Centro de Investigacion y de Estudios
Avanzados
del Instituto Politecnico Nacional

Unidad Zacatenco

Departamento de Computación

Estrategias Meméticas basadas en Técnicas de Búsqueda Local sin Gradiente

Documento realizado por

Sergio Jesús Alvarado García

Como parte de los requerimientos para la obtención del grado de

Doctor en Ciencias en Computación

Supervisor: Dr. Oliver Schütze

Ciudad de México **Junio 2017**

Centro de Investigacion y de Estudios
Avanzados
del Instituto Politecnico Nacional

Unidad Zacatenco

Computer Science Department

Memetic Strategies based on Gradient-free Local Search Techniques

A dissertation submitted by

Alvarado García Sergio Jesús

As the fulfillment of the requirement for the degree of

Doctor in Computer Science

Supervisor: Dr. Oliver Schütze

Mexico City **June 2017**

Agradecimientos

Quiero agradecer al CINVESTAV por dejarme ser parte de su comunidad, así como por todos los apoyos he recibido para la finalización de mi doctorado.

A su vez, también agradezco al CONACyT por el apoyo en forma de beca para la realización de este trabajo.

Agradezco a mi asesor el Dr. Oliver Schütze por toda la guía y el esfuerzo que me brindó durante todos estos años.

Finalmente, quiero agradecer a mis revisores: el Dr. Carlos A. Coello Coello, el Dr. Luis Gerardo De la Fraga, el Dr. Amilcar Meneses Viveros y el Dr. Saúl Zapotecas Martínez por tomarse el tiempo y la consideración de revisar este trabajo de tesis.

Resumen

En el mundo real existen problemas donde uno o más objetivos deben ser optimizados, y los algoritmos meméticos son una de las técnicas más utilizadas para resolverlos. Dichos algoritmos combinan un algoritmo evolutivo con técnicas de búsqueda local. Desafortunadamente, un problema con los algoritmos meméticos, es que en ciertas ocasiones la búsqueda local de los mismos requiere que se proporcione información del gradiente del problema. Como una alternativa para lidiar con esta limitante presentamos en esta tesis tres diferentes técnicas de búsqueda local que no requieren que dicha información sea dada: la Búsqueda Dirigida Discreta (DDS), la Aproximación del Subespacio del Gradiente (GSA) y la Mutación Polinomial de Subespacios (SPM).

El método llamado Búsqueda Dirigida Discreta (DDS), es un método que puede dirigir la búsqueda en cualquier dirección $d \in \mathbb{R}^k$ dada en el espacio de los objetivos. El método DDS surge como una extensión sin gradiente del método de Búsqueda Dirigida (DS), aunque, en lugar de construir una dirección utilizando la Jacobiana de la función, el método DDS usa la información del vecindario. Tal información ya se encuentra calculada en la población actual del algoritmo evolutivo. Utilizando al DDS como método de búsqueda local se propuso un nuevo algoritmo memético, con el cual, se realizaron diversas comparaciones con funciones de prueba del estado del arte. Los resultados obtenidos muestran que el algoritmo memético basado en el método DDS presenta mejores resultados cuando es comparado con el algoritmo evolutivo usado como base; a su vez, presenta resultados similares en comparación con el método DS.

El segundo método presentado en este trabajo es la Aproximación del Subespacio del Gradiente (GSA). Dicho método posee dos diferentes versiones y cada una de ellas depende del número de objetivos del problema que se esté resolviendo. La primera versión del GSA se desarrolló para resolver problemas con un solo objetivo que posean restricciones. En particular, utilizando dicha versión se implementó un

algoritmo memético que utiliza a la Evolución Diferencial (DE, por sus siglas en inglés) y se realizaron experimentos utilizando este algoritmo. Los resultados arrojados por los experimentos muestran que el algoritmo DE/GSA es capaz de ahorrar un monto considerable de recursos (medidos en evaluaciones de las funciones objetivo) en contraparte a los algoritmos con los que se realizó la comparación. La segunda versión del GSA se desarrolló como una extensión del mismo para trabajar en el contexto de problemas multi-objetivo. Dos híbridos se desarrollaron con esta versión: el MOEA/D/GSA y el IG-NSGA-II/GSA. De igual manera, se desarrollaron diversos experimentos, en los cuales, los valores de los indicadores obtenidos muestran una mejora considerable en comparación con el algoritmo base usado para los algoritmos meméticos utilizados.

Finalmente, el último método desarrollado fue la Mutación Polinomial de Subespacios (SPM). Dicho método es capaz de manejar problemas multi-objetivo con restricciones de desigualdad. En particular, la idea principal del operador SPM es el moverse a lo largo de las restricciones activas del problema para así generar nuevas soluciones a lo largo de las mismas. Los experimentos realizados con el operador SPM, muestran que una mejora significativa (medida utilizando indicadores multi-objetivo tales como el hipervolumen y Δ_2) puede ser alcanzada cuando se realiza este tipo de movimiento.

Abstract

In the real world there exist problems where one or more objectives have to be optimized. Memetic algorithms are currently one of the most widely used techniques to solve such problems. Such kind of algorithms hybridize evolutionary algorithms with local search procedures. One drawback of such algorithms is that some of the local search procedures currently available require that explicit gradient information is given. As an alternative to tackle such a problem we present three local search techniques that do not require that explicit gradient information is given: the Discrete Directed Search, the Gradient Subspace Approximation and the Subspace Polynomial Mutation.

The Discrete Directed Search (DDS) is a method that steers the search in any given direction $d \in \mathbb{R}^k$ provided in objective space. The DDS method is a gradient-free realization of the Directed Search (DS) method. Instead of computing a direction using the Jacobian, it utilizes the given information from neighboring solutions of the current population. A new memetic algorithm using the DDS method as a local search was proposed. Such a method was compared using several state-of-the-art test functions. The results of such an experiments demonstrated that the memetic algorithm based on the DDS local search have obtained competitive results in comparison with the standalone algorithm and similar results in comparison to the original DS algorithm.

The second method presented in this work is the Gradient Subspace Approximation (GSA). Two different versions of the GSA are proposed in this work according to the number of objectives of the problem. The first version of the GSA is a method to solve constrained single-objective problems. In particular, to find a solution of a given optimization problem, a memetic algorithm using Differential Evolution was proposed. The experiments of the DE/GSA showed that such an algorithm is capable of saving a considerable amount of resources (measured in function calls) than the

algorithm with respect to which it is compared. A natural extension of the GSA is the increment in the number of objective functions. In particular, a second approach using the GSA method in the multi-objective context was proposed. Two different memetic algorithms are hybridized using the GSA: the MOEA/D/GSA and the IG-NSGA-II/GSA. The indicator values on such an algorithm showed a considerable improvement in comparison with the standalone version of the algorithms.

Finally, the Subspace Polynomial Mutation (SPM) was proposed as a mechanism to handle inequality constrained multi-objective problems. Considering a problem where the Pareto Front is defined by one or more constraints, the SPM operator is capable of computing a new candidate solution steering the search along the active inequality constraints. In particular, the experiments showed that an improvement was archieved in terms of multi-objective indicators (Hypervolume and Δ_2) when such kind of movement is performed.

Contents

List of Figures

List of Tables

List of Algorithms

Contributions

Journal papers

- Oliver Schütze, Adanay Martín, Adriana Lara, Sergio Alvarado, Eduardo Salinas and Carlos A. Coello Coello, The directed search method for multi-objective memetic algorithms, Computational Optimization and Applications, vol. 63, no. 2, pp. 305-332, 2016.

- Oliver Schütze, Sergio Alvarado, Carlos Segura, Ricardo Landa, Gradient subspace approximation: a direct search method for memetic computing, Soft Computing, to appear.

Conference papers

- Adriana Lara, Sergio Alvarado, Saul Salomon, Gideon Avigad, Carlos. A. Coello and Oliver Schütze, The gradient free directed search method as local search within multi-objective evolutionary algorithms, in EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation (EVOLVE II), pp. 153-168, 2013.

- Sergio Alvarado, Adriana Lara, Victor Adrian Sosa Hernandez and Oliver Schütze, An effective mutation operator to deal with multi-objective constrained problems: SPM, in 2016 13th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), pp. 1-6, Sept 2016.

Extended abstracts in international conferences

- Sergio Alvarado, Oliver Schütze, Carlos Segura, Ricardo Landa, DE/GSA: A Memetic Algorithm based on Gradient Subspace Approximation. EVOLVE - A

Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation 2014, Beijing, China.

Extended abstracts in national conferences

- Sergio Alvarado, Oliver Schütze, Carlos Segura, Ricardo Landa, Resolución de problemas de optimización por medio de algoritmos meméticos basados en búsquedas locales sin gradiente. V Congreso de la Sociedad Mexicana de Investigación de Operaciones 2016, Tamaulipas, México.

Chapter 1

Introduction

In real world applications there exist problems where it is necessary to find the optimal parameters of the problem under study to archive some benefits (e.g. to reduce the size of an engine in order to decrease the fuel consumption). Such kind of problems are called optimization problems (OP). Typically, it is not an easy task to find a solution for a given OP and in some cases it becomes necessary to use different techniques to solve it. Some of the widely used techniques to solve optimization problems are the so-called memetic strategies.

Memetic strategies combine evolutionary algorithms with local search strategies to solve optimization problems. Such techniques use the robustness of the evolutionary algorithms and the convergence rate of the local search strategies. One of the drawbacks of memetic strategies, however, is the relatively high cost of their local search procedures (in terms of function calls).

This work presents the implementation of several memetic strategies that can be applied when there exists no complete information about the problem to be optimized. In particular, we are targeting problems where no explicit gradient information is at hand.

Problem

The scope of this work is to develop new memetic strategies to solve optimization problems efficiently. In particular, we consider problems where the gradient information is not at hand such that a gradient-free local search strategy is required. Also,

it is desirable that the new developed techniques can be competitive when they are compared to the state-of-the-art.

Objective

- The main objective is to design memetic strategies for the effective numerical treatment of optimization problems in cases in which the gradient information is not explicitly given.

Particular objectives

- To design gradient-free numerical methods for the efficient treatment of optimization problems.
- To couple the numerical methods with evolutionary strategies.
- To evaluate the new memetic strategies on some of the state-of-the-art problems.

Organization of the work

The organization of this thesis is as follows:

In Chapter 2, some basic concepts to understand the context of this work are presented. This section includes the background for single objective optimization problems and multi-objective optimization problems. The concepts and the state-of-the-art mechanisms are described for each type of optimization problem.

Chapter 3 presents the formulation of the Directed Search method and its gradient free realization, the Discrete Directed Search. Such algorithms are local search strategies that compute a search direction that steers the search by following a given direction $d \in \mathbb{R}^k$ in objective space.

In Chapter 4, the Gradient Subspace Approximation (GSA) method is presented. This method uses neighborhood information to construct descent direction for single and multi objective problems.

Chapter 5 presents a mutation operator for inequality constrained multi-objective problems: the Subspace Polynomial Mutation.

Chapter 6 present the conclusions, the possible improvements and the future work for the techniques that are presented in this work.

Chapter 2

Background

This section presents some of the basic concepts that are required to understand the context of this work. First, the ideas related to single objective optimization are explained. Next, some concepts and techniques used for the treatment of multi-objective optimization are described.

2.1 Single Objective Optimization

Optimization is a process to minimize/maximize an *objective* of a given problem, e.g. to minimize waste produced in a chemical process by reducing the percentage of used water. These kind of problems where a single objective has to be optimized will be referred to as a single objective optimization problem (SOP). In particular, there does not exist a single algorithm that solves all kinds of optimization problems efficiently. In most cases it is left to the user to select a suitable algorithm.

2.1.1 Basis concepts

Objective function

The *objective* is a function that measures the results of the process. In mathematical notation, an objective is defined as follows:

$$f : \mathbb{R}^n \to \mathbb{R}. \tag{2.1}$$

Decision variables

In terms of optimization, a decision variable, which is also called parameter, is a term used to name a quantity that is modified in order to increase/decrease the objective value, e.g. if the weight of a plane is reduced, the fuel consumption decreased.

Formally, the parameters are represented as column vectors constructed with n variables:

$$x = (x_1, x_2, \ldots, x_n)^T .$$

Constraints

Constraints can be considered as the limitations that a problem must fulfill. The constraints force the process to remain in a stable state, e.g., the temperature of an engine must remain under certain threshold such that it does not break. In particular, there exist two types of constraints: equality and inequality constraints.

The inequality constraints force an optimization problem to remain within a given threshold. A solution is considered feasible if its value accomplishes this condition. Formally, inequality constraints are defined as:

$$g_j(x) \leq 0,\ j = 1, \cdots, p. \tag{2.2}$$

Equality constraints can be considered more difficult to accomplish. The complexity of these constraints resides in the fact that a solution is considered feasible only if its value is equal to zero. In mathematical notation, the equality constraints are defined as:

$$h_i(x) = 0,\ i = 1, \cdots, m. \tag{2.3}$$

Gradient vector

A *gradient* is the derivative of a vectorial function. In mathematical terms, the gradient is defined using the ∇ operator. If the function f is differentiable, the gradient is defined as:

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \cdots, \frac{\partial f(x)}{\partial x_n} \right)^T , \tag{2.4}$$

where $\frac{\partial f(x)}{\partial x_i}$ represents the i-th partial derivative of f at x.

2.1.2 Optimality definitions

Using the notation described above we are now in the position to define SOPs. In terms of minimization, a SOP is mathematically defined as:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{s.t. } h_i(x) = 0, i = 1, \ldots, p, \\ &\quad g_j(x) \leq 0, j = 1, \ldots, m. \end{aligned} \tag{2.5}$$

Here, we can define the feasible region Q of Equation (2.5) as follows:

$$Q = \{x \in \mathbb{R}^n : h_i(x) = 0, i = 1, \cdots, m \text{ and } \ g_j(x) \leq 0, j = 1, \cdots, p\}. \tag{2.6}$$

Local and global minimizers

A feasible solution x^* of problem (2.5) is considered a *global minimizer* if:

$$f(x^*) \leq f(x), \forall x \in Q. \tag{2.7}$$

On the contrary, a solution x^* is a *local minimizer* if:

$$f(x^*) \leq f(x), \forall x \in N \cap Q, \tag{2.8}$$

where N is a neighborhood of x^*.

Unconstrained optimal solution

In case that a given SOP does not possess constraints, it is possible to define a local optimal solution of the problem using the gradient and Hessian information.

Theorem 1 ([Fermat, 1629])**.** *Having a function f that is differentiable and continuous at a point x^* where x^* is a local minimal then $\nabla f(x^*) = 0$.*

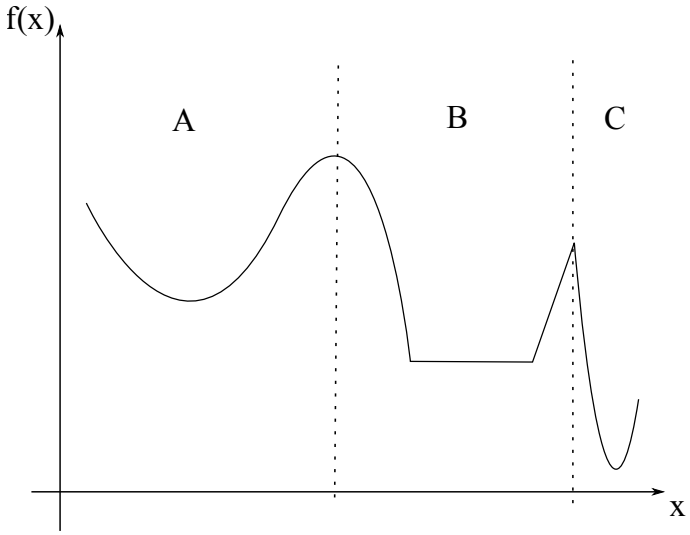


Figure 2.1: Example for local minima in non-smooth problems.

The condition proposed in Theorem 1 states that x^* is a stationary point. A stationary point is defined as a point where $\nabla f(x^*) = 0$. Unfortunately such condition does not guarantee that a stationary point is always a local minimizer of a given SOP. To characterize a minimal solution of an SOP it is necessary to introduce an extra condition.

Theorem 2. *Having a twice differentiable continuous function f and a local minimizer x^* then the Hessian matrix $H(x^*)$ is possitive defined.*

The two conditions proposed above are necessary conditions to demonstrate the optimality of a local solution for smooth problems. A smooth problems is typically defined for functions were the second order information exists and is continuous.

In case that the function is a non-smooth and even discontinuous, the optimality condition can not be applied. However, if the functions consist on smooth pieces separated by discontinuities it is possible to find the minimizer of each smooth piece separately. For example, consider Figure 2.1, where we can obtain the local minima for regions A and C. Unfortunately, we can not obtain such a value from region B.

Karush-Kuhn-Tucker conditions

The conditions proposed by Karush, Khun and Tucker ([Karush, 1939], [Kuhn and Tucker, 1951]) are necessary conditions that determine if a solution x^* of (2.5) is

a local minimizer. The Karush-Kuhn-Tucker conditions (KKT) are described as:

$$\nabla \mathcal{L}(x^*) = \nabla f(x^*) - \sum_{j=1}^{p} \mu_i \nabla g_i(x^*) - \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) = 0 \tag{2.9a}$$

$$g_j(x^*) \leq 0,\ j = 1, \ldots, p \tag{2.9b}$$

$$h_i(x^*) = 0,\ i = 1, \cdots, m$$

$$\mu_i \geq 0, i = 1, \ldots, m \tag{2.9c}$$

$$\lambda_i \geq 0 \tag{2.9d}$$

$$\mu_i g_i(x) = 0, i = 1, \ldots, m, \tag{2.9e}$$

.

In case of an unconstrained SOP, the KKT-conditions of Equation (2.9) are reformulated as:

$$\nabla \mathcal{L}(x^*) = \nabla f(x^*) = 0. \tag{2.10}$$

Descent direction

Consider a vector $\nu \in \mathbb{R}^n$ and a candidate solution x_0. Such vector is called a descent direction if it reduces the value of the objective function.That is, for a given sufficient small step size t it holds that:

$$f(x_0 + t\nu) < f(x_0). \tag{2.11}$$

In case that the function is differentiable. A descent direction can also be defined as follows:

$$\langle \nabla f(x_0), \nu \rangle < 0, \tag{2.12}$$

where $\langle \cdot, \cdot \rangle$ represents the scalar product.

Such claim can be verified using the first order Taylor approximation:

$$f(x + t\nu) = f(x) + t\nu^T \nabla f(x) + O(t^2). \tag{2.13}$$

A direction ν is a descent direction if the angle θ between ν and $\nabla f(x)$ has

$\cos\theta < 0$:

$$\langle \nabla f(x_0), \nu \rangle = \|\nu\| \|\nabla f(x)\| < 0. \tag{2.14}$$

2.1.3 Related Work

Mathematical Programming Techniques

As mentioned before there exist many techniques to find a solution of a given SOP. One large class of these techniques is given by mathematical programming techniques. These techniques use information such as the gradient to find an optimal solution of a problem. In this section, some of the classical mathematical programming techniques are described.

Line search

A line search strategy is an iterative method that aims to find a local optimal value computing a descent direction of the objective function f. Each iteration of this method is performed by a movement along a descent direction. To perform such a movement this method requires a step size $t \in \mathbb{R}^+$. It is important to mention that the choice of the step size is crucial since it is necessary that this step size sufficiently decreases the value of f.

For example, given a descent direction ν at a point x of an unconstrained SOP, the step size with the maximal decay can be obtained by solving the following one-dimensional optimization problem:

$$\min_t f(x + t\nu). \tag{2.15}$$

Obtaining an exact solution of Equation (2.15) can be expensive in terms of computational time. To save some of these resources it is possible to use inexact methods to calculate t, which we discuss in the sequel.

Algorithm 1 presents the pseudocode of a line search technique.

Algorithm 1 Generic line search procedure

Require: Initial solution x_0
Ensure: Set of candidate solutions x_k
1: Set $k := 0$
2: **while** Stopping criteria is not met **do**
3: Find descent direction $\nu_k \in \mathbb{R}^n$
4: Find suitable step size $t_k \in \mathbb{R}^+$
5: Set $x_{k+1} = x_k + t_k \nu_k$
6: $k := k + 1$
7: **end while**

Wolfe conditions

When a line search procedure is used it is necessary to calculate a step size with a 'sufficient' improvement. The existence of the sufficient improvement is necessary in order that the line search procedure converges to the optimal solution x^*. In some cases a 'too small' step size value leads to a difficulty to reach the optimal value. The Wolfe conditions were proposed in order to ensure the convergence to the optimal solution.

Given a constant $c_1 \in (0, 1)$, the Armijo inequality ensures that a line search procedure obtains an improvement based on the step size t and the directional derivate value at direction ν. Having such consideration, the Armijo inequality is defined as:

$$f(x + t\nu) \leq f(x) + c_1 \, t\nu^T \nabla f(x). \tag{2.16}$$

Unfortunately the Armijo inequality is not enough to ensure a 'good improvement' on line search procedures. In this case, all sufficiently small values of t can accomplish Equation (2.16). To avoid that too small step sizes are taken into consideration, a new condition is incorporated into the formulation, the so called curvature condition. Given a value $c_2 \in (c, 1)$, the curvature condition is defined as:

$$\nabla f(x + t\nu)^T \nu \geq c_2 \nabla f(x)^T \nu. \tag{2.17}$$

The curvature condition ensures that the slope increases at the next point of the iterative process. The Armijo inequality together with the curvature condition constitute the so called Wolfe conditions.

Step size backtracking

Assume that the method is applied on a quadratic function. Having a matrix

$A \in \mathbb{R}^{n \times n}$, a vector $b \in \mathbb{R}^n$ and $c \in \mathbb{R}$, a quadratic function is defined as:

$$f(x) = \frac{1}{2} x^T A x - b^T x + c, \tag{2.18}$$

where A is symmetric and positive defined.

Now, assume that for a given initial step size t_0 defined as:

$$t_0 = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_k)^T A \nabla f(x_k)}. \tag{2.19}$$

and a direction ν the required decrease of function value is not accomplished. In such cases, it becomes necessary to modify the value of t_0 in order to reach certain level of improvement. Define a function $\phi : \mathbb{R} \to \mathbb{R}$ such that:

$$\phi(t) = f(x_0 + t\nu), \tag{2.20}$$

where the candidate solution $x_0 \in \mathbb{R}^n$ and direction $\nu \in \mathbb{R}^n$ are given values. Having a value $c \in \mathbb{R}_+$ sufficiently 'small', the condition for the decrease is defined applying the first equation of the Wolfe conditions:

$$\phi(t) \leq \phi(0) + c_1 t \phi'(0). \tag{2.21}$$

A critical mechanism of line search techniques resides in the step size control. Let's consider the example proposed in Figure 2.2. In such example, the initial step size leads to a greater value of ϕ at the starting solution. The main idea of the backtracking is to use the information that has been already calculated to construct an approximation of function ϕ. In the figure, it is presented the second order approximation and, based on that approximation, a new value for t is calculated.

After an iteration of a line search technique is applied with an initial step t_0 such that it does not accomplish Equation (2.21) it is possible to use the information at hand to construct a better approximation of the optimal step size. Let's consider the three pieces of information $\phi(t_0)$, $\phi(0)$ and $\phi'(0)$. Using such information it is possible to construct a quadratic approximation of the function ϕ by interpolating the known information:

$$\phi_1(t) = \left(\frac{\phi(t_0) - \phi(0) - t_0 \phi'(0)}{t_0^2} \right) t^2 + \phi'(0) t + \phi(0). \tag{2.22}$$

Using Equation (2.22) it is possible to obtain a new value $t_1 \in (0, t_0)$ that accom-

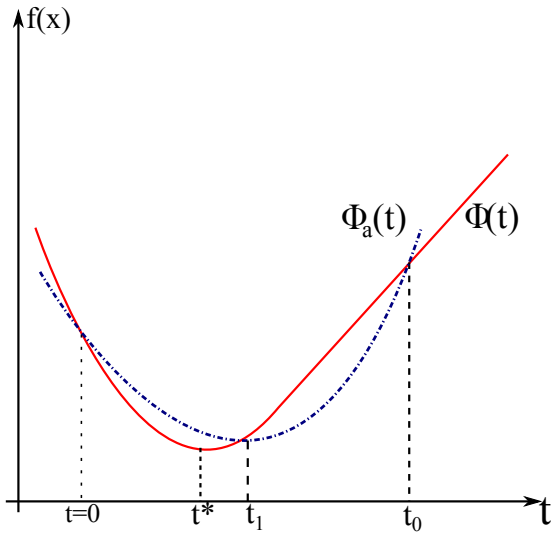


Figure 2.2: Example of backtracking for step size control

plishes the decrease conditions of the problem. In case that such conditions are not met, the known information of the problem (i.e., $\phi(0), \phi'(0), \phi(t_0)$ and $\phi(t_1)$) is used to interpolate a polynomial of higher degree. Such function is defined as follows:

$$\phi_2(t) = at^3 + bt^2 + t\phi'(0) + \phi(0), \tag{2.23}$$

where

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{t_0^2 t_1^2 (t_1 - t_0)} \begin{pmatrix} t_0^2 & -t_1^2 \\ -t_0^3 & t_1^3 \end{pmatrix} \begin{pmatrix} \phi(t_1) - \phi(0) - \phi'(0)t_1 \\ \phi(t_0) - \phi(0) - \phi'(0)t_0 \end{pmatrix}. \tag{2.24}$$

In case that the step size t_2, obtained as a solution of Equation (2.23), does not accomplish the decrease conditions, a new polynomial can be computed incorporating the new information and increasing the degree of the polynomial.

Steepest descent method

The steepest descent method [Debye, 1909] (also known as gradient descent method) is a line search strategy which uses $\nu = -\nabla f(x)$ as its descent direction. This algorithm has the advantage that it does not require second derivative information. Unfortunately, this method presents a poor performance when the complexity of the problem starts to increase.

The use of the negative of the gradient can be motivated from Taylor approxima-

tion given by:

$$f(x+t\nu) = f(x) + t \cdot \nu^T \nabla f(x) + \frac{1}{2} t^2 \cdot \nu^T \nabla^2 f(x+t\nu)\nu + O(\|t\|^2). \tag{2.25}$$

In Equation (2.25), the rate of change of f is given by the linear coefficient of t given by $\nu^T \nabla f(x)$. Thus, the maximal decrease using a normalized direction ν is the solution of the minimization problem:

$$\begin{array}{c} \min\limits_{\nu} \nu^T \nabla f(x) \\ s.t. \, \|\nu\| = 1 \end{array}. \tag{2.26}$$

Since $\nu^T \nabla f(x) = \|\nu\| \, \|\nabla f(x)\| \cos(\theta) = \|\nabla f(x)\| \cos(\theta)$, we know that the maximal decay is given by $\cos(\theta) = -1$. This statement implies that the solution of Equation (2.26) is given by:

$$\nu = -\frac{\nabla f(x)}{\|\nabla f(x)\|}. \tag{2.27}$$

Algorithm 2 presents the steepest descent method for unconstrained SOPs.

Algorithm 2 Pseudocode of steepest descent method for unconstrained SOPs

Require: Initial solution x_0
Ensure: Set of candidate solutions x_k
1: Set $k := 0$
2: **while** $\|\nabla f(x_k)\|_2 \geq tol$ **do**
3: Set $\nu_k := -\nabla f(x_k)$
4: Find suitable step size $t_k \in \mathbb{R}_+$
5: Set $x_{k+1} = x_k + t_k \nu_k$
6: $k := k + 1$
7: **end while**

Conjugate gradient method

The conjugate gradient method (CG) [Shewchuk, 1994, Fletcher and Reeves, 1964] is a method obtained using the observations made on the steepest descent search procedure. The CG principle was defined in order to decrease the convergence rate of the steepest descent method. In principle, the descent direction computed at each iteration on the steepest descent method is orthogonal to the previous one. Because of the computed direction, the convergence rate of the method is decreased. The main

idea behind the CG idea is to move using the maximal step size at a given direction. Let's consider a set of orthogonal directions given by the unit vectors $e_i, i = 1, \ldots, n$. Figure 2.3 presents such idea, starting from the candidate solution x_0. The figure shows the directions given by the steepest descent and the CG method. In the figure, the steepest descent took several steps to reach the optimal solution x^*. Meanwhile, the CG method only requires n steps (in the figure, $n = 2$).

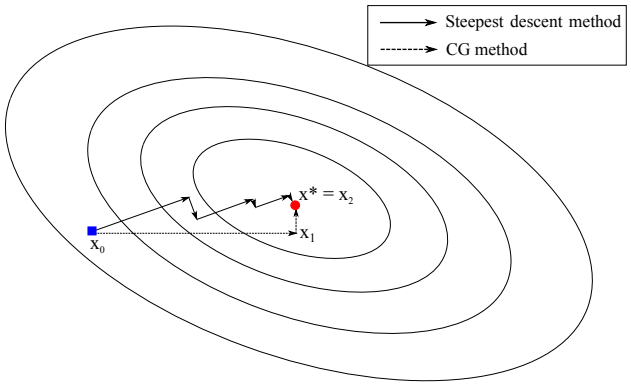


Figure 2.3: Basic idea of CG method.

The CG method is designed to solve quadratic optimization problems of the form of Equation (2.18). The main idea of this method is to replace the direction given by the gradient descent method. For such purpose the CG method typically uses the Gram-Schmidt process to construct the direction.

Direct Search Methods

A direct search method is an optimization technique that does not require information about the gradient of the objective function. Instead, these methods typically construct a descent direction using the information of a set of points in the neighborhood of the current solution. The main advantage of the direct methods is that they do not require that the problem is differentiable in order to work.

Nelder-Mead method

The method proposed in [Nelder and Mead, 1965] is a direct search method that constructs a search direction by using a simplex of $n + 1$ vertices. Given the points $x_i, i = 1, \ldots, n + 1$, the algorithm computes a new candidate solution improving the worst solution of the simplex in each iteration. If the new solution is better than all the solution in the simplex, the algorithm tries to improve even more the candidate

solution. The steps that the Nelder-Mead method uses are the following: sort the solutions and compute the centroid, reflection, expansion, contraction, and shrinkage.

In a fist step, the Nelder-Mead algorithm sorts the solutions according to the function value. After the sorting process, the solutions are arranged such that:

$$f(x_1) \leq f(x_2) \leq \ldots \leq f(x_n) \leq f(x_{n+1}). \tag{2.28}$$

Using the sorted solutions it is possible to compute the centroid d:

$$d = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{2.29}$$

Using the centroid d, the reflection step is performed in order to obtain a new candidate solution x_r. Given a step size $t \in \mathbb{R}_+$, the reflected solution x_r is defined as:

$$x_r = d + t \cdot (d - x_{n+1}). \tag{2.30}$$

In case that $f(x_r) < f(x_1)$, the expansion step is performed. Such step tries to improve x_r using again the centroid d but now with a larger step size. Given a step size $t_e > t$, the expanded solution x_e is calculated as follows:

$$x_e = d + t_e(d - x_{n+1}). \tag{2.31}$$

If the expanded solution is better than x_r, a new simplex is constructed replacing the point x_{n+1} with x_e. In case that $f(x_e) > f(x_r)$, the simplex is constructed by replacing x_{n+1} with x_r.

When the reflected solution is not good enough, i.e. $f(x_r) > f(x_n)$, it is necessary to perform a contraction step. The contraction step improves the reflected solution by performing a search with a smaller step size. Given a step size $t_c \in (0, 0.5 \cdot t]$, the contracted solution x_c is defined as follows:

$$x_c = \begin{cases} d + t_c \cdot (x_r - d) & \text{if } f(x_r) < f(x_{n+1}) \\ d + t_c \cdot (d - x_{n+1}) & \text{otherwise} \end{cases} \tag{2.32}$$

Finally, the shrinkage step is used in case that an improved solution was not found by any of the steps described above. Such step reduces the size of the simplex mod-

ifying all the points. Thus, each solution $x_i, i = 2, \ldots, n+1$, is modified such that $x_i := 0.5 \cdot x_i$.

Pattern Search

The pattern search method proposed in [Hooke and Jeeves, 1961] is a direct search method that predicts a descent direction using the information of the previous steps. This algorithm is based on two kinds of movements: the exploratory movements and the pattern movements.

The exploratory movements are performed in order to obtain information about the neighborhood of the candidate solution x_k. For each component of x_k a search using the canonical vectors is performed. Given a step size t, this search is performed in two different ways. For example, if the canonical vector e_i is used, to compute a new candidate solution x_e, a movement in the positive direction of the canonical vector is performed, i.e. $x_e = x_k + t \cdot e_i$. In case that the function value $f(x_e)$ is greater than $f(x_k)$, a movement using the negative direction of the canonical vector is performed, i.e. $x_e = x_k - t \cdot e_i$. In case that a better point is found, the new candidate solution is stored and the algorithm continues. The search through the canonical vectors is repeated for each $e_i, i = 1, \ldots, n$. If no improvements are found after all the canonical vectors were used, the new step size is set as $t_{\text{new}} = 0.5 \cdot t$ and the search starts again.

Once the exploratory movements are finished, a pattern movement is used to accelerate the process. Such a movement identifies a possible descent direction using the new candidate solution x_e obtained in the previous step. Thus, we can define a pattern search direction as follows:

$$d = x_e - x_k. \tag{2.33}$$

Following the direction d, the algorithm tries to find an improvement for x_e. Using direction d, the pattern candidate solution x_p is computed:

$$x_p = x_e + d. \tag{2.34}$$

The pattern search method has shown to be able to obtain good results when solving SOPs. Unfortunately, one of the drawbacks of this algorithm is the cost (in terms of function calls) of the exploratory movements. In the worst case, the

exploratory movements use $2 \cdot n$ function calls.

Evolutionary Strategies

Nowadays, bio-inspired heuristic algorithms are widely used to solve SOPs. One of these bio-inspired heuristic are Evolutionary Algorithms. Evolutionary Algorithms (EAs) are population-based algorithms that have their foundation in the evolution of species (for more information on EAs, see e.g. [Schwefel, 1993], [Eiben et al., 2003]). There is plenty of evidence that these stochastic methods are a good alternative to solve problems where the gradient methods can not find a global solution. In general, EAs are used to perform a global search in the domain. Unfortunately, one drawback of EAs is that they yield low convergence rates.

EAs start the search with a randomly generated population. Next, the algorithm performs a perturbation on the solutions of the population. Commonly, this perturbation is given by operators such as crossover and mutation. A selection process takes place after the offspring is created. The selection process ensures that only the 'best individuals' survive in each generation. The process continues until some stopping criteria is met. The pseudocode in Algorithm 3 illustrates the main idea of evolutionary algorithms.

Algorithm 3 Generic Evolutionary Algorithm

Require: Fitness function f
Ensure: Final population P_g
1: $g := 1$
2: Randomly initialize the population P_g with N individuals.
3: Evaluate the fitness of each individual in the population using the function f.
4: **while** Stopping criteria is not met **do**
5: Select the subpopulation P'_g from P_g
6: Apply crossover operator to $P'_g \rightarrow P''_g$
7: Apply mutation operator to $P''_g \rightarrow P'''_g$
8: Evaluate the fitness of individuals on population P'''_g
9: Select the best individuals from $P_g \cup P'''_g$ and stored them in P_{g+1}
10: $g := g + 1$
11: **end while**
12: **return** P_g

Memetic Algorithms

Memetic Algorithms (MAs) are strategies that hybridizes a population based method with local search techniques.

The term *memetic* was proposed in [Dawkins, 1989]. Such term is used as the "transmission unit" in the context of cultural evolution. Quoting Dawkins:

"Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."

In terms of computational algorithms the idea of "memes" is used to denote the improvement of particular individuals along with some mechanisms of global cooperation and competition with other solutions in the population.

The main idea of MAs is to perform a local search improving an initial solution by searching within its neighborhood. Once the local search finds the "best" solution in the neighborhood it replaces the original solution. Such process is repeated until the local search can not find a solution in the neighborhood that improves the current solution. Figure 2.4 presents the behavior of a local search technique. In the figure, the solution obtained by the evolutionary algorithm is represented by a circle. The local search is performed by testing several candidate solutions (represented by the triangles) until an improved candidate solution is found (represented by the squares). The search continues until the locally optimal local solution is found or a given budget of function evaluations is spent.

Algorithm 4 presents a generic memetic algorithm. It is important to remark that the local search can be applied in a different position in the algorithm according to the structure of the EA and/or the optimization problem at hand.

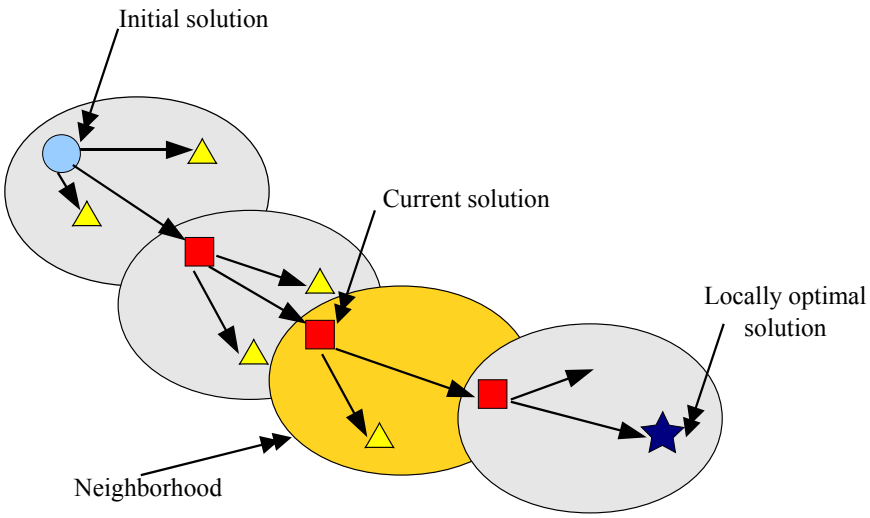


Figure 2.4: Outline of a general local search procedure.

Algorithm 4 Pseudocode of a generic memetic algorithm

Require: Fitness function f
Ensure: Final population P_g

```
g := 1
Randomly initialize the population P_g with N individuals.
Evaluate the fitness of each individual in the population using the function f.
while Stopping criteria is not met do
    Select the subpopulation P'_g from P_g
    Apply crossover operator to P'_g → P''_g
    Apply mutation operator to P''_g → P'''_g
    Local Search in the neighborhood of P''_g (P''_g → P^L_g)
    Evaluate the fitness of individuals on population P^L_g
    Select best individuals from P_g ∪ P^L_g and store them in P_{g+1}
    g = g + 1
end while
return P_g
```

Differential Evolution

Differential Evolution (DE) is a population based algorithm proposed in [Storn and Price, 1997b]. This method proposes a mutation operator that uses individuals in the population to construct a direction. Such a direction is computed using the difference vector of the selected individuals in the population. Given a candidate solution x_i^G in the population at generation G, a new trial vector u_i^{G+1} is computed

as:

$$u_i^{G+1} = x_i^G + F(x_{r1}^G - x_{r2}^G), \tag{2.35}$$

where $r_1, r_2 \in \{1, \ldots, d\}$ are randomly selected indexes in a population of N individuals, which are mutually different, and $F \in [0, 1]$.

DE proposes a crossover operator that uses as parents the candidate solution x_i^G and the trial vector u_i^{G+1}. The new candidate solution x_i^{G+1} is constructed as follows:

$$x_{ij}^{G+1} = \begin{cases} u_{ij}^{G+1} \text{ if } (r_j \leq CR) \text{ or } j = I \\ x_{ij}^G, \text{ otherwise} \end{cases} , j = 1, \ldots, n, \tag{2.36}$$

where $r \in \mathbb{R}^n$ is a vector whose entries are randomly calculated between $[0, 1]$. $I \in \{1, \ldots, d\}$ is a randomly generated integer to ensure that at least one component of the trial vector survives and CR is the crossover probability given by the user.

The operators described above correspond to the $DE/rand/1/bin$ version of the algorithm (for more versions see [Storn and Price, 1997b]). The notation $DE/x/y/z$ is used to classify the different variations of the DE method. Such notation is described as:

- **x.** This value represents that the candidate solution is selected at random.
- **y.** This value represents the number of difference vectors used.
- **z.** This value represents the type of crossover operator adopted.

2.2 Multi-objective Optimization

In the real world it is possible to find problems that require to optimize a certain number of objectives at the same time. Thus, it is possible that the objectives are in conflict. That is, optimizing one objective does not necessarily optimize the other objectives. One of the main difference is that it does exist a single optimal solution in this case. Instead, it is possible to find a set of optimal solutions. For example in the design of a car one is (among many other objectives) interested in a high comfort for the passengers and at the same time in a low production cost of the vehicle. This leads to the bi-objective problem:

$$\begin{pmatrix} \text{maximize} & \text{comfort} \\ \text{minimize} & \text{cost} \end{pmatrix}.$$

2.2.1 Basic concepts

Formal definition of a MOP

Instead of taking a single objective, a multi-objective optimization problem (MOP) has k different objective functions that require to be optimized at the same time. For the unconstrained case, a continuous MOP can be defined as:

$$\min_{x \in \mathbb{R}^n} F(x), \tag{MOP}$$

where F is defined as the vector of objective functions such that :

$$F : \mathbb{R}^n \to \mathbb{R}^k, \tag{2.37}$$

$F(x) = (f_1(x), \ldots, f_k(x))^T$.

In case that we set $k = 2$ on Equation (MOP), we use the term bi-objective problem (BOP) to name such kind of problem.

Pareto dominance

In a MOP the concept of optimal solutions is ambiguous. To compare solutions, a concept of optimality for a MOP was originally proposed in [Edgeworth, 1881] and later generalized in [Pareto, 1896]. This comparison is called *Pareto dominance*, (that is also refered by some authors as *Edgeworth-Pareto Optimality*).

To compare two solutions of a MOP, the dominance relation described by Pareto is defined as follows:

Given two points $x, y \in \mathbb{R}^n$ in decision space, x dominates y ($x \prec y$) if $f_i(x) \leq f_i(y), i = 1, 2, ..., k$ and there exists an index j such that $f_j(x) < f_j(y)$.

