Distance determination algorithms for convex and concave objects

by

Juan Antonio Carretero G.
B. Eng., National University of Mexico (UNAM), 1996
M. A. Sc., University of Victoria, 1998

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Mechanical Engineering.

We accept this dissertation as conforming
to the required standard

Dr. M. Nahon, Supervisor (Dept. of Mechanical Engineering)

Dr. I. Sharf, Departmental Member (Dept. of Mechanical Engineering)

Dr. Z. Dong, Departmental Member (Dept. of Mechanical Engineering)

Dr. W.-S. Lu, Outside Member (Dept. of Electrical and Computer Engineering)

Dr. Kamal Gupta, External Examiner (School of Eng. Science, Simon Fraser U.)

Supervisor: Dr. M. Nahon

Abstract

Determining the minimum distance between two objects is a problem that has been solved using many different approaches. Most methods proposed so far are, in essence, limited to solve the problem amongst convex polyhedra. Thus, to deal with concave objects, these methods partition concave objects into convex sub-objects and solve the convex problem between all possible sub-object combinations. This adds a large computational expense, especially when the concave objects in the scene are complicated, or when concave quadratically bound objects are to be linearized.

In this work, two optimization-based formulations are proposed to solve the minimum distance problem without the need for partitioning concave objects into convex sub-objects. The first one, referred to as the *continuous approach*, uses concepts of computational solid geometry in order to represent objects with concavities. On the other hand, in the second formulation, referred to as the *combinatorial approach*, the geometries of the objects are replaced by large sets of points arranged in surface meshes.

Since the optimization problem is not unimodal (*i.e.*, has more than one local minimum point), global optimization techniques are used. Simulated Annealing and Genetic Algorithms, with constraint handling techniques such as penalty and repair strategies are used in the continuous approach. In order to eliminate the computational expense of determining the feasibility of every trial point, the combinatorial approach replaces the objects' geometry by a set of points on the surface of each object. This reduces the minimum distance problem to an unconstrained combinatorial optimization problem where the combination of points (one on each object) that minimizes the distance between objects is the solution.

Additionally, Genetic Algorithms with niche formation techniques were developed in order to allow the distance algorithm to track multiple minima.

In a series of numerical examples, a preliminary implementation of the proposed algorithms has proven to be robust and equivalent, in terms of computational efficiency, to some conventional approaches.

Examiners:

Dr. M. Nahon, Supervisor (Dept. of Mechanical Engineering)

Dr. I. Sharf, Departmental Member (Dept. of Mechanical Engineering)

Dr. L. Dong, Departmental Member (Dept. of Mechanical Engineering)

Dr. W.-S. Lu, Outside Member (Dept. of Electrical and Computer Engineering)

Dr. Kamal Gupta, External Examiner (School of Eng. Science, Simon Fraser U.)

Table of Contents

List of Figures

List of Tables

Nomenclature

Symbol	Description	Size[1]
α_s	sharing exponent	
C_f	crowding factor	
d	Euclidean distance between two points	
d^*	minimum distance	
d^o	initial distance	
$\mathbf{D}$	distance matrix	$v \times v$
δ	mesh distance	
Δ	distance between two trial points	
f	objective function	
$\mathbf{F}$	facets matrix	varies
G	iteration or generation number	
G_{max}	maximum number of iterations or generations	
Γ	region or niche	
k_B	Boltzmann constant	
k_p	penalty weight	
k_r	repair scaling	
l	geometry level	
m'	niche count	
n	node index	
$\mathbf{N}$	node matrix	$v \times 3$
N_{pop}	population size	
N_Γ	number of niches	
p_b	Boltzmann probability	
p_m	mutation probability	
p_x	crossover probability	
$\mathbf{p}$	point Cartesian coordinates	3×1
$\mathbf{p}'$	locally improved point	3×1
$\mathbf{P}$	predecessor matrix	$v \times v$

Symbol	Description	Size[1]
Φ^*	'exact' global minimum robustness	
Φ	global region robustness	
Π	connectivity matrix	$v \times v$
$\mathbf{q}$	constraint value	varies
q_v	violated constraint value	
$\mathbf{r}$	random move vector	3×1
r	size of random move ($r = \|\mathbf{r}\|$)	
$\mathbf{R}$	rotation matrix	3×3
$\mathbf{R}_i^j$	rotation matrix describing Σ_i in terms of Σ_j	3×3
ρ	randomly generate scalar ($\rho \in [0, 1]$)	
$\boldsymbol{\rho}$	randomly generate direction	3×1
S	sharing function	
σ_s	sharing radius	
σ_m	mating radius	
σ_n	nish radius	
Σ	Cartesian coordinate frame	
Σ_0	inertial coordinate frame	
t	time or simulated annealing temperature	
t_{ini}	initial temperature	
$\mathbf{T}$	homogeneous transform	4×4
$\mathbf{T}_i^j$	homogeneous transform describing Σ_i in terms of Σ_j	4×4
v	total number of nodes in a mesh	
$x_e^*(s)$	best individual convergence (off-line performance)	
$x_e(s)$	population convergence (on-line performance)	
$\mathbf{x}$	trial point	6×1 or 2×1
$\mathbf{x}_{\text{off}}$	trial point offspring	6×1 or 2×1
$\mathbf{x}^*$	optimal trial point	6×1 or 2×1
$\mathbf{x}_o$	initial trial point	6×1 or 2×1
x, y, z	Cartesian coordinates of a physical point	

[1] If no size is specified the variable is a scalar.

Acknowledgements

There are many people that, over the years, have had influence in the path I have followed that I would like to thank.

First of all, I thank Professor Meyer Nahon for his support, constructive criticism and advice that in countless occasions extended beyond his supervisor's duties.

I want to extend my gratitude to my brother, Ricardo, that, through a number of discussions, played a very significant role in the development of this work.

Additionally, I also want to thank all my fellow graduate students, specially Brad Buckham for the numerous discussions in so many different topics that enriched my experience at UVic.

I would also like to thank Dr. Ou Ma from MD Robotics for his input in issues related to the Contact Dynamics Toolbox as well as some of his colleagues for some of their ideas on geometric modelling.

Likewise, the financial assistance from MD Robotics, CSA and NSERC through an NSERC Strategic grant are highly appreciated.

Along the same lines I would like to thank my Grandfather Anselmo for his support and guidance in all the stages of my education. He has always been a source of inspiration and I could only wish to achieve a portion of what he has.

Last but by no means least, I want to thank Mónica for all the time, love and dedication she selflessly spent taking care of the three of us. We made it my love.

To Mónica, Ana Sofía and Sebastián

Chapter 1

Introduction

Ever since the first industrial robotic manipulator was built in the 1960s, the application of such systems in new areas and environments has grown tremendously [Craig, 1986].

As robotic systems become more complicated, the need for tools for the design and simulation of such systems increases. This is particularly true for the design and simulation of robotic manipulators to be operated in hazardous environments such as outer space or underwater. The creation of test facilities that mimic these hazardous conditions is often expensive or infeasible. Thus, realistic computer simulations are essential for replicating such conditions.

Robot design and simulation are very extensive fields that can encompass areas ranging from kinematic design and controller design to task planning and manipulator maintenance. Some of these areas make use of distance determination algorithms in order to evaluate how close a manipulator is to other objects in its environment [Gill and Zomaya, 1998] or to obtain a contact force when the manipulator touches the environment [Ma, 1995].

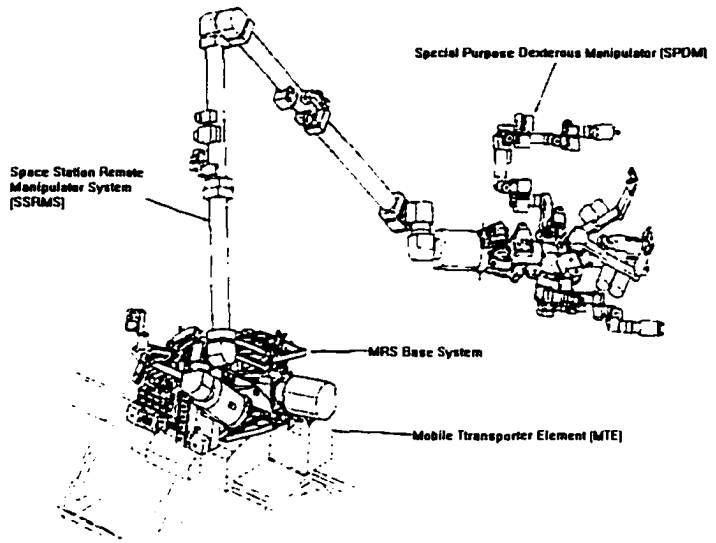


Figure 1.1. Mobile Servicing System (MSS)

The work presented in this thesis focuses on the determination of the distance between two arbitrary objects.

1.1 Motivation

Canada's contribution to the International Space Station (ISS) program consists of the Mobile Servicing System (MSS) (Figure 1.1). One of the components of the MSS is the Space Station Remote Manipulator System (SSRMS) [Doetsch and Middleton, 1987].

On earth, this complex manipulator cannot lift its own weight (its mass is approximately $1,800$ kg), thus a simulation facility is crucial during the development of such system. MacDonald Dettwiler Space and Advanced Robotics Ltd. has developed the MDSF (Manipulator Development and Simulation Facility) [Ma et al., 1997] for this purpose. A necessary component of MDSF is its contact dynamics module which enables MDSF to simulate tasks which include objects in contact with their

environment.

At present, the approach for dealing with concave objects in the contact dynamics module of MDSF is tedious and prone to error. This is primarily due to the distance determination algorithm used in the computation of the forces involved during contact. This motivated the interest to extend the simulation facility to be able to deal more easily with concave bodies.

1.2 Distance determination problems

Distance determination algorithms, *i.e.* obtaining the minimum distance between a pair of objects, is a problem that, over the past two decades, has caught the attention of many researchers. In general, minimum distance algorithms for a pair of objects return the point on one object that is the closest to the other object and *vice versa* (Figure 1.2 shows an example).

This section lists some of the current applications of distance determination and related algorithms. Also, a classification of these methods according to the type of problem they solve is proposed here.

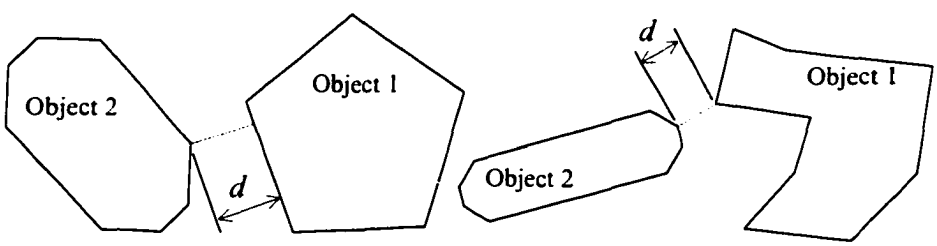


Figure 1.2. Examples of the minimum separation distance between convex and concave objects. Note that in the convex case the minimum distance is unique whereas in the concave case multiple solution may occur.

1.2.1 Applications

Probably the most popular use of the distance determination algorithms is in robot path planning [Liu and Mayne, 1990, Gill and Zomaya, 1998]. In this application, it is desired to plan a collision free trajectory of a robot through a workspace with obstacles. This problem is sometimes known as obstacle avoidance.

Simulation of physical systems and virtual experimentation [Hoffman and Hopcroft, 1987] is another engineering analysis application of the minimum distance problem. In many cases, the simulation of physical systems requires the use of some form of contact dynamics model where the separation or interference distance between the objects is required to [Ma, 1995]. One example of this is the simulation of a robot inserting a peg in a hole, where the need exists to obtain the distance between convex (*i.e.* the peg) and non-convex (*i.e.* the hole) objects when planning the trajectory and computing the contact forces (see Appendix A for an overview of contact dynamics).

Many CAD / CAM applications, such as assembly, tool path planning [Dong and Yuan, 1993], virtual reality [Ponamgi et al., 1997] and virtual prototyping [Hopcroft, 1988] require the use of distance determination algorithms in order to make the assemblies possible or the virtual models more realistic.

Some of the above applications require the minimum distance problem to be solved many times as the scenario changes in time, *e.g.* a robot moving a payload amongst objects. This is often referred to as dynamic distance calculation whereas its counterpart, where the distance is computed only once, is referred to as static distance calculation. In the case of contact dynamics, a detailed knowledge of the interference geometry (interference distance, interference volumes, *etc.*) is often desirable (see Appendix A).

In some cases only the fact that the objects are interfering or not is of interest and no detailed information is needed about separation or interference. This is the case in mechanical assembly.

1.2.2 Problem classification

The determination of the distance between two objects has been investigated by many authors in the last three decades. Each author has treated the problem in a different manner and thus many different methods for solving the distance problem exist.

The various distance problems that can be considered can be classified as follows:

- **Determination of contact**: This refers to methods that only return a boolean answer (*i.e.*, 1 (one) if the objects are interfering or 0 (zero) otherwise) but do not quantify the separation or interference distance.

- **Exact** or **Approximate methods**: Exact methods are those that determine distance (or interference information) in an exact manner (within computer precision). On the other hand, some methods sacrifice precision for speed. These will be referred to as approximate methods which are not designed to return an exact distance (or interference information) but rather an approximation.

- **Interference** or **Separation distance**: As will be described in the following sections, some methods have been proposed for the sole purpose of determining interference information. Most algorithms designed for separation do not determine interference information and *vice versa*.

- **Type of geometries**: Although most distance algorithms have been designed to handle linearly bound objects (*i.e.*, polyhedra), some are able to deal with

other types of geometries such as quadratic surfaces (*e.g.* cylinders and spheres).

- **Convex** or **Concave objects**: This category is of most relevance to the present work. It is this category that identifies methods that can handle concave objects or those that only deal with convex objects. Since convex objects are a subset of more general object such as concave objects, methods that can handle concave objects do handle convex objects as well.

1.3 Literature review

This section reviews prior works in the main areas covered in this document. First, distance determination methods are described. The methods are subdivided into three types (a) those intended to solve the distance problem between convex objects, and (b) those intended for concave objects and finally (c) those used for the determination of the interference distance.

As will be seen in the following chapters, the new methods proposed in this work for solving the concave minimum distance problem make the use of global optimization techniques. Thus, an overview of global optimization techniques is presented with particular attention to Genetic Algorithms and Simulated Annealing.

1.3.1 Distance determination

There exist basically two methods of obtaining the minimum distance between two objects: one is an analytical approach and the other is numerical. The first one is computationally very efficient when the objects involved are relatively simple but becomes quite complicated when the objects are more complex (see for instance [Boyse, 1979,

Chau, 1991]). On the other hand, the numerical solutions have proved to be reliable and accurate for a very wide range of objects (see for instance [Buchal et al., 1989, Gilbert et al., 1988, Hayward et al., 1991]). This section focuses on numerical methods for solving the minimum distance problem.

1.3.1.a Methods for convex bodies

Although distance determination algorithms are used in many different areas, they are found most often in the literature on robot path planning (see for instance [Gill and Zomaya, 1998]). In this application, the first step in the distance determination algorithm is to check if a collision has occurred or not [Bobrow, 1989]. Then, if no collision is detected, a more detailed minimum distance algorithm is used.

Some of the methods used for collision *detection* include using a bounding box (or a circumscribed sphere [Boyse, 1979, Hubbard, 1996]) around the object and aligned with the inertial axes. Then, all these boxes are checked for any overlap by individually checking each axis. If any pair of boxes overlap in all three directions then there exists interference between the two objects [Maruyama, 1972, Forrest, 1974, Boyse, 1979]. This is obviously a rough and conservative approximation but leads to computationally efficient algorithms. On the other hand, fast algorithms to compute the exact separation distance between boxes have been proposed, see for instance [Meyer, 1986].

A more precise collision detection for convex polyhedra is described in [Boyse, 1979] where each edge on object A is analyzed for interference with object B and *vice versa*. If the two endpoints of an edge of object A lie on the same side of a face of object B there is no intersection, whereas if the endpoints fall on different sides of the face, interference may occur. If interference is suspected, the intersection point of the line

and the plane containing the face has to be found, then a line starting at the intersection point and going to infinity (in any direction within the plane) is checked for how many times it crosses the face's edges. If the line crosses an even number of times or does not cross at all there is no interference but if it crosses an odd number of times the edge interferes, the two objects are said to interfere.

Most formulations are limited to linearly constrained objects (*i.e.* objects defined solely by flat surfaces) and only determine if there is interference between the objects or not but **do not** quantify the separation or interference distance. In [Bobrow, 1989] the minimum distance problem is formulated as a constrained optimization problem which quantifies the separation distance. In this method, the objective function for the optimization problem is the distance between two points, one in each object, and a set of inequality constraints correspond to the geometric specification of the objects[1]. This method has been used by other authors, see for instance [Liu and Mayne, 1990] and [Ma and Nahon, 1992], to solve problems such as path planning and contact dynamics. On the other hand, if the two objects interfere, this formulation will return zero distance, *i.e.* it does not provide any interference distance details.

Other distance methods use different geometrical approaches to solve the minimum separation distance problem between convex polyhedra. The method described in [Canny, 1987] and [Donald, 1984] divides the types of contact in two types: vertex-face contact (type A) and edge-edge contact (type B). To compute type A's minimum distance, the distance between each vertex of one body and each surface of the other body is computed and *vice versa*[2]. The minimum of all these computed distances is

[1]This method will be revisited in Chapter 6.

[2]The distance can be computed using a vector dot product between the normal vector of the surface (pointing outwards) and a vector between any point on that surface and the vertex being

recorded as the minimum distance for type A contact. On the other hand, type B contact is computed between all possible pairs of edges and the minimum between all of them is returned as the minimum distance between edges[3]. Finally the minimum between the result of types A and B calculations is returned as the overall minimum distance. This method has the advantage of identifying which of all of the pairs of points and surface or edges are closest to each other. A downside for this type of methods is that its computational expense greatly increases for objects with large number of faces.

Other geometrical methods use *Voronoi regions*[4] to determine the closest features between two objects [Lin and Canny, 1991, Lin, 1993, Mirtich, 1998]. If a feature f_A of a convex polyhedron is inside the associated Voronoi region V_B of feature f_B of a second convex polyhedron and *vice versa*, features f_A and f_B are said to be the closest features of the convex polyhedra. This method, limited to linearly constrained or linearly approximated objects, has successfully been used in a large scale environments where closest features as well as the distance between them are constantly being

---

tested. This method returns positive distance when the objects are not interfering, zero when the vertex is on the surface and a negative distance otherwise.

[3]A vector between one end of an edge on object 1 and one end of an edge on object 2 is projected onto the vector cross product of the direction vector of the edges in question. If the results is equal to zero, the two objects touch at the edges being tested. Otherwise the result represents the distance between the edges. To identify interference from separation the sign of such cross product needs to be monitored. For example, when separation is known, from a previous time-step, and a sign change is detected amongst all the edge to edge distance calculations, the objects are then said to have type B contact.

[4]A Voronoi diagram is a collection of regions (*a.k.a.* Voronoi regions) that divide the space. Each Voronoi region corresponds to a particular site and all the points in a region are closer to the corresponding site than any other site [O'Rourke, 1993].

monitored [Cohen et al., 1995].

Using the *Minkowski subtraction*[5] to 'grow' one of the objects, some authors have successfully managed to reduce the problem to an iterative process where the distance between a point (the origin) and a convex object is obtained [Cameron and Culley, 1986] [Gilbert et al., 1988]. It is also described in [Gilbert et al., 1988] and [Gilbert and Foo, 1990] how the method can be extended to the spherical extensions[6] of linear object rendering the method capable of dealing with some quadratically constrained objects.

1.3.1.b Methods for concave bodies

An interference detection based on Boyse's scheme [Boyse, 1979] (which in turn is based on Maruyama's [Maruyama, 1972]) for convex polyhedra was extended to non-convex polyhedra in [Abdel-Malek and Burton, 1994], where every pair of surfaces is checked for interference. The line of intersection between pairs of planes (containing each of the faces in question) is first obtained. The line is divided in small segments that reach the faces' limits. If any segment is fully contained within the boundaries of both faces then the faces are said to interfere except when one of the surfaces is a hole. If the face has a hole (a concavity) it is modeled as a negative entity, thus if a line segment is contained within the boundary of that face there is no interference. Being an interference detection algorithm, this method only returns boolean type results.

Some works have reported the use of convex partitioning (see Figure 1.3) of concave bodies to solve the distance determination problem between concave bod-

[5] Minkowski sum (or subtraction) K of two sets of points defining polyhedra K_A and K_B refers to the *sum* defined as follows $K = K_A \pm K_B = \{x \pm y \mid x \in K_A, y \in K_B\}$ [O'Rourke, 1993]

[6] The spherical extension of a point is a sphere, that of a line is a cylinder with spherical caps, *etc.*

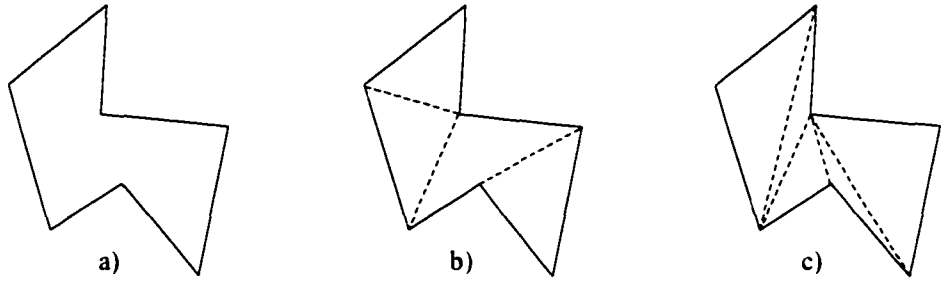


Figure 1.3. Convex partitioning of a concave body, a) shows the original object, b) and c) show two different convex partitions of the same object.

ies. These methods solve the minimum distance problem for every object pair and return the minimum of all the distances as the solution of the concave problem [Gilbert et al., 1988, Ma and Nahon, 1992, Quinlan, 1994]. That is, once the partition is performed, usually off-line, the convex distance problem is solved between all the sub-body pairs. The overall solution will then be obtained as the minimum distance between all pairs. This is the method used in MD Robotics' MDSF mentioned earlier. Although it is effective, partitioning the concave objects tends to be tedious and error prone.

The method described in [Lin and Canny, 1991] has successfully been extended to deal with non-convex polyhedra [Ponamgi et al., 1997]. This was achieved by monitoring the features of the convex hull[7] and using convex partitioning of objects together with a hierarchical interpretation of the geometry.

The partitioning methods have the advantage of relying on algorithms that have proven to be reliable for the convex problem. On the other hand, they have the disadvantage that the minimum distance problem has to be solved many times at each time step to get the global minimum solution for all possible pairs of convex

[7]The convex hull of a concave object A is the smallest convex polyhedron that fully contains object A.

bodies.

Looking to improve the computational efficiency, some works have concentrated on decomposition-free solutions. Such is the case of the collision detection algorithm for non-convex bodies described in [Thomas and Torras, 1994, Jiménez and Torras, 1995]. This method is similar in concept to the work by Canny and Donald [Canny, 1987, Donald, 1984] described above. In this case, pruning strategies are used to reduce the computational expense of computing the distance between all pairs. This method, being only a collision detection algorithm, does not quantify the separation or interference distance.

To the author's knowledge, there has been no published work reporting on the solution of the minimum distance problem for concave bodies using a decomposition-free approach. The main advantages of a decomposition-free approach would be to minimize the number of object pairs to be tested and to eliminate the extra features (surfaces, edges, vertices, *etc.*) usually generated while decomposing concave objects.

1.3.1.c Interference distance

In some applications, such as contact dynamics (see Appendix A for details), the interference distance is required. Since there is no single solution of the interference distance (Figure 1.4a), all approaches to the interference distance problem are approximate. Additionally, all methods reported to date are limited to convex objects with the majority of them only capable of dealing with convex polyhedra.

Some works, such as [Gilbert et al., 1988], have reported the use of the same algorithm used for separation distance when dealing with interference distance. Since the original method reported in [Gilbert et al., 1988] deals only with convex objects, its interference counterpart only deals with convex polyhedra as well. Other methods,

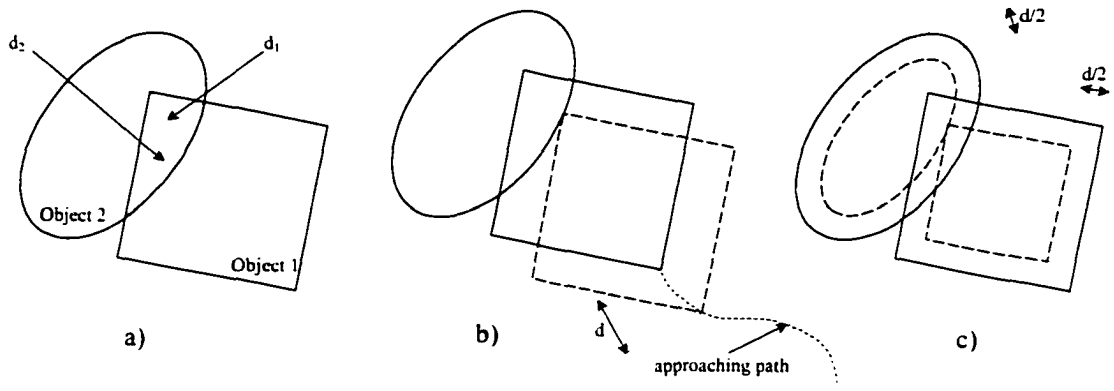


Figure 1.4. Interference distance examples: a) non-uniqueness of the interference distance for two simple objects where $d_2 > d_1$, b) 'move-back' approach and c) 'shrink-body' approach.

not limited to polyhedra, have been proposed. These methods include (*i*) 'move-back' approaches [Buckley and Leifer, 1985, Liu and Mayne, 1990, Sridharan and Keerthi, 2001], where the colliding objects are moved back along the approaching path until they touch at a single point (Figure 1.4b), and (*ii*) the 'shrink-body' approach [Liu and Mayne, 1990, Sharf and Nahon, 1995], where the two bodies are shrunk until the bodies have a single point of contact (Figure 1.4c).

1.3.2 Local vs. global optimization methods

As described earlier, the minimum distance problem can be formulated as an optimization problem. When this is the case, the objective is to find a set of points that will minimize the distance between two bodies while satisfying some constraints. If a concave body is to be represented, the inequality constraints describing the geometry represent a concave set and special care must be taken since the function to be minimized has the possibility of no longer being unimodal (single minimum). Thus, methods for global optimization need to be used to solve the problem.

If a multimodal function (*i.e.* one having more than one local minimum) is to be minimized (or maximized) using a gradient based optimization algorithm, the solution obtained may or may not be the global minimum. That is, when the start point for gradient based methods is close to a local optimum, the algorithm will quickly move towards the local minimum and eventually converge to that minimum[8]. In other words, the algorithms is trapped in the local minimum. This means that if a different start point is used a different solution could be found, Figure 1.5 illustrates this with a single variable multimodal function.

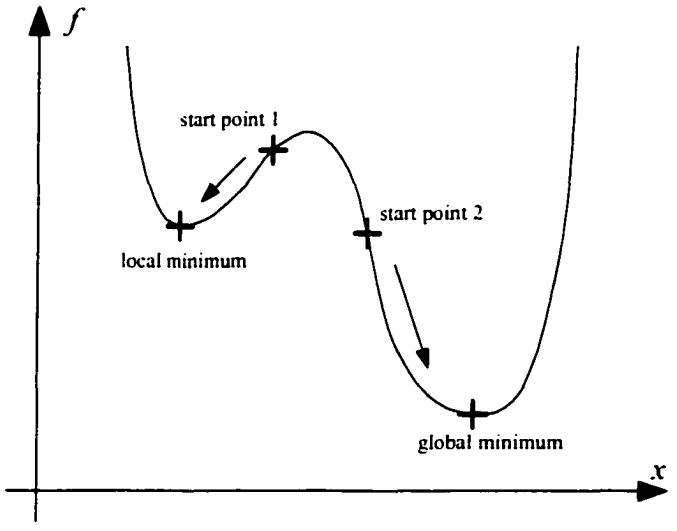


Figure 1.5. Local versus global optimum.

Often, a multi start search is used to bolster the confidence in the minimum value obtained during a local optimization search, see for instance [Carretero et al., 2000].

Other methods for global optimization include [Greenberg, 2000]:

[8] In this text the words optimimize and minimize will be used interchangeably since optimimizing a function consists of finding either its minimum or its maximum. Obtaining the minimum value of a function is the same as obtaining the maximum value of the negative of the function, *i.e.* $\min(f) \equiv \max(-f)$.

1. Ant colony optimization, inspired on the behaviour of real ant colonies [Dorigo and Di Caro, 1999];
2. Genetic algorithm, based on genetics and evolution;
3. Neural networks, based on how the brain functions;
4. Simulated annealing, based on thermodynamics;
5. Tabu search, based on memory-response [Pham and Karaboga, 2000];
6. Target analysis, based on learning.

From these methods, the ones that have not been designed for any particular application, and that have been applied in many different areas are Genetic Algorithms and Simulated Annealing. These two methods are also the ones that have been more thoroughly studied in the literature [Davis and Steenstrup, 1987]. A brief description of these two methods is presented in the next sections.

1.3.2.a Genetic Algorithm

A popular global optimization technique is Genetic Algorithms (GAs). These algorithms mimic the natural evolution processes by natural selection and 'survival of the fittest' methods. These principles were first outlined by Charles Darwin in the middle of the 19^{th} century [Darwin, 1859]. The first complete work published stating the basic principles of Genetic Algorithms was presented by Holland in 1975 [Holland, 1975]. Although little theory has been developed in the area of why GAs work so well in many problems, their use in different areas is growing quickly. As described in [Goldberg, 1989a], the most common application of GAs is in the areas

of function optimization [Wright, 1991], machine learning [Goldberg, 1989a], robot path planning [Gill and Zomaya, 1998] and artificial intelligence [Negnevitsky, 2001] ([Gen and Cheng, 1997] includes a survey of some of the most recent applications). A good introduction to GAs is given in [Beasley et al., 1993a], [Beasley et al., 1993b] and [Whitley, 1994].

The main steps in the original GA described by Holland (often referred to as the *canonical genetic algorithm*) are described as follows:

1. *Initialization*: Randomly create an initial population of size N_{pop}
2. *Evaluation*: Compute the fitness of all individuals in the population
3. *Selection*: Based on their fitness, select individuals that will be used for mating
4. *Mating* (*crossover*): Recombine population to produce N_{pop} offsprings
5. *Mutation*: Randomly mutate the population
6. Repeat steps 2 through 6 until a maximum number of generations (N_{gen}) is reached or convergence is detected.

Most of the research on this area involves the development of new and more efficient mating, mutation and selection methods.

1.3.2.b Simulated Annealing

Simulated Annealing (SA) is another global optimization method which is based on the analogy of the cooling process of molten metals [Kirkpatrick et al., 1983]. Briefly, this method works on the principle of randomly generating new points that are accepted if the objective function is improved. On the other hand, if the objective

function is not improved, the new point is only accepted if it meets a criterion which becomes progressively more difficult to satisfy as the iteration number increases. This randomness is included to allow the search to escape from local minima. This method has proven to be powerful in several applications, see for instance [Ullah and Kota, 1996, Martinez-Alfaro et al., 1998, Buchal et al., 1989].

The basic algorithm is described in point form as follows [Kirkpatrick et al., 1983, Rutenbar, 1989]:

1. Randomly create an initial guess
2. Evaluate the objective function at the initial guess
3. Randomly generate a new guess and evaluate the objective function
4. If the objective function at the new guess is improved, increase the iteration number (in SA parlance: decrease the temperature) and go to step 3 unless the algorithm has converged or the maximum iteration number is exceeded, otherwise continue to step 5.
5. Generate a normally distributed random number ($0 \leqslant \rho \leqslant 1$) and compare it to a Boltzmann probability function[9]. If the random number is smaller than the probability factor (*i.e.*, $\rho < p_b$) accept the new point and return to step 3, otherwise return to step 3. This acceptance criterion is referred to as the *Metropolis criterion* [Aarts and Korst, 1989].

[9] A Boltzmann probability function is one that decreases as the iteration number i increases such as $p_b = e^{\frac{f(x_{i-1}) - f(x_i)}{k_B t}}$, k_B is Boltzmann constant and t refers to the temperature which decreases as the iteration number increases.

The most important aspect of the SA process is the *cooling schedule*: *i.e.* the expression defining the Boltzmann probability function (step number 5 described above). A highly exploratory algorithm has a very slow cooling schedule accepting points with poor objective function even after many iterations have passed. Once enough iterations have passed, only a few moves that worsen the objective function are accepted. By the end of the SA process, only moves that improve the objective function are accepted.

1.3.2.c Constrained optimization

As was mentioned earlier, the purpose of the optimization is to minimize the distance between the two points (one on each body). On the other hand, when dealing with constrained optimization problems, one must incorporate some type of strategy to deal with points that are not feasible (*i.e.* points that are outside the object's bounds). As described in [Michalewicz, 1995], there exist several methods of dealing with constrained optimization problems. The most popular are the rejection, repair and penalty strategies [Gen and Cheng, 1997] and are described as follows.

- **Rejection strategy**: If a trial point lies outside the constraints, *i.e.* violates any of the constraints, it is rejected. This method, *a.k.a.* the death penalty, tends to slow down the convergence process, particularly when the space is highly constrained.

- **Repair strategy**: If a point is found to violate one or more constraints it is repaired by moving it inside the feasible region. This method has proven to surpass other strategies [Liepins and Potter, 1991] but depends on the existence of repair procedures that will move an infeasible point to a feasible region. This

procedure might not be possible in all cases, and when it is, it often involves a large number of computations.

- **Penalty strategy**: This strategy is relatively simple to implement by adding a scaled value of the violated constraints to the objective function when evaluating the fitness of a particular point [Smith and Tate, 1992]. Since this method allows the search through infeasible regions of the search space, it has been shown to be very reliable with highly constrained problems [Glover and Greenberg, 1989].

The rejection strategy can be applied before the selection step of the GA by eliminating the infeasible points from the population. It could also be applied as soon as the point is generated (either in the mutation or the mating steps of the algorithm). The case for the repair strategy is similar, where the points could be repaired as soon as they are created.

On the other hand, in the SA algorithm, the points would be rejected or repaired after they are randomly created. If a rejection strategy is used, a new random point is generated until a feasible point is found.

Finally, the penalty procedure is applied during the evaluation of the fitness (objective) function where some penalty is added to the fitness if a point is not feasible.

1.3.2.d Niche formation in GAs

It is common in nature to have groups of individuals exploiting particular regions of the environment. The idea is that if two or more regions are good for a particular type of creatures, groups of individuals will populate each of these regions or niches. In order to maintain a proper balance, the number of individuals in each region would be proportional to the resources available to them (*e.g.* water, food, shelter, *etc.*).

As a result, when a group of creatures inhabits a single region for a long time, with little to no genetic mix between individuals of other niches, they tend to develop their own genetic trends. This separation is called speciation [Goldberg, 1989a]. Each of these species will then occupy and exploit a particular niche.

Niche formation not only allows several minima to be found, but, by allowing greater population diversity, it increases the chances of the GA of finding the global minimum. Additionally, niche formation in GAs has proven to be a good method to keep track of evolving minima in time varying functions (see for instance [Grunwald, 1999]).

1.3.2.e Hybrid methods (Global search + Local search)

As discussed earlier, global stochastic optimization techniques such as GA and SA can take a long time to converge to the exact (within machine tolerance) solution. For this reason, local optimization techniques are often used within the global search in order to allow the global algorithm to find a better solution (at least locally). Hybrid methods refer to the use of a combination of global optimization techniques (*e.g.* Genetic Algorithm) combined with local optimization techniques (see for instance [Renders and Bersini, 1994, Houck et al., 1996]).

Given a starting point, general local optimization methods (such as the derivatives or Quasi-Newton methods [Bertsimas and Tsitsiklis, 1997, Gill et al., 1981]) are used to find the local minimum in the region of the start point. On the other hand, as will be seen in later chapters, problem specific local optimization techniques would allow a local improvement on the solutions while at the same time lessening the computational expense of using a more general method.

In this section, two hybrid methods are described. First, a parallel approach

is outlined where each point is locally optimized at every iteration of the global optimization method. Next, a sequential approach is described where the local optimization is only run once at the end of the global search.

Notice that, even though these hybrid methods help the global algorithm to get closer to an exact solution, they still do not guarantee that the solution will be global.

Parallel approach Global numerical optimization techniques are iterative by nature. Some authors have proposed to locally optimize every trial point within the global optimization (*i.e.*, the GA or SA) at every iteration [Gen and Cheng, 1997]. This is referred here as a parallel approach since the two optimization algorithms, the global and the local, are run simultaneously.

Genetic algorithms that make use of local optimization techniques are referred to as Memetic Algorithms which evolve using what in GA parlance is referred to as Lamarckian evolution[10] (see for instance [Gen and Cheng, 1997, Whitley et al., 1994]). In GA practice, the best or all individuals of each generation are used as starting points in a local optimization search. The resulting individuals are then used as the new generation that will eventually be mated and mutated. This approach has proven to converge much faster to a global solution than the global methods on their own [Gen and Cheng, 1997].

Also in GAs, a slightly different approach to the use of local optimization techniques is the use of the Baldwin effect [Whitley et al., 1994]. This is a method used in hybrid genetic algorithms which is similar to the Lamarckian method. The difference

[10] Jean-Baptiste Lamarck (1744-1829) was a french zoologist whom challenged Darwin's theory of evolution with his theory of heredity known as the "inheritance of acquired traits". Lamarck's theory was based upon the claim that individuals could inherit knowledge and experience.

between these two methods is that, in GAs exploiting the Baldwin effect, only the individual's fitness value is affected by the local optimization without changing the individual's genetic material to the new local optimum as in the Lamarckian algorithms (see for instance [Whitley et al., 1994, Houck et al., 1996]). That is, GAs exploiting the Baldwin effect work on the premise that individuals located in promising regions will be given preference during the selection process since they have a higher chance of evolving into a better point than individuals located in potentially poor regions.

Sequential approach The sequential approach can be used by any of the global optimization methods mentioned earlier. This method simply uses the solution of the global optimization method as the starting point for a local optimization. This is particularly useful since the global optimization methods usually take a long time to converge to the exact solution. Thus, by using the 'best' point of the global search method as the starting point of the local search, the convergence to the exact minimum is accelerated.

This method might prove faster than its parallel counterpart since the local optimization only runs once, thus reducing the computational expense of each iteration of the global search.

1.4 Thesis overview

The objective of the present work is to develop new methods for solving the distance problem between convex and concave objects. Two new methods for determining the separation distance between two objects are proposed in this thesis. Both methods are formulated as global optimization problems. Since the objective function in

both formulations is multimodal (*i.e.*, it has more than a single minimum), global optimization techniques such as Genetic Algorithms and Simulated Annealing are used. That is, each one of the two approaches can be solved using either GA or SA[11].

The first method, described in Chapter 2 and referred to as the *continuous approach*, formulates the minimum distance problem as a constrained optimization problem where the objects are defined by a series of inequality constraints. In order to solve the concave problem, global optimization methods (*e.g.*, GAs or SA) with constraint handling techniques such as penalty and repair strategies are proposed. These algorithms are also described in Chapter 2 where some details about the implementation are given. Additionally, Chapter 2 concludes with a few simple examples to illustrate the algorithms' capabilities.

Chapter 3 introduces the second formulation proposed in this work. This formulation, referred to as the *combinatorial approach*, avoids the computation of the constraints at run-time by creating a number of points on the surface of each object. The global optimization problem then becomes an unconstrained combinatorial problem where the combination of two points (one on the surface of each body) that minimizes the distance between them is sought. Details of the implementation of Genetic Algorithms and Simulated Annealing for this *combinatorial approach* are also presented in Chapter 3. Also, some simple examples of the minimum distance

[11] It is important to notice that, due to the stochastic nature of the GAs and SA, these methods do not guarantee to converge to the global optimum but they do cover a greater portion of the search space than the local methods. Additionally, as will be discussed in later chapters, there are many techniques that can be used in order to increase the chances that the global algorithms will find the best (global) solution.

problem are reported in Chapter 3 to illustrate the algorithms' capabilities.

Niche formation methods and their implementation into the combinatorial GA formulation of the minimum distance problem are described in Chapter 4. These methods allow the minimum distance algorithms to obtain multiple solution points permitting the distance algorithms to be applied in applications such as contact dynamics in complicated scenarios where multiple contact points are possible. Chapter 4 concludes with a couple of examples.

Next, in Chapter 5, the proposed minimum distance algorithms are evaluated. First, some performance measures are introduced followed by a brief description of all the control parameters of the distance algorithms. These parameters are then varied in order to search the best combination for the minimum distance problem. In Chapter 6 the best algorithm is used in a more complex example. There, the use of the minimum distance algorithms in dynamically changing environments is presented. Also in Chapter 6, the algorithms proposed in this thesis are compared to a method using convex partitioning to handle concave problems. Finally, some conclusions are presented in Chapter 7 and possibilities for future research are proposed.

1.5 Thesis contributions

The original contributions of this work can be partitioned into major and minor ones, as follows.

Major contributions:

- Developed a continuous approach for solving the minimum distance problem between convex and/or concave objects using constrained global optimization

techniques.

- Developed a discrete approach for solving the minimum distance problem between convex and/or concave objects using unconstrained global optimization techniques.
- Used Genetic Algorithms with niche formation to solve for multiple minimum distance points.

Other contributions:

- Created a fast local optimization technique that can be used for convex objects to find the minimum distance.
- Created a method (referred to as the *mesh mating*) that, in the context of combinatorial GAs search, mates two points belonging to a surface or volume mesh.
- Developed a method to identify the separate solution points amongst a large number of data points.
- Developed tools to automatically partition concave bodies into convex pieces.

Chapter 2

Continuous Approach

A *continuous approach* to solve the minimum distance problem using global optimization techniques is proposed here. This approach is designed to handle any combination of convex or concave objects. A penalty approach is used to incorporate the constraints that form the geometry of the object into the objective function of the global optimization algorithm. In order to do this, concepts from Constructive Solid Geometry (CSG) are used to deal with concave objects.

A local optimization routine is also proposed, in order to accelerate convergence of the global optimization algorithm. Additionally, when infeasible points are found, this local optimization technique acts as a repair strategy bringing the infeasible points back into the feasible region.

In this chapter, the basic concepts and operations needed to develop a global optimization routine for the minimum distance problem are described. These are followed by a detailed description of the implementations of the continuous approach using two different global optimization algorithms: Simulated Annealing and Genetic Algorithms.

Finally, a few simple examples are presented to illustrate the capabilities of the continuous approach to solve the distance problem for concave objects.

2.1 Formulation

The *continuous approach* described here is similar in concept to the one described by Bobrow [Bobrow, 1989] with the advantage of being able to handle concave objects. As described in Section 1.3.1.a, Bobrow's approach to the minimum distance problem is formulated as an optimization problem. That is, the optimization problem is formulated as follows:

$$\underset{\mathbf{p}_1,\mathbf{p}_2}{\text{minimize}} \quad : d^2 = (\mathbf{p}_1 - \mathbf{p}_2)^T(\mathbf{p}_1 - \mathbf{p}_2) \tag{2.1}$$

$$\text{subject to} \quad : \mathbf{q}_1(\mathbf{p}_1) \geqslant \mathbf{0} \text{ and } \mathbf{q}_2(\mathbf{p}_2) \geqslant \mathbf{0} \tag{2.2}$$

where $\mathbf{p}_i$ corresponds to the Cartesian coordinates of a point for body i. The inequality constraints shown as $\mathbf{q}_i(\mathbf{p}_i) \geqslant \mathbf{0}$ represent a set of half-spaces that fully define the geometry of body i [1]. Thus, for a point $\mathbf{p}_i$ to be feasible (*i.e.*, to be inside the body) it would have to satisfy all the constraints that define body i.

The minimization techniques used by Bobrow [Bobrow, 1989] and other authors (see for instance [Ma and Nahon, 1992]) to solve the optimization problem described in equations (2.1) and (2.2) cannot handle multiple minima. Although this is not a problem if the objects are described by constraints such as equation (2.2), it poses a limitation when the objects are concave since multiple minima can occur when one or

[1] In the particular case where the objects are bound by flat surfaces, *i.e.* the object is a polyhedron, the constraints become half spaces defined as: $\mathbf{A}_i\mathbf{p}_i - \mathbf{b}_i \geqslant \mathbf{0}$, where $\mathbf{A}_i$ is a matrix and $\mathbf{b}_i$ is a vector.

both bodies are concave. Thus, the method proposed here uses global optimization techniques in order to find the global minimum. In this work, the use of Genetic Algorithms (GAs) and Simulated Annealing (SA) to solve this problem is investigated.

Additionally, the constraints given by equation (2.2) can only represent a convex set. This then motivates a need to use a different representation method.

As described in Chapter 1, constraints can be dealt with in a few different ways when using global optimization techniques. In the formulation described in this section, the use of penalty and repair strategies is suggested since they allow the solution points to 'travel' through infeasible regions. This can potentially accelerate the convergence of the global optimization particularly when the infeasible portion of the space is large.

2.1.1 Representing a concave body

Equations (2.1) and (2.2) define the constrained optimization problem where the constraints define the geometry of objects 1 and 2. These constraints are typically expressed in terms of a body fixed frame, and thus are independent of the object's position and orientation.

Unfortunately, when the constraints are linear, this simple representation using a union of inequality constraints can only define a convex object. Thus if the same type of representation is to be used when dealing with concave bodies, special considerations regarding the constraints have to be made.

In order to have a complete representation of an object with concavities using inequality constraints, this work proposes to use certain concepts of constructive solid geometry (CSG).

First, the object's convex hull is obtained [O'Rourke, 1993]. This is the smallest convex object that fully contains the original object. Second, the original concave object is subtracted from its convex hull thus obtaining a model of the concavities. The concavities then become 'negative bodies', *i.e.*, the sign of the inequality constraints represented in equation (2.2) must be flipped. In other words, in order for a point to be feasible (*i.e.*, one that satisfies the constraints) it has to be inside the convex hull of the original object *and* outside the concavities.

For more complicated objects, this procedure can be nested, *i.e.*, if the negative body of the first concavity is itself concave, then its convex hull would replace the original negative body, and its concavity would become positive and so on.

To illustrate the concept, a simple two dimensional example is given in Figure 2.1. If a point P is to be inside the concave body $\mathbf{A}$, the following two conditions must be satisfied: 1) P must be inside the convex hull of the original object (*i.e.*, object $\mathbf{B}$) and 2) must be outside the concavities[2] of the object (*i.e.*, objects $\mathbf{C}_1$ and $\mathbf{C}_2$). Since the concavity $\mathbf{C}_1$ is itself concave, it needs to be modelled as a concave body, *i.e.*, the convex hull of the concavity is obtained (negative object $\mathbf{D}$). Next, the original negative object $\mathbf{C}_1$ is subtracted from $\mathbf{D}$ yielding positive object $\mathbf{E}$. That is, checking if condition 2 is satisfied or not is done by first checking if the point is inside or outside the convex hull of the concavity. If P is outside all the concavities, it is automatically a feasible point inside the original object. On the other hand, if the point P is within the boundaries of the convex hull of the concavity $\mathbf{C}_1$ it also needs to be part of the solid part $\mathbf{E}$ inside the concavity $\mathbf{C}_1$ in order to belong to the

[2]For a point to be outside a concave subobject, it has to violate at least one of the constraints representing the subobject's geometry.

original body.

In CSG, all the geometry's information can be stored in a tree structure [Zeid, 1991] as illustrated in Figure 2.2 for the object **A** depicted in Figure 2.1. Notice that the convex objects do not need to be further decomposed in subobjects since they can be represented by a union of constraints as described in equation (2.2).

Figure 2.3 shows an alternate representation for the concave objects where only convex sub-objects are considered. This method makes the feasibility check simpler to implement than its tree counterpart. For this reason, the alternate representation method will be used in this work to construct concave objects.

In essence, to evaluate the feasibility of a point the process would perform checks by pairs of levels except when the original object is convex, in which case the object only has one level. The following lines describe the process in point form for objects that are concave (see Figure 2.4).

1. Set geometry level to 1, *i.e.*, $l = 1$.

2. Evaluate if the trial point is inside any of the positive objects in level l (positive object level). If it is, proceed to step 3, otherwise point is not feasible and stop.

3. If object level $l+1$ exists proceed to step 4, otherwise point is feasible and stop.

4. For all subobjects in level $l+1$ (negative object level), evaluate the constraints. If at least one constraint is violated on each subobject the point is feasible and the feasibility check is over. Otherwise, if the trial point is contained in one of the negative subobjects proceed to step 5.

5. Check if the negative subobject containing the trial point has any subobject under it. If it does proceed to step 6, otherwise the point is not feasible and

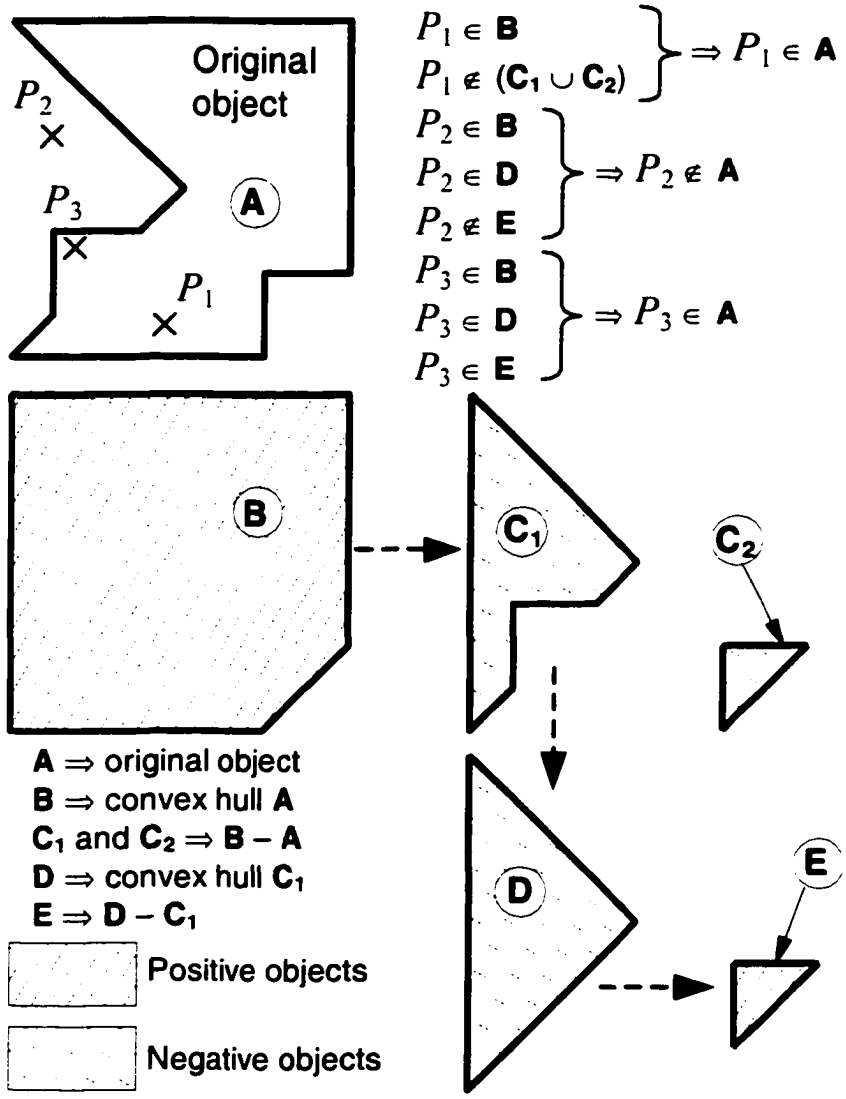


Figure 2.1. Illustration of the creation of a concave body using Constructive Solid Geometry.

