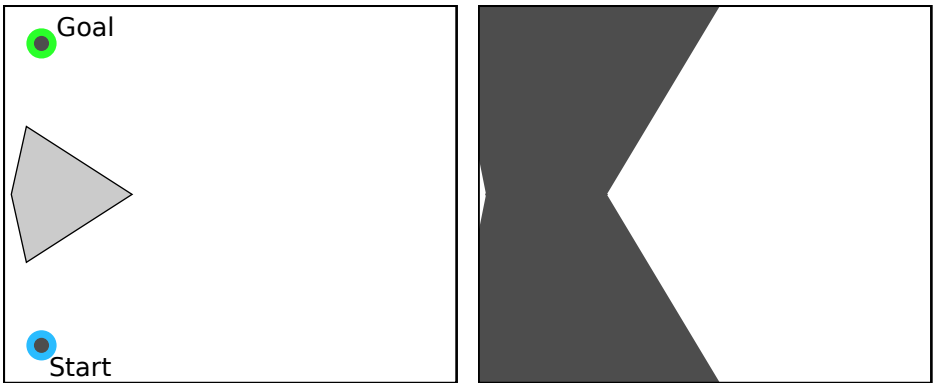
As mentioned, both algorithms failed to find the shortest path in the DETOUR environment. It is here suggested that this is because it is highly improbable that solutions going through the tight passage are generated during initialization. Furthermore, the subspace these solutions constitute is disconnected from the rest of solution space in such a way that it is also highly unlikely that they are generated through crossover and mutation from solutions outside of that subspace.

As an analogy, lets look at a simpler problem and a simpler algorithm. In Figure 7.1a a very simple path planning problem consisting of a single obstacle placed near one edge of the environment. The problem is constructed so that the shortest path passes through the tight passage between the boundary of the environment and the obstacle. The shortest path to the right side of the obstacle is about 14 percent longer than the shortest path to the left side.

The simplified algorithm is an evolutionary algorithm that represents the path using a single milestone, coded as an $x$-value and an $y$-value. The resulting solution space is mapped in Figure 7.1b. The small feasible area to the far left is 355 times smaller than the area to the right. If the population is initialized with feasible individuals that are randomly distributed, and a population size of 100 individuals, there would only be a 24.5% chance of having one or more individuals in the left area.

So around three out of four times the algorithm is run, no part of the initial population will be in the left area. Because of its exploitive nature, a pure crossover operator would not be able to generate a solution in that area either.

The only way left for a solution in the left area to appear would be through mutation. If the fitness function does not differentiate infeasible solutions (for example by assigning worse fitness to individuals digging




**(a)** The problem: Find the shortest path from start to goal.

**(b)** The solution space using a single-milestone EA. Infeasible parts are shaded dark gray. The optimal solution is in the small "island" of feasibility on the far left.

**Figure 7.1:** A Simple Path Planning Problem

deeper into obstacles) then the algorithm will not be any better at finding a solution in the left area than a random search.

If the algorithm does differentiate infeasible solutions (for example by assigning better fitness close to the edges of obstacles) then it might actually be even worse off here, because it might stop solutions from migrating across the infeasible area by leading them back to the nearest feasible point.

The similarities between this example and the situation in the DETOUR environment is striking. Since the passage containing the optimal solution is both narrower and requires more points to define than the way around, the VIS-PRM algorithm has a much higher chance of finding the way around rather than the way between the obstacles. Since the passage is narrow relative to the thickness of the obstacles, the probability of finding the optimal path by random mutation is very small.

This seems to indicate that the initialization method plays an important role in evolutionary path planners. Path planning problems are not only often multi-modal, but both the jumps between local optima and the differences in size of the basins of attraction between them can be so large, that the only way to ensure robust performance might be to ensure that the initialization covers all basins of attraction.

## 7.2 Future Work

An important motivation for exploring neutral algorithms for path planning is for the potential added robustness in changing environments. The algorithm performed its best with the cluttered ROCKY environment, and environments with moving objects might present some similar challenges, so it could be of great interest to apply the algorithm to dynamic environments.

In the tests done in this thesis, the simplest possible local planner (checking if the straight line between two points is obstructed) was used. However, more advanced local algorithms could be used, such as using artificial potential fields or even running an algorithm capable of global planning. Doing this might be beneficial to performance by reducing the number of nodes needed in each chromosome and improving obstacle avoidance in dynamic environments.

The algorithm proposed here, and the reference algorithm used, has some parameters related to the magnitude of mutations that were set to some fixed values during initial experiments. In order to improve fine-optimization and increase robustness to varying environment shapes and sizes it will most likely be beneficial to make these mutation parameters adapt over iterations, for example by adding one or more strategy parameters.

The initialization problems present in evolutionary path planners pose a serious problem to the general performance of these algorithms. Research is needed to find good, robust solutions to this problem for them to work efficiently out in the real world.