In Figure 2.5 we have the images of three points $a, b, c \in \mathbb{R}^n$. The figure shows that the value of $f_2(a)$ and $f_2(c)$ is the same, but the value $f_1(a)$ is less than $f_1(c)$, so we can confirm that $a \prec c$. In the same figure, b dominates c, but a and b do not dominate each other. They are called mutually non-dominated solutions

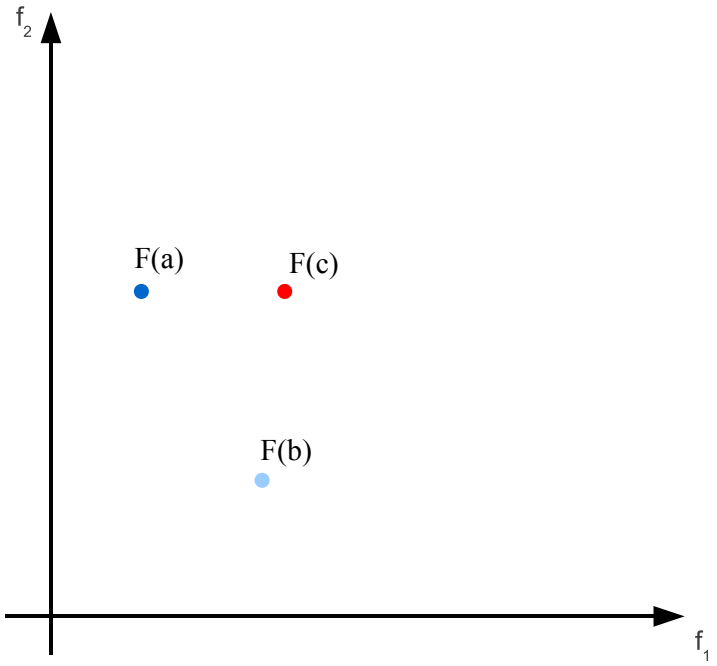


Figure 2.5: Example to demonstrate Pareto dominance

Pareto optimality

More generally, in a MOP it is possible that there exists a set of solutions such that all elements of this set do not dominate each other. Hence, they are the "best" solutions that can be found. From this idea it is possible to present *Pareto optimality*.

Given two points $x, y \in \mathbb{R}^n$, and a vector function F for MOPs a point $x \in \mathbb{R}^n$ is called Pareto optimal if there does not exist a point $y \in \mathbb{R}^n$ that dominates x.

The set of all Pareto optimal solutions of a MOP is called the Pareto set (PS), and can be formalized as:

$$PS = \{x \in \mathbb{R}^n | \nexists y \in \mathbb{R}^n : y \prec x\}. \tag{2.38}$$

Meanwhile, the image of the PS is called Pareto front (PF). Figure 2.6 presents a bi-objective example of the PS and the PF.

Following the definition of Pareto optimality there exists another widely used concept in multi-objective optimization: weakly Pareto optimality. Given a point $x \in \mathbb{R}^n$ and its respective image $F(x) \in \mathbb{R}^k$, we say that a point is weak Pareto

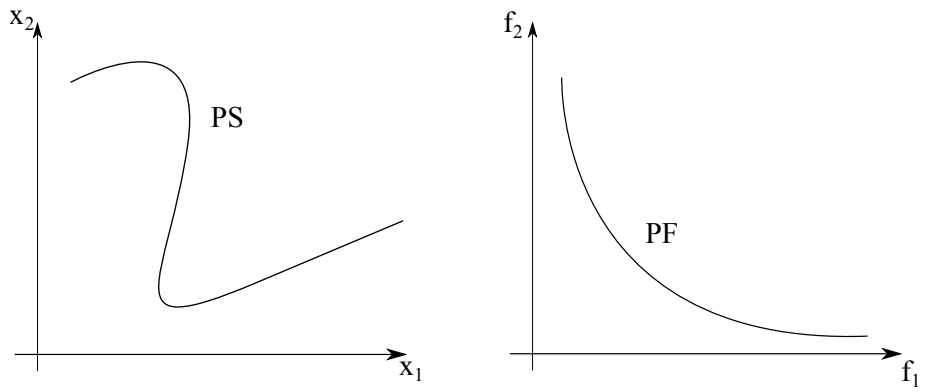


Figure 2.6: Example of a Pareto front and Pareto set

optimal if there do not exist a point $y \in \mathbb{R}^n$ such that:

$$f_i(y) < f_i(x),\ i = 1, \ldots, k. \tag{2.39}$$

Ideal vector

Given a MOP, an ideal vector is a point that dominates the entire Pareto set. Typically it is represented by a point that does not exist inside the solution set. If a set of solution is given, each component of the ideal vector is equal to the minimum objective value found in the entire set. Figure 2.7 presents an example of the ideal vector denoted by Z.

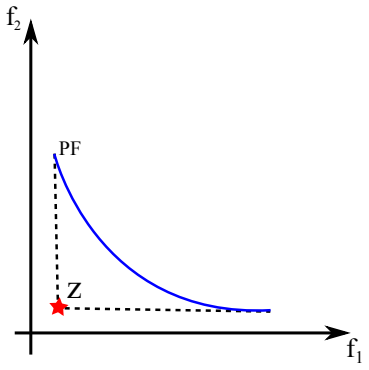


Figure 2.7: Example of ideal vector

Jacobian matrix

The Jacobian matrix denoted as $J(x_0)$ is the matrix formed by the first-order

partial derivatives of (MOP). Such matrix is defined as:

$$J(x) = \begin{pmatrix} \nabla f_1(x)^T \\ :\nabla f_k(x)^T \end{pmatrix} \in \mathbb{R}^{k \times n} \tag{2.40}$$

Descent cones

The concept of descent direction for SOPs is defined using the objective value. Unfortunately, when we have k different objectives such a concept becomes more difficult to explain. Given a candidate solution x_0 there exist three different subspaces for each of the objectives. The first subspace is defined as:

$$f_i^- := \{x : f_i(x) < f(x_0)\}, \tag{2.41}$$

where f_i is the i-th objective of function F. Equation (2.41) defines a subspace, where for any candidate solution, the value of $f_i(x_0)$ decreases. Analogously, we can define a second subspace such that:

$$f_i^+ := \{x : f_i(x) > f(x_0)\}. \tag{2.42}$$

Equation (2.42) defines the subspace where all the solutions increase the objective value in comparison with $f_i(x_0)$. Finally, we define the subspace where the objective value is not changed according to $f_i(x_0)$:

$$f_i^= := \{x : f_i(x) = f(x_0)\}. \tag{2.43}$$

Figure 2.8 presents a hypothetical example of the subspaces described above. To exemplify this, lets consider that the dotted line represents the $f^=$ subspace. The subspace to the right of $f^=$ represents the f_i^- subspace. Finally the subspace f_i^+ is located to the left of $f^=$.

Now, consider the intersection of all the subspaces where all the objectives decrease their value:

$$C^- := \cap_{i=1}^k f_i^-. \tag{2.44}$$

The space defined by D^- is named as descent cone. For any point $x \in D^-$ it holds that:

$$F(x) \prec F(x_0). \tag{2.45}$$

Figure 2.9 presents the main idea behind the descent cones. The hypothetical

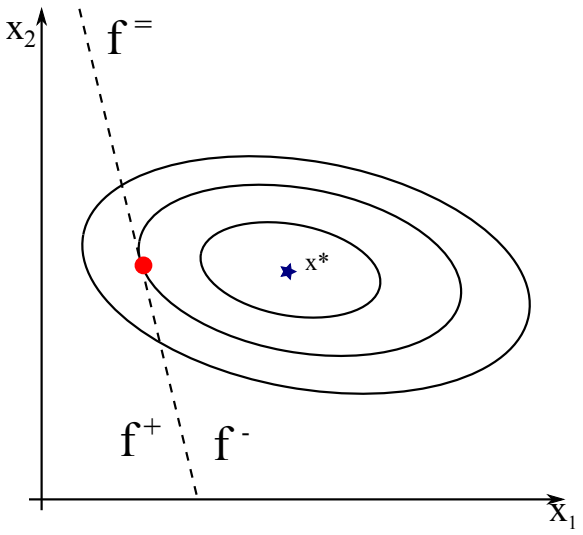


Figure 2.8: Descent half space for SOPs.

example presents a bi-objective function to illustrate this concept. In the figure there exist four possible combinations according to the objectives values. The figure also presents three possible types of region: ascent,descent and diversity.

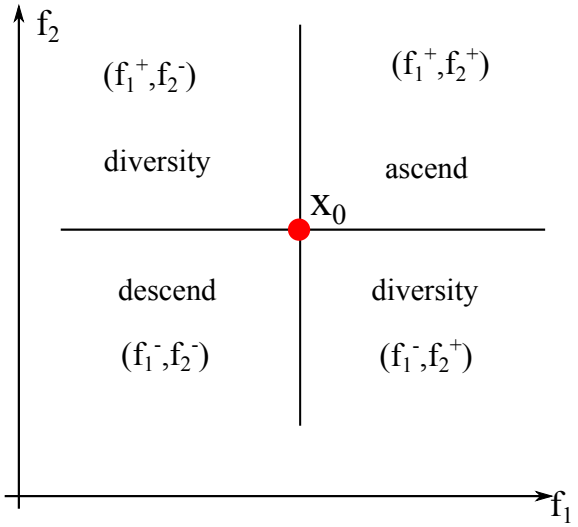


Figure 2.9: Example of cones for MOPs.

Performance indicators

In order to compare two different sets of solutions of a given MOP, we typically use Pareto dominance. Unfortunately, such comparison loses its relevance when the num-

ber of objectives starts to increase. That is, when the number of objectives increases the probability that the given solution sets are mutually non-dominated increases. So, it becomes harder to determine which set is the 'best' between them. In order to avoid such problem, a different approach to compare solution sets is proposed: the use of performance indicators.

A performance indicator transforms a solution set into a scalar value. Hence, the comparison between different set of points is reduced to compare the values obtained by the indicators.

Hypervolume

The hypervolume is a performance indicator proposed in [Zitzler and Thiele, 1999]. This indicator uses a reference point $r \in \mathbb{R}^k$ to compute a scalar value for the entire set. Given a set of points S in objective space, the hypervolume value is computed by the union of the areas of the hypercubes formed by each non-dominated point $s \in S$ with respect to the reference point r. Figure 2.10 presents an example of the hypervolume indicator for $k = 2$.

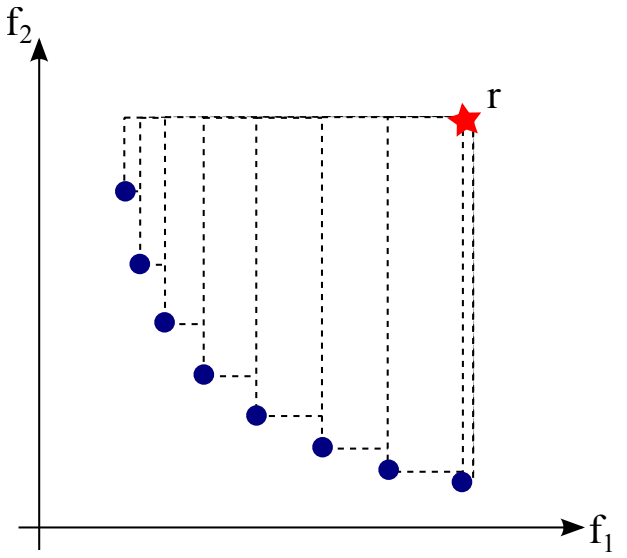


Figure 2.10: Example of hypervolume indicator

Generational Distance and Inverted Generational Distance

The Generational Distance (GD) proposed in [Van Veldhuizen, 1999] is an indicator that measures the distance between a set of points A and a reference set R (e.g., an approximation of the true PF). In mathematical notation, the GD indicator

is computed as:

$$GD(A, R) = \frac{1}{|R|}\left(\sum_{r \in R} \text{dist}(r, A)^p\right)^{\frac{1}{p}}, \tag{2.46}$$

where $\text{dist}(a, B)$ is the distance defined as:

$$\text{dist}(a, B) = \inf_{b \in B} \|a - b\|_p. \tag{2.47}$$

The GD indicator measures the distance from the reference set to the another. Meanwhile, the Inverted Generational Distance (IGD) proposed in [Coello and Cortés, 2005] is computed in a similar way. However, the IGD indicator is measured from the reference set to the set of solutions.

Δ_p *indicator*

The Δ_p indicator proposed in [Schütze et al., 2012] measures the rate of convergence of a solution set along with the distribution of the set of solutions. This method uses a modified version of the GD and IGD indicators to work. Having the solution set A and the reference set R, the modified version of the GD indicator is presented as:

$$GD_p(A, R) = \left(\frac{1}{|R|}\sum_{r \in R} \text{dist}(r, A)^p\right)^{\frac{1}{p}}, \tag{2.48}$$

the modified IGD indicator is defined in a similar way:

$$IGD_p(A, R) = \left(\frac{1}{|A|}\sum_{a \in A} \text{dist}(a, R)^p\right)^{\frac{1}{p}}. \tag{2.49}$$

Using the modified versions of the indicator mentioned above, the Δ_p indicator is defined as:

$$\Delta_p(A, R) = \max(GD_p(A, R), IGD_p(A, R)) \tag{2.50}$$

2.2.2 Related work

In this section we present some of the methods used to solve MOPs.

Scalarization functions

The first approaches to solve MOPs were mostly based on converting the problem into an auxiliary SOP. After the auxiliary SOP is constructed, numerical methods are applied to find the optimal solution. Since the solution of a SOP consists typically of one point, one application of the method will lead ideally to one Pareto optimal solution. An approximation of the entire Pareto set can be obtained by solving a clever sequence of SOPs (see e.g., [Miettinen, 2012]).

Mathematical programming techniques

There exist different mathematical programming techniques used to solve MOPs. The first proposed techniques were based on a decomposition function that transform an MOP into a SOP. But in recent years more specialized techniques have been proposed in order to avoid the need of transforming the problem into an SOP. This section presents some of the mathematical programming techniques used to solve an MOP.

Weighted sum method

One of the oldest algorithms used to solve a MOP is the *weighted sum method*, proposed in [Gass and Saaty, 1955], [Das and Dennis, 1997]. Such method aggregates all objectives f_i into one auxiliary objective F_w by considering a weighted sum of all the f_i's. To be more precise, given weights $w_i, i = 1, ..., k$, $w_i \geq 0$, and $\sum_{i=1}^{k} w_i = 1$, the auxiliary objective is defined as follows:

$$F_w(x) = \sum_{i=1}^{k} w_i f_i(x). \tag{2.51}$$

In general, it is not a trivial task to choose w, because it is not known *a priori* which are the weights that lead us to one or another solution. A drawback of this method is that it does not obtain certain solutions when it is applied on non-convex MOPs.

The ϵ-constraint method

Another widely used method is the ϵ-constraint method ([Haimes et al., 1971]). The main idea of this method is to optimize one objective while the other objectives are transformed into constraints with a permissible error ϵ. Using this idea, a MOP

can be replaced by:

$$\begin{array}{ll} \min & f_i(x) \\ x \in \mathbb{R}^n & \\ f_j(x) \leq \epsilon_j, & j \in \{1, \cdots, k\} \backslash \{i\} \end{array} . \tag{2.52}$$

This method is capable of finding several solutions of a MOP by using different values of ϵ. Also, it is important to mention that this method can be used in non-convex functions. The choice of the values of ϵ_j depends of the kind of problem at hand.

Goal Programming

The Goal Programming (GP) method uses the values proposed by the decision makers in order to scalarize the MOP. To transform the MOP the GP method establishes several 'goals' proposed by the decision maker. Using such goals, an accomplishment level is defined according to the objectives values. In other words, the GP method uses the goals to find the solutions of the MOPs that are bounded by the goals. Given a set of p goals, the GP method introduces a positive deviation $p_i, i = 1, \ldots, k$, and a negative deviation $n_i, i = 1, \ldots, k$, for each of the objective functions. Such deviations measure the proximity of the function to the required goal. Figure 2.11 presents an example where the deviations define the feasible region of the goals.

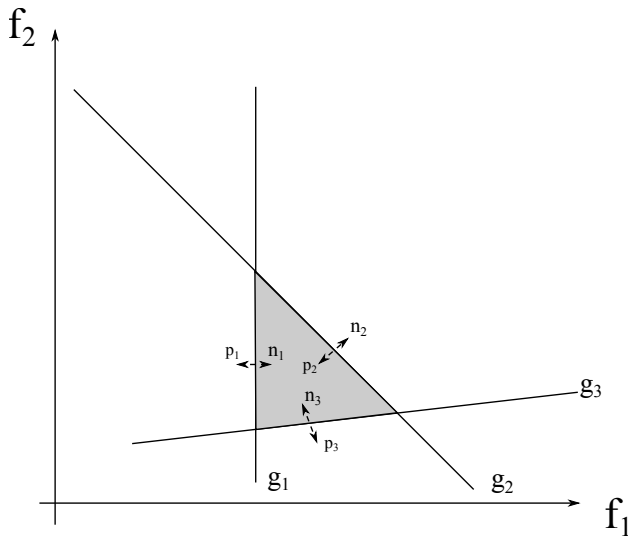


Figure 2.11: Example of region defined by the goals of goal programming method.

Weighted metrics method

This method uses a reference point $z \in \mathbb{R}^k$ to transform a given MOP into an

auxiliary SOP. The main idea behind this method is to minimize the distance between the reference point z and the candidate solution. Besides, as the name indicates, there exists a weight w related to each of the k objectives. The auxiliary function is defined as:

$$M(f^x, w, z) = \left(\sum_{i=1}^{k} w_i |f_i^x - z_i|_p \right)^{\frac{1}{p}}, \tag{2.53}$$

where $f^x = F(x)$.

When the infinity norm is taken (i.e., $p = \infty$), the weighted metric is known as the Tchebycheff scalarization function that is defined as:

$$T(f^x, w, z) = \max_{i=1,\ldots,k} w_i |f_i^x - z_i|, \tag{2.54}$$

Normal Boundary Intersection method

The Normal Boundary Intersection method was proposed in [Das and Dennis, 1998]. The main idea of this method is to construct a Convex Hull of Local Minima (CHIM). The CHIM is the convex hull of the local minima solutions (the local minima solutions are the corners of the convex hull) . NBI method is based in the idea that the intersection between a normal vector emanating from the CHIM and the boundary of F has a high probability to be a Pareto optimal solution. In order to find a solution set instead of a single solution, NBI uses a set of weights to distribute the normal vectors along the CHIM. Unfortunately, even if the weight vectors are well distributed that does not guarantee that the solution set preserves the distribution. Given an ideal vector z^* and a weight vector w, a solution of an MOP can be found by solving the following auxiliary scalar function:

$$\begin{aligned} &\min_{z^* \in \mathbb{R}^k} g^{bi}(x|w, z^*) = d \\ &\text{subject to } z^* - F(x) = dw \end{aligned} \quad . \tag{2.55}$$

Reference point methods

The reference point methods are interactive methods (i.e., methods that require information from the decision maker). In particular, such kind of methods use expected values given by the decision maker that represent the expectation of the de-

cision maker. If the reference points are changed several times and each time the auxiliary scalarization is solved, it is possible to find several parts of the entire Pareto set. Figure 2.12 presents an example of the reference point methods. In the example two different reference points are given $r_1, r_2 \in \mathbb{R}^k$. For each reference point, a single solution of the Pareto front is obtained.

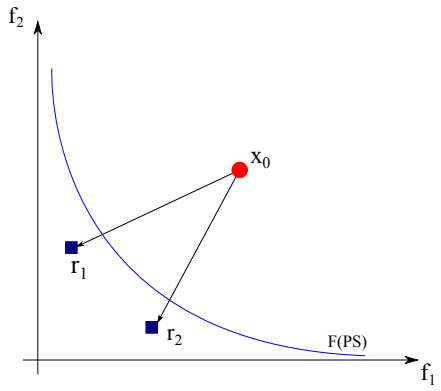


Figure 2.12: Example of the reference method

Achievement Scalarization Function

The Achievement Scalarization Function (ASF) method is intimately related to the weighted metrics described above. This method was introduced in [Wierzbicki, 1980]. ASF introduces the term aspiration levels in order to find a solution of a MOP. Here, the aspiration levels refer to a vector defined by the decision maker that contains certain desired levels for each objective function. An example of the achievement functions is defined as:

$$s(f(x)) = \max(w_i(f_i(x) - z_i)), \tag{2.56}$$

where w_i is a weight associate with the objectives and z is the vector containing the aspiration levels.

One important characteristic of ASF method is that it can generate weakly Pareto solutions. Besides, if there exist a unique solution of the auxiliary function s such solution is Pareto optimal.

Lara direction method

This method proposed in [López et al., 2010] constructs a descent direction for MOPs. The idea behind the computation of this direction is related to the concept

of the descent cones. For example, consider a bi-objective problem and a candidate solution x_0 is 'far away' from the PF, at this point the gradient of the two objectives points almost in the same direction. Using such observation a movement in the average of the two gradients can be a good alternative in order to move towards the PF. Figure 2.13 presents the main idea of the method. In the figure, the half space defined by the normal conformed by the gradient of the objectives conforms almost a half space of the entire space. Furthermore, it is quite possible that the negative normalized sum of the gradients lead the search towards the PF.

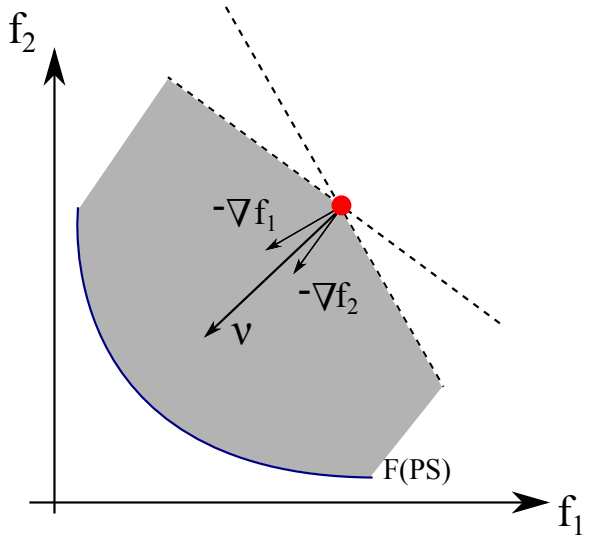


Figure 2.13: Main idea of the Lara direction.

The descent direction of this method is defined follows:

$$\nu = -\left(\frac{\nabla f_1(x)}{\|\nabla f_1(x)\|_2} + \frac{\nabla f_2(x)}{\|\nabla f_2(x)\|_2}\right). \tag{2.57}$$

Directed Search

The Directed Search (DS) is a method proposed in [Schütze et al., 2010]. Such method uses a direction $d \in \mathbb{R}^k$ to compute a direction $\nu \in \mathbb{R}^n$ that steers the search into the given direction. Figure 2.14 illustrates the general idea of the DS method. Figure 2.14.a, presents the image of several solutions that perform a movement in the d-direction. Figure 2.14.b shows the same solutions in parameter space. These solutions are computed using a set of directions $\nu \in \mathbb{R}^n$, which are given by the DS method, to steer the search into direction $d \in \mathbb{R}^k$.

Assuming a candidate solution $x_0 \in \mathbb{R}^n$ where its Jacobian has full rank ($\text{rank}(J(x_0)) =$

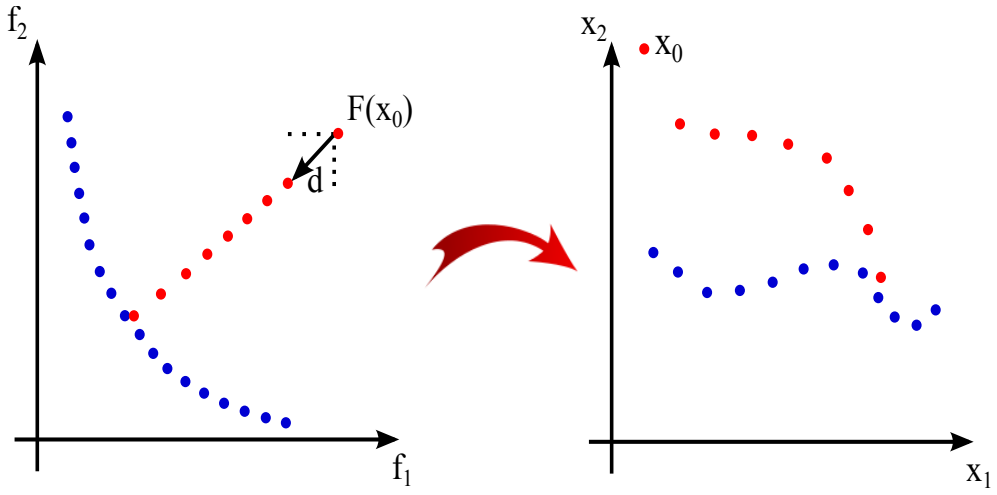


Figure 2.14: Basic idea of DS.

k), and a vector $d \in \mathbb{R}^k$ represents a desired search direction in objective space. A search direction is sought such that for $y_0 := x_0 + t \cdot \nu$, where $t \in \mathbb{R}_+$ (t represents the step size of a line search in direction ν). The direction ν can be found by solving the set of equations given by:

$$\lim_{t \searrow 0} \frac{f_i(y_0) - f_i(x_0)}{t} = \langle \nabla f_i(x_0), \nu \rangle = d_i, \quad i = 1, \ldots, k. \tag{2.58}$$

Using matrix notation, Equation (2.58) can be reformulated as:

$$J(x_0)\nu = d. \tag{2.59}$$

Equation (2.59) shows that the search direction ν can be calculated by solving a linear system of equations. Considering that typically the number of parameters is (much) higher than the number of objectives for a given MOP, i.e., $n \gg k$, the system of Equation (2.59) is (probably highly) underdetermined. This fact implies that its solution is not unique.

Since Equation (2.59) has an infinite number of solutions, we suggest to take the most greedy solution that is given by the following system:

$$\nu_+ := J(x_0)^+ d, \tag{2.60}$$

where $J(x_0)^+ \in \mathbb{R}^{n \times k}$ denotes the pseudo inverse of $J(x_0)$. Notice that if the rank of

the Jacobian is maximal the pseudo inverse is given by:

$$J^{+} = J^{T}(JJ^{T})^{-1}. \tag{2.61}$$

The solution ν_{+} obtained by Equation (2.60), is the one that satisfies Equation (2.59) with the smallest Euclidean norm.

Evolutionary algorithms

Multi-objective evolutionary algorithms (MOEAs) work in principle as the EAs for the treatement of SOPs. The main difference between them is the selection process. In general, the selection process in a MOEA uses Pareto dominance, or other methods that uses a variation of this dominance (see [Deb, 2001], [Coello et al., 2007]).

Also, in recent years, the indicator-based selection techniques have become fashionable. These techniques use an indicator that measures the quality of the entire approximation of the Pareto front. Using this indicator, the comparison between two individuals in the population is made by using an auxiliary scalar value, (see e.g., [Beume et al., 2007], [Schütze et al., 2016a], [Trautmann et al., 2013], [Gómez and Coello, 2013], [Rudolph et al., 2016], [Zitzler and Künzli, 2004]).

Non-dominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II is an algorithm proposed in [Deb et al., 2002a]. It presents two novel ideas to improve the quality of the final solution set.

The *non-dominated sort* is the first mechanism used in the NSGA-II algorithm. It sorts individuals according to Pareto dominance. A rank value is created according to the number of individuals that dominates each solution, the non-dominated solutions receive a rank equals to one, and the others obtain a rank value according to how many subsets dominate them.

To preserve a good spreading of the final solution the NSGA-II uses a mechanism called *crowding distance*. This distance measure the average size of the cuboid formed with the points that enclose a solution i in the population. Algorithm 5 presents the pseudocode to compute the crowding distance of a population. In the algorithm $P_i \rightarrow f^x$ represents the fitness value of the i-th individual in the population.

Algorithm 5 Pseudocode of the crowding distance for NSGA-II

Require: current population P_i
Ensure: vector of distance I

```
l := |P_i|
for i = 1, ..., l do
    Set I_i := 0
end for
for i = 1, ..., k do
    Sort the individuals according to the k-objective
    I_1 := ∞
    I_l := ∞
    for j = 2, ..., (l − 1) do
        f^{x1} = P[j + 1] → f^x
        f^{x2} = P[j − 1] → f^x
        I_j = I_j + (f_i^{x1} − f_i^{x2})
    end for
end for
```

The main idea behind the NSGA-II is to use a selection that preserves the individuals with the lowest rank value. When almost all the solutions exist in the first rank, the algorithm selects the solutions with the highest value of the crowding distance. Algorithm 6 provides the full pseudocode of NSGA-II.

IG-NSGA-II

The IG-NSGA-II proposed in [Liu et al., 2016b] is a memetic algorithm based on the NSGA-II. The IG-NSGA-II uses a local search strategy based on [López et al., 2010] (see Section 2.2.2) to improve its base evolutionary algorithm. Such direction is constructed based on gradient information.

Next to the local search strategy, the IG-NSGA-II implements mechanisms based on chaotic maps created to give a better distribution to the final solution set. The first mechanism proposes to use the chaotic map to give a better distribution to the initial population. The second mechanism proposed in this work computes a solution based on a chaotic map. The main purpose of the chaotic solution is to explore other regions of the given problem. An indicator s_d that measures the distribution of the population is used to trigger the algorithm. When such an indicator is reduced below a certain threshold, then the operator is applied. Algorithm 7 presents the pseudocode of IG-NSGA-II.

Algorithm 6 Pseudocode of the NSGA-II

Require: number of generations G, number of individuals N
Ensure: final population P_G
1: Randomly generate the population P_0
2: Calculates $F(P_0)$
3: Apply genetic operators in P_0 to generate Q_0
4: $i = 0$
5: **while** $i < G$ **do**
6: Set $R_i = P_i \cup Q_i$
7: Calculate the rank value of R_i
8: Calculate the crowding distance of R_i
9: Select the N individuals with the lowest rank and highest crowding distance (P_i).
10: Apply genetic operators in P_i to generate Q_i
11: Set $i = i + 1$
12: **end while**

Algorithm 7 Pseudocode of the IG-NSGA-II

Require: number of generations G, number of individuals N, frequency of local search k_l, threshold for chaotic operator ϵ_d
Ensure: final population P_G
1: Randomly generate the population P_0 using chaotic initialization
2: Calculates $F(P_0)$
3: Apply genetic operators in P_0 to generate Q_0
4: $i = 0$
5: **while** $i < G$ **do**
6: Set $R_i = P_i \cup Q_i$
7: Calculate the rank value of R_i
8: Calculate the crowding distance of R_i
9: Select the N individuals with the lowest rank and higest crowding distance (P_i).
10: Apply genetic operators in P_i to generate Q_i
11: **if** mod(i, k_l) == 0 **then**
12: Apply local search on Q_i
13: **end if**
14: Calculate s_d
15: **if** $s_d < \epsilon_d$ **then**
16: Apply chaotic operator on Q_i
17: **end if**
18: Set $i = i + 1$
19: **end while**

MOEA/D

MOEA/D is a stochastic method introduced in [Zhang and Li, 2006], which is presented as an alternative to solve MOPs through the decomposition of the original problem into N subproblems. The algorithm solves the N subproblems at the same time by evolving a population of solutions. MOEA/D solves each subproblem only using the information allocated in its neighborhood. The pseudocode of MOEA/D is given in Algorithm 8.

The decomposition of the problem can be performed using three different methods: the Tchebycheff decomposition, the PBI method and the weighted sum method.

Algorithm 8 Pseudocode of MOEA/D

Require: Function vector F, number of subproblems N, N weight vector $\lambda_1, \cdots, \lambda_N$, size of the neighborhood T
Ensure: Final population EP

```
Set EP = ∅
Find the T closest weight vectors of each vector.
Set B(i) = i_1, ··· , i_T, with the T closest vectors to λ_i.
Generate a N random vectors x_1, ··· , x_N.
Set FV^i = F(x^i).
Initialize z*
while Stopping criteria is not met do
  for i = 1, ··· , N do
    Select x_k, x_l from B(i)
    Generate y using genetic operators on x_k and x_l.
    Apply an improvement to y and generate y'.
    Update z*
    Update of Neighboring Solutions
    Remove from EP the vectors dominated by F(y').
    Add y' to EP.
  end for
end while
```

Generalized Differential Evolution

This algorithms was proposed in [Kukkonen and Lampinen, 2006]. The main idea behind this work is propose certain modifications to the method proposed in [Storn and Price, 1997a]. In particular, it extends the selection mechanism such that solutions with multiple objective can be compared. In order to work with constrained optimization problems a modification to the original DE algorithms was proposed. The proposed selection scheme takes into consideration the constraint values to decide

which solutions survive. Let's consider an individual x_i^G and its offspring u_i^G created by the DE operators. It is assumed that for these two individuals the values for the function $F(x)$ and the value for its p constraints $g(x)$ are given. Taking into consideration such information, the surviving solution x_i^{G+1} is given by:

$$x_i^{G+1} = \begin{cases} u_i^G \text{ if } \exists k \in \{1,\ldots,p\} : g_k(u_i^G) > 0 \text{ and } \forall k \in \{1,\ldots,p\} : g_k(u_i^G) \leq g_k(x_i^G) \\ u_i^G \text{ if } \forall k \in \{1,\ldots,p\} : g_k(u_i^G) \leq 0 \text{ and } \exists k \in \{1,\ldots,p\} : g_k(x_i^G) > 0 \\ u_i^G \text{ if } \forall k \in \{1,\ldots,p\} : g_k(u_i^G) \leq 0 \text{ and } \forall k \in \{1,\ldots,p\} : g_k(x_i^G) \leq 0 \\ \qquad\qquad \text{and } \forall k \in \{1,\ldots,k\} : f_k(u_i^G) \leq f_k(x_i^G) \\ x_i^G \text{ otherwise} \end{cases} \tag{2.62}$$

Chapter 3

The Discrete Directed Search Method

The Directed Search (DS) method [Lara et al., 2013], [Schütze et al., 2016b] steers a candidate solution using a given vector $d \in \mathbb{R}^k$ in objective space. Unfortunately, one drawback of this method is that it requires explicit gradient information. Such requirement imposes a restriction for the method that can make it not eligible for problems where gradient information is not at hand. One possible solution for this problem is to use one of the several methods that exist in the state-of-the-art to approximate the gradient information. Unfortunately, in some cases such methods are 'very expensive' in terms of function calls.

In this section we propose the Discrete Directed Search (DDS) method. Such method uses neighboring information to approximate the missing gradient information. Besides, to illustrate its advantages we construct a multi-objective memetic algorithm based on it. In particular, we propose a memetic based on the well-known MOEA/D algorithm.

3.1 Main Idea

The DDS method is an extension for the original DS method. The main idea behind the DDS method is to use the neighboring information to compute a direction $\nu \in \mathbb{R}^n$. Such direction steers the search following a given direction $d \in \mathbb{R}^k$ in objective space. The great advantage of the DDS is that its mechanism does not require that the

gradient is explicitly given.

Figure 3.1 presents an example of the information that the DDS algorithm can use. In particular, this figure presents several points in the neighborhood of a given candidate solution $x_0 \in \mathbb{R}^2$. The points are labeled as $x_i, i = 1, \cdots, r$. Besides the x_i points, it is possible to compute other information based on the neighboring information. Using the information of the x_i solutions it is possible to compute the search directions $\nu_i,\ i = 1, \ldots, r$, which has as its starting point the candidate solution x_0. Consider that the function values $F(x_i), i = 1, \ldots, r$ are given. Using such information we can approximate the directional derivatives given as follows:

$$\langle \nabla f_i(x_0), \nu_j \rangle = \frac{f_i(x_j) - f_i(x_0)}{\|x_j - x_0\|_2} + O\left(\|x_j - x_0\|_2\right). \tag{3.1}$$

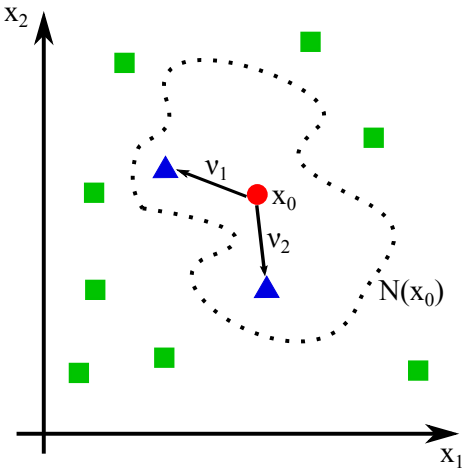


Figure 3.1: Example of neighborhood in a population-based algorithm.

Given an initial solution $x_0 \in \mathbb{R}^n$ and r search directions $\nu_i \in \mathbb{R}^n, i = 1, \cdots, r$, we define the matrix $\mathcal{F}(x_0) \in \mathbb{R}^{r \times n}$ as follows:

$$\mathcal{F}(x_0) = \left(\langle \nabla f_i(x_0), \nu_j \rangle\right)_{\substack{i = 1, \cdots, k \\ j = 1, \cdots, r}}, \tag{3.2}$$

where each entry m_{ij} of the matrix $\mathcal{F}$ is defined by the directional derivative in direction ν_j. Using the information described above we are in position to establish the following proposition:

Proposition 1. *Given $\nu_i \in \mathbb{R}^n, i = 1, \cdots, r$, $\lambda \in \mathbb{R}^r$ and $\nu = \sum_{i=1}^{r} \lambda_i \nu_i$. Then it holds*

that:

$$J(x_0)\nu = \mathcal{F}(x_0)\lambda. \tag{3.3}$$

Proof. The product of $\mathcal{F}(x_0)\lambda$ is defined as follows:

$$\mathcal{F}(x_0)\lambda = \begin{pmatrix} \langle \nabla f_1(x_0), \nu_1 \rangle & \cdots & \langle \nabla f_1(x_0), \nu_r \rangle \\ \vdots & \vdots & \vdots \\ \langle \nabla f_k(x_0), \nu_1 \rangle & \cdots & \langle \nabla f_k(x_0), \nu_r \rangle \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_r \end{pmatrix}. \tag{3.4}$$

Meanwhile, the product $J(x_0)\nu$ is defined as:

$$\begin{aligned} J(x_0)\nu &= J(x_0)\left(\sum_{i=1}^{r} \lambda_i \nu_i\right) \\ &= \sum_{i=1}^{r} \lambda_i \begin{pmatrix} \nabla f_1(x_0)^T \\ \vdots \\ \nabla f_k(x_0)^T \end{pmatrix} \nu_i \end{aligned}. \tag{3.5}$$

Hence for the l-th component of both products it holds:

$$(\mathcal{F}(x_0)\lambda)_I = \sum_{i=1}^{r} \lambda_i \langle \nabla f_i(x_0), \nu_i \rangle = (J(x_0)\nu)_l, \tag{3.6}$$

and the desired identity follows. □

Using Equation (3.2), we can reformulate the linear system proposed in Equation (2.59) as follows:

$$\mathcal{F}(x_0)\lambda = d. \tag{3.7}$$

Now, if we obtain a solution λ obtained from Equation (3.7) we can compute a direction ν as follows:

$$\nu = \sum_{i=1}^{r} \lambda_i \nu_i. \tag{3.8}$$

The next step in our formulation is to demonstrate that Equation (3.7) has a solution. Consider the following identity:

$$\mathcal{F}(x_0) = J(x_0)V, \tag{3.9}$$

where $V = (\nu_1, \ldots, \nu_r) \in \mathbb{R}^{n \times r}$ represents the matrix where each row is given by a search direction ν_i.

Consider that the solution ν_+ of Equation (2.59) has been computed. If one proceeds the search in direction d in the same manner, this is identical to the numerical solution of the following initial value problem:

$$x(0) = x_0 \in \mathbb{R}^n \tag{3.10a}$$

$$\dot{x}(t) = \nu_+ (x(t)), t > 0 \tag{3.10b}$$

.

If d is a 'descent direction' a numerical solution of Equation (3.10) can be viewed as a particular hillclimber for MOPs. Hence, the endpoint x^* of the solution curve of Equation (3.10) does not necessarily have to be a Pareto point, but it is a boundary point in objective space, i.e., $F(x^*) \in \partial F(\mathbb{R}^n)$ which means that the gradients of the objectives in x are linear independent (and hence, that $\text{rank}(J(x)) < k$).

Now, consider that the candidate solution x_0 is not a boundary point. Using such assumption we say that $J(x_0)$ has full rank. Furthermore, using some principles from linear algebra we are in position to propose that:

$$\text{rank}(J(x_0)) = k \Rightarrow \text{rank}(\mathcal{F}(x_0)) = \text{rank}(V). \tag{3.11}$$

Now, consider a case when x_0 is an optimal point. In this case, we know that the rank of the Jacobian matrix is not full, i.e., $\text{rank}(J(x)) < k$. Furthermore, using the rank theorem of matrix multiplication we could state that $rank(\mathcal{F}(x)) < k$ regardless of the choice of V.

The formulations described above indicate that it is possible that the rank of the matrix $\mathcal{F}$ gives us a stopping criteria for the method. Numerically, if the candidate solution x_0 gets closer to its optimal value, the condition number $\kappa_2(\mathcal{F}(x_0))$ will increase. Thus, we can stop the algorithm when the condition number is above a given threshold.

Due to the formulation, it make sense to choose the search directions orthogonal to each other. This is motivated by the fact that $\kappa_2(JV) \leq \kappa_2(J(x))$. Here, if

we consider that $r = n$ and the search directions are orthogonal to each other, a straightforward calculation shows that:

$$V \text{orthogonal} \Rightarrow \kappa_2(F(x)) = \kappa_2(J(x)). \tag{3.12}$$

In that case, the condition number $\kappa_2(F(x))$ can indeed be used as a stopping criteria.

Now we present the formulation to compute a gradient-free realization. For the computation of the missing information, i.e., the matrix $\mathcal{F}$ and the search vectors ν_i, we use information in the neighborhood. Having the candidate solution x_0 and its $x_1, \ldots, x_r$ neighboring points, we can define the search directions $\nu_i, i = 1, \ldots, r$, as follows:

$$\tilde{\nu}_i = \frac{x_i - x_0}{\|x_i - x_0\|_2}, \qquad i = 1, \cdots, r \tag{3.13}$$

Analogously, we can approximate each entry of the matrix $\mathcal{F}(x)$ using the approximation of the directional derivatives of Equation (3.2) :

$$m_{ij} = \langle \nabla f_i(x_0), \nu_j \rangle, \qquad i = 1, \cdots, k, \quad j = 1, \cdots, r. \tag{3.14}$$

One important thing to mention is that Equation (3.2) shows that the error depends only on the Euclidean distance between the neighbors. Hence, if we assume that the size of the neighborhood decreases, the error is not significant.

3.1.1 Influence of the value of r

As described above, the number of neighbors used to construct direction ν directly affects the performance of the algorithm. For example, if a low value for r is used the approximation could lead to a poor performance when a ν direction is computed. From the formulation, it is shown that for an $r > k$ the system of Equation (3.7) can find a descent direction.

The ν direction is constructed is that such a way that it only exists in the subspace formed by $\text{span}\{\nu_1, \ldots, \nu_r\}$. Consider the case where the ν_i directions are orthogonal. If such kind of directions are used to construct the subspace, one could expect that the span of the ν_i directions tends to the $\mathbb{R}^n$ space.