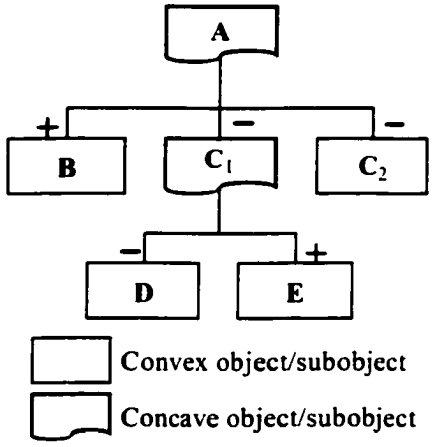


Figure 2.2. Tree structure representing the construction of the object **A** described in Figure 2.1.

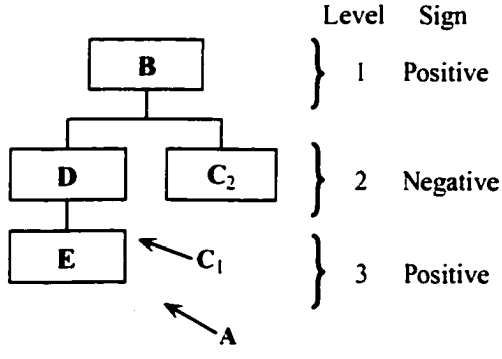


Figure 2.3. Alternative representation method for the construction of concave object **A** described in Figure 2.1.

stop.

6. Increase level number by two, *i.e.*, $l = l + 2$, and proceed to step 2.

2.1.2 Encoding of trial points

Optimization algorithms must encode the design variables of the problem in a way that the optimization algorithm can handle. In most cases, the encoding of trial points consists of a concatenation of the search variables into a single vector. In the continuous approach, the trial points are represented by a six dimensional row vector of the form

$$\mathbf{x} = [\ {}^1x_1 \quad {}^1y_1 \quad {}^1z_1 \quad {}^2x_2 \quad {}^2y_2 \quad {}^2z_2\ ] \tag{2.3}$$

where the first three and last three elements represent the Cartesian coordinates of a physical point in object 1 and object 2, respectively. These points coordinates are expressed with respect to their respective body fixed frame. That is, $\mathbf{x} = [\ {}^1\mathbf{p}_1^T \quad {}^2\mathbf{p}_2^T\ ]$ where ${}^1\mathbf{p}_1 = [\ {}^1x_1 \quad {}^1y_1 \quad {}^1z_1\ ]^T$ and ${}^2\mathbf{p}_2 = [\ {}^2x_2 \quad {}^2y_2 \quad {}^2z_2\ ]^T$.

Note that a trial point is a point in the six dimensional design space whereas a physical point is one in the three dimensional Cartesian space. Thus, the trial points as defined in equation (2.3) contain two physical points.

2.1.3 Objective function

As described earlier, the continuous formulation proposed in this chapter make use of global optimization algorithms to solve the minimum distance problem. Thus, it is necessary to define the objective function to be minimized.

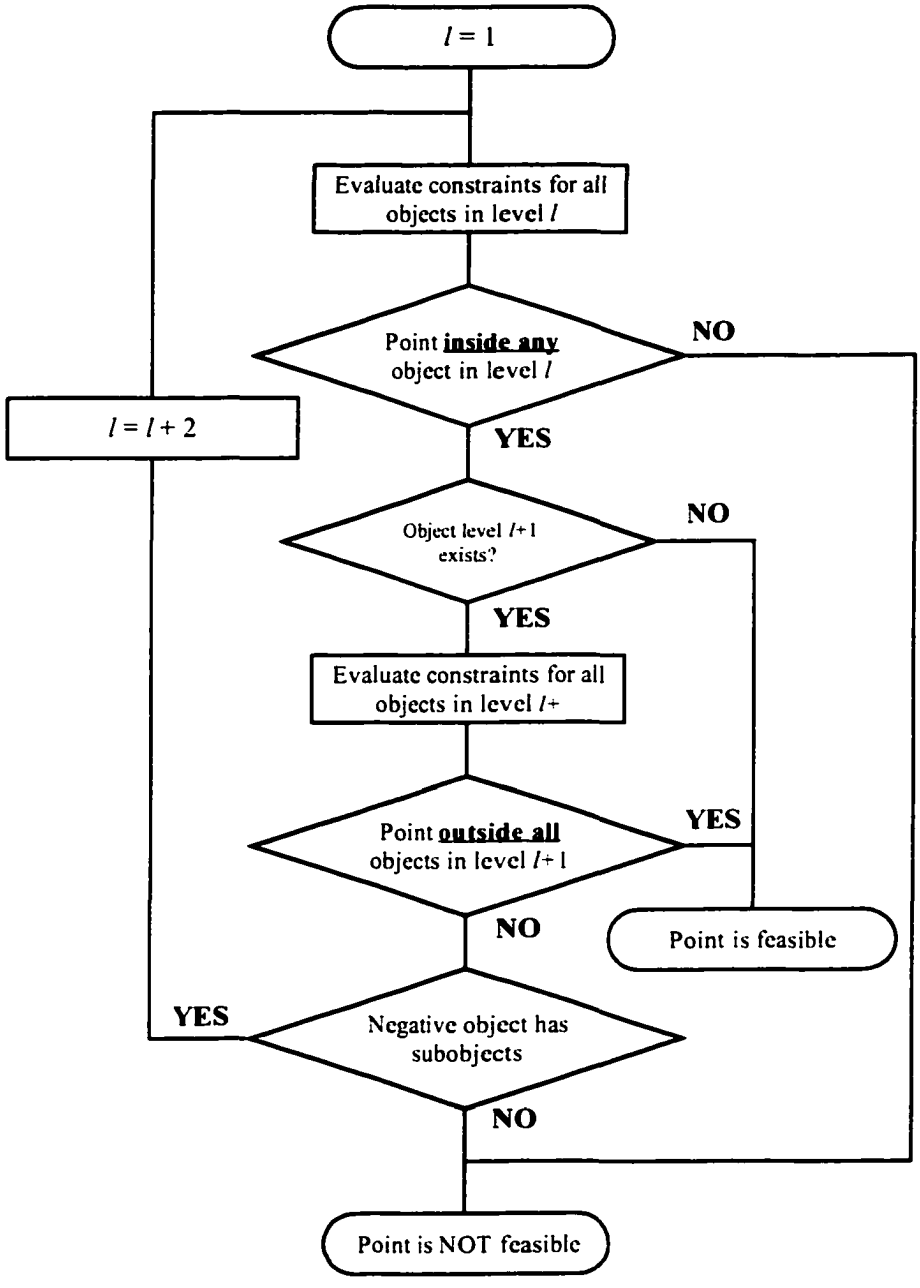


Figure 2.4. Flow diagram for the feasibility check of the alternative tree structure.

To incorporate the constraints into the global optimization algorithms a penalty strategy is proposed. This method was selected over the rejection strategy since it allows the optimization process to move through infeasible regions in order to reach better solution points [Gen and Cheng, 1997]. Thus, to integrate the penalty strategy into the optimization problem, the objective function to be minimized is modified by adding a second term to the square of the distance shown in equation (2.1). That is:

$$f = d^2 + k_p q_v \tag{2.4}$$

where d^2 is the square of the Cartesian distance between the two physical points inside trial point $\mathbf{x}$ defined as $d = [(\mathbf{p}_1 - \mathbf{p}_2)^T(\mathbf{p}_1 - \mathbf{p}_2)]^{\frac{1}{2}}$. Note that, in order to compute d in equation (2.4), points $\mathbf{p}_1$ and $\mathbf{p}_2$ need to be expressed with respect to a common frame. This is performed using the method described in Appendix B.

The second term in equation (2.4), *i.e.*, $k_p q_v$, corresponds to the penalty value which incorporates the constraints into the global optimization routine. Parameter k_p is a user-defined penalty factor and q_v corresponds to the penalty value calculated from the violated constraints.

The value of q_i (equation (2.2)) for all the constraints for a point $\mathbf{p}_j$ is calculated when the point's feasibility is evaluated. Thus, $q_i \geqslant 0$ when the constraint i is satisfied and $q_i < 0$ when it is violated[3]. Thus, one can express q_v as:

$$q_v = \begin{cases} 0 \text{ if } \forall\, q_i \geqslant 0 \\ \sum -q_i \text{ for } q_i < 0 \end{cases} \tag{2.5}$$

[3]When one constraint is active, *i.e.* $q_i = 0$, means the point lies on a face. Similarly, when two constraints are active, it means that the point is on an edge. Finally, three or more active constraints mean that the point is a vertex of the polyhedron.

The penalty factor k_p is scale dependant and has to be determined by experimentation. It should be noted that, if k_p is chosen to be too large, the penalty approach will actually act as a rejection strategy since the individuals that violate the constraints will have a much poorer fitness making it harder to compete against other individuals.

2.1.4 Computational complexity

It is important to notice that, every time a new trial point $\mathbf{x}$ is generated, its two components ($\mathbf{p}_1$ and $\mathbf{p}_2$) have to be checked for feasibility. A feasible point is one that is located inside or on the surface of an object. This entails evaluating all the constraints of both objects at that particular trial point.

For example, if one had two simple objects defined by 20 constraints each, the number of multiplications and additions (*i.e.*, flops) to verify feasibility would be around 240, since it takes 6 flops to check each constraint. By contrast, only 26 flops[4] are needed to evaluate the separation distance between points $\mathbf{p}_1$ and $\mathbf{p}_2$. Thus, the number of flops required to evaluate the distance d in this example is slightly more than 9 times smaller than the number required to check the point's feasibility.

2.1.5 Local optimization and repair strategy

In order to make local improvements to the solution points, a local optimization algorithm, similar to a line search [Gill et al., 1981], can be developed using part of the data calculated during the computation of the constraint equations. When

[4]Expressing the two points with respect to a common frame Σ_1 requires 18 flops and calculating the distance between them requires 8 flops for a total of 26 flops.

the feasibility of a point is calculated, all the constraints need to be evaluated for the current point. In the case when all the constraints are linear and they are all normalized (*i.e.*, the norm of the normal vector to the surface is equal to one), the value obtained when the constraints are evaluated represents the distance from the current point to the plane represented by the constraint. If the smallest constraint value from the evaluation of all the constraints for a particular point $\mathbf{p}_i$ in body i is represented by $q_{i_{\min}}$ it is safe to say that $\mathbf{p}_i$ could be moved a distance $q_{i_{\min}}$ in any direction and still remain within the bounds of the body i. With this in mind, one can move the points in body 1 and body 2 towards each other a distance $q_{1_{\min}}$ and $q_{2_{\min}}$, respectively, to produce points $\mathbf{p}'_1$ and $\mathbf{p}'_2$ (see Figure 2.6). This method, referred to as the *move closer* method, decreases the distance between the points. To obtain $q_{i_{\min}}$, the method described in Section 2.1.1 and illustrated in the flow diagram in Figure 2.4 is used to check the feasibility of the point. Consequently, the constraints are evaluated, *i.e.*, the values of q_i are obtained. Thus the distance from the trial point to the closest surface of the object $q_{i_{\min}}$ is recorded as the minimum of all q_i.

On the other hand, if $\mathbf{p}_i$ is outside object i (*e.g.*, $\mathbf{p}_i$ is in one of the concavities) the *move closer* algorithm could actually drive the point further from the surface of the object resulting on a poorer fitness value due to an increase in the penalty value. On the other hand, in these situations, the point is known to be outside the object's bounds (one or more constraints are violated), thus, a different strategy can be used. That is, if instead of moving the infeasible point *towards* its counterpart on the other body, it is moved $q_{i_{\min}}$ *away* from it (see Figure 2.7), the infeasible point could be brought back into the feasible region[5]. This enables situations like the one pictured

[5]Notice that $q_{i_{\min}}$ in this scenario represents the distance between the point and the furthest

in Figure 2.7 to be partially repaired.

Additionally, to make the *repair strategy* more efficient, the displacement distance $q_{i_{\min}}$ was scaled by a scalar k_r in the case when repairs were needed. If $k_r = 1$, the point $\mathbf{p}_i$ would never move inside the object since the distance $q_{i_{\min}}$ alone would, in the best case scenario[6], bring the point to the surface but not inside. Thus, the scaling with $k_r > 1$ is performed to accelerate the repair process. In this work, a value of $k_r = 1.5$ was used which, after a few trials, was found to repair most points within a couple of iterations.

Note that the repair strategy could actually drive points further into an infeasible region when the original point is on the 'dark side' of an object[7] (see set 2 in Figure 2.8). Although this 'false' repair decreases the efficiency of the repair algorithm, it can actually help the global optimization algorithms to eliminate poor trial points. That is, trial points located on the dark side of an object are not good candidate points for the minimum distance problem since a condition to be a favorable candidate is for the points to be on the 'visible side' of the objects. Thus, by moving these poor points further away from the object their objective function value gets poorer and the global optimization algorithm will tend to reject these inferior points. Alternatively, the fitness of the repaired trial point and the original trial point could be compared. If the repair causes the fitness to decrease (*i.e.* poorer fitness) the original trial point would replace the repaired trial point, otherwise, if the repair was successful, then

---

violated constraint.

[6]The best case scenario happens when the line that joins the trial points is parallel to the normal of the closest violated constraint (see set 1 in Figure 2.8)

[7]When dealing with a pair of objects, the dark side of object 1 is defined here as the side of the object 1 that is not visible from the geometrical centre of object 2.

accept the new point. This technique will be referred to as the *safe repair* technique and is the one used in the remaining of this work (see Figure 2.5).

2.1.6 Initial guess

Numerical optimization algorithms require an initial guess or start point for the optimization routine to start the search. In most cases a good initial guess accelerates the convergence of the optimization routine. If the distance determination problem is solved repeatedly over time it is possible to use the solution of the previous time step as the initial guess for the new time step [Nahon et al., 1998]. It is also possible to include velocity information to further improve the initial guess [Nahon, 1993].

On the other hand, during the first run of the minimum distance algorithm, *i.e.*, at the start of a simulation, the initial guess is not known. In this work, it is assumed that there is no previous knowledge of the solution and thus the initial guess has to be generated by the algorithm. In the present work, it was chosen to obtain the initial point(s) at random.

In the case of the continuous formulation, the points are generated inside the bounding box of the original objects. This can obviously generate both, feasible and infeasible start points. Even if infeasible start points are generated, when using local optimization, after a few iterations most points would eventually find their way inside the feasible region (*i.e.*, the object's geometry). In the worst case, *i.e.*, if the initial guess is not close the optimal solution, the time taken by the algorithm to converge to the global solution would be lengthened.

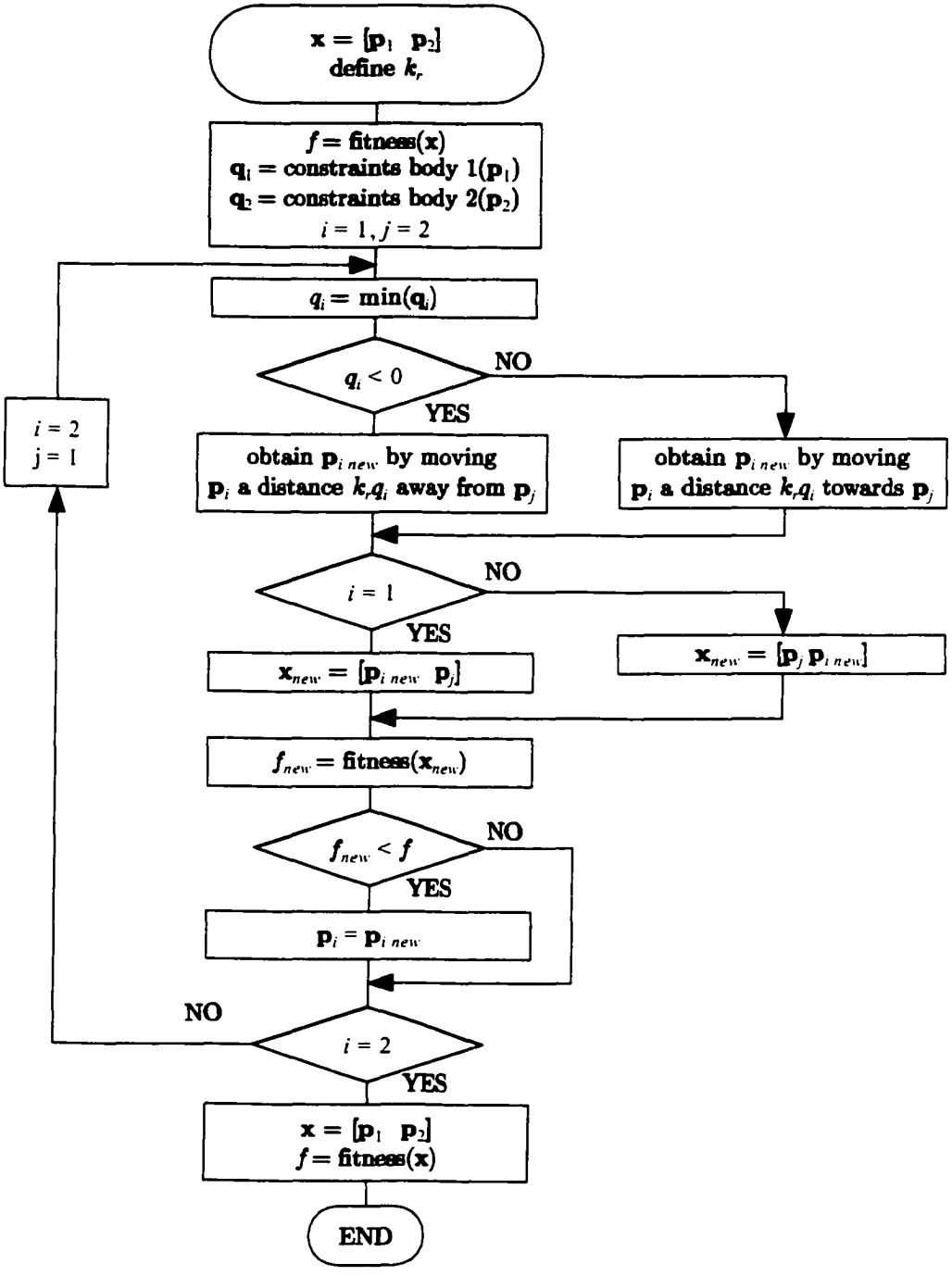


Figure 2.5. Flow diagram of the move closer and *safe repair* strategies.

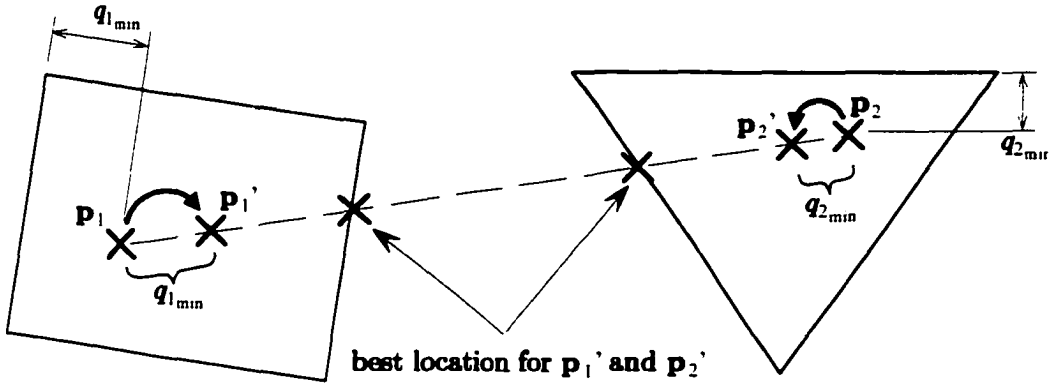


Figure 2.6. Example of the move closer method. Points $\mathbf{p}_1$ and $\mathbf{p}_2$ are moved *towards* each other.

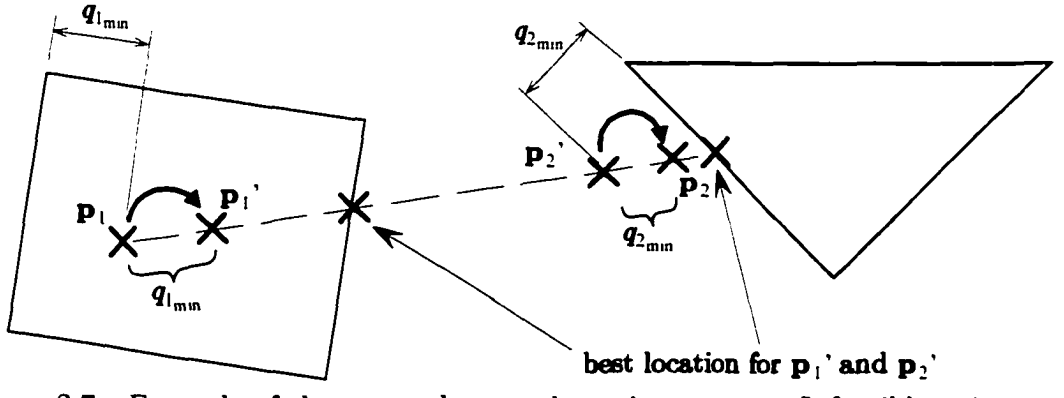


Figure 2.7. Example of the move closer and repair strategy. Infeasible point $\mathbf{p}_2$ is moved *away* from $\mathbf{p}_1$ whereas feasible point $\mathbf{p}_1$ is moved *towards* $\mathbf{p}_2$.

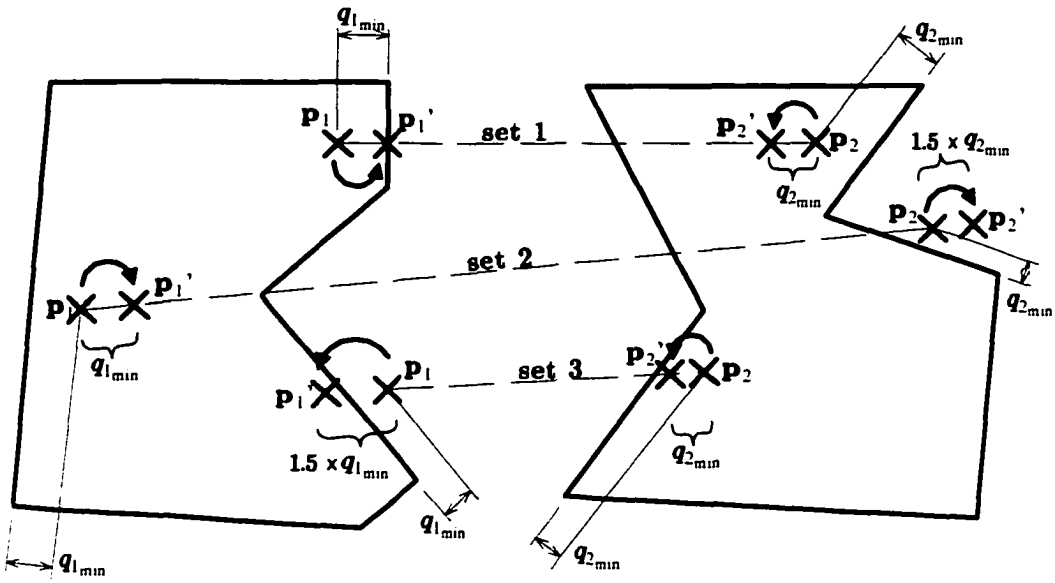


Figure 2.8. Example of the move closer method for three different sets of points. In set 1 both points are moved closer to each other, whereas in sets 2 and 3 points are repaired.

2.2 SA implementation

Although many examples of SA can be found in literature, this stochastic method has to be tailored to the particular problem in order to solve it more efficiently than with a generic implementation. This section describes the most relevant aspects of the algorithm implemented.

2.2.1 Structure

The Simulated Annealing algorithm implemented for the solution of the continuous formulation of the minimum distance problem is illustrated in Figure 2.9. The SA algorithm is initialized by setting the temperature t to its initial value t_{ini}. Also, an initial point ($\mathbf{x}_o$) is generated at random and its objective function (denoted here as

fitness) value is obtained (f_o). Then, the main SA cycle begins where by assigning the initial fitness to f_{i-1}, *i.e.*, $f_{i-1} = f_o$. Next, as the temperature t is lowered, the original point is randomly moved to a new location, *i.e.*, point $\mathbf{x}_i$ is generated. If the new point has a better fitness value (*i.e.*, $f_i < f_{i-1}$), it is accepted, f_{i-1} is updated (*i.e.*, $f_{i-1} = f_i$), and the cycle starts again. On the other hand, if the fitness function is not improved (*i.e.*, $f_i > f_{i-1}$), the point $\mathbf{x}_i$ is given a probability p_b of being accepted[8]. This probability decreases as the temperature is lowered (*i.e.*, the iterations pass) using a Boltzmann function. This process is also known as cooling schedule and will be described in more detail in the next section.

The SA algorithm continues this iterative process until the temperature has reached zero (*i.e.*, the maximum number of iterations is reached) or until convergence is detected. Note that in the algorithms used in this work and depicted in Figure 2.9, only the temperature is used as the termination condition.

Additionally, if a local optimization algorithm such as the *move closer* approach is used, that algorithm is run every time a new point is generated as illustrated in Figure 2.9.

Note that due to the stochastic nature of the SA algorithm, the last trial point found during an SA run is not necessarily the best one. For that reason, as a security measure, the trial point with the best fitness is recorded as the annealing process proceeds and is the one used as the final solution.

---

[8]This is referred to as a Metropolis criterion and the algorithm that checks for the criterion is consequently called a Metropolis algorithm.

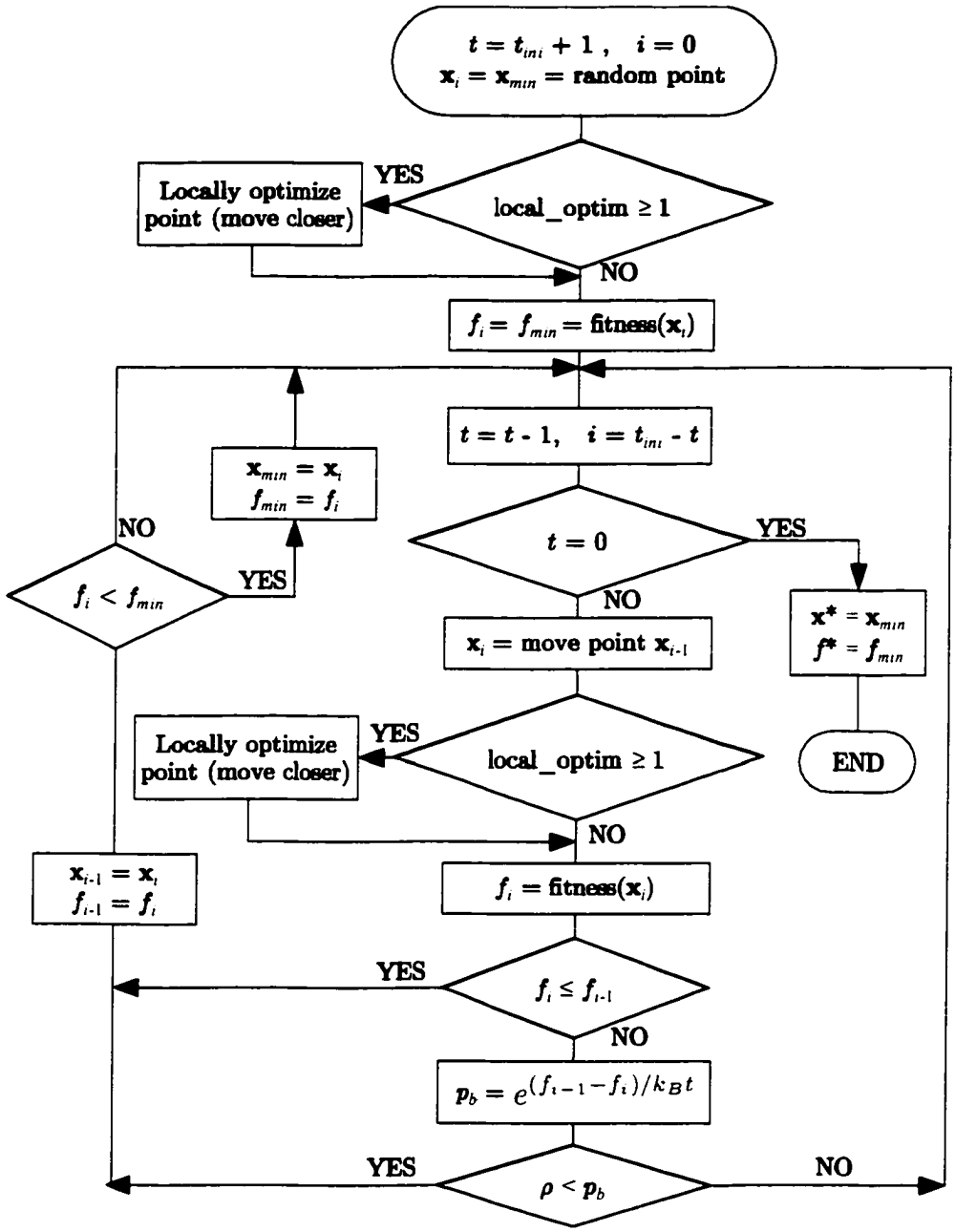


Figure 2.9. Flow diagram for the implemented Simulated Annealing.

2.2.2 Cooling schedule

A Boltzmann probability function expressed as [Aarts and Korst, 1989]

$$p_b = e^{\frac{f_{i-1}-f_i}{k_B t}} \tag{2.6}$$

was used, where k_B is the Boltzmann constant, t is the temperature and f_{i-1} and f_i are the values of the objective function at the preceding point and the current point ($\mathbf{x}_{i-1}$ and $\mathbf{x}_i$), respectively. The Boltzmann constant and the initial temperature t_{ini} are usually found by experimentation, allowing the SA process to run a few times and making sure there is no premature or excessively slow convergence.

The following example illustrates the variation of the Boltzmann probability factor (p_b) as the temperature t is decreased and the influence of k_B has on it. As indicated in Figure 2.9, only points that **do not** satisfy $f_i \leq f_{i-1}$ are put through the *Metropolis algorithm*, that is, accept the new point $\mathbf{x}_i$ if $\rho < p_b$ (where $\rho \in [0, 1]$ is an evenly distributed random number) otherwise reject it. This means that every time p_b is evaluated $f_{i-1} - f_i < 0$. Let us assume for a moment that the SA algorithm is not doing a good job at decreasing the objective function, that is, all points are passed to the Metropolis algorithm. Additionally, for a numerical example (see Figure 2.10), assume no constraints are violated (*i.e.*, $k_p q_v = 0$) and that the average difference between f_{i-1} and f_i is -5 units (average$(f_{i-1} - f_i) = -5$) with a random variation of up to 0.5 units (that is, $f_{i-1} - f_i$ is selected at random and $f_{i-1} - f_i \in [-5.5, -4.5]$). If the initial temperature is set to 1000, p_b would take the shape shown in Figure 2.10. Other, similar plots could be obtained if the initial temperature was varied.

This illustrates the importance of selecting a proper cooling schedule since both the initial temperature t_{ini} and the Boltzmann constant k_B, greatly influence the performance of the SA by accepting or not points during the annealing process.

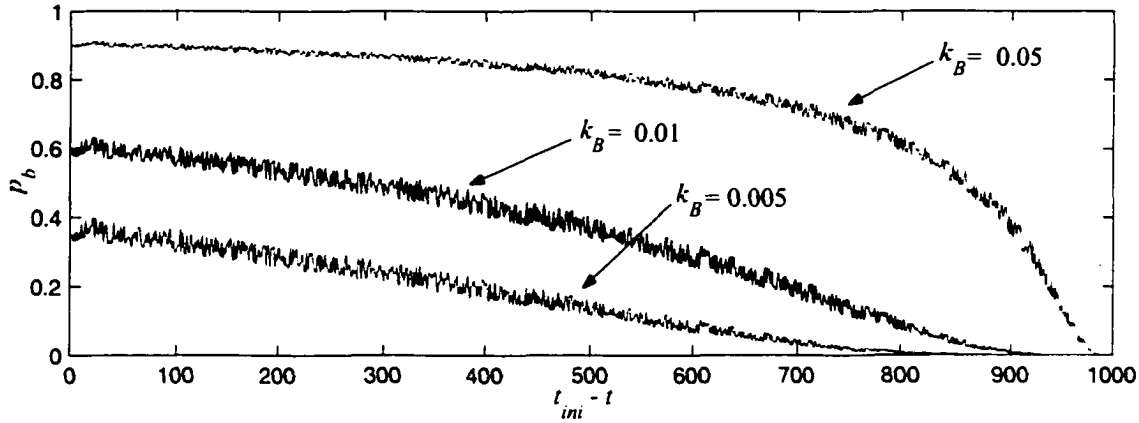


Figure 2.10. Example of the variation of Boltzmann probability factor (p_b) for three different values of k_B, with $t_{ini} = 1000$.

2.2.3 Generation of new trial points

In order to allow the SA algorithm to explore new regions in the solution space and to exploit favorable zones, methods for creating new trial points are necessary. Two different methods are presented here, one that moves points in directions parallel to the body fixed axes and a second one that moves points in a random direction.

2.2.3.a Move in Cartesian directions

This method starts from the initial point and alters one entry of the trial point while leaving the others unchanged. Every time the method is called on a given initial point, the entry to be modified on the trial point vector shown in equation (2.3) is selected with a probability p_m. Once an entry has been selected for modification, its new value is obtained as a random value within the possible range of values for that particular direction. That is, the new value is within the sides of the bounding box in the particular direction that the point is being altered. For instance, if entry 2 of a individual $\mathbf{x}$ is to be altered (*i.e.*, $\mathbf{x}_{[2]}$), the *Cartesian move* program uses the

bounding box of object 1 in the y range[9], and the new entry is calculated as:

$$\mathbf{x}_{[2]_{new}} = y_{1_{new}} = \mathbf{x}_{[2]_{old}} + \left[\rho \times (y_{1_{\max}} - y_{1_{\min}}) - (\mathbf{x}_{[2]_{old}} - y_{1_{\min}})\right] \tag{2.7}$$

that can be simplified to

$$\mathbf{x}_{[2]_{new}} = y_{1_{new}} = y_{1_{\min}} + \rho \times (y_{1_{\max}} - y_{1_{\min}}) \tag{2.8}$$

where $\rho \in [0, 1]$ is an evenly distributed random number and $y_{1\,\min}$ and $y_{1_{\max}}$ represent the sides of the bounding box of the object 1 in the y direction. Figure 2.11a shows an example of this *Cartesian move* method where only object 1 is considered. In that figure, the bounding box is represented by dotted lines and denotes the limits where the mutated points could lie. In Figure 2.11a, two examples of the altered point $\mathbf{p}_1$ are shown, namely $\mathbf{p}'_1$ and $\mathbf{p}''_1$, which have been altered in the x and y directions, respectively.

2.2.3.b Move in random direction

To allow the points to more freely, it is proposed to move the point that is to be displaced in a random direction instead of a single Cartesian direction. Additionally, the distance the point is displaced is selected at random while at the same time taking into account the size of the object. For this purpose, the *random direction* method proposed here starts by obtaining a random direction by obtaining a set of three random scalars in the range $[-1, 1]$ and arranging them as a vector, *i.e.*, $\boldsymbol{\rho} = [\rho_1, \rho_2, \rho_3]^T$. Then, the displacement vector $\mathbf{r}_i$ for object i is obtained as:

$$\mathbf{r}_i = r_i \frac{\boldsymbol{\rho}}{\|\boldsymbol{\rho}\|} \tag{2.9}$$

[9]Entries 1, 2 and 3 correspond to object 1 and entries 4, 5 and 6 correspond to object 2.

where $\|*\|$ represents the norm of $*$ and r_i is defined as the size factor[10]. It is proposed to use half the maximum object size as the size factor, that is:

$$r_i = \frac{1}{2}\left\| \begin{bmatrix} x_{i_{max}} \\ y_{i_{max}} \\ z_{i_{max}} \end{bmatrix} - \begin{bmatrix} x_{i_{min}} \\ y_{i_{min}} \\ z_{i_{min}} \end{bmatrix} \right\| \tag{2.10}$$

where the terms within brackets, *i.e.*, $[\ x_{i_{min}} \quad y_{i_{min}} \quad z_{i_{min}}\ ]^T$ and $[\ x_{i_{max}} \quad y_{i_{max}} \quad z_{i_{max}}\ ]^T$, correspond to the Cartesian coordinates of the two corners of the bounding box of object i. Note that scaling in equation (2.10) is performed since, on average, this would produce moves that keep the points inside the object.

Thus, the displaced point $\mathbf{p}'_i$ on object i can be obtained as:

$$\mathbf{p}'_i = \mathbf{p}_i + \rho \mathbf{r}_i \tag{2.11}$$

where ρ is a evenly distributed random scalar in the range from 0 to 1.

Figure 2.11b illustrates the point $\mathbf{p}_i$, on object i, that has been displaced in two different directions, $\mathbf{r}'_i$ and $\mathbf{r}''_i$, to obtain points $\mathbf{p}'_i$ and $\mathbf{p}''_i$, respectively. Since the alteration is performed with a maximum displacement radius of r_i in a random direction from $\mathbf{p}_i$, it is possible to determine the area or volume where this point can lie as shown in Figure 2.11b.

---

[10]The direction could also be obtained using two random values instead of three. This is done by generating two angles, α and β within the ranges $[-180, 180]$ and $[-90, 90]$ respectively. Thus, the point's displacement $\mathbf{r}_i$ could be calculated as: $\mathbf{r}_i = r_i \frac{1}{\sqrt{1+\sin^2 \beta}} \begin{bmatrix} \cos\alpha & \sin\alpha & \sin\beta \end{bmatrix}^T$. This method was not used since it was considered computationally more intensive than the method using three random numbers.

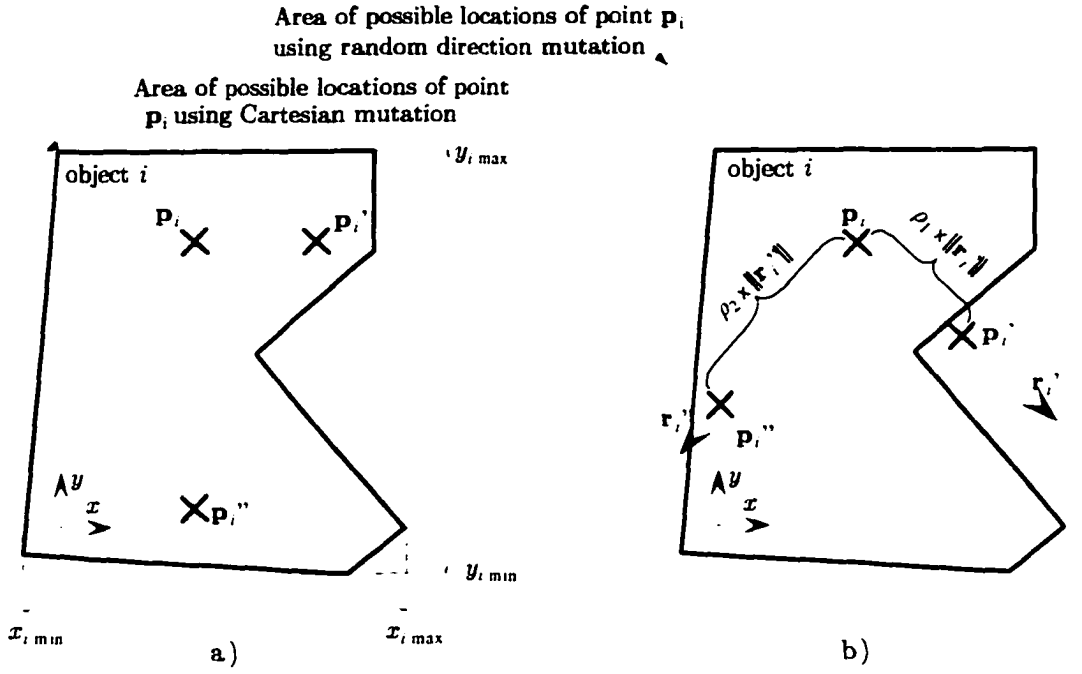


Figure 2.11. Two dimensional example of point alteration methods for the continuous approach: a) Cartesian direction method and b) random direction method.

2.2.3.c Variable displacement

In order to allow the SA algorithm to converge, the displacements described above can be scaled using a Boltzmann type of variation. That is, at the beginning of a run, points are capable of making large displacements, but as the temperature decreases (*i.e.*, the number of iterations increases), the motions get smaller.

To reduce the computational expense of generating a second Boltzmann factor, the Boltzmann probability factor used by the SA's cooling schedule can be used here. Thus, incorporating this factor into the Cartesian move methods involves modifying equation (2.7) as follows:

$$\mathbf{x}_{[2]} = y_{1_{new}} = \mathbf{x}_{[2]_{old}} + p_b \left[\rho \times (y_{1_{\max}} - y_{1_{\min}}) - (\mathbf{x}_{[2]_{old}} - y_{1_{\min}})\right] \tag{2.12}$$

where p_b is the Boltzmann probability factor obtained from equation (2.6).

Similarly, the random direction methods is modified by adding the factor p_b to equation (2.11) as follows:

$$\mathbf{p}_i' = \mathbf{p}_i + p_b \rho \mathbf{r}_i \tag{2.13}$$

2.3 GA implementation

2.3.1 Structure

A Genetic Algorithm (GA) was implemented using real number representation. Each individual in the population is represented by a vector (*a.k.a.* chromosome) with six entries (*a.k.a.* genes), see Figure 2.12. The first three entries correspond to the coordinates of a point in object one whereas the last three elements correspond to the coordinates of a point in object two. In both cases the points are represented in

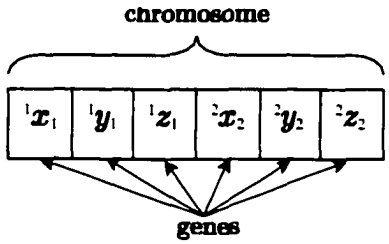


Figure 2.12. Graphical representation of the genes in a chromosome.

their respective body-fixed frame. This has been found to be computationally more efficient than having to convert the constraints (the bodies' geometry) to the inertial frame when evaluating the feasibility of any given point [Ma and Nahon, 1992].

Moreover, since the distance between two points is not dependent on the frame in which it is measured, the coordinates of the points in body two are expressed in terms of the coordinates of body frame one. This eliminates the computational expense of having to express points of body one in terms of the inertial frame. Clearly, the calculation of the homogeneous transform to transform coordinates from frame two to frame one only needs to be calculated once for any given position and orientation of the two objects.

The implemented algorithm is illustrated in Figure 2.13 and is outlined here:

Step 1: An initial population is generated randomly inside an axis aligned bounding box surrounding each object. The size of the original population is an even number denoted by N_{pop}. The whole population is stored in a matrix (the population matrix) where row i corresponds to the i-th individual of the population thus the size of the population matrix is $N_{pop} \times 6$. Note that, due to the mating process, N_{pop} needs to be an even number.

Step 2: The fitness of the population is evaluated using a penalty approach.

Step 3: Based on their fitness and using stochastic universal sampling (SUS)

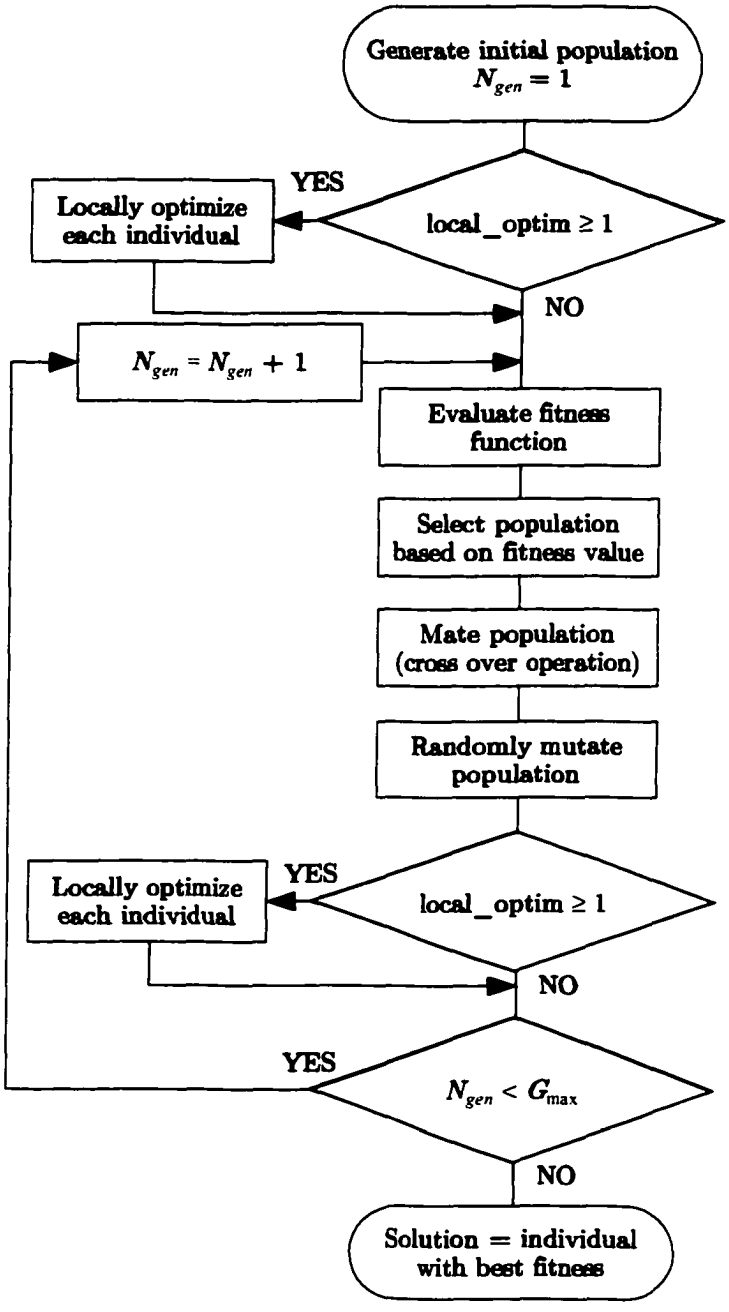


Figure 2.13. Flow diagram for the implemented Genetic Algorithm.

[Gen and Cheng, 1997], the population is selected for mating, producing a new population matrix also with N_{pop} rows.

Step 4: Individuals of the selected population go, by pairs, through a mating (crossover) operation with a probability of crossover of p_x producing two offspring per couple for a total of N_{pop} offspring. The crossover operation is performed by using linear scaling (*i.e.*, the offsprings are obtained by a linear combination of the coordinates of both parents). The offspring matrix produced by genetic replacement (*i.e.*, the offspring replace their parents) is $N_{pop} \times 6$ in size.

Step 5: With a probability p_m, the offspring are mutated using one of the *Move in Cartesian direction* or *Move in random direction* methods.

Step 6: If local optimization is used, it is at this point where it would be performed.

Steps 2 through 5 (or 6 if local optimization is used) are repeated for a fixed number of generations (G_{max}). At the end of the run the individual with the best fitness evaluated during the run is used as the solution.

2.3.2 Genetic operators

2.3.2.a Selection

Many selection processes have been proposed in the literature (see [Goldberg, 1989a, Davis, 1991] for a few examples). The simplest of them is the one called Roulette Selection (*a.k.a.* Stochastic Sampling or SS). In SS, all individuals in the population share a roulette wheel with the size of the slices directly proportional to the fitness of the individuals. Then, the roulette is spun a number of times equal to the population size (N_{pop}) and the individuals are selected for reproduction (see Figure 2.14a). Since

the slice in the roulette is proportional to the individual's fitness, the probability of a good individual with a large slice in the roulette being selected is greater than that for an individual with a poor fitness which would have a smaller section of the roulette.

Although roulette wheel selection has been used successfully in many applications, it has the problem that in an unlucky generation, one might get many poor individuals or that the best individuals are not selected for reproduction. Also of concern is the creation of *super individuals* with this method of selection. The term *super individual* refers to the creation of a population with many copies of a single individual which eliminates population diversity and thereby causes premature convergence of the algorithm.

To eliminate these problems, it is suggested to use Stochastic Universal Sampling (SUS) [Baker, 1987]. This method is a variation of the Roulette Wheel Selection in that the wheel is only spun once and the individuals are selected by N_{pop} equally spaced pointers where N_{pop} is equal to the population size (see Figure 2.14b). SUS allows a greater population diversity and guarantees that at least one copy of the best individual is passed on to the next generation. At the same time, SUS prevents the creation of *super individuals* [Gen and Cheng, 1997].

The selection method produces the selected population matrix with N_{pop} individuals that are to continue for mating. On the other hand, the couples to be used for mating need to be determined amongst the selected individuals. In the present case, the selected population is shuffled and the couples are made by selecting the fist two individuals, followed by the third and fourth and so on. It is for this reason that the population size (N_{pop}) has to be an even number.

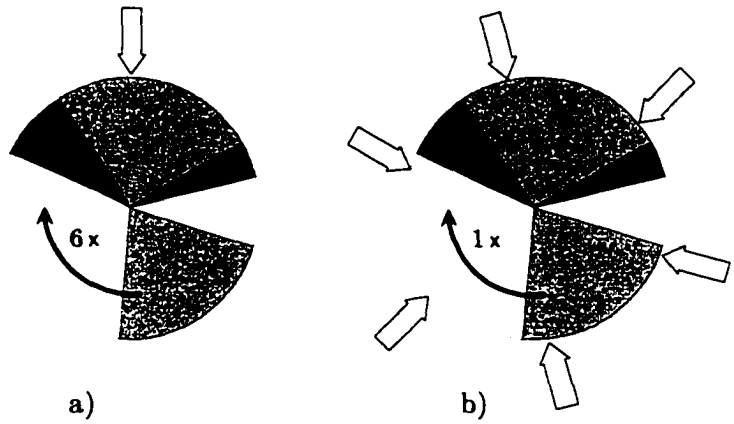


Figure 2.14. Example of roulette wheel selection for a population of 6 individuals using a) Stochastic Sampling and b) Stochastic Universal Sampling.

2.3.2.b Mating

In order to allow the GA to carry useful information to the next generation, the mating method should allow the algorithm to pass on the good traits of fit individuals to their respective offspring. In the minimum distance determination problem, an individual with good traits can be interpreted physically as a set of physical points lying on promising regions of their respective objects. That is, if individual $\mathbf{x}_1$ with fitness f_1 is to be mated to individual $\mathbf{x}_2$ with fitness f_2 with $f_1 > f_2$ (*i.e.*, according to the definition of f in equation (2.4), $\mathbf{x}_2$ is better than $\mathbf{x}_1$), one expects that the region around individual $\mathbf{x}_2$ to be more promising than the region where $\mathbf{x}_1$ is located.