Using several of the many different graph construction methods in classical path planning might help. For example, initializing one individual with a visibility graph of the environment would almost guarantee that the shortest-distance path would be found. However, for many problems complete planners such as the visibility graph and Voronoi graph are too expensive to be run, or unknown or dynamic elements in the environment that are too complex could render them useless.

Both ERM and the reference planner has proved to be quite proficient at taking a sub-optimal path along a certain route and optimizing for that given route. So what might really be needed is a simple but robust meta-planner that can produce a graph of (all) possible routes.

## 7.3 Conclusion

This thesis presented evolving roadmaps (ERM), a multi-objective evolutionary path planner with neutrality. The algorithm introduces neutrality by coding chromosomes as nodes in a roadmap instead of a single path. Chromosomes then produce solutions by applying a graph traversal algorithm on it.

When optimizing for several objectives this method naturally results in multiple solutions being produced per chromosome. In the proposed algorithm several such solutions can be associated with a single chromosome, breaking with EA convention. To accommodate this a new multi-objective parent selection method was presented, along with a modification to the survival selection method of NSGA-II.

The performance of the algorithm was compared to that of a reference algorithm in four different environments and using both one and two objectives. A normalized distance measure was used as the first objective, and to act as a second objective, an estimation of the physical work needed

to follow the path at a constant speed was introduced.

Four different population sizes were used for each of the two algorithms in all tests. Lower population sizes were used for ERM than the reference algorithm after it was determined during initial tests that ERM could produce better solutions with low population sizes than what was possible with the reference algorithm. With the lower population sizes run time was of the same order of magnitude for both algorithms.

When optimizing for distance as a lone objective the reference algorithm was clearly superior to ERM in most of the environments. This may be attributed to the mutation operators in the reference algorithm being more specialized toward this objective function. In the multi-objective case the two algorithms were more even, with the reference algorithm showing a slight advantage in run time while ERM had somewhat less variation in solution quality.

In one environment, the results deviated significantly from the description above. This environment was more crowded, with many obstacles scattered around, creating a very large number of possible combinations of routes between the obstacles.

Here, ERM performed at least as well as the reference algorithm with both one and two objectives. In the single-objective case the reference algorithm fell into different discrete levels of fitness, a clear indication that it got stuck in local optima.

In the multi-objective case both algorithms fell into discrete levels of hypervolume, but ERM had a significantly higher probability of reaching the most favorable level. With ERM the chance of finding solutions on that level increased with population size, reaching 27% with the highest population size tested. In contrast, the reference algorithm's chances of finding that level was 7% the lowest population size, and actually decreased with population size.

This supports the hypothesis that a neutral algorithm would be more robust in complex, cluttered environments. However one of the other environments was set up to have a shortcut that would be very hard to find. This shortcut was not found by either algorithm, illustrating a limit to the power of evolutionary algorithms in general: When the method of initialization is unable to generate solutions from a basin of attraction with any certainty, the algorithm is no better (and usually worse) than a random search at finding solutions in that basin.

# Bibliography


[1] F. Ahmed and K. Deb. Multi-objective path planning using spline representation. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (IEEE-ROBIO 2011), Piscatway*, pages No–pp. Proceedings of the IEEE International Conference on Robotics and Biomimetics (IEEE-ROBIO 2011), Piscatway, 2011.

[2] I. Ashiru and C. Czarnecki. Optimal motion planning for mobile robots using genetic algorithms. In *Industrial Automation and Control, 1995 (I A C'95), IEEE/IAS International Conference on (Cat. No.95TH8005)*, pages 297 –300, 5-7jan 1995.

[3] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, September 1991.

[4] V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1018 –1023 vol.2, 1999.

[5] John Canny and Bruce Donald. Simplified voronoi diagrams. *Discrete & Computational Geometry*, 3:219–236, 1988. 10.1007/BF02187909.

[6] C.A.C. Coello, G.B. Lamont, and D.A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer-Verlag New York Inc, 2007.

[7] K. Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. Wiley, 2001.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, apr 2002.

[9] M. Ebner, P. Langguth, J. Albert, M. Shackleton, and R. Shipman. On neutral networks and evolvability. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 1–8. IEEE, 2001.

[10] M. Ebner, M. Shackleton, and R. Shipman. How neutral networks influence evolvability. *Complexity*, 7(2):19–33, 2001.

[11] A.E. Eiben and J.E. Smith. *Introduction to evolutionary computing (Natural computing series)*. Springer, 2008.

[12] A. Elshamli, H.A. Abdullah, and S. Areibi. Genetic algorithm for dynamic path planning. In *Electrical and Computer Engineering, 2004. Canadian Conference on*, volume 2, pages 677 – 680 Vol.2, may 2004.

[13] C.M. Fonseca, P.J. Fleming, et al. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the fifth international conference on genetic algorithms*, volume 1, page 416. Citeseer, 1993.

[14] T. Fraichard. Trajectory planning in a dynamic workspace: a'state-time space'approach. *Advanced Robotics, 13*, 6(8):75–94, 1999.

[15] K. Fujimura. Path planning with multiple objectives. *Robotics Automation Magazine, IEEE*, 3(1):33 –38, mar 1996.