To illustrate the above idea it is possible to rewrite the solution of (3.7) as follows:

$$\nu_r^{(+)} = V(J(x)V)^+ d. \tag{3.15}$$

As the value of r increases the solution presented on Equation (3.15) becomes closer to the solution of the original DS method ν_+. To accomplish such proposition, it becomes necessary that $r = n$ and the ν_i search directions are chosen orthogonal to each other. One way to demonstrate such statement is as follows:

Let $v_1, \cdots, v_n$ be an orthogonal basis of the $\mathbb{R}^n$, where $v_1, \cdots, v_r$ are identical to the column vectors of V. Further, let's define a vector $x \in \mathbb{R}^n$. Since $\{v_1, \cdots, v_r\}$ are a basis of $\mathbb{R}^n$, there exist scalars $\mu_1, \cdots, \mu_n \in \mathbb{R}$ such that $x = \sum_{i=1}^{n} \mu_i v_i$. Then:

$$\begin{aligned} VV^T &= \sum_{i=1}^{r} v_i v_i^T \left(\sum_{j=1}^{n} \mu_i v_i \right) \\ &= \sum_{i=1}^{r} \sum_{j=1}^{n} \mu_j \langle v_i, v_j \rangle = \sum_{i=1}^{r} \mu_i v_i \end{aligned}. \tag{3.16}$$

Here it holds that:

$$\left\| VV^T x - Ix \right\| = \left\| \sum_{i=r+1}^{n} \mu_i v_i \right\|. \tag{3.17}$$

From Equation (3.17) it becomes clear that when $r \to n$ the value of the norm decreases. In particular when $r = n$ it holds that:

$$\begin{aligned} \nu_r^{(+)} &= V\mathcal{F}(x)^+ d = VV^T J(x)^T \left(J(x)VV^T \right)^{-1} d \\ &= J(x)^T \left(J(x)J(x)^T \right)^{-1} d = J(x)^+ d = \nu_+ \end{aligned}. \tag{3.18}$$

Now we present an example of the formulations described above. Consider the bi-objective problem described in [Schütze et al., 2011]:

$$\begin{aligned} F &: \mathbb{R}^2 \to \mathbb{R}^2 \\ f_i(x) &= \|x - a_i\|_2^2, \quad i = 1, 2, \end{aligned} \tag{3.19}$$

where $a_1 = (1, \ldots, 1)^T, a_2 = (-1, \ldots, -1)^T \in \mathbb{R}^n$.

The Pareto set of Equation (3.19) is given by the line segment between a_1 and a_2 that is defined as follows:

$$\mathcal{PS} = \{x \in \mathbb{R}^n \ : \ x_i = 2\alpha - 1, i = 1, \ldots, k, \alpha \in [0, 1]\}. \tag{3.20}$$

Besides, we consider that the value of $r = 2$ is given and the search directions are defined by the canonical vectors $\nu_1 := e_i$ and $\nu_2 := e_j$. Using these considerations we can write the matrix $\mathcal{F}$ as follows:

$$\mathcal{F}(x) = \begin{pmatrix} x_1 - 1 & x_2 - 1 \\ x_1 + 1 & x_2 + 1 \end{pmatrix}. \tag{3.21}$$

Now we try to find the set where the matrix $\mathcal{F}$ has not a unique solution, i.e. matrix $\mathcal{F}$ has no full rank. In other words, we try to find the points where $det(\mathcal{F}) = 0$. The determinant of (3.21) is given by:

$$det(\mathcal{F}) = 2(x_1 - x_2). \tag{3.22}$$

We know that the system does not posses a solution when $det(\mathcal{F}) = 0$. Such condition is accomplished when $x_1 = x_2$. Let's define the set B that contains such points:

$$B := \{x \in \mathbb{R}^n \ : \ x_i = x_j\}. \tag{3.23}$$

Now, let's consider a set that does not contain a boundary point in $\mathbb{R}^n$. On this set of points, the probability is one that for a randomly chosen point $x \in \mathbb{R}^n$ the matrix $\mathcal{F}(x)$ has full rank. Furthermore, Equation (3.2) has a unique solution. To be more precise, it is $\nu = \lambda_1 e_1 + \lambda_2 e_2$, where

$$\begin{aligned} \lambda = \mathcal{F}^{-1}(x)d &= \frac{1}{\det(\mathcal{F}(x))} \begin{pmatrix} x_2 + 1 & -x_2 + 1 \\ -x_1 - 1 & x_2 - 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \\ &= \frac{1}{2(x_1 - x_2)} \begin{pmatrix} x_2(d_1 - d_2) + d_1 + d_2 \\ x_1(d_2 - d_1) - d_1 - d_2 \end{pmatrix}. \end{aligned} \tag{3.24}$$

The above considerations show that already for $r = k$ search directions ν_i, $i =$

$1, \ldots, r$, one can find a descent direction $\tilde{\nu}$ by solving Equation (3.2).

Next, we perform a discussion on the relation between $\nu_+^{(r)}$ and ν_+ for non-boundary points x. In particular, we consider that the search directions ν_i's are orthonormal: it is

$$\nu_+ = J^+(x)d = J(x)^T(J(x)J(x)^T)^{-1}d \tag{3.25}$$

and

$$\begin{aligned} \lambda = \mathcal{F}(x)^+ d &= V^T J(x)^T (J(x)VV^T J(x)^T)^{-1} d \\ &= V^T J(x)^T (J(x)J(x)^T)^{-1} d = V^T \nu_+ \end{aligned} \tag{3.26}$$

and hence

$$\nu_+^{(r)} = \sum_{i=1}^{r} \lambda_i \nu_i = \sum_{i=1}^{r} \langle \nu_i, \nu_+ \rangle \nu_i \tag{3.27}$$

For instance, when choosing $\nu_i = e_{j_i}$, Equation (3.27) gets simplified:

$$\nu_+^{(r)} = \sum_{i=1}^{r} \nu_{+,j_i} e_{j_i}, \tag{3.28}$$

i.e., $\nu_+^{(r)}$ has only r entries which are identical to the corresponding entries of ν_+. In both cases $\nu_+^{(r)}$ gets closer to ν_+ increasing and for $r = n$ it is $\nu_+^{(r)} = \nu_+$.

Finally, we propose an experiment to show the influence of r into the DDS method. Let's consider the problem described in Equation (3.19). In this experiment we use the value of $n = 10$. Using the following starting point:

$$x_0 = \begin{pmatrix} 1 \\ -1 \\ \vdots \\ 1 \\ -1 \end{pmatrix} \in \mathbb{R}^{10}. \tag{3.29}$$

Besides the Jacobian matrix of the candidate solution x_0 is given by

$$J(x_0) = \begin{pmatrix} 0 & -4 & \dots & 0 & -4 \\ 4 & 0 & \dots & 4 & 0 \end{pmatrix} \in \mathbb{R}^{2\times 10} \tag{3.30}$$

If we consider that the search directions are given by the canonical vectors, i.e., $\nu_1 = e_1, \dots, \nu_r = e_r$. Our experiment uses different values of $r \in \{2, 4, 6, 8, 10\}$ and compares the results. In this experiment a direction $d = \frac{1}{\sqrt[2]{2}}(-1, -1)^T$ and a step size t_0 are considered. Figure 3.2 shows the results obtained on this experiment. From the figure one can observe that, as expected, when the value of r increases, the new candidate solution presents a better improvement. In particular, when $r = 10$, the new image of the candidate solution reaches the solution obtained via the DS algorithm.

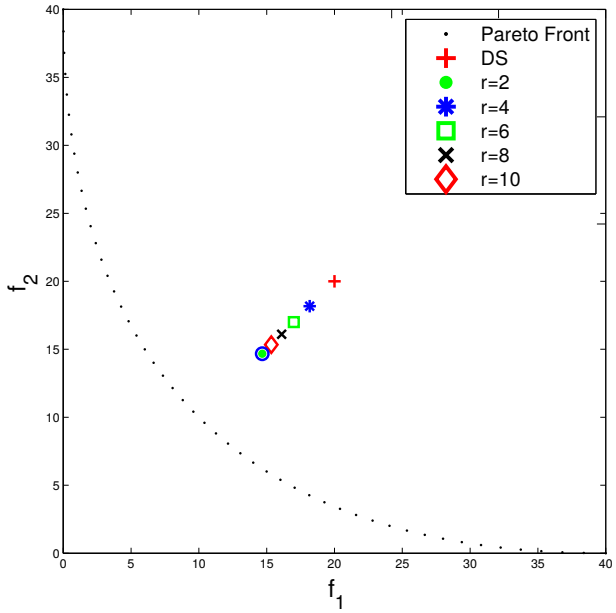


Figure 3.2: Example of the r influence.

3.1.2 Standalone DDS

Using all the formulations above we are in position to define the standalone algorithm for the DDS method. Algorithm 9 presents such realization. The algorithm requires that a candidate solution x_0 along with the $x_i, i = 1, \dots, r$ points are given. Besides, the function values of the given points are also required. That is, $F(x_0)$ and $F(x_i), i = 1, \dots, r$, respectively.

Algorithm 9 Standalone DDS method

Require: Candidate solution x_0, Neighboring points x_i, Function value $F(x_0)$, Neighboring function values $F(x_i)$
Ensure: New candidate solution x_{new}
1: Compute matrix $\mathcal{F}$ as in Equation (3.4)
2: Compute ν solving the Equation (3.7)
3: Compute $t_0 \in \mathbb{R}^+$
4: Set $x_{new} = x_0 + t_0\nu$
5: **return** x_{new}

3.2 Numerical Results

In this section we perform some experiments in order to demonstrate the effectiveness of the DS method. In particular, a comparison between the standalone MOEA/D algorithm [Zhang and Li, 2006] and two different memetic algorithm is performed. The first memetic algorithm uses the DS as its local search procedure. Next, a second memetic coupled with the DDS method is used.

3.2.1 Memetic MOEA/D

As mentioned above, MOEA/D uses a decomposition approach to convert a MOP into a certain number of scalar optimization subproblems. For each scalar subproblem it uses a weight vector λ_i. In particular, such weights can be used as a steering direction for the DDS algorithm. Algorithm 10 describes the coupling of DDS (or DS) and MOEA/D. In particular, we use the code described in [Zhang et al., 2009a]. The notation regarding MOEA/D procedures and parameters is taken from [Zhang and Li, 2006].

We performed a comparison experiment to illustrate the improvement obtained by the memetic algorithm in comparison with its standalone version. In the first set of experiments we used the modified version of the ZDT benchmark presented in [Shukla, 2007]. We use these functions since they are differentiable along their entire domain. Table 3.1 presents the parameters used by the algorithms.

Algorithm 10 Hybrid MOEA/D (MOEA/D/DS and MOEA/D/DDS)

Set the weight vectors λ_i and the neighborhoods $B(i) = \{i_1, \ldots, i_T\}$ for each decomposed problem ($\lambda_{i_1}, \ldots, \lambda_{i_T}$ are the T closest weight vectors to λ_i).
Set up an initial population $P_0 = \{x_1, \ldots, x_N\}$.
Initialize the reference point z, $EP = \emptyset$, $gen = 1$.
repeat
  for $i = 1, \ldots, N$ **do**
    Select two indices k, l from $B(i)$ and generate, using genetic operators, a new solution y_i from x_k and x_l.
    Improving stage: use y_i to replace x_i and its corresponding x_j, $j \in B(i)$, regarding the corresponding scalar problem to λ_i.
    Apply the LS procedure to y_i. (Algorithm 5)
    Update the reference point z.
    Remove from EP all the vectors dominated by y_i, and add it to EP if no vectors in EP dominate y_i.
  end for
  $gen = gen + 1$.
until Stopping criteria is satisfied

return EP.

Algorithm 11 LS Procedure (gen, i)

if $Start_{ls} \leq gen$ **then**
  if $mod(gen, k_{ls}) == 0$ and $mod(i, h_{ls}) == 0$ **then**
    if T_{ls} has not been reached **then**
      Apply, up to D_{ls} times, the LS (DS/DDS) procedure to y_i in order to get y_i'
      Set $y_i \leftarrow y_i'$.
    end if
  end if
end if

return y_i

Table 3.1: Parameters for the experiments on MOEA/D with the ZDT test problems.

Parameter	**Value**
Population size	50
Neighborhood size T	10
Crossover probability	0.95
Mutation probability	$1/n$
Initial step size t_0	1
Frequency of generations k_{ls}	20
Number of neighbors for (D)DS r	7
Number of steps for the (D)DS D_{ls}	3

In this experiment the local search is applied over all the individuals in the population. To avoid that the solution set loses diversity we restrict that local search to be applied until certain limit of function evaluations. For ZDT1-3 the limit is set as 500 function calls. For ZDT4 the limit is extended to $1,000$ function calls. Furthermore, in order to obtain good quality neighbors, DDS is only applied after generation 10.

A statistical study was performed to compare the algorithms. For this study we performed 30 independent runs and measured the average and standard deviation of the results using state-of-the-art indicators. In particular, two indicators were used in the experiments: Δ_2 [Schütze et al., 2012] and the hypervolume indicator [Zitzler et al., 2000a]. Table 3.2 presents the statistical results obtained on these series of experiments. Here, MOEA/D is abbreviated as MD. From these results it is possible to conclude that the memetic version that uses the DDS method as its local search engine obtained a better approximation in comparison with the standalone algorithm. Besides, the memetic algorithm that uses DDS obtained competitive results when it is compared against the MOEA/D/DS. It is important to mention that the latest results can be considered to be a great advantage since DDS does not require gradient information.

A second series of experiments was performed in order to demonstrate the contribution offered by the DDS operator over more complex test problems. In these experiments we measured the performance of the algorithms using the set of test problems of the CEC09 competition suite [Zhang et al., 2008]. Such problems are referred to as UFs. In a similar way that in the ZDT benchmark, the local search is applied after generation 10 and is not applied anymore until $15,000$ function calls had been reached. Again, local search is applied over all the individuals in the population.

Table 3.2: Results on the ZDT benchmark for MOEA/D/(D)DS at 15,000 function calls.

	Δ_2			*Hypervolume*		
Problem	MD	MD/DDS	MD/DS	MD	MD/DDS	MD/DS
ZDT 1	0.52723	**0.47869**	**0.41636**	114.6688	**115.8445**	**116.6319**
(std.dev)	(0.19902)	(0.22428)	(0.22606)	(1.5695)	(1.7952)	(1.8084)
ZDT 2	0.91824	**0.74658**	**0.66601**	110.4825	**112.1839**	**113.0544**
(std.dev)	(0.42883)	(0.4998)	(0.49617)	(3.2805)	(3.9638)	(4.0418)
ZDT 3	0.81688	**0.77392**	**0.71203**	119.552	**120.4113**	**121.2138**
(std.dev)	(0.25095)	(0.25119)	(0.27828)	(2.6732)	(2.7322)	(2.8983)
ZDT 4	7.6429	**7.4658**	**7.5273**	40.2909	**42.1296**	**41.5298**
(std.dev)	(2.813)	(2.8661)	(2.8202)	(30.3759)	(30.8694)	(30.4259)

Table 3.3 presents the parameters used in the experiments. The stopping criteria for the algorithms is set as 50,000 function calls.

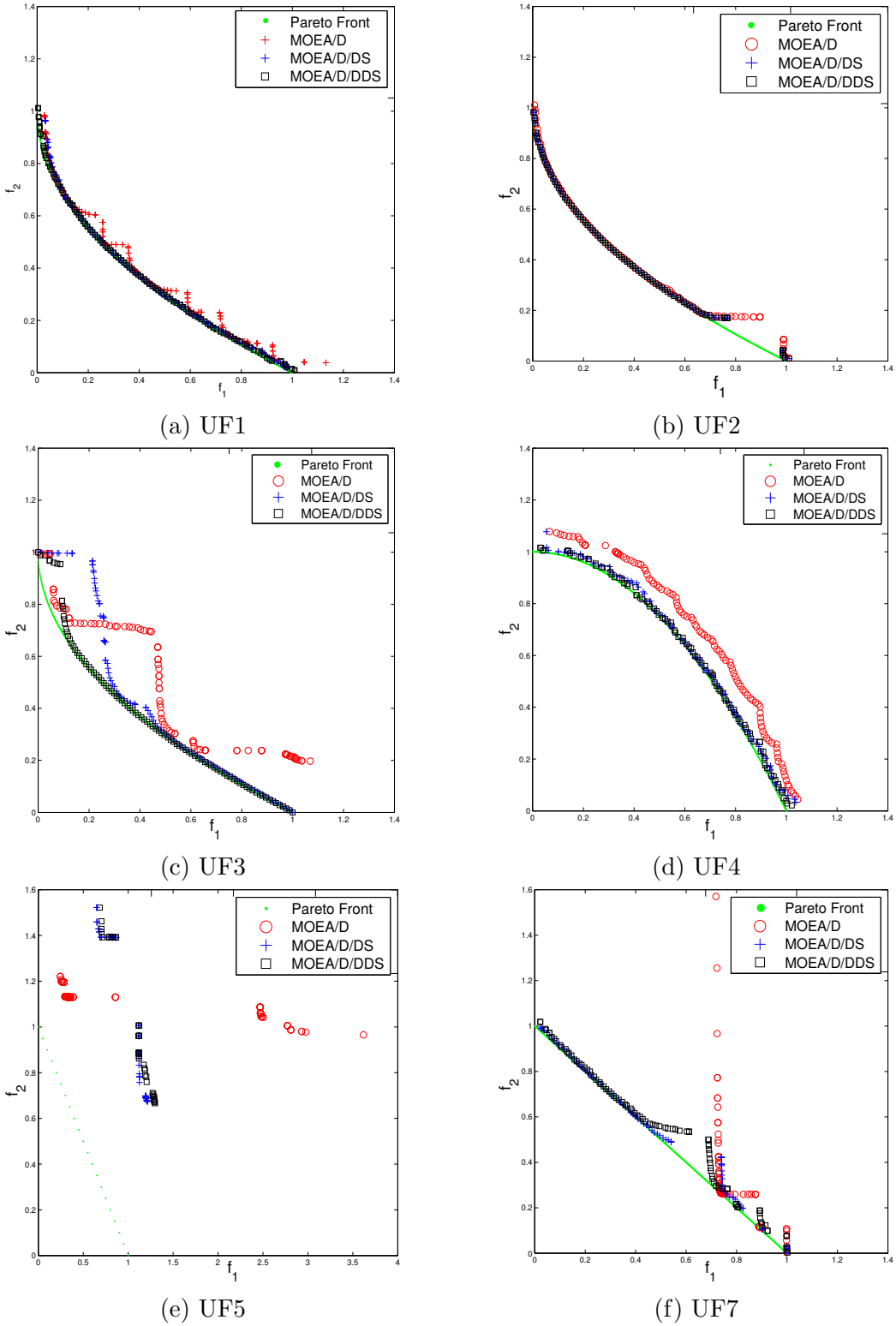
Table 3.3: Parameters for the experiments on MOEA/D with the ZDT test problems.

Parameter	**Value**
Population size	120
Neighborhood size T	60
Crossover probability	0.95
Mutation probability	$1/n$
Initial step size t_0	1
Frequency of generations k_{ls}	10
Number of neighbors for (D)DS r	7
Number of steps for the (D)DS D_{ls}	3

Finally, the most representative results in problems with 2 objectives are presented in Figure 3.3. Results for three objectives are presented in Figure 3.4. In the figures it is possible to observe the runs where MOEA/D/DDS presented a significant improvement with respect to the other algorithms. In case of UF5 and UF10, the three algorithms were unable to reach the Pareto front.

3.2.2 Discussion of results

The results obtained by MOEA/D/DDS show the effectiveness of the DDS method. In particular, the experiments show that DDS is able to improves the convergence



(a) UF1 (b) UF2

(c) UF3 (d) UF4

(e) UF5 (f) UF7

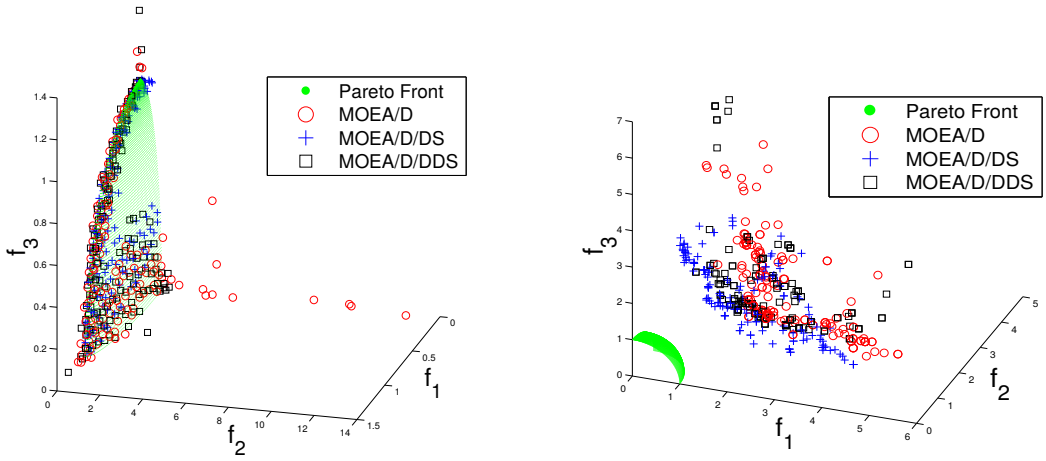
Figure 3.3: Pareto fronts obtained in the UF benchmark with 2 objectives.

Table 3.4: Results on the CEC 09 benchmark for MOEA/D/(D)DS.

	Δ_2			*Hypervolume*		
Problem	MD	MD/DDS	MD/DS	MD	MD/DDS	MD/DS
UF1	0.0355	**0.0167**	**0.0198**	0.9663	**0.9789**	**0.9763**
(std.dev.)	(0.0015)	(0.0008)	(0.0005)	(0.0018)	(0.0013)	(0.0011)
UF2	**0.0277**	0.0280	**0.0243**	0.9722	**0.9731**	**0.9784**
(std.dev.)	(0.00178)	(0.0025)	(0.0015)	(0.0020)	(0.0022)	(0.0015)
UF3	0.0925	**0.0668**	**0.0798**	0.9124	**0.9170**	**0.9192**
(std.dev.)	(0.0034)	(0.0024)	(0.0023)	(0.0055)	(0.0034)	(0.0033)
UF4	0.0878	**0.0822**	**0.0797**	0.9186	**0.9261**	**0.9266**
(std.dev.)	(0.0006)	(0.0004)	(0.0077)	(0.0008)	(0.0006)	(0.0095)
UF5	0.8261	**1.1005**	**0.9493**	0.7672	**0.7047**	**0.7198**
(std.dev.)	(0.0166)	(0.02086)	(0.0197)	(0.0050)	(0.0047)	(0.0053)
UF6	0.2918	**0.2417**	**0.2527**	0.8325	**0.8537**	**0.8505**
(std.dev.)	(0.0158)	(0.0115)	(0.0140)	(0.0073)	(0.0060)	(0.0064)
UF7	0.0258	**0.0148**	**0.0137**	0.9703	**0.9844**	**0.9854**
(std.dev.)	(0.0035)	(0.0005)	(0.0005)	(0.0041)	(0.0063)	(0.0066)
UF8	0.2441	**0.1230**	0.1967	0.9494	**0.96642**	0.9529
(std.dev.)	(0.1326)	(0.0674)	(0.1349)	(0.0185)	(0.0086)	(0.0181)
UF9	0.3385	**0.2652**	0.3535	0.9644	0.9760	**0.9800**
(std.dev.)	(0.1004)	(0.0877)	(0.1337)	(0.0198)	(0.0228)	(0.2072)
UF10	2.5567	**2.3465**	2.9263	0.0543	**0.0807**	0.0537
(std.dev.)	(0.3437)	(0.3277)	(0.2498)	(0.0390)	(0.0666)	(0.0343)

rate of the base algorithm. Moreover, such performance is competitive when it is compare with respect to MOEA/D/DS. Such property, clearly illustrate that the aim of the DDS algorithm has been accomplished. The ν directions computed by the DDS method steer the search in a similar direction that the directions computed by the DS method.

The DDS method obtained better results when it is compared using the proposed indicators. In particular, it is important to mention that it only presents a significant loss in UF2. Unfortunately there exist one drawback for the DDS method. The required number of individuals to compute an approximation depends on the number of parameters. That is, for any given value of n, DDS requires a number of $r \approx 0.4\,n$ individuals. Such restriction limits the method in terms of its applicability. For example, consider a problem with a large number of parameters, i.e., $n > 50$. For such problem, we require approximately 20 individuals to compute matrix $\mathcal{F}$. Using such



(a) UF8 (b) UF10

Figure 3.4: Pareto fronts obtained in the UF benchmark with 3 objectives.

number of individuals it is quite possible that the numerical stability of the method is compromised.

A second point to be considered is how to balance the local search and the evolutionary algorithm. The memetic strategy proposed in this section applies the local search after a fixed number of generations on a fixed number of individuals. This can be described as an initial approach. Unfortunately, such balancing mechanism can be inefficient according to the problem on which it is applied. It is desirable that if we detect that DDS is generating good candidate solutions the probability of being applied increases. But, if DDS does not improve performance we need to reduce the budget of the function calls that the local search can spend.

Finally, we require a better correction method for the step size t_0. That is, we need to find a mechanism that corrects the new candidate solution to the descent cone with the lowest number of function calls. For $MOEA/D/DDS$, after a new candidate solution is computed with initial step size $t_0 = 0.1$ we reduce its value such that $t_0 = 0.1\, t_0$ until a non dominated solution is found.

Chapter 4

The Gradient Subspace Approximation

Among all the state-of-the-art algorithms, memetic strategies have shown to be effective techniques to solve optimization problems. Memetic strategies hybridize a local search strategy with an evolutionary algorithm. In some of the local search procedures it is quite common to use gradient information. Unfortunately, there exist optimization problems where the gradient information is not explicitly given. If one needs to apply a method that uses gradient information, an alternative is to approximate such information. However, in some cases the cost in terms of function calls of approximation methods can be expensive. An example is the gradient approximation methods (e.g. see [Nocedal and Wright, 2006]) where the cost is linear and proportional to the dimension n of the decision space. In this case, when the number of parameters increases, the number of required function calls increases as well.

A possible alternative to avoid gradient dependence is to use direct search methods. Such methods explore several directions and compute a descent direction using them (e.g., [Zapotecas-Martnez and Coello, 2016]). Unfortunately, such techniques in principle also require at least n additional function calls for one local search move. This also classifies direct search techniques as expensive techniques.

Gradient Subspace Approximation (GSA) is a method that saves the extra function calls that other local search strategies require. In order to decrease the cost of the procedure, GSA computes an approximation of the gradient information using information that is already given. In particular, such information is obtained from the population of a given evolutionary algorithm. To save function calls the approxi-

mation is only computed for the subspace that is implicitly defined by the given data out of the population.

Another advantage of the GSA method is that both inequality and equality constraints can be incorporated directly into its computation. Thus, it gives us the opportunity to solve also constrained SOPs.

In this chapter we present the realization and application of the GSA method to solve optimization problems. The first part of the chapter presents the GSA method in the context of scalar optimization. A second approach extends the method in order to the context of MOPs.

4.1 Basic Idea

Let's consider a population-based algorithm. Such algorithm has certain information that is calculated at each generation (e.g., the function value of each individual). The main idea behind GSA is to compute a gradient approximation that in principle does not require additional function calls. But, the information comes out of the population.

Given a candidate solution $x_0 \in \mathbb{R}^n$ as well as r neighboring points $x_i \in N(x_0)$, a set of different search directions can be calculated. Now, lets consider that we can obtain $\nu_i, i = 1, \ldots, r$, as in Equation (3.13). Using such information, the aim of GSA is to compute an approximation of the gradient within:

$$W := \text{span}\{\nu_1, \ldots, \nu_r\}, \tag{4.1}$$

which is the subspace spanned by given search directions $\nu_i, i = 1, \ldots, r$.

For example lets consider the case presented in Figure 4.1. Starting with the candidate solution x_0 we can construct the two search direction along with x_a and x_b. Now, let's consider that x^* represents the optimal value of the problem. From the figure it is clear that there exist several directions in the subspace between ν_a and ν_b. But the direction ν is the direction with the most decay of the function. The aim of the GSA is to give a mathematical formulation to compute the descent direction ν.

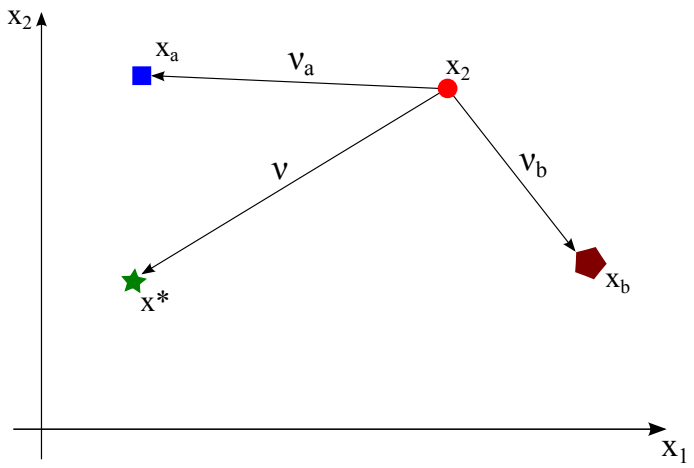


Figure 4.1: Example of the GSA subspace

The idea of GSA is to use it coupled along with set-based heuristics. In this way, it is possible to take advantage on each generation of the information computed by the heuristic procedures. Now, we proceed to give an example to measure the amount of neighboring information existing inside the current population of the adopted evolutionary algorithm. Here, we propose to use a population P generated by the Differential Evolution (DE) algorithm to conduct our experiment. N denotes the number of individuals in the population. Besides, in our experiment we labeled x_0 to the best individual found in population P. As a first step we compute the 2-norm of $\|x_i - x_0\|_2, x_i \in P \setminus x_0$. Given a $\delta \in \mathbb{R}$, the neighborhood of the solution x_0 is given by:

$$N(x_0) = \{x_i | x_i \in P \setminus x_0 \text{ and } \|x_i - x_0\|_2 \leq \delta\}. \tag{4.2}$$

Figure 4.2 shows the results of our experiment. We measured the number of neighboring solutions $r = |N(x_0)|$ of the best found individual on each generation of the algorithm. For this purpose, we take the DE/rand/bin variation. The figure presents the averaged results over 20 independent runs. The plots present the averaged number of neighboring solutions obtained on each generation. The experiments present the results using two different state-of-the-art-functions: Rosenbrock's [Rosenbrock, 1960] function on the left and Ackley's [Ackley, 1987] function (both are multi-modal) on the right. The number of parameters is set to $n = 10$ and several values of $\delta \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$ have been adopted.

The results of these experiments show that for different optimization functions it

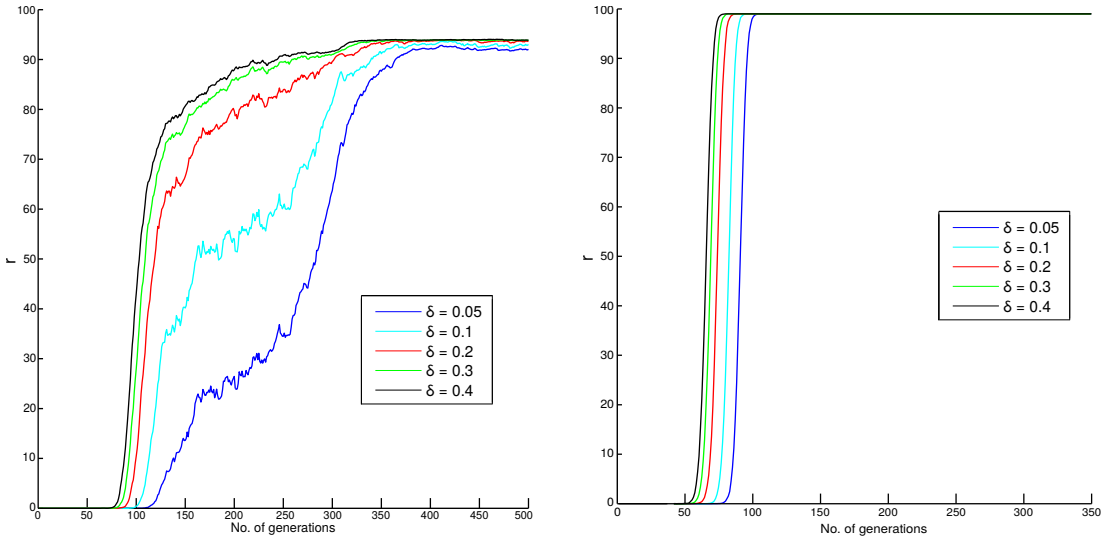


Figure 4.2: Results of neighboring experiments for SOPs.

is possible to obtain sufficient neighboring information. As stated above, such information is of particular interest when GSA is applied. As we observe, after a certain number of generations, the number of neighbors reaches more than 20 percent of the number of individuals. As the number of neighbors increases, the amount of information that can be incorporated into GSA also increases.

The idea to approximate the gradient and use it as a descent direction has already been studied. For example, it is possible to approximate the gradient using the finite difference (FD) method (e.g., [Nocedal and Wright, 2006]). In comparison, GSA also uses a technique that approximates the gradient, but GSA accepts in principle samples in all directions. Thus, GSA is more suited for population-based algorithms since neighboring individuals are typically not aligned in coordinate directions. Besides, if we compare both methods, GSA requires in principle, a lower number of function calls to compute an approximation in comparison with the FD method.

A similar idea can be found in [Brown and Smith, 2005a], where the authors use a technique to approximate the Jacobian of the objective map of a multi-objective optimization problem. The authors of [Brown and Smith, 2005a], however, restrict themselves on the unconstrained case and give no discussion on details such as step size control and integration into global search heuristics.

The resulting GSA standalone algorithm can be described as a variant of the

pattern search method ([Hooke and Jeeves, 1961]). Both algorithms use information within the neighborhood. But, the pattern search method samples new points in order to construct a descent direction. Meanwhile, GSA instead of sampling around the neighborhood computes the search direction based on them. If we compare both algorithms in terms of choosing sampling points we can claim that GSA is more flexible.

The use of pattern search inside a memetic algorithm is not a novel idea. For example, in [Zapotecas Martínez and Coello, 2012] it was used in the context of memetic multi-objective optimization and in [Bao et al., 2013] it was used for the parameter tuning of support vector machines. In those instances, it could be of interest to apply GSA as the local searcher engine and analyze the behavior of such algorithm in comparison with the pattern search technique.

In Figure 4.1 we present the main idea behind the GSA method. There, the ν_i search directions are plotted. But, it is important to mention that there exists more information that can be subtracted from the population P, e.g., the function value for the individuals.

4.2 GSA for Unconstrained SOPs

From now on we will proceed with the mathematical formulations to compute the approximation through the GSA method. Assume that a SOP as the one presented in Equation (2.5) is given. Besides, let's consider that we are solving an unconstrained problem, i.e., the number of equality and inequality constraints are equal to zero. For the given SOP, the most greedy search direction at a point $x_0 \in \mathbb{R}^n$ is given by:

$$g(x_0) := -\frac{\nabla f(x_0)}{\|\nabla f(x_0)\|_2} \in \mathbb{R}^n. \tag{4.3}$$

$g(x_0)$ can be expressed as the solution of the following SOP:

$$\begin{aligned} &\min_{\nu \in \mathrm{R}^n} \langle \nabla f(x_0), \nu \rangle \\ &\text{s.t. } \|\nu\|_2^2 = 1 \end{aligned}. \tag{4.4}$$

Furthermore, we consider that we are given search directions $\nu_1, \ldots, \nu_r \in \mathrm{R}^n$,

$r \leq n$. Define a matrix V as follows:

$$V = (\nu_1, \ldots, \nu_r) \in \mathrm{R}^{n \times r}, \tag{4.5}$$

where each column of matrix V represents one of the search directions. Until now, we claimed that it is possible to construct a descent direction ν in the subspace constructed using the search directions. That is, there exists a vector $\lambda \in \mathbb{R}^r$ such that:

$$\nu = \sum_{i=1}^{r} \lambda_i \nu_i = V\lambda. \tag{4.6}$$

If we incorporate the information presented in Equation (4.6) into the system of Equation (4.4), the most greedy search direction $\nu \in \text{span}\{\nu_1, \ldots, \nu_r\}$, can be computed by solving the following problem:

$$\begin{aligned} \min_{\lambda \in \mathrm{R}^r} & \left(\langle \nabla f(x_0), \sum_{i=1}^{r} \lambda_i \nu_i \rangle \right) = \left(\sum_{i=1}^{r} \lambda_i \langle \nabla f(x_0), \nu_i \rangle \right) \\ \text{s.t.} & \left\| \sum_{i=1}^{r} \lambda_i \nu_i \right\|_2^2 - 1 = \lambda^T V^T V \lambda - 1 = 0. \end{aligned} \tag{4.7}$$

Consider that $\lambda^* \in \mathrm{R}^r$ is a solution of (4.7), then we set

$$\nu^* := \sum_{i=1}^{r} \lambda_i^* \nu_i = V\lambda^* \tag{4.8}$$

as its associated search direction.

The next step to compute the approximation of the GSA is to find the solution of Equation (4.7). First, let's define the KKT system of (4.7) as follows:

$$\nabla_\lambda L(\lambda, \mu) = \begin{pmatrix} \langle \nabla f(x_0), \nu_1 \rangle \\ \vdots \\ \langle \nabla f(x_0), \nu_r \rangle \end{pmatrix} + 2\mu V^T V \lambda = 0 \tag{4.9}$$

$$h(\lambda) = \lambda^T V^T V \lambda - 1 = 0. \tag{4.10}$$

Since Equation (4.10) is considered only for normalization, if we remove it, the

system can be redefined as the following normal equation system:

$$V^T V\lambda = -V^T \nabla f(x_0). \tag{4.11}$$

Proposition 2. *Let $\nu_1, \ldots, \nu_r \in \mathrm{R}^n$, $r \leq n$, be linearly independent and*

$$\tilde{\lambda}^* := -(V^T V)^{-1} V^T \nabla f(x_0). \tag{4.12}$$

Then

$$\lambda^* := \frac{\tilde{\lambda}^*}{\|V\lambda^*\|_2^2} \tag{4.13}$$

is the unique solution of (4.7), and thus,

$$\nu^* = \frac{-1}{\|V\lambda^*\|_2^2} V(V^T V)^{-1} V^T \nabla f(x_0) \tag{4.14}$$

is the most greedy search direction in span$\{\nu_i, \ldots, \nu_r\}$.

Proof. Follows by the above discussion. We set $2\mu = \|V\lambda^*\|_2^2$. Next, we apply Equations (4.12) and (4.13) onto the Lagrangian of Equation (4.10):

$$\nabla_\lambda L(\lambda^*, \frac{\|V\lambda^*\|_2^2}{2}) = V^T \nabla f(x_0) + \|V\lambda^*\|_2^2 V^T V \left(\frac{-(V^T V)^{-1} V^T \nabla f(x_0)}{\|V\lambda^*\|_2^2} \right) = 0. \tag{4.15}$$

Performing the mathematical operations in Equation (4.15):

$$\nabla_\lambda L(\lambda^*, \frac{\|V\lambda^*\|_2^2}{2}) = V^T \nabla f(x_0) - V^T \nabla f(x_0) = 0. \tag{4.16}$$

Equation (4.16) states that the solution λ^* is indeed a solution of Equation (4.7). To determine if λ^* is a minimum we proceed to compute the Hessian matrix. Such a matrix is given by:

$$\nabla^2_{\lambda\lambda} L(\lambda, \mu) = V^T V. \tag{4.17}$$

From Equation (4.17) it is clear that for any value of λ the Hessian matrix is always positive definite. Using both statements (from Equations (4.16) and (4.17)) we found that our proposition is correct.

□

It is important to mention that the solution of the system of Equation (4.7)

presents several properties that have to be analyzed. Such properties can be enumerated as follows:

- Let's define a function $f_\alpha(\lambda)$ such that:

$$f_\alpha(\lambda) := \sum_{i=1}^{r} \lambda_i \langle \nabla f(x_0), \nu_i \rangle. \tag{4.18}$$

  If λ^* is a solution of (4.7) then $f_\alpha(\lambda^*) \leq 0$.

  To see this, $\tilde{\lambda} := -\lambda^*$ is also feasible and it holds $f(\tilde{\lambda}) = -f(\lambda^*) < 0$.
  Thus, if there exists a direction ν_i, $i \in \{1, \ldots, r\}$, such that $\langle \nabla f(x_0), \nu_i \rangle \neq 0$, then $f(\lambda^*) < 0$, and hence, the related direction ν^* is a descent direction. Further, note that for a randomly chosen direction $\nu_i \in \mathrm{R}^n$ the probability that $\langle \nabla f(x_0), \nu_i \rangle = 0$ is zero. Hence, the probability is one to obtain a descent direction via (4.7) using randomly chosen directions.

- If more search directions are incorporated into the system of Equation (4.7) the resulting search direction becomes better, i.e., a more greedy direction is computed. To be more precise, let λ_r^* be the solution of (4.7) using the directions $\nu_1, \ldots, \nu_r$, and λ_{r+l}^* be the solution of (4.7) where the search directions $\nu_1, \ldots, \nu_r, \nu_{r+1}, \ldots, \nu_{r+l}$ are used, then $f(\lambda_{r+l}^*) \leq f(\lambda_r^*)$. For $r = n$ and if the ν_i's are linearly independent, then ν^* coincides with g.

- Since we are solving a normal system of equations, the numerics of the problem needs to be analyzed. In particular, let's analyze the condition number of Equation (4.11). In order to find a solution of such system it is necessary that the condition number of $V^T V$ does not increase. In Equation (4.11), if the rank of V is maximal it holds that $\kappa_2(V^T V) = \kappa(V)^2$. That is, if the rank of the matrix V starts to increase and the system will become numerically unstable. In order to avoid such instability, a process to check the search directions incorporated into V has to be considered. Hence, when choosing the ν_i search directions, the condition number of V must be computed. In particular, directions that point nearly in the same direction have to be avoided.

- As a result of the previous observation, the orthogonal directions are preferred for the construction of matrix V. In that case, we obtain

$$\nu^* = \frac{-1}{\|\lambda^*\|_2^2} V V^T \nabla f(x_0), \tag{4.19}$$

i.e., the orthogonal projection of $\nabla f(x_0)$ onto $span\{\nu_1, \ldots, \nu_r\}$. Hence, ν^* can be seen as the best approximation of the most greedy search direction g in the subspace $span\{\nu_1, \ldots, \nu_r\}$.

As we claimed above we use the information given by a population within an evolutionary algorithm to compute the missing information. The first piece of missing information are the search directions computed. Hence, we can obtain such information using Equation (3.13).

Having $x_i, i = 1, \ldots, r$ solutions in the neighborhood $N(x_0)$, first we need to define a matrix $\tilde{V} \in \mathbb{R}^{r \times n}$ as:

$$\tilde{V} = (\tilde{\nu}_i, \ldots, \tilde{\nu}_r), \tag{4.20}$$

where each column vector of matrix $\tilde{V}$ can be obtained by Equation (3.13).