To mate the selected couples, *linear random scaling* is proposed in which the coordinates of the two parents are combined and scaled in order to find an offspring on the line joining the two parents (see Figure 2.15). To explain this, it is possible to look only at one of the two objects, namely object i. If parents $\mathbf{p}_{i_1}$ and $\mathbf{p}_{i_2}$ are to be mated, all possible offspring $\mathbf{p}_{i_{off}}$ lie on the line that joins these two points. Thus,

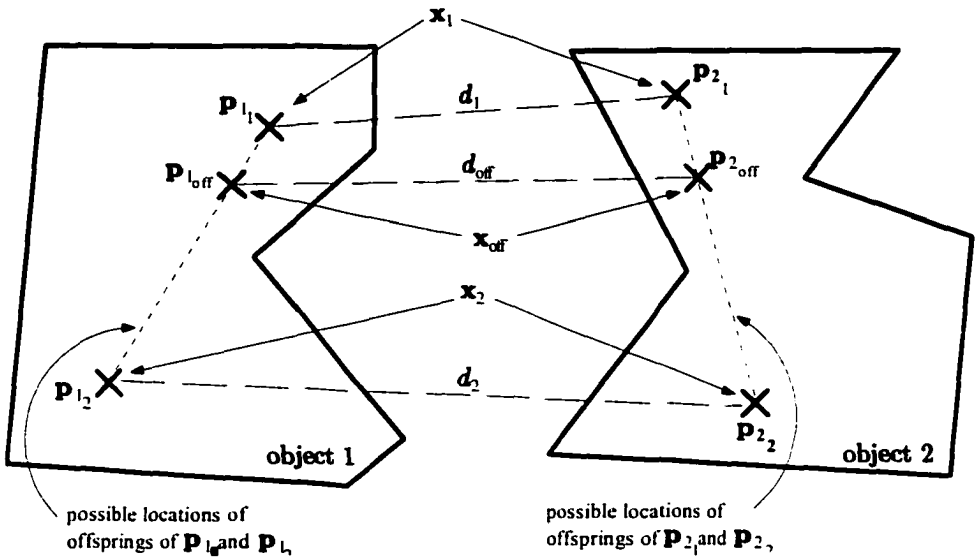


Figure 2.15. Example of the linear random scaling operation used at the mating operator.

if linear random scaling is used, the coordinates of point $\mathbf{p}_{i_{off}}$ can be found as:

$$\mathbf{p}_{i_{off}} = \mathbf{p}_{i_1} + \rho(\mathbf{p}_{i_2} - \mathbf{p}_{i_1}) \tag{2.14}$$

where ρ is a randomly distributed number in the range $[0, 1]$.

To further exploit promising regions, the mating could be weighted in order to force the offspring to be closer to the parent with better fitness (*i.e.*, smaller value of the fitness). To perform this weighting, the random generated number ρ would be replaced by a fitness dependent scaling factor as follows:

$$\mathbf{p}_{i_{off}} = \mathbf{p}_{i_1} + \left|\frac{f_1}{f_1 + f_2}\right| (\mathbf{p}_{i_2} - \mathbf{p}_{i_1}) \tag{2.15}$$

where f_j represents the fitness of point set j (*i.e.*, $\mathbf{x}_j$) to which point $\mathbf{p}_{i_j}$ belongs to since point set $\mathbf{x}_j = [\ \mathbf{p}_{1_j}^T \quad \mathbf{p}_{2_j}^T\ ]^T$.

2.3.2.c Mutation

In order to introduce population diversity and to allow the GA to explore unvisited regions, a mutation operation that 'moves' points randomly to new locations is necessary. The mutation methods proposed for the GA implementation work under the same principles as the methods described in Section 2.2.3 for the generation of new points in the SA implementation.

The main difference is that the mutation would not be performed to every individual. In fact, the mutation would only be performed with a probability of p_m (probability of mutation) which is determined by the user at the beginning of the simulation. Additionally, the distance by which points are mutated could be fixed instead of having a Boltzmann type of scaling (*i.e.*, mutation size exponentially decreases proportional to the iterations number).

2.3.3 Local optimization

Test were made where the location of the local optimization step within the GA was changed. After a few tests, it was observed that locally optimizing the individuals right before the selection process eliminates a lot of the population diversity. This makes the selection and mating processes less effective.

As discussed earlier, the elimination of population diversity creates problems of premature convergence which reduces the algorithm's ability to find the global optimum. On the other hand, if the local optimization is performed after the selection, the population diversity is better preserved. For this reason, in all algorithms described herein, the local optimization step is performed after the selection and mutation steps have been performed.

2.4 Examples

This section presents two sets of numerical examples where the distance determination algorithms described earlier in this chapter are applied to solve the minimum distance problem. Each set of examples uses a different set of objects. In both cases the algorithms are tested with the objects in two different configurations (*i.e.*, position and orientation with respect to each other). The two examples are presented in order of geometrical complexity from the simplest geometries to the most complicated ones. Note that, for comparison reasons, the same geometries and configurations will be used in subsequent chapters to demonstrate other algorithms and methods.

In order to evaluate the capabilities of the proposed algorithms under normal conditions, the following examples solve the minimum distance problem using randomly generated starting points. This allows us to evaluate the worst case scenario in terms of computational efficiency since no prior knowledge of the location of the minimum solution is assumed. Additionally, for every example presented here, the random number generator was reset before each test.

In order to classify the algorithms tried here, the following naming technique was adopted. First, the first three letters of the approach name are used (*i.e.*, 'cont' for continuous), second the optimization algorithm acronym is used (*i.e.*, GA or SA) and finally two letters identifying if local optimization is used or not are added (*i.e.*, 'wl' for with local optimization and 'nl' for no local optimization). This results in algorithm names such as: cont-GA-wl, which states that a genetic algorithm without local optimization using the continuous approach is used. This same naming technique will be used in the following chapters where different approaches are described.

The test results presented in the following examples are each reported in a table

listing the algorithm's name as described above, the minimum distance obtained by the algorithm and a column labeled as 'Region' that identifies whether the solution given by the algorithm is or is not located in the neighbourhood of the global solution. Also, the last row in each table lists the minimum distance between the two objects in question obtained analytically or through a CAD package (depending on the complexity of the objects).

Note that, in order to demonstrate the conceptual design of the algorithms, all the computer implementations presented herein were developed in Matlab. The implementation of these algorithms in a computer language such as Fortran or C would greatly improve their speed. In order to estimate the computational time for each test, an approximation of the number of computations (flops) required by the algorithm to perform a particular test are presented. Then, the computational time is estimated using tabulated performances for modern PCs [Dongarra, 2002].

2.4.1 Boxes with notches

The geometries used in this simple example are shown in Figure 2.16 for position 1 and in Figure 2.17 for position 2. Some of the distances between objects (obtained analytically) are shown, the minimum of all is highlighted in bold and was calculated to be 0.2984 units for position 1 and 0.0979 for position 2.

The objects shown in Figures 2.16 and 2.17 are extruded objects in the direction perpendicular to the page. Thus, all the minimum distance determination tests herein are three dimensional. Although the geometries in this first example look simple (18 faces in total), the problem of solving for the minimum distance between this pair of objects offers a few challenges. One of them is the fact that a conventional

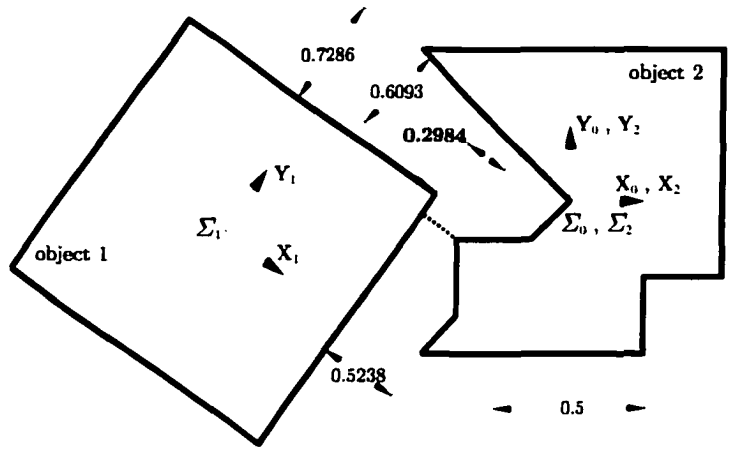


Figure 2.16. Geometry and configuration of the boxes with notches example (Geometry set 1 in position 1). Note that the minimum distance is shown in bold face ($d^* = 0.2984$).

optimization routine could easily get trapped in one of the multiple local minima. Second, since the objects are extruded, the actual minimum solution is not a single point but an entire line of possible solutions.

2.4.1.a GA trials

After a number of trials comparing the influence on the final solution of all the GA parameters, the parameter set listed in Table 2.1 was found to be a good balance between exploration and exploitation. That is, the algorithm allows some points to randomly move to new regions in order to investigate their quality, while at the same time allowing the existing points to exploit present regions for any possible global minimum. Additionally, Table 2.2 shows the genetic operators that were used while producing these examples. Although these operators and their variations will be studied thoroughly in Chapter 5, for this example they were fixed as described in Table 2.2 since they were found, after a few trials, to give reasonable results.

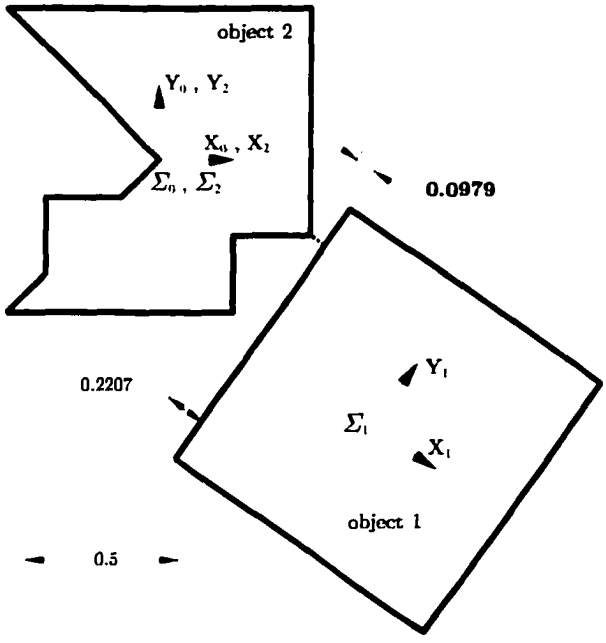


Figure 2.17. Geometry and configuration of the boxes with notches example (Geometry set 1 in position 2). Note that the minimum distance is shown in bold face ($d^* = 0.0979$).

Parameter	Symbol	Value
population size	N_{pop}	50
probability of mutation	p_m	0.02
probability of crossover	p_x	0.6
penalty weight	k_p	10
max. number of generations	G_{max}	100
repair scaling (safe repair)	k_r	1.5

Table 2.1. Parameters for cont-GA-wl and cont-GA-nl runs.

Operator	Type used in example
Selection	stochastic universal sampling
Mutation	random direction with variable displacement
Mating	fitness proportional
Repair	safe repair

Table 2.2. Genetic operators used with the GA runs for the continuous approach.

The initial population for all the runs was generated at random using the bounding box of each object. Figures 2.18 to 2.21 shows the results of the GA tests for both configurations. Figures 2.18 and 2.20 show the time history of the solution for the two positions considered here. Each of these figures contains two cases, one using the local optimization (cont-GA-wl) and one that does not (cont-GA-nl). The minimum distance obtained in all four runs as well as the exact solution obtained analytically is presented in Table 2.3 and is illustrated in Figures 2.19 and 2.21.

In all cases, it can be seen in Figures 2.18 and 2.20 that even though there is improvement of the quality of the solution after the 60^{th} generation, it is very small. This might imply that only 60 or 70 generations are needed to obtain a good result. However, in trials with several other positions of the two bodies, one hundred iterations appeared to be a good number since, in some other cases, the global minimum was not found until the 80^{th} iteration for the cases were cont-GA-wl was used.

In this first example, the GA run using the local optimization routine (*i.e.*, cont-

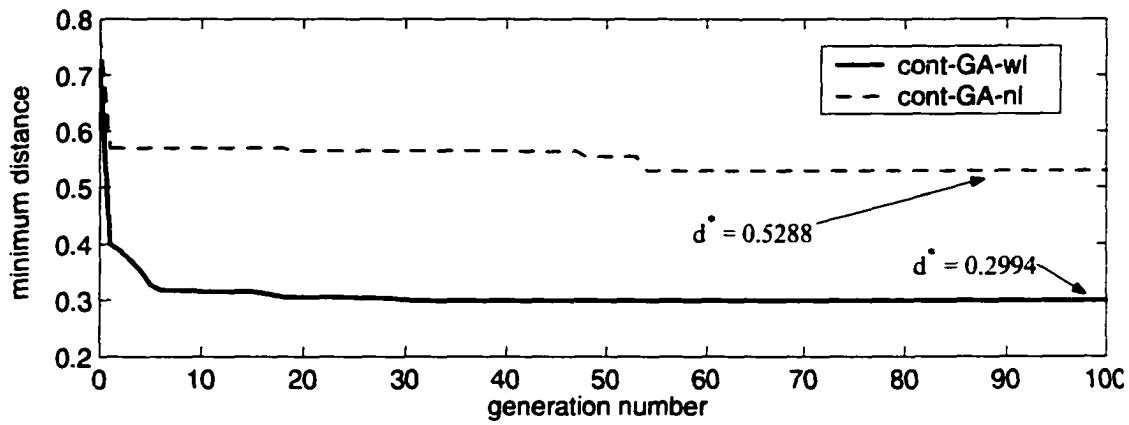


Figure 2.18. Time history of the best individual in the population for cont-GA-wl and cont-GA-nl (Geometry set 1 in position 1).

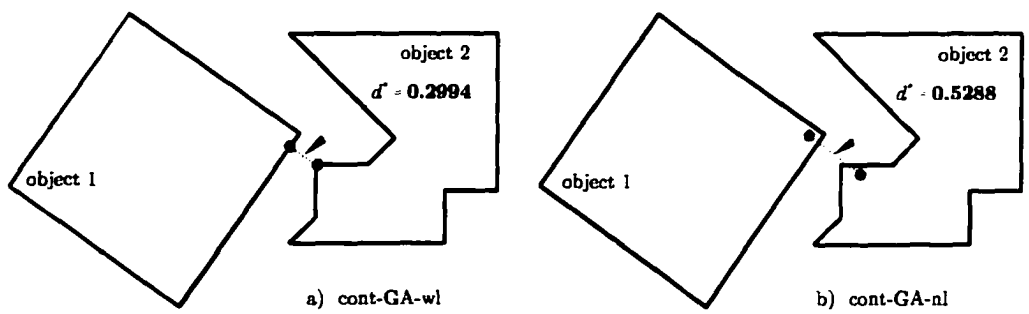


Figure 2.19. Results obtained by a) cont-GA-wl and b) cont-GA-nl (Geometry set 1 in position 1).

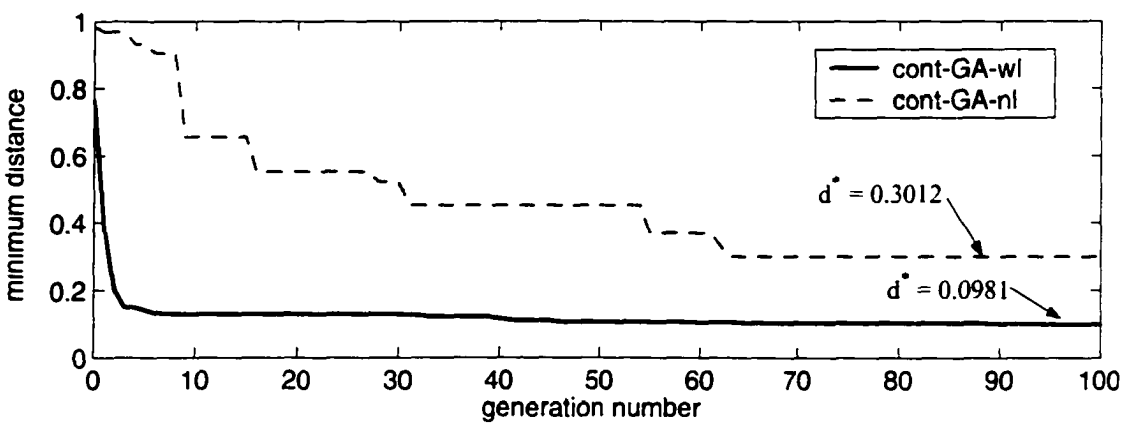


Figure 2.20. Time history of the best individual in the population for cont-GA-wl and cont-GA-nl (Geometry set 1 in position 2).

	Position 1	Region	Position 2	Region
cont-GA-wl	0.2994	✓	0.0981	✓
cont-GA-nl	0.5288	✓	0.3012	✓
exact	0.2984		0.0979	

Table 2.3. Minimum distance results for the example 1 in positions 1 and 2 for algorithms cont-GA-wl and cont-GA-nl.

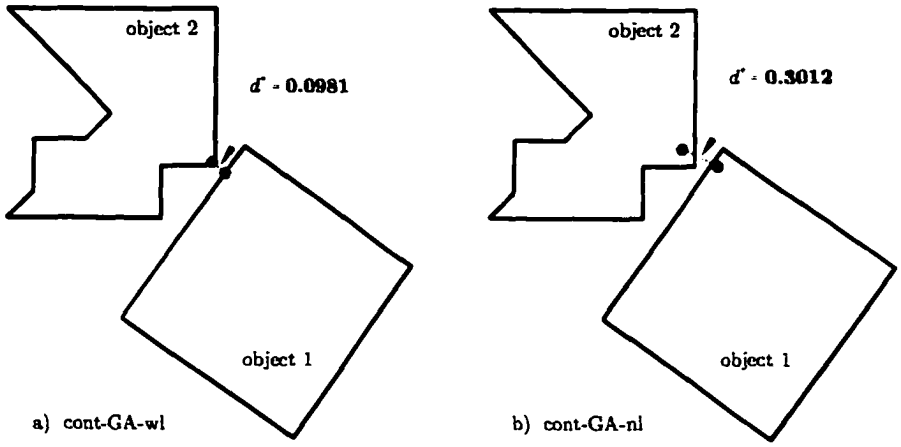


Figure 2.21. Results obtained by a) cont-GA-wl and b) cont-GA-nl (Geometry set 1 in position 2).

GA-wl) obtained much better results than the GA not using the local optimization but took approximately twice as much computational time as cont-GA-nl. This is due to the fact that the objective function is calculated twice, once before the local optimization in order to obtain the displacement distance for the move closer method, and one after in order to obtain the fitness value that will be used in the selection process.

Algorithm cont-GA-wl used 3.14 Mflops whereas cont-GA-nl used 1.34 Mflops. Thus, in a current 2200 MHz PC which can perform in excess of 1030 Mflop per second [Dongarra, 2002], these operations would take around 3.0 and 1.3 ms. In this preliminary implementation, about 84% of the computational time is spent calculating the objective function value of which close to 90% is spent computing the constraints

Parameter	Symbol	Value
initial temperature	t_{ini}	5,000
Boltzmann constant	k_B	0.0005
penalty weight	k_p	10
repair scaling (safe repair)	k_r	1.5

Table 2.4. Parameters for cont-SA-wl and cont-SA-nl runs.

values alone (*i.e.*, about 75% of the total time).

2.4.1.b SA trials

To allow a fair comparison between the results of the GA and SA optimization methods, both algorithms were allowed to run a similar number of trials. That is, the initial temperature of the SA runs was set to $t_{ini} = N_{pop} \times G_{max}$, where N_{pop} and G_{max} were set using the values in Table 2.1. The Boltzmann constant k_B was obtained by experimentation until a reasonable result was obtained. Table 2.4 summarizes the SA parameters used for the example presented here.

The results for the SA runs for positions 1 and 2 are illustrated in Figures 2.22 to 2.25 and are summarized in Table 2.4. It is important to notice that little improvement of the solution was made after the first 1,000 iterations in the case of cont-SA-wl and 3,000 for cont-SA-nl (see Figures 2.22 and 2.24). That is, after locating a promising region, the algorithms only improved the solution locally.

The quality of the solutions obtained using cont-SA-wl surpassed the solutions obtained by cont-SA-nl. Cont-SA-wl results are better due to its proximity to the exact solution. Additionally, cont-SA-wl was able to find the global minimum region rather than converging to a local minimum as the con-SA-nl algorithm did with the objects in position 1. The fact that this problem only has weak minima rendered the task of finding the exact minimum solution more difficult. That is, since the object

	Position 1	Region	Position 2	Region
cont-SA-wl	0.3325	✓	0.1117	✓
cont-SA-nl	0.8372	×	0.4965	✓
exact	0.2984		0.0979	

Table 2.5. Minimum distance results for the example 1 in positions 1 and 2 for algorithms cont-SA-wl and cont-SA-nl.

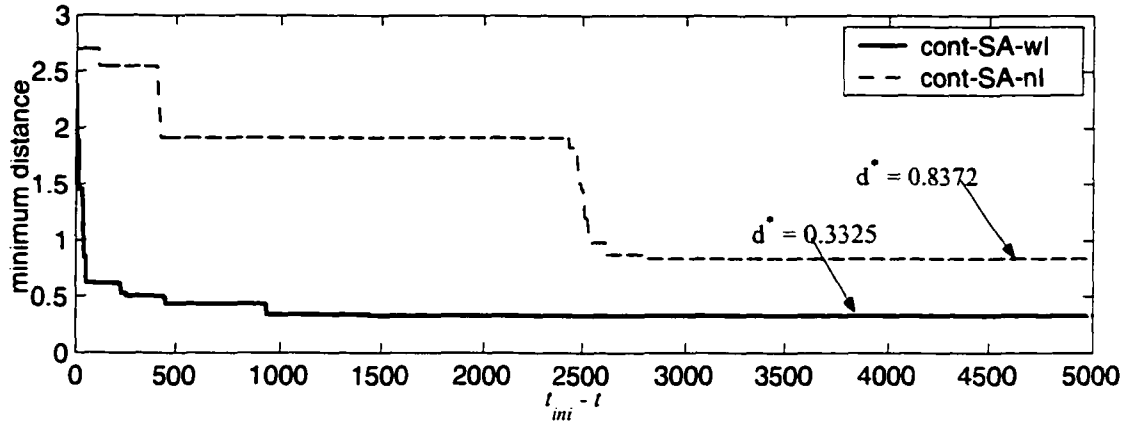


Figure 2.22. Time history of the minimum distance of the trial point for cont-SA-wl and cont-SA-nl (Geometry set 1 in position 1).

edges that are perpendicular to the plane of the page are parallel, there are an infinite number of trial points along these edges that minimize the distance between objects 1 and 2. As a matter of fact, similar cases were tried slightly tilting the object 1 along the edges on the plane of the page, and the SA algorithms (particularly cont-SA-wl) were able to better approximate the exact solution since the minimum solution is no longer a weak minimum but is rather a strong one.

In terms of computational expense, cont-SA-wl took approximately 3.8 Mflops whereas cont-SA-nl took approximately 2.2 Mflops.

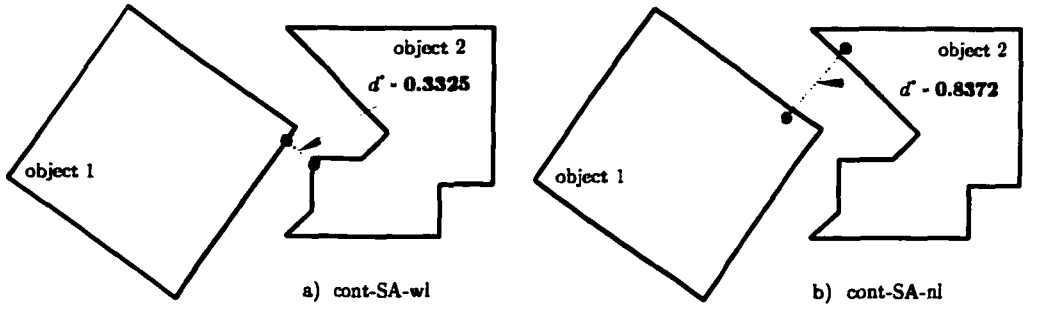


Figure 2.23. Results obtained by a) cont-SA-wl and b) cont-SA-nl (Geometry set 1 in position 1).

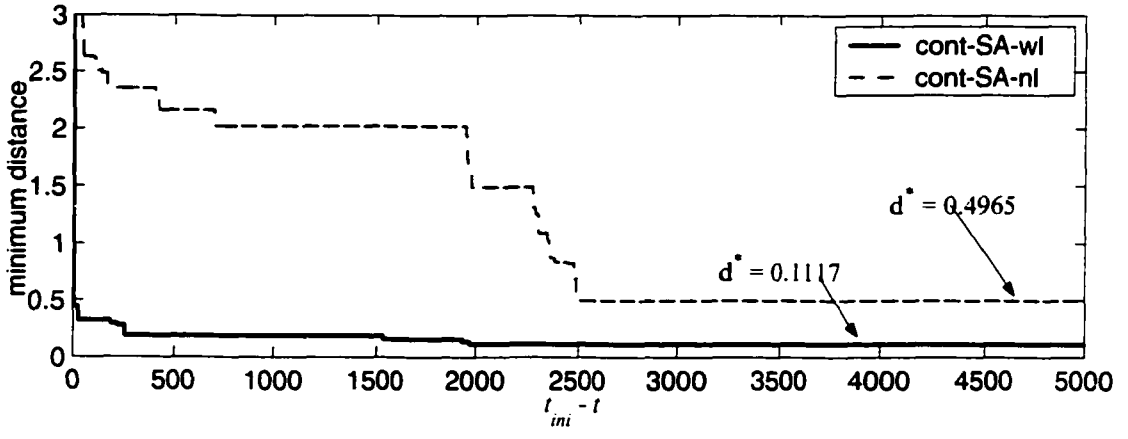


Figure 2.24. Time history of the minimum distance of the trial point for cont-SA-wl and cont-SA-nl (Geometry set 1 in position 2).

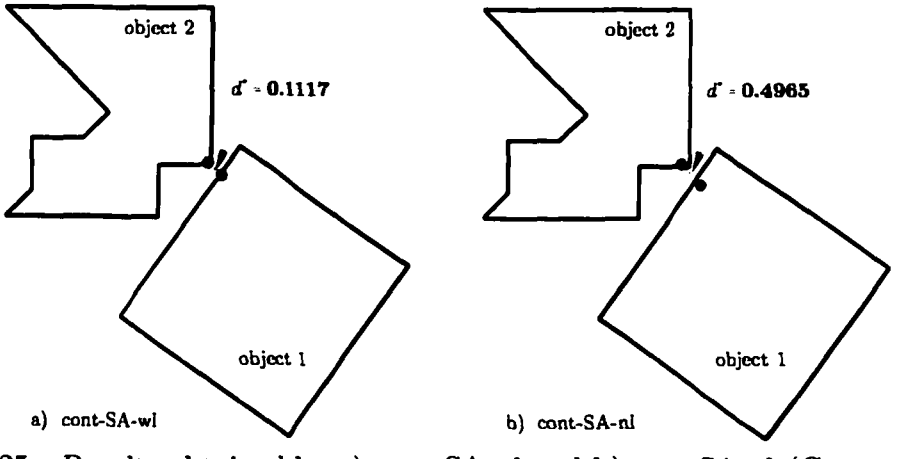


Figure 2.25. Results obtained by a) cont-SA-wl and b) cont-SA-nl (Geometry set 1 in position 2).

2.4.2 Cylinder and semi-extruded object

Figure 2.26 shows the geometries of the two objects used for this example. This example consists of a set of two objects where one is a body of revolution (cylinder with a groove) while the second one is a more complex object with a few concavities. Although the continuous approach is capable of handling any type of surface (*i.e.*, linear, quadratic, *etc.*), the curved surfaces of the objects were linearized to simplify the computer implementation. The objects, once linearized, have a combined total of 88 faces which can be represented using 88 linear inequality constraints.

The two configurations shown in Figure 2.27 were considered to better illustrate the capabilities of the proposed algorithms. The configuration of the objects illustrated in Figure 2.27a only has multiple minimum regions. In addition to demonstrating the possibility of handling concave objects, this example illustrates the capabilities of the algorithm to handle concave situations where the global minimum is a weak minimum [Gill et al., 1981] (*i.e.*, the fitness function does not vary much around the area surrounding the minimum).

On the other hand, the second position considered for this geometry (see Figure 2.27b) illustrates the algorithms' capabilities to handle more complex object where more that one local minimum is found.

2.4.2.a GA trials

The GA parameters used here are identical to the ones used in the previous example (see Table 2.1).

Figures 2.28 and 2.30 show the convergence of the solution using cont-GA-nl and cont-GA-wl for positions 1 and 2, respectively. Additionally, Figures 2.29 and 2.31

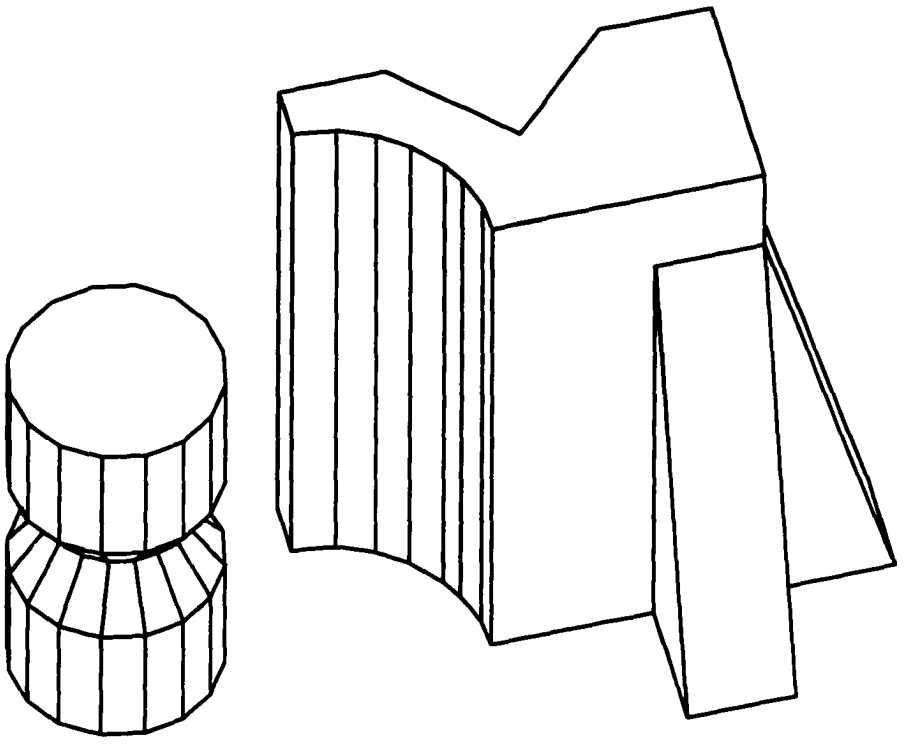


Figure 2.26. Geometry of objects 1 and 2 for example 2: cylinder and semi-extruded object.

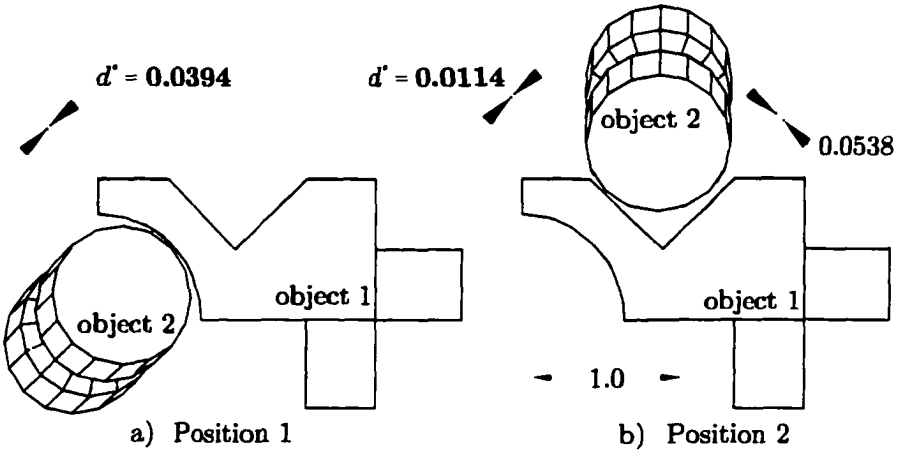


Figure 2.27. Geometry and configuration of the cylinder and semi-extruded object examples with the geometries in a) Position 1 and b) Position 2. Note that in both cases, the exact minimum distance is shown in bold face and labelled as d^*.

show the final solution of the GA runs for positions 1 and 2, respectively. While doing these sample runs, it was noticed that the use of the local optimization not only helped to improve the quality of the solution, but it also helped the algorithm to find the global minimum region. For example, with the bodies in position 2 (Figure 2.31), cont-GA-nl converged to the bottom side of the V-shape groove on the cylindrical object while cont-GA-wl converged correctly. This trend was observed in a few cases emphasizing the importance of the use of the local optimization.

As expected from the complexity of the geometries, both algorithms used more computational time to arrive to the solution than the computational time needed to solve the first example. In this example, the number of flops used by the algorithms to solve for the objects in either position 1 or position 2 was observed to be 6.75 Mflops and 3.27 Mflops for cont-GA-wl and cont-GA-nl, respectively. Thus, at 1030 Mflop per second [Dongarra, 2002], these operations would take around 6.5 and 3.2 ms.

Finally, one can notice that, for the two examples given above, neither cont-GA-wl nor cont-GA-nl converges to the exact minimum solution. This is due to the nature of the GAs. In theory, one would have to allow the GA to run an infinite number of generations in order to find the exact minimum solution. To avoid this, it is proposed that the solution of the current GA be used as the start point for a local optimization routine such as the one described in [Bobrow, 1989]. That is, the GA running a limited number of generations can be used to pinpoint the region of the bodies where the closest points lie.

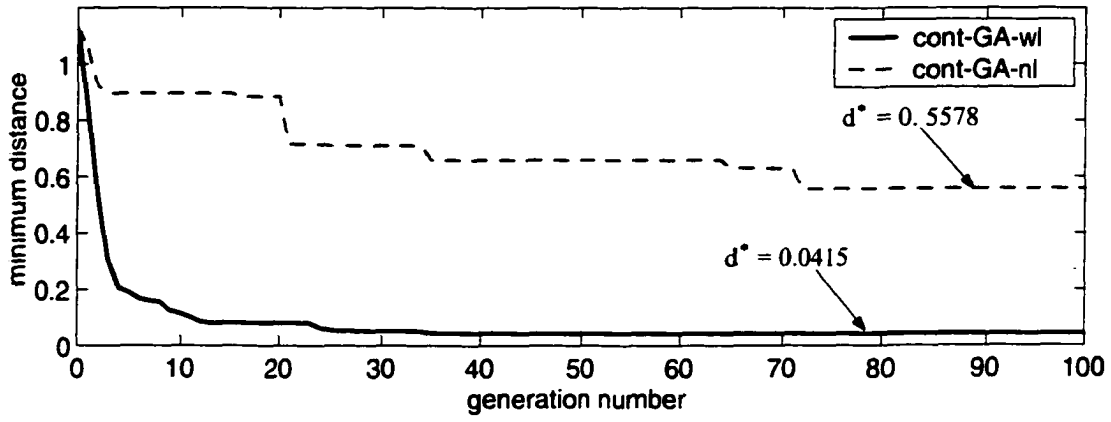


Figure 2.28. Time history of the best individual in the population for cont-GA-wl and cont-GA-nl (Geometry set 2 in position 1).

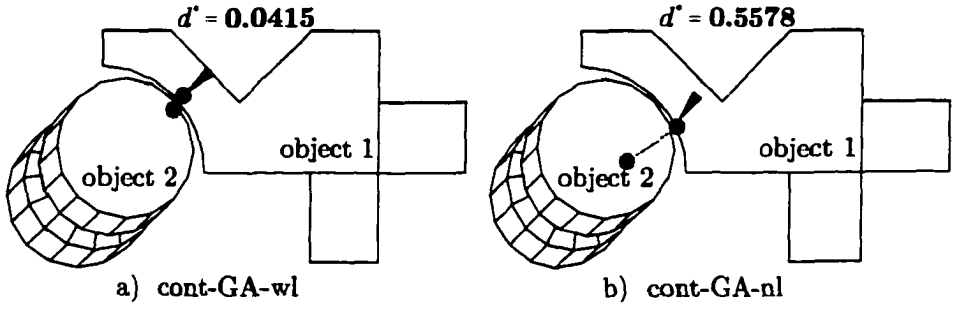


Figure 2.29. Results obtained by a) cont-GA-wl and b) cont-GA-nl (Geometry set 2 in position 1).

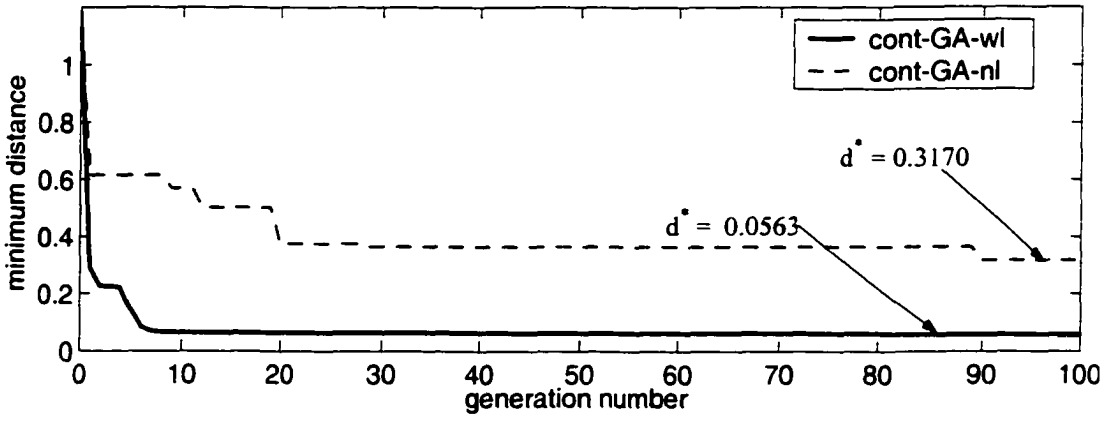


Figure 2.30. Time history of the best individual in the population for cont-GA-wl and cont-GA-nl (Geometry set 2 in position 2).

	Position 1	Region	Position 2	Region
cont-GA-wl	0.0415	✓	0.0563	✓
cont-GA-nl	0.5578	✓	0.3170	×
exact	0.0394		0.0114	

Table 2.6. Minimum distance results for the example 2 in positions 1 and 2 for algorithms cont-GA-wl and cont-GA-nl.

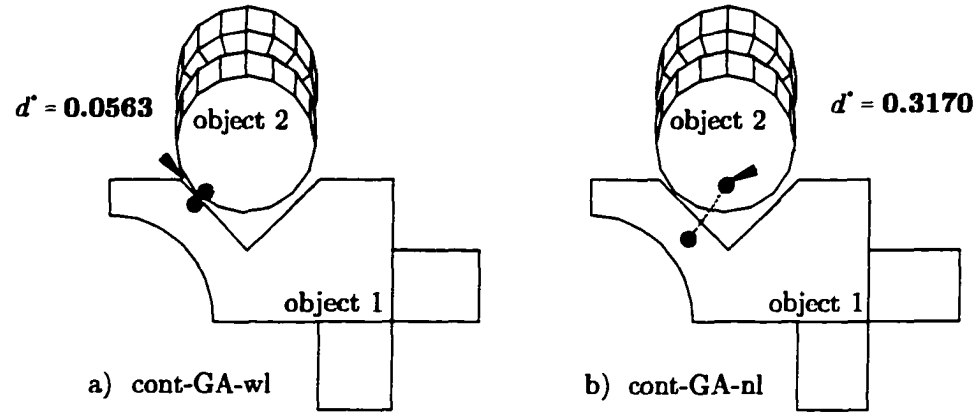


Figure 2.31. Results obtained by a) cont-GA-wl and b) cont-GA-nl (Geometry set 2 in position 2).

2.4.2.b SA trials

The SA parameters used for this second example are identical to the ones used in the first example (see Table 2.4).

Figures 2.32 to 2.35 show the results obtained using cont-SA-nl and cont-SA-wl for positions 1 and 2. Also, the results are summarized in Table 2.7. As seen in earlier trials, the solution obtained by the algorithm using local optimization (*i.e.*, cont-SA-wl) is much closer to the exact solution than the one obtained by cont-SA-nl. Additionally, cont-SA-nl was not able to locate the region where the minimum solution is located with objects in position 2 (see Figure 2.35b).

Similar to the trend observed in the previous example using SA, the cont-SA-wl algorithms was able to locate the minimum region early in the run (around the

	Position 1	Region	Position 2	Region
cont-SA-wl	0.0628	✓	0.0390	✓
cont-SA-nl	0.5764	✓	0.3502	×
exact	0.0394		0.0114	

Table 2.7. Minimum distance results for the example 2 in positions 1 and 2 for algorithms cont-SA-wl and cont-SA-nl.

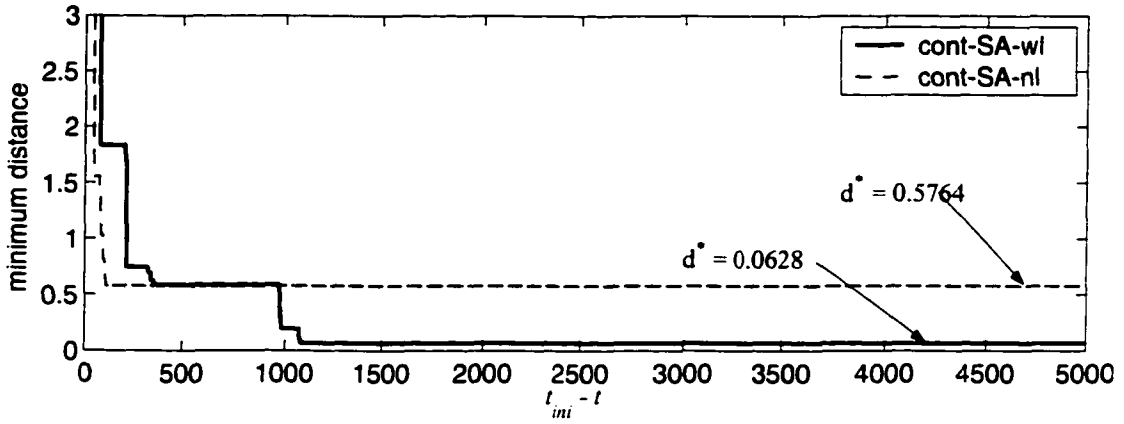
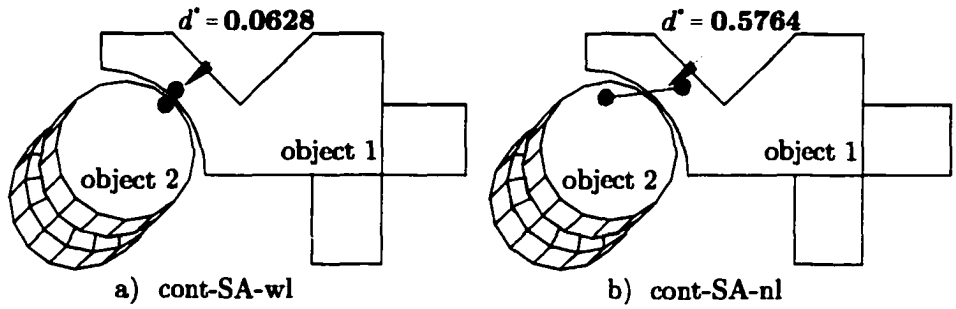


Figure 2.32. Time history of the minimum distance of the trial point for cont-SA-wl and cont-SA-nl (Geometry set 2 in position 1).

1,500th iteration) and was able to slightly improve it in the remainder of the run.

Since the SA algorithms were allowed to run for the same number of trial points as the GA, the computational time taken by the SA runs in this example was similar to the one used by the GA. That is, 6.5 Mflops and 3.1 Mflops for cont-SA-wl and cont-SA-nl, respectively, as compared to 6.75 Mflops and 3.27 Mflops for cont-GA-wl and cont-GA-nl. This slightly lower computational expense, as compared to the GA implementation, is expected since the GA implementation requires some extra computations during the selection and mating stages not necessary in the SA implementations.



a) cont-SA-wl b) cont-SA-nl

Figure 2.33. Results obtained by a) cont-SA-wl and b) cont-SA-nl (Geometry set 2 in position 1).

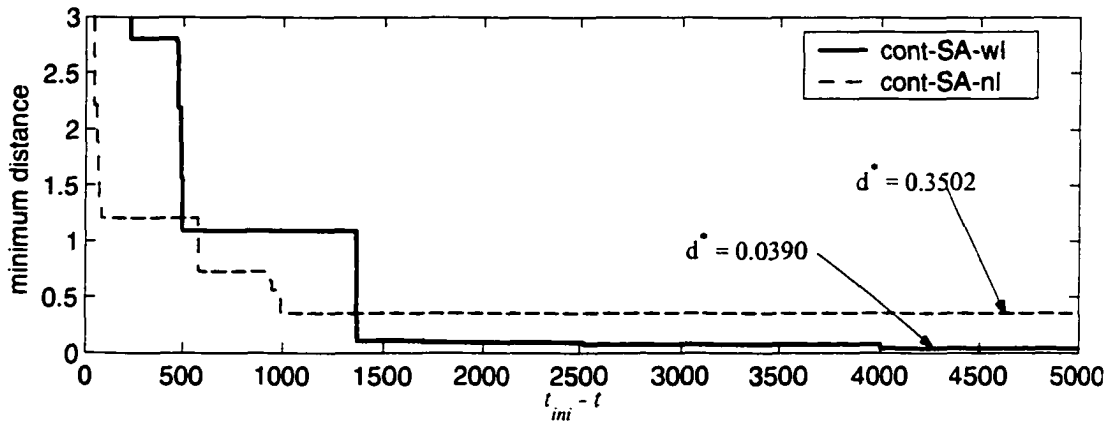


Figure 2.34. Time history of the minimum distance of the trial point for cont-SA-wl and cont-SA-nl (Geometry set 2 in position 2).

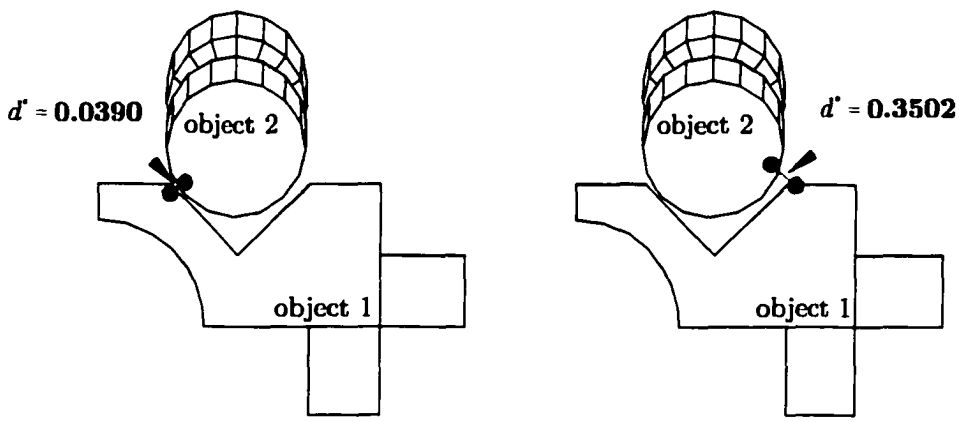


Figure 2.35. Results obtained by a) cont-SA-wl and b) cont-SA-nl (Geometry set 2 in position 2).

Chapter 3

Combinatorial Approach

In Chapter 2, it was concluded that the bulk of the computational burden (up to 90%) of the distance problem comes from having to determine whether or not a point is within the object. In this chapter, a method is proposed to eliminate this on-line computational expense. This new approach replaces the geometry of the object, originally described by half spaces (see equation (2.2)), by a number of points on or in the object itself (*i.e.*, all trial points are then feasible). For this method, it is proposed to use surface or volume meshes to structure the surface or volume points. These mesh points are referred to as nodes. As will be seen later in this chapter, the new approach allowed the creation of a local optimization routine that enables points to be moved along the edges of the mesh.

Since the objects are now described by a series of points (the nodes of the meshes) the optimization becomes a combinatorial problem in which the combination of points (one on each object) that minimizes the distance between them is sought. Thus the name *combinatorial approach*. To solve this combinatorial problem, it is proposed to use global optimization techniques, particularly Simulated Annealing and Genetic

Algorithms.

Based on the general description of the SA and GA algorithms presented in Chapter 2, these two optimization methods were adapted to the combinatorial optimization problem and are presented here. The details that make the combinatorial implementation of the SA and GA algorithms different from their continuous counterparts are outlined in this chapter and some of the challenges of this approach are highlighted.

Finally, this chapter concludes with a few simple examples that demonstrate the capabilities of the proposed combinatorial method at solving the minimum distance problem between concave objects.

3.1 Formulation

To avoid having to calculate the constraints at every iteration, it is proposed to replace the geometry of the objects by a finite number of points on the surface of the object. Thus, if a fixed number of points is given, the minimum distance problem is reduced to a combinatorial optimization problem. That is: which is the best combination of two points (one on each body) which yields the minimum distance between them. Since the points on the surfaces of each body can be generated off-line, the number of algebraic calculations per iteration will be substantially reduced.

Although this method eliminates the computational expense involved in checking the feasibility of any point, it poses some accuracy problems. That is, the obtained solution of the minimum distance problem will be an *approximation* to the real solution and its accuracy will be limited by the number of points on each object and the distribution of these points.

In principle, finding the solution of this problem by enumeration is possible (see

Appendix D) but, in practice, with most realistic problems where the number of combinations is very large, this task is very often infeasible in a reasonable amount of time [Gen and Cheng, 1997]. This is the case with the minimum distance problem, thus, the use of a global optimization techniques such as GA or SA is proposed.

This method also imposes a few challenges such as (a) the creation of evenly-distributed points on the entire surface of the objects and (b) the implementation of a combinatorial optimization algorithm [Bjorndal et al., 1995].

3.1.1 Random points on the surface

One possible method to obtain points on the surface of each body is by randomly distributing points on the surface of the body. To ensure an even distribution of points, one could calculate the point density by calculating the entire object's area and dividing it by the total number of points. It is then possible to locate a limited number of points on each surface proportional to its area. Furthermore, while working with linearly bounded objects, it is possible to add extra points at every vertex and evenly on each edge since it is there that the solutions of the distance problem are most likely to be found.

3.1.2 Points on a mesh

Meshes can be used in order to create an even distribution of points on the surface of each body (surface mesh) or on the surface *and* the internal volume of the object (volumetric mesh or grid). Regardless of the type of mesh used, meshing algorithms create meshes with points at each of the vertices of the object and, depending on the size of each edge, a few points along every edge of the object [Zeid, 1991]. Having

node points on the vertices and edges is very important for the distance problem since the solution of the minimum separation distance problem, particularly amongst polyhedra, is most likely to be found at a vertex or edge.

Most finite element analysis packages, commercial or not, offer the capability of creating meshes. Surface generation methods vary mainly according to the type of elements they produce, for instance triangular elements, quadrilateral elements, *etc.* Similarly, volumetric meshes can be produced using different types of elements, typically tetrahedral or hexahedral (polyhedra of 4 and 6 faces, respectively).

3.1.2.a Mesh storage

Typically, meshes are stored in matrices. The first of these matrices is the node matrix $\mathbf{N}$ that contains the Cartesian coordinates of all v mesh points (*a.k.a.* nodes) with respect to a body frame. That is, each row of $\mathbf{N}$ corresponds to one point and the columns contain the x, y and z coordinates of that point. The second matrix is called the element or facet matrix and gives a list of all the elements of the mesh. Each row of the element matrix $\mathbf{F}$ contains m integer elements that correspond to the m vertices of the polygon. That is, a surface mesh with square facets would have an element matrix $\mathbf{F}$ with four columns. The value of each element in the element matrix corresponds to a particular point (*i.e.*, row) in the coordinate matrix. Additionally, volumetric meshes contain an extra matrix, the rows of which contain the indexes of all the polygonal elements that make the facets of each three dimensional element.