[16] Edgar Galván-López, Riccardo Poli, Ahmed Kattan, Michael O'Neill, and Anthony Brabazon. Neutrality in evolutionary algorithms... what do we know? *Evolving Systems*, 2:145–163, 2011. 10.1007/s12530-011-9030-5.

[17] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.

[18] P. Hajela, E. Lee, and C.Y. Lin. Genetic algorithms in structural topology optimization. *Topology design of structures*, 227:117–134, 1993.

[19] S. Handley. The genetic planner: The automatic generation of plans for a mobile robot via genetic programming. In *Intelligent Control, 1993., Proceedings of the 1993 IEEE International Symposium on*, pages 190 –195, aug 1993.

[20] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100 –107, july 1968.

[21] Inman Harvey and Adrian Thompson. Through the labyrinth evolution finds a way: A silicon ridge. In Tetsuya Higuchi, Masaya Iwata, and Weixin Liu, editors, *Evolvable Systems: From Biology to Hardware*, volume 1259 of *Lecture Notes in Computer Science*, pages 406–422. Springer Berlin / Heidelberg, 1997. 10.1007/3-540-63173-9_62.

[22] D. Henrich, C. Wurll, and H. Worn. On-line path planning by heuristic hierarchical search. In *Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE*, volume 4, pages 2239 –2244 vol.4, aug-4 sep 1998.

[23] C. Hocaoglu and A.C. Sanderson. Planning multi-paths using speciation in genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 378 –383, may 1996.

[24] J.H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

[25] Kao-Ting Hung, Jing-Sin Liu, and Yau-Zen Chang. A comparative study of smooth path planning for a mobile robot by evolutionary multi-objective optimization. In *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on*, pages 254 –259, june 2007.

[26] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration for fast path planning. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 2138 – 2145 vol.3, may 1994.

[27] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[28] P. Khosla and R. Volpe. Superquadric artificial potentials for obstacle avoidance and approach. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 1778 – 1784 vol.3, apr 1988.

[29] M. Kimura et al. Evolutionary rate at the molecular level. *Nature*, 217(5129):624, 1968.

[30] J.R. Koza. *Genetic programming*. Citeseer, 1992.

[31] J.C. Latombe. *Robot motion planning*. Kluwer international series in engineering and computer science. Kluwer Academic, 1991.

[32] S.M. LaValle. *Planning algorithms*. Cambridge Univ Pr, 2006.

[33] C. Leger. *Automated synthesis and optimization of robot configurations: an evolutionary approach*. PhD thesis, Citeseer, 1999.

[34] H.S. Lin, J. Xiao, and Z. Michalewicz. Evolutionary algorithm for path planning in mobile robot environment. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 211–216. IEEE, 1994.

[35] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32:108–120, 1983.

[36] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, October 1979.

[37] L. Lulu and A. Elnagar. A comparative study between visibility-based roadmap path planning algorithms. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3263 – 3268, aug. 2005.

[38] T. Murata and H. Ishibuchi. Moga: multi-objective genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 289, nov-1 dec 1995.

[39] M. Naderan-Tahan and M.T. Manzuri-Shalmani. Efficient and safe path planning for a mobile robot using genetic algorithm. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2091 –2097, may 2009.

[40] C. Nissoux, T. Simeon, and J.-P. Laumond. Visibility based probabilistic roadmaps. In *Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 3, pages 1316 –1321 vol.3, 1999.

[41] M.H. Overmars et al. A random approach to motion planning. Technical report, 1992.

[42] F. Rothlauf and D.E. Goldberg. Redundant representations in evolutionary computation. *Evolutionary Computation*, 11(4):381–415, 2003.

[43] Eivind Samuelsen, Kyrre Harald Glette, and Kazi Shah Nawaz Ripon. A neutral evolutionary path-planner. In Masanori Sugisaka and Hiroshi Tanaka, editors, *Proceedings of the 17th International Symposium on Artificial Life and Robotics*, pages 851–854, 2012.

[44] J.D. Schaffer. *Some experiments in machine learning using vector evaluated genetic algorithms*. PhD thesis, Vanderbilt Univ., Nashville, TN (USA), 1985.

[45] M. Shackleton, R. Shipman, and M. Ebner. An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 493–500. IEEE, 2000.

[46] R. Shipman, M. Shackleton, M. Ebner, and R. Watson. Neutral search spaces for artificial evolution: A lesson from life. *Artificial Life*, 7:162–169, 2000.

[47] A.R. Soltani and T. Fernando. A fuzzy based multi-objective path planning of construction sites. *Automation in Construction*, 13(6):717 – 734, 2004.

[48] N. Srinivas and K. Deb. Muiltiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[49] Bradley S. Stewart and Chelsea C. White, III. Multiobjective a*. *J. ACM*, 38(4):775–814, October 1991.

[50] M. Toussaint and C. Igel. Neutrality: A necessity for self-adaptation. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1354–1359. IEEE, 2002.