The second piece of information that can be obtained is an approximation of the directional derivatives. Assume that beside the x_i points, we also have their function value $f(x_i)$. Then, it is possible to approximate the directional derivatives for each of the search directions as follows:

$$\langle \nabla f_i(x_0), \nu_j \rangle = \frac{f(x_i) - f(x_0)}{\|x_i - x_0\|_2} + O(\|x_i - x_0\|), \qquad i = 1, \cdots, r. \tag{4.21}$$

Performing the proper substitution onto (4.7), we start our computations by solving:

$$\begin{aligned} &\min_{\lambda \in \mathrm{R}^r} \left(\sum_{i=1}^{r} \lambda_i \langle \nabla f(x_0), \nu_i \rangle \right). \\ &\text{s.t. } \lambda^T \tilde{V}^T \tilde{V} \lambda - 1 = 0 \end{aligned} \tag{4.22}$$

Now, in order to remove the gradient dependency we define a vector $\tilde{d} \in \mathbb{R}^r$ as follows:

$$d_a = \begin{pmatrix} \langle \nabla f(x_0), \nu_1 \rangle \\ \vdots \\ \langle \nabla f(x_0), \nu_r \rangle \end{pmatrix}. \tag{4.23}$$

Instead of using the exact gradient information to compute the vector d_a, we use the approximation of Equation (4.21). Hence, we can perform our computation using a vector $\tilde{d} \approx d_a$. Substituting the vector $\tilde{d}$ in Equation (4.22) the SOP can be modified as follows:

$$\begin{array}{l} \min\limits_{\lambda \in \mathrm{R}^r} \tilde{\lambda}^T \tilde{d} \\ \text{s.t. } \tilde{\lambda}^T \tilde{V}^T \tilde{V} \tilde{\lambda} - 1 = 0 \end{array}. \tag{4.24}$$

Following a similar procedure as the one used to obtain $\tilde{\lambda}^*$, we found that the solution of Equation (4.24) is given by:

$$\tilde{V}^T \tilde{V} \tilde{\lambda} = -\tilde{d}. \tag{4.25}$$

If $\hat{\lambda}^*$ is a solution of Equation (4.25), the most greedy search direction can be approximated as follows:

$$\tilde{\nu}^* = \frac{-1}{\|\tilde{V}\hat{\lambda}^*\|_2^2} \tilde{V} (\tilde{V}^T \tilde{V})^{-1} \tilde{d}. \tag{4.26}$$

A particular solution that can be obtained solving Equation (4.25) occurs when coordinate directions are chosen, i.e.,

$$x_i = x_0 + t_i e_{j_i}, \ i = 1, \ldots, r, \tag{4.27}$$

where e_j denotes the j-th unit vector, we compute for the j_i-th entry of $\tilde{\nu}^*$ (without normalization)

$$\tilde{\nu}^*_{j_i} = \frac{f(x_0 + t_i e_{j_i}) - f(x_0)}{|t_i|}. \tag{4.28}$$

That is, for $x_i = x_0 + t_i e_i$, $i = 1, \ldots, n$, the search direction coincides with the result of the forward difference method.

4.3 GSA for Constrained SOPs

As mentioned before, the incorporation of constraints into the formulation of the GSA is a straight-forward procedure. In the following section we present the mechanisms to incorporate constraints into the realization of the GSA method.

4.3.1 Equality constraints

Let's start by taking a SOP with only equality constraints. That is, consider the SOP of Equation (2.5) and set $m = 0$. Performing such reformulation, the SOP to be solved is defined as follows:

$$\begin{aligned} &\min_{x \in \mathrm{R}^n} f(x) \\ &\text{s.t. } h_i(x) = 0,\ i = 1, \ldots, p \end{aligned} . \tag{4.29}$$

In order to apply the GSA method we consider that each constraint $h_i : \mathrm{R}^n \to \mathrm{R}$ is differentiable. In the unconstrained GSA formulation we were only concerned with finding the most greedy direction. But in this case, now we are also concerned on how the equality constraints are involved into the GSA formulation. Consider that we try to find a direction that presents the maximal decay of f, we transform the Problem (4.29) as follows:

$$\begin{aligned} &\min_{\nu \in \mathrm{R}^n} \langle \nabla f(x_0), \nu \rangle \\ &\text{s.t. } \|\nu\|_2^2 = 1 \\ &\qquad \langle \nabla h_i(x), \nu \rangle = 0,\ i = 1, \ldots, p. \end{aligned} \tag{4.30}$$

Furthermore, it is important to consider that we are looking for a descent direction that exists in the subspace created by the search directions $\nu_i,\ i = 1, \ldots, r$. To perform such modification we proceed in a similar way that for the unconstrained case. That is, we introduce a vector $\lambda \in \mathbb{R}^r$ used to combine the ν_i search directions. Applying the λ vector into Equation (4.30) the SOPs can be rewritten as follows:

$$\begin{aligned} &\min_{\lambda \in \mathrm{R}^r} \sum_{i=1}^{r} \lambda_i \langle \nabla f(x_0), \nu_i \rangle \\ &\text{s.t. } \lambda^T V^T V \lambda - 1 = 0 \\ &\qquad \sum_{i=1}^{r} \lambda_i \langle \nabla h_j(x_0), \nu_i \rangle = 0,\ j = 1, \ldots, p \end{aligned} \tag{4.31}$$

Now that we define the SOP for equality constrained problems we proceed to

compute its solution. First, we define a matrix $H \in \mathbb{R}^{p \times n}$ as:

$$H := \begin{pmatrix} \nabla h_1(x_0)^T \\ \vdots \\ \nabla h_p(x_0)^T \end{pmatrix}. \tag{4.32}$$

Using the KKT system of Equation (4.31) reads as:

$$\begin{aligned} V^T \nabla f(x_0) + 2\mu_0 V^T V \lambda + (HV)^T \mu &= 0 && (4.33) \\ HV\lambda &= 0 && (4.34) \\ \lambda^T V^T V \lambda - 1 &= 0. && (4.35) \end{aligned}$$

Once again we can apply the same 'normalization trick' as for (4.11). Hence, removing the normalization term we can transform the Equation (4.35) into:

$$\begin{pmatrix} V^T V & H^T V^T \\ HV & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} -V^T \nabla f(x_0) \\ 0 \end{pmatrix} \tag{4.36}$$

Proposition 3. *Let $\nu_1, \ldots, \nu_r \in \mathrm{R}^n$ be linearly independent where $p \leq r \leq n$, let $rank(H) = p$, and*

$$\begin{pmatrix} \tilde{\lambda}^* \\ \tilde{\mu}^* \end{pmatrix} = \begin{pmatrix} V^T V & H^T V^T \\ HV & 0 \end{pmatrix}^{-1} \begin{pmatrix} -V^T \nabla f(x_0) \\ 0 \end{pmatrix}, \tag{4.37}$$

then

$$\lambda^* := \frac{\tilde{\lambda}^*}{\|V\lambda^*\|_2^2} \tag{4.38}$$

is the unique solution of (4.31), and thus

$$\nu^* = \frac{-1}{\|V\lambda^*\|_2^2} V (V^T V)^{-1} V^T \nabla f(x_0) \tag{4.39}$$

is the most greedy search direction in span$\{\nu_i, \ldots, \nu_r\}$.

Proof. To show that the matrix in (4.37) is regular, let $y \in \mathrm{R}^r$ and $z \in \mathrm{R}^p$ such that

$$\begin{pmatrix} V^T V & H^T V^T \\ HV & 0 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = 0 \tag{4.40}$$

It follows that $HVy = 0$ and hence that

$$0 = \begin{pmatrix} y \\ z \end{pmatrix}^T \begin{pmatrix} V^TV & H^TV^T \\ HV & 0 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = yV^TVy \tag{4.41}$$

Thus, it is $y = 0$ since V^TV is positive definite. Further, by (4.40) it follows that $V^TH^Tz = 0$. Since $V^T \in \mathrm{R}^{n\times r}$ has rank $r \geq p$, it follows that V^TH^T has rank p. This implies that also $z = 0$, and thus, that the matrix in (4.37) is regular.

Now that it has been demonstrated that λ^* is a solution of Equation (4.37) the next step is to demonstrate that such value is a minimum. First, we can compute the Hessian of the Lagrangian as follows:

$$\nabla^2_{\lambda\lambda}L(\lambda,\mu) = 2\mu_0V^TV. \tag{4.42}$$

If we substitute the value of μ_0 with $\| \sum_{i=1}^r \tilde{\lambda}_i^*\nu_i\|_2^2$ the equation of the Hessian is transformed into:

$$\nabla^2_{\lambda\lambda}L(\lambda,\mu) = V^TV, \tag{4.43}$$

where for any value of λ, the matrix is positive definite (which implies that the solution is always a minimum).

□

Again, our computations assumed that certain pieces of information were given at the start of the process. Using Equations (4.21) and (3.13) we can approximate some of the missing information, i.e., the directional derivatives and the search directions, respectively. But such information is not enough to completely remove the gradient information requirement. Here, the approximation of the directional derivatives of the equality constraints is also required. To perform such approximation, assume that x_0, the points $x_1, \ldots, x_r$ along with their respective constraint values: $h(x_0)$ and $h(x_1), \ldots, h(x_r)$ are given. In principle such information is taken from the population-based algorithm. Analogously to Equation (4.21) we can compute an approximation for each directional derivative $\langle \nabla h_j(x_0), \nu_i\rangle$, $i = 1, \ldots, r$, $j = 1, \ldots, p$ as follows:

$$\langle \nabla h_j(x_0), \nu_i\rangle \approx m_{ji} = \frac{h_j(x_i) - h_j(x_0)}{\|x_i - x_0\|_2}. \tag{4.44}$$

If we remember that the system proposed in Equation (4.37) uses derivative in-

formation (the matrix HV is defined using the gradients), such matrix can be defined in the following form:

$$HV = \begin{pmatrix} \langle \nabla h_1(x_0), \nu_1 \rangle & \dots & \langle \nabla h_1(x_0), \nu_r \rangle \\ \vdots & & \vdots \\ \langle \nabla h_p(x_0), \nu_1 \rangle & \dots & \langle \nabla h_p(x_0), \nu_r \rangle \end{pmatrix} \in \mathbb{R}^{p \times r}. \tag{4.45}$$

Now, we are in position to compute an approximation for the matrix HV instead. Defining a matrix $\tilde{M} \in \mathbb{R}^{p \times r}$ where each entry is computed by Equation (4.44):

$$HV \approx \tilde{M} := \begin{pmatrix} m_{11} & \dots & m_{1r} \\ \vdots & & \vdots \\ m_{p1} & \dots & m_{pr} \end{pmatrix}. \tag{4.46}$$

Using the approximation matrix $\tilde{M}$ we can transform Equation (4.37) as follows:

$$\begin{pmatrix} \tilde{V}^T \tilde{V} & \tilde{M}^T \\ \tilde{M} & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} -d \\ 0 \end{pmatrix}. \tag{4.47}$$

The solution of Equation (4.47) gives the gradient-free realization to compute the most greedy direction of an SOP with only equality constraints.

4.3.2 Inequality constraints

In our previous formulation we incorporated equality constraints into GSA. The next step of the realization of the algorithm is to incorporate inequality constraints. Hence, we start with the system described in Equation (4.7). Consider that there exist m inequality constraints in the SOP. Let $l \leq m$ be the number of active constraints, thus the problem can be reformulated as:

$$\begin{aligned} &\min_{x \in \mathrm{R}^n} f(x) \\ &\text{s.t. } g_i(x) \leq 0,\ i = 1, \dots, l. \end{aligned} \tag{4.48}$$

Departing from the SOP of Equation (4.48), we start the realization of the GSA method with inequality constraints. Performing the proper modifications to Equation (4.4) we assume that the most greedy search direction can be found by solving the

following problem:

$$\begin{aligned} \min_{\nu \in \mathrm{R}^n} \; & \langle \nabla f(x_0), \nu \rangle \\ \text{s.t. } & \|\nu\|_2^2 = 1 \\ & \langle \nabla g_i(x), \nu \rangle \leq 0, \; i = 1, \ldots, l \end{aligned} \quad . \tag{4.49}$$

As in the previous realizations of the GSA, we consider that the direction ν exists in a subspace created by the search directions ν_i, $i = 1, \ldots, r$. As in previous formulations, we incorporate a value λ into our system in order to create the most greedy direction. Thus, the optimization problem can be reformulated as

$$\begin{aligned} \min_{\lambda \in \mathrm{R}^r} \; & \sum_{i=1}^{r} \lambda_i \langle \nabla f(x_0), \nu_i \rangle \\ \text{s.t. } & \lambda^T V^T V \lambda - 1 = 0 \\ & \sum_{i=1}^{r} \lambda_i \langle \nabla g_j(x_0), \nu_i \rangle \leq 0, \; j = 1, \ldots, l. \end{aligned} \tag{4.50}$$

There exist different methods to numerically solve the system of Problem (4.50). For example, the system can be solved numerically by using active set methods (e.g., [Nocedal and Wright, 2006]) together with the results obtained on equality constrained problems. Another possibility to find a solution to (4.50) can be obtained using a gradient projection method. In particular, we use the gradient projection method to handle inequality constraints.

We start our formulation considering a problem with only one active inequality constraint (i.e., $l = 1$) (later the realization is going to be extended for the general case). The projection method performs an orthogonal projection of the search direction ν^*, which is the solution of the unconstrained SOP (Equation 4.7), to the orthogonal space of $\nabla g(x_0)$. Figure 4.3 presents an example on how the projection of the direction ν^* occurs. To calculate such projection, we propose to compute a QR decomposition of $\nabla g(x_0)$, i.e.,

$$\nabla g(x_0) = QR = (q_1, \ldots, q_n)R. \tag{4.51}$$

The vectors $q_2, \ldots, q_n$ obtained in Equation (4.51) build an orthonormal basis of

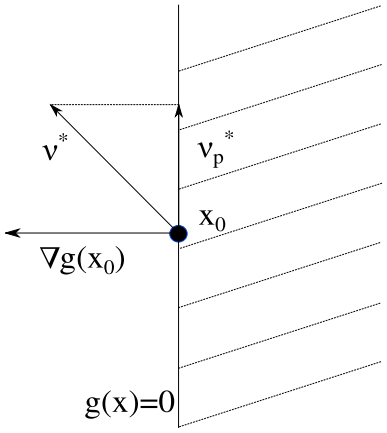


Figure 4.3: Gradient projection with one active inequality constraint

$\nabla g(x_0)^{\perp}$. Defining $Q_g = (q_2, \ldots, q_n)$, the projected vector is hence given by:

$$\nu_{new} = Q_g Q_g^T \nu^*. \tag{4.52}$$

In the previous formulation we considered that the gradient information was given. But, as in the previous formulation for the GSA method we will formulate the gradient-free realization of the method. Here, we consider a different method to calculate an orthonormal basis. Define a matrix $M \in \mathrm{R}^{1\times r}$ as follows:

$$M := \nabla g(x_0)^T V = (\langle \nabla g(x_0), \nu_1 \rangle, \ldots, \langle \nabla g(x_0), \nu_r \rangle). \tag{4.53}$$

Thus, if a vector w is in the kernel of M this is equivalent to having Vw to be orthogonal to $\nabla g(x_0)$. Hence, we can compute the matrix:

$$K = (k_1, \ldots, k_{r-1}) \in \mathrm{R}^{r\times(r-1)}, \tag{4.54}$$

where its column vectors build an orthonormal basis of the kernel of M. If the search directions $\nu_i, i = 1, \ldots, r$ are orthogonal, then also the vectors $Vk_1, \ldots, Vk_{r-1}$ are orthogonal. Such vectors are the column vectors of $VK \in \mathrm{R}^{n\times(r-1)}$. Hence, the projected vector to the kernel of M is given by:

$$\tilde{\nu}_{new} = VK(VK)^T \nu^* = VKK^T V^T \nu^* \tag{4.55}$$

To extend the projection method to a general value of l we proceed to define a

matrix $G \in \mathbb{R}^{l \times r}$ as follows:

$$G := \begin{pmatrix} \nabla g_1(x_0)^T \\ \vdots \\ \nabla g_l(x_0)^T \end{pmatrix}. \tag{4.56}$$

Using the matrix of Equation (4.56) now let's define a matrix $\hat{M} \in \mathbb{R}^{l \times r}$ constructed using the directional derivatives information of the active inequality constraints:

$$\hat{M} := GV = \begin{pmatrix} \langle \nabla g_1(x_0), \nu_1 \rangle & \dots & \langle \nabla g_1(x_0), \nu_r \rangle \\ \vdots & & \vdots \\ \langle \nabla g_l(x_0), \nu_1 \rangle & \dots & \langle \nabla g_l(x_0), \nu_r \rangle \end{pmatrix}. \tag{4.57}$$

Once we have defined the matrix $\hat{M}$, we proceed to construct the projection of the direction ν^*. The first step in such computation is to obtain an orthonormal basis $K \in \mathrm{R}^{r \times (r-l)}$ of the kernel of M. Using such basis we proceed to compute a QR decomposition as follows:

$$VK = QR = (q_1, \dots, q_{r-m}, \dots, q_n)R. \tag{4.58}$$

Now that we obtained the matrix Q we are in position to perform the projection of the direction ν^*. Our next step is to define a matrix $O \in \mathrm{R}^{n \times (r-l)}$ using the column vectors of Q as follows:

$$O := (q_1, \dots, q_{r-l}) \tag{4.59}$$

The next step is to compute the projected direction ν_p^*. Such direction can be computed as follows:

$$\nu_p^* := OO^T \nu^*. \tag{4.60}$$

Finally, the last step of the realization is to remove the gradient information requirement. Similar to the previous formulation of the GSA we proceed to approximate the derivative information. Hence, consider that the following pieces of information are given: a candidate solution x_0, the value of its active constraints $g_j(x_0)$, $j = 1, \dots, l$, its neighboring solutions $x_i, i = 1 \dots, r$, and the information of the active inequality constraints of the neighbors, i.e. $g_j(x_i)$, $i = 0, \dots, r$, $j = 1, \dots, l$. Besides, we can easily obtain the matrix V using Equation (3.13). Finally, we can

compute an approximation of any of the directional derivatives as follows:

$$\bar{m}_{ji} = \langle \nabla g_j(x_0), \nu_i \rangle := \frac{g_j(x_i) - g_j(x_0)}{\|x_i - x_0\|_2}, \tag{4.61}$$

where $i \in \{1 \dots r\}$ and $j \in \{1 \dots l\}$.

Hence, instead of directly using matrix $\hat{M}$ we can instead use an approximation of it. Define a matrix $\bar{M} \in \mathbb{R}^{l \times r}$ as follows:

$$\hat{M} \approx \bar{M} = \begin{pmatrix} \bar{m}_{11} & \dots & \bar{m}_{1r} \\ \vdots & & \vdots \\ \bar{m}_{l1} & \dots & \bar{m}_{lr} \end{pmatrix}. \tag{4.62}$$

Thus, we can perform the computation of the orthonormal basis $\bar{K} \in \mathrm{R}^{r \times (r-l)}$ of the kernel $\bar{M}$. Next, we proceed to compute the QR decomposition such that:

$$V\bar{K} = QR = (q_1, \dots, q_{r-m}, \dots, q_n)R. \tag{4.63}$$

Finally once we have obtained the basis Q we can obtain the projected direction ν_p^* using Equation (4.60).

4.4 GSA as Standalone Algorithm

In the previous section we describe several formulations to construct a descent direction using the GSA method. Such formulations will help us to describe an algorithm for the standalone GSA method. This section presents all the elements required by the GSA method to define such algorithm.

4.4.1 Computing the direction

The first important step is to unify all the ideas to compute a descent direction. That is, we present the algorithms to compute a descent direction for any given differentiable SOP. From here on we will consider that there exists a method $\mathcal{N} = \text{computeNeighborhood}(x_0)$. Such method, as its name indicates, returns the neighboring information from a given candidate solution x_0. The information that can accessed using the neighborhood is the following:

- $\mathcal{N} \rightarrow x_i$. It refers to the parameter vector of the i-th neighbor.
- $\mathcal{N} \rightarrow f_i$. It refers to the function value of the i-th neighbor.
- $\mathcal{N} \rightarrow h_i$. It refers to the function vector that contains the values of equality constraints of the i-th neighbor.
- $\mathcal{N} \rightarrow g_i$. It refers to the function vector that contains the values of inequality constraints of the i-th neighbor.

For now, we consider that the neighboring procedure is given and we will not enter into its details. It will be defined later on since its definition depends on how the GSA is used: as a standalone algorithm or inside an evolutionary algorithm.

Our first study case is how to compute a descent direction for unconstrained problems. Algorithm (12) presents the pseudocode to compute the direction ν^* of a candidate solution. In the algorithm, $r_{\mathcal{N}}$ is defines the neighborhood size. Meanwhile r refers to the parameter described in the GSA procedure. It is important to remember that we must provide certain stability to the system in numerical terms. So, we propose to use a threshold δ in order to control the condition number of matrix V.

In case that we need to solve a constrained SOP there exist two different cases of study for the candidate solution x_0: the solution is feasible or the solution is not feasible. In case that the solution is not a feasible one, it is desirable to compute a direction that steer the search into the feasible region. In order to compute such direction we propose to apply a penalty function along with the GSA method. Given a candidate solution along with its constraint values (both types of them: equality and inequality constraint values). The penalty function $\mathcal{P}(x, g(x), h(x)) \in \mathbb{R}$ is defined as follows:

$$\mathcal{P}(x, g(x), h(x)) = f(x) + C_p \sum_{i=1}^{l} \max\left(0, g_i(x)\right) + C_p \sum_{j=1}^{p} h_j(x)^2, \tag{4.64}$$

where $g_i(x_0), i = 1, \ldots, m$ are the inequality constraints. Meanwhile, $h_j(x_0), j = 1, \ldots, p$ represents the equality constraint values. The function uses a constant $C_p = 10^3$ to compute the penalty value.

Algorithm 12 Pseudocode for unconstrained direction of the GSA.

Require: Initial point x_0, $f_0 = f(x_0)$, GSA neighborhood size r, threshold δ
Ensure: Direction $\tilde{\nu}^*$

Set $\mathcal{N} :=$ computeNeighborhood(x_0)
Set $r_{\mathcal{N}} := |\mathcal{N}|$
$V := \emptyset$
$\tilde{d} := 0 \in \mathbb{R}^r$
$j := 0$
for $i = 1, \ldots, r_{\mathcal{N}}$ **do**
  Set $x_a := \mathcal{N} \to x_i$
  Set $f_a := \mathcal{N} \to f_i$
  Compute $\nu_a := \frac{x_a - x_0}{\|x_a - x_0\|_2}$
  Add ν_a to V as the column at the $(j+1)$-th position
  if cond$(V) \geq \delta$ **then**
    Remove column at the $(j+1)$-th position of V
  else
    Compute $\tilde{d}_{j+1} := \frac{f_a - f_0}{\|x_a - x_0\|_2}$
    Set $j := j + 1$
    if $j \geq r$ **then**
      break
    end if
  end if
end for
if $j \leq r$ **then**
  return $0 \in \mathbb{R}^n$
else
  Compute the solution $\hat{\lambda}^*$ of Equation (4.25)
  Compute $\tilde{\nu}^*$ using Equation (4.26)
  return $\tilde{\nu}^*$
end if

Using the penalty function described in Equation (4.64) we propose to modify the GSA method to use it. That is, we define a vector $\bar{p} \in \mathbb{R}^r$ as follows:

$$\bar{p} = \begin{pmatrix} \bar{p}_1 \\ \vdots \\ \bar{p}_r \end{pmatrix}, \tag{4.65}$$

where each vector entry $\bar{p}_i$ is given by:

$$\bar{p}_i = \frac{P(x_i, g(x_i), h(x_i)) - P(x_0, g(x_0), h(x_0))}{\|x_i - x_0\|_2}, \; i \in \{1, \ldots, r\}. \tag{4.66}$$

Now we are in position to define an algorithm that computes a direction to steer the candidate solution into a feasible region. Algorithm (13) presents the computation of the search direction using Equation (4.66). Using the penalty function we take the system of Equation (4.25) an reformulate it as follows:

$$\tilde{V}^T\tilde{V}\hat{\lambda} = -\bar{p}. \tag{4.67}$$

Finally, the GSA direction can be obtained as follows:

$$\bar{\nu}^* = \tilde{V}\hat{\lambda}. \tag{4.68}$$

The last direction to be computed is the one for constrained SOPs when a feasible point is given. To obtain such a direction, we start with the direction for equality constraint problems and if there exist inequality constraints we proceed to correct back the direction. Algorithm 16 presents the realization of the GSA constrained direction. In the algorithm, l refers to the number of active constraints of x_0. To compute the indexes of the inequality constraints elements we propose the indexActive($g(x_0)$) procedure. Such function return the indexes $\mathcal{I}$ such that:

$$\mathcal{I} = \{i | i \in \{1, \ldots, m\} \cup g_i(x_0) \leq 0 \cup |g_i(x_0)| \leq \epsilon\}, \tag{4.69}$$

where ϵ is a threshold used to detect the active inequality constraints. In our case we set $\epsilon = 10^{-5}$.

4.4.2 Correcting the step size

Now that we defined the computations for GSA directions, the next step is to define the step size control algorithm. Typically, The GSA is used inside a line search technique. Such technique uses any of the computed directions described above. Unfortunately, the line search techniques require extra parameters to be defined, e.g. the initial step size t_0. Moreover, the exist some cases when a backtracking step is required in order to find the optimal value. Consider the example of Figure 4.4 where x_i represents the actual candidate solution, x^* represents the optimal value and $x_{i+1} = x_i + t_0\nu$. On the figure there exist three different regions: A, B and C. We assume that for any three points $a \in A$, $b \in B$ and $c \in C$ it holds that $f(a) < f(b) < f(c)$. The example shows that even when we use a descent direction

Algorithm 13 GSA direction on infeasible solutions.

Require: Initial point x_0, $f_0 = f(x_0)$, $g_0 = g(x_0)$, $h_0 = h(x_0)$, GSA neighborhood size r, threshold δ
Ensure: Direction $\bar{\nu}^*$

Compute $p_0 = \mathcal{P}(x_0, g_0, h_0)$
Set $\mathcal{N} :=$ computeNeighborhood(x_0)
Set $r_{\mathcal{N}} := |\mathcal{N}|$
$V := \emptyset$
$\bar{p} := 0 \in \mathbb{R}^r$
$j := 0$
for $i = 1, \ldots, r_{\mathcal{N}}$ **do**
  Set $x_a := \mathcal{N} \to x_i$
  Set $f_a := \mathcal{N} \to f_i$
  Set $g_a := \mathcal{N} \to g_i$
  Set $h_a := \mathcal{N} \to h_i$
  Compute $p_a = \mathcal{P}(x_a, g_a, h_a)$
  Compute $\nu_a := \frac{x_a - x_0}{\|x_a - x_0\|_2}$
  Add ν_a to V as the column at the $(j+1)$-th position
  if cond$(V) \geq \delta$ **then**
    Remove column at the $(j+1)$-th position of V
  else
    Compute $\hat{p}_a = \frac{p_a - p_0}{\|x_a - x_0\|_2}$
    Set the $(j+1)$-th entry of vector $\bar{p}$ as $\hat{p}_a$
    Set $j := j + 1$
    if $j \geq r$ **then**
      break
    end if
  end if
end for
if $j \leq r$ **then**
  return $0 \in \mathbb{R}^n$
else
  Solve $\tilde{V}^T \tilde{V} \hat{\lambda} = -\bar{p}$
  Compute $\bar{\nu}^*$ as in Equation (4.68)
  return $\bar{\nu}^*$
end if

(it leads the search into region A) we do not find a solution such that $f(_{i+1}) < f(x_i)$. Unfortunately, since we take a large value for t_0, we reached the C region and the function value increased.

The problem described above is just a rough example on how the size of the step

Algorithm 14 GSA direction on feasible solutions.

Require: Initial point x_0, $f_0 = f(x_0)$, $g_0 = g(x_0)$, $h_0 = h(x_0)$, GSA neighborhood size r, threshold δ
Ensure: Direction $\tilde{\nu}^*$

Set $\mathcal{N} :=$ computeNeighborhood(x_0)
Set $r_{\mathcal{N}} := |\mathcal{N}|$
$V := \emptyset, \tilde{M} := \emptyset, \hat{M} := \emptyset$
$\hat{d}_{\mathcal{P}} := 0 \in \mathbb{R}^r$
$\mathcal{I} =$ indexActive(g_0)
$p = |h_0|$
$\bar{g}_0 =$ obtainActiveConstraints(g_0, I)
$l = |\mathcal{I}|$
$j := 0$
for $i = 1, \ldots, r_{\mathcal{N}}$ **do**
  Set $x_a := \mathcal{N} \to x_i$
  Set $f_a := \mathcal{N} \to f_i$
  if $l > 0$ **then**
    $g_a =$ obtainActiveConstraints$(\mathcal{N} \to g_i, I)$
  end if
  Compute $\nu_a := \frac{x_a - x_0}{\|x_a - x_0\|_2}$
  Add ν_a to V as the column at the $(j+1)$-th position
  if cond$(V) \geq \delta$ **then**
    Remove column at the $(j+1)$-th position of V
  else
    Compute $\hat{d}_{j+1} := \frac{f_a - f_0}{\|x_a - x_0\|_2}$
    if $l > 0$ **then**
      Compute $\bar{m} := \frac{g_a - \bar{g}_0}{\|x_a - x_0\|_2} \in \mathbb{R}^l$
      Set $\bar{m}$ as the $(j+1)$-th column of $\hat{M}$
    end if
    if $p > 0$ **then**
      Compute $m := \frac{h_a - h_0}{\|x_a - x_0\|_2} \in \mathbb{R}^p$
      Set m as the $(j+1)$-th column of $\tilde{M}$
    end if
    Set $j := j + 1$
    if $j \geq r$ **then**
      break
    end if
  end if
end for
if $j \leq r$ **then**
  return $0 \in \mathbb{R}^n$
else
  Compute $\nu^* =$ solveDirection$(V, \hat{d}, \tilde{M}, p)$
  Compute $\nu_p^* =$ projectDirection$(\nu^*, V, \hat{M})$
  return ν_p^*
end if

Algorithm 15 $\nu^* = \text{solveDirection}(\tilde{V}, \hat{d}, \tilde{M}, p)$

Require: Approximation vector $\hat{d}$, Neighborhood matrix $\tilde{V}$, Equality approximations $\tilde{M}$, number of equality constraints p
Ensure: Direction ν^*
1: **if** $p > 0$ **then**
2: Compute $\tilde{\lambda}$ as in Equation (4.47)
3: **else**
4: Compute $\tilde{\lambda}$ as in Equation (4.25)
5: **end if**
6: Compute $\nu^* = \tilde{V}\tilde{\lambda}$
7: **return** ν^*

Algorithm 16 $\nu_p^* = \text{projectDirection}(\nu^*, V, \hat{M})$

Require: Direction ν^*, Neighbor matrix $\tilde{V}$, Inequality approximations $\hat{M}$, Number of active inequality constraints l
Ensure: Projected direction nu_p^*
1: **if** $l == 0$ **then**
2: **return** ν^*
3: **else**
4: Compute orthonormal basis VK of matrix $\hat{M}$
5: Compute QR decomposition of VK
6: Compute O matrix as in Equation (4.59)
7: Compute projected direction nu_p^* as in Equation (4.60)
8: **return** ν_p^*
9: **end if**

affects the search. In this section we will describe the algorithm to adjust the value of t such that we find a new point that improves the candidate solution. Our formulation for the step size is based in the Wolfe conditions [Wolfe, 1969]. Considering that we are given a descent direction ν and a candidate solution x_0 we can define a function $\phi(t) : \mathbb{R} \to \mathbb{R}$ as follows:

$$\phi(t) = f(x_0 + t\nu). \tag{4.70}$$

The Wolfe conditions propose that a value of t must accomplishes a 'sufficient decrease condition'. The decrease condition can be defined as follows:

$$\phi(t) \leq \phi(0) + c_1 t\phi'(0), \tag{4.71}$$

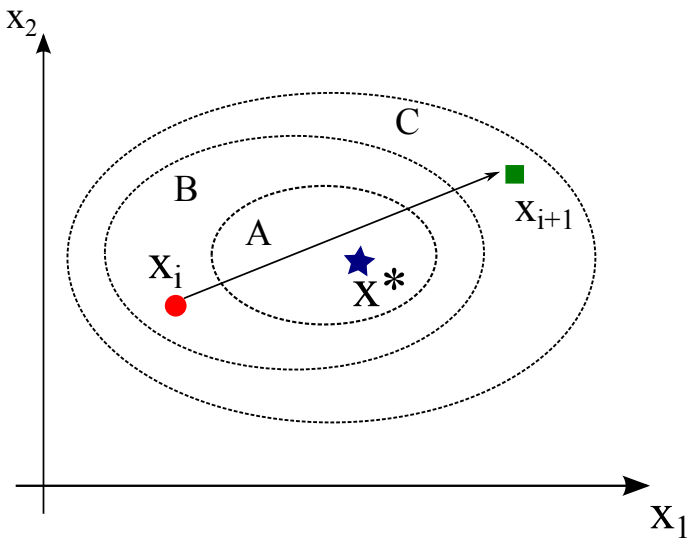


Figure 4.4: Example of large step size

where $c_1 = 10^{-4}$ and $\phi'(0) = \nabla f(x_0)^T \nu$. Now, let's consider the problem presented in Figure 4.4 the original step size t_0 lead us to increase the function value, which clearly violates the Wolfe conditions, if we assume that it is possible to plot the function of (4.78) for such problem. Then, we possibly obtain a plot similar to Figure 4.5.

Using the three pieces of given information ($\phi(0) = f(x_0)$, $\phi(t_0) = f(x_0 + t_0\nu)$ and $\phi'(0)$ it is possible to construct a quadratic approximation of function $\phi(t)$. Such approximation is given by:

$$\phi_q(t) = \frac{\phi(t_0) - \phi(0) - t_0\phi'(0)}{t_0^2} t^2 + \phi'(0)t + \phi(0). \tag{4.72}$$

The step size that gives the minimal of the quadratic approximation of Equation (4.72) is given by:

$$t_1 = \frac{\phi'(0)t_0^2}{(\phi(t_0) - \phi(0) - t_0\phi'(0))}. \tag{4.73}$$

Unfortunately, the new step size described in Equation (4.73) requires the gradient information. In this case, we can use the computations of the GSA to approximate the value of $\phi'(0)$. Considering the gradient approximation given by the GSA, a step

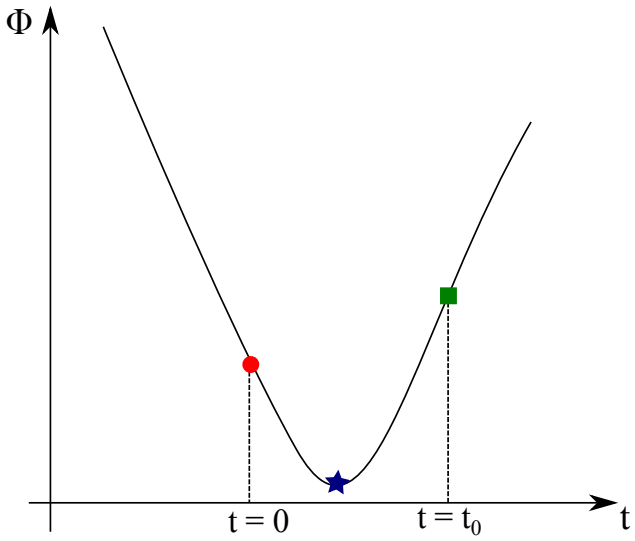


Figure 4.5: Example of a function $\phi(t)$

size $\tilde{t}_1$ using such approximation can be obtained as:

$$\tilde{t}_1 = \frac{t_0^2\,(\tilde{V}\tilde{\lambda})^T\nu}{\Big(\phi(t_0) - \phi(0) - t_0\,(\tilde{V}\tilde{\lambda})^T\nu\Big)}. \tag{4.74}$$

If the value $\tilde{t}_1$ does not accomplish the Wolfe conditions we can incorporate the new information to the system an compute a cubic approximation of the function $\phi(t)$. Such approximation is given by:

$$\phi_c(t) = a\,t^3 + b\,t^2 + (\tilde{V}\tilde{\lambda})^T\nu\,t + \phi(0), \tag{4.75}$$

where a and b are given by:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{t_0^2\tilde{t_1}^2(\tilde{t_1} - t_0)} \begin{pmatrix} t_0^2 & -\tilde{t_1}^2 \\ -t_0^3 & \tilde{t_1}^3 \end{pmatrix} \begin{pmatrix} \phi(\tilde{t_1}) - \phi(0) - (\tilde{V}\tilde{\lambda})^T\nu\tilde{t_1} \\ \phi(t_0) - \phi(0) - (\tilde{V}\tilde{\lambda})^T\nu t_0 \end{pmatrix}. \tag{4.76}$$

The minimum of Equation (4.75) is defined as:

$$t_2 = \frac{-b + \sqrt{b^2 - 3a(\tilde{V}\tilde{\lambda})^T\nu}}{3a} \tag{4.77}$$

In GSA formulations we propose different realizations to compute directions: one for the unconstrained case and two for the constrained case. Unfortunately, some

modifications to the step size are required in order to handle constrained problems. If we consider a constrained SOP, we can apply the penalty function proposed in Equation (4.66) to compute the step size control. Incorporating the penalty function we obtain a function $\phi_P(t) : \mathbb{R} \to \mathbb{R}$ defined as follows:

$$\phi_P(t) = f(x + t\nu) + C_p \sum_{i=1}^{l} \max(0, g_i(x + t\nu)) + C_p \sum_{j=1}^{p} h_j(x + t\nu)^2, \tag{4.78}$$

where C_p, g and h are defined as in Equation (4.64).

Using GSA formulations to compute directions for non feasible solutions we are in position to modify the quadratic approximation used in step size control. Moreover, the minimum of such approximation can be obtained by applying (4.78) into the formulation of t_1:

$$\bar{t_1} = \frac{t_0^2 (\tilde{V}\hat{\lambda})^T \nu}{\left(\phi(t_0) - \phi(0) - t_0 (\tilde{V}\hat{\lambda})^T \nu\right)}. \tag{4.79}$$

Applying a similar substitution into the coefficient computations for cubic approximation given in Equation (4.75) they can be estimated as:

$$\begin{pmatrix} \bar{a} \\ \bar{b} \end{pmatrix} = \frac{1}{t_0^2 \bar{t_1}^2 (\bar{t_1} - t_0)} \begin{pmatrix} t_0^2 & -\bar{t_1}^2 \\ -t_0^3 & \bar{t_1}^3 \end{pmatrix} \begin{pmatrix} \phi(\bar{t_1}) - \phi(0) - (\tilde{V}\hat{\lambda})^T \nu \bar{t_1} \\ \phi(t_0) - \phi(0) - (\tilde{V}\hat{\lambda})^T \nu t_0 \end{pmatrix}, \tag{4.80}$$

hence the step size that gives the minimum for the approximation can be obtained as:

$$\bar{t_2} = \frac{-\bar{b} + \sqrt{\bar{b}^2 - 3\bar{a}(\tilde{V}\hat{\lambda})^T \nu}}{3\bar{a}}. \tag{4.81}$$

Finally, Algorithm 17 presents the pseudocode for the computation of the step size. The algorithm computes a step size that accomplishes the Wolfe conditions and returns it.

Algorithm 17 Pseudocode of the step size control for GSA

Require: Candidate solution x_0, Direction ν, Initial step size t_0, Number of inequality constraints m, Number of equality constraints p, Neighbor matrix $\tilde{V}$
Ensure: Optimal step size t_{new}

if $p > 0$ or $m > 0$ **then**
  Compute $\hat{\lambda}$ from Equation (4.67)
  if $\phi(t_0) \leq \phi(0) + c_1\, t_0\, (\tilde{V}\hat{\lambda})^T \nu$ **then**
    Set $t_{new} := t_0$
  else
    Compute $\bar{t}_1$ from Equation (4.79)
    if $\phi(\bar{t}_1) \leq \phi(0) + c_1\, \bar{t}_1\, (\tilde{V}\hat{\lambda})^T \nu$ **then**
      $t_{new} := \bar{t}_1$
    else
      Compute $\bar{t}_2$ from Equation (4.81)
      Set $t_{new} := \bar{t}_2$
    end if
  end if
else
  Compute $\tilde{\lambda}$ from Equation (4.25)
  if $\phi(t_0) \leq \phi(0) + c_1\, t_0\, (\tilde{V}\tilde{\lambda})^T \nu$ **then**
    Set $t_{new} := t_0$
  else
    Compute t_1 from Equation (4.73)
    if $\phi(t_1) \leq \phi(0) + c_1\, t_1\, (\tilde{V}\tilde{\lambda})^T \nu$ **then**
      $t_{new} := t_1$
    else
      Compute t_2 from Equation (4.74)
      $t_{new} := t_2$
    end if
  end if
end if
return t_{new}

4.4.3 GSA standalone algorithm

After we perform one iteration of the GSA method it is possible that a feasible solution becomes an infeasible one. So, in order to correct such a solution we need a procedure to return the solution to the feasible region. Consider that if the solution is in the infeasible region we need to minimize the value of the constraints. Hence, in order to correct our solution we propose to use the penalty function described in Equation (4.64) to minimize such constraint violation. But one modification is re-

quired on such equation: we change the value of $c_1 = 10^2$. Next, we construct a GSA direction using the penalty function following the procedure described in Section 4.4.1.

As mentioned above the computation of a neighborhood is an important step to compute any GSA direction. In particular, the standalone version of the GSA requires that such neighborhood is computed since we do not have any population to obtain such information. To compute the neighborhood we propose to construct orthogonal neighbors by performing the procedure of Algorithm (18). In the algorithm, the value of $\epsilon_{\mathcal{N}}$ is set as $1e-3$ and q_i represents the i-th column of matrix Q.

Algorithm 18 Pseudocode of the neighborhood structure of the GSA standalone algorithm.

Require: Initial point $x_{(0)}$, Neighborhood size r
Ensure: Neighborhood structure $\mathcal{N}$
1: Set $\mathcal{N} = \emptyset$
2: Compute a random solution $x_r \in \mathbb{R}^n$
3: Compute the decomposition $QR = x_r$
4: **for** $i = 1, \ldots, r$ **do**
5: Set $x_a = x_0 + \epsilon_{\mathcal{N}}\, q_i$
6: Set $\mathcal{N} \rightarrow x_i := x_a$
7: Set $\mathcal{N} \rightarrow f_i = f(x_a)$
8: Set $\mathcal{N} \rightarrow g_i = g(x_a)$
9: Set $\mathcal{N} \rightarrow h_i = h(x_a)$
10: **end for**

Finally, with all the considerations described above we are in the position to define the standalone algorithm for the GSA method. Algorithm 19 presents the pseudocode of such a procedure.

Algorithm 19 Pseudocode of the standalone GSA for SOP.

Require: Initial point $x_{(0)}$, Initial step size t_0, GSA neighborhood size r
Ensure: Sequence $x_{(k)}$ of candidate solutions, Step size t

Compute $f_0 = f(x_0)$
Compute $g_0 = g(x_0)$
Compute $h_0 = h(x_0)$
Set $k := 1$
Set $x_a = x_0$
while Stopping condition is not met **do**
  Compute $\mathcal{N}$ using Algorithm 18
  if $|g_0| > 0$ or $|h_0| > 0$ **then**
    if x_a is feasible **then**
      Compute ν using Algorithm 13
    else
      Compute ν using Algorithm 16
    end if
  else
    Compute ν using Algorithm 12
  end if
  Compute step size 17
  Set $x_{new} := x_a + t\,\nu$
  if x_{new} is infeasible **then**
    while x_{new} is infeasible **do**
      Compute ν using Algorithm 13
      Set $x_{new} := x_{new} + t\,\nu$
    end while
    Set $x_a := x_{new}$
    Set $x_k := x_{new}$
  end if
  Set $k := k + 1$
end while

4.5 GSA within DE

The next step for GSA is to define the algorithms to construct the memetic strategy. In particular, we are interested in constructing a memetic algorithm where we propose to use the Differential Evolution [Storn and Price, 1995] as a base algorithm. In this section we define the necessary mechanisms to compute the memetic strategy: we called it DE/GSA.