In the present work, a third matrix, a connectivity matrix Π, is also stored with each mesh. The matrix Π is a square matrix with as many rows and columns as node points on the mesh, *i.e.*, v. The connectivity matrix Π only contains binary elements which correspond to the particular elements of the mesh that are connected to each

other. That is, if element $\Pi_{[i,j]} = 1$ means that nodes n_i and n_j are connected to each other. Since Π is a sparse symmetric matrix, it can be stored more compactly using a cell array with v rows. Each row would only contain the integer numbers corresponding to the adjacent nodes. That is, to see what all the connecting points to node n_i are, one has to look at row i of Π.

To obtain the connectivity matrix, which gives a list of adjacent nodes, the facet matrix $\mathbf{F}$ is analyzed. Since every row of $\mathbf{F}$ contains m connections corresponding to the polygon that creates the facet, each pair of consecutive elements in that row correspond to connecting node points.

3.1.2.b Mesh distances (distance and predecessor matrices)

Dijkstra's algorithm [Dijkstra, 1959] is commonly used in computer networks in order to find the shortest path to send information through a network of computers [Tanenbaum, 1996]. Also, some path planning formulations use Dijkstra's algorithm to determine the shortest path in a visibility graph [O'Rourke, 1993].

In order to produce effective mating and mutation operations, it is proposed to use Dijkstra's algorithm to obtain the distance and predecessor matrices for the meshes[1]. These two matrices determine the shortest distance and path between two elements of the same mesh. This process is computationally expensive but, in the framework of the work that is proposed here, this is not so important since it is performed off-line. These two matrices would only need to be obtained once for each object and the results stored for later use in the minimum distance algorithms.

[1] In the context of SA algorithms, the generation of a new random point can be done by a mutation as will be explained later in this chapter.

Following is a brief description of the distance and predecessor matrices. For a mesh with v nodes, the distance and predecessor matrices ($\mathbf{D}$ and $\mathbf{P}$, respectively) will both be of size $v \times v$. For $\mathbf{D}$ and $\mathbf{P}$, the row indexes correspond to source points and the columns indexes to destination points. Thus, the distance δ between nodes n_s and n_t (*i.e.*, $\delta_{s \to t}$) is stored in $\mathbf{D}_{[s,t]}$, *i.e.*, $\delta_{s \to t} = \mathbf{D}_{[s,t]}$. Thus, the distance matrix $\mathbf{D}$ contains the shortest distance between every possible pair of points on the mesh. In addition, this matrix is symmetric.

On the other hand, in order to obtain the shortest path between two nodes on the mesh, namely n_s and n_t, element $[s,t]$ of matrix $\mathbf{P}$ contains the last element to be reached in the path from point n_s to point n_t, namely n_k. That is, in order to reach the destination point n_t from the source point n_s, it is necessary to first reach node n_k (see Figure 3.1). Then, to get the element in the path before n_k that need to be reached, it is necessary to look at $\mathbf{P}_{[s,k]}$ which contains another element n_{k-1}. The process would be repeated until the node n_s is found as the entry $\mathbf{P}_{[s,k-j+1]}$. Thus, the path $\mathbf{p}$ to go from n_s to n_t is $\mathbf{p}_{n_s \to n_t} = [n_s, n_{k-j+1}, \ldots, n_{k-1}, n_k, n_t]$. It follows that the distance $\delta_{s \to t}$ is equal to the addition of the distance between the intermediate nodes, that is, $\delta_{s \to t} = \delta_{s \to k-j+1} + \ldots \delta_{k-1 \to k} + \delta_{k \to t}$.

3.1.3 Encoding of trial points

One of the most important aspects of the implementation of an optimization algorithm is the encoding of the trial points, *i.e.*, the encoding of the search variables. In the present problem, the encoding consists of a set of two indices each referring to a particular node on the mesh of each body, *e.g.*

$$\mathbf{x} = [\ n_1 \quad n_2\ ] \tag{3.1}$$

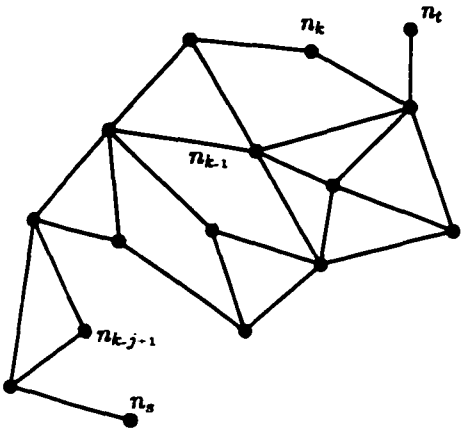


Figure 3.1. Example of minimum distance path to reach destination point n_t from source point n_s.

where n_i is an integer that identifies a particular node on the object's node matrix $\mathbf{N}_i$. When an object's mesh is created, each node on the mesh is assigned a particular index. In the present case, the Cartesian coordinates of the nodes of each object's mesh are stored in a matrix $\mathbf{N}_k$ where k is the object's number. Each row of $\mathbf{N}_k$ contains the three Cartesian coordinates of the node with respect to a body-fixed frame. Thus, in equation (3.1), n_1 corresponds to the point stored in row n_1 of the node matrix of body 1 (*i.e.*, $\mathbf{N}_1$) and similarly, n_2 corresponds to the point stored in row n_2 of the second body's node matrix $\mathbf{N}_2$.

3.1.4 Objective function

Since the constraints have been eliminated by the use of meshes, the objective function used in this combinatorial formulation only includes the square of the Euclidean distance between two points. That is, the problem is formulated as an unconstrained minimization problem as

$$\min : d^2 = (\mathbf{p}_1 - \mathbf{p}_2)^T(\mathbf{p}_1 - \mathbf{p}_2) \tag{3.2}$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ correspond to the Cartesian coordinates of points on body 1 and 2, respectively. The Cartesian coordinates of $\mathbf{p}_i$ are extracted from the n_i-th row of the node matrix $\mathbf{N}_i$. These coordinates are expressed in terms of the body frame Σ_i. That is, ${}^i\mathbf{p}_i$ contains the coordinates extracted from $\mathbf{N}_i$. Note that the preceding superscript in ${}^i\mathbf{p}_i$ is used to denote the reference frame in which the coordinates of the points are expressed.

To calculate d^2, both points need to be expressed with respect to a common frame. In order to reduce the computational expense of expressing both points with respect to the inertial frame Σ_0, the method described in Appendix B is used where points are described with respect to Σ_1 instead of Σ_0.

Notice that, in equation (3.2), it is the square of the distance that is minimized and not the distance d itself. This is done to avoid the computational expense of calculating the square root in equation (3.2) which can be done since minimizing a is equivalent to minimizing $a^{\frac{1}{2}}$ if a is a positive number [Ma and Nahon, 1992].

3.1.5 Local optimization

As was done for the constrained optimization method, a local optimization procedure is also proposed. This time, the local optimization method is based on the information that can be extracted from the mesh. The connectivity matrix, obtained when the mesh is created, provides the indices of all the neighbouring nodes to a particular node.

Using the mesh's connectivity matrix (that is, the matrix Π that defines all connections (edges) of the mesh), the local optimization method for this *combinatorial approach* tries to reduce the distance between two selected points, one on each body.

The algorithm works by moving each node in the trial point along the edges of the mesh to a neighbouring node.

Starting from an initial trial point $\mathbf{x}$ formed by two node indices, *i.e.*, $\mathbf{x} = [\ n_1 \quad n_2\ ]$, node n_1 on object 1 is moved successively to each of its neighbours while node n_2 on object 2 is fixed. The neighbour of n_1 that minimizes the distance between n_1 and n_2 is accepted as the new location of the node n_1 for the trial point $\mathbf{x}$. This process of finding a closer neighbouring node is repeated until the best n_1 is found, *i.e.*, no further improvement can be made. Then, the entire process is repeated by fixing n_1 and moving n_2. This procedure is repeated, alternating the moving and the fixed nodes until the distance between the nodes is minimized, *i.e.*, until no neighbouring node can be found that reduces the distance between n_1 and n_2.

The results of using this local optimization algorithm are illustrated in Figure 3.2 for two different starting conditions. In both cases, the nodes in the initial trial point $\mathbf{x}_0 = [\ n_{1_0} \quad n_{2_0}\ ]$ are moved along the edges of the mesh using this gradient based method until trial point $\mathbf{x}^* = [\ n_1^* \quad n_2^*\ ]$, the local optimum, is found. Notice that in the particular cases illustrated in Figure 3.2a and 3.2b, the distance between nodes n_{1_0} and n_{2_0}, *i.e.*, d_0, is significantly larger than d^* (the distance between nodes n_1^* and n_2^*) showing a big improvement of the trial points.

On the other hand, since this is effectively a gradient based method, it is clear that, on the local scale, the solution will greatly depend on the start point. This is illustrated in Figures 3.2a and 3.2b where in the first case the local optimization stated at $\mathbf{x}_0 = [\ n_{1_0} \quad n_{2_0}\ ]$ converged to the global solution $\mathbf{x}^* = [\ n_1^* \quad n_2^*\ ]$ and in the latter case the local optimization started at a different trial point and was 'trapped' in a local minimum. However, in the larger scale, the global optimization algorithms

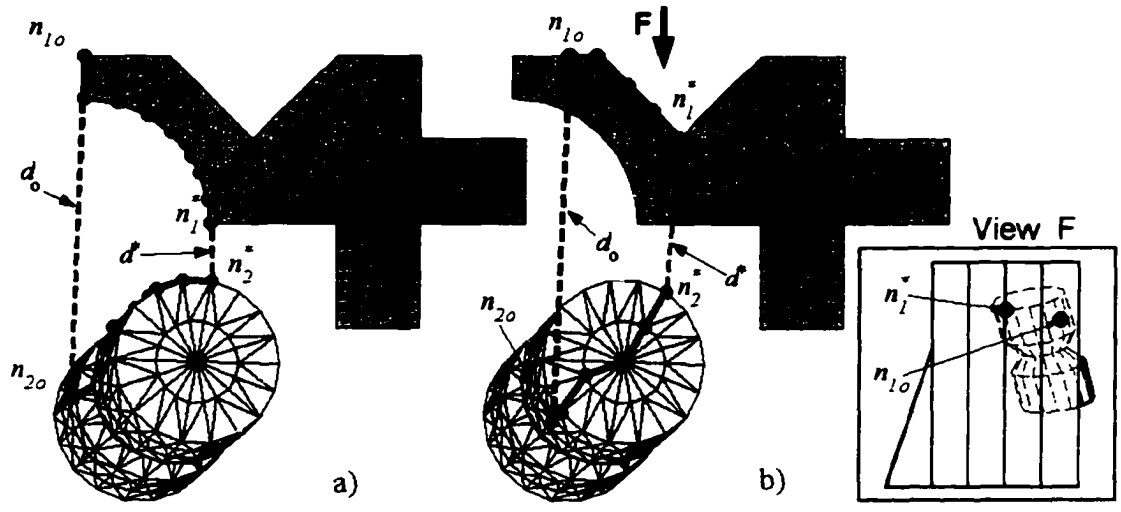


Figure 3.2. Example of the local optimization for the combinatorial approach. In a) the local minimum $\mathbf{x}^* = [n_1^* \; n_2^*]$ is the global minimum whereas in b) the local optimization was 'trapped' in a local minimum (node n_1^* is in the opposite side of object 1). Note that the objects are three dimensional (see inset and Figure 2.26).

are expected to introduce the necessary ability to jump out of local minima 'traps'.

Although, as will be seen later, this local optimization algorithm largely improves the quality of the solution, it poses other problems such as the computational expense to carry out the local optimization. In a variety of meshes the average number of connections (*i.e.*, neighbours) for every node was found to be around 5. Thus, for every step the local optimization takes, around 60 multiplication and 70 additions (130 flops) are needed to make the decision where to move the node point. Since this must be repeated until the best points are found, the solution will greatly depend on the location of the start point and the coarseness of the mesh. That is, a fine mesh allows only for small improvements at each iteration thus needing a much greater number of computations to perform a local search than if a coarser grid was used. On the other hand, a finer grid would allow the algorithm to obtain a more accurate local solution.

3.1.6 Initial guess

For the initial iteration of the combinatorial approach, sets of points are generated at random. In this case, the initial population is formed by picking points randomly from the list of nodes from each object's mesh.

3.2 SA implementation

3.2.1 Structure

The Simulated Annealing algorithm implemented for the combinatorial approach is, in essence, identical to the one described in Section 2.2 for the continuous approach. Only the operations specific to the combinatorial approach are changed but the SA structure described in Figure 2.9 remains untouched.

Most issues specific to the combinatorial approach were described earlier in this chapter (encoding, objective function and local optimization). Only the method proposed to create new points for the SA search still needs to be described.

3.2.2 Generation of new trial points

The exploration and exploitation in the SA algorithm is achieved by generating new points in the solution space. Thus, the development of mechanisms to generate new points is of vital importance to the SA algorithm. Two methods are proposed here, one that generates a new trial point at a random node in or on the object and a second one that moves the current trial point to a new node which is within a prescribed distance.

3.2.2.a New point

As described earlier in this chapter, each trial point $\mathbf{x}$ consists of two integers n_1 and n_2 arranged as a vector, *i.e.*, $\mathbf{x} = [\ n_1 \quad n_2\ ]$. These two integers n_1 and n_2 correspond to node indices (rows) in the node matrix $\mathbf{N}_1$ and $\mathbf{N}_2$, respectively. Knowing that matrices $\mathbf{N}_1$ and $\mathbf{N}_2$ contain a total of v_1 and v_2 nodes each, generating a new trial point involves creating a new set of point indices by randomly selecting a different node from the list of nodes. That is, the new value for n_i would correspond to a random integer in the range $[1, v_i]$.

3.2.2.b Random move

In an effort to give the SA algorithm the capability to explore regions *from* the current trial point, a *random move* method was devised. Since the method is based on moving the current point to another trial point near the current point, when the displacement is small, the method will also allow the SA to exploit promising regions of the space.

The method uses mesh information such as the distance and predecessor matrices described in previous sections (*i.e.*, matrices $\mathbf{D}$ and $\mathbf{P}$, respectively). First, if a point on mesh i is to be modified, a totally new random point ($n_{i_{\text{rand}}}$) is generated as described in the previous section. Next the path between the current point (n_i) and $n_{i_{\text{rand}}}$ is determined using the predecessor matrix $\mathbf{P}$ as indicated in Section 3.1.2.b, that is $\mathbf{p}_{n_i \rightarrow n_{i_{\text{rand}}}}$ is obtained. Finally, one of the points along the path $\mathbf{p}_{n_i \rightarrow n_{i_{\text{rand}}}}$ within a user defined radius $r_{\max}$ from n_i is selected as the trial point $n_{i_{new}}$. See Figure 3.3 for an example of this method. Note that the method uses the mesh distance δ between n_i and $n_{i_{new}}$ (*i.e.*, $\delta_{n_i \rightarrow n_{i_{new}}}$) and not the Cartesian distance since

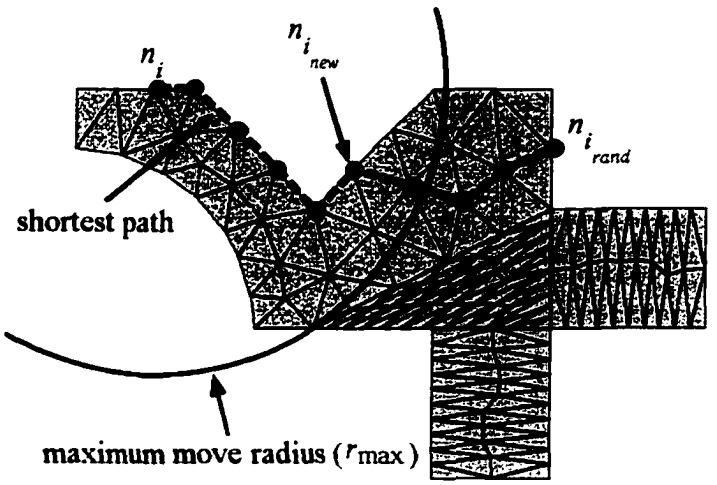


Figure 3.3. Example of the random move method for the combinatorial approach. Note that the method uses the mesh distance between nodes n_i and $n_{i_{rand}}$ and not the Cartesian distance.

it is the mesh distance that is stored in the distance matrix **D**.

3.2.2.c Random move with variable displacement

As was done in the continuous approach, it is suggested to limit the size of the random moves according to the iteration number using a Boltzmann type of variation. This would allow the algorithm to exploit the promising region found towards the end of the simulation allowing it to converge to a better solution.

The method used for scaling the size of the maximum displacement is by scaling the radius r_{max} using the same Boltzmann probability factor used during the SA run. That is, $r_{max_{scaled}} = p_b r_{max}$ where p_b is the Boltzmann probability factor obtained using equation (2.6).

3.3 GA implementation

3.3.1 Structure

The basic Genetic Algorithm was described in Section 2.3. For this new combinatorial approach, it is proposed to use the same basic GA structure as the continuous approach does. Obviously, since the formulation has changed, all genetic operations need to be adapted to the new combinatorial approach.

3.3.2 Genetic operators

In order for a genetic algorithm to be successful at evolving a population of randomly generated individuals, the mating operator has to be effective at allowing the algorithm to exploit good traits of superior individuals. On the other hand, in order to allow the GA to explore new regions, the mutation operator needs to be effective at creating new individuals in the population that lie in a new region of the search space.

3.3.2.a Selection

The selection process in a GA is essential in order to exploit promising regions of the space. At the same time, by randomly selecting some poor individuals to proceed to the next generation, it creates the population diversity needed for the exploration of the entire search space.

For this combinatorial approach, it is proposed to use Stochastic Universal Sampling [Baker, 1987] (*a.k.a.* modified roulette wheel selection) which, as described Section 2.3, is a simple selection method that has shown good results in many appli-

cations such as function optimization.

3.3.2.b Mating

The mating process of a GA is the mechanism that allows it to pass to the next generations the traits of the selected individuals. In order to do so, the mating strategy has to be able to extract traits from each individual and transfer them to its offsprings.

Genotypic mating In this work, *genotypic mating* refers to a method that only uses the values of the genes in order to produce offspring. In the combinatorial approach, the encoded points (*i.e.*, the genes) do not represent any physical characteristic of the point they represent. They merely represent where the coordinates of the point are stored within the node matrix $\mathbf{N}$. Thus, by obtaining offsprings using any linear scaling between two individuals in the encoded space (*i.e.*, between node numbers), one would not necessarily obtain offspring with physical traits similar to either one of its parents. That is, if linear random scaling between node numbers is used, a pair of parents such as $\mathbf{x}_1 = [\ 1 \quad 25\ ]$ and $\mathbf{x}_2 = [\ 8 \quad 32\ ]$ could have an offspring such as $\mathbf{x}_{off_1} = [\ 4 \quad 28\ ]$ (see Figure 3.4). Even though the indices obtained in this way are within the indices of its parents in their respective node matrices $\mathbf{N}_1$ and $\mathbf{N}_2$, the Cartesian coordinates of the new points would not necessarily be related. As a matter of fact, since the node points on a mesh are stored in no particular order, the offspring $\mathbf{x}_{off_1}$ could be located on the opposite side of the body. This problem is illustrated in Figure 3.4 where parents in object 1 (nodes 1 and 8) are mated to produce an offspring (node 4) which is located at the opposite side of the mesh.

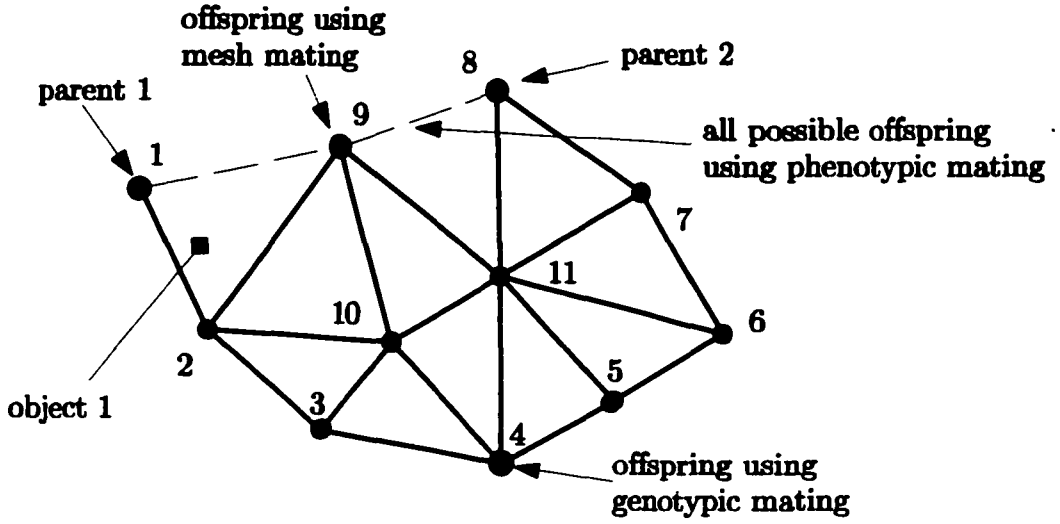


Figure 3.4. Examples of genotypic, phenotypic and mesh mating for a simple mesh.

Phenotypic mating To allow parents to pass on traits to their offspring, the mating of the two points would have to be performed in the decoded space. This method, referred to here as *phenotypic mating*, considers the Cartesian coordinates of the parents and mates them in the decoded space (*i.e.*, Cartesian space) using a strategy such as the *linear random scaling* described in Section 2.3.2.b (the dashed line in Figure 3.4 shows the location of possible offspring using *phenotypic mating*). Once the offspring are produced, *i.e.*, Cartesian coordinates of the offspring are obtained, they have to be encoded back as genes, that is, translating the point coordinates into a node on the mesh. This requires finding the node on the mesh that is the closest to the particular offspring which in itself is an optimization problem. Windowing [Edelsbrunner, 1987] and other techniques (such as enumeration) that could be used to perform this task can be extremely complex and time consuming.

Mesh mating To solve the problems inherent to the two previous mating methods (*genotypic* and *phenotypic mating*), a method for mating points on a mesh is proposed

here. As in the *phenotypic mating*, *mesh mating* takes place in the decoded space. In order to allow the parents to pass on their genetic traits to their offspring, *mesh mating* involves the use of the distance and predecessor matrices.

Once two individuals $\mathbf{x}_1 = [\ n_1^1 \quad n_2^1\ ]$ and $\mathbf{x}_2 = [\ n_1^2 \quad n_2^2\ ]$ have been picked for mating, their nodes are mated pair wise[2]. That is, the first node on $\mathbf{x}_1$ is paired with the first node of $\mathbf{x}_2$ and similarly for the second node of each parent. Thus, the rest of this analysis herein only considers the first gene (node) of each solution point to describe the *mesh mating* method.

The path between the selected nodes n_1^1 and n_1^2 ($\mathbf{p}_{n_1^1 - n_1^2}$) is obtained using the method described in Section 3.1.2.b where all nodes within the path are considered to be eligible as offspring. To select the offspring, a method similar to the one described in Section 3.2.2.b is used. That is, a particular radius from one of the parents determines the maximum mesh distance between that parent and the offspring. This radius is centered around the parent with best fitness and its size would be weighted to give preference to the parent with better fitness.

Figure 3.4 shows an example where two parents (nodes 1 and 8) are mated using *mesh mating* to produce an offspring (node 9).

Due to its superior capabilities at creating offspring that inherit parent traits as well as its computational efficiency, *mesh mating* is the method that is used in the implemented GA for the combinatorial approach.

---

[2]Note that the trailing superindex in n_i^j denotes the trial point to which n_i belongs to. That is, n_i^j is the n-th node in the object's i mesh which belongs to trial point j.

3.3.2.c Mutation

A mutation of trial point $\mathbf{x}$ can be seen as nothing more than a random displacement of the trial point $\mathbf{x}$. For this purpose, the use of the *random move* described in Section 3.2.2.b is proposed here. That is, when a trial point $\mathbf{x}$ is to be mutated into $\mathbf{x}_m$, generate a second point $\mathbf{x}_{\text{rand}}$ based on $\mathbf{x}$ where one or both genes of $\mathbf{x}$ has been randomly modified to another node index. Then, find all possible offsprings of trial points $\mathbf{x}$ and $\mathbf{x}_{\text{rand}}$ and select $\mathbf{x}_m$ at random as one of their offsprings. Thus $\mathbf{x}_m$ is used as the mutated version of the original trial point $\mathbf{x}$ and replaces $\mathbf{x}$ in the population.

In order to select $\mathbf{x}_m$ from all possible offsprings of $\mathbf{x}$ and $\mathbf{x}_{\text{rand}}$, the a maximum mutation radius $r_{\max}$ can be used. In order to allow points in early generations to explore new regions of the solution space, the maximum move radius $r_{\max}$ could be set relatively large. Then, as the generation number increases, $r_{\max}$ could be scaled down using a linear function or a Boltzmann probability factor.

3.4 Examples

In order to allow a comparison between the results obtained using the combinatorial method presented in this chapter and the ones obtained with the continuous method presented the previous chapter, the same geometries and configurations were considered for the numerical examples. The objects and their configurations were described in Section 2.4. In this section, only the results from the combinatorial algorithm trials are presented.

Additionally, all GA and SA parameters were kept identical to the ones used in the

Parameter	Symbol	Value
population size	N_{pop}	50
probability of mutation	p_m	0.02
probability of crossover	p_x	0.6
max. number of generations	G_{max}	50

Table 3.1. Parameters for comb-GA-wl and comb-GA-nl runs.

Operator	Type used in example
Selection	stochastic universal sampling
Mutation	random move
Mating	mesh mating

Table 3.2. Genetic operators used in the GA runs for the combinatorial approach.

previous chapter except for the number of iterations that each algorithm is allowed to run. After a few initial trials, it was noticed that the combinatorial algorithms converged *much faster* to the solution than the algorithms using the continuous approach. Thus, the maximum number of generations G_{max} was set to 50 and the initial temperature, in the case of the SA algorithms, was set to $t_{ini} = G_{max} \times N_{pop} = 50 \times 50 = 2500$. Additionally, since the initial temperature changed, the Boltzmann constant had to be adjusted. After a number of trials, $k_B = 0.0001$ was found to give an appropriate variation of the probability factor p_b. Tables 3.1 and 3.2 summarize the GA parameters and operators used in the following examples whereas Table 3.3 summarizes the SA parameters.

Surface meshes of the objects presented in the following examples were obtained using a commercial software package called GiD [CIMNE, 2002]. To keep the memory use at a reasonable level, a relatively coarse grid was used in the following examples.

Parameter	Symbol	Value
initial temperature	t_{ini}	2,500
Boltzmann constant	k_B	0.0001

Table 3.3. Parameters for comb-SA-wl and comb-SA-nl runs.

In order to identify the type of algorithm used in a particular example, the same naming scheme as the one used in Section 2.4 is used here. To reflect the use of the combinatorial approach, the algorithms' names in this section will have the four initial letters of the approach (*i.e.*, 'comb'). That is, an algorithm such as comb-SA-nl is one that uses simulated annealing without local optimization to solve the combinatorial approach of the minimum distance problem.

As was done in the previous chapter, the test results presented in the following examples are each reported in a table listing the algorithms' name as described above. the minimum distance obtained by the algorithm and a column labeled as 'Region' that identifies whether the solution given by the algorithm is or is not located in the neighbourhood of the global solution. In addition to the row with the exact solution obtained analytically, an extra row has been added to the column showing the minimum distance between the two meshes obtained by enumeration (see Appendix D).

As outlined in Section 2.4, all the algorithms used to generate results in this work were developed in Matlab. The use of high level languages such as Matlab creates a large overhead in the computational time the algorithms take at finding a solution. Thus, an approximate flop count is presented here instead of presenting the computational time itself. With the total number of floating point operations required for a particular run, it is possible to approximate the computational time required if the algorithms were to be implemented in a lower level language such as Fortran or C. The approximation of the computational time can be performed by using published computer performance data such as that reported in [Dongarra, 2002] for a broad range of computer platforms and processor speeds.

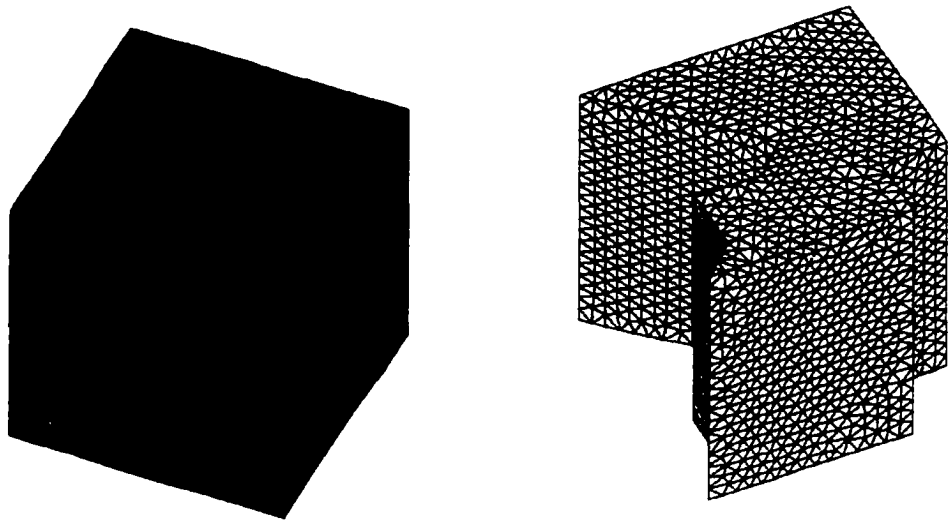


Figure 3.5. Surface meshes of the boxes with notches test case. The mesh representing the cube has 2635 nodes whereas the cube with notches has 2643 nodes.

3.4.1 Boxes with notches

The geometries for this first example were introduced in Section 2.4. Figure 3.5 shows the meshes created for the two objects. Setting the grid size to 0.1 units resulted in a cube with 2635 nodes and 5266 facets and a cube with notches with 2643 nodes and 5262 facets.

Table 3.4 summarizes the results for the four algorithms that were tried in this example. Figures 3.6 to 3.9 show the time history of the results for the current example. It is interesting to notice that in all cases, when local optimization was used, the global solution was found within the first few iterations.

3.4.2 Cylinder and semi-extruded object

As described in Section 2.4, this second numerical examples consist of a set of two objects where one is a body of revolution (cylinder with a groove) while the second one

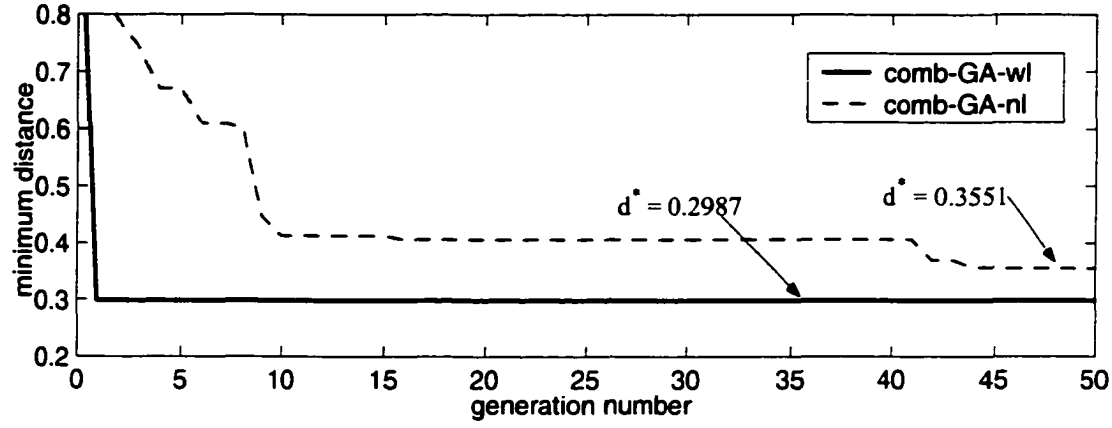


Figure 3.6. Time history of the best individual in the population for comb-GA-wl and comb-GA-nl (Geometry set 1 in position 1).

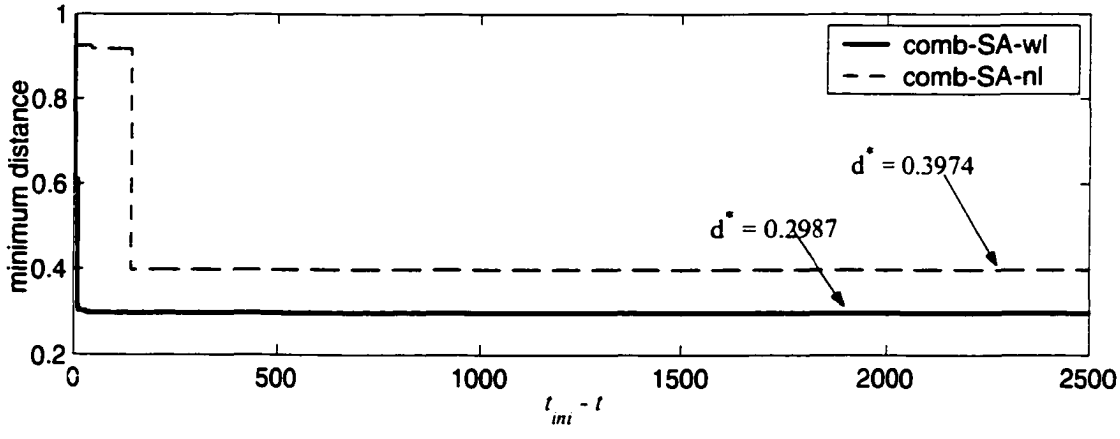


Figure 3.7. Time history of the minimum distance of the trial point for cont-SA-wl and cont-SA-nl (Geometry set 1 in position 1).

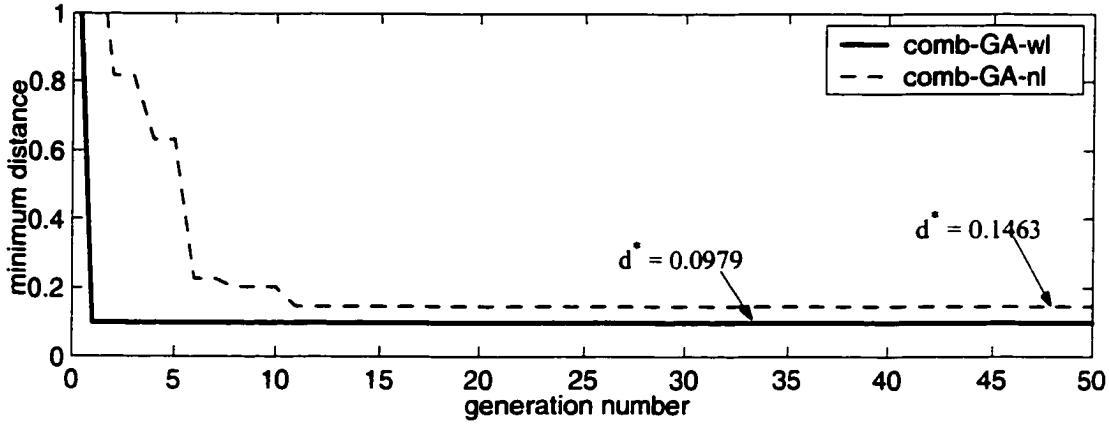


Figure 3.8. Time history of the best individual in the population for comb-GA-wl and comb-GA-nl (Geometry set 1 in position 2).

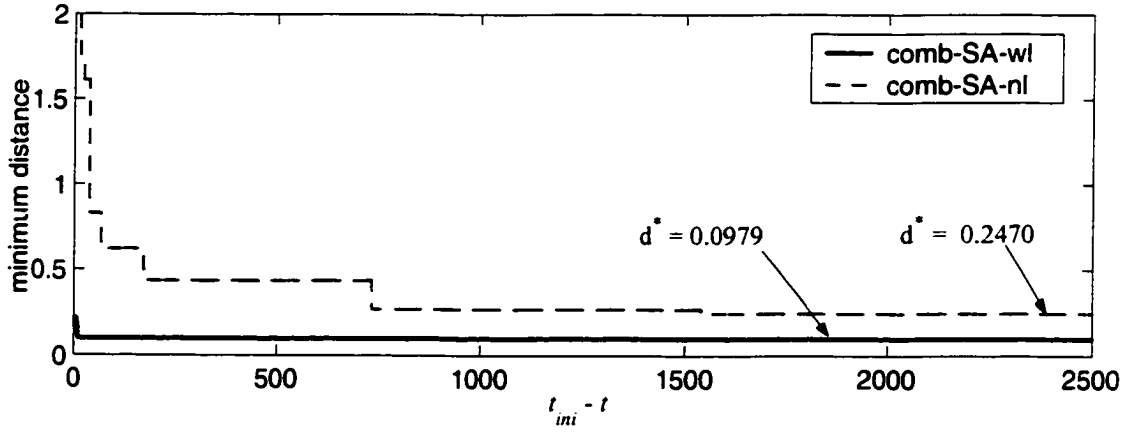


Figure 3.9. Time history of the minimum distance of the trial point for cont-SA-wl and cont-SA-nl (Geometry set 1 in position 2).

	Position 1	Region	kflops	Position 2	Region	kflops
comb-GA-wl	0.2987	✓	1 590.4	0.0979	✓	1 634.6
comb-GA-nl	0.3551	✓	138.6	0.1463	✓	135.5
comb-SA-wl	0.2987	✓	10 018.4	0.0979	✓	13 308.6
comb-SA-nl	0.3974	✓	122.5	0.2470	✓	122.9
enumeration	0.2987		181 072	0.0979		181 072
exact	0.2984			0.0979		

Table 3.4. Results for example 1 in positions 1 and 2 for algorithms: comb-GA-wl, comb-GA-nl, comb-SA-wl and comb-SA-nl.

is a more complex object with a few concavities. Figure 3.10 illustrates the objects and includes the meshes used for the numerical examples. To simplify the examples, a relatively coarse grid was used where the maximum size of the mesh elements was set to 0.1 units when generating the meshes. The cylindrical object in this case has 1101 nodes and 2198 facets whereas the semi-extruded object has 2579 nodes and 5154 facets.

The same two relative positions of the object were tried and the results are reported here for the GA and SA implementations. Additionally, the global minimum was obtained by enumeration to verify the results from the optimization algorithms.

Table 3.5 shows the results obtained for the two different relative positions of the objects. Additionally, Figures 3.11 to 3.14 illustrate the time history of the distance between the best trial point for the four algorithms and with the two objects in the two considered positions.

It is interesting to notice that, due to the stochastic nature of the optimization method, the algorithms do not always converge to the true closest point when no local optimization is used. However, in all cases the global minimum point was found whenever the local optimization was used regardless of the optimization method (*i.e.*, GA or SA). That is, when the local optimization is combined with the global method,

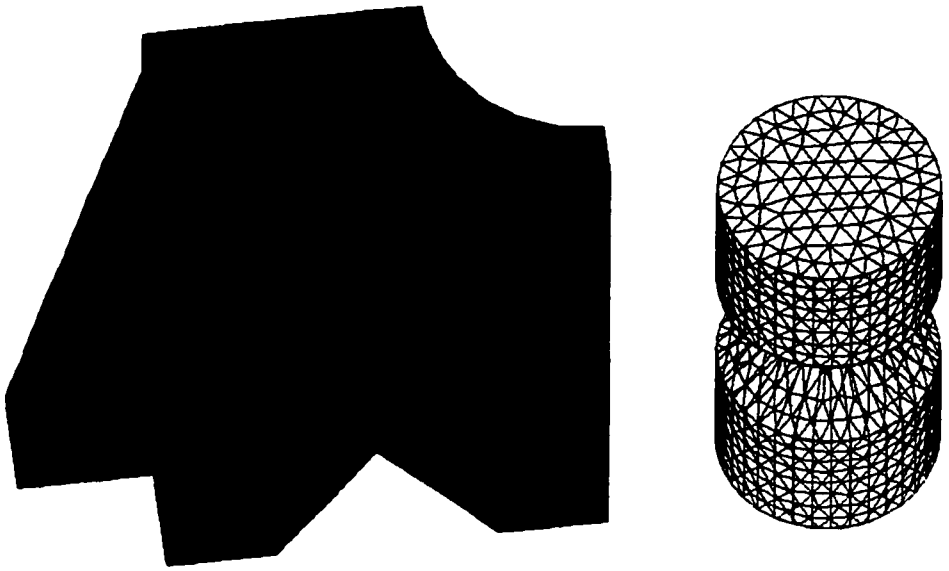


Figure 3.10. Surface meshes of the cylinder and semi-extruded object test case. The mesh representing the object of revolution has 1101 nodes whereas the more complex object has 2579 nodes.

the solution returned by the combinatorial algorithms for the examples considered here is always the same as if enumeration had been used, regardless of the initial point used in the simulation.

For all the examples considered in this chapter, algorithm comb-GA-wl converged to the global minimum within the first 2 generations. Additionally, algorithm comb-SA-wl also converged to the global minimum for all the numerical examples presented here. In most cases, comb-SA-wl converged to the global solution within the first 100 iterations.

It is important to notice that, although in the present examples algorithm comb-GA-wl was capable of finding the global solution in the first iteration, this is not be the case in more complicated scenarios. Thus the use of GAs, rather than multi point local search, will bolster the confidence in finding the global solution.

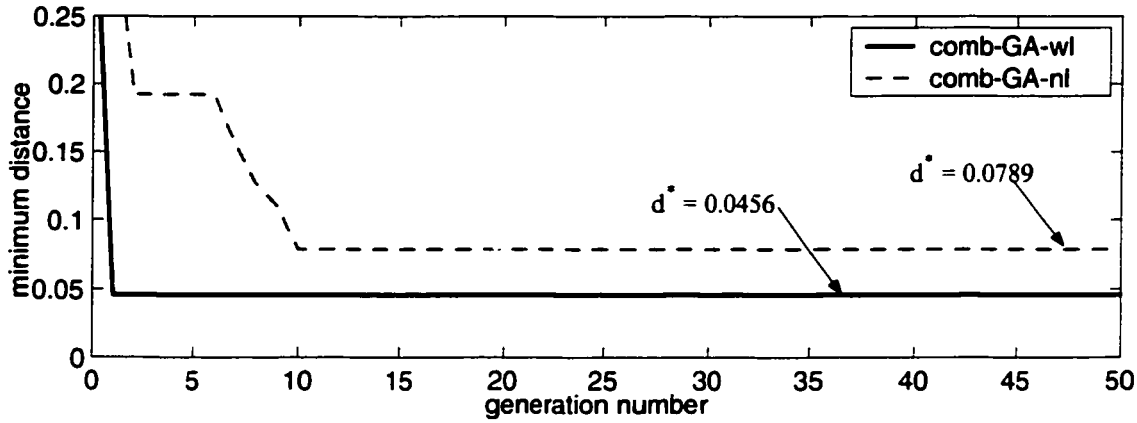


Figure 3.11. Time history of the best individual in the population for comb-GA-wl and comb-GA-nl (Geometry set 2 in position 1).

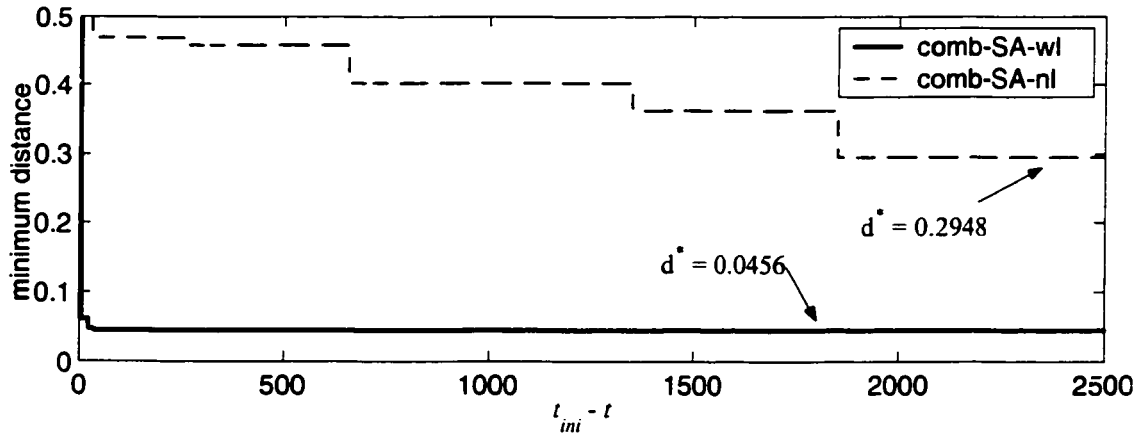


Figure 3.12. Time history of the minimum distance of the trial point for cont-SA-wl and cont-SA-nl (Geometry set 2 in position 1).

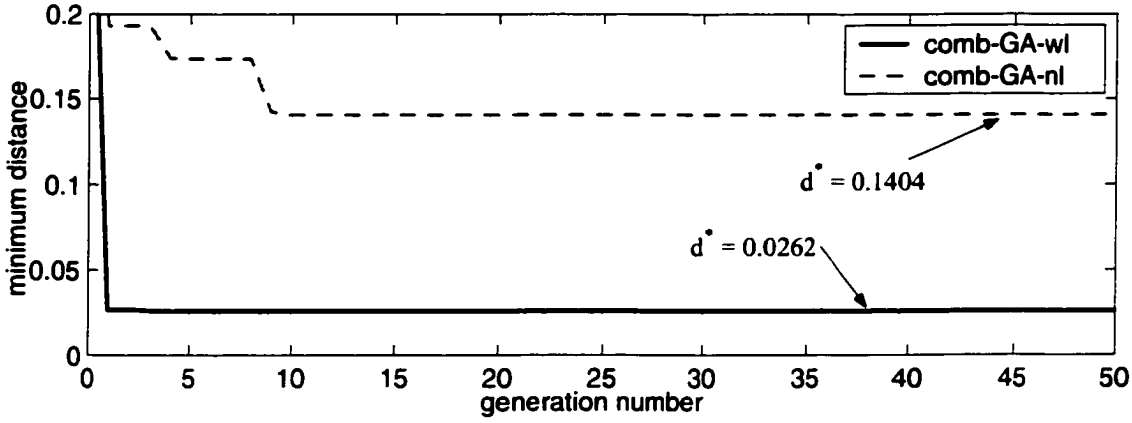


Figure 3.13. Time history of the best individual in the population for comb-GA-wl and comb-GA-nl (Geometry set 2 in position 2).

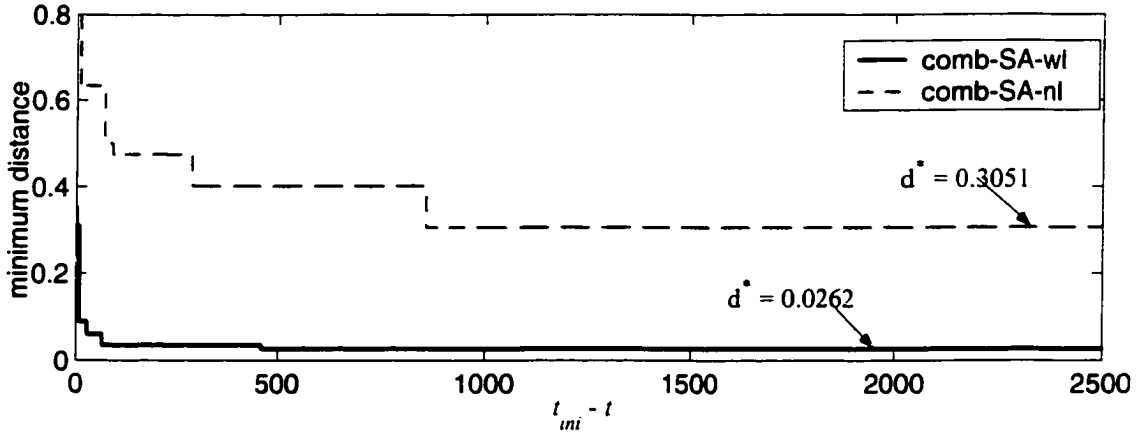


Figure 3.14. Time history of the minimum distance of the trial point for comb-SA-wl and comb-SA-nl (Geometry set 2 in position 2).

	Position 1	Region	kflops	Position 2	Region	kflops
comb-GA-wl	0.0434	✓	1 496.7	0.0262	✓	1 671.1
comb-GA-nl	0.0789	✓	133.5	0.1404	×	133.4
comb-SA-wl	0.0434	✓	8 454.1	0.0262	✓	8 619.5
comb-SA-nl	0.2948	✓	123.3	0.3051	×	123.6
enumeration	0.0434		73 826	0.0262		73 826
exact	0.0394			0.0114		

Table 3.5. Results for example 2 in positions 1 and 2 for algorithms: comb-GA-wl, comb-GA-nl, comb-SA-wl and comb-SA-nl.

Chapter 4

Multiple Solution Points

The methods described in earlier chapters allow the calculation of the minimum distance between two convex or concave objects. On the other hand, in situations where two or more points of the objects in question might come in contact (see Figure 4.1), the previous algorithms would only return a single solution. For example, consider a case in which two trial points, namely $\mathbf{x}_1$ and $\mathbf{x}_2$, have exactly (or very close to) the same fitness ($d_1 \approx d_2$) and initially have exactly the same number of trial points in their region or neighborhood (Γ_1 and Γ_2, respectively), *e.g.* 20 points in each region (see Figure 4.1b). As generations pass and due to the random nature of the selection, mating and mutation operations, one of the regions, namely Γ_1, can end up with one extra trial point relative to the other, that is 21 in $\Gamma_1$versus 19 in Γ_2. At this point, the probability of having a point from Γ_1 selected becomes greater and thus more genetic material from this region will go through to the next generation. If this process is repeated, after a few generations all the genetic material belonging to region Γ_2 will eventually disappear. In GA parlance this phenomenon is known as *genetic drift* and is accentuated when small population sizes are used [Goldberg, 1989a]. These

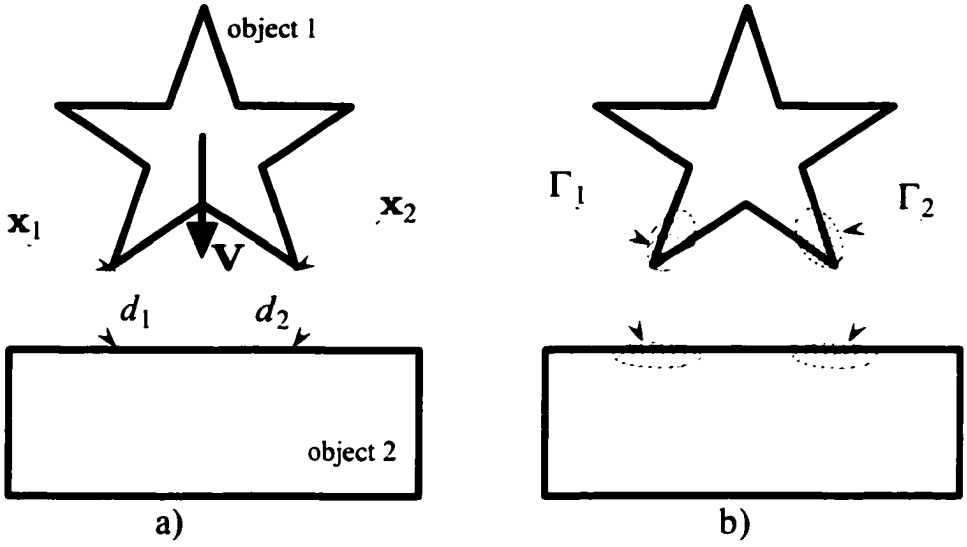


Figure 4.1. Object configuration with two solutions points ($\mathbf{x}_1$ and $\mathbf{x}_2$) and regions Γ_1 and Γ_2 assosciated to solution points $\mathbf{x}_1$ and $\mathbf{x}_2$.

problems can be solved by using large population size or by a carefully designed genetic algorithms using the concept called niche formation described in detail in this chapter.