[51] J. Vannoy and Jing Xiao. Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes. *Robotics, IEEE Transactions on*, 24(5):1199 –1212, oct. 2008.

[52] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski. Adaptive evolutionary planner/navigator for mobile robots. *Evolutionary Computation, IEEE Transactions on*, 1(1):18–28, 1997.

[53] E. Zitzler, L. Thiele, E. Zitzler, E. Zitzler, L. Thiele, and L. Thiele. *An evolutionary algorithm for multiobjective optimization: The strength pareto approach*. Citeseer, 1998.

# Appendix A

# Environment Definition Listings

The environments used in the benchmark are listed below as Listings A.1, A.2, A.3 and A.4. The environments are defined in a simple Lisp-like syntax, and should be relatively easy to parse:

The `(bounds ...)` expression describes the boundary of the environment, and each `(object ...)` expression defines an obstacle.

`(polygon x0 y0 x1 y1 ...)` defines a polygon by pairs of numbers representing the $x$ and $y$ coordinates of the corners of the polygon. The geometric definition of the border and obstacles was separated in order to make it easy to add other types of geometry later (for example circular obstacles).

`(start x y)` and `(goal x y)` defines the position of the start and goal points respectively.

**Listing A.1:** ROCKY definition

```
(bounds (polygon 2.5 2  11.5 2  11.5 9.9  2.5 9.9))
(object (polygon 4.7 4.2  4.5 4.5  4.8 4.9  5.3 4.5))
(object (polygon 6.4 4.9  6.1 5.3  6.5 6  7 5.4))
(object (polygon 5.3 6  5 6.6  5.3 7.4  6.2 6.6))
(object (polygon 7.3 6.4  7 7.3  7.9 7.4  7.9 6.7))
(object (polygon 7.6 5.1  7.5 5.7  8.1 6.2  8.2 5.6))
(object (polygon 6.2 7.5  5.8 8.4  6.2 9.4  7.2 8.8  7 7.8))
(object (polygon 5.5 2.8  5.2 3.4  5.8 4  6.5 3.2  6 2.7))
(object (polygon 6.7 2  6.5 2.6  7.2 3  7.7 2.5  7.5 2))
(object (polygon 5.5 4.8  5.2 5.2  5.7 5.7  6.1 4.9  5.9 4.5))
(object (polygon 7.2 4.3  6.8 4.5  7.1 5  7.4 4.7))
(object (polygon 8.7 2.7  8.3 3.3  8.9 3.7  9.4 3.5  9.3 3))
(object (polygon 9.2 4.6  8.8 5.1  9 5.4  9.7 5.2  9.6 4.7))
(object (polygon 9 6.3  8.5 6.5  8.5 7.1  9.5 7.1  9.5 6.7))
(object (polygon 8.7 7.9  7.9 8.5  8.3 9.2  9 9.1  9.4 8.5  9.3 8.1))
(object (polygon 4.4 8  3.8 8.6  4.3 9.1  5 8.7  5 8.2))
(object (polygon 3.8 5.2  3.6 5.7  3.9 6.4  4.8 5.9  4.5 5.4  4.3 5.3))
(object (polygon 3.4 7  2.6 7.8  3 8  3.7 8  3.8 7.4))
(object (polygon 10.5 3.7  9.7 4.2  10.1 4.5  10.7 4.4  10.8 3.9))
(object (polygon 10.4 5.6  9.7 5.9  10 6.5  10.4 6.4  10.6 6))
(object (polygon 5.3 9.2  4.5 9.6  4.5 9.9  5.8 9.9  5.8 9.6))
(object (polygon 8.3 2  8.3 2.1  9.3 2.5  9.4 2))
(object (polygon 8.1 4  8 4.5  8.5 4.7  8.6 4.1  8.4 4))
(object (polygon 6.6 3.7  6.5 4.3  7.2 4  7 3.6))
(object (polygon 7.5 9.2  6.8 9.6  6.7 9.9  8.1 9.9  8 9.6))
(object (polygon 5.6 2  5.5 2.2  6.3 2.5  6.4 2))
(object (polygon 4.1 3  3.7 3.9  4.7 3.9  4.7 3.4))
(start 2.9 5.5)
(goal 10.6 5.2)
```

**Listing A.2:** SPIRALS definition

```
(bounds (polygon 2.5 2  11.5 2  11.5 9.9  2.5 9.9))
(object (polygon 4.5 4.5  5 4.5  5 5  4 5  4 3.5  6 3.5  6 6  4 6  4 6.5
  6.5 6.5  6.5 3  3.5 3  3.5 5.5  5.5 5.5  5.5 4  4.5 4))
(object (polygon 8.5 7  9 7  9 7.5  8 7.5  8 6  10 6  10 8.5  8 8.5  8 9
  10.5 9  10.5 5.5  7.5 5.5  7.5 8  9.5 8  9.5 6.5  8.5 6.5))
(start 4.6 4.7)
(goal 8.7 7.2)
```