4.5.1 Computing the neighborhood

As done with the standalone algorithm, it is necessary to compute the neighboring information used by the GSA. Consider a population P^i, where such population is observed at the i-th generation. Also, let's consider that we can obtain the following information from such population:

- $P^i \rightarrow x^j$ represents the parameter vector for the j-th individual in the population.
- $P^i \rightarrow f^j$ represents the function value for the j-th individual in the population.
- $P^i \rightarrow h^j$ represents the equality constraints vector for the j-th individual in the population.
- $P^i \rightarrow g^j$ represents the inequality constraints vector for the j-th individual in the population.

Consider that we have a candidate solution x_0 where GSA needs to be applied. The first step to compute the neighborhood $\mathcal{N}$ is to remove all the possible copies of x_0 that exist in P^i. To perform such procedure we consider that for copies of the candidate solution it holds that $\|x_0 - P^i \rightarrow x^j\|_2 = 0$, $j = 1, \ldots |P^i|$. Using such property we can define a new population $\bar{P}^i$ as follows:

$$\bar{P}^i = \left\{ \bar{p} | \bar{p} \in P^i \cup \|x_0 - \bar{p} \rightarrow x^j\|_2 > 0 \right\}. \tag{4.82}$$

Now, let's assume that we sort the population $\bar{P}^i$ such that for each element in $\bar{p} \in \bar{P}^i$ it holds that $\|x_0 - \bar{p_1}\|_2 \leq \ldots \leq \|x_0 - \bar{p_{\mathcal{M}}}\|_2$, where $\mathcal{M} = |\bar{P}^i|$. Using the sorted population we can construct our neighborhood as described in Algorithm 20. In the algorithm we use the parameter τ that defines the neighborhood size (do not confuse such parameter with the parameter r of the GSA).

Algorithm 20 Pseudocode of the neighborhood structure for DE/GSA.

Require: Neighborhood size τ, Sorted population $\bar{P}^i$
Ensure: Neighborhood structure $\mathcal{N}$
1: Set $\mathcal{N} = \emptyset$
2: Set $\mathcal{M} = |\bar{P}^i|$
3: **for** $j = 1, \ldots, \mathcal{M}$ **do**
4: Set $\mathcal{N} \rightarrow x_j = \bar{P}^i \rightarrow x^j$
5: Set $\mathcal{N} \rightarrow f_j = \bar{P}^i \rightarrow f^j$
6: Set $\mathcal{N} \rightarrow g_j = \bar{P}^i \rightarrow g^j$
7: Set $\mathcal{N} \rightarrow h_j = \bar{P}^i \rightarrow h^j$
8: **if** $i \geq \tau$ **then**
9: **break**
10: **end if**
11: **end for**
12: **return** $\mathcal{N}$

4.5.2 Initial step size

One of the challenges for the effective realization of line search techniques is the proper computation of the initial step size. A large value of the initial step size can lead us to increase the cost of the technique (the correction step will require more function calls). But a small step size can lead the algorithm to meaningless improvements, e.g., the Wolfe conditions are not accomplished. In this section, we describe a mechanism to tackle this problem combining the information of previous iterations in the line search and the information of the population.

Consider a given individual in population P. From such individual we can obtain a value $P \rightarrow t^j$ (j represents that the j individual is used). The parameter t represents the step size for such individual. The value of such parameter has been previously saved for the individual. That is, assume that at the $i - th$ generation we computed the step size using Algorithm 17. After performing the line search we can set $P \rightarrow t^j := t$, where t is the step size obtained from Algorithm 17. Hence, for any generation $k > i$ we already have a value stored for $P \rightarrow t^j$.

Unfortunately, there exists one case that has to be considered: the initialization of the value $P \rightarrow t^j$. To initialize all the values of t we propose the following approach. Consider that we have an individual $\hat{p} \in P$, for such individual we computed its neighboring information using Algorithm 20. Given a candidate solution $x_0 := \hat{p} \rightarrow x$ and its $\mathcal{M}$ neighbors we can compute a matrix $D \in \mathbb{R}^{n \times \mathcal{M}}$ as follows:

$$D = \left(\ |x_0 - x_1|, \ldots, |x_0 - x_{\mathcal{M}}| \ \right), \tag{4.83}$$

where $|\cdot| : \mathbb{R}^n \to \mathbb{R}^n$ computes the absolute value of each given vector and $x_j = \mathcal{N} \to x_j$, $j = 1, \ldots, \mathcal{M}$. Hence, an initial step size can be computed using Algorithm 21.

Algorithm 21 Pseudocode for computing the initial step size of DE/GSA.

Require: Neighborhood matrix D
Ensure: Step size t
1: Set $t := \infty$
2: **for** $i = 1, \ldots, n$ **do**
3: Set $t_a := 0 \in \mathbb{R}$
4: **for** $j = 1, \ldots, \mathcal{M}$ **do**
5: Set $t_a := t_a + D_{ij}$
6: **end for**
7: Set $t_a := \frac{t_a}{\mathcal{M}}$
8: **if** $t_a < t$ **then**
9: $t := t_a$
10: **end if**
11: **end for**

Algorithm 21 defines an initialization for the step size. However, the same algorithm can be used to reset the step size at some point. For example, consider that for some reason the memetic algorithm archived a local optimal solution. In this case, the step size stored in the population has decreased. Unfortunately, the decrement of the stored step size presents a problem since the line search improvement is directly related to such parameter. To avoid such a problem we introduce a parameter β that randomly re-initializes a step size according to Algorithm 21. Finally, using all the mechanisms described above we are in position to introduce Algorithm 22. This algorithm computes the step size at the i-th generation for the j-th individual in the population.

4.5.3 Balancing the operators

One of the main problems that arise when using a memetic algorithm is to implement a balance between the local search and the evolutionary operators. In particular, for the DE/GSA we propose to apply the procedure given in [LaTorre, 2009, LaTorre

Algorithm 22 Pseudocode for computing the DE/GSA step size.

Require: Population P^i, Individual index j , Neighborhood size τ, β
Ensure: Step size t
1: Set $x_0 := P^i \rightarrow x^j$
2: Set $t := P^i \rightarrow t^j$
3: Compute a random value $q_r \in [0, 1]$
4: **if** $t = \emptyset$ or $q_r \leq \beta$ **then**
5: Compute x_0 neighborhood $\mathcal{N}$ using Algorithm 20
6: Compute D as in Equation (4.83)
7: Compute t using Algorithm 21
8: **end if**
9: **return** t

et al., 2011]. To implement such a procedure, we assume that the evolutionary operators of DE and the ones from GSA are two separated techniques. The procedure that balances such techniques is called MOS. This procedure measures at each generation the improvement archived for each technique involved in the creation of new candidate solutions. In particular, we propose certain modifications to MOS such that it handles constrained problems.

As we stated above, we consider that the GSA and DE operators are two different components of the same algorithm. In each generation, GSA generates an offspring population named O^{GSA}. Analogously, the individuals created by the DE operators define the offspring population named O^{DE}. Using a similar nomenclature as the one defined for other mechanisms we can define:

- $O^{GSA} \rightarrow x^j$, $O^{DE} \rightarrow x^j$ represents the parameter vectors for the j-th individual in the GSA population and the DE population respectively.
- $O^{GSA} \rightarrow f^j$, $O^{DE} \rightarrow f^j$ represents the function values for the j-th individual in the GSA population and the DE population respectively.
- $O^{GSA} \rightarrow h^j$, $O^{DE} \rightarrow h^j$ represents the equality constraints vectors for the j-th individual in the GSA population and the DE population respectively.
- $O^{GSA} \rightarrow g^j$, $O^{DE} \rightarrow g^j$ represents the inequality constraints vectors for the j-th individual in the GSA population and the DE population respectively.

The MOS algorithm uses a quality term to measure the improvement given by each operator. Unfortunately, such a procedure was only developed for unconstrained

problems. Hence, we propose to use Equation (4.64) to modify the quality term proposed in MOS. Consider that we have an individual at the i-th generation from any offspring population O^* (the $*$ represents that it could be either O^{GSA} or O^{DE}). Moreover, we can obtain several values from such individual: $x^i := O^* \rightarrow x^j$, $f^i := O^* \rightarrow f^j$, $h^i := O^* \rightarrow h^j$ and $g^i := O^* \rightarrow g^j$. Right after we apply the operators we can obtain the values for the new individual: x^{i+1}, f^{i+1}, h^{i+1} and g^{i+1}. Using all these values we can defined the quality term of the j-th individual as follows:

$$\text{quality}(x^i, f^i, h^i, g^i, x^{i+1}, f^{i+1}, h^{i+1}, g^{i+1}) = \left\| \frac{\mathcal{P}(x^{i+1}, f^{i+1}, h^{i+1}, g^{i+1}) - \mathcal{P}(x^i, f^i, h^i, g^i)}{\mathcal{P}(x^{i+1}, f^{i+1}, h^{i+1}, g^{i+1})} \right\|. \quad (4.84)$$

Define two vectors $q^{GSA} \in \mathbb{R}^{|O^{GSA}|}$ and $q^{DE} \in \mathbb{R}^{|O^{DE}|}$ where each entry of such vectors can be computed using Equation (4.84). After the offspring populations have been created we also obtained the vectors q^{GSA} and q^{DE}. Using vector q^{GSA} we compute the quality of the population created by the GSA method as follows:

$$Q^{GSA} = \frac{\sum_{i=1}^{|q^{GSA}|} q_i^{GSA}}{|q^{GSA}|}. \quad (4.85)$$

Analogously, we can compute the quality for the DE method:

$$Q^{DE} = \frac{\sum_{i=1}^{|q^{DE}|} q_i^{DE}}{|q^{DE}|}. \quad (4.86)$$

Once the quality of the offspring of each method is computed we proceed to compute the so called participation ratios. Algorithm 23 presents the mechanism to compute such participation ratios for each technique ($\mathcal{R}^{GSA}$ and $\mathcal{R}^{DE}$). The participation ratios define the percentage of resources to be used for each technique. Here, we use two parameters to control the computation of the participation ratios: $\mathcal{R}_c$ and $\mathcal{R}_-$. $\mathcal{R}_c$ controls the velocity of adaptation and $\mathcal{R}_-$ defines a lower bound limit for the participation ratios.

Algorithm 23 Pseudocode of the participation ratio calculation.

Require: Initial participation ratios $\mathcal{R}^{GSA}$, $\mathcal{R}^{DE}$, Technique qualities Q^{GSA}, Q^{DE}, $\mathcal{R}_c$, $\mathcal{R}_-$
Ensure: Participation ratios $\bar{\mathcal{R}}^{GSA}$, $\bar{\mathcal{R}}^{DE}$
1: **if** $Q^{GSA} > Q^{DE}$ **then**
2: $r_a := \mathcal{R}_c \left(\frac{Q^{GSA} - Q^{DE}}{Q^{GSA}} \right) \mathcal{R}^{GSA}$.
3: **if** $(\mathcal{R}^{DE} - r_a) < \mathcal{R}_-$ **then**
4: $r_a := \mathcal{R}^{DE} - \mathcal{R}_-$.
5: **end if**
6: Set $\bar{\mathcal{R}}^{GSA} := \mathcal{R}^{GSA} + r_a$.
7: Set $\bar{\mathcal{R}}^{DE} := \mathcal{R}^{DE} - r_a$.
8: **else if** $Q^{DE} > Q^{GSA}$ **then**
9: $r_a := \mathcal{R}_c \left(\frac{Q^{DE} - Q^{GSA}}{Q^{DE}} \right) \mathcal{R}^{DE}$.
10: **if** $(\mathcal{R}^{GSA} - r_a) < \mathcal{R}_-$ **then**
11: $r_a := \mathcal{R}^{GSA} - \mathcal{R}_-$.
12: **end if**
13: Set $\bar{\mathcal{R}}^{DE} := \mathcal{R}^{DE} + r_a$.
14: Set $\bar{\mathcal{R}}^{GSA} := \mathcal{R}^{GSA} - r_a$.
15: **else**
16: Set $\bar{\mathcal{R}}^{GSA} := \mathcal{R}^{GSA}$.
17: Set $\bar{\mathcal{R}}^{DE} := \mathcal{R}^{DE}$.
18: **end if**

Finally, Algorithm 24 presents the realization of the memetic GSA. At the beginning of the algorithm the participation ratios of both algorithms are set with the same probability. Besides, we propose to use values $\mathcal{R}_- = 0.1$ and $\mathcal{R}_c = 0.05$. We have to take into consideration that in some cases it is possible that some individuals are not taking into consideration to create an offspring. That is, since we are rounding off the offspring population size it is possible that in some cases $N \neq |O^{GSA}| + |O^{DE}|$. In such case, we split the difference $d_N := N - (|O^{GSA}| + |O^{DE}|)$ using $\lfloor d_N \rfloor$ and $\lceil d_N \rceil$.

Algorithm 24 Pseudocode of the memetic DE/GSA.

Randomly create initial population P_0.
Set $\mathcal{R}^{DE} := \mathcal{R}^{GSA} := 0.5$.
Set $i := 0$.
while Stopping criteria is not met **do**
  Set $N^{GSA} := \lfloor N\,\mathcal{R}^{GSA} \rfloor$
  Set $N^{DE} := \lfloor N\,\mathcal{R}^{DE} \rfloor$
  Compute $d_N := N - (N^{GSA} + N^{DE})$
  if $d_N > 0$ **then**
    Set $N^{GSA} := N^{GSA} + \lfloor d_N \rfloor$
    Set $N^{DE} := N^{DE} + \lceil d_N \rceil$
  end if
  Randomly select N^{DE} individuals from P_i to create O^{DE}
  Randomly select N^{GSA} individuals from P_i to create O^{GSA}
  Set $q^{DE} := 0 \in \mathbb{R}^{N^{DE}}$
  for $j = 1, \ldots, N^{DE}$ **do**
    Compute crossover and mutation on $O^{DE} \rightarrow x^j$
    Compute q_j^{DE} using Equation (4.84)
  end for
  Compute Q^{DE} using Equation (4.86)
  Set $q^{GSA} := 0 \in \mathbb{R}^{N^{GSA}}$
  for $j = 1, \ldots, N^{GSA}$ **do**
    Compute neighborhood of $O^{GSA} \rightarrow x^j$
    Compute initial step size t_0 using Algorithm 21
    Compute new candidate solution x^{j+1} and corrected step size t using Algorithm 19
    Store the step size t on P
    Compute q_j^{GSA} using Equation (4.84)
  end for
  Compute Q^{GSA} using Equation (4.85)
  Update participation ratios $\mathcal{R}^{DE}$, $\mathcal{R}^{GSA}$ using Algorithm 23
  Set $P_{i+1} := O^{DE} \cup O^{GSA}$
  Set $i := i + 1$
end while

4.6 Numerical Results

In this section, we present a set of experiments to illustrate the performance of the GSA method. We performed several experiments using the standalone and the memetic version of the GSA. For the standalone algorithm, we performed a comparison between GSA, Pattern Search [Hooke and Jeeves, 1961], and Nelder-Mead

algorithms [Nelder and Mead, 1965]. To compare the memetic strategy, we used the set of functions proposed for the CEC'06 contest on constrained optimization [Liang et al., 2006] (for reference see Appendix A).

4.6.1 Standalone algorithm

We start our experiments for standalone GSA using two academic problems defined by Equations (4.87) and (4.88).

$$\begin{array}{ll} \min & \sum_{i=1}^{2} x_i^2 \\ \text{s.t.} & -11 \leq x_1 \leq 7 \\ & 2 \leq x_2 \leq 10 \end{array} . \tag{4.87}$$

$$\begin{array}{ll} \min & \sum_{i=1}^{2} x_i^2 \\ \text{s.t.} & x_1 + x_2 \geq 1 \end{array} . \tag{4.88}$$

To perform an experiment we randomly generate an initial solution x_0 and we apply the three different methods: GSA, Pattern Search and Nelder-Mead. For the GSA method we use Algorithm 19 to compute the candidate solutions. This algorithm was implemented using MATLAB [1]. In this experiment we set the parameter $r = 2$. We perform a comparison using the MATLAB versions of the pattern search and the Nelder-Mead algorithm. Since there is no constrained version of the Nelder-Mead algorithm in MATLAB, we propose to use the unconstrained version along with the penalty function defined in Equation (4.64).

We performed $1,000$ independent runs for each algorithm and we averaged the number of function calls required for each algorithm. We previously computed the optimal solution on each problem x^*. The stopping criteria for each alggorithm is defined using the error term:

$$\epsilon = \|x^* - x_i\| \leq 5e - 3, \tag{4.89}$$

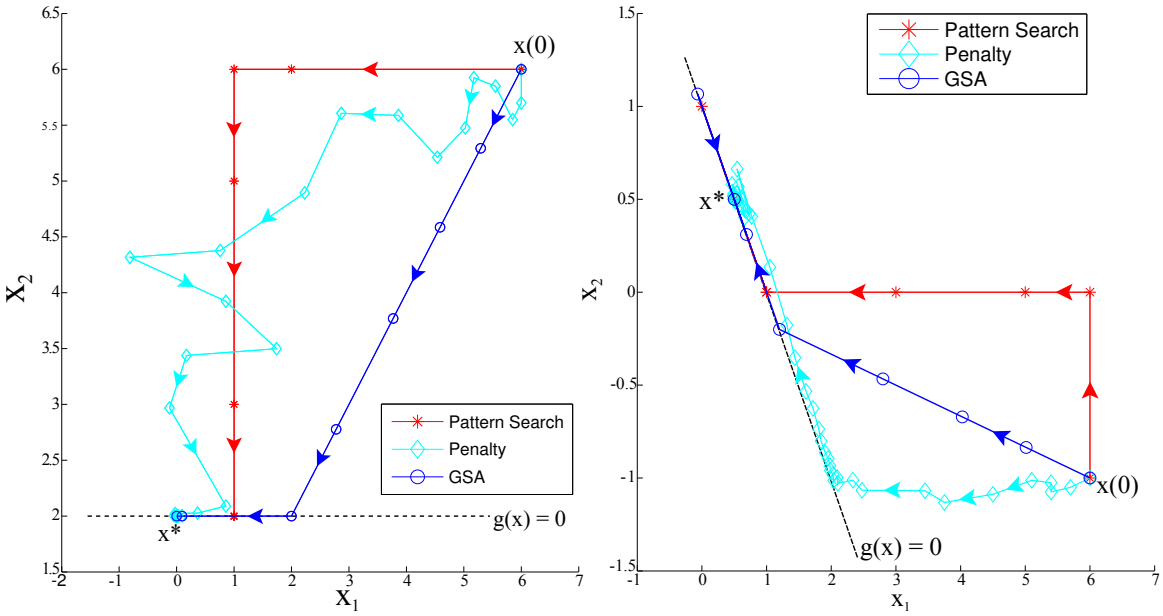
where x_i is the candidate solution at the i-th iteration. Table 4.1 present the averaged results for the proposed algorithms. As we observed the GSA saves a considerable amount of function calls in comparison with the other two algorithms.

Figure 4.6 presents a single run to illustrate the results obtained for the three

[1] MATLAB©is a software property of MathWorks®

Table 4.1: Results for standalone GSA on the 2-dimensional test problems

Problem	**Algorithm**	**Function calls**
(4.87)	Pattern Search	263.7
	Nelder Mead with Penalty Function	147.5
	GSA	**35.3**
(4.88)	Pattern Search	302.4
	Nelder Mead with Penalty Function	161.7
	GSA	**38.4**



(a) Equation (4.87) (b) Equation (4.88)

Figure 4.6: Single run for academic problems experimentation

algorithms. From the figure it is clear that GSA saves a considerable amount of resources by reaching the optimal value x^* in less steps than its competitors.

To extend the previous experiments we used a set of test problems that have a higher number of parameters. The test functions are defined in Equations (4.90) to (4.94).

$$\min \sum_{i=0}^{10} x_i^2$$

$$\begin{aligned} s.t. -1 \leq x_1 \leq 7 \\ 2 \leq x_2 \leq 10 \\ 1 \leq x_3 \leq 9 \end{aligned} \tag{4.90}$$

$$\min \sum_{i=0}^{10} x_i^2 \tag{4.91}$$
$$s.t. x_1 + x_2 + x_3 \geq 1$$

$$\min \sum_{i=0}^{10} x_i^2 \tag{4.92}$$
$$s.t. \sum_{i=1}^{10} x_i \geq 1$$

$$\min \sum_{i=0}^{10} x_i^2 \tag{4.93}$$
$$s.t. \sum_{i=1}^{10} x_i = 1$$

$$\min \sum_{i=0}^{10} x_i^2 \tag{4.94}$$
$$s.t. (x_1 + 1)^2 + \sum_{i=2}^{10} x_i^2 = 4 \quad .$$

Table 4.2 shows the results obtained by the algorithms using the equations described above. The values are averaged over 1,000 independent runs coming from different feasible starting points. The setup parameters for this experiment are the same that in the previous one except that we modify the value $r = 5$. As expected, the GSA standalone algorithm required about one order of magnitude less function evaluations to reach its goal. We observe that in some cases some of the algorithms do not archive the goal. So we set a maximal number of function calls for each algorithm at $10,000$. In case that an algorithm does not reach the expected error ϵ the run is considered invalid and is denoted by a "$-$" symbol.

Table 4.2: Results for the second experiment using GSA as a standalone algorithm.

Problem	**Algorithm**	**Function calls**
(4.90)	Pattern Search	4218.9
	Nelder Mead with Penalty Function	1285.9
	GSA	**377.7**
(4.91)	Pattern Search	3461.5
	Nelder Mead with Penalty Function	1558.3
	GSA	**339.8**
(4.92)	Pattern Search	4015.8
	Nelder Mead with Penalty Function	-
	GSA	**310.3**
(4.93)	Pattern Search	3818.3
	Nelder Mead with Penalty Function	-
	GSA	**271.0**
(4.94)	Pattern Search	-
	Nelder Mead with Penalty Function	-
	GSA	**1277.8**

4.6.2 GSA within DE

The next step for our experiments is to compare the memetic strategy that couples the DE and GSA methods as a search technique. Such experiments were performed using the set of functions proposed for the competition at the 2006 IEEE Congress on Evolutionary Computation [Liang et al., 2006].

We performed a comparison between DE (rand/1/bin version) as a standalone algorithm, and the GSA/DE method and we defined a second memetic algorithm that uses the LS1 method [Tseng and Chen, 2008] as a local search engine. The results are presented using two sets of experiments.

To handle constraints we considered as our base algorithm the variant of DE proposed in [Kukkonen and Lampinen, 2006]. The parameters for the DE algorithm are defined as recommended by the authors. Specifically, NP, F and CR were set to 100, 0.8 and 0.95, respectively.

First Set of Experiments

In this first set of experiments we aim to analyze the convergence of each algorithm at different stages of the run. The first stage is measured when the algorithm reached $5,000$ function evaluations. We call it as the short term stage. The second stage measures the algorithms at $50,000$ function calls. This is called the medium term stage. Finally, the algorithms are compared in a long term stage, i.e., after $500,000$ function evaluations.

The algorithm's performance is measured using a set of statistical comparisons. For each problem, the experiment was repeated 30 times. Similar guidelines to the one applied in [Durillo et al., 2010] were considered. Specifically, the following tests were applied, assuming a significance level of 5%. First, a *Shapiro-Wilk test* was performed to check whether or not the values of the results followed a Gaussian distribution. If so, the *Levene test* was used to check for the homogeneity of the variances. If samples had equal variance, an ANOVA *test* was done; if not, a *Welch test* was performed.

Tables 4.3, 4.4 and 4.5 present several statistical values computed using the error

Table 4.3: Solutions obtained by the different DE variants at 5,000 evaluations.

Problem	Median			Average		
	DE	**DE/GSA**	**DE/LS1**	**DE**	**DE/GSA**	**DE/LS1**
$g1$	**4.88e+00**	5.39e+00	5.55e+00	**4.76e+00**	5.15e+00	5.62e+00
$g2$	4.81e-01	4.92e-01	**3.48e-01**	4.74e-01	4.87e-01	**3.49e-01**
$g3$	1.00e+00	**8.96e-01**	1.00e+00	9.92e-01	**8.80e-01**	9.93e-01
$g4$	6.90e+01	**4.41e+01**	1.25e+02	7.46e+01	**4.54e+01**	1.19e+02
$g5$	8.87e+01	6.73e-02	**1.82e+02**	2.30e+02	1.30e+00	**3.11e+02**
$g6$	2.16e+02	**3.75e+01**	3.62e+02	2.12e+02	**5.11e+01**	3.67e+02
$g7$	1.53e+02	2.00e+02	1.66e+02	1.73e+02	2.22e+02	1.82e+02
$g8$	2.82e-08	**7.70e-12**	6.83e-07	7.22e-08	**3.55e-11**	1.85e-06
$g9$	1.55e+02	1.59e+02	**6.93e+01**	1.55e+02	1.70e+02	**6.90e+01**
$g10$	4.74e+03	4.71e+03	**4.64e+03**	4.75e+03	4.72e+03	**4.61e+03**
$g11$	1.04e-01	**1.30e-03**	6.60e-02	9.48e-02	**1.94e-02**	9.86e-02
$g13$	9.25e-01	8.79e-01	8.01e-01	1.25e+00	1.02e+00	7.10e-01
$g14$	5.50e+00	3.60e+00	5.08e+00	5.76e+00	3.87e+00	5.92e+00
$g15$	1.76e+00	**4.73e-04**	1.65e+00	1.94e+00	**2.03e-03**	2.26e+00
$g16$	**1.11e-01**	1.59e-01	1.64e-01	**1.21e-01**	1.65e-01	1.72e-01
$g17$	2.45e+02	**8.90e+01**	2.39e+02	2.62e+02	**1.05e+02**	2.83e+02
$g18$	8.40e-01	6.88e-01	6.04e-01	9.85e-01	7.81e-01	6.87e-01
$g19$	4.30e+02	2.51e+02	5.24e+02	4.17e+02	2.62e+02	5.12e+02
$g20$	6.95e+00	8.10e+00	3.42e+00	7.03e+00	8.09e+00	3.65e+00
$g21$	6.37e+02	5.85e+02	5.13e+02	5.93e+02	5.54e+02	4.96e+02
$g22$	1.25e+04	9.28e+03	8.12e+03	9.72e+03	1.07e+04	8.94e+03
$g23$	4.00e+02	4.55e+02	**2.94e+02**	**3.92e+02**	4.53e+02	4.04e+02
$g24$	2.01e-03	**1.49e-04**	6.61e-03	2.41e-03	**2.43e-04**	6.98e-03

terms. Such error term can be computed as follows:

$$\epsilon_f := f(x_{best}) - f(x^*), \tag{4.95}$$

where x_{best} represents the best individual at the generation and x^* is the best known solution in the state-of-the art. The tables present the results for the short term, middle term and long term respectively. On such tables, the blue values represent the algorithm with the best individual. The quantities are only denoted if they accomplish all the statistical test described above. In those cases where the differences were not statistically significant, the solution is presented in a regular format.

Unfortunately, presenting only the error terms for constrained problems is not enough information to compare the results. The second piece of information that has

Table 4.4: Solutions obtained by the different DE variants at 50,000 evaluations.

Problem	Median			Average		
	DE	**DE/GSA**	**DE/LS1**	**DE**	**DE/GSA**	**DE/LS1**
$g1$	3.57e-03	**6.59e-04**	2.14e-02	3.90e-03	**6.73e-04**	2.29e-02
$g2$	3.16e-01	3.24e-01	**1.71e-01**	3.13e-01	3.28e-01	**1.69e-01**
$g3$	9.81e-01	**6.46e-01**	8.16e-01	9.52e-01	**6.05e-01**	8.00e-01
$g4$	1.01e-08	**2.71e-10**	1.16e-07	1.36e-08	**4.39e-10**	2.06e-07
$g5$	8.78e+01	**8.67e-04**	1.32e+02	1.24e+02	**1.54e-01**	1.73e+02
$g6$	1.64e-11	1.64e-11	1.64e-11	1.64e-11	1.64e-11	1.64e-11
$g7$	2.38e+00	**1.04e-01**	1.59e+00	2.49e+00	**1.29e-01**	1.63e+00
$g8$	4.16e-17	4.16e-17	3.47e-17	3.61e-17	3.56e-17	3.47e-17
$g9$	1.87e-02	**7.90e-04**	2.91e-02	1.82e-02	**9.28e-04**	3.04e-02
$g10$	4.95e+03	4.95e+03	4.95e+03	4.95e+03	4.95e+03	4.95e+03
$g11$	0	0	0	0	0	0
$g13$	9.46e-01	8.55e-01	**2.11e-02**	9.37e-01	7.60e-01	**2.39e-02**
$g14$	4.98e+00	**7.53e-03**	5.22e+00	5.03e+00	**2.40e-02**	5.18e+00
$g15$	1.46e-01	**4.65e-05**	9.97e-01	6.56e-01	**2.09e-04**	1.56e+00
$g16$	**1.87e-06**	5.75e-06	7.76e-06	**2.31e-06**	5.91e-06	7.83e-06
$g17$	9.60e+01	**1.22e+01**	9.64e+01	8.97e+01	**2.50e+01**	1.00e+02
$g18$	7.55e-02	**9.16e-03**	4.17e-02	7.66e-02	**1.06e-02**	4.23e-02
$g19$	1.85e+01	**1.47e+01**	2.47e+01	1.81e+01	**1.41e+01**	2.43e+01
$g20$	9.88e-01	9.67e-01	9.87e-01	9.86e-01	9.74e-01	9.95e-01
$g21$	1.33e+02	1.31e+02	1.49e+02	1.21e+02	1.43e+02	2.12e+02
$g22$	1.15e+04	**5.71e+03**	1.01e+04	1.15e+04	**7.30e+03**	9.95e+03
$g23$	3.95e+02	**1.77e+02**	4.00e+02	4.51e+02	**1.83e+02**	5.09e+02
$g24$	3.29e-14	3.29e-14	3.29e-14	3.29e-14	3.29e-14	3.29e-14

to be considered is concerned about the constraints of the problem. We present such a information on Tables 4.6, 4.7 and 4.8. On each table the average and the median number of violated constraints are presented. On each problem, a column that represents the original number of constraint that the problem is set. The inequality constraints are represented by the letter E. Analogously, the inequality constraints are represented by the letter I.

As an additional tool to understand the constraints we present Figures 4.7 to 4.11. On such plots we present the averaged constraint values where the best solution is infeasible. To present the data in a clever way we only present the plots with at least one violated constraint. To present the constraints in an uniform way we transform

Table 4.5: Solutions obtained by the different DE variants at 500,000 evaluations.

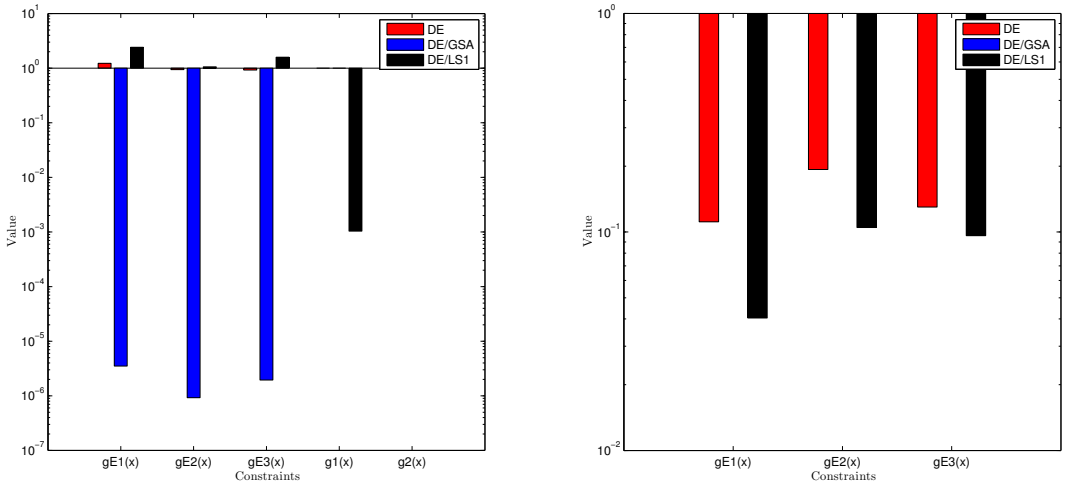
Problem	Median			Average		
	DE	**DE/GSA**	**DE/LS1**	**DE**	**DE/GSA**	**DE/LS1**
$g1$	0	0	0	0	0	0
$g2$	1.38e-02	2.28e-02	**1.38e-01**	1.53e-02	2.82e-02	**1.39e-01**
$g3$	8.44e-01	**4.31e-01**	8.16e-01	8.21e-01	**4.07e-01**	7.97e-01
$g4$	3.64e-12	3.64e-12	3.64e-12	3.64e-12	3.64e-12	3.64e-12
$g5$	1.82e-12	1.82e-12	1.82e-12	7.22e-05	**1.82e-12**	1.06e+00
$g6$	1.64e-11	1.64e-11	1.64e-11	1.64e-11	1.64e-11	1.64e-11
$g7$	2.84e-05	**7.49e-09**	3.15e-02	2.98e-05	**2.39e-08**	3.86e-02
$g8$	2.78e-17	2.78e-17	2.78e-17	2.78e-17	2.78e-17	2.78e-17
$g9$	1.14e-13	1.14e-13	1.14e-13	**1.21e-13**	1.52e-13	1.63e-13
$g10$	4.95e+03	4.95e+03	4.95e+03	4.95e+03	4.95e+03	4.95e+03
$g11$	0	0	0	0	0	0
$g13$	4.07e-01	3.85e-01	**2.11e-02**	4.05e-01	**3.25e-01**	2.39e-02
$g14$	1.51e-07	**7.11e-13**	1.27e-04	1.69e-07	**6.17e-13**	2.13e-04
$g15$	1.14e-13	1.14e-13	1.14e-13	1.14e-13	1.14e-13	1.14e-13
$g16$	3.77e-15	3.77e-15	3.77e-15	3.77e-15	3.77e-15	3.77e-15
$g17$	7.68e+01	**2.99e+00**	8.83e+01	5.37e+01	**1.29e+01**	7.04e+01
$g18$	3.75e-05	8.28e-05	**1.64e-05**	4.28e-05	1.07e-04	**3.17e-05**
$g19$	7.43e+00	**7.42e+00**	1.31e+01	7.44e+00	**7.42e+00**	1.22e+01
$g20$	1.00e+00	1.01e+00	**9.79e-01**	1.00e+00	1.01e+00	**9.77e-01**
$g21$	1.31e+02	**6.55e+01**	1.31e+02	8.73e+01	**6.55e+01**	8.30e+01
$g22$	1.10e+04	6.56e+03	**3.76e+03**	1.04e+04	6.67e+03	**3.44e+03**
$g23$	1.09e+00	5.00e+00	1.26e+02	3.10e+01	1.03e+01	1.08e+02
$g24$	3.29e-14	3.29e-14	3.29e-14	3.29e-14	3.29e-14	3.29e-14

each equality constraint into an inequality constraint such that:

$$g_E = |h(x)| - \epsilon_E, \tag{4.96}$$

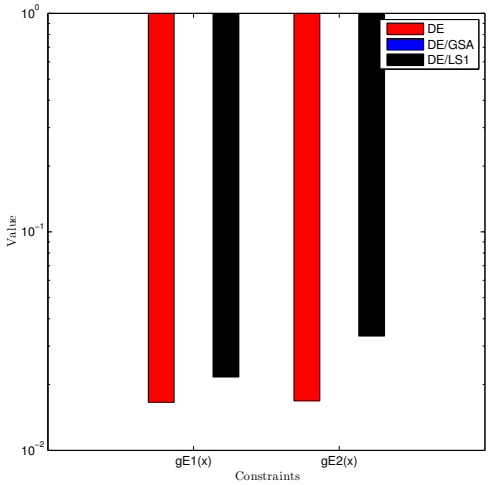
where $\epsilon_E = 1e-4$.

Finally, Figures 4.13 to 4.15 present the converge plots of each algorithm. To construct such plot we use the penalty function described in Equation (4.64). The plots show the averaged penalty value for each of the test functions. In the plots, a dotted line represents the function values of infeasible solutions. Meanwhile, a continuous line represents a feasible solution.

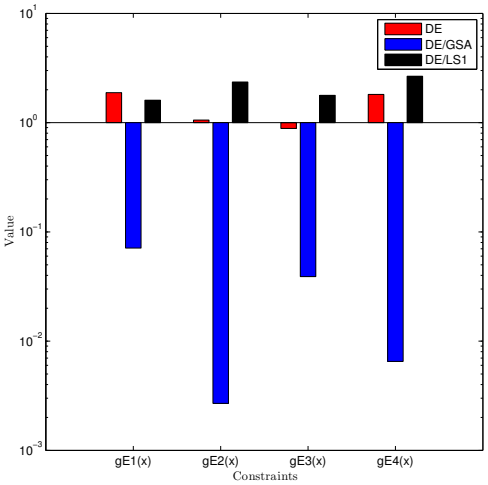


(a) $g5$

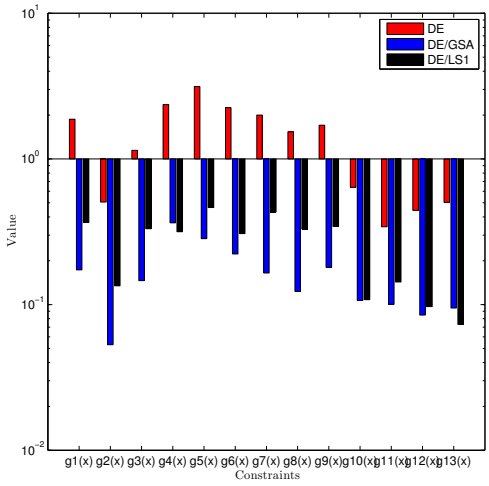
(b) $g14$



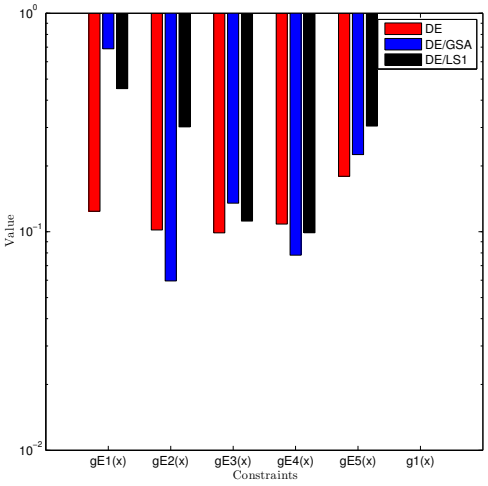
(c) $g15$



(d) $g17$

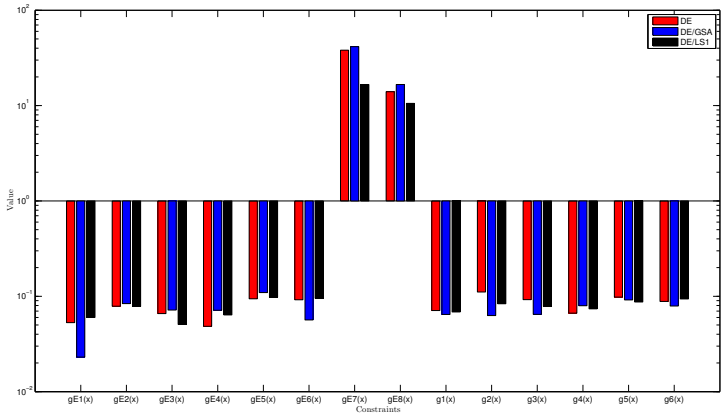


(e) $g18$

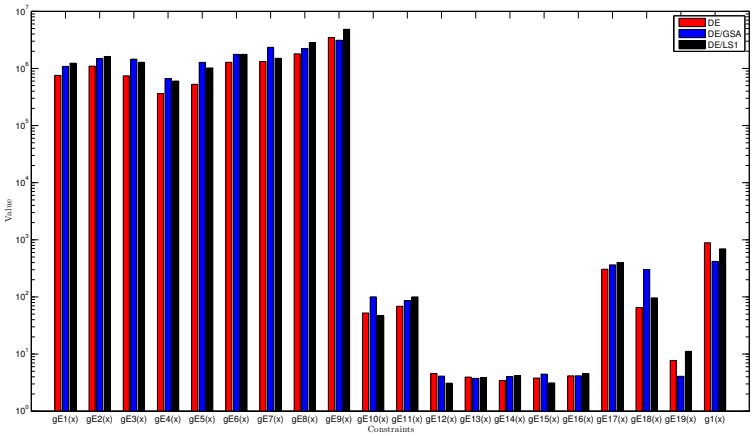


(f) $g21$

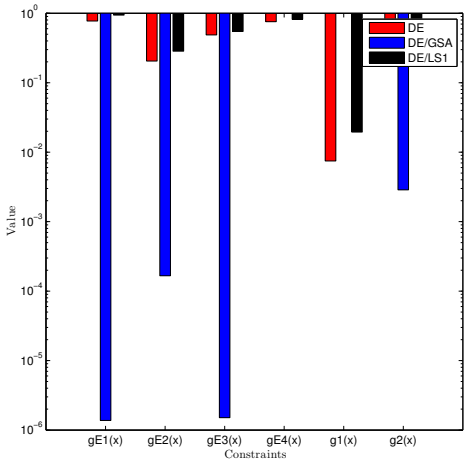
Figure 4.7: Averaged constraint values for the different DE variations at $5,000$ function calls.