4.1 Relevance to contact dynamics applications

As explained in Section 2.1.6, in contact dynamics as well as other applications, the minimum distance problem is solved repeatedly over time as the simulation progresses. In these cases, while using an optimization based approach, the initial guess in the first evaluation of the minimum distance algorithm is generated at random (or arbitrarily set as a particular point such as the object's origin [Ma and Nahon, 1992]) and in all subsequent evaluations the solution of the previous evaluation of the minimum distance algorithm is used as the initial guess [Ma and Nahon, 1992]. Some authors have even proposed the use of velocity information in order to better determine an

initial guess [Nahon, 1993]. On the other hand, in complicated scenarios, the solution can easily jump from one region of the object to another (*e.g.*, peg and hole insertion).

Consider the example in Figure 4.1a where object 1 is moving towards object 2 with a velocity **V** and zero angular velocity. Assume that object 1 was slightly tilted clockwise before the first impact, that is, $d_2 < d_1$. Thus it would be correct to assume that the minimum distance algorithms as described in earlier chapters would return $\mathbf{x}_2$ as the solution to the minimum distance problem. After the first impact, and assuming the gravity is acting downwards, object 1 would bounce and rotate counter-clockwise around its axis to make a second contact with object 2, probably contacting at a point in the region Γ_1. Thus, between collisions, the minimum distance algorithm would use the solution of the previous time step as initial point for the next iteration, *i.e.*, $\mathbf{x}_2$. This initial guess is essentially wrong since it is no longer in the region were the global minimum is found and the distance algorithm would be capable of escaping from the local minimum region around $\mathbf{x}_2$ in order to find the new global minimum $\mathbf{x}_1$.

For this reason, the capability of being able to track not only the global minimum solution but also most other local minima would allow the distance algorithm to keep track of evolving minima rather than obtaining only the global minimum every time the distance algorithm is evaluated.

Additionally, in the context of contact dynamics, it is essential to consider the possibility that more than a single contact point may exist when two objects collide with each other. Thus, enabling the distance algorithm to return points in multiple minimum regions is essential. That is, in the example presented in Figure 4.1 where object 1 approaches object 2 with $d_1 = d_2$, the minimum distance algorithm should return a point in region Γ_1 **and** one in region Γ_2, *i.e.*, solution points $\mathbf{x}_1$ and $\mathbf{x}_2$, as

possible contact points.

4.2 Niche formation in GAs

In trying to mimic the natural process of speciation, a few different methods have been created in GAs for multimodal function optimization. As many other concepts in GAs, the first to introduce the notion of speciation was Holland [Holland, 1975]. In his work, Holland illustrates the concept using the two-armed bandit. In this example, a two-armed bandit with equal payoffs (namely 1) but different probabilities of payoff is used, namely p_{right} and p_{left} for the right arm and left arm, respectively. When one looks to maximize the payoff per individual one can divide the population of N_{pop} individuals (*i.e.*, trial points) into two groups (m_{right} and m_{left}) and assign each group to one arm of the bandit. Then, each individual in group i (that is, *right* or *left*) is forced to share the payoff from its arm with the other $m_i - 1$ individuals in its group. Then, there is a balance where the population can be divided in order to have all N_{pop} individuals with the same payoff no matter which group they belong to or which payoff probability their side of the two-armed bandit has. This balance is perfect when $\frac{p_{right}}{m_{right}} = \frac{p_{left}}{m_{left}}$. In some cases this perfect balance has to be approximated due to the fact that m_{right} and m_{left} have to be integers. For example, if 20 individuals constitute the entire population and $p_{right} = 1/5$ and $p_{left} = 4/5$ then, by locating 4 individuals on the right arm and 16 on the left arm (*i.e.*, $m_{right} = 4$ and $m_{left} = 16$), one would expect that, after a large number of trials of both arms of the bandit, the same payoff would be received by all individuals.

Many niche formation methods exist, each with many variations. Most of these methods can be categorized in two main classes: *sharing methods* (such as the one

proposed in [Goldberg and Richardson, 1987]) and *crowding methods* (*e.g.* [Cavicchio, 1970, De Jong, 1975])[1]. In order to promote the formation of niches, sharing methods usually affect the fitness of the individuals whereas crowding methods modify the mating and/or selection steps. On the other hand, methods such as *multi-niche crowding* [Vemuri and Cedeño, 1995], use a mix of ideas between crowding and sharing in order to achieve a greater diversity in the population and to allow more niches to be formed.

In addition to the niching method described above, *mating restriction* techniques [Booker, 1982] are used in order to promote speciation and to reduce the production of lethals [Beasley et al., 1993b]. Mating restriction refers to the use of a particular criteria (usually a proximity criteria) that both parents need to meet if they are to be mated.

4.3 Sharing methods

Sharing methods use a concept similar to the one introduced by Holland [Holland, 1975]. In practice, a fitness value derating function is used which decreases the fitness value of an individual proportionally to its closeness to other individuals in the population [Booker, 1982, Goldberg and Richardson, 1987]. To achieve this, it is first necessary to establish a sharing radius which determines the size of a niche. Then, all the individuals in that region will share the same value of the adjusted fitness. Sharing methods have successfully been used in areas such as machine learning [Booker, 1982], multimodal function optimization [Goldberg, 1989a] and financial ap-

[1] For a thorough comparison of these methods refer to [Deb and Goldberg, 1989] and [Sareni and Krähenbühl, 1998].

plications [Chopard et al., 1995]. In this work, it is proposed to use sharing methods in order to obtain more than a single solution point.

As described earlier, sharing methods make use of a sharing function to determine the degradation of an individual's original fitness f_i (*a.k.a.* potential fitness) due to other individuals in the same region. The sharing fitness f_i' value can be calculated as a function of the original fitness value f_i as follows [Goldberg and Richardson, 1987]:

$$f_i' = \frac{f_i}{m_i'} \tag{4.1}$$

where m_i' refers to the niche count which is a sum of all share function values (and not the number of individuals in a particular niche). For a trial point i, the niche count is calculated as

$$m_i' = \sum_{j=1}^{N_{pop}} S(\Delta(\mathbf{x}_i, \mathbf{x}_j)) \tag{4.2}$$

where N_{pop} is the total number of individuals in the population and $S(\Delta)$ refers to the sharing function. In equation (4.2), $\Delta(\mathbf{x}_i, \mathbf{x}_j)$ refers to the distance, phenotypic or genotypic[2], between trial points $\mathbf{x}_i$ and $\mathbf{x}_j$. Note that, in equation (4.2), the sum includes the trial point itself. Thus if a trial point is all by itself in its own niche ($m_i' = 1$), it receives its full potential fitness value.

It is proposed to use the following power law function to determine the individual's

[2] *Phenotypic* distance is measured in the decoded space, whereas *genotypic* distance is measured in the coded space, *i.e.* the strings themselves. In the context of the work presented here, phenotypic distance can refer to the Cartesian distance between to trial points whereas the genotypic distance, such as the Hamming distance, refers to how different the two strings are in the coded space [Deb and Goldberg, 1989].

sharing value:

$$S(\Delta) = \begin{cases} 1 - \left(\frac{\Delta}{\sigma_s}\right)^{\alpha_s}, & \Delta < \sigma_s \\ 0, \quad \text{otherwise} \end{cases} \tag{4.3}$$

where σ_s and α_s are user defined constants. When $\alpha_s = 1$ it is called a triangular sharing function. σ_s is the maximum distance considered for two trial points to share a region (*a.k.a.* proximity threshold).

Additionally, the expected number of individuals at a particular niche can be calculated as

$$N_{niche_i} = \frac{N_{pop} f_{niche_i}}{\sum_{j=1}^{N_{pop}} f_{niche_j}} \tag{4.4}$$

where N_{pop} is the population size and f_{niche_i} corresponds to the potential fitness at niche i.

4.4 Crowding methods

Crowding methods borrow the idea of crowding from nature. That is, the individuals compete for a limited number of spots in the niche they belong to. In these methods it is the selection process of the GA that is modified by only allowing a certain population density.

Since duplicate genetic trends are avoided, crowding methods enforce population diversity. Some practical application of crowding methods in function optimization range from classifier systems in machine learning [Goldberg, 1989a] to the display formats for terminal area traffic control [Grunwald, 1999].

4.4.1 Simple crowding

In the most basic form of crowding, developed by De Jong [De Jong, 1975] after the idea of preselection [Cavicchio, 1970][3], each offspring is compared to a random subset of the population[4].

1. A subpopulation of C_f individuals is drawn at random from the original population. Typically, $C_f = 2$ or 3.

2. Obtain distance (phenotypic or genotypic[5]) between offspring and all the elements in subpopulation.

3. Replace closest individual in subpopulation by the offspring.

4. Repeat steps 1 to 3 for second offspring (second offspring can potentially replace first offspring).

4.4.2 Deterministic crowding

Crowding methods, as described earlier, promote speciation by eliminating similar individuals. That is, each new offspring $\mathbf{x}_{\text{off}}$ is compared to the closest individual ($\mathbf{x}_{\text{closest}}$) in the entire population or a portion of it. The individual of these two (*i.e.*, $\mathbf{x}_{\text{off}}$ and $\mathbf{x}_{\text{closest}}$) having the best fitness goes through the next generation while

[3]In preseselection [Cavicchio, 1970], each offspring replaces its most inferior parent if its fitness is better than of its inferior parent.

[4]In many practical applications, that subset is the parents themselves [Goldberg, 1989a].

[5]Genotypic distance is the distance between two solution points measured in the coded space, that is the distance between genes. Phenotypic distance, on the other hand, is the distance between two individuals in the solution space.

the other is eliminated from the population [De Jong, 1975]. It is worth noticing that, to search all the population for the closest individual for each offspring can be computationally expensive.

To alleviate this computational burden, *deterministic crowding* [Mahfoud, 1992] makes use of the fact that the individuals in the population that most likely have more similarity to the offspring $\mathbf{x}_{\text{off}}$ are its parents. For this reason, *deterministic crowding* only uses a subset of the entire population, that is, the parents of the particular offspring being analyzed. That is, each offspring's fitness is compared to the fitness of the closest parent[6]. If the offspring's fitness is better than its parent's, it replaces the parent. Otherwise, the offspring is eliminated and the parent goes untouched to the next generation.

4.5 Mating restriction

In order to eliminate the creation of points that do not belong to any particular minimum region (*a.k.a.* lethals), a few authors have proposed the use of mating restriction [Booker, 1982, Goldberg, 1989a]. When mating restriction is used, only parents that satisfy a certain mating criterion are allowed to produce offspring. When two parents are selected for mating, they are first checked if they meet the mating criterion. If they do, the mating process continues; otherwise, a new set of parents is made. This process is repeated until two suitable parents are found.

One such methods is *phenotypic mating restriction.* In this method, a threshold σ_m is established which defines the maximum phenotypic distance between any two parents. This means that if the two parents are within a distance σ_m of each other

[6] Phenotypic or genotypic distance can be used to measure the distance between individuals.

then they are mated, otherwise one of the parents is replaced. Then, the check for suitability is performed again until a parent within the mating radius σ_m of the first parent is found. For example, in Figure 4.2, mating two parents each belonging to regions Γ_1 and Γ_2 will likely produce offspring in the region labeled as Γ_3 which is not a local minium. On the other hand, selecting sets of parents belonging both to the same region (namely Γ_1 or Γ_2) would guarantee that the region the parents belong to is represented by at least one individual in the next generation.

Looking for a suitable set of parents can considerably slow the mating process. For this reason, some authors have proposed to try to find a suitable mate for any parent only twice, otherwise, pick a third candidate as mate randomly from the population without verifying the mating criterion [Deb and Goldberg, 1989].

Other mating methods exist where the distance metric is measured as the genotypic distance. One such distance is the one defined as the Hamming distance [Bäck, 1996]. In a binary encoding of a GA where each encoded trial point consists of a string of ones and zeros, the Hamming distance refers to the number of bits in a string that makes two strings different.

The use of mating restriction methods, particularly using phenotypic distance as the distance metric, within sharing methods has proven to improve the GA performance in function optimization applications [Deb and Goldberg, 1989].

4.6 Implementation in the combinatorial approach

Earlier in this chapter, the use of niche formation in GAs in order to allow the algorithms to obtain multiple solution points was explained. Due to its computational simplicity, the combinatorial approach to the minimum distance problem was chosen

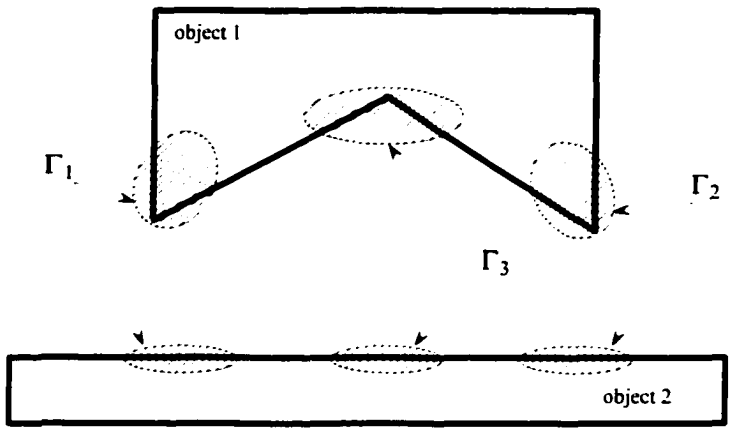


Figure 4.2. Offsprings having one parents in region Γ_1 and the other in region Γ_2 have a great chance of being created in a poor region such as Γ_3.

to implement the niche formation algorithms.

Here, some implementation details about two of the niching methods described earlier are presented, namely sharing and deterministic crowding methods. Additionally a mating restriction algorithm was implemented and some of its implementation details are described here.

4.6.1 Sharing

Sharing methods in GAs affect the fitness of all the individuals in the population in order to reflect the number of individuals that share a particular region of the search space. Thus, if the mesh distance is used to determine the proximity of a particular point to its neighbours, the sharing method would be called *mesh sharing*. Mesh sharing is similar in concept to *phenotypic sharing* in that the distance metric is measured in the decoded space. The difference, as outlined in Section 3.3.2.b (Chapter 3) when *phenotypic mating* and *mesh mating* were described, revolves around the method used to define the path between two points. In the *phenotypic* case, the path

is a straight line between the two points whereas in the *mesh* case it is the path along the mesh edges that is used to measure the distance.

Sharing methods act on the GA as an extra block inserted between the function evaluation and the selection methods (see Figure 4.3).

For a trial point $\mathbf{x}_i$, the niche count is calculated using equation (4.2) where in the present case $\Delta(\mathbf{x}_i, \mathbf{x}_j)$ refers to the total mesh distance, between trial points $\mathbf{x}_i$ and $\mathbf{x}_j$. In Chapter 3, the mesh distance δ was defined as the distance along the mesh edges between two nodes on the same mesh. In the present case the total mesh distance Δ_{ij} between two trial points $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as the sum of the mesh distance between the two pairs of nodes, that is

$$\Delta_{ij} = \Delta(\mathbf{x}_i, \mathbf{x}_j) = \delta_{n_1^i - n_1^j} + \delta_{n_2^i - n_2^j} \tag{4.5}$$

Thus, the niche count m_i' is calculated as

$$m_i' = \sum_{j=1}^{N_{pop}} S(\Delta(\mathbf{x}_i, \mathbf{x}_j)) \tag{4.6}$$

where N_{pop} is the population size and the sharing function $S(\Delta)$ that was implemented is the same power law function described by equation (4.3). The sharing radius σ_s in equation (4.3) in this implementation refers to the maximum total mesh distance (Δ) considered for two individuals to share a region. The influence of factors σ_s and α_s (equation (4.3)) on the sharing method will be studied in Chapter 5.

4.6.2 Deterministic crowding

As described earlier, in *deterministic crowding*, each offspring's fitness is compared to the fitness of the closest parent. If the offspring fitness is better than that of its

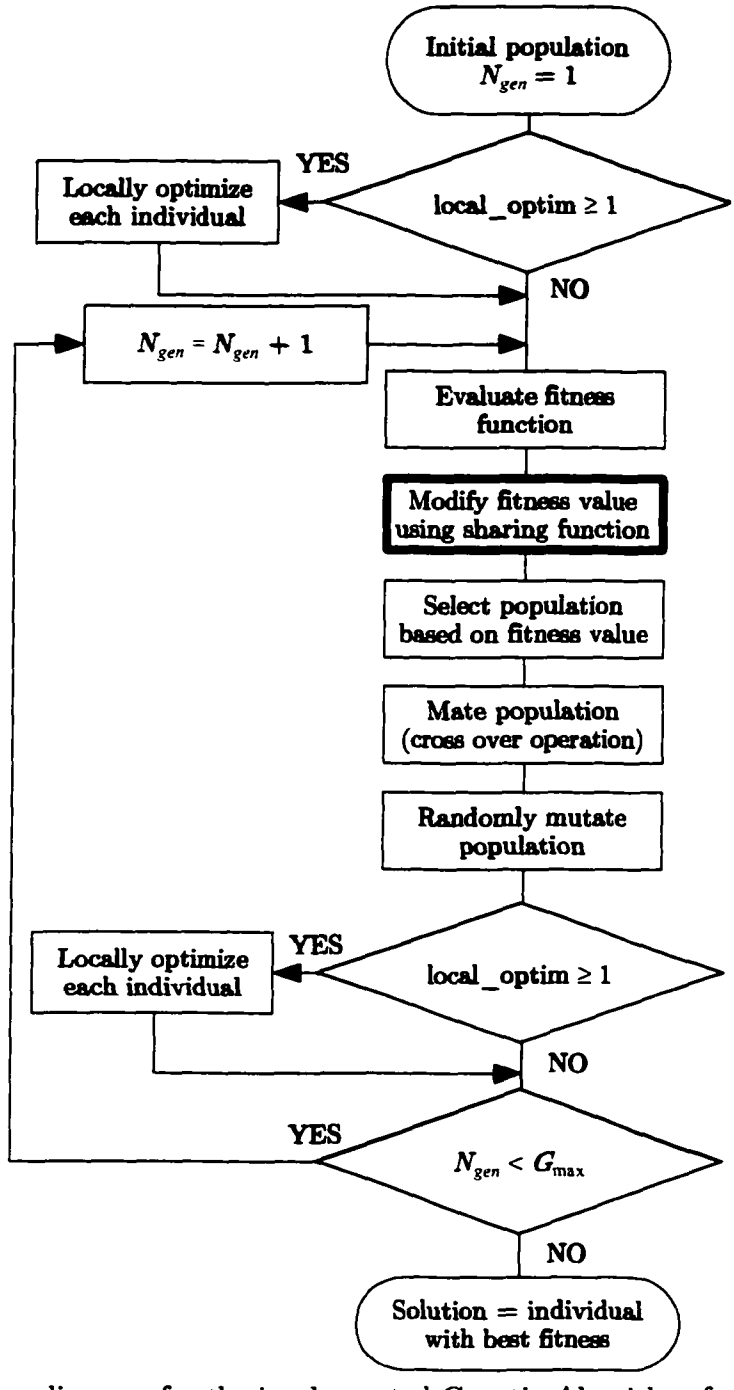


Figure 4.3. Flow diagram for the implemented Genetic Algorithm for the combinatorial approach with sharing.

closest parent it is the offspring that goes through to the next generation, otherwise, it is the parent that remains in the population.

In the present study, deterministic crowding was implemented for the combinatorial approach to the minimum distance problem. The measure of the distance between the offspring and its parent is evaluated as the total mesh distance Δ described in equation (4.5). As described earlier, the total mesh distance Δ is close to the phenotypic distance since it is the distance measured in the decoded space.

In order to implement the deterministic crowding method into the combinatorial approach, an additional block is added to the genetic algorithm described in section 2.3 and illustrated in Figure 2.13. The deterministic crowding block is added between the mating and mutation operations as described in Figure 4.4.

4.6.3 Mating restriction

In the combinatorial minimum distance determination method described in Chapter 3, *mating restriction* can be implemented using a mating radius σ_m. That is, the parents have to lie within a distance σ_m of each other in order to be allowed to mate otherwise, a new couple that meets this criteria is sought. For computational efficiency, the distance between parents is measured along the mesh edges. That is, the total mesh distance Δ defined in equation (4.5) is used to measure the distance between two candidate parents.

When mating restriction is used within the sharing methods, it is suggested to use a mating radius σ_m larger than the sharing radius σ_s, *i.e.*, $\sigma_m > \sigma_s$. This will allow the points that are marginally outside a niche to converge to the closest niche improving the overall convergence of the GA.

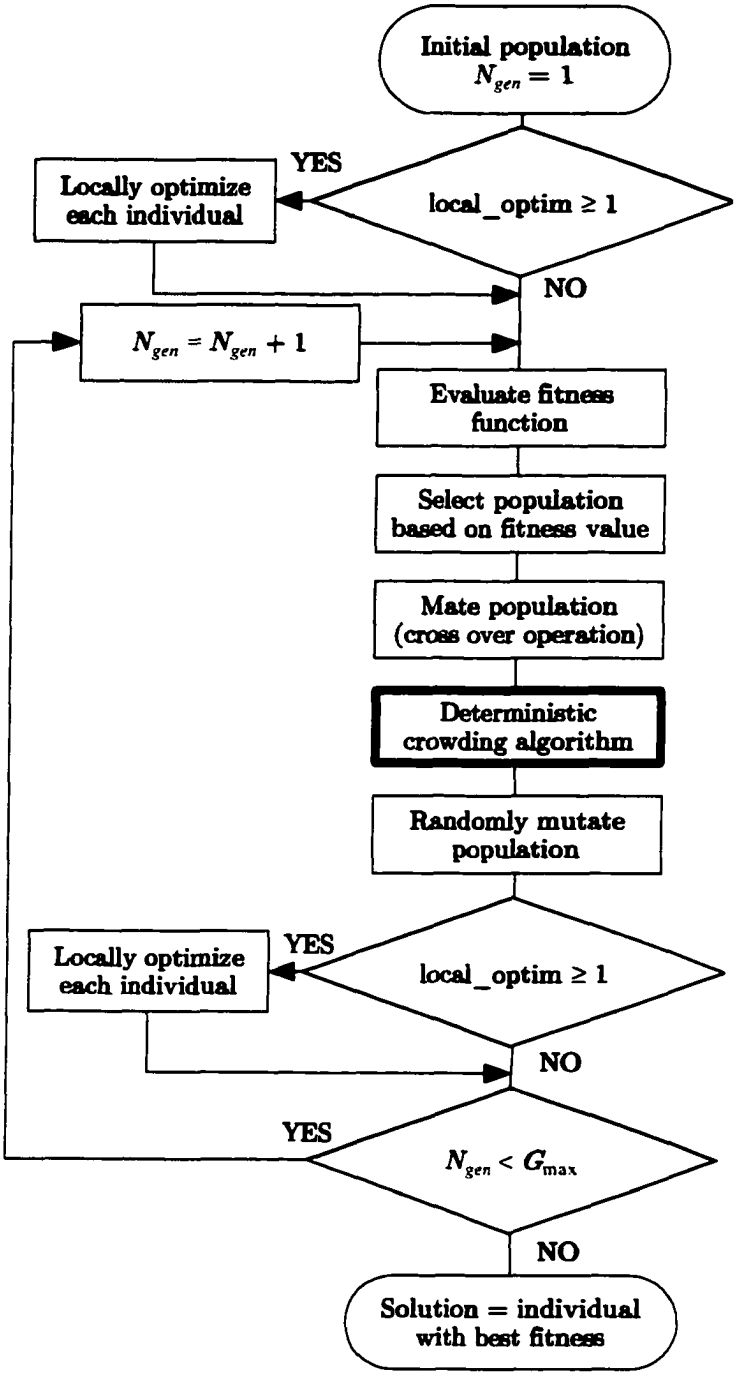


Figure 4.4. Flow diagram for the implemented Genetic Algorithm for the combinatorial approach with deterministic crowding.

4.6.4 Identifying niches

The output from the minimum distance algorithm consists of the best solution point in the population and its fitness or its corresponding distance. Additionally, the entire final population and the fitness of each individual in the population is obtained. Thus, once the final population is obtained, it is necessary to identify the different niches. Many methods exist in the literature regarding the identification of clusters amongst a number of data points (see for instance [Hartigan, 1975, Goldberg, 1989a]). These methods include the minimum spanning tree, k-means clustering and fuzzy c-means clustering.

In the present work, a simple greedy niche identification technique is proposed similar to the one described in [Miller and Shaw, 1996]. Initially, the solution point with the smallest fitness ($\mathbf{x}_{\Gamma_1}$) is selected. Next, the distance between $\mathbf{x}_{\Gamma_1}$ and all the other individuals in the population is computed using equation (4.5). Then, all points in the population that lie within a niche radius[7] (σ_n) from $\mathbf{x}_{\Gamma_1}$ are taken as part of the niche Γ_1. Finally, the points in niche Γ_1 are counted and removed from the original population, that is, the population is stripped of all the points lying in niche Γ_1. The search for the trial point $\mathbf{x}_{\Gamma_2}$ with the smallest fitness in started again, this time using the reduced population. The process continues until the population is empty and all N_Γ niches have been found. The outcome would then be a matrix with all Γ_i niches which will have N_Γ rows where row i represents the trial point with minimum distance in niche i.

In order to reduce the number of user-defined parameters, the niche radius (σ_n)

[7]The niche radius σ_n used in this work is measured along the mesh edges. In practice the niche radius σ_n could be set equal to sharing radius σ_s or the mating radius σ_m.

can be determined as a function of the overall size of the object or the maximum mesh distance between any two protrusions on a single object.

4.7 Examples

This section presents a few examples created using the combinatorial GA with niche formation methods. The geometries considered for the first two examples are identical to the ones considered in previous chapters. Here, only one pose of the objects is considered.

In addition to the examples seen in previous chapters, a third set of geometries is presented here. The third example represents an Orbital Replacement Unit (ORU) Battery and its fixture. This example is considered to best illustrate the concept of obtaining multiple solution points in complicated scenarios.

All the examples presented in this chapter were created using two different niche formation techniques: sharing and deterministic crowding. In both cases, the global solution is sought as well as other possible local minima. In order to report the solution, the number of trial points in each niche was counted and is reported together with the niche's fitness.

As seen in Section 3.4, the combinatorial approach of the minimum distance algorithms performed best while using the local optimization algorithm. That is, for the examples presented there, algorithm comb-GA-wl outperformed all other variations of the combinatorial method. For this reason, Lamarckian type of evolution is considered for all the following examples (*i.e.*, all algorithms used here make use of local optimization at every trial point).

Additionally, in order to promote the creation of niches, mating restriction is used

with the sharing method. The mating radius σ_m used for each example was determined by experimentation and was found to be a function of the overall dimensions of the objects and their geometries. That is, if two niches are foreseen to occur near each other, the mating radius σ_m has to be smaller than half the distance between the two niches. For each example, the mating radius σ_m is also reported.

The algorithm described in Section 4.6.4 to identify niches was used with the following examples. Like the mating radius σ_m, the niche radius σ_n was determined by experimentation and was also found to be a function of the objects' geometries and their overall dimensions. The results for the examples presented here show the location of the niches as well as their fitness and the number of points at each niche.

In the following examples, the naming technique used in previous chapters is also used. The algorithms used for these examples will have two extra characters at the end of the algorithm's name to identify the niching method. For instance, 'sh' will be used to identify sharing method while 'dc' will be used to identify deterministic crowding. That is, comb-GA-wl-sh is a combinatorial genetic algorithm with local optimization and sharing.

4.7.1 Boxes with notches

The geometry and pose used in this example are identical to the ones presented in Section 3.4 for the first example in position number 1 (see Figure 4.5). That is, the meshes for the box and box with notches were created using a maximum grid size of 0.1 units resulting in a cube with 2635 nodes and 5266 facets and a cube with notches with 2643 nodes and 5262 facets (see Figure 3.5). All other parameter values are listed in Table 4.1. Where applicable, the parameter values and operators were kept

Parameter / Operator	Symbol	Value / Type
population size	N_{pop}	50
probability of mutation	p_m	0.02
probability of crossover	p_x	0.6
max. number of generations	G_{max}	50
mating radius (phenotypic)	σ_m	0.4
sharing radius	σ_s	0.25
sharing exponent	α_s	1
niche radius (only for sharing)	σ_n	1.5
selection		SUS
mating		mesh mating
mutation		random move
local optimization		Lamarckian evolution

Table 4.1. Parameters for comb-GA-wl and comb-GA-nl runs.

identical to the ones used in the numerical examples in Chapter 3.

In order to identify the location of all niches, Figure 4.5 shows three regions where the local minimum points are to be found in order of proximity, that is, region 1 corresponds to the global minimum and regions 2 and 3 to local minima.

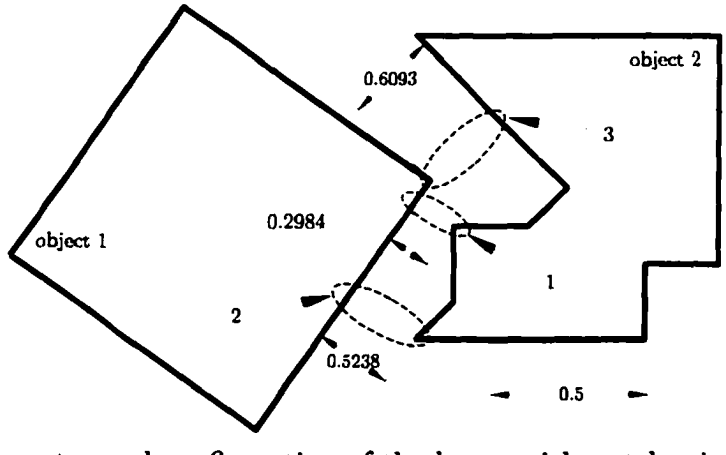


Figure 4.5. Geometry and configuration of the boxes with notches in position 1.

Figure 4.6 shows the location of the niches at the last iteration of the minimum distance algorithm. In both cases the global minimum was found within the first two iterations. Algorithm comb-GA-wl-sh found a total of 6 niches whereas comb-

Niche number	Region number	Distance	Number of individuals
1	1	0.2987	20
2	1	0.2987	16
3	1	0.2991	7
4	2	0.5256	2
5	2	0.5245	2
6	3	0.6096	3

Table 4.2. Results for the boxes with notches example using comb-GA-wl-sh. The region number refers to the regions identified in Figure 4.5.

Niche number	Region number	Distance	Number of individuals
1	1	0.2991	46
2	1	0.3022	2
3	3	0.6096	1
4	other	1.9077	1

Table 4.3. Results for the boxes with notches example using comb-GA-wl-dc. The region number refers to the regions identified in Figure 4.5.

GA-wl-dc only found 4 of which one is a poor solution point. Tables 4.2 and 4.3 give more details about the regions, shown in Figure 4.5, where the niches were found for algorithms comb-GA-wl-sh and comb-GA-wl-dc, respectively. The exact distance for each of the niches can be verified in Figure 4.5. Algorithm comb-GA-wl-sh took close to 2 Mflops to solve the problem whereas comb-GA-wl-dc only took 1.6 Mflops.

In the present example, more than a single niche was found around the same region, particularly for comb-GA-wl-sh. This is due to the fact that the objects are three dimensional objects extruded in the direction perpendicular to the plane of the page. Thus,a the niches are separated from each other in the direction perpendicular to the plane of the page.

In this first example, algorithm comb-GA-wl-sh seems to perform better in finding the niches than algorithm comb-GA-wl-dc.

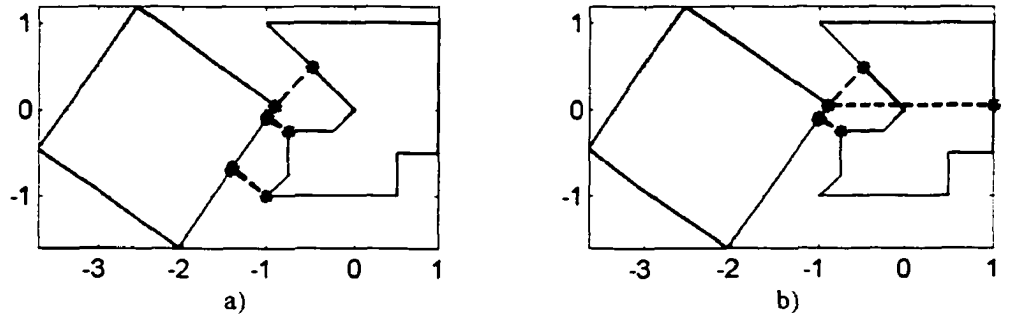


Figure 4.6. Example of the comb-GA-wl using a) sharing and b) deterministic crowding in the boxes with notches case.

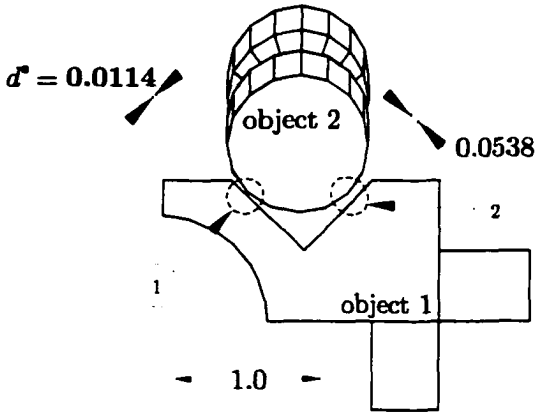


Figure 4.7. Cylinder and semi-extruded object in position 2.

4.7.2 Cylinder and semi-extruded object

For this example, the geometry is identical to the one presented in the second example in Section 3.4. The mesh representing the cylinder object has 962 nodes and 1800 facets whereas the one representing the semi-extruded object has 2579 nodes and 5154 facets (see Figure 3.10). Both meshes were created in GiD with a maximum grid size of 0.1 units. In the present example, it is the second relative position of the objects that is presented here, that is, the objects are positioned as illustrated in Figure 4.7

The GA parameters and operators used in this example are identical to the ones listed in Table 4.1 for the previous example except for the niche radius σ_n which was

Niche number	Region number	Distance	Number of individuals
1	1	0.0345	40
2	2	0.0898	5
3	other	0.1417	4
4	other	0.3816	1

Table 4.4. Results for cylinder and semi-extruded object example using comb-GA-wl-sh. Region numbers refer to regions identified in Figure 4.7.

Niche number	Region number	Distance	Number of individuals
1	1	0.0317	48
2	other	0.4071	1
3	other	0.1417	1

Table 4.5. Results for cylinder and semi-extruded object example using comb-GA-wl-dc. Region numbers refer to regions identified in Figure 4.7.

set to 2 for the present example.

As was the case in the previous example using the combinatorial GA with local optimization, the global minimum was found within the first 2 generations of the GA run. Additionally, as illustrated in Figure 4.8, comb-GA-wl-sh was able to locate the two regions were the minimum solutions are found.

Figure 4.8 illustrates the population at the last generation for both algorithms. In the present example, comb-GA-wl-sh ended the simulation with 40 trial points in region 1 and 5 in region 2 whereas comb-GA-wl-dc found 48 and 0, respectively. Tables 4.4 and 4.5 list these niches and others found by algorithms comb-GA-wl-sh and comb-GA-wl-dc, respectively.

Also in this example it is possible to see that the GA using sharing method is better at keeping track of multiple minima compared to the GA using deterministic crowding. Algorithm comb-GA-wl-sh took close to 2.1 Mflops to solve the problem whereas comb-GA-wl-dc only took 1.7 Mflops.

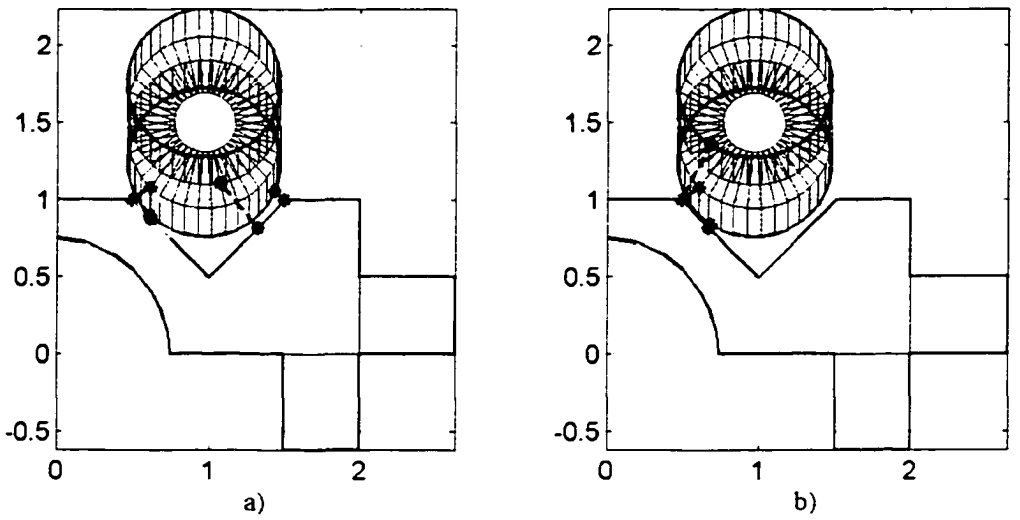


Figure 4.8. Example of the GA using a) sharing and b) deterministic crowding on the cylinder and semi-extruded object case.

4.7.3 Battery and fixture

In this example, more complicated geometries are used. The objects shown in Figures 4.9 and 4.10 illustrate a simplified version of the ORU battery and its fixture of the International Space Station [Nahon et al., 1998].

Surface meshes were created from the two objects using a commercial grid generator called GiD [CIMNE, 2002]. The battery and fixture meshes have a grid size of 0.05 units resulting in 1241 nodes for the battery and 1842 nodes for the fixture (see Figure 4.11).

In the present example, most parameters and operators were kept identical to the ones presented in Table 4.1 for the first example in this chapter. Due to the complexity of the geometries, the number of individuals was increased to 100 to allow enough individuals to be allocated at every niche. Additionally, after some GA runs, the niche radius σ_n was set to 0.6 since it was considered the best value in order to better identify the separate niches.

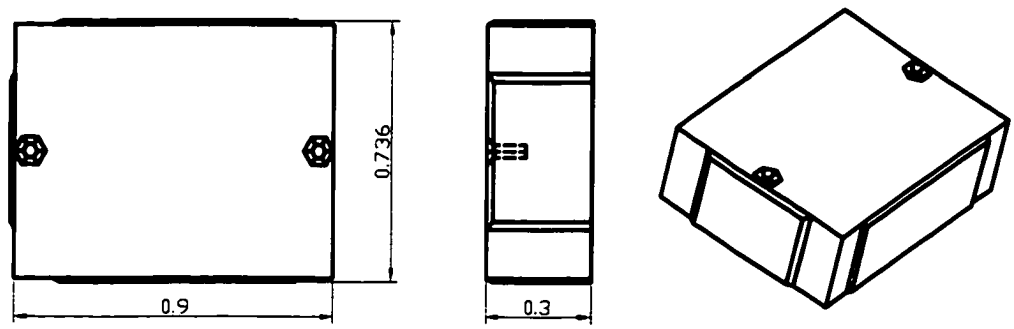


Figure 4.9. Geometry of the battery. Note the concavities formed by the two holes for the guiding pins and the lateral guides.

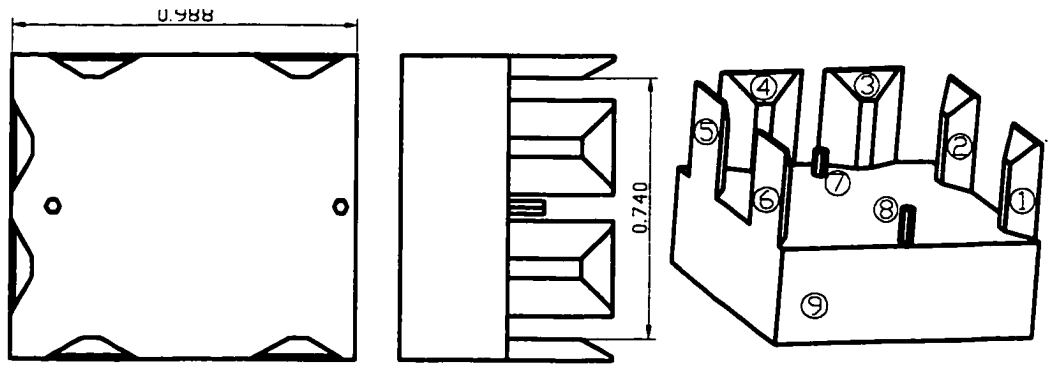


Figure 4.10. Geometry of the fixture.

Niche number	Region number	Distance	Number of individuals
1	1	0.2971	6
2	2	0.2953	5
3	3	0.2039	9
4	4	0.1336	11
5	5	0.1499	9
6	6	0.0757	53
7	7	0.4026	2
8	8	0.3731	1
9	9	0.3938	3
10	9	0.7923	1

Table 4.6. Results for the battery and fixture example using comb-GA-wl-sh. The region number refers to the regions identified in Figure 4.10.

Figure 4.12 shows the results of one run using sharing and deterministic crowding methods on the battery and fixture problem. Additionally, Tables 4.6 and 4.7 list the results for the battery and fixture example using comb-GA-wl-sh and comb-GA-wl-dc, respectively. In both cases, the GA was capable of obtaining the global minimum. While comb-GA-wl-sh was capable of obtaining up to 10 different niches (one for each protrusion of the fixture and two on the main fixture), comb-GA-wl-dc only located three of them. Notice that for the 10th niche in Table 4.6, the node corresponding to the fixture is located on the surface of the fixture opposite to where the battery is located, *i.e.*, it is on the dark side of the fixture.

In the present example, since the population size was doubled as compared to the previous examples, the number of computations required to solve the problem by both algorithms substantially increased. Algorithm comb-GA-wl-sh took slightly over 5.1 Mflops to solve the problem whereas comb-GA-wl-dc took close to 3 Mflops.

In all the examples presented above, the results show that the combinatorial GA with the sharing method outperforms its deterministic crowding counterpart at obtaining the multiple local minima in addition to the global solution. This was par-

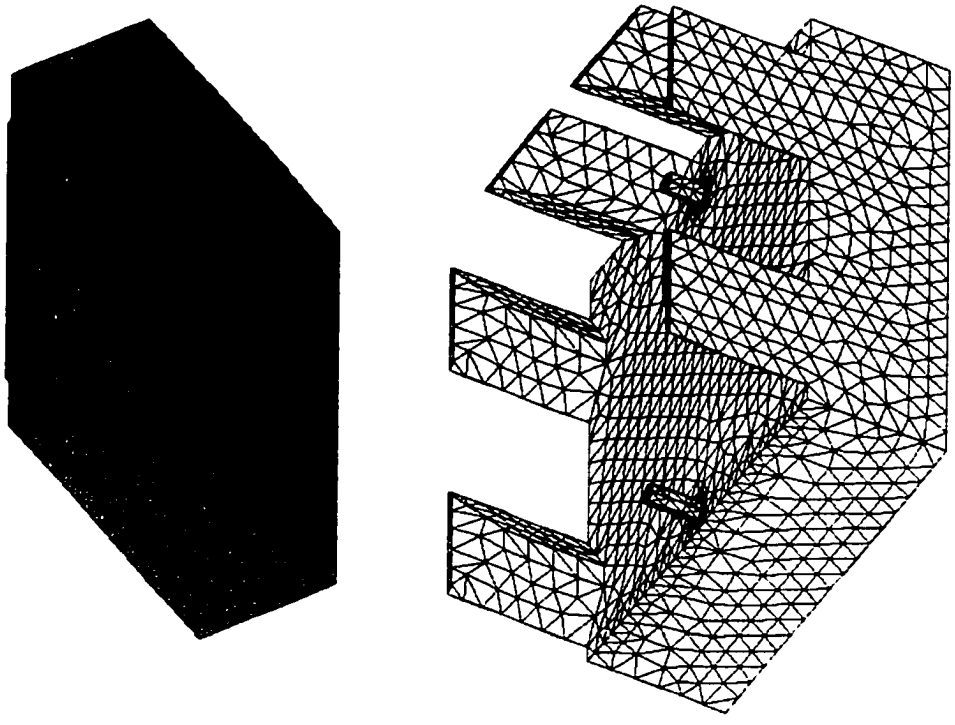


Figure 4.11. Mesh representation of the battery (1241 nodes) and fixture (1842 nodes).

Figure 4.12. Example of the GA using a) sharing and b) deterministic crowding on the battery and fixture case.

Niche number	Region number	Distance	Number of individuals
1	6	0.0757	97
2	4	0.1336	2
3	8	0.3721	1

Table 4.7. Results for the battery and fixture example using comb-GA-wl-dc. The region number refers to the regions identified in Figure 4.10.

ticularly evident in the battery and fixture example where **all** nine local minima were found by the comb-GA-wl-sh algorithm whereas comb-GA-wl-dc only obtained three of them. It is especially striking that the comb-GA-wl-dc algorithm did not detect region 2 (see Figures 4.10 and 4.12) which is the next closest region after the global minimum, and this seems a serious deficiency. In all cases, the concentration of solution points was proportional to the niche's fitness, *i.e.*, more solution points are located in areas with better fitness and fewer in less predominant local minima.

Considering the computations were performed at a rate of 1033 Mflops/s (PC at 2.2 GHz [Dongarra, 2002]) they would take only close to 5 ms for comb-GA-wl-sh on the complicated scenario whereas comb-GA-wl-dc would take close to 3 ms for the same test case.

Chapter 5

Algorithm Evaluation

Two basic formulations of the concave minimum distance problem have been discussed in this work. The proposed methods make use of global optimization techniques, particularly Simulated Annealing and Genetic Algorithms. As seen in earlier chapters, all the proposed minimum distance algorithms have a few variations. These include basic SA, SA with local optimization, simple GA, GA with sharing, GA with Lamarckian evolution, *etc.* In order to understand the differences between these algorithms and to illustrate some of their capabilities and shortcomings, extensive tests must be performed. These tests will also help tailor better algorithms for the task of finding the minimum distance between concave objects.

Two of the most important issues to keep in mind while performing tests on the algorithms are convergence and robustness. Convergence refers to whether or not the algorithm has found a definitive solution at or before the end of a fixed number of generations. In the simple GA case, for instance, this is when the average fitness value of the entire population is close to the fitness value of the best individual[1], that

[1] Note that the fitness value in GAs is always positive.

is, most of the individuals in the final population are located in the same region. On the other hand robustness refers to the ability of the algorithm to find the global solution at every run (*i.e.*, find the global minimum rather than a local one). As an example, local optimization methods, such as Newton or steepest descent methods, are very good at converging to extreme points, but are not very robust at finding the global solution (it all depends on the initial guess). On the other hand, Simulated Annealing and Genetic Algorithms (and other stochastic search methods) tend to be more robust than local optimizers at finding the region where the global optimum is located but can take a long time (*i.e.*, many iterations) to converge to the exact minimum.

This chapter first defines the performance measures that are used to compare the minimum distance algorithms. Next, all possible parameters to be varied in the numerical tests are listed and their expected effect on the solution is discussed. Then, the results of a series of numerical tests are analyzed qualitatively for large modifications on the algorithms. Additionally, a quantitative analysis of more subtle changes on the most promising algorithms is presented.

5.1 Performance measures

5.1.1 Robustness

The measure of robustness is defined here to reflect the success of the algorithm at finding the global solution. In the combinatorial approach, for instance, an enumeration technique was used (see Appendix D) in order to find the global solution. The results obtained from the GA simulations in a large number of runs are then

compared to the known global solution. The robustness (Φ^*) of the particular GA or SA implementation will be reported as a percentage success rate or as a fraction where the numerator identifies the number of successful runs over the total number of runs with the same parameters. Thus, a 199/200 success rate means that the algorithm was able to find the global solution in 199 trials out of 200 with the same optimization parameters.

Since some of the tests were performed on more than one scenario (*i.e.*, sets of geometries and poses), the distance value obtained by the algorithms could vary from one set of tests to the other even if the global solution is found. Thus, in order to be able to compare results from different numerical tests, the robustness measure, instead of the distance value itself, is reported. Additionally, the average generation number $\overline{G_i}$ (or iteration number in the SA cases) where the minimum point was found is often reported along with statistical data (*i.e.*, G_i's maximum, minimum and standard deviation values).

As discussed in Chapters 2 and 3, the search for the exact minimum, particularly in the continuous formulation[2], is not feasible in a realistic amount of time. Therefore, a second measure of robustness is proposed. The new measure, referred to as the *global region robustness* and denoted by Φ, quantifies the number of runs successful at locating the region where the global solution is located. For this purpose, a method for identifying the region where the global minimum is located is proposed here.

The method, only suitable for an off-line evaluation such as the present one, works on the principle that: a trial point is said to be successful at finding the global minimum region only if its associated distance is lower than the distance associated to

[2]It is actually expected that Φ^* will be zero, or close to zero, for the continuous tests.

the second local minima (see Figure 5.1). Additionally, the trial point's distance has to be within 10% of the actual minimum distance . Thus, as seen in the example in Figure 5.1, any trial point whose distance is lower than d_2 as well as lower than $1.1 \times d_1$ is considered to be in the region of the global solution. These criteria guarantee that the solution point found after each run is not located in a local minimum region or too far from the actual global solution. Thus, in order to evaluate global region robustness Φ, the second minimum needs to be found. For the tests presented in this work, the distance corresponding to the second minimum (*i.e.*, d_2) was obtained, for the geometries and configuration considered in the test cases, using the methods proposed in Chapter 4 and verified using a CAD package.

Additionally, since the penalty approach is used in the continuous approach, additional criteria need to be used to guarantee the solution points given by the distance algorithms are actually located in the region where the global solution is found. Thus, in order to have a trial point qualify as a successful solution point, the trial point had to be feasible. In other words, the value q_v for that solution point had to be zero (see equation (2.4)).

5.1.2 Number of niches

When multiple solution points are sought (*i.e.*, niching algorithms are used), the ratio of the number of niches found to the number of actual niches (obtained by inspection) is also reported. This measure will give an idea of how efficient each algorithm is at finding multiple minima rather than just the global minimum.

Note that, even when regular GAs are used (*i.e.*, no niching technique are included), the last population of any GA run may be put through the niche identifi-

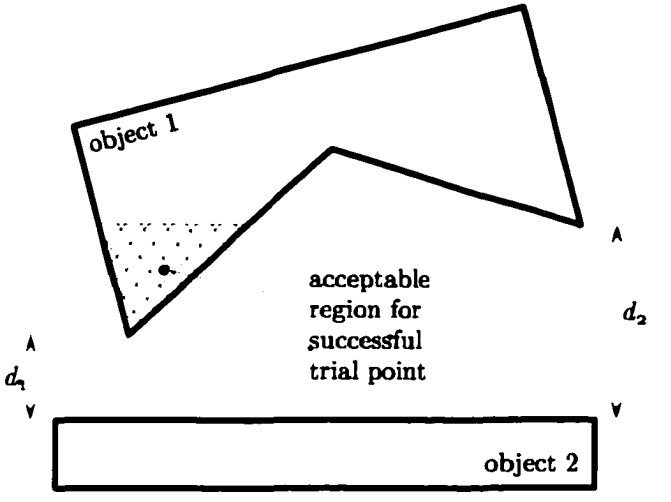


Figure 5.1. Identifying global minimum region for the continuous approach tests.

cation algorithm described in Section 4.6.4 in order to identify any existing niches. Thus, to allow a comparison with the niching methods, the number of niches will also be reported for some basic GA tests.