**Listing A.3:** DETOUR definition

```
(bounds (polygon -8 -4  -8 13  14 13  14 -4))
(object (polygon -8 -1  1 -1  1 0  -3 0  -3 3  1 3  1 4  -3 4  -3 7
  1 7  1 8  -3 8  -3 10  -8 10))
(object (polygon 4 -1  4 1  -1 1  -1 2  4 2  4 5  -1 5  -1 6  4 6
  4 9  -1 9  -1 10  9 10  9 -1))
(start 0 11)
(goal 0 -2)
```

**Listing A.4:** STAR definition

```
(bounds (polygon 11 -4  -8 -4  -8 14  11 14))
(object (polygon -8 0  -4 0  -4 -3  -3 -3  -3 1  -8 1))
(object (polygon 11 10  7 10  7 13  6 13  6 9  11 9))
(start 9 12)
(goal -6 -2)
```

# Appendix B

# A Neutral Evolutionary Path Planner

# A neutral evolutionary path-planner

Eivind Samuelsen, Kyrre Harald Glette, and Kazi Shah Nawaz Ripon

Department of Informatics, University of Oslo, Norway
{eivinsam,kyrrehg,ksripon}@ifi.uio.no

**Abstract:** This paper explores methods for path-planning using evolutionary algorithms. Inspired by research on neutral mutations in evolutionary algorithms, we propose an algorithm based on the idea of introducing redundancy in the solutions, adding explicit neutrality to the evolutionary system. The algorithm introduce explicit neutrality by evolving roadmaps rather than single paths. Since some of the mutation and crossover operators used in conventional evolutionary path-planners are not well suited for this representation, appropriate evolutionary operators will also be explored. The performance of this algorithm on shortest distance path planning problems is compared to a known good genetic algorithm in three different static environments.

**Keywords:** genetic algorithm, neutrality, path-planning, roadmap

## 1 INTRODUCTION

Path planning is the problem of finding an optimal obstacle-free path through an environment. In a shortest distance problem, the optimal path is the one going from a start point to a goal point in the shortest possible distance travelled. Many methods have been proposed for solving this optimization problem. They all have certain trade-offs between planning time, robustness, requirements on environment representation and so on. An overview of the most common no-evolutionary methods can be found in [1].

Probabilistic roadmaps (PRMs) are one of the more common methods in path planning. A PRM samples random points in order to simplify the problem and scale well with environment dimensionality. However, it will seldom find optimal solutions unless the amount of samples taken is made impractically large. They also have problems with finding solutions at all in narrow passages and maze-like environments, a problem that to some extent can be remedied by sampling adaptively [2].

Several path-planning methods using evolutionary algorithms (EAs) have also been proposed. A straightforward approach to the shortest distance problem is presented in [3]. Once a good set of solutions has been found by an EA path-planner, it can easily adapt to changes in the environment simply by running the a few more iterations of the algorithm on the updated environment data. However, finding a good set of initial solutions from a set of random solutions can take some time. A remedy for this is to initialize the population with results from a PRM planner has been proposed in [4].

There has recently been some interest in EAs using neutrality and neutral mutations. Neutrality is having redundancy or extra information in the chromosomes so that they can change in ways that do not affect fitness. It is claimed that this can be greatly beneficial to EAs, making them both converge faster to the optimum and also escape local optima more easily. This is inspired by similar theories in biological evolution and some experimental findings, though many of these findings are considered inconclusive by others. A summary of this research can be found in [5].

Although the effects of neutrality in EAs in general are still inconclusive, it is an interesting avenue to explore in solving path-planning problems. One such method is proposed in this paper. Instead of trying to evolve good paths directly, this method tries to evolve good roadmaps for finding paths. We will rank the roadmaps only by the best path found in them by a graph traversal algorithm. Since this ranking does not consider the points in the roadmap that are not part of the resulting path it has explicit neutrality.

## 2 PROPOSED METHOD

Starting out with a set of PRMs as initial population, our algorithm evolves these roadmaps over many generations. We call this method a roadmap evolver. Roadmaps well suited to find a good path are more likely to survive to the next generation. The nodes that are not on a roadmap's best path can mutate without affecting the roadmap's fitness. This can enable the population to escape local optima more easily.

### 2.1 Representation and evaluation function

Each chromosome is a set of floating-point vectors that together with the start point and the goal constitute the nodes, or milestones, of a roadmap. If one can draw a straight line between two nodes without intersecting any obstacle, they are connected. A graph traversal algorithm is run to evaluate the chromosomes, finding the best path through it from the start to the goal node.

When an optimal path through the roadmap has been found, the fitness of that path is taken as the fitness of the roadmap. In the experiments done in this paper the path's are optimized for the shortest path. So the fitness of the path

is the path's curve length, and the graph traversal is done by A*.

Since not all nodes are necessarily visited during the traversal, the connectivity of the roadmap graph is done dynamically during the graph traversal in order to reduce the number of connectivity checks needed. Still, in general, $O(n^2)$ connectivity checks will be needed in the traversal, where $n$ is the number of nodes in the roadmap.

### 2.2 Evolutionary operators

The algorithm proposed has a single crossover operator and three mutation operators: nudge, insert and reduce size.