(a) $g20$

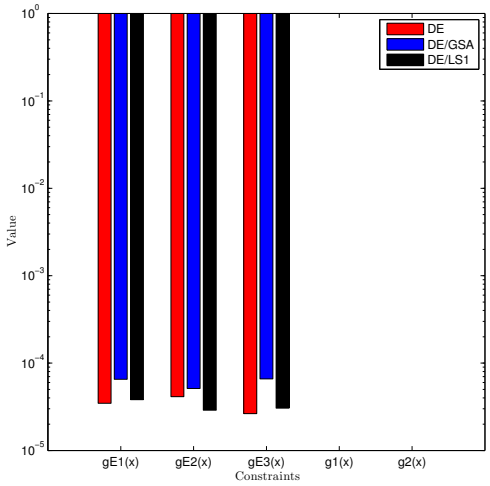


(b) $g22$

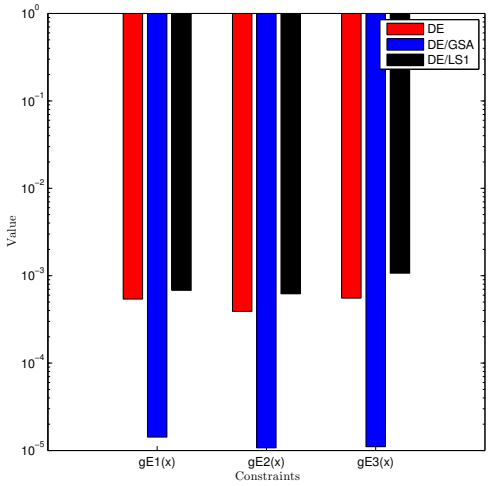


(c) $g23$

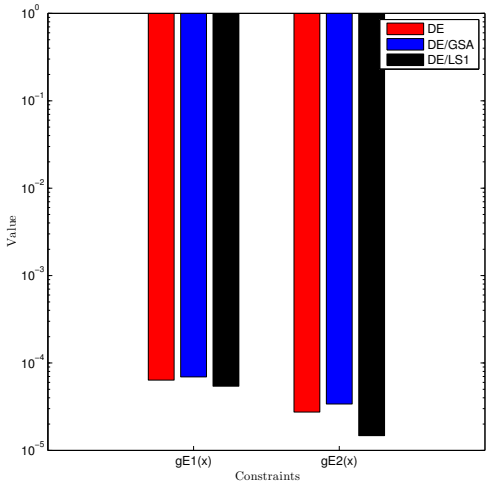
Figure 4.8: Cont'd Averaged constraint values for the different DE variations at $5,000$ function calls.



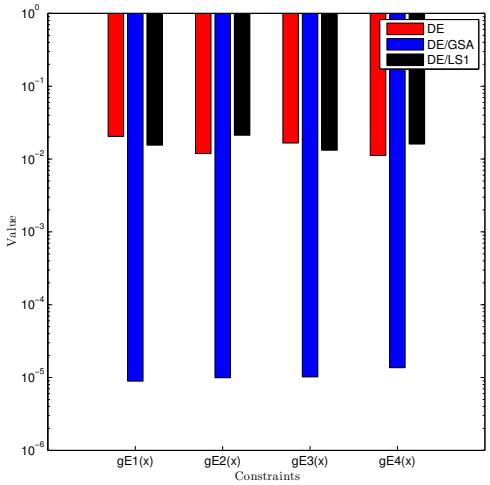
(a) $g5$



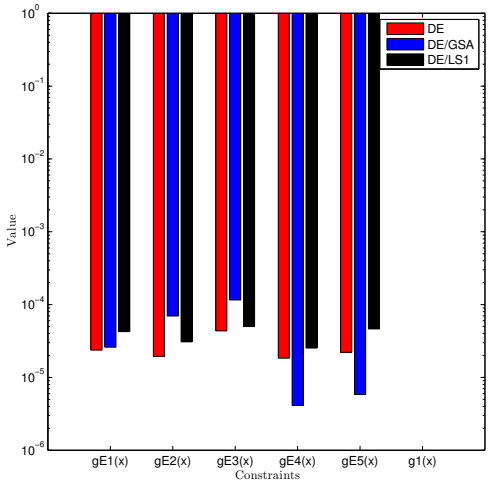
(b) $g14$



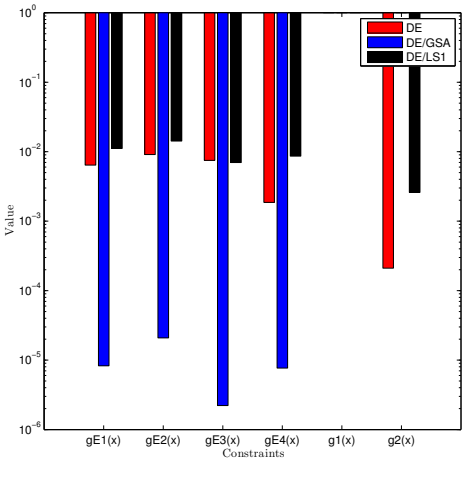
(c) $g15$



(d) $g17$

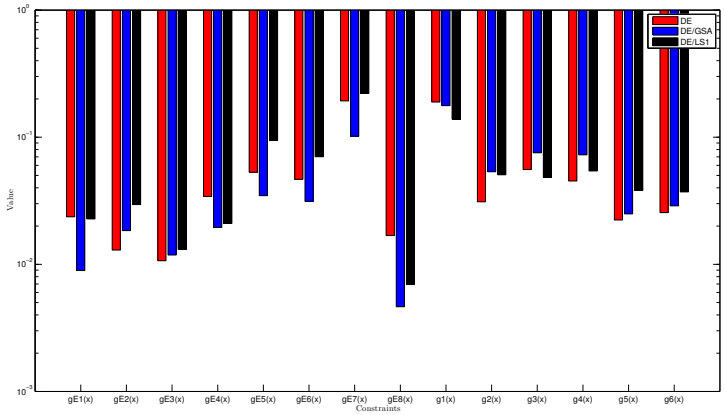


(e) $g21$

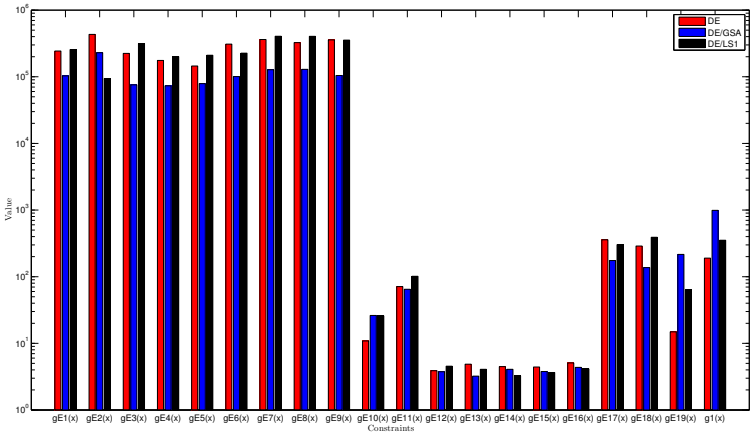


(f) $g23$

Figure 4.9: Averaged constraint values for the different DE variations at $50,000$ function calls

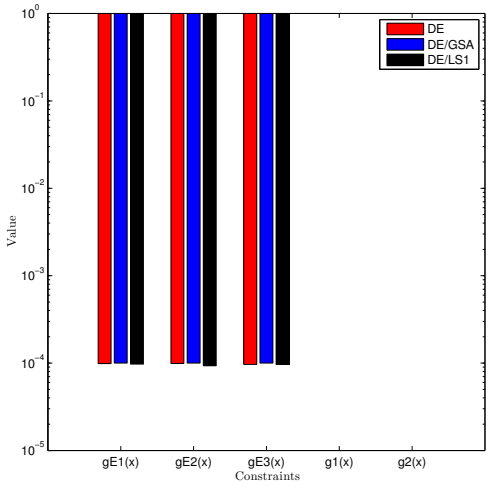


(a) $g20$

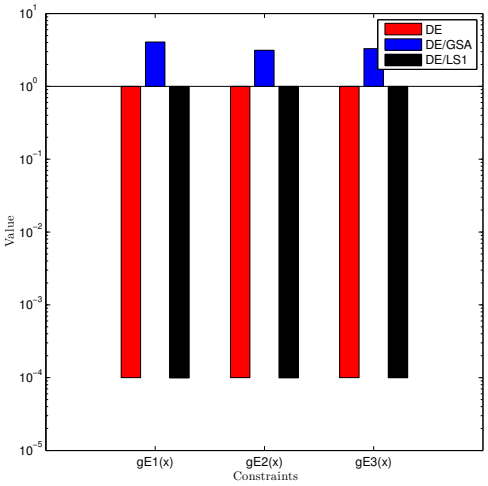


(b) $g22$

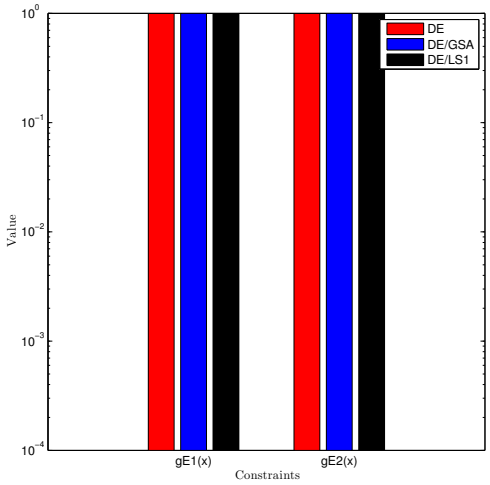
Figure 4.10: Cont'd Averaged constraint values for the different DE variations at $50,000$ function calls.



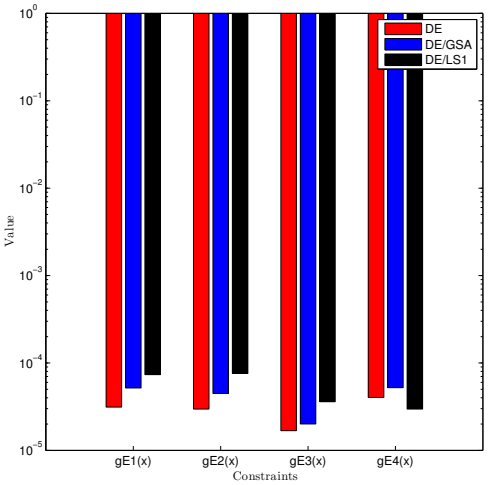
(a) $g5$



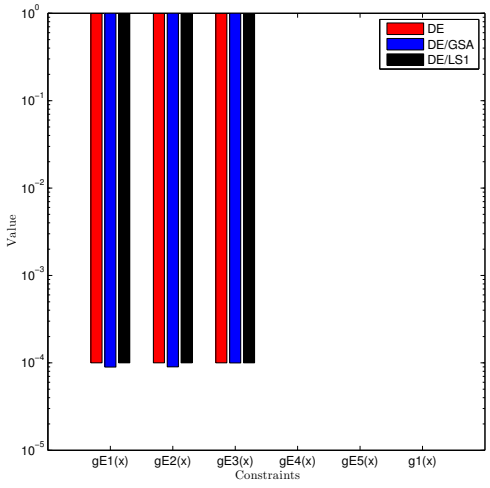
(b) $g14$



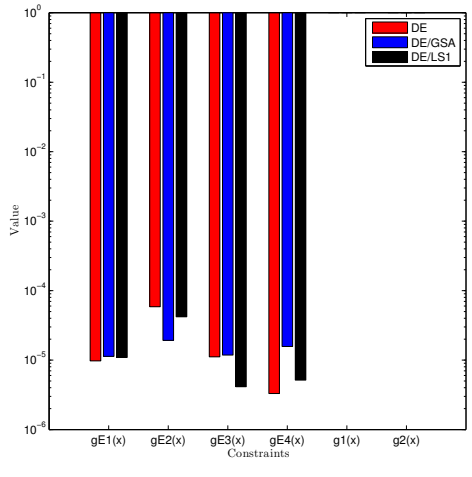
(c) $g15$



(d) $g17$

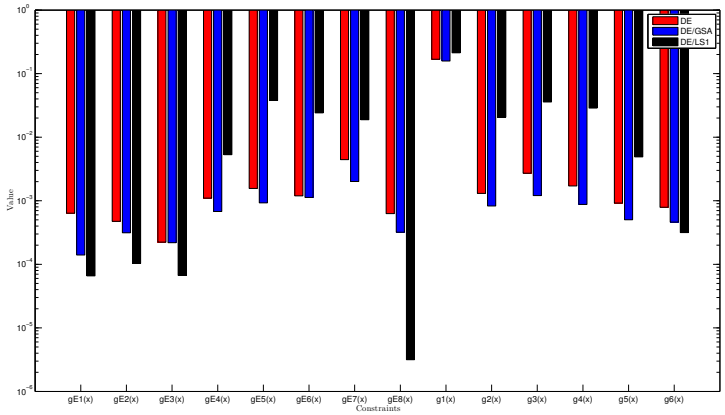


(e) $g21$

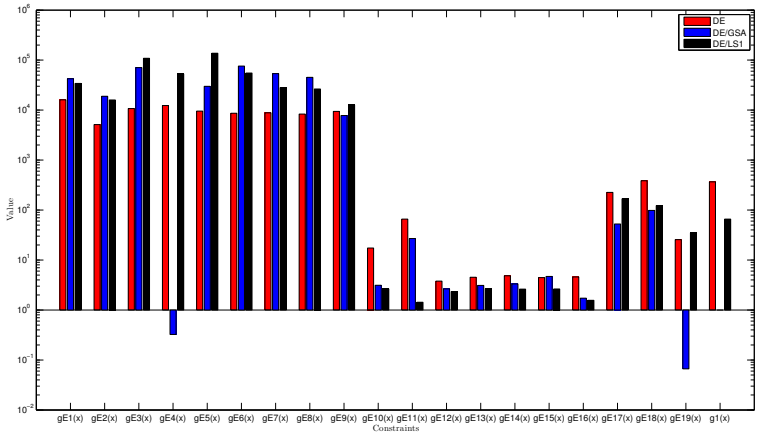


(f) $g23$

Figure 4.11: Averaged constraint values for the different DE variations at $500,000$ function calls

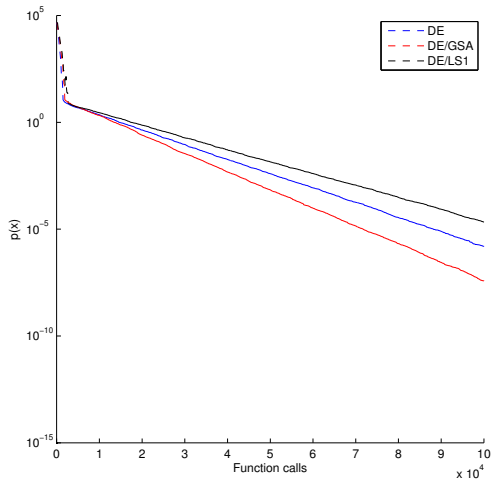


(a) $g20$

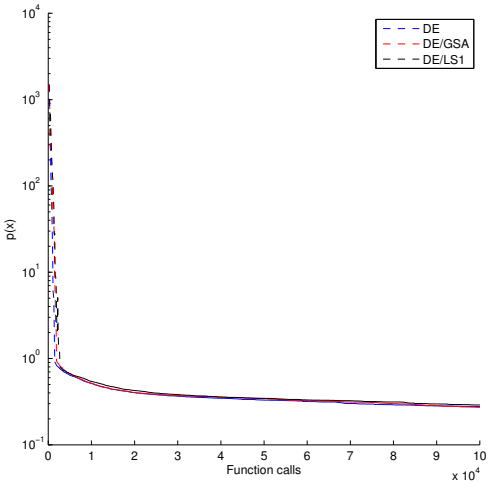


(b) $g22$

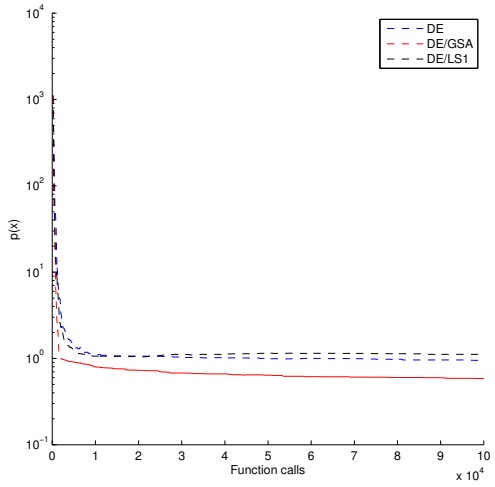
Figure 4.12: Cont'd Averaged constraint values for the different DE variations at $500,000$ function calls.



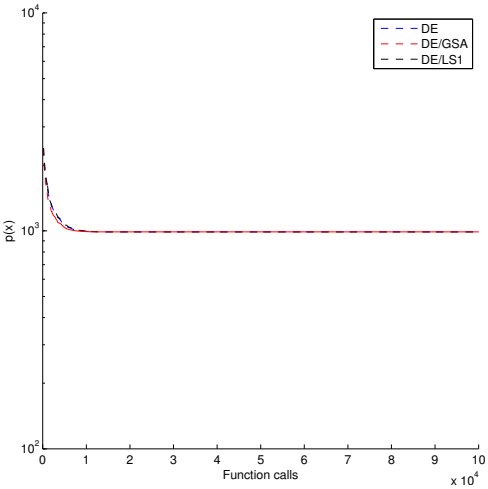
(a) $g1$



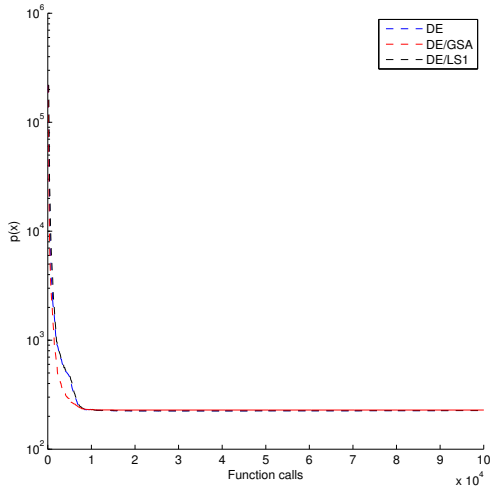
(b) $g2$



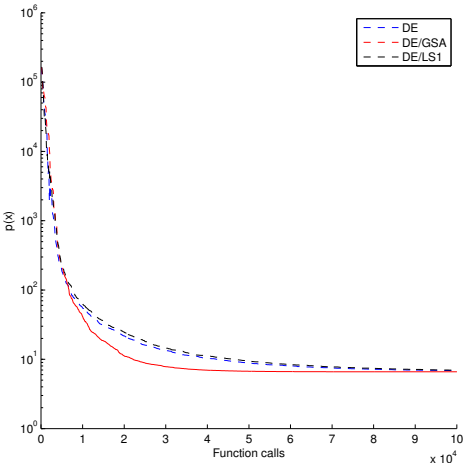
(c) $g3$



(d) $g4$

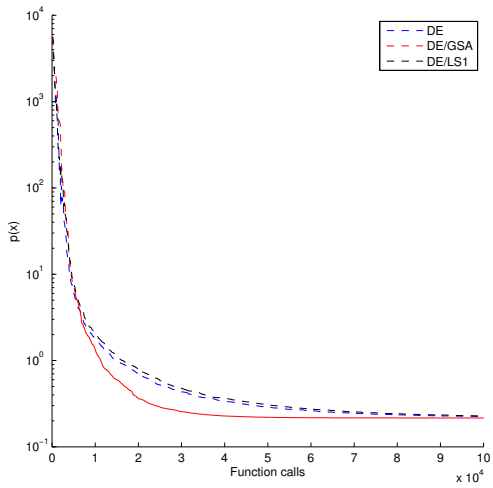


(e) $g6$

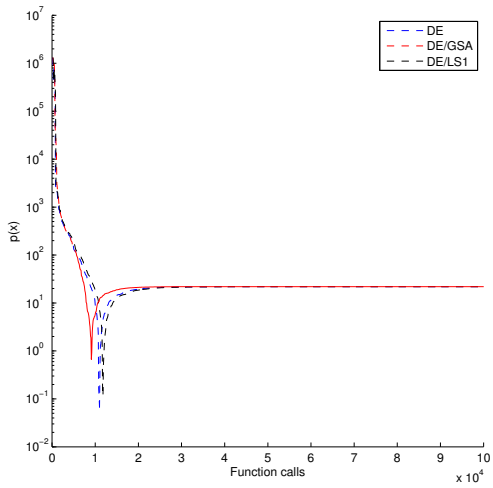


(f) $g7$

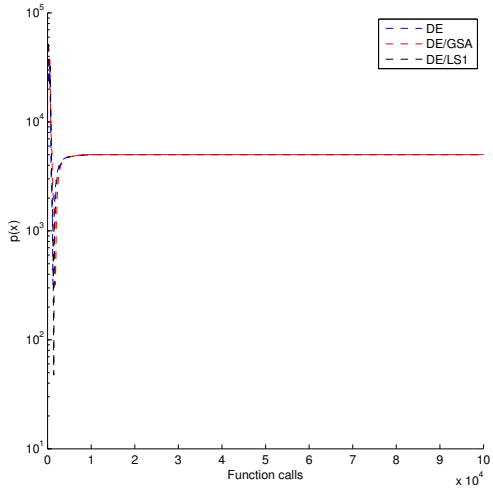
Figure 4.13: Convergence plots for the different DE variations.



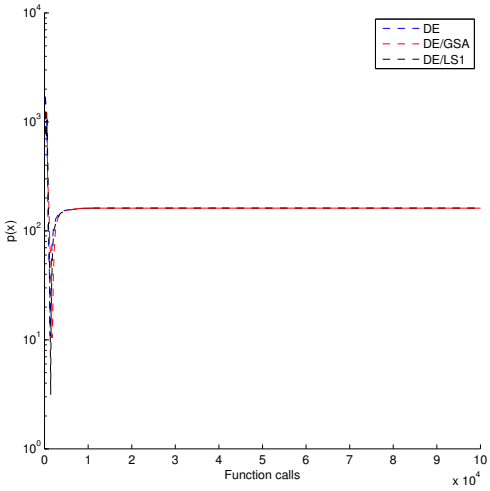
(a) $g8$



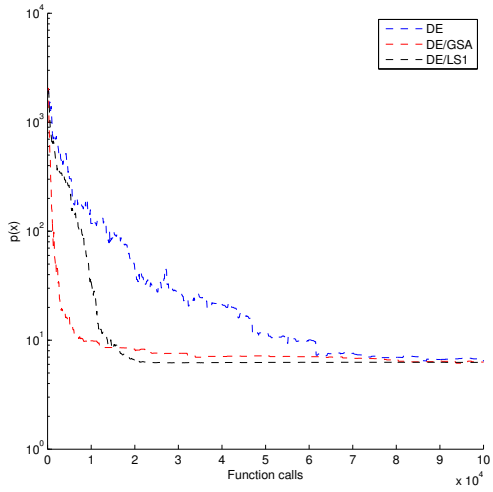
(b) $g9$



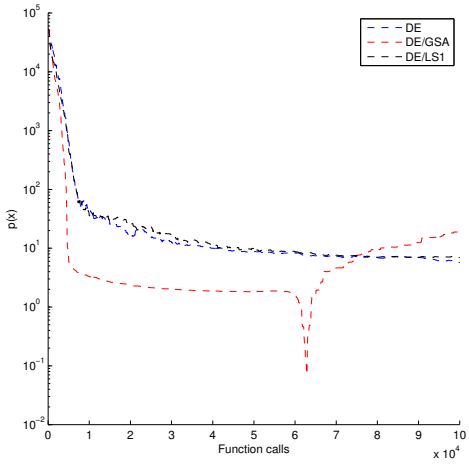
(c) $g10$



(d) $g11$

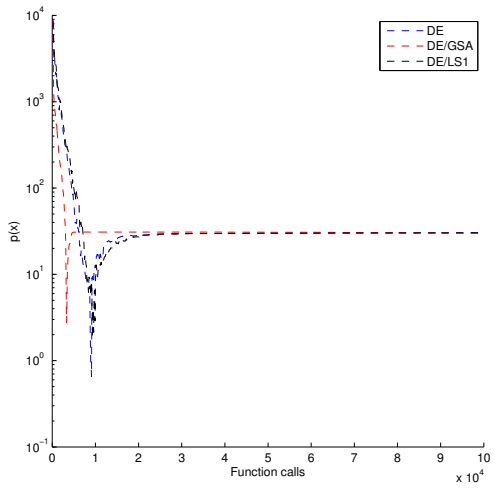


(e) $g13$

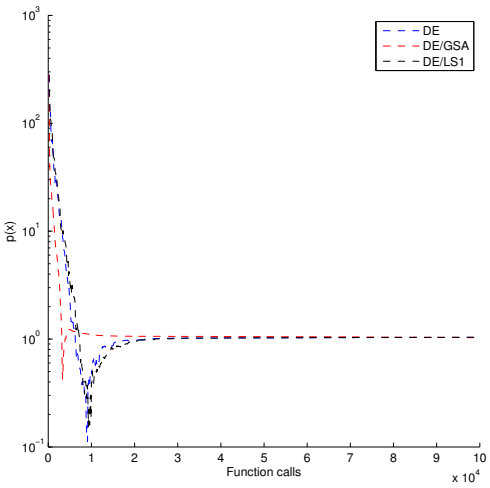


(f) $g14$

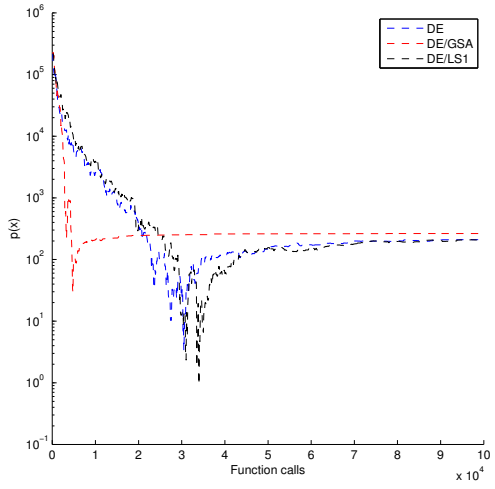
Figure 4.14: Convergence plots for the different DE variations.



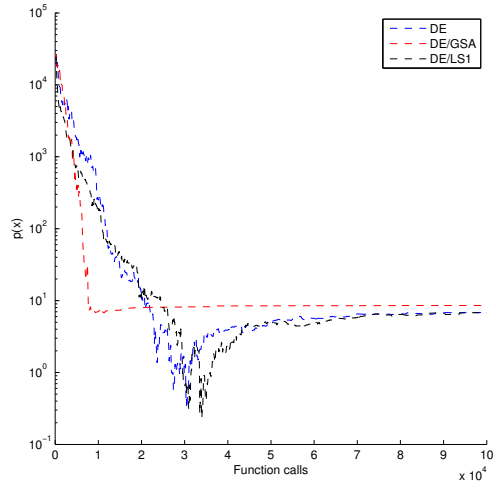
(a) $g15$



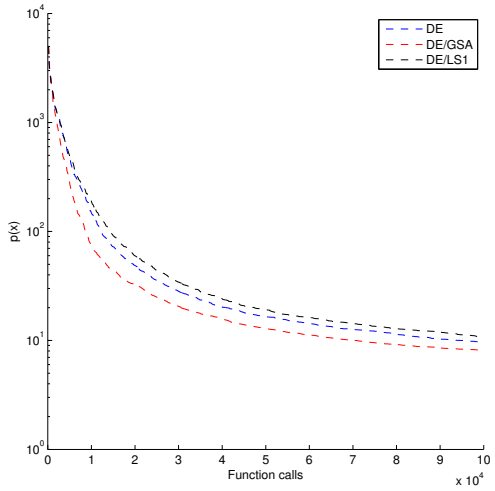
(b) $g16$



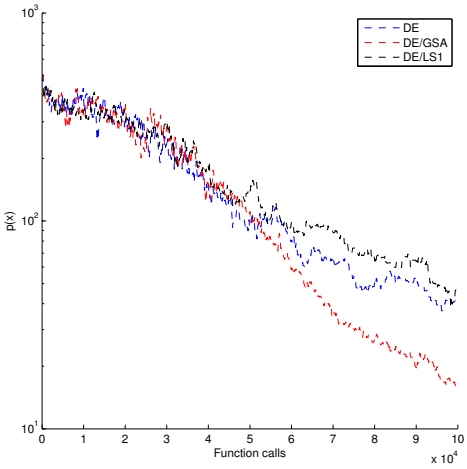
(c) $g17$



(d) $g18$



(e) $g19$



(f) $g20$

Figure 4.15: Convergence plots for the different DE variations.

Table 4.6: Constraints violated by the different DE variants at 5,000 evaluations.

Problem	Constraints		Median						Average					
			DE		DE/GSA		DE/LS1		DE		DE/GSA		DE/LS1	
	E	I	E	I	E	I	E	I	E	I	E	I	E	I
$g1$	0	9	0	0	0	0	0	0	0	0	0	0	0	0
$g2$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g3$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$g4$	0	6	0	0	0	0	0	0	0	0	0	0	0	0
$g5$	3	2	3	0	0	0	3	0	3	0	0	0	3	0.1
$g6$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g7$	0	8	0	0	0	0	0	0	0	0	0	0	0	0
$g8$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g9$	0	4	0	0	0	0	0	0	0	0	0	0	0	0
$g10$	0	6	0	0	0	0	0	0	0	0	0	0	0	0
$g11$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$g13$	3	0	3	0	0	0	3	0	2.9	0	0.3	0	2.7	0
$g14$	3	0	3	0	0	0	3	0	2.9	0	0	0	2.9	0
$g15$	2	0	1	0	0	0	2	0	1.5	0	0	0	1.8	0
$g16$	0	38	0	0	0	0	0	0	0	0	0	0	0	0
$g17$	4	0	4	0	0	0	4	0	4	0	2.0	0	4	0
$g18$	0	13	0	9	0	3	0	5	0	8	0	3.7	0	4.9
$g19$	0	5	0	0	0	0	0	0	0	0	0	0	0	0
$g20$	14	6	7	6	7	6	8	6	6.8	6	6.6	6	7	6
$g21$	5	1	5	0	5	0	5	0	4.6	0	4.6	0	4.8	0
$g22$	19	1	19	0	19	0	19	0	19	0.2	19	0.2	19	0.2
$g23$	4	2	3	2	0	0	4	1	3.7	1.0	0.1	0.1	3.9	0.9
$g24$	0	2	0	0	0	0	0	0	0	0	0	0	0	0

Second Set of Experiments: Saving Resources

A second experiment that measures the benefits of the GSA as local search strategy was performed. We tried to conduct an experiment to analyze the amount of resources, which are measured in function calls, that were saved by our approach. Run-length distributions show the relationship between success ratios and number of evaluations, where the success ratio is defined as the probability of achieving a certain quality level. To establish a quality level, in each problem we consider the highest median obtained by any of the three models considered in 500,000 function evaluations.

Table 4.9 presents the number of evaluations required for each of the models to

Table 4.7: Constraints violated by the different DE variants at 50,000 evaluations.

Problem	Constraints		Median						Average					
			DE		DE/GSA		DE/LS1		DE		DE/GSA		DE/LS1	
	E	I	E	I	E	I	E	I	E	I	E	I	E	I
$g1$	0	9	0	0	0	0	0	0	0	0	0	0	0	0
$g2$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g3$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$g4$	0	6	0	0	0	0	0	0	0	0	0	0	0	0
$g5$	3	2	0	0	0	0	0	0	0	0	0	0	0	0
$g6$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g7$	0	8	0	0	0	0	0	0	0	0	0	0	0	0
$g8$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g9$	0	4	0	0	0	0	0	0	0	0	0	0	0	0
$g10$	0	6	0	0	0	0	0	0	0	0	0	0	0	0
$g11$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$g13$	3	0	0	0	0	0	0	0	0.1	0	0	0	0	0
$g14$	3	0	0	0	0	0	0	0	0	0	0	0	0	0
$g15$	2	0	0	0	0	0	0	0	0	0	0	0	0	0
$g16$	0	38	0	0	0	0	0	0	0	0	0	0	0	0
$g17$	4	0	3	0	0	0	4	0	2.6	0	0	0	3.4	0
$g18$	0	13	0	0	0	0	0	0	0	0	0	0	0	0
$g19$	0	5	0	0	0	0	0	0	0	0	0	0	0	0
$g20$	14	6	6	6	5	6	6	6	5.4	6	4.9	6	5.9	6
$g21$	5	1	0	0	0	0	0	0	0	0	0	0	0	0
$g22$	19	1	19	0	19	1	19	0	19.0	0.1	19	0.4	19	0.2
$g23$	4	2	2	0	0	0	0	0	1.1	0	0	0	1.5	0.1
$g24$	0	2	0	0	0	0	0	0	0	0	0	0	0	0

attain a success ratio equal to 50%.

Discussion of the results

In these experiments we performed several tests to show the correctness of the GSA method. In particular, we applied the method in two different ways: as a standalone algorithm and within a memetic algorithm.

In the first case we artificially generated the individuals of the neighborhood. Using such information we compute several steps for the GSA. To illustrate the aim of the GSA we compare it against two directed search method. From such comparison we obtained that the GSA saved a considerable amount of resources measured in

Table 4.8: Constraints violated by the different DE variants at 500,000 evaluations.

Problem	Constraints		Median						Average					
			DE		DE/GSA		DE/LS1		DE		DE/GSA		DE/LS1	
	E	I	E	I	E	I	E	I	E	I	E	I	E	I
$g1$	0	9	0	0	0	0	0	0	0	0	0	0	0	0
$g2$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g3$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$g4$	0	6	0	0	0	0	0	0	0	0	0	0	0	0
$g5$	3	2	0	0	0	0	0	0	0	0	0	0	0	0
$g6$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g7$	0	8	0	0	0	0	0	0	0	0	0	0	0	0
$g8$	0	2	0	0	0	0	0	0	0	0	0	0	0	0
$g9$	0	4	0	0	0	0	0	0	0	0	0	0	0	0
$g10$	0	6	0	0	0	0	0	0	0	0	0	0	0	0
$g11$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$g13$	3	0	0	0	0	0	0	0	0	0	0	0	0	0
$g14$	3	0	0	0	0	0	0	0	0	0	0.4	0	0	0
$g15$	2	0	0	0	0	0	0	0	0	0	0	0	0	0
$g16$	0	38	0	0	0	0	0	0	0	0	0	0	0	0
$g17$	4	0	0	0	0	0	0	0	0	0	0	0	0	0
$g18$	0	13	0	0	0	0	0	0	0	0	0	0	0	0
$g19$	0	5	0	0	0	0	0	0	0	0	0	0	0	0
$g20$	14	6	0	6	0	6	3	6	0.1	6	0	6	1.9	6
$g21$	5	1	0	0	0	0	0	0	0	0	0	0	0	0
$g22$	19	1	19	0	19	0	19	0	19	0.2	19	0	18.9	0
$g23$	4	2	0	0	0	0	0	0	0	0	0	0	0	0
$g24$	0	2	0	0	0	0	0	0	0	0	0	0	0	0

function calls. Thus, GSA can obtain a solution with a similar error using approximately 10% of the budget of the other two algorithms.

The second set of experiments measure a memetic algorithm that uses GSA as a local search engine called DE/GSA. For such algorithm we obtained promising results when it was compared against the standalone DE and with another local search engine, LS1. The test was performed using the CEC'06 constrained functions. In the first set of results the GSA method obtained significant results in eight functions at the medium term. The best results were obtained in the problems with equality constraint. By such result we can confirm the effectiveness of GSA for those types of problems. The effectiveness of the GSA method on equality constraints is given

Table 4.9: Evaluations required by the different DE variants to obtain a fixed quality level and percentage of saved resources.

	DE	DE-LS1		DE/GSA	
	Eval.	Eval.	Saved (%)	Eval.	Saved (%)
g01	221250	281738	-27.33	274679	-24.18
g02	223200	438630	-96.51	185456	16.91
g03	378600	34205	90.96	7862	97.92
g04	66900	74409	-11.22	63274	5.42
g05	343000	417438	-21.70	143974	58.02
g06	35650	39905	-11.93	39424	-10.58
g07	188550	469315	-148.90	60750	67.78
g08	56550	49020	13.31	55824	1.28
g09	290400	343752	-18.37	235126	19.03
g10	32450	37468	-15.46	9062	72.07
g11	37700	42292	-12.18	32702	13.25
g13	494550	25001	94.94	265106	46.39
g14	327350	496336	-51.62	101196	69.08
g15	134700	222260	-65.00	98313	27.01
g16	147500	164214	-11.33	246482	-67.10
g17	226900	364715	-60.73	10194	95.50
g18	440850	96543	78.10	480413	-8.97
g19	75450	482225	-539.13	58110	22.98
g20	162550	143728	11.57	58079	64.27
g21	485950	497234	-2.32	286394	41.06
g22	256400	482546	-88.20	55090	78.51
g23	110300	366143	-231.95	133544	-21.07
g24	33150	36641	-10.53	43808	-32.15
HC-10	454200	77384	82.96	206230	54.59
HC-20	489900	106768	78.20	89969	81.63

by two different factors: the computation of the direction and the correction step. Incorporating a linearization of the equality constraints into the computation of the direction create a descent direction that moves along the constraints. Meanwhile, the correction step adjust a candidate solution if it becomes infeasible. Such mechanisms improve the convergence rate of the memetic algorithm.

Besides, if we compare the results at long term, we can confirm that DE/GSA also has the capability to the same solutions that the DE method. Using such a result we can also state that the balancing mechanism does not affect the convergence of the memetic algorithm.

Finally, the table of saved resources shows the potential of GSA in this regards. In particular, we could observe functions where our method helps to save a considerable amount of resources. In some cases such savings exceed the 90%. With such results we can claim that GSA has shown that it can improve an evolutionary algorithm by accelerating its convergence rate.

The different mechanism proposed for the memetic DE/GSA method helped convergence. But, we believe that the results can be improved by using a more sophisticated method for constraint handling and for the movement to the optimal solution of an infeasible solution. For example, consider the interior point methods [Nocedal and Wright, 2006]. Using such type of procedure we can improve the results when we are in a feasible region defined by inequality constraints.

Besides, it is possible to improve the penalty function by using more sophisticated methods, e.g. [Fletcher, 1975],[Olsen, 1994] and [Liu et al., 2016a].

4.7 Multi-objective GSA

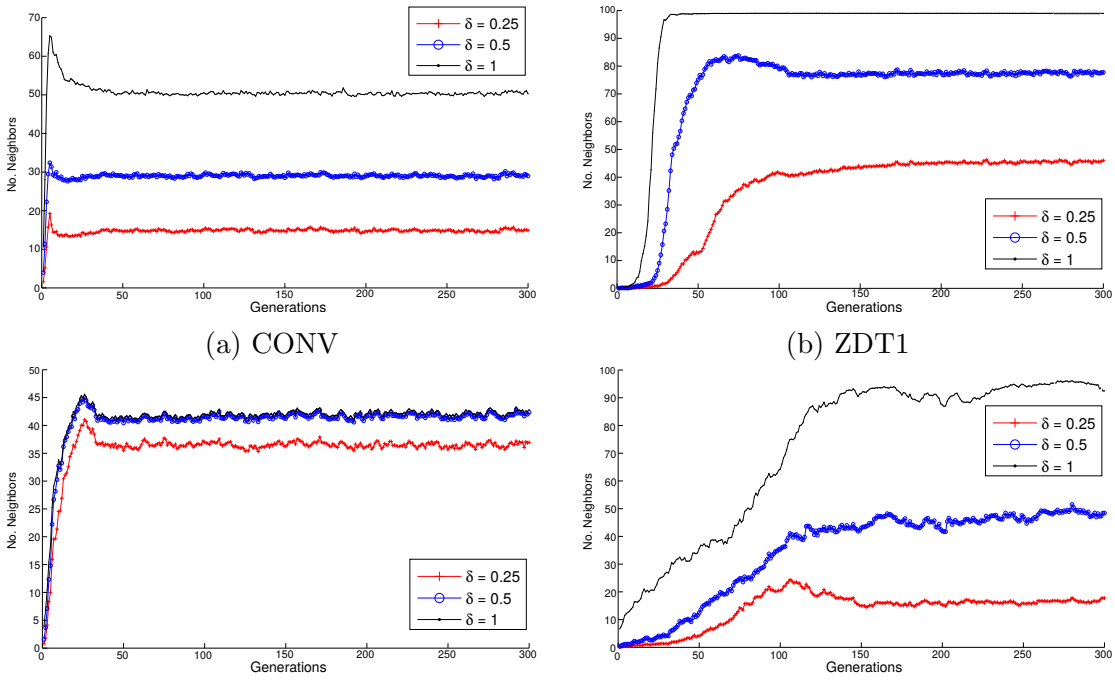
The GSA method has shown to be a powerful tool for solving SOPs. Hence, we propose to extend such a method to the multi-objective context. In this section, we discuss the realization and application of GSA to such a context. As the first step, we analyze the applicability of the method for MOPs. That is, we discuss if there exists a minimal number of individuals in the neighborhood of a candidate solution such that GSA can be applied. Next, we present the mechanisms required for GSA to handle several objectives. Finally, we present the application as a local search tool within memetic strategies. In particular, we apply the GSA along with two different state-of-the-art algorithms: MOEA/D [Zhang and Li, 2006] and IG-NSGA-II [Liu et al., 2016b].

4.7.1 Applicability of GSA within MOEAs

As discussed above the neighborhood selection of the GSA is one of the crucial studies of the method. In particular, when we applied such a method for SOPs we empirically showed that the number of neighboring individuals for a candidate solution x_0 increases when the algorithm starts to converge. By such property, the size of the neighborhood is small enough such that there exist sufficient samples around any candidate solution x_0. In MOPs, a whole solution set needs to be computed. Thus, not all the individuals in the population group around the same solution. Hence, it becomes necessary to demonstrate that there exists a certain number of neighbors around any candidate solution in the population to apply the GSA method.

In order to demonstrate the proposition described above we apply a similar experiment that the one for the single-objective GSA. That is, we compute the number of candidate solutions that exists around a given candidate solution x_0. For such an experiment, we obtained different populations through MOEA/D [Zhang and Li, 2006] using four different MOPs: CONV, ZDT1 [Zitzler et al., 2000b], Kursawe [Kursawe, 1990] and DTLZ3 [Deb et al., 2002b] with three objectives (for the problem formulation please refer to Appendix B). For each MOP we compute the averaged number of neighboring solutions for a individual $p \in P$. To measure the number of neighbors of individual p we use the 2-norm such that the neighborhood is defined as:

$$N_\delta(p_0) := \{p \in P \setminus \{p_0\} : \|p - p_0\|_2 \leq \delta\}. \tag{4.97}$$



(a) CONV (b) ZDT1

(c) Kursawe (d) DTLZ3

Figure 4.16: Results for neighborhood size for multi-objective GSA

Once we defined the neighborhood we are in position to measure the number of neighbors as $|N_\delta(p_0)|$. For our experiments we used three different values of δ: 0.25, 0.5 and 1. Figure 4.16 presents the averaged results for the numbers of neighboring solutions with a population size $N = 100$. The figures shows the computations for 30 independent runs on each of the test functions.

From the results it is clear that we can obtain a 'good' number of neighboring solutions at early stages of MOEA/D, e.g., after 50 generations. Such results show that it is possible to obtain the information that the GSA requires in order to approximate the gradient. Hence, we can apply a method to construct the neighborhood for a candidate solution as in the single-objective GSA. Such procedure will be defined according to the MOEA to be used as a base algorithm for the memetic strategy: MOEA/D or IG-NSGA-II.

4.7.2 Approximating the Jacobian

GSA approximates the gradient information of a SOP. But, it becomes necessary to extend this process in order to compute the Jacobian matrix $J(x_0) \in \mathbb{R}^{k \times n}$. Assume a MOP as in Equation (MOP) along with a candidate solution x_0 and r search directions $\nu_r, \ldots, \nu_r \in \mathbb{R}^n$. Thus, the r search directions form the subspace:

$$S := \text{span}\{\nu_1, \ldots, \nu_r\}. \tag{4.98}$$

Hence, we can give a gradient approximation for the subspace S. That is, every gradient $\nabla f_i(x_0)$ of the Jacobian matrix can be computed using the GSA method as follows:

$$\tilde{g}_i(x_0) = V(V^T V)^{-1} V^T \nabla f_i(x_0) \in \mathbb{R}^n, \tag{4.99}$$

where $i = 1, \ldots, k$.