Also note that, SA algorithms are not well suited to finding multiple niches (*i.e.*, only one niche may be obtained) since the algorithm uses a single point at every iteration. Thus, no niche information will be reported for the SA tests.

5.1.3 Convergence measure

To evaluate the performance of a Genetic Algorithm, a few measures have been discussed in the literature. Two of these methods were proposed by De Jong [De Jong, 1975] and are described here. The first one is the one De Jong called the *off-line* performance measure, here referred to as the *best individual convergence measure*. As described in [Goldberg, 1989a], the best individual convergence measure can be

calculated as follows:

$$x_e^*(s) = \frac{1}{G_{max}} \sum_{t=1}^{G_{max}} f_e^*(t) \tag{5.1}$$

where $x_e^*(s)$ is the convergence measure of strategy s on environment e and $f_e^*(t)$ is the fitness of the best individual at instant t. That is, the best individual convergence measure $x_e^*(s)$ is the average fitness value of the best individual over a GA run.

In order to evaluate the convergence of the entire population, the measure referred to as the ongoing or *on-line* performance measure in [De Jong, 1975] is used. In this work this measure is referred to as the *population convergence measure* and is calculated as follows [Goldberg, 1989a]:

$$x_e(s) = \frac{1}{G_{max}} \sum_{t=1}^{G_{max}} \bar{f}_e(t) \tag{5.2}$$

where $x_e(s)$ is the population convergence measure and $\bar{f}_e(t)$ is the average value of the fitness function at instant t (*i.e.*, at generation t). In other words, $x_e(s)$ is the average value of all the fitness values during the entire GA run.

The value of these two performance measures is dependent on the value of the global minimum of the objective function being minimized. For this reason, in order to make the convergence measures independent of the pose of the objects, their values will be reported as a ratio of the convergence measure and the known global minimum. That is, the convergence measures that will be reported are $x'_e = x_e / f^*_{global}$ and $x^{*\prime}_e = x^*_e / f^*_{global}$ where f^*_{global} is the objective function value at the true global minimum (obtained analytically in the case of the continuous formulation and by enumeration in the combinatorial formulation - see Appendix D).

The convergence measures can also be used with the SA algorithms. Moreover, since SA algorithms have a single trial point at each iteration, the two convergence

measures are identical. Thus, while reporting the results for the SA implementations, only the best individual convergence measure x_e^* will be presented. Note that to use the performance measures described in equations (5.1) and (5.2) in the context of SA algorithms, the maximum number of generations G_{max} must be replaced by the initial temperature t_{ini}.

On the other hand, these two convergence measures cannot be used when dealing with GAs using sharing methods. This is due to the fact that GAs using sharing methods modify the fitness value of each individual according to a sharing function (see Section 4.3). That is, the shared fitness value itself varies according to the number of trial points in each niche, so that the fitness of a given point might change from one generation to the next. For this reason, the use of the convergence measures based on the fitness will not reflect the algorithm's real performance. Thus, the performance of the sharing methods will not be compared to the other method in terms of these two measures. Instead, performance measures such as the robustness and total number of niches will be used.

5.2 Independent variables

It is important at this point to distinguish between two classes of modifications that will be made to the minimum distance algorithms during the tests. First, there are changes to the algorithms themselves, *i.e.*, adding or subtracting different genetic operations or changing the algorithm type. Henceforth, these changes will be referred to as architectural changes. Second, there are changes that include varying the parameters within a particular algorithm which will be referred to as operating parameter changes. One would expect that architectural changes will greatly affect the results,

while more subtle changes will be observed when the operating parameters are varied.

In the present study, each variation of a GA or SA will require a new test, *i.e.*, whether it is only a change of a parameter or a change of the architecture.

In order to keep tests simple, one or two architectural or operating parameter modifications will be made between each set of tests. Although some parameters have a significant direct impact on the results regardless of the value of the other parameters, most effects are correlated, that is, the operating parameters are not independent of one another. As a matter of fact, finding the optimal operating parameter set of a genetic algorithm or a simulated annealing algorithm is an optimization in itself that allows many optima. Some authors that have reported using 'meta-genetic algorithms' to fine tune the parameters for a particular genetic algorithm (see for instance [Grefenstette, 1986]). Although designing such algorithms would be feasible it was deemed to be beyond the scope of the present study. It was found more relevant to determine which algorithm is best for the task rather than fine tuning each particular implementation. For this reason, only a limited semi-manual search for optimal operating parameters was performed.

5.2.1 Architectural changes

5.2.1.a Local optimization

In addition to the global optimization techniques proposed in this work to solve the minimum distance problem, local optimization techniques were proposed to improve upon the solution points at every step of the global optimization.

As explained in previous chapters, Genetic Algorithms that make use of local optimization algorithms are referred to as Memetic Algorithms that evolve to the solution

using a Lamarckian evolutionary strategy. Also, the exploitation of the Baldwin effect (as described in Chapter 1) will be tested and compared with the Memetic and Genetic algorithms. Like Memetic algorithms, those GAs using the Baldwin effect use local optimization techniques to improve each individual locally. In contrast to Memetic algorithms, the GAs using the Baldwin effect only modify the individual's fitness but do not modify the individual's genetic material [Whitley et al., 1994, Houck et al., 1996].

The SA implementations can also make use of local optimization. In fact, this will be the only architectural modification considered for the SA implementations of the minimum distance algorithms for the present tests.

It is expected that the minimum distance algorithms using the global optimization combined with the local optimization will converge to a solution, local or global, faster than if the global algorithms were used on their own. Additionally, as a result of using the local optimization techniques, it is expected that the computational expense required by the algorithm to perform a run will increase, particularly for the combinatorial formulation.

5.2.1.b Crossover method

The two genetic operations that have the greatest effect on a GA's performance are the selection process and the crossover operation. Of these two, the crossover operation is the one that determines the genetic material of the offspring in terms of their parents. Furthermore, it is the crossover operation what determines whether the offspring will replace the parents or the parents themselves will continue into the next generation.

This section describes the different crossover strategies studied here and their expected effects within the overall GA operation.

Genetic replacement If genetic replacement is used during the crossover operation, the offspring replace the parents in the population, no matter which ones are fitter. As a result, the population's fitness could worsen particularly when the probability of crossover (p_x) is high. This, in turn, would slow down the convergence process if the initial population is large and diverse.

Deterministic crowding This technique eliminates individuals that are too close together. That is, when two offspring are created, they are compared to their closest parent and replace it only if their fitness is better, otherwise the parent stays in the population.

Although deterministic crowding is used to promote the creation of niches, it can also help reduce premature convergence by maintaining population diversity. This would allow the algorithm to explore areas of the search space other than where individuals (or clusters of them) are already located. This in turn, is expected to increase the robustness of the algorithm at finding the global minimum.

Mating restriction When mating restriction is used, only parents that are within a predefined mating radius (σ_m) of each other are considered for mating. If the mating radius is selected appropriately, mating restriction will allow the GA to keep diversity in the population, thus increasing the algorithm's robustness.

Mating restriction can be used with most crossover operations since it acts as a screening process before each set of parents is mated. That is, mating restriction can be used in combination with the crossover methods presented earlier, *e.g.*, deterministic crowding or genetic replacement.

5.2.1.c Sharing

Sharing allows the GA to create clusters or niches of individuals by decreasing the fitness of an individual in proportion to the number of individuals sharing a particular region of the search space. That is, if five individuals are within a predefined sharing radius (σ_s) the fitness of these five individuals will each be divided by five (or some other number proportional to the distance between individuals and the number of individuals in the niche).

Like deterministic crowding, the sharing method is expected to keep higher population diversity in later generations as compared to the basic GA, thus increasing the robustness (Φ) of the GA.

It is important to note that, when sharing methods are used, the computational complexity of the overall GA increases, particularly if large populations are used, due to the need to compare each individual of the population with all other individuals.

5.2.2 Operating parameter changes

In addition to the architectural changes made to different algorithms, changes to the algorithms' operating parameters will also be made. The operating parameters which follow have been described thoroughly when the GA and SA algorithms were introduced in Chapters 2 through 4. Here, only a brief description of each of these parameters is given and their expected effect on the results is outlined.

5.2.2.a SA parameters

Cooling schedule There are two basic independent parameters to vary in an SA algorithm: the initial temperature and the Boltzmann constant, which constitute what

is called the cooling schedule. These two parameters can be modified independently but in fact they have a great effect on each other (see equation 2.6).

In general terms, if a large initial temperature (t_{ini}) is given to the SA algorithm, the algorithm will have a better chance of obtaining the global minimum since the number of trial points in a single run is proportional to the initial temperature. Therefore the number of computations per SA run is directly proportional to t_{ini}. On the other hand, the Boltzmann constant (p_b) will have a great effect on the capability of the algorithm in escaping from a local minimum trap. As seen in Figure 2.10 (page 46), a relatively large Boltzmann constant will allow the algorithms to accept moves that do not improve the fitness function value well into the simulation. On the other hand, a relatively small Boltzmann constant yields a small probability for accepting moves that do not improve the fitness value even in early stages of the SA run.

Move size As discussed in Sections 2.2.3.c and 3.2.2.c, fixed or variable size moves can me made in order to try new points in the SA runs. It is expected that large moves at the beginning of the SA run will allow the SA algorithm to explore new regions of the search space thus increasing the algorithm's robustness. Moreover, reducing the move size in the later parts of the SA run will allow the algorithm to locally improve the solution (whether it is a local or global minima).

On the other hand, while using local optimization within the global method, the contribution of the small moves in later iteration in the SA run are expected to be less important or even useless since most trial points in the nearby neighborhood will yield the same local minimum. Thus, relatively large move size are expected to be better suited for exploring new regions of the search space in SA with local

optimization particularly in the combinatorial formulation.

5.2.2.b GA parameters

Mutation probability The mutation probability (p_m) is an indication of the rate at which the mutation operation is carried out. It was discussed in earlier chapters that the mutation operation is necessary to maintain population diversity and allow the GA to explore new regions of the search space.

It is expected that a very large mutation probability will greatly affect the convergence of the algorithms since points that have already been found in potentially good regions could be mutated thus losing these good regions. On the other hand a very low mutation probability will not allow the algorithms to explore enough new areas of the search space. In the extreme case when the mutation probability is set to zero, the only areas of the search space that will be explored are contained in a sub-manifold of the entire search space spanned by the initial population.

Crossover rate As described in earlier sections, the probability of crossover (p_x) determines the rate at which the crossover operation is performed. A large probability of crossover would be expected to accelerate convergence. A small or zero crossover probability makes the GA ineffective as the GA would act as a random search. Furthermore, a high crossover rate would result in a higher degree of exploitation of the populated areas but little or no exploitation of the less densely populated areas since convergence to fit regions would be accelerated.

Population size The size of the population (N_{pop}) plays a major role in the robustness of the GAs due to the fact that a large population generated at random at

the initial iteration allows for a larger diversity at the start of the run. This, in turn, enables the GA to search different areas of the space simultaneously.

On the other hand, a very large population may waste computational resources and slow down convergence since it takes more generations to 'move' all offsprings to a good (or at least better) region of the space [Michalewicz, 1994, p. 72]. Conversely, a very small population may create problems of premature converge thus not allowing the algorithm to explore different regions of the space.

Many authors have published thorough studies where the population size is the main focus, see for instance [De Jong, 1975, Goldberg, 1989b, Mahfoud, 1995]. In some practical applications, including function optimization, populations in the range from 16 of 100 have been reported [Goldberg, 1989a]. In the present tests, *small population* will refer to populations with less than 30 individuals, whereas *large populations* will contain 80 or more individuals.

Number of generations The maximum number of generations (G_{max}) in a GA determines how long an algorithms is allowed to run. Due to the inherent nature of the GAs, it is expected that any GA allowed to run for a large number of generations will result in a better solution than the same algorithm with a smaller number of generations. On the other hand, the GA may converge to a minimum long before reaching the maximum number of iterations. As was explained in earlier chapters, the GAs tend to obtain the region where a minimum point is located, but finding the exact minimum might take a long time since the exploration is mainly done at random (that is, of course, if no local optimization is used).

Additionally, when local optimization is used, particularly within the combinatorial GA, it is expected that the algorithms will find the global minimum soon after

the start of the GA run (*i.e.*, within the first few generations) particularly if the population is large (*i.e.*, 80 or more individuals). Thus, these algorithms will tend to need a small total number of generations (*i.e.*, less than 50) for the task of finding the global minimum.

5.2.2.c Niche formation parameters

Sharing radius The sharing radius σ_s determines the maximum distance between two trial points to be considered as part of the same niche. A large sharing radius will tend to produce one single niche of trial points whereas a small σ_s will tend to produce a large number of niches. Thus, the proper selection of the sharing radius σ_s will greatly affect the number of niches produced during a particular GA run. In the present problem, a sharing radius proportional to the overall size of the objects and the number of protrusions on the objects can be used.

On the other hand, while using Memetic Algorithms (those GAs using local optimization at every iteration), the trial points after being locally optimized will naturally tend to be relatively close to each other. This is due to the fact that several initial points might have a single local optimum. Thus, all trial points in a niche will tend to be very close to each other or will share exactly the same nodes or coordinates. In those cases, the under estimation of the sharing radius is more desirable and less critical for the operation of the GA with sharing.

Sharing exponent Figure 5.2 shows how the sharing value $S(\Delta)$ varies as the phenotypic distance is varied relative to the niche radius σ_s for five different values of the sharing exponent α_s. A careful analysis of the effects of α_s on the overall GA operation shows that decreasing the sharing exponent lower than unity ($\alpha_s < 1$)

tends to produce small sharing function values $S(\Delta)$ (equation (4.3)). In turn, small $S(\Delta)$ values produce a smaller niche count value m_i' (equation (4.2)) even for very broad niches (*i.e.*, niches in which trial points are not concentrated). This results in a shared fitness f_i' (equation (4.1)) value better than if a sharing exponent $\alpha_s > 1$ was used. Thus, when $\alpha_s < 1$, the niches with better fitness tend to predominate attracting a larger number of trial points to the same niche until the niche count increases and a balance is found.

Similarly, $\alpha_s > 1$ will tend to produce large sharing function values $S(\Delta)$ thus decreasing the shared fitness value f_i' of the individuals in that niche. This, in turn, puts a lot of pressure on the niches with a large number of trial points eventually losing some of the trial points in the selection process. As a result, it is expected that GAs using sharing with a sharing exponent $\alpha_s > 1$ will tend to have a larger number of niches than those using $\alpha_s < 1$. Additionally, niches with relatively low fitness will tend to have more trial points in them than in the case where $\alpha_s > 1$.

Mating restriction As described in Chapter 4, the mating radius σ_m determines the maximum distance between two parents in order to be considered for reproduction. That is, if the parents are within a distance σ_m of each other, they are considered for mating, otherwise another set of parents is sought.

As was done for the sharing radius σ_s, the mating radius could be set as a function of the overall size of the objects.

If a large value of σ_m is used, trial points of separate niches are allowed to mate which might slow down convergence of the GA by creating offsprings that do not belong to either niche.

There is an interplay between the mating radius and the sharing radius. It is

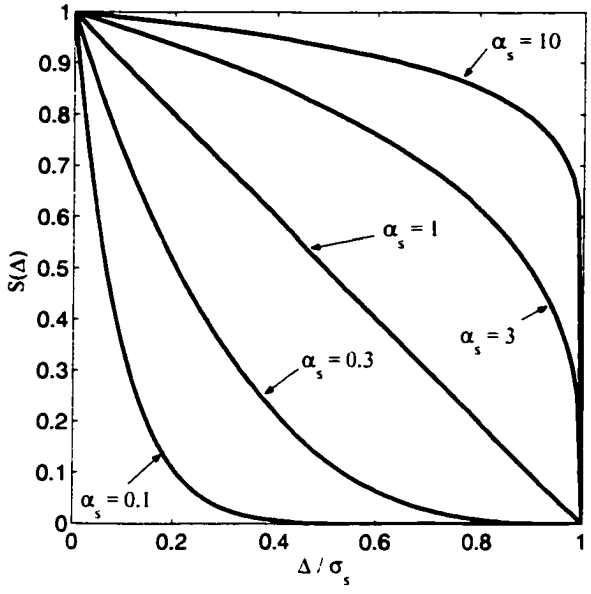


Figure 5.2. Effect of the sharing exponent α_s on the sharing function value $S(\Delta)$.

expected that the mating radius σ_m will have little effect on the operation of the Memetic Algorithms, especially if a value of σ_m smaller that σ_s is used since most trial points will be concentrated at local minima.

Niche radius It was discussed in the previous chapter that the niche radius σ_n determines the proximity threshold used in the niche identification algorithm in order to determine the size of a niche (see Section 4.6.4). Since the identification of the niches is performed after the GA run is terminated, the selection of the niche radius σ_n does not affect the performance of the GA algorithms. However, σ_n will greatly affect the outcome of the distance algorithm since it is the last population of the GA run that is put through the niche identification algorithm. Therefore, σ_n needs to be carefully determined. Additionally, to allow a proper comparison between the GAs with niching methods, σ_n was kept constant for each set of geometries throughout the tests of those algorithms.

In the present problem, σ_n can be set as a function of the overall size of the objects, the total number of niches that are expected to be found, and, most importantly, as a function of the shortest distance expected between two adjacent niches. This last criteria is of upmost importance since setting a niche radius too large would tend to merge adjacent niches together into a single niche thus eliminating a possible local minimum.

On the other hand, using a small niche radius would tend to produce a large number of niches even when some of them might not need to be considered as such. Nonetheless, as was pointed out for the sharing and mating radii, while using Memetic Algorithms, most trial points would tend to be relatively close to one another making less critical the underestimation of the niche radius. Additionally, sharing and mating restriction would tend to make the niches more compact thus making the niche identification more effective and less sensitive to the selection of the niche radius.

5.3 Results

GA and SA algorithms are stochastic in nature due to the presence of random operations. In order to better determine the performance measures for a particular algorithm, tests for each algorithm have to be performed a number of times under the same conditions. In the present study, most tests are presented as the results of up to 1000 runs with the same parameters. Additionally, in order to evaluate the worst possible scenario, in each run no prior knowledge of the solution is assumed. That is, all runs start with a randomly generated initial trial points.

In order to keep sets of tests consistent while varying the parameters, the seed of the random number generator was reset each time a new set of tests was per-

formed. This ensured that the difference between results was, in fact, a function of the changes in the algorithms' operating parameters or architecture rather than caused by stochastic drift.

Since each test is performed over many runs, the average of the particular performance measure being considered is reported, together with some statistical information of its values over the entire set of tests. In most cases, the reported results will include the maximum value, the minimum value and the standard deviation of the particular performance measure being considered.

The geometries and poses used in the following tests are the ones presented in the numerical examples in Chapters 2 through 4. In most cases, due to the complexity of the problem, it is the ORU battery and fixture test case (see Section 4.7.3) that is used for the tests. This complex problem allows the algorithms to demonstrate their performance in a more realistic scenario where the objective function contains many local minima. Additionally, the ORU battery and fixture problem tests the capabilities of the GAs with niche formation techniques at finding several multiple minimum regions.

The numerical tests are organized as follows. First, the continuous and combinatorial formulations are tested and compared in terms of their robustness. The use of local optimization methods within the global search is also evaluated with both formulations. Then, for one of the formulations, namely the combinatorial formulation, the SA and GA implementations are compared also in terms of their robustness. Next, in the combinatorial formulation, the niche formation methods proposed in Chapter 4 are compared in terms of their robustness as well as their ability to find multiple minima in a couple of scenarios. Finally, some of the algorithms are 'fine-tuned' in order to demonstrate the process of how to determine the best set of operating

parameters for a particular GA or SA implementation.

Notice that, in the context of the distance determination algorithms, it is the robustness that is most important. As mentioned earlier, it is the robustness that determines if the global solution or the region where the global solution is located has been found or not. Thus, the initial tests on the continuous and combinatorial formulations, as well as the ones on the SA and GA implementations, will mainly focus on the robustness of the algorithms. On the other hand, while fine-tuning the minimum distance algorithms, the convergence measures described in Section 5.1.3 are used in order to identify more subtle changes in the algorithm's performance.

5.3.1 Architecture

5.3.1.a Combinatorial *vs.* continuous approaches

In order to evaluate and compare the continuous and combinatorial formulations, tests of their GA implementations were performed. The default operating parameters and genetic operators used in the present tests are listed in Tables 5.1 and 5.2 for algorithms cont-GA and comb-GA, respectively. That is, unless otherwise specified, the GA operating parameters were set as in Tables 5.1 and 5.2.

Several tests were performed using different geometries and poses. Here, the results obtained on tests with three different scenarios are presented. The first two are simple scenarios which are illustrated in Figures 5.3 and 5.4. The third one, on the other hand, is the ORU and battery test case presented in Chapter 4 (see Figure 5.5) which is more complicated since it has up to 9 local minima.

Table 5.3 shows the most representative results for runs on simple problems with different population sizes and the mutation probability set to $p_m = 1/N_{pop}$. In

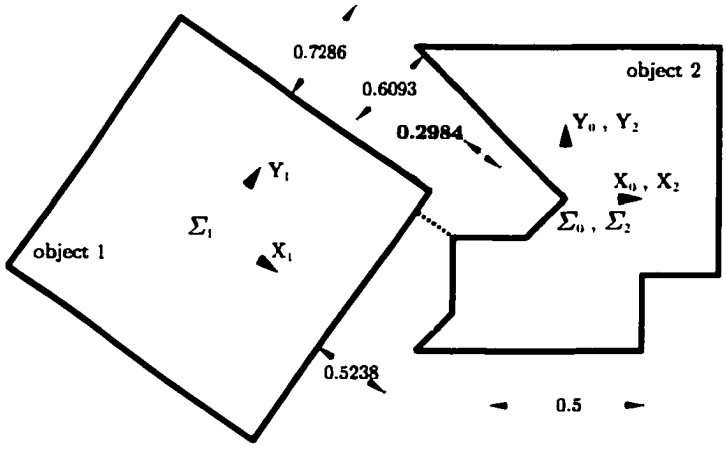


Figure 5.3. Geometry and configuration of the boxes with notches example (Geometry set 1 in position 1). Note that the minimum distance is shown in bold face ($d^* = 0.2984$).

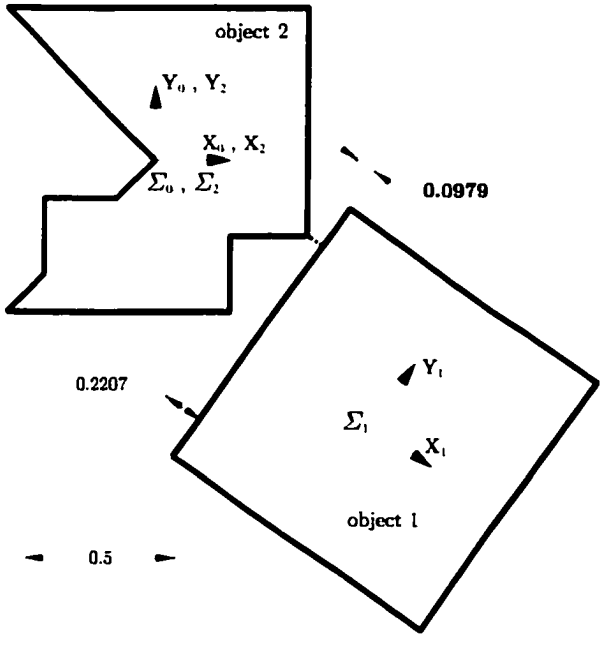


Figure 5.4. Geometry and configuration of the boxes with notches example (Geometry set 1 in position 2). Note that the minimum distance is shown in bold face ($d^* = 0.0979$).

Parameter / Operator	Symbol	Value / Type
population size	N_{pop}	80
probability of mutation	p_m	0.0125
probability of crossover	p_x	0.6
max. number of generations	G_{max}	100
penalty weight	k_p	10
repair scaling (safe repair)	k_r	1.5
Selection		stochastic universal sampling
Mutation		random direction with variable disp.
Mating		fitness proportional

Table 5.1. Default parameter set for cont-GA-wl.

Parameter / Operator	Symbol	Value / Type
population size	N_{pop}	80
probability of mutation	p_m	0.0125
probability of crossover	p_x	0.6
max. number of generations	G_{max}	100
Selection		stochastic universal sampling
Mutation		random move
Mating		mesh mating

Table 5.2. Default parameter set for comb-GA-wl.

these cases, the continuous GA was able to locate the global minimum region with a global region robustness Φ up to 65% while using local optimization but had very low robustness when no local optimization was used.

The combinatorial implementations had much higher robustness measure than their continuous counterparts, particularly when using local optimization. This was also observed for tests with small population sizes. In the two simple examples shown in Table 5.3, the robustness Φ^* was up to 1000/1000 when using the comb-GA-wl algorithm with a population of 80 individuals (Φ^* was 914/1000 with $N_{pop} = 10$).

On the other hand, Table 5.4 presents the results of the tests performed on the minimum distance algorithms on the more complicated scenario of the battery and fixture. It can bee seen in Table 5.4 that the cont-GA-wl algorithm was not capable

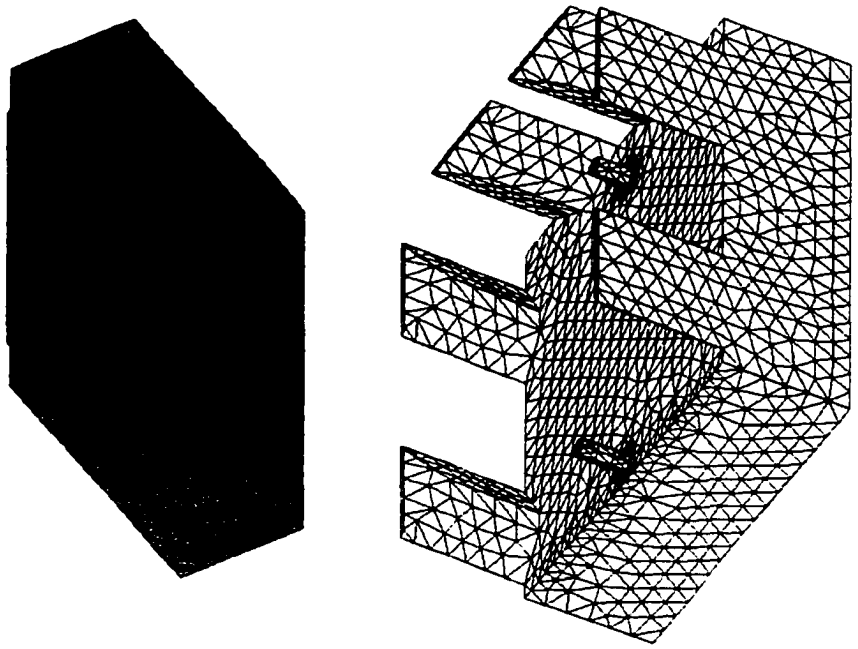


Figure 5.5. Mesh representation of the battery (1241 nodes) and fixture (1842 nodes).

of locating the region where the global minimum was located ($\Phi = 0/50$ with $N_{pop} = 80$). By contrast, the comb-GA-wl algorithm had a much better robustness with $\Phi^* = \Phi = 1000/1000$ with $N_{pop} \geq 80$.

Additionally, it is important to notice that the combinatorial GA using local optimization has an excellent robustness even for complicated scenarios such as the battery and fixture problem. As a matter of fact, if the population size N_{pop} was selected greater that 80, both measures of robustness were found to be 1000/1000. Moreover, even when using relatively small population sizes down to $N_{pop} = 20$, the robustness measures were still found to be quite good (see Table 5.4).

5.3.1.b Genetic Algorithms *vs.* Simulated Annealing implementations

In order to allow a fair comparison between the SA and GA implementations of the minimum distance problem, in most cases, both algorithms were allowed to test a

Algorithm	N_{pop}	boxes with notches pose 1		boxes with notches pose 2	
		Φ^*	Φ	Φ^*	Φ
cont-GA-nl	80	0/50	0/50	0/50	0/50
cont-GA-wl	80	0/50	7/50	0/50	6/50
cont-GA-wl	250	0/25	6/25	0/25	6/25
cont-GA-wl	1500	0/25	8/25	0/25	16/25
comb-GA-nl	80	1/1000	702/1000	83/1000	398/1000
comb-GA-wl	80	999/1000	1000/1000	1000/1000	1000/1000
comb-GA-wl	10	808/1000	1000/1000	914/1000	1000/1000

Table 5.3. Results of the continuous and combinatorial GA implementations on two simple scenarios ($p_m = 1/N_{pop}$).

Algorithm	N_{pop}	battery and fixture	
		Φ^*	Φ
cont-GA-nl	80	0/50	0/50
cont-GA-wl	80	0/50	0/50
cont-GA-wl	250	0/25	0/25
cont-GA-nl	1500	0/25	0/25
comb-GA-nl	80	2/200	34/200
comb-GA-wl	80	1000/1000	1000/1000
comb-GA-wl	20	950/1000	995/1000

Table 5.4. Results of the continuous and combinatorial GA implementations on a complex scenario ($p_m = 1/N_{pop}$).

similar number of trial points. Thus, the number of iterations allowed for the SA runs (*i.e.*, the initial temperature t_{ini}) was set as the product of the maximum number of generations (G_{max}) and the population size (N_{pop}) that is, $t_{ini} = N_{pop} \times G_{max}$. For example, if a GA was run for $G_{max} = 100$ generations with a population $N_{pop} = 80$, the initial temperature in the equivalent SA test was then set to $t_{ini} = G_{max} \times N_{pop} = 100 \times 80 = 8000$.

Although the above mentioned method was used to determine t_{ini}, it was noticed after a few tests that the combinatorial SA implementation using local optimization required up to 15 to 25 times more computations than the combinatorial GA

to perform a simulation. This is due to the fact that each trial point in the SA implementation is generated at random well into the run. Thus, the total number of steps the local optimizer needs to take to optimize each trial point locally was, on average, much larger than in the GA implementation resulting in a substantially longer computational time for the SA implementation. As a result, it was decided to use a smaller number of iterations to run the SA tests with local optimization.

In Tables 5.5 and 5.6, the results from some of the numerical tests performed on the combinatorial SA implementation are presented. In all cases the cooling schedule parameters (*i.e.*, the initial temperature and the Boltzmann constant) used for each test is presented together with the results. Note that, in order to generate new points, the method described in Section 3.2.2.c, referred to as the random move with variable displacement method, was used.

Tables 5.5 presents the test results for two poses of the simple problem first described in Section 2.4 (see Figures 2.16 and 2.17) whereas Table 5.6 contains the results of the ORU battery and fixture example described in Chapter 4. In all the tests presented here, it can be seen that, unless local optimization is used, the robustness Φ is always lower than 4% while $\Phi^* < 0.5\%$. On the other hand, when local optimization is used, both robustness measures increased substantially to values comparable to those of the combinatorial GA implementations presented in the previous section (Tables 5.3 and 5.4).

5.3.1.c Local optimization

As expected, the algorithms using local optimization outperform the ones that do not. This is due to the fact that, when locally optimizing individuals, the global optimization uses only 'good' points to perform genetic operations since all the points

Algorithm	t_{ini}	k_B	boxes with notches pose 1		boxes with notches pose 2	
			Φ^*	Φ	Φ^*	Φ
comb-SA-nl	8000	0.0005	1/200	142/200	1/200	18/200
comb-SA-nl	2500	0.001	1/200	10/200	0/200	3/200
comb-SA-wl	8000	0.0005	200/200	200/200	200/200	200/200
comb-SA-wl	150	0.01	181/200	200/200	200/200	200/200

Table 5.5. Results of the combinatorial SA implementation on two simple scenarios.

Algorithm	t_{ini}	k_B	battery and fixture	
			Φ^*	Φ
comb-SA-nl	8000	0.0005	1/200	7/200
comb-SA-nl	2500	0.0001	0/200	1/200
comb-SA-wl	8000	0.0005	200/200	200/200
comb-SA-wl	150	0.01	200/200	200/200

Table 5.6. Results of the combinatorial SA implementation on a complex scenario.

in the genetic pool are 'good', at least locally, the offsprings tend to be relatively good as well.

Similarly, for the SA implementations, all the random moves are made from a local minimum. This has the advantage that, in the minimum distance problem, the local and global minima can in many cases be located in similar region of the objects (see for instance the cube with notches example in Chapters 2 and 3).

On the other hand, when local optimization is not used, there is no guarantee that an exact minimum, even a local one, can be found unless the algorithm is left running for a large number of generations/iterations while allowing some new random trial points to be generated. For instance, by using local optimization in the combinatorial formulation, one guarantees that each trial point that is selected for any genetic operation is the best point found in that surrounding area.

Figure 5.6 shows the best point of the final population for 200 and 500 runs on the

Figure 5.6. Best point of the final population on several runs for a) comb-GA-nl and b) comb-GA-wl.

combinatorial GA with and without local optimization, respectively. The algorithm used to produce Figure 5.6a was allowed to run for 80 generations with a population of 1000 whereas for Figure 5.6b local optimization was used on a population of 100 individuals allowed to run for 50 generations. It is important to notice that the vast majority of the solution points are located on a local minimum even if no local minimization algorithm is used.

During some extensive testing, it was observed that most GA combinatorial algorithms using local optimization would find the best point, on average, in the first 1 or 2 iterations when a populations of 20 or more individuals for simple geometries, and 80 or more individuals for complicated geometries were used. In some cases with small population sizes, it took up to 28 generations to find the global minimum. That is, while mutations are allowed, the smaller the population the longer it takes

to find the global minimum. As will be explained in a later section, this is due to population diversity which is directly proportional to the size of the population.

It was noticed that, for the combinatorial formulation, the local optimization can account for up to 80% of the computational time taken by the regular GA implementations and about 50% in the case of the GAs with sharing.

5.3.1.d Lamarckian evolution *vs.* the Baldwin effect

In previous sections, only GAs with Lamarckian evolution have been investigated and tested. Here, a series of tests on the combinatorial GA to verify the advantages and disadvantages of the two different local optimization strategies (*i.e.*, Lamarckian evolution *vs.* the Baldwin effect) are presented.

After a number of tests summarized in Table 5.7, it is concluded that, the GAs using Lamarckian evolution outperform the ones using the Baldwin effect in both aspects studied here, *i.e.*, robustness and convergence. Table 5.7 presents the results performed on a simple scenario. As seen in Table 5.7, while the LGA (Lamarckian GA) was able to find the global solution in all of the runs, the BGA (Baldwin GA) would only succeed around 19% of the time while using $p_m = 0.01$.

On the other hand, it was noted that a higher mutation rate, such as $p_m = 0.1$, increased the ability of the BGA to find the global solution to $\Phi^* = 169/200$. Despite this increase in the robustness of the BGA, it is still much lower that the robustness values observed for the LGA. Additionally, as expected, the robustness of BGA is increased as the maximum number of generations is increased. This is an expected result that comes with a computational cost directly proportional to G_{max} (as discussed in Section 5.2.2.b).

Due to its low robustness, as compared to the Lamarckian Genetic Algorithms,

Algorithm	p_m	Φ^*	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$x_e^{*\prime}$	$x_e^{\prime}$
LGA	0.01	200/200	1.024 [1 − 13], 0.5	1.00002	1.8727
BGA	0.01	38/200	1.045 [1 − 6], 0.4	1.00016	2.7092
LGA	0.1	200/200	1.000 [1 − 1], 0.0	1.00000	2.8541
BGA	0.1	169/200	1.005 [1 − 2], 0.1	1.00001	3.6617

Table 5.7. Local optimization tests: comparison of LGA and BGA algorithms with $N_{pop} = 100$, $G_{max} = 30$ and $p_x = 1$.

the GAs using the Baldwin effect were not pursued further in these numerical tests. Essentially, the computational expense of obtaining the local optimum is not used efficiently in the BGA.

5.3.1.e Niche formation

Extensive tests were performed to compare the combinatorial GAs using niche formation techniques. Here, only the most relevant tests are reported to illustrate the differences between the two niche formation techniques described in Chapter 4 (*i.e.*, sharing and deterministic crowding). In both cases, the main purpose of the algorithm is to generate multiple clusters of solution points rather than a single conglomerate of points. Thus, the number of niches created with the different algorithms is reported here as compared to the expected number of niches for the particular test case.

To evaluate the algorithms under the most demanding conditions available, the battery and fixture geometries were used for the tests. Thus, as discussed in Chapter 4, a total of 9 niches is expected in most configurations of the battery and fixture case (*i.e.*, one for each protrusion of the fixture plus one for its main body).

To identify the niches created during the GA runs, the final population of the GAs was put through the niche identification algorithm described in Section 4.6.4.

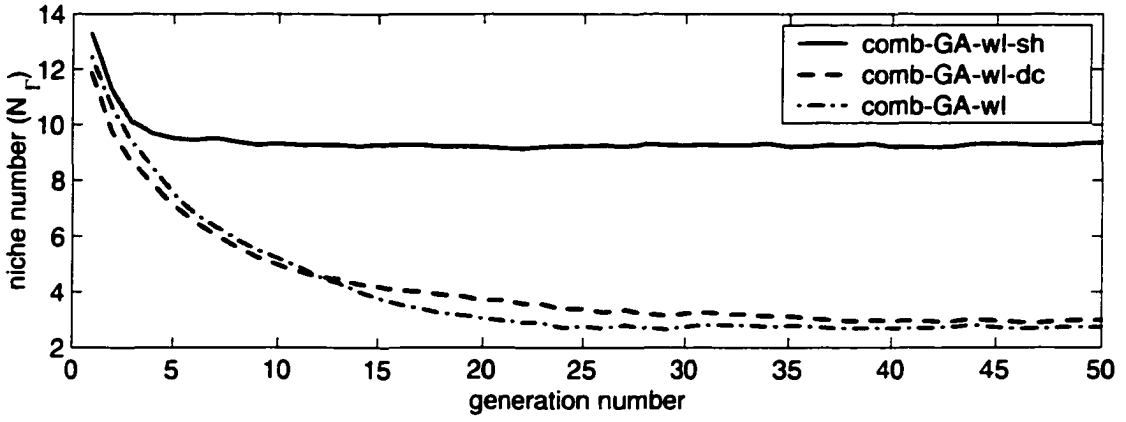


Figure 5.7. Time history, averaged over 200 runs, of the number of niches (N_Γ) for comb-GA-wl, comb-GA-wl-sh and comb-GA-wl-dc all with $N_{pop} = 100$.

In order to identify the niches in the ORU battery and fixture problem, a niche radius of $\sigma_n = 0.6$ was used in all the tests in this section. The value for σ_n was determined by inspection and verified through numerical tests and was deemed to produce reasonable results for both niche formation methods in the battery and fixture problem. All other parameters values used in the tests in this section are listed in Table 5.8.

Figure 5.7 illustrates an example of the average over 200 runs of the number of niches as a function of the generation number for the basic combinatorial GA, combinatorial GA with sharing and combinatorial GA with crowding. It is important to notice that, although the GAs with local optimization finds the global solution in a few iterations, it takes significantly longer to allocate all the population to the appropriate niches. In order to obtain the results shown in Figure 5.7, the niche identification algorithm was run on a new population generated at random. It can be observed in Figure 5.7 that in such randomly generated populations the number of niches is much greater than the one found after the algorithm has converged.

The influence of the population size on the number of niches obtained with niche formation methods, as well as with the basic GA, was investigated. Tables 5.9 and 5.10 show the effect of varying the population size on the algorithms described in Chapter 4, *i.e.*, comb-GA-wl-sh and comb-GA-wl-dc, respectively, and Table 5.11 presents the results obtained with the comb-GA-wl. As expected, in the comb-GA-wl-sh algorithms, the number of niches approached the actual number of niches as the population size is increased allowing the distance algorithms to keep track of all niches. On the other hand, from the results presented in Tables 5.10 and 5.11, the deterministic crowding method did not show any improvement in the formation of niches relative to the basic GA.

It is important to notice that for the comb-GA-wl-sh algorithm, the robustness Φ^* was extremely high. As a matter of fact, in many cases, Φ^* was seen to be higher relative to the robustness of the basic GA. This is due to the fact that the niching algorithms promote population diversity thus allowing some points to explore new regions of the solution space.

One of the parameters that was observed to have the strongest effect within the niching methods was the use of mating restriction. It was found that the use of mating restriction was essential for the sharing algorithm to be capable of creating the necessary population diversity. To allow points outside the niches to mate with points inside them, the mating radius σ_m used was slightly larger than the niche radius (*i.e.*, $\sigma_m > \sigma_n$).

The GA with sharing techniques was identified as the best of the three methods in order to obtain the largest possible number of niches. As a matter of fact, in a large portion of the numerical tests, the sharing methods identified all 9 niches in the problem whereas, under similar conditions, the GA with deterministic crowding

Parameter / operator	Symbol	Value / type
probability of mutation	p_m	$1/N_{pop}$
probability of crossover	p_x	0.6
max. number of generations	G_{max}	50
sharing radius	σ_s	0.6
sharing exponent	α_s	1
niche radius	σ_n	0.6

Table 5.8. Parameter values for tests on combinatorial GA with sharing.

N_{pop}	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$\overline{N_\Gamma}$ $[N_{\Gamma min} - N_{\Gamma max}]$, σ_{N_Γ}
20	500/500	500/500	2.51 [1 – 29], 4.2	5.45 [2 – 9], 1.17
50	500/500	500/500	1.10 [1 – 8], 0.6	7.81 [5 – 11], 0.96
100	500/500	500/500	1.00 [1 – 1], 0.0	9.19 [7 – 12], 0.66
150	500/500	500/500	1.00 [1 – 1], 0.0	9.28 [8 – 12], 0.59
200	500/500	500/500	1.00 [1 – 1], 0.0	9.37 [8 – 12], 0.60

Table 5.9. Niche formation tests on the comb-GA with sharing while varying the population size. The operating parameters are listed in Table 5.8.

was only able to obtain 3 of the niches (*i.e.*, the global minimum and two other local optima). Additionally, the number of niches found by the comb-GA-dc-wl algorithm was found to be similar to the number of niches found by the basic GA.

In terms of computational efficiency, as expected, the basic GA and the GA with deterministic crowding required a similar computational expense to perform similar runs[3]. Additionally, the computational expense was seen to be directly proportional to the population size. On the other hand, due to the calculation of the sharing value, the GAs using the sharing method took longer to perform similar runs. In fact, the additional computational expense was also proportional to the population size bringing the sharing algorithm's overall computational expense to be proportional to the square of the population size.

[3]The term 'similar runs' refers here to runs which operating parameters are equal but in which the type of operators or algorithms are different.

N_{pop}	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$\overline{N_\Gamma}$ $[N_{\Gamma min} - N_{\Gamma max}]$, σ_{N_Γ}
20	452/500	500/500	3.92 [1 – 48], 7.8	2.99 [1 – 8], 1.21
50	491/500	500/500	1.27 [1 – 47], 2.6	2.95 [1 – 7], 1.22
100	499/500	500/500	1.00 [1 – 1], 0.0	2.89 [1 – 7], 1.18
150	500/500	500/500	1.00 [1 – 1], 0.0	2.92 [1 – 7], 1.19
200	500/500	500/500	1.00 [1 – 1], 0.0	2.98 [1 – 6], 1.15

Table 5.10. Niche formation tests on the comb-GA with deterministic crowding while varying the population size. The operating parameters are similar to the ones listed in Table 5.8.

N_{pop}	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$\overline{N_\Gamma}$ $[N_{\Gamma min} - N_{\Gamma max}]$, σ_{N_Γ}
20	494/500	500/500	3.92 [1 – 30], 5.2	2.93 [1 – 7], 1.23
50	500/500	500/500	1.16 [1 – 21], 1.3	3.12 [1 – 6], 1.12
100	500/500	500/500	1.00 [1 – 1], 0.0	3.14 [1 – 7], 1.26
150	500/500	500/500	1.00 [1 – 1], 0.0	3.29 [1 – 8], 1.21
200	500/500	500/500	1.00 [1 – 1], 0.0	3.23 [1 – 7], 1.23

Table 5.11. Niche formation tests on the comb-GA with no niching method while varying the population size. The operating parameters are similar to the ones listed in Table 5.8.

5.3.2 GA parameters

5.3.2.a Penalty weight

In an initial set of trials, it was noted that while using a relatively small penalty ($k_p \leq 10$) a large proportion of the solutions returned by the minimum distance algorithm were infeasible trial points. That it, at least one of the points was located outside the object.

To investigate this effect a few tests were conducted with all operating parameters identical between runs except for the penalty weight. Table 5.12 presents the results of the continuous GA tests on a simple scenario. In simple scenarios, a large k_p value such as 10000 produced 25/25 success rate in finding the global region but only 5 of

those 25 trials had distance within 10% of the global solution. This essentially made the penalty strategy act as a rejection strategy eliminating all infeasible points.

k_p	Φ^*	Φ	trial points with distance lower than second niche's distance
0	0/25	1/25	7/25
10	0/25	10/25	23/25
100	0/25	9/25	23/25
1000	0/25	6/25	23/25
10000	0/25	5/25	25/25

Table 5.12. Penalty weight tests: cont-GA-wl with $N_{pop} = 250$, $p_x = 0.6$, $G_{max} = 100$ and $p_m = 1/250$.

5.3.2.b Population size

Tests were performed using the combinatorial GA with local optimization and setting different population sizes while fixing the mutation probability to zero. In cases where only the global solution is sought (*i.e.*, no niching techniques are used), it was observed that over 1000 runs, a minimum of 20 individuals was required to get a 100% success rate in simple cases whereas $N_{pop} \geq 80$ was required for more complicated scenarios such as the battery and fixture problem (see Table 5.13). Similar tests were performed setting the probability of mutation inversely proportional to the population size. In those cases, particularly when the population was set relatively small (*i.e.*, $N_{pop} \leq 30$), the algorithms were able to find the global solution at an earlier generation as compared to the algorithms not using any mutation.

Large population sizes increase the robustness of the distance algorithms but come with a high computational expense. As a matter of fact, the number of computations needed to perform a test is linearly proportional to the population size while using

the basic GA. On the other hand, when sharing methods are used, the computational expense increases proportionally to the square of the population size since the calculation of the sharing function value needs to determine the distance between all possible node combinations. Additionally, large population sizes slow the convergence of the genetic algorithms since it takes longer to allow all the trial points to migrate to a minimum region.

N_{pop}	p_m	G_{max}	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}
6	0.01	20	914/1000	987/1000	4.66 [1 – 20], 5.36
10	0.02	20	950/1000	995/1000	2.49 [1 – 20], 3.68
20	0.03	20	979/1000	1000/1000	1.79 [1 – 20], 2.89
50	0.04	20	996/1000	1000/1000	1.17 [1 – 19], 1.27
80	0.05	20	1000/1000	1000/1000	1.02 [1 – 18], 0.55
100	0.01	20	1000/1000	1000/1000	1.00 [1 – 1], 0.00

Table 5.13. Population size tests: comb-GA-wl on the battery and fixture problem with $p_x = 0.6$.

5.3.2.c Number of generations

The theory behind Genetic Algorithms implies that GAs are inherently meant to improve a solution with time, thus, the longer a GA is left running the better the solution will become. While this was observed when no local optimization was used, it was found that, when Lamarckian algorithms are used, the algorithms would find the global minimum within the first 20 generations. As a matter of fact, the GA would often find the global minimum at the first iteration with most occurrences on or before the seventh generation (particularly when $N_{pop} \geq 80$). Thus, since the basic GA finds the global minimum in early iterations within the GA run, there is no need to have a large number of generations since no further improvements in the robustness as defined in Section 5.1.1 can be made.

Although the minimum distance algorithms were able to find the global minimum within the first few runs, the GAs take a couple of dozen generations to reallocate all the points in the population to a stable location. This is reflected by the time history of the average fitness illustrated in Figure 5.8. As seen in Figure 5.8, the simple GA and the GA using crowding, require more generations to converge than the GA with sharing. This accelerated convergence of the GA with sharing is due to the fact that, due to the local optimization technique, individuals in the population can find niches relatively quickly and most of these individuals will remain in those niches rather than having to migrate to the global minimum as in the simple GA. Also note that the comb-GA-wl-dc algorithm takes slightly longer to converge as compared to the simple GA due mainly to the fact that the deterministic crowding technique promotes diversity in the population.

Note that in Figure 5.8 the average fitness value at which the comb-GA-wl-sh converges is substantially larger that the one for the other two methods plotted in the same figure. This is due to the fact that the comb-GA-wl-sh locates some individuals at niches other than the global minimum which have a fitness higher that the actual global solution at which the comb-GA-wl algorithm converges.

5.3.2.d Crossover rate

Six tests were made varying the crossover probability (p_x) from 0 to 1. For each test the population size was set to 40 and the probability of mutation $p_m = 0.025$. Table 5.14 shows the results for these tests where it can be observed that the robustness (Φ^* and Φ) decreases as p_x increases. This decrease in robustness may be due to the fact that potentially good points are mated with points that are located in suboptimal locations thus producing inferior offspring.

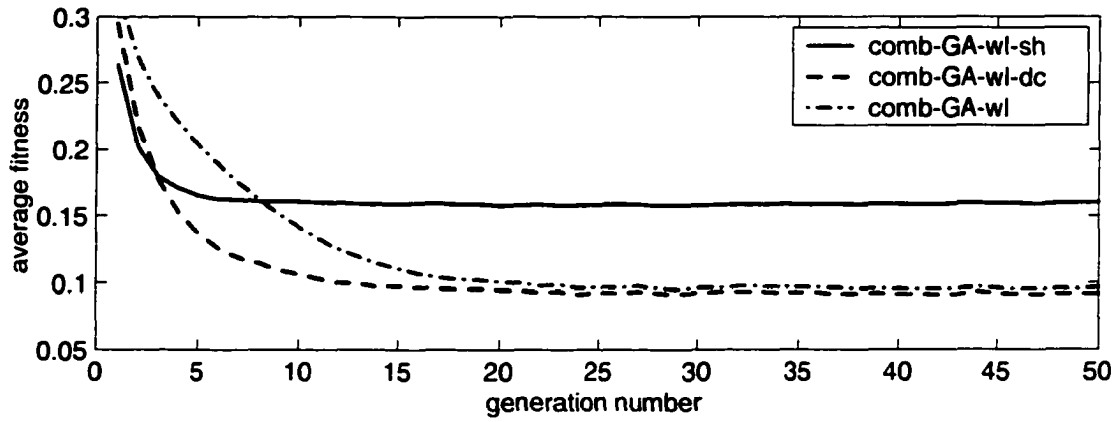


Figure 5.8. Time history, averaged over 200 runs, of the average fitness for comb-GA-wl, comb-GA-wl-sh and comb-GA-wl-dc all with $N_{pop} = 100$.

The same tests were repeated with mating restriction activated. For the tests shown in Table 5.15, the mating radius σ_m was set to 0.6. In this case, one can see that mating radius allowed the genetic algorithm to slightly increase its robustness Φ^*.

On the other hand, when no local optimization is used, the robustness of the combinatorial GA increases as the probability of crossover is increased. This is illustrated by the tests presented in Table 5.16. However, in all these cases, the robustness remains very poor.

Finally, as illustrated in Table 5.17, it was observed that, the cross over rate does not have any effect on the robustness for runs with population size of $N_{pop} \geq 80$ and making use of local optimization.