In order to emphasize average-sized children we use a crossover operator similar to uniform crossover: First each point in parent A is given to either child A or child B with equal probability. Then the points in parent B are distributed in the same way. The size of each child is the sum of $n$ coin flips, so it has a binominal distribution. The chromosomes are sets of points and thus inherently unordered. Therefore, there is no need to try to maintain any ordering.

The nudge mutation goes through all the points in a chromosome. Each point has a probability $P_{\text{nudge}}$ of being displaced a small distance in some random direction, but only if that displacement does not move the point into or over an obstacle: A normal-distributed vector is added to the point if the straight line between the original point and the point plus the random vector is obstacle-free. If this fails, it is reattempted up to 6 times with a new random vector with increasingly narrow normal distribution. If all attempts fail that point is left unchanged.

The insert mutation tries to add a single point to an individual. A candidate point is generated randomly with uniform probability within the bounds of the environment. If the candidate is obstructed, a short random walk is performed until an unobstructed point is found, or a maximum of three steps are taken. If the candidate is still obstructed, we try it up to 5 more times with different random start points. If no feasible candidate has been found, the operator leaves the individual unchanged.

The reduce size mutation goes through all the points in a chromosome and removes that point with a probability $P_{\text{remove}}$ . Thus, on average $n \times P_{\text{remove}}$ points are removed from the set, where $n$ is the original size of the set.

### 2.3 Initial population and evolutionary process

The population is initialized with $N$ PRMs with an average of $N_g$ non-obstructed nodes in uniform distribution over the environment. If none of the roadmaps give a feasible path from start to a goal then $N_g$ is increased by a constant and $N$ new PRMs are created. This is repeated until at least one initial feasible solution is found.

Each generation is generated as follows: $N$ new individuals are created by crossover. The parents for each crossover operation are selected by simple tournament selection with a tournament size of 2. This is repeated until two different parents have been selected.

All the new individuals are then subjected to mutation. The nudge mutation is performed once, then the insert mutation is performed once with probability $P_{\text{insert}}$ . Finally the reduce size mutation is performed. After that, new population is merged with the old. To reduce the population size back to normal, $N$ one-round tournaments are held, and the losing individuals are removed.

Each new individual has $P_{\text{insert}}$ points added on average, while $n \times P_{\text{remove}}$ points are removed. This helps the average chromosome size to stay relatively stable, while allowing for variations in order to adapt to different environments.

Table 1: Average run-time, including initialization

| Environment | Reference path-planner (per iteration) | Roadmap evolver (per iteration) |
|---|---|---|
| "Rocks" | 19.0s (10.6ms) | 78.3s (174.0ms) |
| "House" | 49.0s (27.2ms) | 163.6s (363.6ms) |
| "Spirals" | 25.7s (14.3ms) | 131.0s (291.1ms) |

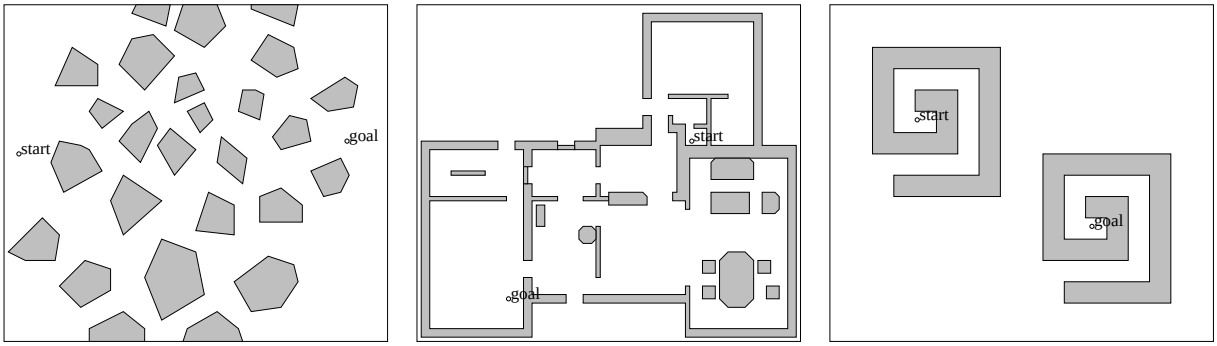
## 3 RESULTS AND DISCUSSION

The algorithm has been tested against a reference path-planner in different environments. The reference algorithm used is similar to the one described in [4], differing mainly in the parameters and selection operators used. It is run with a population size of 40 and a tournament-based selection. It should be noted that the parameters and implementation of the reference algorithm are not fine-tuned or optimized, and can therefore only serve as a general guideline to the performance of a reasonably robust evolutionary path-planner.

The roadmap evolver was run with a population size of 35 and an average initial number of nodes per chromosome of 35 or more. Mutation parameters were set to $P_{\text{nudge}} = 0.2$, $P_{\text{insert}} = 0.4$ and $P_{\text{remove}} = 0.05$.