Considering the Jacobian matrix defined as in Equation (2.40) and the approximation given by (4.99), the GSA approximation can be computed as:

$$\tilde{J}(x_0) = \begin{pmatrix} \tilde{g}_1(x_0)^T \\ \vdots \\ \tilde{g}_k(x_0)^T \end{pmatrix} = J(x_0)^T V(V^T V)^{-1} V^T. \tag{4.100}$$

Finally, we proceed to define the gradient-free realization of the procedure above. Consider the matrix $\bar{\mathcal{F}} \in \mathbb{R}^{k \times r}$ defined as follows:

$$\bar{\mathcal{F}} = J(x_0)^V \begin{pmatrix} \langle \nabla f_1, \nu_1 \rangle & \ldots & \langle \nabla f_1, \nu_r \rangle \\ \vdots & & \vdots \\ \langle \nabla f_k, \nu_1 \rangle & \ldots & \langle \nabla f_k, \nu_r \rangle \end{pmatrix}, \tag{4.101}$$

where each entry $\bar{\mathcal{F}}_{ij}$ of such matrix can be computed using Equation (3.14).

4.7.3 Computing a descent direction

One of the main differences between the single-objective approach and the multi-objective one is how to compute the descent direction. In this work we consider two

different approaches to compute such a direction according to the base algorithm. In case of MOEA/D we propose to use the scalarization function in order to compute the descent direction. On the other side, to apply the GSA into the IG-NSGA-II we use the concept of descent direction proposed in [Schütze et al., 2011].

Descent direction using a scalarization function

A descent direction can be computed taking advantage of the scalarization function used within MOEA/D. In particular, we are interested in the scalarization function described in Equation (2.54) for unconstrained MOPs. Using such a scalar function we can redefine the computation of the descent direction for the GSA approach. Consider a candidate solution x_0, a set of r neighboring points $x_i, \ldots, x_r \in \mathbb{R}^n$ along with r search directions $\nu_1, \ldots, \nu_r$. Using the search directions, we define a vector $\mathcal{T} \in \mathbb{R}^r$ where each entry can be obtained by:

$$\mathcal{T}_i = \frac{T(f(x_i), w, z) - T(f(x_0), w, z)}{\|x_i - x_0\|_2}, \tag{4.102}$$

where w is the weight vector and z is the ideal point.

If we consider that the Tchebycheff scalarization function is applied in the context of the gradient-free GSA we can rewrite Equation (4.25) as follows:

$$\tilde{V}^T \tilde{V} \lambda_{\mathcal{T}} = -\mathcal{T}. \tag{4.103}$$

Considering that $\lambda^*_{\mathcal{T}}$ is a solution of Equation (4.103), then, the most greedy direction can be computed as:

$$\tilde{\nu}^*_{\mathcal{T}} = \frac{-1}{\|\tilde{V}\lambda^*_{\mathcal{T}}\|_2^2} \tilde{V} (\tilde{V}^T \lambda^*_{\mathcal{T}})^{-1} \mathcal{T}. \tag{4.104}$$

Descent direction using Lara direction

Our second approach computes a descent direction combining Lara's descent direction [Schütze et al., 2011] and the approximation given in Equation (4.100):

$$\nu_L = \left(\frac{\tilde{g1}}{\|\tilde{g1}\|} + \frac{\tilde{g2}}{\|\tilde{g2}\|} \right) \in \mathbb{R}^n, \tag{4.105}$$

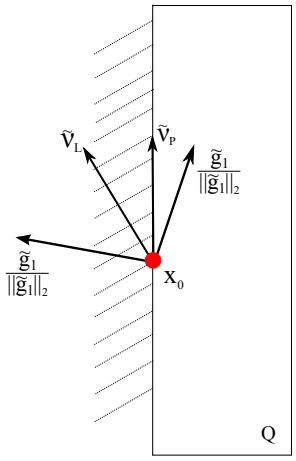


Figure 4.17: Example of correction step for Lara direction

where $\tilde{g1}$ and $\tilde{g2}$ represent the row vectors of the matrix $\tilde{J}(x_0)$.

Unfortunately, Equation (4.105) can only be applied for unconstrained MOPs. In order to avoid such a restriction we extend the direction to handle inequality constrained problems. For such a purpose we use the gradient projection method of the GSA. Hence, we propose to compute the direction ν_L and correct it if needed. For example, Figure 4.17 presents a hypothetical example of the correction step. On such example Lara's direction points into the infeasible region (denoted by dotted lines) and the correction step computes the new direction $\tilde{\nu_p}$.

If we consider that there exist l inequality constraints, we define matrix $\hat{M}$ as in Equation (4.62). Next, we construct the projection of the direction by computing matrix $V\bar{K}$ as in Equation (4.63). Finally, the desired projected direction is given by:

$$\tilde{\nu_p} = V\bar{K}(V\bar{K})^T\nu_L. \tag{4.106}$$

4.7.4 Memetic algorithm

As mentioned above in order to show the effectiveness of GSA for MOPs, we propose two memetic algorithms using different base algorithms. The first one, MOEA/D/GSA, is based on MOEA/D and is designed for the treatment of unconstrained MOPs. The second approach is based on IG-NSGA-II to solve inequality constrained MOPs. In this section, we describe the algorithms and present the necessary mechanism for the

memetic realization.

Computing the initial step size

As mentioned above, one of the main challenges for the GSA realization is the computation of the initial step size. If we remember given a candidate solution x_0 we compute the new candidate solution as:

$$x_{new} = x_0 + t_0 \, \nu. \tag{4.107}$$

In this case, we need to compute the initial step size. For such purpose, we propose to take advantage of a given neighborhood structure (the computation of such structure will be described later on). Given a neighborhood structure $\mathcal{N}$ such as the one proposed in Section 4.5.1, the initial step size t_0 can be computed using Algorithm 25.

Algorithm 25 Pseudocode for computing the initial step size of multi-objective GSA.

Require: Candidate solution x^0, Neighborhood $\mathcal{N}$
Ensure: Initial step size t_0

```
Set r_N := |N|
Set n := |x_0|
Set t_0 := 0
for i = 1, ..., rN do
    Set y := N → x^i
    for j = 1, ..., n do
        Set t_0 := t_0 + |x_j^0 − y_j|
    end for
end for
Set t_0 := t_0 / (n r_N)
return t_0
```

MOEA/D/GSA

The first memetic algorithm to be defined is the one that uses MOEA/D as its base algorithm. One of the great advantages of such an algorithm is that it uses a scalarization function within its mechanism. Such property gives us an opportunity to apply several of the mechanisms that have been proposed for the single-objective version of GSA.

MOEA/D computes a neighborhood structure according to its weight vectors w_i. Thus, we could use such a structure for the computations of GSA.

As mentioned before, one of the crucial mechanisms for the memetic algorithm is the balancing of the shared resources, i.e. the distribution of the function calls for the local search and the stochastic operators. Since we have the MOEA/D scalar function, we are in the position to use the procedure described in [LaTorre, 2009, LaTorre et al., 2011]. For such a procedure we need to compute the quality term q of each of the techniques (GSA and MOEA/D). Consider a solution x_i and its offspring x_{i+1} along with their images: $F(x_i)$ and $F(x_{i+1})$ respectively. The quality term according to the Tchebycheff function can be obtained as follows:

$$q^* = \frac{T(F(x_{i+1}), w, z) - T(F(x_{i+1}), w, z)}{T(F(x_{i+1}), w, z)}, \tag{4.108}$$

where q^* refers to the quality of either of the techniques (q^{GSA} or q^{MD} for MOEA/D, respectively).

Using a notation similar to Section 4.5.3 we named as O^{MD} to the offspring subpopulation created using the MOEA/D procedures. The quality of O^{MD} can be computed as:

$$Q^{MD} = \frac{\sum_{i=1}^{|q^{MD}|} q_i^{MD}}{|q^{MD}|}. \tag{4.109}$$

For the quality of the GSA subpopulation O^{MD}, we proceed analogously as in Equation (4.85).

Once we have computed the quality term we proceed to define the algorithm that computes the participation ratios for each technique($\mathcal{R}^{GSA}$ and $\mathcal{R}^{MD}$). Algorithm 26 presents the pseudocode to compute such ratios. On the algorithm, we propose to use values of $\mathcal{R}_- = 0.1$ and $\mathcal{R}_c = 0.05$.

Algorithm 26 Pseudocode of the participation ratio for MOEA/D/GSA.

Require: Initial participation ratios $\mathcal{R}^{GSA}$, $\mathcal{R}^{MD}$, Technique qualities Q^{GSA}, Q^{MD}, $\mathcal{R}_c$, $\mathcal{R}_-$
Ensure: Participation ratios $\bar{\mathcal{R}}^{GSA}$, $\bar{\mathcal{R}}^{MD}$
1: **if** $Q^{GSA} > Q^{MD}$ **then**
2: $r_a := \mathcal{R}_c \left(\frac{Q^{GSA} - Q^{MD}}{Q^{GSA}} \right) \mathcal{R}^{GSA}$.
3: **if** $(\mathcal{R}^{MD} - r_a) < \mathcal{R}_-$ **then**
4: $r_a := \mathcal{R}^{MD} - \mathcal{R}_-$.
5: **end if**
6: Set $\bar{\mathcal{R}}^{GSA} := \mathcal{R}^{GSA} + r_a$.
7: Set $\bar{\mathcal{R}}^{MD} := \mathcal{R}^{MD} - r_a$.
8: **else if** $Q^{MD} > Q^{GSA}$ **then**
9: $r_a := \mathcal{R}_c \left(\frac{Q^{MD} - Q^{GSA}}{Q^{MD}} \right) \mathcal{R}^{MD}$.
10: **if** $(\mathcal{R}^{GSA} - r_a) < \mathcal{R}_-$ **then**
11: $r_a := \mathcal{R}^{GSA} - \mathcal{R}_-$.
12: **end if**
13: Set $\bar{\mathcal{R}}^{MD} := \mathcal{R}^{MD} + r_a$.
14: Set $\bar{\mathcal{R}}^{GSA} := \mathcal{R}^{GSA} - r_a$.
15: **else**
16: Set $\bar{\mathcal{R}}^{GSA} := \mathcal{R}^{GSA}$.
17: Set $\bar{\mathcal{R}}^{MD} := \mathcal{R}^{MD}$.
18: **end if**

Finally, Algorithm 27 presents the realization of the memetic GSA. In the algorithm, N represents the number of subproblems for MOEA/D.

Algorithm 27 Pseudocode of the memetic MOEA/D/GSA.

Initializes the N weight vectors
Randomly create initial subproblem population P_0.
Compute ideal point z
Set $\mathcal{R}^{MD} := \mathcal{R}^{GSA} := 0.5$.
Set $i := 0$.
while Stopping criteria is not met **do**
  Set $N^{GSA} := \lfloor N\,\mathcal{R}^{GSA} \rfloor$
  Set $N^{MD} := \lfloor N\,\mathcal{R}^{MD} \rfloor$
  Compute $d_N := N - (N^{GSA} + N^{MD})$
  if $d_N > 0$ **then**
    Set $N^{GSA} := N^{GSA} + \lfloor d_N \rfloor$
    Set $N^{MD} := N^{MD} + \lceil d_N \rceil$
  end if
  Randomly select N^{MD} individuals from P_i to create O^{MD}
  Randomly select N^{GSA} individuals from P_i to create O^{GSA}
  Set $q^{MD} := 0 \in \mathbb{R}^{N^{MD}}$
  for $j = 1, \ldots, N^{MD}$ **do**
    Compute crossover and mutation on $O^{MD} \to x^j$
    Compute q_j^{MD} using Equation (4.84)
  end for
  Compute Q^{MD} using Equation (4.109)
  Set $q^{GSA} := 0 \in \mathbb{R}^{N^{GSA}}$
  for $j = 1, \ldots, N^{GSA}$ **do**
    Set $x := O^{GSA} \to x^j$
    Compute neighborhood of x
    Compute initial step size t_0 using Algorithm 25
    Compute ν as in Equation (4.104)
    Set $x_a := x + t\nu$
    Compute q_j^{GSA} using Equation (4.84)
  end for
  Compute Q^{GSA} using Equation (4.85)
  Update participation ratios $\mathcal{R}^{MD}$, $\mathcal{R}^{GSA}$ using Algorithm 26
  Update the MOEA/D subproblems population P_{i+1} using O^{MD} and O^{GSA}
  Update ideal point z
  Set $i := i + 1$
end while

IG-NSGA-II/GSA

In order to handle constrained MOPs we present our next realization: the IG-NSGA-II/GSA. In this particular method, we change the procedure to apply the local search

strategy. In particular, we propose to apply the local search using two different parameters: ϕ_G and ϕ_I. The ϕ_G value is used to control at which generation the local search is applied (for our method we set the value as 10). The second parameter ϕ_I gives us a probability that defines if an individual is affected by the local search or not. For the memetic algorithm we set $\phi_I = 0.1$.

For the construction of our method we refer to Algorithm 7. In particular, we modify only the mechanism that defines how the local search is applied. The first step to define is the computation of the neighborhood structure. Given a candidate solution x_0 we compute its neighborhood structure as in Algorithm 20.

Algorithm 28 present the local search application. For such method we need to know what is the actual generation so we introduce the parameter I_G that give us such value. Besides, we use the function **mod** that computes the module of a number. Finally, we use the modified Lara's direction procedure proposed in Equation (4.105). Also, if needed, we compute the projection of such direction.

Algorithm 28 Pseudocode for local search of IG-NSGA-II/GSA.

Require: Candidate solution x_0, Neighborhood structure $\mathcal{N}$, Generation number I_G
Ensure: New candidate solution x_{new}

```
if mod(phi_G, I_G) = 0 then
    Compute random value u ∈ [0, 1]
    if u ≤ φ_I then
        Compute ν using Equation (4.105)
        Compute the number of active inequality constraints l
        if l > 0 then
            Project ν using Equation (4.106)
        end if
        Compute initial step size t_0 using Algorithm 25
        Set x_new := x_0 + t_0 ν
    end if
end if
```

4.7.5 Numerical Results

In this section, we present several experiments to show the effectiveness of GSA as an effective tool within multi-objective memetic algorithms. We performed three different experiments to illustrate the performance of GSA on MOPs. The first ex-

periment tests the standalone GSA used along with Lara's direction. The second experiment presents the results for MOEA/D/GSA with unconstrained MOPs. Finally, we present the results of the memetic IG-NSGA-II/GSA.

GSA with Lara's direction

First, we investigate the influence of the parameter r in the construction of the GSA direction. As an example we consider the unconstrained convex bi-objective problem given by Equation (3.19) with $n = 10$, $a_1 = (1, \ldots, 1)^T \in \mathbb{R}^n$ and $a_2 = (-1, \ldots, -1)^T \in \mathbb{R}^n$.

For the experiment, we generated a random candidate solution x_0. Next, we compute several candidate solutions using different values of r to compute the descent direction. To compute any of the new candidate solutions we fix the step size values $t = 1$. Figure 4.18 presents the result of our experiment.The y_0 value represents the image of the original candidate solution. Meanwhile, the y_1 value indicates the candidate solution generated using the exact gradient information. From the figure, it is visible that we obtain an improvement for the computation of the ν direction as the number of neighbors increases.

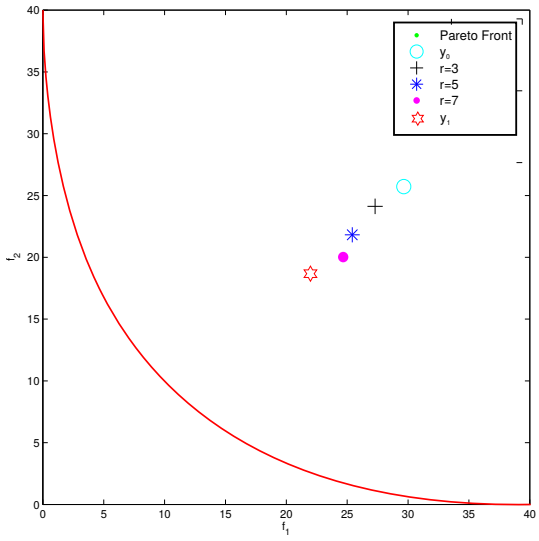


Figure 4.18: Results of the approximation using several values of r

Since the consideration of one step has no significance, we consider an entire trajectory of solutions starting with x_0 using $r = 5$. In Figure 4.19, a sequence of 15

iterations is shown. As it can be seen, the iterations come close to the Pareto front.

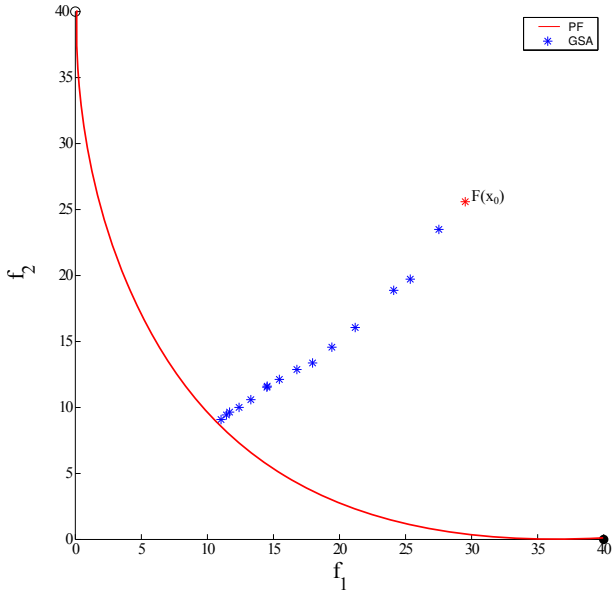


Figure 4.19: Standalone GSA applied on a quadratic function

As already mentioned above, more theoretical investigations are required to fully understand our proposed GSA-based local search engine which we, however have to leave to future research. In the following we focus on the effect of the GSA-based local search engine within memetic multi-objective evolutionary algorithms.

MOEA/D/GSA

For this experiment we use the original version of the MOEA/D algorithm as it was presented in [Zhang and Li, 2006]. A comparison using two different state-of-the-art indicators is performed. The Δ_2 indicator [Schütze et al., 2012] and the hypervolume indicator [Zitzler and Thiele, 1999] were selected for the comparison. We propose to compare the algorithms (the standalone and the memetic version) after they have spent a certain number of function calls. We performed our comparison when the algorithms reached $30,000$ function evaluations for the problems with two objectives and $50,000$ evaluations for $k = 3$. We stress out that MOEA/D is not explicitly aiming at one of any existing performance indicators. Instead, it measures the success via improvements in the scalarization functions. Hence, along with the two state-of-the-art indicators, we propose a specialized indicator that measuares the averaged scalar value of the whole population (and thus, in a sense the overall success of the

MOEA/D variants). Such an indicator can be defined as follows:

$$W(P) = \sum_{i}^{N} T(F(x_i), w_i, z), \tag{4.110}$$

where $x_i \in P$ is the best individual for the i-th subproblem.

To confirm that the GSA method can really improve a memetic algorithm it becomes necessary to make a comparison on different test functions. In particular, for this memetic strategy we selected two of the state-of-the-art benchmark suites. The first selected set of functions is the ZDT benchmark functions proposed in [Zitzler et al., 2000b]. For the second part of the experiments we use the DTLZ benchmark for $k = 3$ [Deb et al., 2002b].

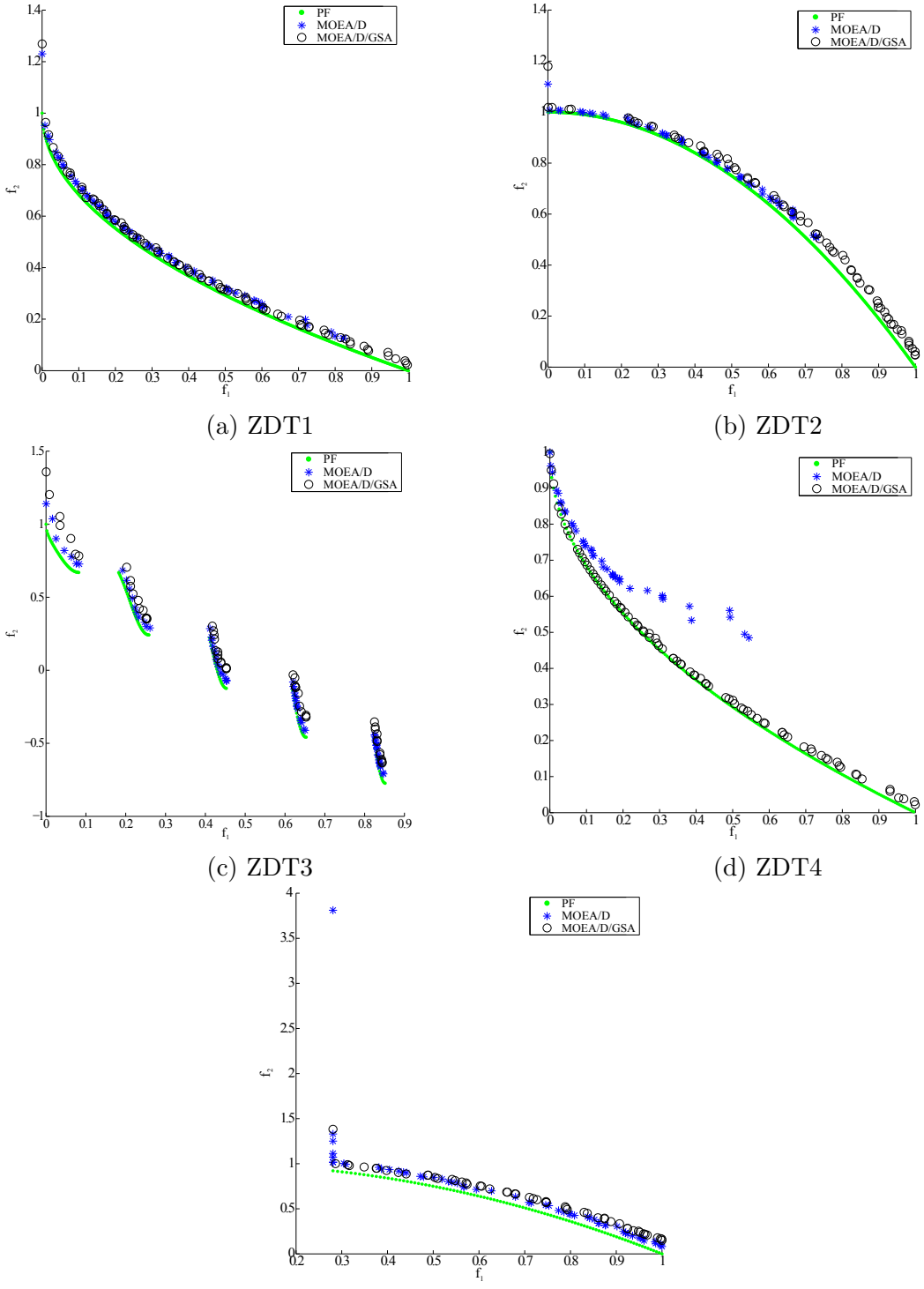
Table 4.10 presents the parameters used for each of these benchmarks.

Table 4.10: Parameters for the MOEA/D/GSA algorithm.

Parameter	**Description**	**Value**	
		ZDT	**DTLZ**
η_c	Distribution index for crossover	20	
η_m	Distribution index for mutation	20	
p_c	Crossover probability	0.95	
p_m	Mutation probability	$1/n$	
N	Number of subproblems	100	300
k	Number of objectives	2	3
r	Number of neighbors for GSA	5	
G_{max}	Maximum participation GSA	0.2	

We performed a statistical analysis of the algorithms. To create such study, we performed 30 independent runs on each test problem. Table 4.11 presents the averaged values for the indicators. The Nadir points used to compute the hypervolume indicator are $(11, 11)^T \in \mathbb{R}^2$ and $(5, 5, 5)^T \in \mathbb{R}^3$. For the new indicator W, the novel hybrid outperforms its base algorithm significantly in 11 out of the 12 cases.

Figure 4.20 presents the PF for the best individual on each of the ZDT problems. Here, it is possible to observe that in most cases, the GSA helps the algorithm to achieve a 'better' approximation of the entire Pareto Front.



(a) ZDT1

(b) ZDT2

(c) ZDT3

(d) ZDT4

(e) ZDT6

Figure 4.20: Pareto Fronts obtained for the ZDT problems by MOEA/D and MOEA/D/GSA.

Table 4.11: Averaged indicator results for unconstrained problems

Function	Hypervolume		Δ_2		$W(x)$	
	MOEA/D	MOEA/D/GSA	MOEA/D	MOEA/D/GSA	MOEA/D	MOEA/D/GSA
ZDT1	24.2531	**24.5305**	0.8066	**0.5015**	16.0381	**13.7135**
(St. Dev.)	(0.2278)	(0.0433)	(0.3926)	(0.1375)	(1.5135)	(0.1881)
ZDT2	22.1539	**24.0682**	4.2378	**0.6025**	40.2538	**20.0983**
(St. Dev.)	(1.7131)	(0.0702)	(3.9923)	(0.1714)	(12.7199)	(0.1391)
ZDT3	27.6687	**27.6976**	**0.3903**	0.4213	**30.6727**	31.1881
(St. Dev.)	(0.3988)	(0.1476)	(0.1812)	(0.1484)	(2.7405)	(0.7829)
ZDT4	23.1709	**24.1375**	2.3827	**0.9671**	24.9274	**16.1069**
(St. Dev.)	(0.5267)	(0.2766)	(1.1386)	(0.4930)	(3.0761)	(1.2013)
ZDT6	22.5058	**22.8532**	1.5651	**0.7226**	29.8828	**24.0028**
(St. Dev.)	(0.1151)	(0.0921)	(0.7848)	(0.1022)	(0.3314)	(0.3410)
DTLZ1	120.8728	**120.8730**	0.0123	**0.0094**	8.2649	**8.2592**
(St. Dev.)	(0.0007)	(0.0005)	(0.0117)	(0.0084)	(0.0103)	(0.0064)
DTLZ2	120.2100	120.2100	0.0695	**0.0694**	21.5236	**21.5234**
(St. Dev.)	(0.0001)	(0.0001)	(0.0001)	(0.0002)	(0.0001)	(0.0001)
DTLZ3	**120.1706**	120.1600	0.3464	**0.2913**	28.7256	**22.0747**
(St. Dev.)	(0.0233)	(0.0399)	(0.2708)	(0.2183)	(10.7103)	(0.2694)
DTLZ4	119.2377	**120.2100**	0.8813	**0.0616**	28.6428	**21.5237**
(St. Dev.)	(3.0711)	(0.0003)	(2.5885)	(0.0006)	(13.1820)	(0.0003)
DTLZ5	**1,319.0190**	1,318.9923	0.4415	**0.4374**	47.3239	**47.3158**
(St. Dev.)	(0.0484)	(0.1619)	(0.0087)	(0.0108)	(0.0141)	(0.0010)
DTLZ6	1,307.9325	**1,309.7949**	72.8745	**60.4012**	67.3016	**64.3943**
(St. Dev.)	(1.5831)	(1.1897)	(9.5533)	(7.5192)	(2.3840)	(1.7276)
DTLZ7	992.4941	**992.2704**	**11.8537**	19.6412	325.7372	**269.0209**
(St. Dev.)	(0.1976)	(0.1150)	(0.5595)	(0.6125)	(0.0969)	(0.0107)

IG-NSGA-II

The next set of experiments were designed in order to show that the GSA can be used to handle constrained problems. Since we are replacing the local search method of the IG-NSGA-II we adopted all the parameters from such algorithm. First, we consider some of the test problems presented in [Coello et al., 2007]. The definitions of such functions can be found in Appendix C. The maximization problems presented on the definitions were transformed into minimization problems. The constraints of the problems were also transformed into the form $g(x) \leq 0$. We stress out that all constraints are relatively easy in the sense that the feasible set is not of a highly complex structure. Table 4.12 presents the values for the parameters for the memetic algorithm.

The results obtained in Table 4.13 show that GSA can improve the results in most of the test problems according to the Δ_2 indicator. Table 4.13 also presents the number of function evaluations used as our stopping criterion. Moreover, the table also presents the Nadir point used to compute the hypervolume indicator. It is important to mention that some indicator values obtained by GSA significantly outperformed the standalone algorithm.

Taking these results into consideration, now we are in position to confirm the efficiency of GSA when it is used within a memetic strategy. As a last experiment we used the constrained CF test functions proposed in [Zhang et al., 2009b]. Such constraints can considered to be complex. We performed a similar comparison as in the previous method. We measured Δ_2 and the hypervolume at certain stage of

Table 4.12: Parameters for the IG-NSGA-II/GSA algorithm.

Parameter	**Description**	**Value**
η_c	Distribution index for crossover	20
η_m	Distribution index for mutation	20
γ_1	Crossover probability	0.9
γ_2	Mutation probability	$1/n$
N	Number of individuals	100
r	Number of neighbors for GSA	5
γ_3	Frequency of the local search	3

Table 4.13: Averaged indicator results for constrained problems

Function	**Hypervolume**		Δ_2		**Max Eval.**
	IG-NSGA-II	**IG-NSGA-II/GSA**	**IG-NSGA-II**	**IG-NSGA-II/GSA**	**(Nadir point)**
Belegundu	212.8721	**213.1354**	1.6844	**1.5900**	3,000
(std. dev.)	(0.3205)	(0.2261)	(0.1587)	(0.1030)	(12,12)
Binh(2)	10,294.0037	**10,300.6309**	0.7047	**0.6172**	3,000
(std. dev.)	(17.2207)	(9.8055)	(0.2812)	(0.1667)	(250,50)
Binh(4)	705.4772	**728.7873**	0.8863	**0.5061**	30,000
(std. dev.)	(12.2397)	(8.0167)	(0.1915)	(0.1239)	(5,7,5)
Obayashi	**22.0321**	21.9269	0.8084	**0.7792**	20,000
(std. dev.)	(0.7574)	(0.7103)	(0.2869)	(0.2772)	(5,5)
Osyczka	**59.8672**	59.5651	3.2946	**2.3881**	20,000
(std. dev.)	(1.4790)	(1.7873)	(1.9568)	(1.4040)	(30,30)
Osyczka(2)	12,780.7978	**13,701.1984**	47.8491	**30.8044**	30,000
(std. dev.)	(42.6364)	(11.0854)	(4.3380)	(1.0707)	(0,85)
Srinivas	212.8191	**213.1927**	1.4871	**1.3925**	3000
(std. dev.)	(0.3723)	(0.0714)	(0.2364)	(0.1769)	(250,50)
Tamaki	**124.3239**	124.3054	**0.1353**	0.1515	10,000
(std. dev.)	(0.0363)	(0.0252)	(0.0252)	(0.0366)	(5,5,5)
Tanaka	22.9022	**24.9672**	0.0672	**0.0468**	10,000
(std. dev.)	(0.7516)	(0.0249)	(0.0472)	(0.0100)	(5,5)
Viennet(4)	190.2843	**191.6098**	0.1015	**0.0978**	10,000
(std. dev.)	(0.6212)	(0.4552)	(0.0048)	(0.0060)	(8,-10,30)

the evolution (i.e. at $30,000$ and $50,000$ function evaluations). For the experiments we set the nadir point for hypervolume as $(5,5)^T \in \mathbb{R}^2$ and $(5,5,5)^T \in \mathbb{R}^3$. Table 4.14 presents the averaged results measured by the proposed indicators. The results are obtained taking into consideration only the feasible solutions obtained by the algorithms. By such reason CF10 is not in the statistical results since none of the algorithms obtained feasible solutions.

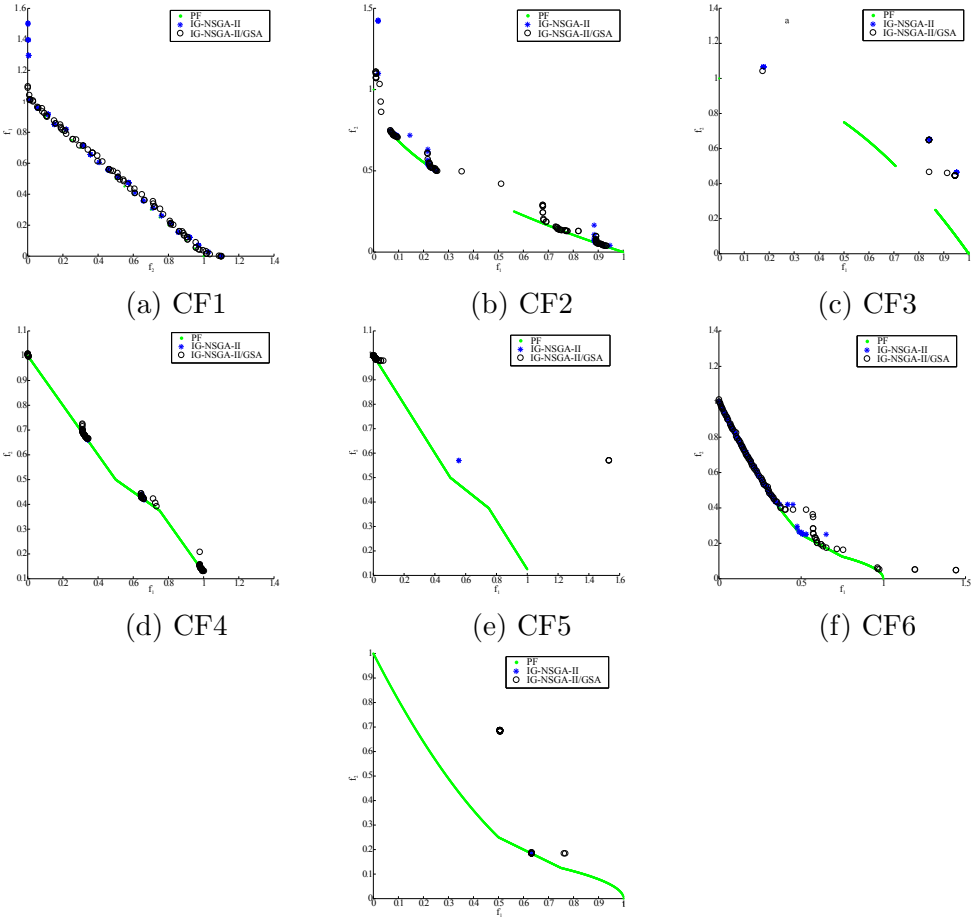
Table 4.14: Averaged indicator results for the CF test problems

Function	**Hypervolume**		Δ_2	
	IG-NSGA-II	**IG-NSGA-II/GSA**	**IG-NSGA-II**	**IG-NSGA-II/GSA**
CF1	22.6991	**22.7063**	0.3488	**0.2809**
(St. Dev.)	(0.0108)	(0.0160)	(0.1134)	(0.0457)
CF2	23.9966	**24.1846**	3.7094	**2.6267**
(St. Dev.)	(0.2304)	(0.1409)	(0.8217)	(1.1457)
CF3	21.7779	**21.9743**	8.8626	**8.6404**
(St. Dev.)	(0.7876)	(0.8831)	(1.3550)	(1.7076)
CF4	22.5691	**22.9710**	4.0301	**3.4467**
(St. Dev.)	(0.6465)	(0.5158)	(0.8088)	(0.7596)
CF5	**20.8112**	20.6415	**10.0656**	11.6749
(St. Dev.)	(1.0878)	(1.1346)	(2.1827)	(3.3711)
CF6	23.6467	**24.0427**	3.6043	**1.7815**
(St. Dev.)	(0.4994)	(0.2493)	(1.8984)	(0.5572)
CF7	20.9140	**21.3019**	16.1164	**13.6514**
(St. Dev.)	(0.6613)	(0.9006)	(5.2138)	(6.0834)
CF8	153.2641	**154.5096**	55.8651	**47.8166**
(St. Dev.)	(3.3233)	(3.2633)	(7.8279)	(10.3452)
CF9	123.3425	**123.6872**	15.1201	**14.6615**
(St. Dev.)	(0.7494)	(0.3098)	(2.8301)	(1.9893)

Figure 4.21 presents some of the Pareto Fronts obtained by the algorithms on the CF test suite.

Discussion of the results

In this section, we have discussed the applicability of the GSA in the context of multi-objective optimization. The GSA utilizes existing neighborhood information from a given candidate solution and approximates the best approximation of the gradient within the subspace of the decision variable space. Thus the computation of the search direction via GSA comes ideally for free for population based search algorithms such as evolutionary algorithms. Here, we have discussed the GSA for the case that multiple objectives have to be considered concurrently. To this end, we have



(a) CF1 (b) CF2 (c) CF3

(d) CF4 (e) CF5 (f) CF6

(g) CF7

Figure 4.21: Pareto Fronts obtained by IG-NSGA-II and IG-NSGA-II/GSA for the CF test problems.

- empirically shown that the existing neighborhood information within populations of multi-objective evolutionary algorithms is sufficient for the application of GSA,
- discussed how to approximate the Jacobian matrix at a given candidate solution x_0 via GSA,
- discussed how to adapt Lara's search direction in case inequality constraints are at hand and how to choose the step size control,
- proposed two particular GSA-based memetic algorithms,
- presented numerical results on some selected benchmark problems. In both cases, the application of the local search which comes basically for free in terms of additional function calls significantly improved the performance of the base algorithm.

Interpreting the results we can confirm that GSA can be coupled with some state-of-the-art algorithms to improve its convergence rate. In this work, we present several mechanisms to couple the algorithms. But, there exists some more work to do. For example, for the case when we do not use a scalarization function we need to improve the balancing mechanisms. For example, we can extend the balancing mechanisms presented in [LaTorre et al., 2011] for such purpose. Finally, the local search procedure requires more specializad constraint handling mechanism. For example, we can proceed by adapting the interior method points to use GSA as an approximation method.

Chapter 5

Subspace Polynomial Mutation

In most state-of-the-art MOEAs the solution of a constrained MOP is obtained only based on selection mechanisms. Such mechanisms consider the constraint violation to decide which solutions survive to the next generation of the algorithm. In this context there are few MOEAs that incorporate specialized mechanisms to generate an offspring for constrained MOPs. Moreover, in some cases the use of the same evolutionary operators for unconstrained and constrained problems is quite common. In this chapter, we propose an specialized mutation operator that handles constrained problems. The Subspace Polynomial Mutation (SPM) [Alvarado et al., 2016] is an operator that generates a new candidate solution performing an exploratory movement along the active constraints of the problem.

SPM is an operator that modifies the state-of-the-art polynomial mutation (PBX) proposed in [Deb and Goyal, 1996]. The main idea of SPM is to extend the PBX method in order to incorporate movements along the constraints into its mechanism. To present the realization of such operator we analyze some of the ideas on which such operator is based on. The first idea used by SPM is the Multi-objective Stochastic Local Search (MOSLS) presented in [Schütze et al., 2016].

5.1 Multi-objective Stochastic Local Search

The MOSLS method presents the relation that exists between a stochastic sampling in a neighborhood of a candidate solution and its movement towards and along the Pareto front. Such movements can be described using sampling around the neighborhood of the candidate solution $x_0 \in \mathbb{R}^n$. According to the samples we can describe

the expected movement of a search direction according to the position of $F(x_0)$ in the objective space. The positions described in this work are proposed in [Brown and Smith, 2005b]. Consider that there exist three different positions for a given candidate solution; the first one is when $F(x_0)$ is 'far away' from the PF. A second position is defined when $F(x_0)$ is 'near' to the PF. Finally, the last position 'in between',i.e., when the objective value $F(x_0)$ is not far neither near to the PF on unconstrained MOPs.

5.1.1 MOSLS movement according to the position of x_0

Consider a search direction $\nu \in \mathbb{R}^n$ and the Jacobian of the candidate solution $J(x_0) \in \mathbb{R}^{k \times n}$. Using such information we are in the position to analyze the relation between ν and the movement in the objective space given by $J(x_0)\nu$. As described above, such movement possesses a relation according to the existing distance between $F(x_0)$ and the PF.

$F(x_0)$ is 'far away' from PF

As described in [Brown and Smith, 2005b] when a solution is sufficiently 'far away' from the PF it has been observed that typically the gradient of x_0 points into the same direction. That is, given a vector $\bar{\mu} \in \mathbb{R}^k$ where $\bar{\mu}_i > 0$ it holds that:

$$\nabla f_i(x_0) = \bar{\mu}_i \nabla f_1(x_0), i = 1, \ldots, k, \tag{5.1}$$

where $\bar{\mu}_i$ represents the i-th entry of vector $\bar{\mu}$.

Using such property we can describe the movement given by a search direction ν as follows:

$$J(x_0)\nu = \begin{pmatrix} \nabla f_1(x_0)^T \\ \vdots \\ \nabla f_k(x_0)^T \end{pmatrix} \nu = \begin{pmatrix} \bar{\mu}_1 \nabla f_1(x_0)^T \nu \\ \vdots \\ \bar{\mu}_k \nabla f_1(x_0)^T \nu \end{pmatrix} = \nabla f_1(x_0)^T \nu \bar{\mu}. \tag{5.2}$$

In this case, the dimension of the kernel of $J(x_0)$ is $k-1$. Given such property the probability is equal to one that $J(x_0)\nu \neq 0$. Such statement indicates that for a randomly generated search direction ν one always obtains a movement in the $\bar{\mu}$

direction. Since we have selected a vector $\bar{\mu}$ where each of the entries are greater that zero, the ν vector generates either a dominated or a dominating solution. Assume that we compute a dominated candidate solution y such that:

$$y = x_0 + t\nu, \tag{5.3}$$

where $t \in \mathbb{R}_+$. In this case, we can compute a descent direction as follows:

$$\tilde{\nu} = \frac{x_0 - y}{\|x_0 - y\|_2}. \tag{5.4}$$

$F(x_0)$ is 'near' to PF

The second stage to be analyzed is the one where the candidate solution is close to the set of KKT solutions. Consider that for x_0 there exists a vector $\alpha \in \mathbb{R}^k$ such that $\sum_{i=1}^{k} \alpha_i = 1$, where $\alpha_i > 0$, $i = 1, \ldots, k$ and $J(x_0)^T\alpha = 0$. Applying the same analysis as above we proceed to describe the movement in objective space given by the search direction ν:

$$\langle J(x_0)\nu, \alpha\rangle = \langle \nu, J(x_0)^T\alpha\rangle = 0. \tag{5.5}$$

Equation (5.5) specifies that for a given search direction ν the an orthogonal movement with respect to α is performed. Hence, the movement is also orthogonal to the linearized PF because α is also orthogonal to such linearization. As we stated above, the rank of $J(x_0)$ of a KKT point is $k - 1$. Such statement confirms that we can expect with a probability one a movement along the linearized PF.