5.3.2.e Mutation rate

Several tests were performed in the combinatorial GAs in order to investigate the effect of the mutation rate (p_m) on the robustness of the algorithms. It was observed

p_x	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$\overline{x_e^{*\prime}}$	$\overline{x_e'}$
0	993/1000	1000/1000	1.33 [1 – 30], 2.3	1.00074	1.59093
0.3	990/1000	1000/1000	1.29 [1 – 27], 1.9	1.00071	1.82617
0.6	987/1000	998/1000	1.42 [1 – 15], 2.7	1.00116	1.89934
0.8	986/1000	998/1000	1.47 [1 – 29], 2.6	1.00174	1.90251
1	980/1000	998/1000	1.91 [1 – 30], 4.1	1.00423	1.91014

Table 5.14. Probability of cross over tests: comb-GA-wl using $N_{pop} = 40$, $p_m = 0.025$ and $G_{max} = 30$.

p_x	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$\overline{x_e^{*\prime}}$	$\overline{x_e'}$
0	993/1000	1000/1000	1.33 [1 – 30], 2.3	1.00074	1.59093
0.3	993/1000	1000/1000	1.31 [1 – 30], 2.2	1.00079	1.59319
0.6	990/1000	1000/1000	1.43 [1 – 29], 2.7	1.00120	1.58783
0.8	983/1000	998/1000	1.30 [1 – 27], 2.1	1.00117	1.58698
1	986/1000	999/1000	1.40 [1 – 28], 2.5	1.00105	1.58242

Table 5.15. Probability of cross over tests: comb-GA-wl using mating radius of 0.6 and $N_{pop} = 40$, $p_m = 0.025$ and $G_{max} = 30$.

that small mutation rates do not allow the GA to explore enough of the search space, particularly when the population size was relatively small ($N_{pop} \leq 30$). On the other hand, when using GAs with local optimization and large populations ($N_{pop} \geq 80$), the probability of mutation did not have any effect on the robustness of the distance algorithms. This is due to the fact that a large initial population is diverse enough that it does not require extra mutations in order to explore additional regions of the search space. Thus, in order to be able to better see the effects of varying p_m on the

p_x	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$\overline{x_e^{*\prime}}$	$\overline{x_e'}$
0	0/200	0/200	22.86 [1 – 50], 18	4.08056	7.79123
0.3	1/200	1/200	33.56 [1 – 50], 16	3.86606	7.68664
0.6	1/200	4/200	34.16 [1 – 50], 14	3.84648	7.56650
0.8	1/200	3/200	33.66 [1 – 50], 14	3.77177	7.59187
1	1/200	5/200	32.35 [1 – 50], 16	3.77728	7.57833

Table 5.16. Probability of cross over tests: comb-GA-nl with $N_{pop} = 100$, $p_m = 0.01$ and $G_{max} = 100$.

p_x	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}	$\overline{N_\Gamma}$ $[N_{\Gamma min} - N_{\Gamma max}]$, σ_{N_Γ}
0	200/200	200/200	1 [1 – 1], 0	9.19 [7 – 12], 0.69
0.3	200/200	200/200	1 [1 – 1], 0	9.16 [8 – 11], 0.64
0.6	200/200	200/200	1 [1 – 1], 0	9.15 [8 – 11], 0.65
0.8	200/200	200/200	1 [1 – 1], 0	9.16 [7 – 11], 0.72
1	200/200	200/200	1 [1 – 1], 0	9.13 [8 – 11], 0.68

Table 5.17. Probability of cross over tests: com-GA-wl-sh with $N_{pop} = 100$, $p_m = 0.01$ and $G_{max} = 15$.

robustness, the tests presented here used $N_{pop} = 20$.

Table 5.18 lists some results where the robustness measures as well as the iteration where the minimum point was found (*i.e.*, G_i) are reported for tests on the basic combinatorial GA with and without local optimization. The tests were performed on the ORU battery and fixture problem.

As expected, increasing the mutation rate increased the robustness of the algorithms. Additionally, the generation where the global minimum point is found G_i decreases as the probability of mutation is increased. This is due to the fact that the mutation allows the GA to explore new regions of the search space. However, very large mutation rates were found to considerably slow down the GAs using local optimization. This is due to the fact that every mutated points, which is not necessarily close to a local minimum, needs to be locally optimized.

In the present tests, it can be observed that a mutation rate of 0.05 was enough to produce good robustness results. Thus, it is concluded that for the combinatorial implementations using local optimization, the mutation rate needs to be set inversely proportional to the population size[4] (*i.e.*, $p_m = 1/N_{pop}$) in order to keep a good level

[4] In an extensive study by De Jong [De Jong, 1975], it was also concluded that, for function optimization tasks, the probability of mutation was best when inversely proportional to the population size [Goldberg, 1989a].

of exploration while keeping the computational expense at a reasonable level.

algorithm	p_m	G_{max}	Φ^*	Φ	$\overline{G_i}$ $[G_{i_{min}} - G_{i_{max}}]$, σ_{G_i}
comb-GA-nl	0	500	0/100	0/100	12.4 [1 – 73], 12.1
comb-GA-nl	0.02	500	0/100	7/100	346.7 [45 – 499], 116.9
comb-GA-nl	0.05	500	0/100	13/100	331.9 [24 – 500], 121.3
comb-GA-nl	0.1	500	0/100	26/100	289.8 [45 – 500], 127.1
comb-GA-nl	0.2	500	0/100	29/100	277.0 [45 – 500], 127.1
comb-GA-nl	0.5	500	0/100	23/100	283.9 [53 – 497], 126.1
comb-GA-wl	0	100	80/100	96/100	1.4 [1 – 7], 1.3
comb-GA-wl	0.02	100	99/100	100/100	8.8 [1 – 98], 18.5
comb-GA-wl	0.05	100	100/100	100/100	3.9 [1 – 58], 8.1
comb-GA-wl	0.1	100	100/100	100/100	2.5 [1 – 34], 4.2
comb-GA-wl	0.2	100	100/100	100/100	1.7 [1 – 11], 1.7
comb-GA-wl	0.5	100	100/100	100/100	1.4 [1 – 7], 0.9

Table 5.18. Probability of mutation tests: basic GA on the battery and fixture problem with $N_{pop} = 20$ and $p_x = 0.6$.

5.3.2.f Sharing parameters

This section presents the tests performed on the combinatorial GA with sharing in order to demonstrate the effects of the sharing radius σ_s and the sharing exponent α_s on the robustness and the number of niches. All tests use the default parameters listed in Table 5.8 with a population size of 100 individuals (*i.e.*, $N_{pop} = 100$).

Table 5.19 shows the results for five different tests where it can be seen that α_s has no apparent effect on the number of niches the comb-GA-wl-sh algorithm finds. In order to better see the effects of the sharing exponent α_s on the niches formed by the niching algorithm, the number of trial points in each niche needs to be analyzed. Table 5.19 also lists the average over 100 runs of the number of trial points allocated on the first five niches. As predicted, the distribution of trial points tends to be less homogeneous when $\alpha_s < 1$. That is, when $\alpha_s < 1$, the number of trial points

on the global minimum is much greater than the number of niches on all other local optima. On the other hand, when $\alpha_s > 1$, niches with fitness poorer than that of the global minimum tend to get more trial points than when $\alpha_s < 1$ creating a more homogeneous distribution of trial points. Additionally, as seen in Figure 5.9, values of $\alpha_s \geq 1$ allow the distribution of trial points to take place in fewer generations than when $\alpha_s < 1$.

Table 5.20 presents the results of tests performed on the comb-GA-wl-sh algorithm while varying the sharing radius σ_s. As explained in Chapter 4, σ_s should be determined as a function of the geometry, but the tests presented in Table 5.20 serve as an indication. As expected, as the niche radius is increased, trial points in distinct local minima are forced to share a single niche. This, in turn, yields to the loss of the least fit of the local minima thus reducing the total number of niches found by the GA.

α_s	$\overline{N_\Gamma}$ $[N_{\Gamma min} - N_{\Gamma max}]$, σ_{N_Γ}	Average number of trial points in first 5 niches				
		Γ_1	Γ_2	Γ_3	Γ_4	Γ_5
0.1	9.1 [7 – 11], 0.7	59.6	8.3	6.8	4.1	6.7
0.3	9.1 [8 – 11], 0.6	53.7	10.9	9.1	5.4	6.3
1	9.2 [8 – 11], 0.7	47.6	14.4	11.6	6.8	5.4
3	9.1 [8 – 11], 0.7	46.0	16.0	13.0	7.5	4.5
10	9.0 [7 – 11], 0.7	46.9	16.4	13.1	7.6	4.1

Table 5.19. Sharing parameters tests: effect of sharing exponent on 100 runs of the comb-GA-wl-sh algorithm for the battery and fixture problem.

5.4 Test conclusions

Many numerical tests were performed in order to evaluate the different minimum distance algorithms. While some tests helped compare the different algorithms relative

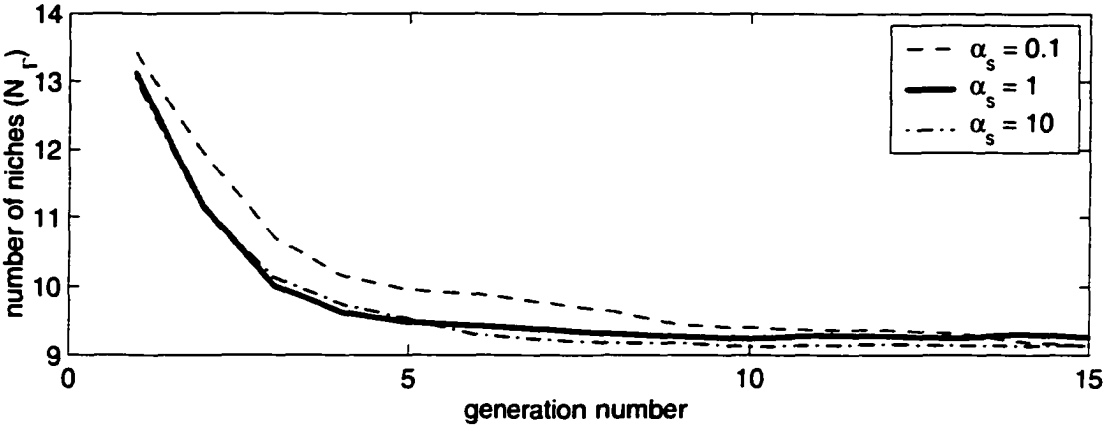


Figure 5.9. Time history of the average over 100 runs of the number of niches for three values of α_s.

σ_s	$\overline{N_\Gamma}$ $[N_{\Gamma min} - N_{\Gamma max}]$, σ_{N_Γ}	Average number of trial points in first 5 niches				
		Γ_1	Γ_2	Γ_3	Γ_4	Γ_5
0.3	9.2 [7 – 11], 0.6	51.5	11.8	9.8	5.7	6.3
0.6	9.2 [8 – 11], 0.7	47.6	14.4	11.6	6.8	5.4
1	7.5 [6 – 11], 0.9	50.0	16.3	13.1	7.6	4.8
1.5	6.4 [5 – 8], 0.6	57.1	16.8	12.6	3.3	4.9
3	2.9 [2 – 6], 1.2	88.8	10.1			

Table 5.20. Sharing parameters tests: effect of the sharing radius on 100 runs of the comb-GA-wl-sh algorithm for the battery and fixture problem.

to each other, others were targeted at improving their operating parameters.

Through this preliminary analysis it was possible to determine that, in a significant number of runs, the continuous GA implementations were not capable of locating the region where the minimum region is located in simple tests with up to 1500 individuals in the GA with a robustness higher than 95% for simple geometries. As well, after performing a series of tests on more complicated geometries, it was determined that the continuous GA was not capable of obtaining the region where the global minimum is located with the robustness higher than 28%.

Additionally, it was noticed that the continuous approach required on the order of 5 – 10 times (depending on the complexity of the geometries) more computations to perform the same task than its combinatorial counterpart. Furthermore, the number of computations required by the combinatorial algorithm was noticed to be proportional to the number of nodes on the grid but not on the complexity of the geometry as was the case of the continuous approach. Thus, due to the computational expense (as compared to the combinatorial approach) and its low robustness at finding the region where the minimum point is located, the continuous approach was deemed inferior to the combinatorial approach.

In separate tests, the SA and GA were compared to each other. In a large number of tests, the combinatorial SA was capable of obtaining the global solution with similar robustness to that of the combinatorial GA implementations. On the other hand, since every trial point in the SA runs is generated at random, the SA runs took substantially longer than the GA runs while using local optimization. The difference is mainly due to the fact that, after the first iteration that takes a relatively long time, the GA tends to keep the trial points in relatively good regions (except, of course, for the trial points that are mutated). Thus, trial points that are located in these relatively good regions do not need many iteration to be locally optimized. By contrast, the SA algorithms create a new random trial point, not necessarily near a local minima, at each temperature (iteration) which needs to be locally optimized.

Amongst the most important findings in this chapter are the ones related to the use of local optimization within the global algorithms. It was shown that the use of local optimization greatly improved the results of both formulations. Although the robustness of the cont-GA-wl algorithm was much better than that of cont-GA-nl, it was still much lower than the robustness of the comb-GA-wl which in most cases

was able to locate the global minimum within the mesh. This was particularly true for complicated scenarios where the local optimization method played a very large role in the value of the robustness of the overall minimum distance algorithms. As a matter of fact, as described in Section 5.3.1.c, while using local optimization at every iteration, most algorithms were able to find the global solution within the first 5 iterations.

The niching methods described in Chapter 4 were also compared. After a number of tests, it was observed that the sharing method was better suited to the present application, to find multiple minimum regions. That is, in the present tests and despite their lower computational expense, the GAs using deterministic crowding were not capable of obtaining all local minima whereas the GAs using sharing methods did. This is mainly due to the fact that the diversity of the population in the deterministic crowding implementation was not preserved probably due to a very high selective pressure.

Additionally, it was observed that the use of mating restriction within the GAs with sharing methods was necessary to give the algorithm the necessary ability to find all or at least most local minima.

In terms of the operating parameters for simple combinatorial GA with local optimization, the parameters listed in Table 5.21 are suggested in order to locate the global minimum region for a wide range of scenarios. On the other hand, if multiple minima are sought, the suggested operating parameters for the combinatorial GA with sharing are listed in Table 5.22.

Parameter	Symbol	Suggested value
population size (minimum)	N_{pop}	50
probability of mutation	p_m	$1/N_{pop} < p_m < 10/N_{pop}$
probability of crossover	p_x	0.6
max. number of generations	G_{max}	100

Table 5.21. Suggested operating parameter set for comb-GA-wl.

Parameter	Symbol	Suggested value
population size (minimum)	N_{pop}	100
probability of mutation	p_m	$1/N_{pop}$
probability of crossover	p_x	0.6
max. number of generations	G_{max}	30
sharing radius	σ_s	function of geometry
sharing exponent	α_s	$1 \leq \alpha_s \leq 10$
mating restriction radius	σ_m	$\sigma_s \leq \sigma_m \leq 1.5\ \sigma_s$
niche radius	σ_n	σ_s

Table 5.22. Suggested operating parameter set for comb-GA-wl-sh.

Chapter 6

Comparison of Concave Minimum Distance Algorithms

In this chapter, one of the minimum distance algorithms developed in this work is compared to a more conventional convex partitioning approach. As described in Chapter 1, some minimum distance algorithms formulate the problem as follows: place a point inside each of the objects in question, then, move the points in order to minimize the distance between them while at the same time ensuring the points remain within the bounds of their respective object (*i.e.*, a set of inequality constraints defining the objects' geometries are satisfied [Bobrow, 1989]). This approach has been shown to work well even in complicated scenarios [Nahon et al., 1998] but has the limitation that the objects considered must be convex. However, when dealing with concave objects, the user can break down the objects into convex pieces. Then, the distance between all sub-object pairs is obtained and the minimum of all distances is reported as the overall minimum distance between the two original objects.

This partition task is usually performed off-line, prior to the beginning of a simula-

tion. It was described in the introduction that methods relying on convex partitioning of concave objects typically do so by manually partitioning the concave objects. This is a tedious and error prone process. For this reason, a tool to automatically partition concave objects into convex sub-pieces is presented here. The tool is based on a commercially available software called GEOMPACK90 [Joe, 1994].

A relatively complex example is then used to compare the performance of different distance determination methods. First, the concave objects are partitioned into convex sub-pieces and the resulting objects are used in a convex minimum distance method based on Bobrow's approach [Bobrow, 1989] and implemented in Matlab. On the other hand, the concave objects are used, without partitioning, in one of the combinatorial minimum distance algorithm presented in Chapter 4. The results are then compared in terms of precision of the solution and computational efficiency.

In the example presented here, the minimum separation distance problem is solved continuously in a dynamic scenario as the objects approach each other. In order to accelerate convergence and to allow the minimum point(s) to be tracked as a function of time, the minimum distance problem is solved at each time step using the solution at the previous time step as the initial guess.

6.1 Partitioning approach

6.1.1 Minimum distance for convex objects

The minimum distance problem can be formulated as the following constrained optimization problem [Bobrow, 1989]:

$$\underset{\mathbf{p}_1,\mathbf{p}_2}{\text{minimize}} \quad : \sqrt{(\mathbf{p}_1 - \mathbf{p}_2)^T(\mathbf{p}_1 - \mathbf{p}_2)} \tag{6.1a}$$

$$\text{subject to} \quad : g_{1_j}(\mathbf{p}_1) \geq 0 \quad \text{and} \quad g_{2_j}(\mathbf{p}_2) \geq 0 \quad \text{for all } j \tag{6.1b}$$

where g_{i_j} corresponds to the j-th primitive of object i (*i.e.*, function that defines surface j on object i) and $\mathbf{p}_i$ is a point in or on object i if the equation (6.1b) is satisfied.

The minimization problem in equations (6.1) can be rewritten as:

$$\underset{\mathbf{x}}{\text{minimize}} \quad : \mathbf{x}^T\mathbf{W}_0\mathbf{x} \tag{6.2a}$$

$$\text{subject to} \quad : g_{i_j}(\mathbf{x}) \geq 0 \tag{6.2b}$$

where $\mathbf{W}_0 = \begin{bmatrix} \mathbf{I}_3 & -\mathbf{I}_3 \\ -\mathbf{I}_3 & \mathbf{I}_3 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix}$, since the point that minimizes $\mathbf{x}^T\mathbf{W}_0\mathbf{x}$ also minimizes $\sqrt{\mathbf{x}^T\mathbf{W}_0\mathbf{x}}$. This is only true because the Euclidean distance between two points is always positive, *i.e.*, $\mathbf{x}^T\mathbf{W}_0\mathbf{x} \geq 0$.

If objects 1 and 2 are only composed of linear surfaces, *i.e.*, planar surfaces, the problem becomes a LC-QP (Linearly Constrained-Quadratic Programming Problem). Whereas a more general case, where the objects are defined either by linear or quadratic surfaces, is a QC-QP (Quadratically Constrained-Quadratic Programming Problem).

Since a given object can have a large number of constraints defining its geometry, it is easier to express g_{1j} and g_{2j} in body fixed frames. On the other hand, the points in the objective function have to be expressed with respect to a common frame, *e.g.* the *inertial frame* (*i.e.* a fixed frame, usually called frame 0).

$$[\mathbf{p}_1]_0 = \mathbf{R}_{01}[\mathbf{p}_1]_1 + [\mathbf{o}_1]_0 \tag{6.3a}$$

$$[\mathbf{p}_2]_0 = \mathbf{R}_{02}[\mathbf{p}_2]_2 + [\mathbf{o}_2]_0 \tag{6.3b}$$

where $\mathbf{R}_{01}$ and $\mathbf{R}_{02}$ correspond to the rotation matrices to express the components of a vector in body frame 1 and 2, respectively, with respect to the inertial frame. Additionally, $[\mathbf{o}_i]_0$ corresponds to the Cartesian coordinates of the origin of body frame i with respect to the origin of the inertial frame, *i.e.*, frame 0. Note that $[*]_i$ denotes that the vector or matrix quantity $*$ is expressed in terms of frame i.

Thus, for a set of linearly constrained objects the minimum distance problem can be reformulated as [Ma and Nahon, 1992]:

$$\underset{\mathbf{x}}{\text{minimize}} \quad : \mathbf{x}^T\mathbf{W}\mathbf{x} + \mathbf{w}^T\mathbf{x} \tag{6.4a}$$

$$\text{subject to} \quad : \mathbf{A}\mathbf{x} - \mathbf{b} \geq \mathbf{0} \tag{6.4b}$$

where the weighting matrix $\mathbf{W}$ and the weighting vector $\mathbf{w}$ are calculated as:

$$\mathbf{W} = \begin{bmatrix} \mathbf{I}_3 & -\mathbf{R}_{01}^T\mathbf{R}_{02} \\ -\mathbf{R}_{02}^T\mathbf{R}_{01} & \mathbf{I}_3 \end{bmatrix} \tag{6.5}$$

$$\mathbf{w} = \begin{bmatrix} \mathbf{R}_{01}^T[(\mathbf{o}_1 - \mathbf{o}_2)]_0 \\ \mathbf{R}_{02}^T[(\mathbf{o}_1 - \mathbf{o}_2)]_0 \end{bmatrix} \tag{6.6}$$

where $\mathbf{I}_3$ is an identity matrix of dimension 3. On the other hand, matrix $\mathbf{A}$ and vector $\mathbf{b}$ in equation (6.4b) determine the linear constraints that define objects 1 and

2 and are defined as

$$\mathbf{A} = \begin{bmatrix} [\mathbf{A}_1]_1 & \mathbf{0} \\ \mathbf{0} & [\mathbf{A}_2]_2 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} [\mathbf{b}_1]_1 \\ [\mathbf{b}_2]_2 \end{bmatrix}$$

where $[\mathbf{A}_i]_i[\mathbf{p}_i]_i - [\mathbf{b}_i]_i \geq \mathbf{0}$ defines all the linear constraints for object i and the number of rows of $[\mathbf{A}_1]_1$ is m and of $[\mathbf{A}_2]_2$ is n.

The optimization problem represented by equations (6.4) can be solved using convex quadratic programming algorithms (see for instance [Goldfarb and Idnani, 1983, Gill et al., 1981]). However, it is emphasized again that this approach will only work if both objects are convex.

6.1.2 Automatic partitioning of concave objects

This section briefly describes a software tool developed to automatically perform the decomposition of concave objects into convex polyhedral sub-objects using a modified version the algorithm described in [Chazelle, 1984] and implemented in GEOMPACK90 [Joe, 1994].

6.1.2.a Method

In order to decompose a concave polyhedron into convex polyhedral sub-objects, the algorithm implemented in GEOMPACK90 first searches for edges with an interior dihedral angle (θ) greater than π. The object is then cut by a plane containing the reflex edge e (*i.e.*, edge with $\theta > \pi$) and containing at least one edge intersecting e. That is, the cutting plane would contain face F_1 or face F_2 that create reflex edge e.

Based on a cutting plane selection criterion (thoroughly described in [Joe, 1994]), other cutting planes such as the bisecting plane between F_1 and F_2 are used in order to reduce the number of polyhedra produced in the slicing process.

6.1.2.b Geometry Toolbox

In order to include in a single package several tools for handling complex geometries, the Matlab Geometry Toolbox (MGT) was created. This toolbox was developed, for the most part, in Matlab and includes a Graphical User Interface (GUI) to facilitate its use. Amongst other capabilities, this tool allows the user to partition concave objects into simpler convex sub-objects using GEOMPACK90 [Joe, 1994]. Additionally, MGT allows the user to visualize objects without leaving the Matlab environment. Following is a list of the capabilities that were included in the MGT.

- **Importing capabilities:** a filter was written in AutoLisp to transform AutoCAD solid objects into a customary format in Matlab. An analogous filter was also created using MAXScript to export 3DStudio MAX geometrical models into Matlab. The MGT can also read the file format generated by GEOMPACK90. This is particularly useful to read the geometry processed by this package such as the convex partitioned geometries.

- **Exporting capabilities**: a filter to generate files for input to GEOMPACK90 was implemented. Algorithms to obtain vertex, edge and face connectivity had to be developed in Matlab. Also, filters to translate from the Matlab customary format into AutoCAD or 3DStudio MAX were developed for the MGT.

- **Simplify geometry**: Once in Matlab, the objects (particularly the ones coming from 3D Studio) are simplified since the format they are exported in contains

many redundant vertices and edges[1]. Simplifying the geometry results in a cleaner object without redundant faces, edges or vertices.

- **Visualization**: the MGT uses the powerful Matlab graphics to display three-dimensional objects and allow the user to rotate them in order to verify any geometrical operation such as the convex partitioning.

- **Convex partitioning**: once the objects are in GEOMPACK90 format, the objects are partitioned inside GEOMPACK90 and the results imported back into Matlab for verification.

The Matlab Geometry Toolbox is a powerful tool that greatly facilitates the manipulation, visualization and decomposition of three-dimensional objects in Matlab. Of particular interest is the toolbox's capability to partition concave bodies into simpler convex objects, thus allowing the use of conventional tools for applications such as distance determination.

Figures 6.1 to 6.3 show examples of partitioned objects. Notice that, in order to partition objects into convex polyhedra, all cylindrical parts first need to be linearized, thus, the resulting number of convex sub-objects will greatly depend on the level of detail needed to represent quadratic or higher order surfaces of a concave body.

Although the convex partitioning tool works reliably for a wide range of geometries, there are some geometries that are not properly partitioned. This is the case of the object illustrated in Figure 6.3 where the sub-object on the foreground was not successfully partitioned. This is a limitation of the GEOMPACK90 partitioning

[1] 3DStudio Max uses a linearized and triangulated version of the objects to store an object's geometry.

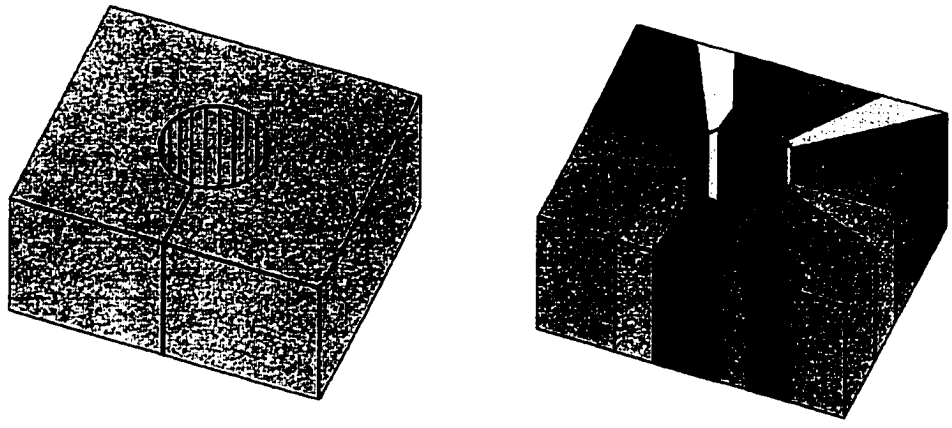


Figure 6.1. Object with a cylindrical hole, linearized and successfully partitioned into 24 subobjects.

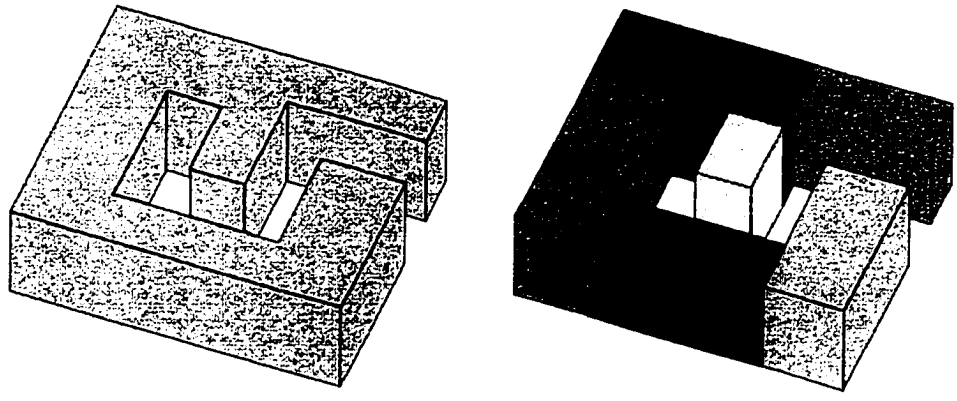


Figure 6.2. Object with concavities successfully partitioned into 6 subobjects.

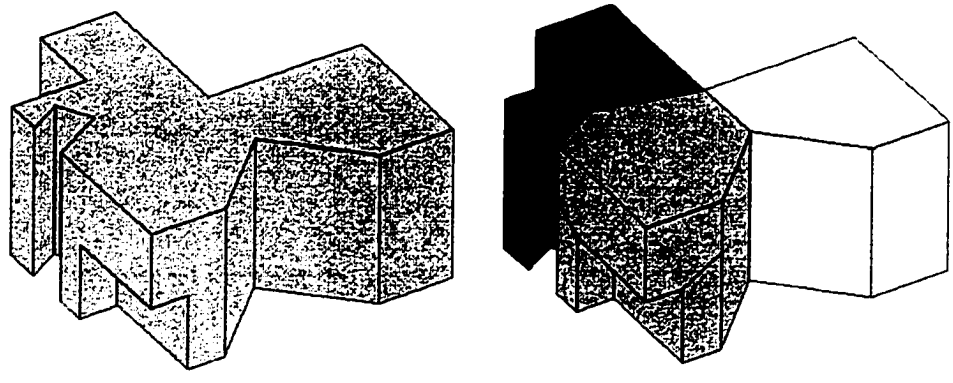


Figure 6.3. Object with concavities partitioned into 5 subobjects. Notice that the element on the foreground was not successfully partitioned.

engine. To overcome this problem, it is proposed to partition the objects using the automatic tool and visually inspect if any part is still concave. If this is the case, it was observed that, in a few cases, reprocessing that part through the partitioning algorithms actually finished the job. If that still leaves concave sub-objects, a manual partition of that particular sub-object would be needed.

6.1.3 Implementation details

The solution of the minimum distance problem described by equations (6.4) was implemented in Matlab using the quadratic programming routine **quadprog** included in Matlab's Optimization Toolbox [Grace, 1992][2]. The result of the optimization routine consists of the Cartesian coordinates of the point pair (each one expressed in terms of body fixed coordinates) that minimize the distance between the object pair.

To solve the minimum distance problem for concave objects, all possible convex sub-object pairs are passed to the minimum distance algorithm, one set at a time. Thus, if object 1 has b_1 convex sub-objects and object 2 has b_2 convex sub-objects, the outcome of the minimum distance determination between the concave objects would consist of all $b_1 \times b_2$ point pairs and all $b_1 \times b_2$ distances. That is, the outcome of the overall minimum distance problem would consist of a $b_1 \times b_2 \times 6$ three dimensional matrix containing all optimal trial point pairs and a second $b_1 \times b_2$ matrix containing the minimum distance between all sub-object pairs. Additionally, for the overall minimum distance problem, the sub-object pair indices that corresponds to the closest sub-objects would be desirable.

[2]The **quadrpog** routine uses an active set method similar to the one described in [Gill et al., 1981].

6.2 Dynamic minimum distance calculation

In order to evaluate the minimum distance algorithms in a more realistic environment, the minimum distance problem can be solved dynamically. That is, the two objects move relative to each other and the minimum distance between them is calculated at fixed time intervals along the trajectory of the objects. Thus, instead of solving the minimum distance with an initial guess generated at random, as was done in previous chapters, only at the first iteration will the initial point be generated at random. In all other subsequent calls, the minimum distance algorithm will use the solution point at the previous time step as the initial guess for the next iteration. This method has been proven to accelerate the convergence process of the minimum distance algorithm presented in [Ma and Nahon, 1992].

6.2.1 Partitioned objects method

At every step, the solution of the minimum distance for all sub-object pair combinations is obtained. As mentioned in Section 6.1.3, the solution of the minimum distance for the convex partitioned method consists of the Cartesian coordinates of all the trial points that minimize the distance between all sub-objects. Thus, each of the solution points from the previous time step, stored in the $b_1 \times b_2 \times 6$ matrix, can be used as the initial points for the next iteration greatly reducing the computational expense of initiating the search from a random location.

6.2.2 Combinatorial concave method with niches

As described in Section 4.6.4, in the case of the combinatorial concave method using niching techniques, the solution found at any time consists of a $N_\Gamma \times 2$ matrix where

row i represents the trial point with minimum distance in niche Γ_i. Typically, the number of niches would be substantially lower than the size of the population, *i.e.*, $N_\Gamma \ll N_{pop}$. As a result, the initial population for the next GA run would have N_{pop_Γ} previously generated trial points and the remaining $N_{pop} - N_{pop_\Gamma}$ trial points would be randomly generated trial points.

One method to determine the part of the population that is passed on would be to put a fixed number of trials points for each niche, *e.g.*, two for each niche so that $N_{pop_\Gamma} = 2N_\Gamma$. That is, in a population with 100 individuals in a problem with 10 niches, the initial population for a second or subsequent iteration could have 20 trial points corresponding to 2 points per niche and 80 randomly generated trial points to form the entire initial population for the second GA run. Thus, at every time step, the search algorithm will still have a large amount of exploration in order to look for new evolving minima.

On the other hand, a more elaborate scheme could be devised to keep a number of trial points at each niche proportional to their fitness. That is, the number of trial points per niche would be variable according to the expected number of trial points at that niche (see equation (4.4) in Section 4.3). That is, the number of trial points allocated to niche i can be calculated using:

$$N_{niche_i} = \frac{N_{pop_\Gamma} f_{niche_i}}{\sum_{j=1}^{N_{\Gamma pop}} f_{niche_j}} \tag{6.7}$$

where N_{pop_Γ} is the size of the portion of the population assigned to the niches and f_{niche_i} corresponds to the potential fitness at niche i.

To allow the GA to maintain a large exploration capability, a small enough N_{pop_Γ} has to be used. On the other hand, N_{pop_Γ} has to be large enough to allow the algorithm to keep all the niche points the GA was able to find in the previous runs.

For this purpose, N_{pop_Γ} can be set proportional to the number of niches (N_Γ) found in the previous population up to a maximum number of trial points ($N_{pop_{\Gamma max}}$). That is:

$$N_{pop_\Gamma} = \begin{cases} \alpha_\Gamma N_\Gamma, & \alpha_\Gamma N_\Gamma \leq N_{pop_{\Gamma max}} \\ N_{pop_{\Gamma max}}, & \text{otherwise} \end{cases} \tag{6.8}$$

where $\alpha_\Gamma \geq 1$ is the average number of trial points to inherit per niche from one GA run to the next and $N_{pop_{\Gamma max}}$ is the maximum total niche points to inherit.

Due to the nature of the GA with niche formation, particularly when using sharing methods, it was determined that a fitness-proportional method to determine the inherited portion of the initial population would better benefit the convergence of the GA as compared to a fixed number of points per niche.

Regardless of the method used to determine the initial population, once the initial population has been determined and before starting the new GA run, the entire population would have to be locally optimized. Note that the part of the population coming from the previous run would also need to be locally optimized since the minimum point locations might have shifted due to the relative motion of the objects.

6.3 Numerical example

A numerical example is presented here to compare the novel combinatorial method described in previous chapters to the convex partitioning method described in Section 6.1. This comparison is carried out in terms of computational efficiency and the precision of their solution.

6.3.1 Description of the geometries

The geometries used for this example are the same as the ones presented in the third example of Chapter 4, that is, the ORU battery and fixture example.

Figure 6.4 illustrates the partitioned objects used to solve the minimum distance problem as formulated and described in Section 6.1. Notice that in order to partition the battery, the cylindrical holes on the battery were first linearized. For this purpose, 6 faces were used yielding a partitioned battery with 14 sub-objects. Thus, with a fixture partitioned into 9 sub-objects, the convex minimum distance algorithm needs to solve a total of 126 minimum distance sub-problems and return the minimum of all 126 distances. However, if a more exact representation of the holes in the battery was needed, the number of convex sub-pieces would increase substantially. In the present case, the number of sub-objects for the battery is given by $2n + 2$ where n is the number of faces used to linearize each cylindrical hole. That is, if a more precise representation of the holes with 20 faces (instead of 6) was needed, the total number of sub-object pairs to be checked would be $9 \times (2 \times 20 + 2) = 378$.

Likewise, in the case of the combinatorial approach where the objects are represented by a surface mesh, the cylindrical holes are inherently linearized by the meshing process. Thus, if a more exact representation of the holes in the battery was needed, the grid density in and around the holes should be increased. This would increase the total number of nodes in the mesh but would increase the computational time very slightly since the hole regions are relatively small compared to the rest of the battery.

To test the implementation of the combinatorial approach, surface meshes were used. The meshes representing the battery and fixture for this example are the same

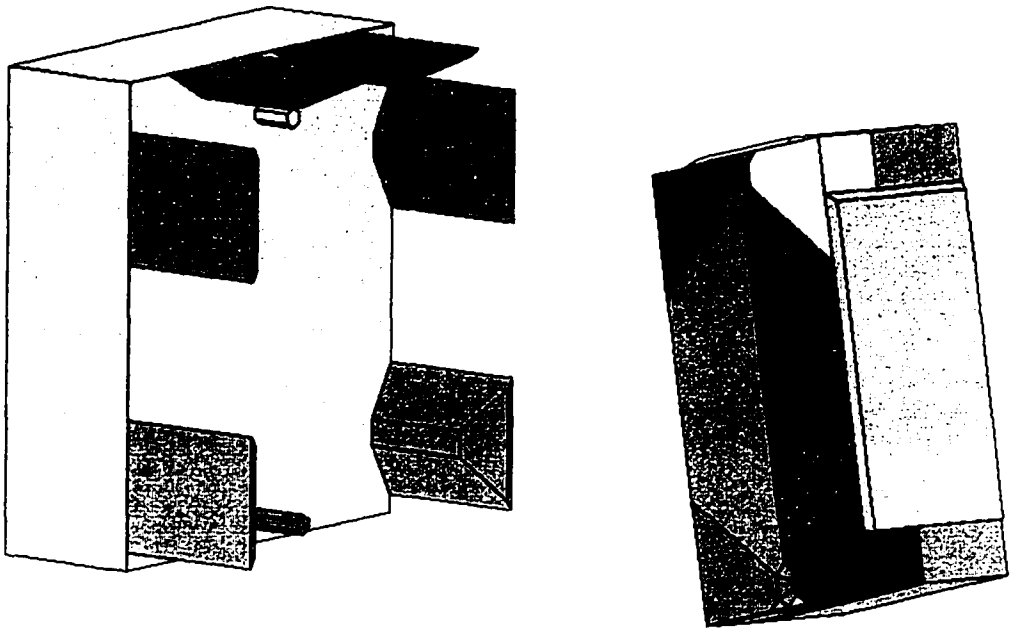


Figure 6.4. ORU battery and fixture geometries partitioned into 14 and 9 convex subobject, respectively.

as those described in Section 4.7, that is, the battery's mesh contains 1241 nodes whereas the fixture's mesh contains 1842 (see Figure 4.11).

6.3.2 Objects' trajectories

The trajectory simulated in the this example is one where the fixture remains fixed and the battery approaches it. The time history of the position and orientation of the battery with respect to the fixture for the entire manoeuvre is shown in Figure 6.5. In this simulated task, the battery translates and rotates with respect to the fixture before reaching the insertion configuration where the battery is inserted in the fixture. Since the algorithms presented in this work do not handle interference situations, the entire manoeuvre was carefully designed in order to have a collision free path at all times.

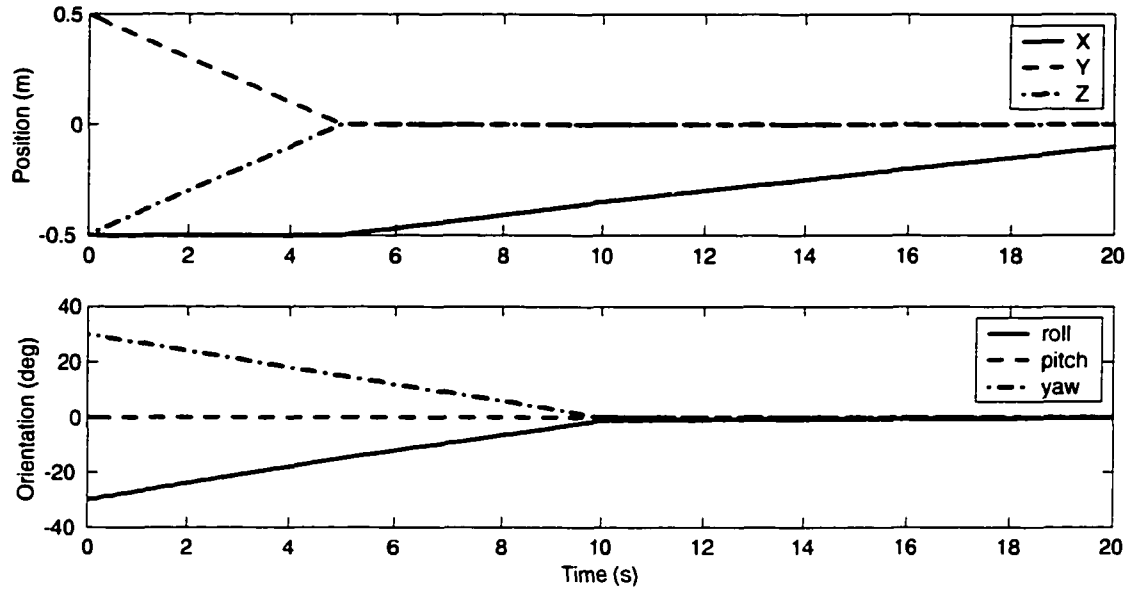


Figure 6.5. Time history of the position and orientation of the battery with respect to the fixture.

The entire manoeuvre was designed to take 20 seconds. In the first 10 seconds, the battery approaches the fixture and in the last 10 seconds the battery is inserted in the fixture. The distance calculations will be performed at 10 Hz thus a total of two hundred minimum distance problems will be solved.

6.3.3 Numerical results

To solve the concave minimum distance problem, the algorithm used in the present example is the one referred to as comb-GA-wl-sh (see Section 4.7). The optimization parameter values were set according to the values recommended in Chapter 5. The GA parameters as well as the operator types used for the numerical simulation are listed in Table 6.1.

Figure 6.6 illustrates the minimum distance between the battery and fixture for the entire manoeuvre for both algorithms presented in this example. Additionally,

Parameter / Operator	Symbol	Value / Type
population size	N_{pop}	100
probability of mutation	p_m	0.01
probability of crossover	p_x	0.6
max. number of generations	G_{max}	20
mating radius (phenotypic)	σ_m	0.6
sharing radius	σ_s	0.6
sharing exponent	α_s	1
niche radius	σ_n	0.6
ave. niche points to inherit per niche	α_Γ	3
max. total niche points to inherit	$N_{pop\Gamma_{max}}$	30
number of niche points to inherit		fitness proportional method
selection		SUS
mating		mesh mating
mutation		random move
local optimization		Lamarckian evolution

Table 6.1. Operating parameters and genetic operators for algorithm comb-GA-wl-sh used in dynamic minimum distance example.

Figure 6.7 shows the difference between the results obtained using the quadratic programming method and the GA method. The results obtained by comb-GA-wl-sh were also compared to the ones obtained by enumeration and the error is also presented in Figure 6.7. The difference between the results of the enumeration and the comb-GA-wl-sh algorithm was equal to zero except in three instances with a maximum difference of 9.88×10^{-4} at $t = 19.9$ s. It can be noticed that, at all times, the minimum distance is slightly overestimated while using the combinatorial GA approach. This difference is due to the discretization of the surface into a mesh and, as discussed earlier, can be reduced but not eliminated, with some penalty in the computational expense, by increasing the grid density.

It is important to notice that the comb-GA-wl-sh algorithm tends to return a slightly larger difference, relative to the convex method, when the objects are close

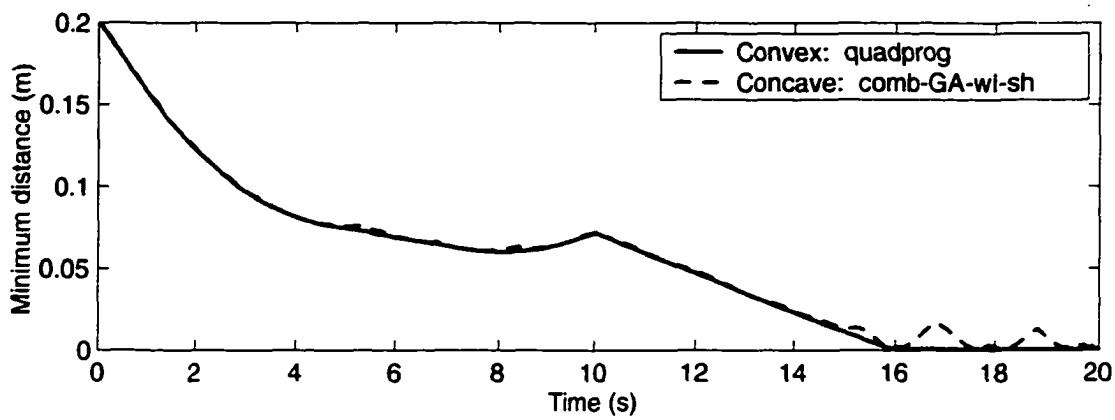


Figure 6.6. Time history of the minimum distance between the battery and the fixture using the convex paritioning method and the combinatorial GA with sharing method.

to each other. That is the case for the insertion phase of the trajectory between $t = 10$ s and $t = 20$ s in Figures 6.6 and 6.7. As seen in Figure 6.7, this effect on the difference is also present when the minimum distance is obtained by enumeration which indicates that it is mainly due to the density of the grid.

It was noted that, for the initial run at $t = 0$, the number of iterations taken by the quadratic programming algorithm was, for all 126 pairs of sub-objects, within the range [4 13] with an average number of iterations equal to 7.38. As a result of passing the initial guess from the previous time step to the algorithm in subsequent evaluations, the number of iterations taken by the quadratic programming algorithm was reduced by about 25%. That is, the number of iterations per run for each sub-object pair was in the range [4 9] with an average number of iterations equal to 5.52. The reduction in the average number of iterations per sub-object pair results in a reduction of the overall computational expense of the algorithms of close to 25%.

For the combinatorial algorithms, on the other hand, the computational expense did not change significantly by passing on the initial population for runs where $t >$

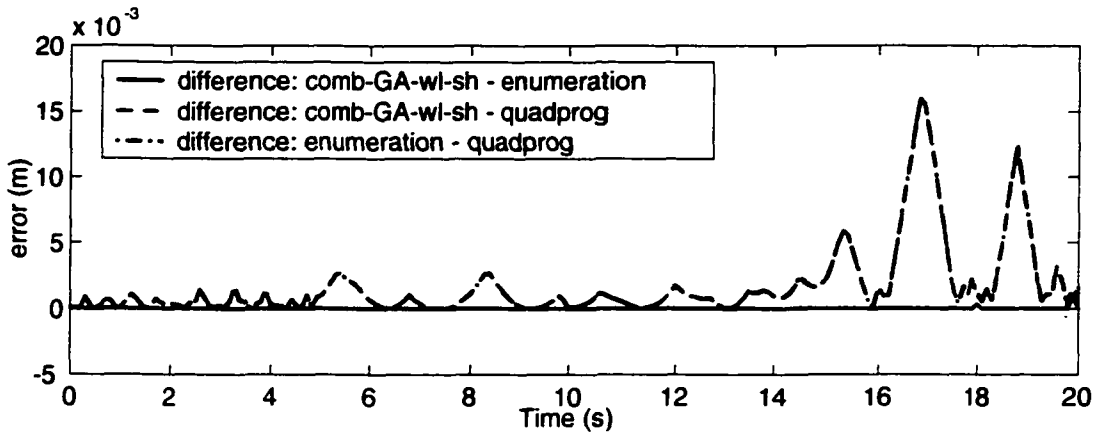


Figure 6.7. Time history of the error between the minimum distance obtained by the different methods.

0. Careful analysis of individual GA results for some algorithm runs at random locations along the trajectory revealed differences in the convergence process of the GA. It was noted that, after the initial run at the start of the manoeuvre, where the initial trial points are generated at random, the GA algorithm converged at an earlier iteration. While in the initial run the algorithm converged close to the 20th iteration, in the second and subsequent runs the algorithms converged on or before the 15th iteration. In future implementations, this would allow the use of a different value for the maximum number of generations (G_{max}) at the initial iteration than during the rest of the simulation thus saving some computational expense.

The number of niches during the entire simulation using the comb-GA-wl-sh algorithm was found to be, on average, 7.93 niches. Thus, to ensure that most local minima were tracked over time, the dynamic distance calculation method introduced, at each GA run, an average of 24 individuals (*i.e.*, $N_{pop_\Gamma} = \alpha_\Gamma N_\Gamma = 3 \times 7.93 \approx 24$). Additionally, it was noticed that at the end of the simulation the number of niches slightly increased. In a few cases N_Γ reached 11 in which case, the algorithm had

to constrain the number of points passed to the next run to $N_{pop_{\Gamma max}} = 30$. This phenomenon is probably due to the fact that, at the end of the manoeuvre, the two objects are only a few millimeters away from each other and with two large surfaces very close to parallel. This creates a large weak minimum area, thus offering a perfect scenario to produce a large number of niches.

In terms of computational expense, the entire simulation with 126 sub-body pairs took close to 309.1 Mflop for the convex partition method for an average of 1.545 Mflop at each time step. On the other hand, the comb-GA-wl-sh algorithms took 406.4 Mflops to compute the minimum distance for the entire trajectory of the objects for an average of 2.032 Mflop at each time step. For both algorithms, the initial run took longer than the subsequent runs with a more significant reduction for the convex partition method. That is, the initial run for the convex partition method took close to 2.27 Mflop whereas comb-GA-wl-sh performed 2.16 Mflop.

6.3.4 Discussion

In this preliminary study, the combinatorial GA implementation using sharing techniques demonstrated, for a realistic problem, to be capable of solving the minimum distance problem in a similar amount of computations as the conventional convex partition method.

Further improvements to the comb-GA-wl-sh could be made to improve its computational efficiency. One such improvement would be the incorporation a termination condition that will enable the algorithm to return a value for the minimum distance before the total number of generations has passed. Furthermore, the

use of generation gaps [De Jong and Sarma, 1992][3] and/or dynamic niche sharing [Miller and Shaw, 1996][4], could prove beneficial in increasing the computation efficiency of the comb-GA-wl-sh algorithm. Additionally, further speed comparison in a lower-level language should be made in order to better determine the new algorithm's capabilities.

Although there is still lots of room for improving the computational efficiency of the methods proposed in this work, the preliminary results shown in this chapter suggest that, in environments with 150 sub-objects or more, the novel method presented in this work outperform the more conventional optimization-based convex partitioning method. That is, by linearly scaling the average number of floating point operations required by the optimization-based convex partitioning method of close to 13 kflops per body pair by 150 pairs it comes close to 2 Mflop which is the average number of computations required by the comb-GA-sh-wl algorithm.

Additionally, it is important to realize that the two methods presented in this chapter complement each other rather than compete against each other. That is, the novel methods presented in this work solve the distance problem approximately thus, if an exact solution was needed, a more exact distance determination method, such

[3]Genetic Algorithms with generation gap use, in each iteration of the GA, a subset of the entire population to perform the genetic operations rather that the entire population thus reducing the overall computational cost.