Three different environments, shown in Fig.1 are tested. Each environment has different characteristics: The "rocky" environment consists of a number of scattered convex shapes, and has a large number of intersecting local optima. The second environment imitates the interior of a single-floor home, and has wall- and corridor-like obstacles, leading to a few, far-apart local optima. The third environment contains two spiral structures and has only one hard-to-find optimum.

The result of 100 runs of each algorithm in each environment is shown in Fig.2. The reference algorithm is run for 1800 iterations, while the roadmap evolver is run for 450 iterations. The large difference in number of iterations is due to differences in run-time per iteration and to differences in how




(a) The "rocks" environment (b) The "house" environment (c) The "spirals" environment

Figure 1: The different environments tested

quickly the algorithms converge. The average run-times are shown in Table 1. The average iteration time of the roadmap evolver is more than ten times that of the reference algorithm.

The reference algorithm shows very good performance in the "house" environment, with a good start and little variation. In "rocks" it converges quickly, but seems to get stuck in local optima, because the variation between runs stays high. In the last environment it closes in on the optimum with low variation, but at a very slow pace.

The roadmap evolver has good performance in the "rocks" environment, converging quickly towards a near-optimal solution quickly and with little variation between runs. In the other two environments it closes in on the optimal solution at a good average pace, but the pace is very uneven, with large variations in fitness between runs for many generations.

Compared to the reference algorithm in terms of generations only, the roadmap evolver performs as good or better in both "rocks" and "spirals" environments, but needs more generations to reliably produce near-optimal solutions in the "house" environment. In "rocks" it seems to avoid local optima better, while in "house" it shows signs of jumping out of local optima it have been temporarily stuck in. Note that both algorithms fail to ever find the globally optimal path in the "rocks" environment.

The good performance of the reference algorithm on "house" might be explained by the way the populations are initialized - the reference algorithm guarantees $N$ different feasible solutions, while the roadmap evolver only guarantees at least one.

## 4 CONCLUSION

In this paper, we proposed a path-planning method using a genetic algorithm that has explicit neutrality. The algorithm was compared to an existing genetic path-planning algorithm that has no neutrality. The proposed algorithm closes in on the optimal solution in comparatively few generations, and shows signs of both avoidance and escape of local minima in a small selection of tested environments. However, run-time performance is poor, most likely because of comparatively high complexity in the fitness function.

The algorithm did display properties expected of a neutral evolutionary system. For future work it would be of interest to examine whether these properties are displayed in other situations too, such as dynamic or partially unknown environments, or path-planning problems with multiple objectives. Initializing the population in the same way as the reference algorithm might increase performance in early generations. Another possibility is developing other different and possibly more efficient neutral path-planning EAs and examine if the same properties appear in them.

## REFERENCES

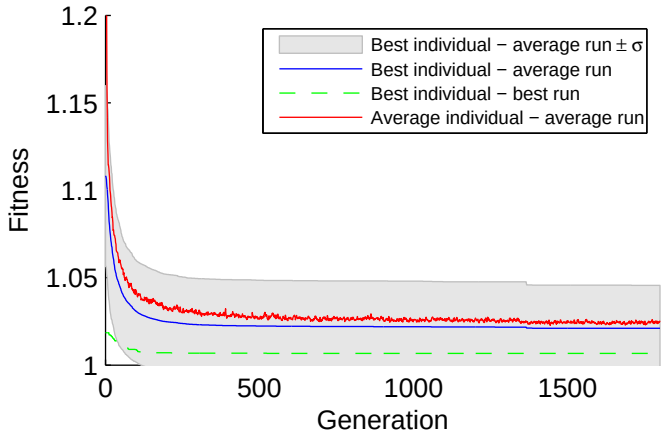
[1] S. LaValle, *Planning algorithms*. Cambridge Univ Pr, 2006.

[2] V. Boor, M. Overmars, and A. van der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, pp. 1018 –1023 vol.2, 1999.

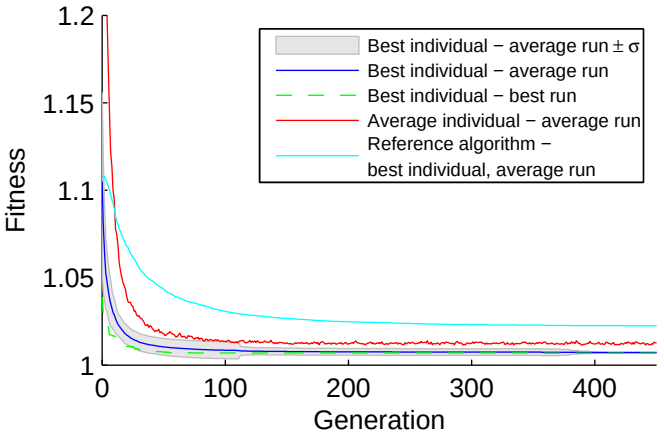
[3] H. Lin, J. Xiao, and Z. Michalewicz, "Evolutionary algorithm for path planning in mobile robot environment," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pp. 211–216, IEEE, 1994.