[4]GAs with dynamic niche sharing are a variation of the GAs with regular sharing. The main difference is that, in dynamic niche sharing, the computational expense of obtaining the sharing function value at each iteration is reduced by identifying the niches at each iteration and assigning to all the trial points in each niche a predetermined fitness value. In [Miller and Shaw, 1996], the authors claim a 50% reduction of the computational expense as compared to the regular sharing method.

as that of Bobrow [Bobrow, 1989], would be needed. The methods introduced in this work could actually function as a sophisticated pruning strategy in order to determine the smallest subset of sub-objects possible in order to check for the minimum distance. In such cases, a coarser grid might be used to further accelerate the operation of the implementations of the combinatorial methods proposed in this work.

Chapter 7

Conclusions

Two different approaches to solve the minimum distance problem between convex and/or concave bodies were proposed. Both approaches formulate the minimum distance problem as an optimization problem. That is, for two objects A and B, find the point on object A closest to object B and *vice versa*.

The first formulation, referred to as the *continuous approach*, uses concepts of computational solid geometry in order to represent objects with concavities. On the other hand, in the second formulation, referred to as the *combinatorial approach*, the geometries of the objects are replaced by sets of points arranged in surface meshes.

In both formulations, the optimization problem needs to be solved using global optimization techniques since the objective function, *i.e.*, the distance between the two objects, can be multimodal when dealing with concave objects. As a result, global optimization algorithms, particularly Genetic Algorithms and Simulated Annealing algorithms, were designed and implemented to solve the minimum distance problem.

For the continuous approach, the global optimization algorithms use penalty strategies in order to deal with the constraints representing the geometry of the

objects. That is, trial points that do not satisfy the constraints are still accepted but their objective function value is penalized proportionally to their distance to the feasible region. Additionally, a local optimization method was developed in order to accelerate the convergence process of the continuous global algorithms. The local optimization method also works as a repair strategy when the trial points are located outside the feasible region by moving them closer to it.

It was shown that, within the optimization routines for the continuous approach, 75% of the computational expense is consumed by the verification of the trial point's feasibility. Thus, in order to accelerate the computation of the minimum distance between two objects, the combinatorial method was proposed.

In the combinatorial approach, the geometry of the objects is replaced by a mesh. Thus, all the mesh nodes are guaranteed to be feasible. Additionally, the mesh is generated off-line. Therefore, the computational expense related to the generation or verification of feasible trial points is eliminated. As a result, the global optimization problem becomes one of finding the pair of nodes, one on each body, that minimizes the distance between the two nodes. To solve this problem, combinatorial GA and SA algorithms were designed and implemented. For the combinatorial GA algorithm, a novel mating method, referred to as *mesh mating*, was proposed in order to allow the GA to pass on genetic traits from one generation to the next.

In order to accelerate the convergence process of the combinatorial methods, local optimization methods were again proposed in order to improve all points locally. As seen in the numerical examples, these local optimization techniques greatly improved the solution of the GA and SA. As a matter of fact, when using local optimization within the combinatorial GA implementations, the global solution was found within the first couple of iterations for a fair variety of geometries and poses.

Both novel formulations were demonstrated, through a series of numerical tests, to be capable of locating the region where the points that minimize the distance between the objects is located. The Genetic Algorithms based on the combinatorial approach proved to be the most robust and reliable.

Due to its computational efficiency, the combinatorial approach was selected to be further developed in order to allow the minimum distance algorithm to find more than just the closest pair of points. That is, rather than looking for a single set of closest points, *i.e.*, the global minimum, the algorithm looks for other local minima as well as the global minimum. As demonstrated by a few numerical examples, this was successfully achieved using Genetic Algorithms with niche formation techniques where the best results were obtained using GAs with sharing methods.

In this study, the combinatorial GA implementation using sharing techniques was shown, for a realistic problem, to be capable of solving the minimum distance problem in a similar amount of computations to the conventional convex partition method (when using about 150 sub-body pairs in the current implementation). A more optimized implementation of the methods described in this thesis may reduce the time spent to find the solution. This reduction in computational time is important for real time simulations in applications such as contact dynamics and path planning.

Although the proposed methods do not solve the exact minimum distance problem, they could be used in conjunction with a conventional convex algorithm in order to locate the exact minimum. That is, the novel methods proposed in this thesis would be used to determine the sub-object pairs to process with a conventional quadratic-programming-based minimum distance algorithm. Thus, the number of sub-body pairs to be used to solve the convex minimum distance problem would be greatly reduced. This would be particularly useful in scenes with a large number of sub-

object pairs.

7.1 Future Work

The following are suggested topics for future research in the area of distance determination. While some of the following suggestions represent extensions to the current work, others propose modifications to reduce the computational expense taken by the new methods presented here.

1. **Extend to interference situations**

   In order to allow the current algorithms to be used in contact dynamics applications, identification and analysis of the collision points/regions would be necessary. The use of meshes in the combinatorial approach could be exploited in order to determine the geometrical properties of the contact region (*e.g.*, contact surface and/or volume). This may then allow more accurate calculation of contact forces.

2. **Sweep and prune algorithms**

   In earlier works (*e.g.*, [Cohen et al., 1995]), sweep and prune algorithms have been used to reduce the number of objects or features to be used in the distance calculations. These algorithms have yet to be developed for the optimization-based approaches and could prove to substantially reduce their computational expense.

3. **Velocity information**

   If dynamic distance determination is of interest, the initial guess for the optimization algorithms could be better approximated if the velocity and accelera-

tion of the solution points were taken into account. Although this has already been suggested in the literature (see for instance [Nahon, 1993]) it has yet to be implemented.

4. **Finding the exact solution**

   Although the proposed methods proved to be very robust at finding the region where the global minimum is located, they do not return an exact solution. Using a list of faces that surround the solution point at the global solution, it would be possible to create a hybrid algorithm that would determine the exact solution of the minimum distance problem using quadratic programming algorithms.

5. **Improvements to the current implementation**

- **Parallel genetic algorithms**

  An important asset of some global optimization algorithms, such as the GAs, is that their implementation can be easily parallelized [Bäck, 1996], rendering the algorithms much faster. Population size [Goldberg, 1989b] as well as the use of niche formation strategies should be investigated in the context of parallel GAs.

- **Generation gaps**

  The use of generation gaps [De Jong and Sarma, 1992], *i.e.*, the use of a small subset of the population at each generation of the GA, has been proven to reduce the computational intensity of the overall GA while at the same time allowing the niching algorithms to keep track of multiple minima [Grunwald, 1999].

- **Dynamic niche sharing**

In GAs using sharing, the computational expense required to determine the shared fitness value takes a large proportion of the overall computational expense. Dynamic niche sharing [Miller and Shaw, 1996] could potentially reduce this expense and should be investigated.

- **Termination condition**

  As seen in Chapter 5, some of the proposed GAs converge within the first 20 generations. To improve the GA's efficiency, a termination condition other than a fixed number of iterations could be designed. This would minimize the time taken by each GA run by detecting when the algorithm has converged.

- **Variable mutation rate**

  Although in the present study the mutation probability remains constant during a run, it has been shown that using a variable mutation probability (*e.g.*, with a Boltzmann type variation) can be beneficial in some applications [Mahfoud and Goldberg, 1995, Pham and Karaboga, 1997].

- **Local optimization method in combinatorial formulation**

  The computational expense of the local optimization method in the combinatorial formulation could potentially be reduced (currently up to 80% of the overall computations). That is, the local optimization algorithm could be given the possibility to move along the mesh more than a single node at a time. For this purpose, the use of different types of meshes (*e.g.*, structured meshes) as well as alternate methods for mesh storage should be investigated.

References

[Aarts and Korst, 1989] Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley and Sons, Chichester.

[Abdel-Malek and Burton, 1994] Abdel-Malek, K. A. and Burton, P. (1994). Interference detection of non-convex solids for manipulators. In *Proceedings of ASME Conference on Advances in Design Automation, Flexible Assembly Systems*, number DE-Vol. 73, pages 85–95, Minneapolis, Minnesota, USA.

[Bäck, 1996] Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, New York.

[Baker, 1987] Baker, J. E. (1987). Adaptive selection methods for genetic algorithms. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 100–111, MIT, Cambridge, MA. Lawrence Erlbaum Associates, Publishers.

[Beasley et al., 1993a] Beasley, D., Bull, D. R., and Martin, R. R. (1993a). An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69.

[Beasley et al., 1993b] Beasley, D., Bull, D. R., and Martin, R. R. (1993b). An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181.

[Bertsimas and Tsitsiklis, 1997] Bertsimas, D. and Tsitsiklis, J. N. (1997). *Introduction to Linear Optimization.* Athena Scientific, Belmont, Massachusetts.

[Bjorndal et al., 1995] Bjorndal, M., Caprara, A., Cowling, P., Croce, P., Lourence, H., Malucelli, F., Orman, A., Pisinger, D., Rego, C., and Salazar, J. (1995).

Some thoughts on combinatorial optimization. *European Journal of Operational Research*, 83(2):253–270.

[Bobrow, 1989] Bobrow, J. E. (1989). A direct minimization approach for obtaining the distance between convex polyhedra. *International Journal of Robotics Research*, (3):65–76.

[Booker, 1982] Booker, L. B. (1982). *Intelligent Behavior as an Adaptation to the Task Environment.* PhD thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI.

[Boyse, 1979] Boyse, J. W. (1979). Interference detection among solids and surfaces. *Communications of the ACM*, 22(1):3–9.

[Buchal et al., 1989] Buchal, R. O., Cherchas, D. B., Sassani, F., and Duncan, J. P. (1989). Simulated off-line programming of welding robots. *International Journal of Robotics Research*, 8(3):31–43.

[Buckley and Leifer, 1985] Buckley, C. E. and Leifer, L. J. (1985). A proximity metric for continuum path planning. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1096–1102, Los Angeles, California, USA.

[Cameron and Culley, 1986] Cameron, S. A. and Culley, R. K. (1986). Determining the minimum translational distance between two convex polyhedra. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 591–596, San Francisco, California, USA.

[Canny, 1987] Canny, J. (1987). *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, Mass.

[Carretero et al., 2000] Carretero, J. A., Podhorodeski, R. P., Nahon, M. A., and Gosselin, C. M. (2000). Kinematic analysis and optimization of a new three degree-of-freedom spatial parallel manipulator. *Journal of Mechanical Design*, 122(1):17–24.

[Cavicchio, 1970] Cavicchio, D. J. (1970). *Adaptive search using simulated evolution.* PhD thesis, University of Michigan.

[Chau, 1991] Chau, B. C. T. (1991). Task space kinematics monitor program: manifold pairs analysis of benchmark cases. Technical report, Dynacon Report 38-904/0205.

[Chazelle, 1984] Chazelle, B. (1984). Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507.

[Chopard et al., 1995] Chopard, B., Oussaidène, M., Pictet, O., Schirru, R., and Tomassini, M. (1995). Evolutionary algorithms for multimodal optimization in financial applications. In *Proceedings of the SPP-IF Seminar*, pages 139–142.

[CIMNE, 2002] CIMNE (2002). *GiD - The personal pre and postprocessor, http://gid.cimne.upc.es/.* International Center for Numerical Methods in Engineering, Barcelona, Spain.

[Cohen et al., 1995] Cohen, J., Lin, M., Manocha, D., and Ponamgi, K. (1995). I-COLLIDE: An interactive and exact collision detection system for large-scaled environments. In *Proceedings of ACM Int. 3D Graphics Conference*, pages 189–196.

[Craig, 1986] Craig, J. J. (1986). *Introduction to Robotics: mechanics and control.* Addison-Weslay Publishing Co., Reading, Massachusetts, USA.

[Darwin, 1859] Darwin, C. (1859). *The Origin of Species.* (Can be accessed electronically at http://www.literature.org/).

[Davis, 1991] Davis, L. (1991). *The Handbook of Genetic Algorithms.* Van Nostrand Reingold, New York.

[Davis and Steenstrup, 1987] Davis, L. and Steenstrup, M. (1987). Genetic algorithms and simulated annealing: An overview. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing*, Research Notes in Artificial Intelligence, pages 1–11, London. Pitman Publishing.

[De Jong, 1975] De Jong, K. (1975). *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor.

[De Jong and Sarma, 1992] De Jong, K. A. and Sarma, J. (1992). Generation gaps revisted. In *FOGA-92, Foundations of Genetic Algorithms*, Vail, Colorado.

[Deb and Goldberg, 1989] Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In Schaffer, J. D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50, George Mason University, San Mateo, CA. Morgan Kaufmann.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271.

[Doetsch and Middleton, 1987] Doetsch, K. H. and Middleton, J. A. (1987). Canada's space station program. *Canadian Aeronautics and Space Journal*, 33(4):61–89.

[Donald, 1984] Donald, B. R. (1984). *Local and Global Techniques for Motion Planning.* Master thesis, MIT, Cambridge, Mass.

[Dong and Yuan, 1993] Dong, Z. and Yuan, J. (1993). A formulation for collision identification and distance calculation in motion planning using neural netwoks. *International Journal of Advanced Manufacturing Technology*, 8:227–234.

[Dongarra, 2002] Dongarra, J. J. (2002). Performance of various computers using standard linear equations software. Technical Report CS-89-85, Computer Science Department, University of Tennessee.

[Dorigo and Di Caro, 1999] Dorigo, M. and Di Caro, G. (1999). The ant colony optimization meta-heuristic. *Ideas in Optimization*, pages 11–32.

[Edelsbrunner, 1987] Edelsbrunner, H. (1987). *Algorithms in Combinatorial Geometry.* Springer–Verlag, Heidelberg.

[Forrest, 1974] Forrest, A. R. (1974). Computational geometry - achievements and problems. In *Computer Aided Geometric Design, Proceedings of a conference held at the University of Utah*, pages 17–44, Salt Lake City, Utah. Academic Press.

[Gen and Cheng, 1997] Gen, M. and Cheng, R. (1997). *Genetic Algorithms and Engineering Design.* John Wiley and Sons, Inc., New York.

[Gilardi and Sharf, 2002] Gilardi, G. and Sharf, I. (2002). Literature survey of contact dynamics modelling. *Accepted for publication in Mechanism and Machine Theory.*

[Gilbert and Foo, 1990] Gilbert, E. G. and Foo, C.-P. (1990). Computing the distance between general convex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 6(1):53–61.

[Gilbert et al., 1988] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203.

[Gill and Zomaya, 1998] Gill, M. A. C. and Zomaya, A. Y. (1998). *Obstacle Avoidance in Multi-Robot Systems: Experiments in Parallel Genetic Algorithms.* Series in Robotics and Intelligent Systems - Vol. 20. World Scientific, Singapore.

[Gill et al., 1981] Gill, P. E., Murray, W., and Wright, M. H. (1981). *Practical Optimization.* Academic Press, London, UK.

[Glover and Greenberg, 1989] Glover, F. and Greenberg, H. (1989). New approaches for heuristic search: a bilateral linkage with artificial intelligence. *European Journal of Operational Research*, 39:119–130.

[Goldberg, 1989a] Goldberg, D. E. (1989a). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley.

[Goldberg, 1989b] Goldberg, D. E. (1989b). Sizing populations for serial and parallel genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79, George Mason University, San Mateo, CA. Morgan Kaufman.

[Goldberg and Richardson, 1987] Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, J. J., editor, *Genetic Algorithms and their Applications (ICGA'87)*, pages 41–49. Lawrence Erlbaum Associates, Publishers.

[Goldfarb and Idnani, 1983] Goldfarb, D. and Idnani, A. (1983). A numerically satable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27(1):1–33.

[Grace, 1992] Grace, A. (1992). *Optimization Toolbox User's Guide.* The MathWorks, Inc.

[Greenberg, 2000] Greenberg, H. (1996–2000). *Mathematical Programming Glossary.* World Wide Web, http://www.cudenver.edu/ hgreenbe/glossary/glossary.html.

[Grefenstette, 1986] Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-16(1):122–128.

[Grunwald, 1999] Grunwald, A. J. (1999). Advanced interactive display formats for terminal area traffic control. Technical report, TECHNION - Israel Institute of Technology, Faculty of Aerospace Engineering, Haifa, Israel.

[Hartigan, 1975] Hartigan, J. A. (1975). *Clustering Algorithms.* John Wiley and Sons, New York, NY.

[Hayward et al., 1991] Hayward, V., Aubry, S., and Foisy, A. (1991). Collision prediction among moving objects. Technical report, McRCIM, McGill University.

[Hoffmann and Hopcroft, 1987] Hoffmann, C. M. and Hopcroft, J. E. (1987). Simulation of physical systems from geometric models. *IEEE Journal of Robotics and Automation*, RA-3(3):194–206.

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

[Hopcroft, 1988] Hopcroft, J. (1988). Electronic prototyping. *IEEE Transactions on Aerospace and Electronic Systems*, 24(2):663–667.

[Houck et al., 1996] Houck, C. R., Joines, J. A., and Kay, M. G. (1996). Utilizing Lamarckian evolution and the Baldwin effect in hybrid genetic algorithms. Technical Report NCSU-IE TR 96-01, North Carolina State University, Department of Industrail Engineering.

[Hubbard, 1996] Hubbard, P. M. (1996). Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3):179–210.

[Jiménez and Torras, 1995] Jiménez, P. and Torras, C. (1995). Collision detection: A geometric approach. pages 68–85.

[Joe, 1994] Joe, B. (1994). Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions. *International Journal for Numerical Methods in Engineering*, 37:693–713.

[Johnson, 1985] Johnson, K. L. (1985). *Contact Mechanics*. Cambridge University Press, Cambridge, U. K.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 4598(220):671–680.

[Liepins and Potter, 1991] Liepins, G. and Potter, W. (1991). A genetic algorithm approach to multiple fault diagnosis. *Handbook of Genetic Algorithms*, pages 237–250.

[Lin, 1993] Lin, M. C. (1993). *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, Berkeley, California.

[Lin and Canny, 1991] Lin, M. C. and Canny, J. F. (1991). A fast algorithm for incremental distance determination. In *Proceedings of the 1991 IEEE Conference on Robotics and Automation*, pages 1008–1014, Sacramento, California, USA.

[Liu and Mayne, 1990] Liu, C. Y. and Mayne, R. W. (1990). Distance calculations in motion planning problems with interference situations. In *Proceedings of the 1990 ASME DETC - 16th Design Automation Conference*, pages 145–152, Chicago, Illinois.

[Ma, 1995] Ma, O. (1995). Contact dynamics modelling for the simulation of the space station manipulators handling payloads. In *Proceedings of the 1995 International Conference on Robotics and Automation*, volume 2, pages 1252–1258, Nagoya, Japan.

[Ma et al., 1997] Ma, O., Buhariwala, K., Roger, N., MacLean, J., and Carr, R. (1997). MDSF - a generic development and simulation facility for flexible, complex robotic systems. *Robotica*, 15:49–62.

[Ma and Nahon, 1992] Ma, O. and Nahon, M. (1992). A general method for computing the distance between two objects using optimization techniques. In *Proceedings of ASME Conference on Advances in Design and Automation*, volume 1, pages 109–117, Phoenix, Arizona, USA.

[Mahfoud, 1992] Mahfoud, S. W. (1992). Crowding and preselection revisited. In Männer, R. and Manderick, B., editors, *Parallel problem solving from nature, 2: Proceedings of the Second Conference on Parallel Problem Solving from Nature*, pages 27–36, Brussels, Belgium. Elsevier Science Pub, Amsterdam.

[Mahfoud, 1995] Mahfoud, S. W. (1995). Population size and genetic drift in fitness sharing. In Whitley, L. D. and Vose, M. D., editors, *Foundations of genetic algorithms 3*, pages 185–224, San Francisco, CA. Morgan Kaufmann.

[Mahfoud and Goldberg, 1995] Mahfoud, S. W. and Goldberg, D. E. (1995). Parallel recombinative simulated annealing: a genetic algorithm. *Parallel Computing*, 21(1):1–28.

[Marhefka and Orin, 1996] Marhefka, D. W. and Orin, D. E. (1996). Simulation of contact using nonlinear damping model. In *Proceedings of the 1996 IEEE Conference on Robotics and Automation*, pages 1662–1668, Minneapolis, Minnesota, USA.

[Martinez-Alfaro et al., 1998] Martinez-Alfaro, H., Valdez, H., and Ortega, J. (1998). Linkage synthesis of a four bar mechanism for n precision points using simulated annealing. In *Proceedings of 1998 ASME DETC Conference*, page 7, Atlanta, Georgia.

[Maruyama, 1972] Maruyama, K. A. (1972). A procedure to determine intersection between polyhedral objects. *International Journal of Computer and Information Sciences*, 1(3):255–266.

[Meyer, 1986] Meyer, W. (1986). Distance between boxes: Applications to collision detection and clipping. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 597–602, San Francisco, California, USA.

[Michalewicz, 1994] Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs.* AI series. Springer-Verlag, New York.

[Michalewicz, 1995] Michalewicz, Z. (1995). A survey of constraint handling techniques in evolutionary algorithms. *Evolutionary Progamming IV*, pages 135–155.

[Miller and Shaw, 1996] Miller, B. L. and Shaw, M. J. (1996). Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 786–791, Nagoya, Japan.

[Mirtich, 1998] Mirtich, B. (1998). V-Clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics (TOG)*, 17(3):177–208.

[Mirza et al., 1993] Mirza, K., Hanes, M., and Orin, D. E. (1993). Dynamics simulation of enveloping power grasps. In *Proceedings of the 1993 IEEE Conference on Robotics and Automation*, pages 430–435.

[Nahon, 1993] Nahon, M. (1993). Determination of the minimum distance between moving objects including velocity information. In *Proceedings of ASME Conference on Advances in Design Automation*, volume 1, pages 693-698.

[Nahon, 1994] Nahon, M. (1994). Determination of the interference distance between two objects using optimization techniques. In *Proceedings of ASME Conference on Advances in Design Automation*, volume 1, pages 1–6, Minneapolis, Minnesota, USA.

[Nahon et al., 1998] Nahon, M., Ma, O., Piedbœuf, J.-C., and DeCarufel, J. (1998). A distance determination algorithm for real-time simulation of contact dynamics. In *Proceedings of the 7th CASI Conference on Astronautics*, volume 1, pages 1–7, Ottawa, ON, Canada.

[Negnevitsky, 2001] Negnevitsky, M. (2001). *Artificial Intelligence: A Guide to Intelligent Systems.* Pearson Education, Harlow, England.

[Okrouhlik, 1994] Okrouhlik, M., editor (1994). *Mechanics of Contact Impact.* Number AMR140. ASME.

[O'Rourke, 1993] O'Rourke, J. (1993). *Computational Geometry in C.* Cambridge University Press, Cambridge.

[Pham and Karaboga, 1997] Pham, D. T. and Karaboga, D. (1997). Genetic algorithms with variable mutation rates: application to fuzzy logic controller design. *Proc. Inst. Mech. Eng. I, J. Syst. Control Eng. (UK)*, 211(I2):157–167.

[Pham and Karaboga, 2000] Pham, D. T. and Karaboga, D. (2000). *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks.* Springer-Verlag, London.

[Ponamgi et al., 1997] Ponamgi, M. K., Manocha, D., and Lin, M. C. (1997). Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51–64.

[Quinlan, 1994] Quinlan, S. (1994). Efficient distance computation between non-convex objects. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3324–3329, San Diego, California. IEEE.

[Renders and Bersini, 1994] Renders, J.-M. and Bersini, H. (1994). Hybridizing genetic algorithms with hill-climbing methods for global optimization: Two possible ways. In *Proceedings of the First IEEE Conference on Evolutionary Computation: IEEE World Congress on Computational Intelligence*, Orlando Florida.

[Riley, 1993] Riley, F. W. (1993). *Engineering Mechanics: Dynamics.* John Wiley and Sons, Inc., New York.

[Rutenbar, 1989] Rutenbar, R. A. (1989). Simulated annealing algorithms: An overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26.

[Sareni and Krähenbühl, 1998] Sareni, B. and Krähenbühl, L. (1998). Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106.

[Sharf and Nahon, 1995] Sharf, I. and Nahon, M. (1995). Interference distance calculation for two objects bounded by quadratic surfaces. In *Proceedings of ASME Design Engineering Technical Conference*, volume 1, pages 634–641.

[Smith and Tate, 1992] Smith, A. and Tate, D. (1992). Genetic optimization using penalty functions. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 499–505, San mateo, CA.

[Sridharan and Keerthi, 2001] Sridharan, K. and Keerthi, S. S. (2001). Computation of a penetration measure between 3d convex polyhedral object for collision detection. *Journal of Robotic Systems*, 18(11):623–631.

[Tanenbaum, 1996] Tanenbaum, A. S. (1996). *Computer Networks.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edition.

[Thomas and Torras, 1994] Thomas, F. and Torras, C. (1994). Interference detection between non-convex polyhedra revisited with a practical aim. In *Proceedings of the*

1994 IEEE International Conference on Robotics and Automation, pages 587–594, San Diego, California.

[Ullah and Kota, 1996] Ullah, I. and Kota, S. (1996). Globally-optimal synthesis of mechanisms for path generation using simulated annealing and Powell's method. In *Proceedings of 1996 ASME DETC Conference*, page 8.

[Vemuri and Cedeño, 1995] Vemuri, V. R. and Cedeño, W. (1995). Multi-niche crowding for multi-modal search. *Practical Handbook of Genetic Algorithms: New Frontiers*, II:5–29.

[Wang et al., 1992] Wang, Y.-T., Kumar, V., and Abel, J. (1992). Dynamics of rigid bodies undergoing multiple frictional contacts. In *Proceedings of the 1992 IEEE Conference on Robotics and Automation*, pages 2764–2769, Nice, France.

[Whitley, 1994] Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85.

[Whitley et al., 1994] Whitley, D., Gordon, V. S., and Mathias, K. (1994). Lamarckian evolution, the Baldwin effect and function optimization. In *Parallel Problem Solving from Nature, PPSN III: International Conference on Evolutionary Computation*, pages 6–15, Jerusalem, Israel. Springer-Verlag.

[Wright, 1991] Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In Rawlins, G., editor, *Foundations of Genetic Algorithms*, pages 205–220, San Mateo, CA. Morgan Kaufmann Publishers.

[Zeid, 1991] Zeid, I. (1991). *CAD/CAM Theory and Practice*. McGraw-Hill, Inc., New York.

Appendix A

Contact dynamics

Several different approaches to model the dynamics of colliding bodies can be found in the literature [Gilardi and Sharf, 2002]. The methods can basically be classified into (a) ones using an impulse and momentum formulation, and (b) ones where the contact forces are used to calculate ensuing accelerations.

The impulse and momentum formulations (such as Newton's model, Poisson's model or the Energy model [Wang et al., 1992]) look at the impact as being an instantaneous event by considering the bodies as being rigid [Riley, 1993]. These methods differ amongst them in the way they model the coefficient of restitution. These methods are mostly used when there exists a single impact point and when the impact forces are of no interest.

Contact force-based models look at the internal forces that occur during impact more closely. These methods require the damping and stiffness coefficients to be specified. The simplest method, the spring-dashpot approach, uses constants for these two values during the entire impact [Mirza et al., 1993]. On the other hand, the Hertz approach [Johnson, 1985] and the Hertz approach with non-linear damping

[Marhefka and Orin, 1996] use different methods to account for non-linearities in the modelling of impact forces.

Alternatively, there are other methods that focus on the stress wave propagation theory (for the analysis of the impact) [Okrouhlik, 1994]. These methods are suitable to analyze the internal stresses of the colliding bodies. These methods are quite limited, since the wave propagation in complicated bodies is a complex problem to solve thus they are only used in relatively simple objects.

The contact model used in MD Robotics' MDSF (Manipulator Development Simulation Facility) is defined as a quasi-static approach [Ma, 1995] and it is based on the Hertz model mentioned earlier. The contact forces are calculated as follows:

$$\mathbf{F}_c = \left(k_c d + \mu_d \dot{d}\right) \mathbf{n} + \mathbf{F}_f \tag{A-1}$$

where d is the interference distance, μ_d is the damping coefficient obtained by experimentation, $\mathbf{n}$ is a vector normal to the contacting surfaces, and k_c is the contact stiffness obtained from the physical properties of both colliding bodies. Finally, $\mathbf{F}_f$ denotes all the friction forces involved during the contact[1]. Some information about the geometry of the contact region is also included in coefficient k_c as follows:

$$k_c = c \frac{E_1 E_2}{E_1 \left(1 - \nu_2^2\right) + E_2 \left(1 - \nu_1^2\right)} a \tag{A-2}$$

where c is a surface loading coefficient, a is the contact radius ($a = \sqrt{A/\pi}$, A being the contacting surface area[2]) and E_i and ν_i correspond to the Young's Modulus and the Poisson's ratio of body i, respectively.

[1] Currently, a bristle model is being used to calculate the contact forces.

[2] In MDR's Contact Dynamics Toolbox A is approximated using a proprietary algorithm based on the information of the size/volume of the involved objects and the amount of penetration between them.

As seen in equation (A-1), in order to calculate the contact force $\mathbf{F}_c$, the interference distance (d) between the two analyzed objects needs to be calculated.

Moreover, once the contact forces are determined, they need to be applied to the objects in question at the contact point(s). To date, the contact point and the contact distance inside MDSF are obtained using the minimum distance algorithm developed by Nahon and Ma which is based on the solution of a constrained optimization problem (see for instance [Ma and Nahon, 1992] [Nahon, 1994]).

Appendix B

Expressing points with respect to a common frame

Even though the Euclidean distance between two points is not frame dependent, it is necessary to express the coordinates of both points with respect to the same reference frame. Most authors use an inertial frame Σ_0 as the common reference frame. This entails the use of two point transformations, one to transform points in body frame 1 (Σ_1) into the inertial frame and a second one that serves a similar purpose for points in body frame 2 (Σ_2). That is,

$$[\mathbf{p}_1]_0 = \mathbf{R}_{01}[\mathbf{p}_1]_1 + [\mathbf{o}_1]_0 \tag{B-1a}$$

$$[\mathbf{p}_2]_0 = \mathbf{R}_{02}[\mathbf{p}_2]_2 + [\mathbf{o}_2]_0 \tag{B-1b}$$

where $\mathbf{R}_{0i}$ is a 3×3 proper orthonormal matrix that describes frame Σ_i with respect to the inertial frame Σ_0, $[\mathbf{p}_i]_j$ is a three dimensional vector representing the Cartesian coordinates of a point in body i expressed in terms of frame Σ_j. Finally, $[\mathbf{o}_i]_0$ represents the Cartesian coordinates of the origin of frame Σ_i in terms of frame Σ_0.

In practice, matrix $\mathbf{R}_{0i}$ and point are assembled into a single matrix $\mathbf{T}_{0i}$ known as the homogeneous transform as follows [Craig, 1986]

$$\mathbf{T}_{0i} = \begin{bmatrix} \mathbf{R}_{0i} & [\mathbf{o}_i]_0 \\ \mathbf{0} & 1 \end{bmatrix} \tag{B-2}$$

Thus, when a point $[\mathbf{p}_i]_i$ needs to be expressed with respect to frame Σ_0, that is $[\mathbf{p}_i]_0$ is to be obtained, the following expression is used

$$\begin{bmatrix} [\mathbf{p}_i]_0 \\ 1 \end{bmatrix} = \mathbf{T}_{0i} \begin{bmatrix} [\mathbf{p}_i]_i \\ 1 \end{bmatrix} \tag{B-3}$$

Where equations (B-1) are equivalent to equation (B-3) for $i = 1, 2$.

Thus, if equations (B-1) or (B-3) were to be used in order to express points $[\mathbf{p}_1]_1$ and $[\mathbf{p}_2]_2$ with respect to a common frame Σ_0, 18 multiplications and 18 additions would be needed for each point pair.

In order to reduce the computational expense of expressing both points with respect to the inertial frame, it is proposed to express both points with respect to the same body fixed frame, namely frame Σ_1 attached to body 1. Thus, only the points expressed in frame 2 (*i.e.*, Σ_2) need to be multiplied by the homogeneous transform describing body frame 2 with respect to body frame 1. That is,

$$\begin{bmatrix} [\mathbf{p}_2]_1 \\ 1 \end{bmatrix} = \mathbf{T}_{12} \begin{bmatrix} [\mathbf{p}_2]_2 \\ 1 \end{bmatrix} = \mathbf{T}_{10}\mathbf{T}_{02} \begin{bmatrix} [\mathbf{p}_2]_2 \\ 1 \end{bmatrix} \tag{B-4}$$

where matrices $\mathbf{T}_{01}$ and $\mathbf{T}_{02}$ are known and $\mathbf{T}_{10}$ is obtained from $\mathbf{T}_{01}$ using the following expression[1]

$$\mathbf{T}_{10} = \begin{bmatrix} \mathbf{R}_{10} & [\mathbf{o}_0]_1 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{01}^T & -\mathbf{R}_{01}^T[\mathbf{o}_1]_0 \\ \mathbf{0} & 1 \end{bmatrix} \tag{B-5}$$

[1] Notice that $\mathbf{R}_0^1 = \mathbf{R}_1^{0^{-1}} = \mathbf{R}_1^{0^T}$ is one of the properties of proper orthonormal matrices.

Each time the minimum distance algorithm is called, the position and orientation of both bodies are fixed, *i.e.*, $\mathbf{T}_{01}$ and $\mathbf{T}_{02}$ are fixed, and matrix $\mathbf{T}_{12}$ would only need to be computed once for the minimum distance algorithm run. Thus, the number of computations required to express point pairs with respect to a common frame is reduced by half to 9 multiplications and 9 additions.

Appendix C

Object Structure

C.1 Continuous approach

Using concepts of Constructive Solid Geometry, the objects are represented using addition and/or subtraction of simpler elements (see Section 2.1.1 for more details). In order to store all the information of each object more compactly and efficiently, the objects are stored in a structure with a basic layout as shown in Figure C.1.

As seen in Figure C.1, different elements constitute the object structure. The different basic features of the object are nested within the object structure using the field *children*. The object at the base of the structure will be referred to as the root object. For instance, an object such as the one shown in Figure 2.1 (p. 31) which is constructed as shown in Figure 2.3 (p. 32), would have object **B** (convex hull of original object **A**) as the root object with two children, *i.e.*, negative objects **D** and $\mathbf{C}_2$. Additionally, since negative object **D** is itself concave, it would have a positive child (object **E**).

In order to check the feasibility of a point, the algorithms described in Section

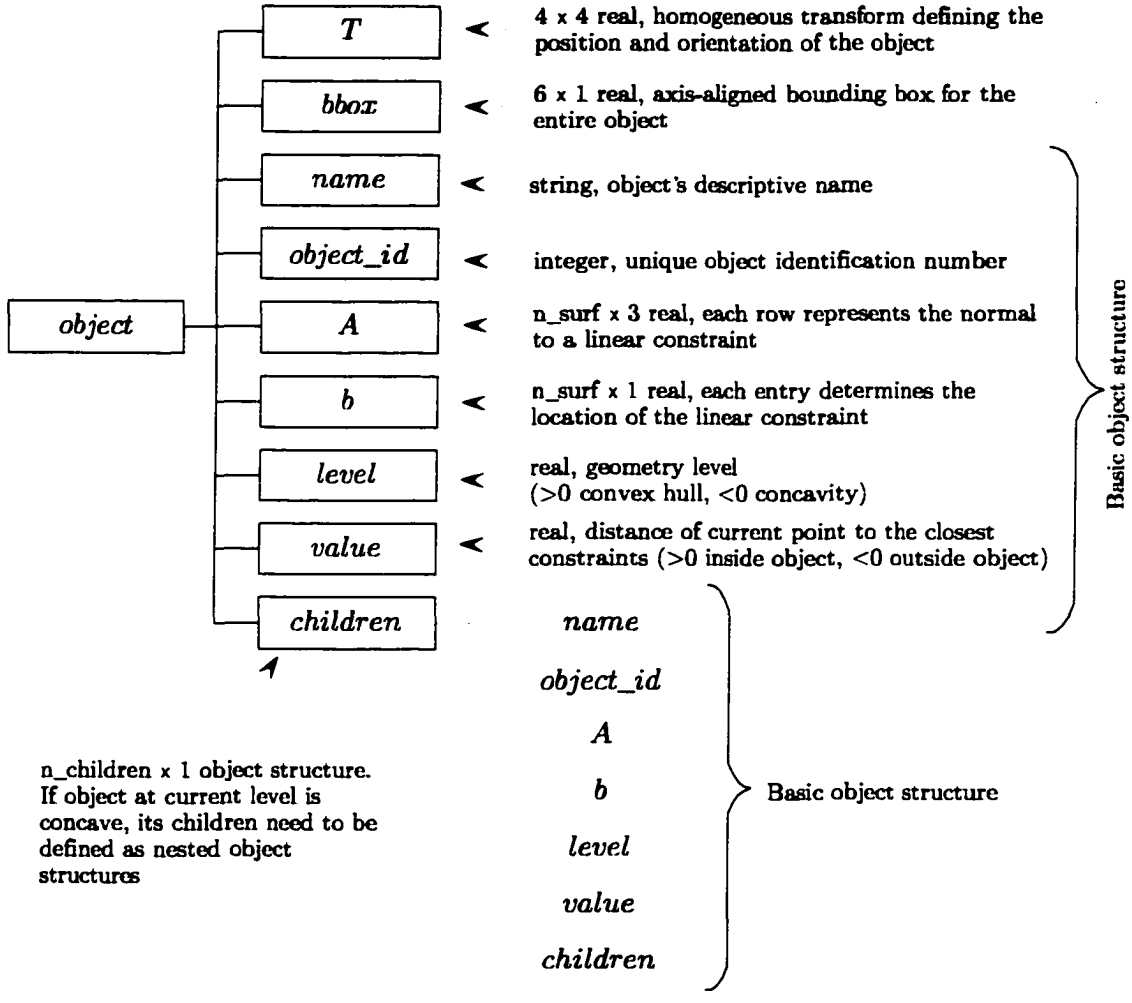


Figure C.1. Basic object structure used to concatenate all of the object's information into a single structure for the continuous approach.

2.1.1 (p. 28) are used. For example, if the root object contains two negative children (field *children* contains a vector of two object structures), the trial points would have to be checked to make sure they are outside both of these children before the point can be said to be feasible. On the other hand, if the field *children* of the root object is empty it means that the object is convex. Thus, a point would only need to satisfy the constraints of the root object in order to be feasible.

The structure field *level* is an integer that determines the level of nesting where the particular object is found. Also, the sign and parity of the object's level determine whether the object is a concavity or not. That is, if an object's level is 3, this means that it is a positive sub-object that belongs to a negative object (in level -2) of the root object (level 1). That is, odd levels are positive and even levels are negative. Although this scheme is redundant, it was chosen since, in a computer implementation, it is less computationally expensive to check the sign of a scalar than its parity. Thus, the fact that the current object is a hole or a solid object can be verified by checking the sign of the object's *level*.

Additionally, in order to track which children belongs to which parent, the structure field *object_id* is used. This field contains a vector of integers that gives the location of the particular sub-piece. Fox example, if at the time of evaluating the constraints of a subobject one obtains that $object\_id = [\ 1\ \ 3\ \ 2\ ]$ it would mean that the object in question is the second child of another object which itself is the third child of the root object. Note that the first element of the vector *object_id* is always equal to 1 meaning that each object has only one single object at its root which is the convex hull of the original object.

Matrix **A** and vector **b** are also amongst the object structure fields and represent the constraints. Presently, the geometry is represented by n_surf linear constraints

with a matrix $\mathbf{A}$ of size $n\_surf \times 3$ and $\mathbf{b}$ of size $n\_surf \times 1$ where $\mathbf{Ap}+\mathbf{b} \geq \mathbf{0}$ needs to be satisfied for $\mathbf{p}$ to be within the bounds of the object. Notice that, although higher order constraints could be used, it was deemed that for the purpose of demonstrating the algorithms' capabilities, the use of linear constraints was sufficient.

To perform the local optimization, the distance to the closest constraint is recorded at the time the point's feasibility is determined as $value = \min(\mathbf{Ap}+\mathbf{b})$. Notice that, if the sub-object's *level* is negative (*i.e.*, it is a concavity), the move closer method has to work as a repair strategy moving the point back into the feasible region. To do so, field *value* is assigned the negative distance from the point to the closest constraint in the subobject, that is $value = -\min(\mathbf{Ap}+\mathbf{b})$. Thus, in general, the numerical value of the field *value* can be determined using the following equation:

$$value = \text{sign}(level) \times \min(\mathbf{Ap}+\mathbf{b}) \tag{C-1}$$

The use of the *value* field in the object's structure allows the algorithm to use the calculated value of the constraints for the penalty value in the objective function as well as the displacement value $q_{i_{\min}}$ needed by the *move closer* algorithm. Thus, it increases the computational efficiency of the algorithm since the calculation of the constraints is only performed once.

Finally, two fields are unique to the root object structure: T and *bbox*. Field T contains the 4×4 homogeneous transform that describes the body fixed frame with respect to the inertial frame. The *bbox* field in the object structure contains a 6 dimensional vector representing the two opposite corners of an axis-aligned bounding box. That is, the bounding box for object i is stored as $object(i).bbox = [x_{i_{\min}}\ y_{i_{\min}}\ z_{i_{\min}}\ x_{i_{\max}}\ y_{i_{\max}}\ z_{i_{\max}}]^T$.

Note that, all quantities contained within the object's i fields are expressed in

terms of the body fixed frame Σ_i except for the homogeneous transform matrix T which is expressed with respect to the inertial frame (Σ_o). Additionally, the axis aligned bounding box is aligned with the body frame and not with the inertial frame.

C.2 Combinatorial approach

The object structure used in the combinatorial approach is illustrated in Figure C.2. The base object in this case, contains three fields, *name*, T and *mesh*. The field *name* is used to store a string value that describes the object whereas field T is used to store the homogeneous transform that describes the object's coordinate frame with respect to the inertial coordinate frame. Finally, the field *mesh* of the structure *object* is itself a structure that contains all the mesh information and is described as follows.

The field *vertices* of the mesh structure contains the node matrix $\mathbf{N}_i$ of size $v \times 3$ where v is the total number of nodes in a mesh. Each row corresponds to a node on the mesh and the three columns contain the three Cartesian coordinates of the nodes with respect to the body frame Σ_i. That is, row 15 will contain the Cartesian coordinates, with respect to the body fixed frame, of the node labelled as 15 on the mesh.

Next, field *connections* contains a cell array of integer vectors with as many rows as points in the mesh (*i.e.*, v) which is constructed from information extracted from the connectivity matrix Π. The number of columns of each row in the *connections* array is determined by the maximum number of connections a node point in the mesh has. For instance, the 10-th row of the *connections* array may contain a vector like [12 9 8 13] which specifies that node 10 is connected to the nodes labelled as 12, 9,

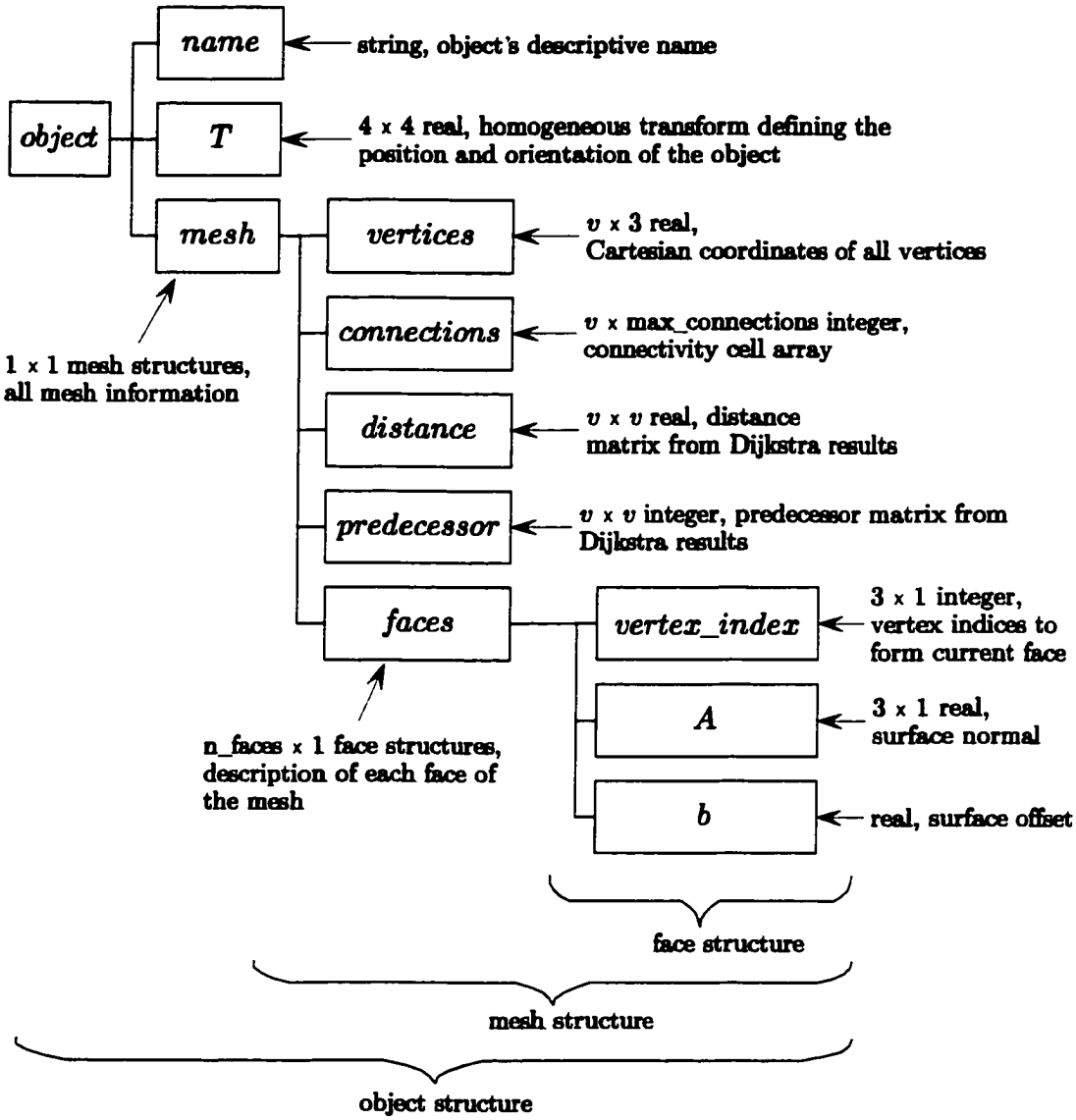


Figure C.2. Basic object structure used to concatenate all of the object's information into a single structure for the combinatorial approach.

8 and 13. This matrix is the one used by the local optimization method in order to obtain all the neighbouring points.

As described in section 3.1.2, the mesh is processed through the Dijkstra algorithm [Dijkstra, 1959] in order to obtain the distance and predecessor matrices. Once obtained, these two matrices are stored in the mesh structure as fields *distance* and *predecessor*, respectively. As discussed in Chapter 3, these two matrices are square of order v (*i.e.*, total number of nodes in the mesh). It should be noted that, matrix *distance* is real, while matrix *predecessor* is an integer matrix.

As shown in Figure C.2, field *faces*, the last field of the mesh structure, is a structure with n_{faces}, where n_{faces} denotes the number of facets in the mesh[1]. The *faces* field contains the information necessary to describe each face of the mesh. That is, each face contains a vector with the indices of the vertices of the face (*i.e.*, the row numbers in the vertex matrix *vertices* that contains the coordinates of the vertices), namely *vertex_index* as well as vector A and scalar b. These last two, together define the equation of the plane that contains the vertices of the face. It is important to note that the information in the *faces* field is not used during the optimization but is used for graphical display purposes.

[1] In a mesh, each face corresponds to the smallest surface element. If three sided mesh elements are selected, then the faces will be triangles whereas in quadrilateral meshes the faces would have four sides. In the context of mesh generation, the word *facets* is often used to refer to the faces.

Appendix D

Finding global minima by enumeration

In order to ensure globality of the solutions obtained with the combinatorial global optimization algorithms, an enumeration process is applied to all mesh points for every given configuration. The enumeration technique obtains the distance between *all* the mesh points on one object and *all* the mesh points of the other object. Although this is feasible even for very large data sets, it is computationally very expensive.

In order to compute the distance between the points on different objects, the point's coordinates need to be expressed with respect to a common frame. In order to reduce the computational expense, one of the body's frame is used, namely the frame attached to body 1, *i.e.*, Σ_1 (see Appendix B).

Even with this significant reduction in computations, 12 multiplications and 14 additions per point pair are still required to evaluate the square of the Euclidean distance, *i.e.*, $d^2 = ({}^1\mathbf{p}_1 - {}^1\mathbf{p}_2)^T({}^1\mathbf{p}_1 - {}^1\mathbf{p}_2)$ where $\begin{bmatrix} {}^1\mathbf{p}_2 \\ 1 \end{bmatrix} = \mathbf{T}_2^1 \begin{bmatrix} {}^2\mathbf{p}_2 \\ 1 \end{bmatrix}$

Table D.1 shows the computational expense required to obtain the minimum distance between two meshes for the ORU Battery and fixture problem. The ORU battery and fixture are illustrated in Figure D.1 where the mesh density can be appreciated.

Mesh type	Nodes body 1 / body 2	Total Combinations	Mult. / Additions (millions)	Mflops
Surface	1241/1842	2 285 922	27.43 / 32	59.43
Volume	2921/3844	11 228 324	134.74 / 157.2	291.94

Table D.1. Number of flops needed to obtain global minimum by enumeration for the ORU-battery case.

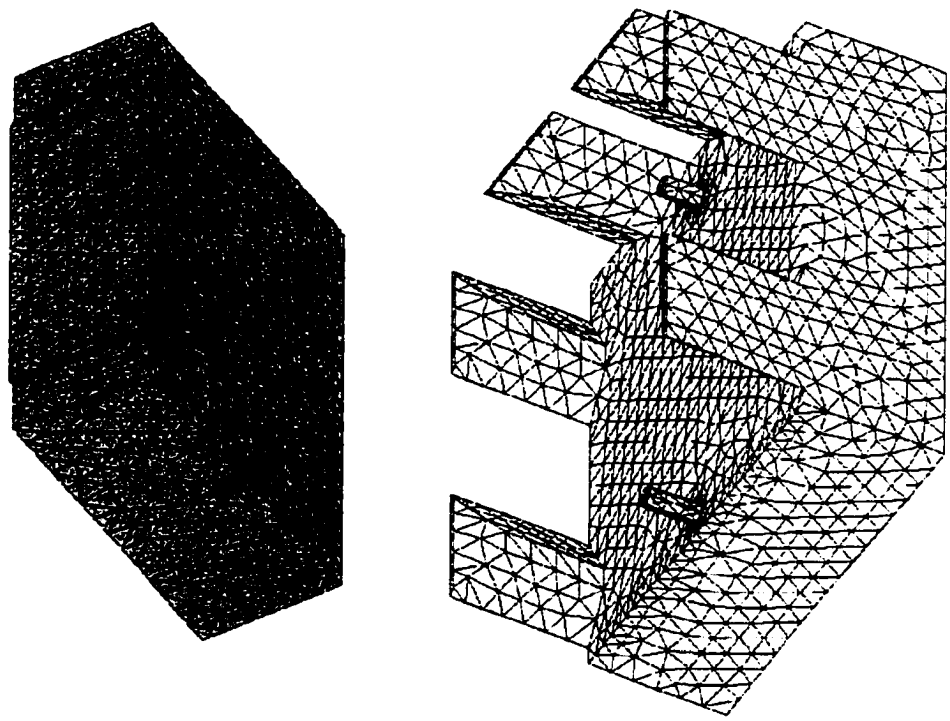


Figure D.1. Mesh representation of the battery and fixture.