[4] M. Naderan-Tahan and M. Manzuri-Shalmani, "Efficient and safe path planning for a mobile robot using genetic algorithm," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pp. 2091 –2097, may 2009.

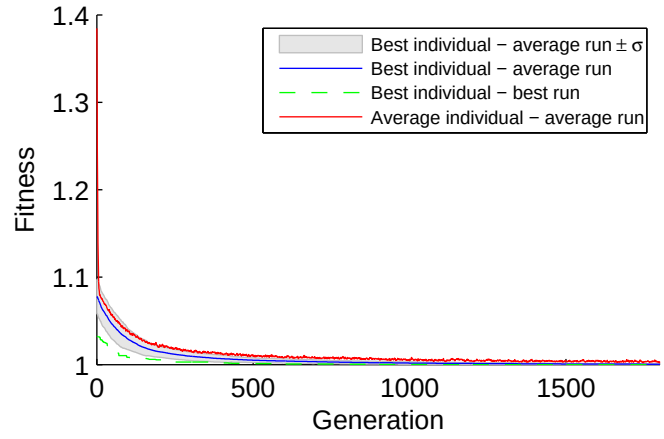
[5] E. Galván-López, R. Poli, A. Kattan, M. O'Neill, and A. Brabazon, "Neutrality in evolutionary algorithms... what do we know?," *Evolving Systems*, vol. 2, pp. 145–163, 2011. 10.1007/s12530-011-9030-5.



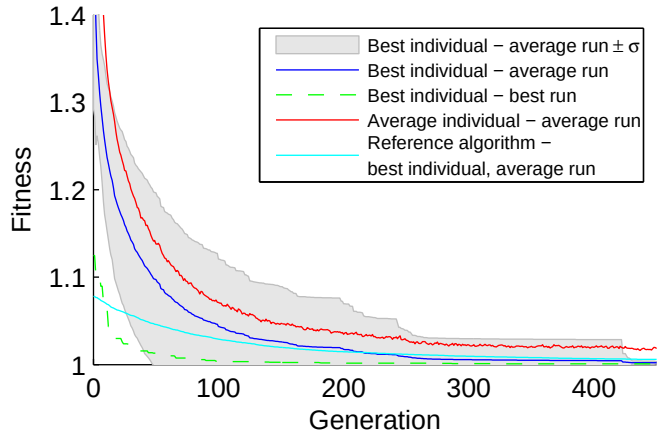

(a) "Rocks" environment - reference path-planner



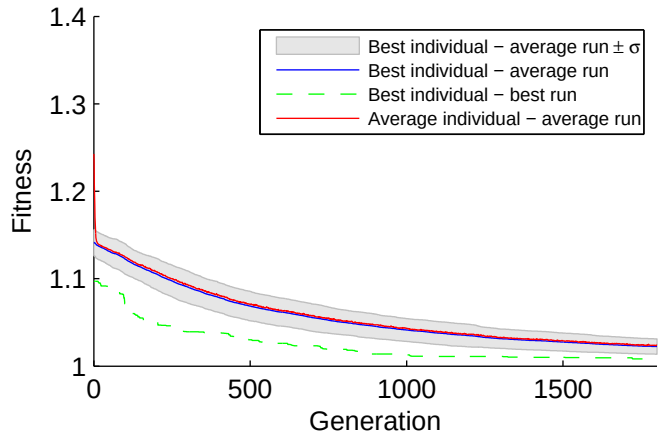

(b) "Rocks" environment - roadmap evolver



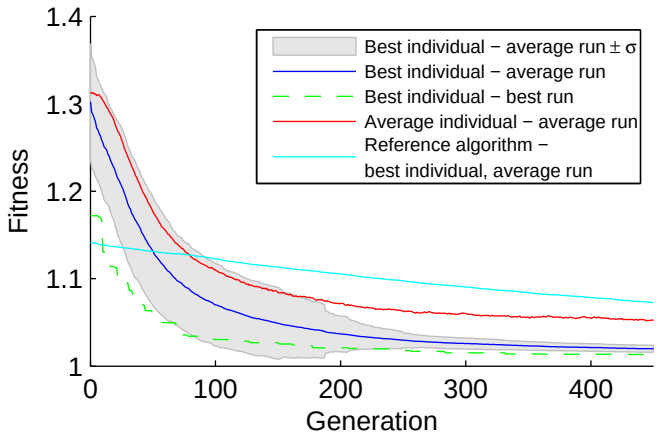

(c) "House" environment - reference path-planner




(d) "House" environment - roadmap evolver




(e) "Spirals" environment - reference path-planner




(f) "Spirals" environment - roadmap evolver

Figure 2: Fitness development for each algorithm in each environment. All fitness values are relative to the optimal solution as found by the vismap algorithm. The blue line is the average best fitness for each run. The light grey area around it signifies one standard deviation from that average. The red line is the average fitness of all feasible solutions in the current generation, averaged over all runs. The cyan line in the roadmap evolver plots shows the average best fitness for the reference algorithm for the same iterations.