$F(x_0)$ is 'in between'

The final location to be analyzed is the one where the image of the candidate solution is on neither of both cases described above. At this position, the movements performed by ν can not be described as toward the PF or orthogonal to it. Instead, given a direction $d \in \mathbb{R}^k$ there exist a $(n - k)$ dimensional subspace $S(d) \subset \mathbb{R}^n$ such that for any ν it holds that:

$$J(x_0)\nu = d. \tag{5.6}$$

Example

To illustrate the statements described above, we present an example for the three different positions of $F(x_0)$. Thus, we propose to use the function described in Equation (3.19). In particular, for this experiment we propose the values: $n = 2$, $k = 2$ and the vectors $a_1 = (1, 1)^T$ and $a_2 = (-1, -1)^T$. The different positions for x_0 are described using three different points: the far away point using $x_0^1 = (30, -30)^T$, the in between point as $x_0^2 = (1, 1)^T$ and the near point as $x_0^3 = (0, 0)^T$. For each of the points, we generate 100 new points in the neighborhood $N(x_0)$ such that:

$$x_i = x_0 + t\nu,\ i = 1, \ldots, 100, \tag{5.7}$$

where t is randomly computed such that $t \in (0, \delta]$, where δ is a given value that defines the size of the neighborhood. Using the value of δ, the neighborhood $N(x_0)$ can be defined as follows:

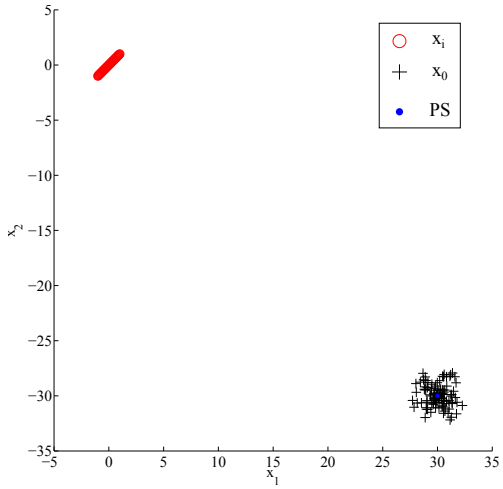
$$N(x_0) = \{x \in \mathbb{R}^n : \|x_0 - x\| \leq \delta\}. \tag{5.8}$$

The values of δ are set according to the position of $F(x_0)$: $\delta = 2.5$ for the far away case and $\delta = 0.25$ for the in between and the near cases. Algorithm 29 presents the computation of a solution in $N(x_0)$.

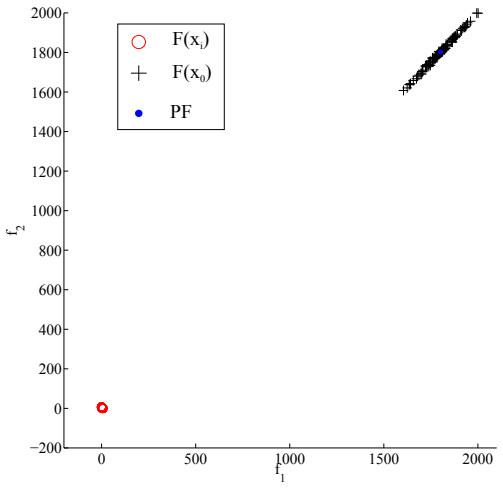
Algorithm 29 Pseudocode for computation of a candidate solution $y \in N(x_0)$

Require: Candidate solution x_0 , Neighborhood size δ
Ensure: Neighboring solution y
1: Compute a random vector $\nu \in \mathbb{R}^n$
2: Set $\nu := \frac{\nu}{\|\nu\|}$
3: Compute a random value for $t \in (0, \delta]$
4: Set $y := x_0 + t\,\nu$
5: **return** y

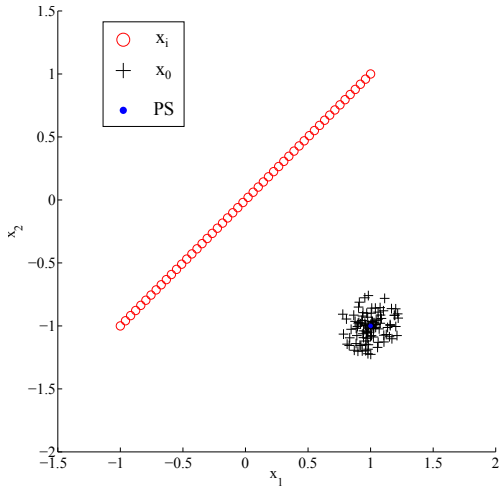
Figure 5.1 presents the results of the computations described above. The Pareto set and the Pareto front are shown for each of the positions of x_0. As expected, in case that $F(x_0)$ is far away from the PF, the randomly generated solutions are either dominating or dominated by x_0. Analogously, when a candidate solution is near to the PF it is shown that the new candidate solutions in the neighborhood of x_0 are generated along the linearized PF. Furthermore, if x_0 is in between, the generated solutions can exist in all the possible directions around $F(x_0)$.



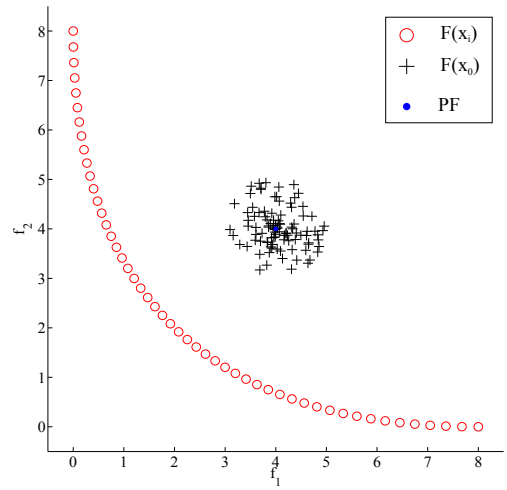
(a) Parameter space for far away position



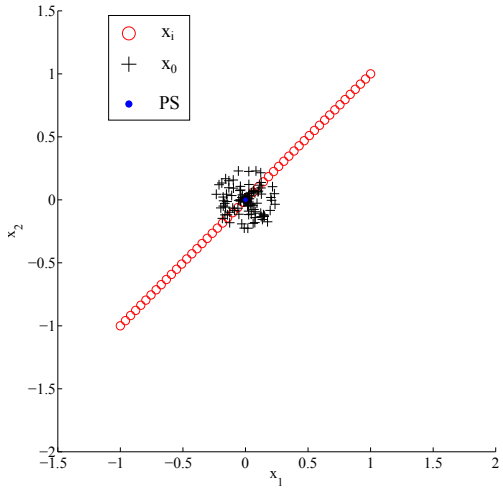
(b) Objective space for far away position



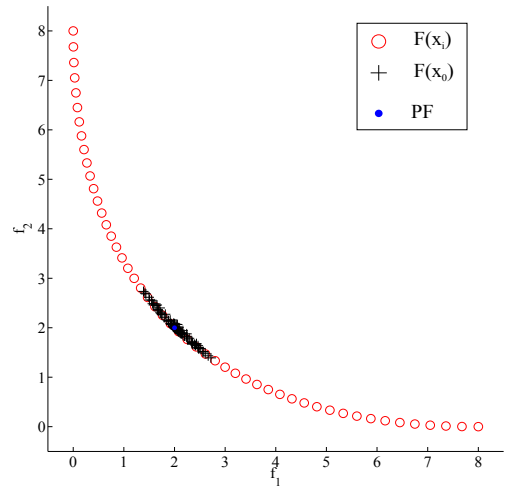
(c) Parameter space for in between position



(d) Objective space for in between position



(e) Parameter space for near position



(f) Objective space for near position

Figure 5.1: Example for MOSLS randomly generated solutions

5.1.2 MOSLS with inequality constraints

The next step to be considered is the incorporation of constraints within MOSLS. In particular, the SPM realization only considers inequality constraints so we define the MOSLS for such kind of problems. The first step for the constrained MOSLS realization is to consider a given MOP that is defined with only inequality constraints as follows:

$$\begin{array}{ll} \min_x & F(x) \\ \text{s.t.} & g_i(x) \leq 0 \end{array}. \tag{5.9}$$

Consider a KKT point x^* of Equation (5.9) where there exist l active constraints. Define a matrix $\bar{G} \in \mathbb{R}^{l \times n}$ as follows:

$$\bar{G} := \begin{pmatrix} \nabla g_1(x^*)^T \\ \vdots \\ \nabla g_l(x^*)^T \end{pmatrix}. \tag{5.10}$$

Considering a vector $\lambda \in \mathbb{R}^l$ and a vector $\alpha \in \mathbb{R}^k$ such that $\sum_{i=1}^{k} \alpha_i = 1$, $i = 1, \ldots, k$ and $\alpha_i \geq 0$ we can define the Lagrangian of the point x^* as follows:

$$\sum_{i=0}^{k} \alpha_i \nabla f(x^*) + \sum_{j=0}^{l} \lambda_j \nabla g(x^*) = J(x^*)^T \alpha + G^T \lambda = 0. \tag{5.11}$$

Now, assume that a vector $\nu \in \mathbb{R}^n$ is given such that:

$$Gv = 0. \tag{5.12}$$

Using Equation (5.12) we are in position to claim that if we perform a movement along ν it holds that such movement is also along the linearization of the PF :

$$\langle J(x^*)\nu, \alpha \rangle = \langle \nu, J(x^*)^T \alpha \rangle = -\langle \nu, G^T \lambda \rangle = -\langle G\nu, \lambda \rangle = 0. \tag{5.13}$$

To compute an orthogonal movement along the constraints, we need to find the kernel vectors of G. Such computations are divided on two different cases: when the problem has only linear inequality constraints and in the general case, i.e. for a non-linear $g(x)$.

Linear Inequality Constraints

The first case computes the kernel for a problem where the constraints are defined as:

$$g_i(x) = a_i^T x - b_i, i = 1, \ldots, l, \tag{5.14}$$

where $x \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$. Thus, the gradient of each of the constraint is defined as follows:

$$\nabla g_i(x) = a_i, i = 1, \ldots, l. \tag{5.15}$$

We start the computation of the kernel by defining a matrix $A \in \mathbb{R}^{n \times l}$ as follows:

$$A = (a_1, \ldots, a_l). \tag{5.16}$$

The next step is to obtain a vector ν orthogonal to all a_i. For this task, we propose to compute a QR decomposition of matrix A such that:

$$A = QR = (q_1, \ldots, q_l, q_{l+1}, \ldots, q_n)R. \tag{5.17}$$

By construction, $a_i \in span\{q_1, \ldots, q_l\}$ and the orthonormal basis of the kernel can be computed using a matrix $\bar{Q} = (q_{l+1}, \ldots, q_n)$. Algorithm 30 presents the pseudocode to compute direction $\nu \in span\{q_{l+1}, \ldots, q_n\}$. The algorithm uses a random vector $\bar{\nu} \in \mathbb{R}^{n-l}$ such that for each entry of the vector it holds that $\bar{\nu}_i \in (0, 1)$.

Algorithm 30 Pseudocode for MOSLS with linear inequalities

Require: Candidate solution x_0 , Neighborhood size δ
Ensure: Neighboring solution y
1: Compute a random vector $\bar{\nu}$
2: Compute matrix A as in Equation (5.16)
3: Compute QR decomposition of A
4: Set $\bar{Q} := (q_{l+1}, \ldots, q_n)$
5: Compute $\nu := \bar{Q}\,\bar{\nu}$
6: Set $\nu := \frac{\nu}{\|\nu\|}$
7: Compute a random value for $t \in (0, \delta]$
8: Set $y := x_0 + t\,\nu$
9: **return** y

One special case occurs when the constraints are defined as box constraints. In this case, each constraint is defined in the form:

$$g_i(x) = \pm x_j - b_i \leq 0,\ i = 1, \ldots, l, \tag{5.18}$$

where x_j is the j-th component of x and $b_i \in \mathbb{R}$. Analogously, the gradient of the constraints is as follows:

$$\nabla g_i(x) = \pm e_j,\ i = 1, \ldots, l, \tag{5.19}$$

where e_j represent the j-th canonical vector. Defining the indexes:

$$I = \{1, \ldots, n\} \setminus \{j_1, \ldots, j_l\}, \tag{5.20}$$

where j_l is the index of the canonical vector for the l-th gradient obtained using Equation (5.19). Hence, we can compute a neighboring solution of x_0 using Algorithm 31.

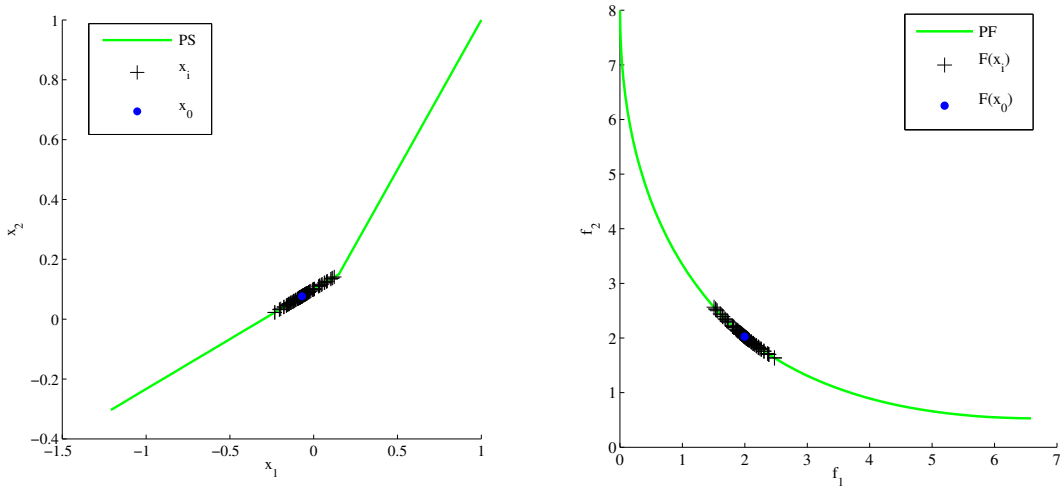
Algorithm 31 Pseudocode for MOSLS with box constraints

Require: Candidate solution x_0 , Neighborhood size δ
Ensure: Neighboring solution y
1: Compute a random solution $\bar{y}$ using Algorithm 29
2: Compute I as in Equation (5.20)
3: Set $y := x_0$
4: **for** $i = 1, \ldots, n$ **do**
5: **if** $i \notin I$ **then**
6: $y_i := \bar{y}_i$
7: **end if**
8: **end for**
9: **return** y

To illustrate the effectiveness of the MOSLS we propose the following example. Assume a MOP as the one defined in Equation (3.19). But, we add the following inequality constraint to the problem:

$$g_1(x) = \frac{1}{3}x_1 - x_2 + 0.1 \leq 0. \tag{5.21}$$

For such a problem we set the values $n = 2$, $k = 2$, $a_1 = (1, 1)^T$ and $a_2 = (-1, -1)^T$. Thus, we generate 100 points using Algorithm 31. The candidate solution is proposed as $x_0 = (-0.07, 0.0767)^T$ along with a $\delta = 0.25$. Figure 5.2 presents the results of applying MOSLS on linear inequality constrained problems. On such



(a) Parameter space

(b) Objective space

Figure 5.2: Example for MOSLS with linear inequality constraints

figure the PS and the PF show the correctness of the method. As expected, MOSLS generates points along the constraint and such points are also part of the PF.

Analogously, we apply the MOSLS method to box constrained problems. For such experiment, we use again the problem defined in Equation (3.19). Besides, for our purposes we consider the values $n = 2$, $k = 2$, $a_1 = (1, 1)^T$ and $a_2 = (-1, -1)^T$. Finally, we add the following box constraint to the problem:

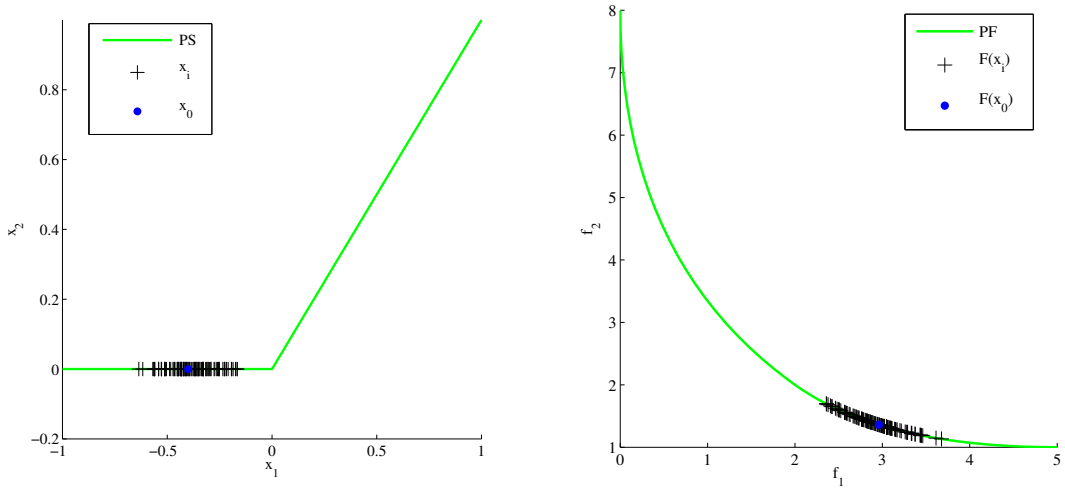
$$g_1(x) = -x_2 \leq 0. \tag{5.22}$$

Given a $x_0 = (-0.4, 0)^T$ as the initial candidate solution we generate 100 solutions using Algorithm 31. In the computation we define a value for $\delta = 0.25$. Figure 5.3 presents the PS and the PF obtained by the method. From Figure 5.3 it is clear that the the new candidates solutions are also distributed along the constraint.

General Inequality Constraints

The next step for the realization of MOSLS is to describe the procedure to handle general inequality constraints. Assume that we are given search directions $\nu_i, i = 1, \ldots, r$, along with the directional derivatives for such directions:

$$\langle \nabla g_j(x_0), \nu_i \rangle,\ i = 1, \ldots, r,\ j = 1, \ldots, l. \tag{5.23}$$



(a) Parameter space

(b) Objective space

Figure 5.3: Example for MOSLS with box constraints

Given a randomly generated vector $\sigma \in \mathbb{R}^r$ we are interested in search directions ν within the subspace $span\{\nu_1, \ldots, \nu_r\}$, such that:

$$\langle \nabla g_j(x_0), \nu_i \rangle = \langle \nabla g_j(x_0), \sum_{i=1}^{r} \sigma_i \nu_i \rangle = \sum_{i=1}^{r} \sigma_i \langle \nabla g_j(x_0), \nu_i \rangle = 0,\ i = 1, \ldots, l,\ j = 1, \ldots, r. \tag{5.24}$$

Define a matrix $V \in \mathbb{R}^{n \times r}$ as follows:

$$V = (\nu_1, \ldots, \nu_r), \tag{5.25}$$

and the matrix multiplication GV as:

$$GV = \begin{pmatrix} \langle \nabla g_1(x_0), \nu_1 \rangle & \ldots & \langle \nabla g_1(x_0), \nu_r \rangle \\ \vdots & & \vdots \\ \langle \nabla g_l(x_0), \nu_1 \rangle & \ldots & \langle \nabla g_l(x_0), \nu_r \rangle \end{pmatrix}. \tag{5.26}$$

Using the definition for GV we are now in the position to compute an orthonormal basis. Next, we compute a search direction ν in the subspace of $\nu_i, i = 1, \ldots, r$. Assuming that $r > l$ it holds that:

$$GV\sigma = 0. \tag{5.27}$$

Hence, we are interested in the computation of the kernel of GV. Using the QR

factorization we can compute the orthonormal basis:

$$(GV)^T = QR = (q_1, \ldots, q_l, q_{l+1}, q_n)R. \tag{5.28}$$

After we compute the QR decomposition of matrix GV we can define a matrix

$$\tilde{Q} = (q_{l+1}, \ldots, q_n) \in \mathbb{R}^{n \times (r-l)}, \tag{5.29}$$

such that the new search direction is defined as:

$$\nu = \tilde{Q}\sigma. \tag{5.30}$$

In the formulation described above we assumed that certain pieces of information were given, such as the ν_i search directions and the values of the directional derivatives. As we previously did with other methods we can approximate such information using neighboring points of x_0. Given $x_i, i = 1, \ldots, r$, we approximate the search directions using Equation (3.13). Analogously, if we have the constraint values of the x_i points we can approximate the directional derivatives of Equation (5.26) using Equation (4.61).

Algorithm 32 presents the pseudocode to compute a search direction ν using the mechanism described above. In principle, such realization is more expensive in comparison with the linear inequalities algorithm. Such cost increases since we need to compute several points and their constraints values before constructing the ν direction. The algorithm uses the same neighborhood structure as the one presented in Section 4.4.1. In the algorithm, $r \in \mathbb{N}$ represents the number of search direction to be considered in the computation. Meanwhile, $\delta \in \mathbb{R}$ is a numerical value that defines the maximal 2-norm of any individual in the neighborhood.

We introduce an example to illustrate the effectiveness of the method. Consider the bi-objective problem of Equation (3.19) with $n = 2$, $k = 2$, $a_1 = (1, 1)^T$ and $a_2 = (-1, -1)^T$. Now, we add a constraint to the problem in the form:

$$g_1(x) = -x_1^2 + x_2 + 1 \leq 0. \tag{5.31}$$

For the problem described above we generate 100 points in the neighborhood using two different candidate solutions: $x^1 = (0.20534, -0.95659)^T$ and $x^2 = (-0.90317, -0.18197)^T$. For each candidate solution we apply the method described in Algorithm

Algorithm 32 Pseudocode for MOSLS with general inequality constraints

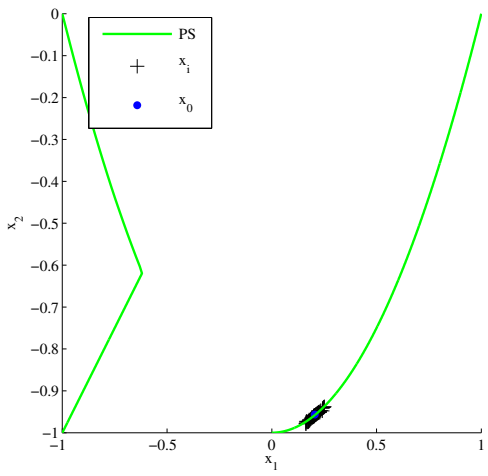
Require: Candidate solution x_0 , Neighborhood $\mathcal{N}$ of x_0, Neighborhood size δ
Ensure: Neighboring solution y

Set $GV := \emptyset$
for $i = 1, \ldots, r$ **do**
  Set $x^a := \mathcal{N} \rightarrow x^i$
  Set $g^0 := g(x_0) \in \mathbb{R}^m$
  Set $g^a = \mathcal{N} \rightarrow g^i$
  Set $j := 1$
  for $s = 1, \ldots, m$ **do**
    if g_s^0 is active **then**
      Compute $\tilde{m} = \frac{g_s^a - g_s^0}{\|x_a - x_0\|_2}$
      Set the entry of GV_{ij} as m
      Set $j := j + 1$
    end if
  end for
end for
Compute QR decomposition of GV
Compute matrix $\tilde{Q}$ as in Equation (5.29)
Compute a random vector σ
Compute ν as in Equation (5.30)
Set $\nu := \frac{\nu}{\|\nu\|}$
Compute a random value for $t \in (0, \delta]$
Set $y := x_0 + t\,\nu$
return y

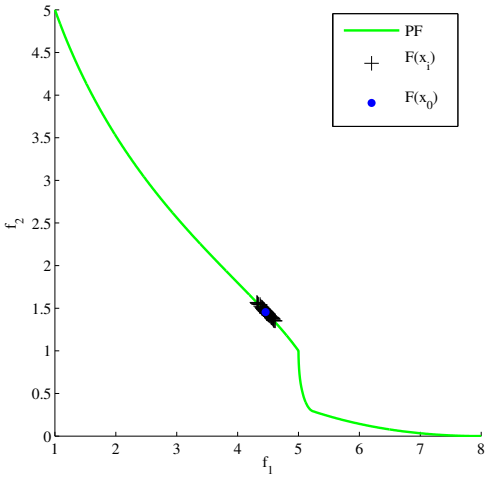
32. For this purpose, we set the value of $\delta = 0.1$. Figure 5.4 presents the neighboring solutions generated this way. From such figures it is visible that the solutions are generated in the linearization of the constraint. Hence, if the value δ increases the generated neighboring solution gets far away from the constraint and, therefore, a correction step would be desirable.

5.1.3 Main idea of SPM

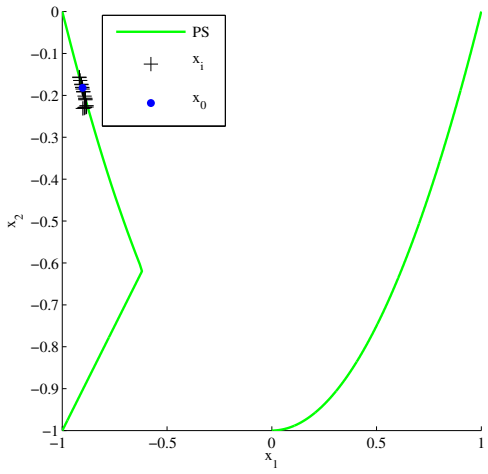
As mentioned above SPM is a modified version of the PBX mutation operator. The main idea of SPM is to use all the properties observed by MOSLS to perform a movement along the active constraints. Such movement, in principle, generates a new feasible candidate solution that belongs to the PF (or at least it's close enough to it). The application of the new mutation operator can lead to generate solutions along



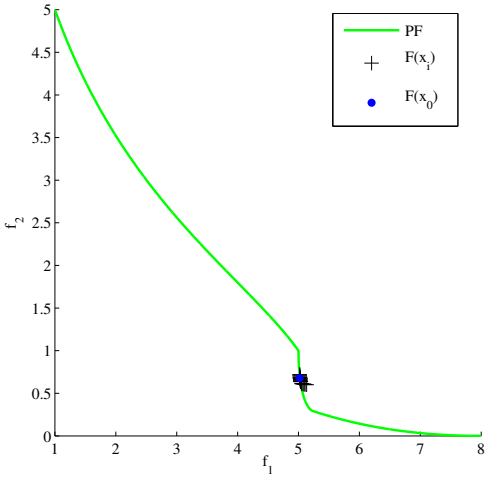
(a) Parameter space for x^1



(b) Objective space for $F(x^1)$



(c) Parameter space for x^2



(d) Objective space for $F(x^2)$

Figure 5.4: Example for MOSLS with general inequality constraints

the PF, which, in consequence, gives us a better approximation of the PF.

First, we will describe the PBX mutation operator as in [Deb and Goyal, 1996]. PBX mutates each component of the candidate solution using a polynomial probability distribution. For each entry of the candidate solution $x_i, i = 1, \ldots, n$, a random value $u \in (0, 1)$ is generated. Next, a perturbation factor δ is computed using a distribution index n_m:

$$\delta_q = \begin{cases} (2u + (1-2u)(1-\delta)^{n_m+1})^{\frac{1}{n_m+1}} - 1, \; u \leq 0.5 \\ 1 - ((2(1-u) + 2(u-0.5)(1-\delta)^{n_m+1})^{\frac{1}{n_m+1}}, \; u > 0.5 \end{cases}, \tag{5.32}$$

where δ uses the upper bound x^u and the lower bound x^l of the i-th entry of x into its computation:

$$\delta = \frac{\max((x_i - x^l), (x_i - x^u))}{x^u - x^l}. \tag{5.33}$$

Finally, each entry of the mutated vector can be computed as follows:

$$y_i = x_i + \delta_q(x^u - x^l). \tag{5.34}$$

Opposite to the PBX operator, the SPM method uses a direction ν that perturbates the candidate solution x, but using the idea of the PBX we incorporate the boundaries of the variables into the computation. Such realization ensures that the new candidate solution generated by the SPM operator remains in the feasible subspace. Hence, consider that a $\nu \neq 0 \in \mathbb{R}^n$ is given. Then, the new mutated vector y is computed as $y = x + t\nu$. Algorithm 33 presents the computation of the t_m value. Such value is used as the upper limit for t such that $t \in (0, t_m)$. The algorithm incorporates the boundary limits of the variables into the procedure. We introduced a parameter $\gamma = 0.1$ such that it controls the maximal value of the step size t_m.

We proceed to define how the ν search direction is computed. Thus, we use the computations presented on MOSLS to generate such a vector. In this case, we are solving a MOP with only inequality constraints and the candidate solution x_0 has l active constraints. In particular, we are concerned in the computation of the orthonormal base $\mathcal{O}$. To compute such a base we divide the computation into the same cases as for MOSLS: we consider that we either (i) have only linear inequality con-

Algorithm 33 Pseudocode for computation of maximal step size t_m

Require: Candidate solution x , Search direction ν, Step size control value t_m
Ensure: Maximal step size t_m

```
Set t := ∞
for i = 1, ..., n do
    if ν_i > 0 then
        Compute t_a := (x^u − ν_i) / ν_i
    else if ν_i < 0 then
        Compute t_a := (ν_i − x^l) / ν_i
    else
        Set t_a := ∞
    end if
    if t_a < t then
        Set t_m := t_a
    end if
end for
return t_m
```

straints or (ii) the general case.

Algorithm 34 presents the pseudocode to compute the orthonormal base for linear inequality constraints.

Algorithm 34 Pseudocode for orthonormal base of SPM with linear inequality constraints

Require: Candidate solution x_0
Ensure: Orthonormal base $\mathcal{O}$

```
Compute matrix A as in Equation (5.16)
Compute QR decomposition of A
Set O := (q_{l+1}, ..., q_n)
return O
```

The second stage of the SPM realization consists in computing the orthonormal base for general inequality constraints. Algorithm 35 presents the pseudocode to compute such a basis.

Finally, using the orthonormal basis described in the algorithms we are in position to define the SPM mutation operator. Algorithm 36 presents the pseudocode for such an operator. In the algorithm, we compute the orthonormal basis for each type of inequality constraints. Thus, if we have information about the problem and also we

Algorithm 35 Pseudocode for orthonormal base of SPM with general inequality constraints

Require: Candidate solution x_0 , Neighborhood $\mathcal{N}$ of x_0
Ensure: Orthonormal base $\mathcal{O}$
Set $GV := \emptyset$
for $i = 1, \ldots, r$ **do**
  Set $x^a := \mathcal{N} \rightarrow x^i$
  Set $g^0 := g(x_0) \in \mathbb{R}^m$
  Set $g^a = \mathcal{N} \rightarrow g^i$
  Set $j := 1$
  for $s = 1, \ldots, m$ **do**
    if g_s^0 is active **then**
      Compute $\tilde{m} = \frac{g_s^a - g_s^0}{\|x_a - x_0\|_2}$
      Set the entry of GV_{ij} as m
      Set $j := j + 1$
    end if
  end for
end for
Compute QR decomposition of GV
Compute matrix $\mathcal{O} := (q_{l+1}, \ldots, q_n)$
return $\mathcal{O}$

know that the problem only has linear inequality constraints, we compute the basis as in Algorithm 34. But, if we do not have information of the problem or the inequality constraints are not linear we proceed to compute the basis $\mathcal{O}$ as in Algorithm 35. In the algorithm, vectors $o_i, i = 1, \ldots, (n - l)$ representing the column vector of the orthonormal basis $\mathcal{O}$ are introduced.

5.1.4 NSGA-II/SPM

In order to illustrate the effectiveness of the new mutation operator it is necessary to couple it with some state-of-the-art MOEA. In particular, as a first approach for the operator, we have selected the NSGA-II [Deb et al., 2002a] as our base algorithm. We selected this algorithm because the NSGA-II already incorporates a selection mechanism to handle constrained problems. The modifications are only performed over the offspring creation algorithm of the NSGA-II. In particular, the modifications of the algorithm only affect the mutation operators. Other components of the offspring generation algorithm remain intact (e.g., it adopts Simulated Binary Crossover (SBX) [Deb and Goyal, 1996]). Algorithm 37 presents the pseudocode of the offspring gen-

Algorithm 36 Pseudocode for the SPM mutation operator

Require: Candidate solution x_0 , Neighborhood $\mathcal{N}$ of x_0
Ensure: Mutated vector y
Compute $g_0 := g(x_0)$
if Active constraints from g_0 are linear **then**
  Compute $\mathcal{O}$ using Algorithm 34
else
  Compute $\mathcal{O}$ using Algorithm 35
end if
Set $\nu := 0 \in \mathbb{R}^n$
for $i = 1, \ldots, (n - l)$ **do**
  Compute random value $u \in (0, 1)$
  if $u \leq 0.5$ **then**
    $\nu := \nu + o_1$
  else
    $\nu := \nu - o_1$
  end if
end for
Set $\nu := \frac{\nu}{\|\nu\|}$
Compute maximal step size t_m using Algorithm (33)
Compute random value of $t \in (0, t_m]$
Compute $y := x_0 + t\nu$
return y

eration for the NSGA-II/SPM. Opposite to the PBX mutation, the application of the SPM does not depends on a mutation probability p_m. Instead, the SPM is applied on 'useful' moments. That is, the SPM operator is only applied when we detect that there exists at least one active constraint. In this case, we consider that a constraint its active when its value its above certain threshold $\epsilon = -1e - 7$. After we detect at least one active constraint, there exists a probability of 0.5 that the SPM operator generate a new mutated vector. Given the neighborhood structure requirement of the SPM, we compute such information using Algorithm 37.

5.1.5 Numerical results

Finally, to illustrate the advantages of the SPM operator we perform several experiments comparing NSGA-II/SPM with respect to NSGA-II. Table 5.1 presents the parameters used for such experiments.

Algorithm 37 Pseudocode for offspring generation in NSGA-II

Require: Candidate solution x
Ensure: Children solution y
Generate y^c using SBX crossover
if there exists an active constraint on x_0 **then**
  Generate a random value $u \in (0, 1)$
  if $u \leq p_m$ **then**
    Generate y using SPM
  else
    Set $y := y^c$
  end if
else
  Generate a random value $u \in (0, 1)$
  if $u \leq p_m$ **then**
    Generate y using PBX mutation
  else
    Set $y := y^c$
  end if
end if
return y

Table 5.1: Parameters for the NSGA-II/SPM experiments

Parameter	**Value**
Population size N	100
Crossover probability p_c	0.95
Mutation probability p_m	0.01
Distribution index for crossover n_x	20
Distribution index for mutation n_m	20
r	$\min(l, 5)$
δ	0.1

The comparison of the algorithms is performed using the Δ_2 indicator proposed in [Schütze et al., 2012]. The performance of the algorithms is measured adopting several state-of-the-art constrained problems: Belegundu, Binh(1-4), Obayashi, Osyczka(2), Srinivas, Tanaka and Tamaki (for the definition please refer to Appendix C). Besides, to show the effectiveness of the method on linear inequality constrained problems we modified the state-of-the-art ZDT benchmark [Zitzler et al., 2000a]. For the benchmark functions we added a single linear inequality constraint defined as follows:

$$g_1(x) = -x_1 - x_2 + 1.0 \leq 0. \tag{5.35}$$

The modified problems will be referred to as $ZDT - C$. The results for all the problems are obtained using 30 independent runs. Next, a comparison is performed using the Δ_2 values of the algorithms in two different convergence stages: 'middle' term and 'long' term. The middle term is measured when the algorithms reach $15,000$ function calls. Analogously, the long term is define as the stage when the algorithm has spent $50,000$ function calls. At each stage, we consider all the individuals in the population. Next, we remove the infeasible solutions along with the dominated ones. After we filter out the solutions, we compute the Δ_2 values such that they can be used to perform statistical comparisons between the algorithms. Tables 5.2 and 5.3 present the averaged results obtained for each algorithm. The tables present several statistical values such as the average, mean, the minimal value and the maximal value obtained for each problem. Besides, we performed the Wilcoxon rank sum test to define if there exists a significant improvement given by SPM. The results of the Wilcoxon test are presented using the P-value column. The winner for each value is highlighted only if the results are statistical significant. That is, a value of $p \leq 0.05$ is obtained.

To illustrate the results obtained by the tables above we present the PF of the best run for each problem. Figures 5.5 to 5.8 present the obtained PF of the best run at each stage of the algorithms ($15,000$ and $50,000$ respectively). Only the function where there exists a significant winner are presented.

Finally, Figures 5.9 to 5.10 present the averaged convergence plots of the Δ_2 value for each of the problems described above.

Discussion of the results

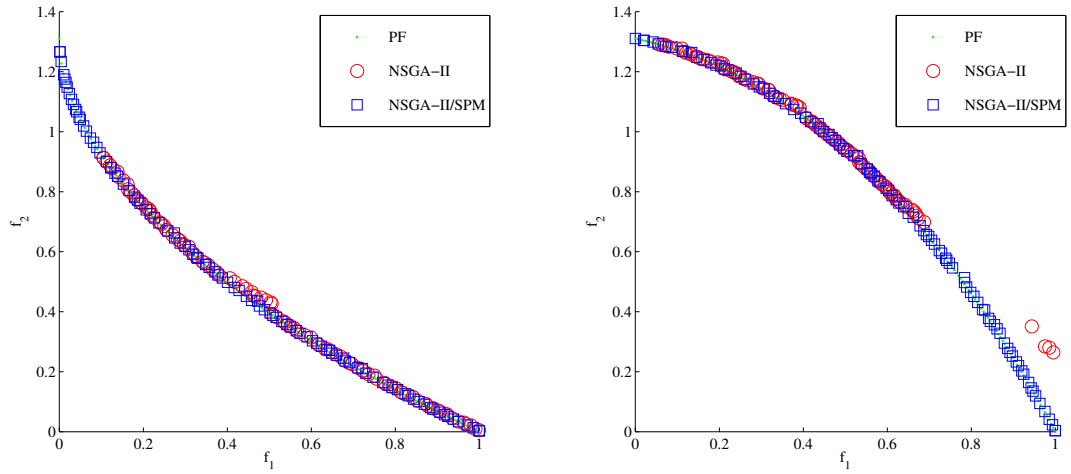
The results demonstrate that SPM is a promisory method, in particular, if we consider the problems where linear inequality constraints are given. In such a case, the computational cost to compute ν is dramatically reduced. Such advantage can be easily visualized in the results obtained on the ZDT-C benchmark. For such problems the results show a significant improvement of the NSGA-II algorithm in four out

Problem	**Parameter**	Algorithm		**P-value**
		NSGA-II	**NSGA-II / SPM**	
ZDT1-C	Average	0.027649	**0.007403**	0.000183
	Max	0.096856	0.007826	
	Min	0.010161	0.007107	
	Std. Dev.	0.025584	0.000238	
ZDT2-C	Average	0.061387	**0.007642**	0.000143
	Max	0.258192	0.008269	
	Min	0.013764	0.007294	
	Std. Dev.	0.079515	0.000302	
ZDT3-C	Average	0.040732	**0.013064**	0.001315
	Max	0.062510	0.013961	
	Min	0.013163	0.011953	
	Std. Dev.	0.022034	0.000672	
ZDT4-C	Average	1.403286	1.431395	0.909722
	Max	2.582335	2.877414	
	Min	0.285733	0.839480	
	Std. Dev.	0.902602	0.686087	
ZDT6-C	Average	0.464149	**0.060621**	0.000183
	Max	1.039935	0.110679	
	Min	0.175954	0.036111	
	Std. Dev.	0.264507	0.026668	
Binh(2)	Average	5.336551	5.336919	1.000000
	Max	5.357580	5.366246	
	Min	5.312041	5.308829	
	Std. Dev.	0.014709	0.019269	
Binh(4)	Average	3.141843	**0.223570**	0.000769
	Max	9.384341	0.585141	
	Min	0.190107	0.165311	
	Std. Dev.	3.105293	0.128207	
Obayashi	Average	0.041570	**0.006829**	0.000183
	Max	0.063213	0.007239	
	Min	0.029827	0.006410	
	Std. Dev.	0.011323	0.000254	
Osyczka(2)	Average	48.017675	**13.245735**	0.014019
	Max	103.558534	75.732281	
	Min	3.157917	1.501807	
	Std. Dev.	39.196448	23.992073	
Srinivas	Average	1.546977	1.534552	0.623176
	Max	1.692464	1.764385	
	Min	1.436139	1.383971	
	Std. Dev.	0.090367	0.114916	
Tanaka	Average	0.058619	**0.006182**	0.000183
	Max	0.090148	0.006537	
	Min	0.027259	0.005670	
	Std. Dev.	0.019901	0.000269	
Tamaki	Average	0.293233	**0.109924**	0.000183
	Max	1.528184	0.117326	
	Min	0.121339	0.102847	
	Std. Dev.	0.434693	0.004131	

Table 5.2: Δ_2 results of $NSGA-II/SPM$ for $15,000$ function calls.

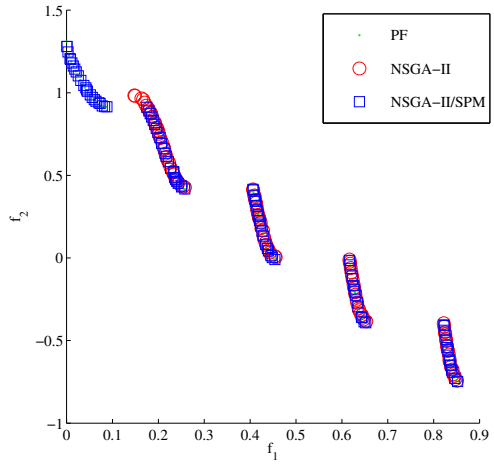
Problem	**Parameter**	Algorithm		**P-value**
		NSGA-II	**NSGA-II / SPM**	
ZDT1-C	Average	0.016852	**0.007450**	0.000183
	Max	0.035176	0.007910	
	Min	0.009376	0.006841	
	Std. Dev.	0.009244	0.000327	
ZDT2-C	Average	0.011455	**0.007540**	0.000173
	Max	0.018135	0.008517	
	Min	0.008988	0.006813	
	Std. Dev.	0.002654	0.000531	
ZDT3-C	Average	0.032820	**0.013384**	0.011330
	Max	0.060173	0.014210	
	Min	0.012957	0.012605	
	Std. Dev.	0.022977	0.000560	
ZDT4-C	Average	0.212473	0.073799	0.472676
	Max	1.708808	0.312584	
	Min	0.010483	0.010676	
	Std. Dev.	0.534402	0.098566	
ZDT6-C	Average	0.128530	**0.040937**	0.000769
	Max	0.362475	0.097499	
	Min	0.075981	0.030011	
	Std. Dev.	0.085411	0.020288	
Binh(2)	Average	5.342034	5.345931	0.791337
	Max	5.382925	5.393762	
	Min	5.302956	5.310413	
	Std. Dev.	0.026971	0.030965	
Binh(4)	Average	2.444430	**0.180125**	0.011330
	Max	9.860060	0.208120	
	Min	0.167861	0.156663	
	Std. Dev.	3.284948	0.016583	
Obayashi	Average	0.026787	**0.007400**	0.000183
	Max	0.056089	0.007699	
	Min	0.012673	0.007011	
	Std. Dev.	0.014353	0.000211	
Osyczka(2)	Average	64.474206	**12.355404**	0.064022
	Max	162.504424	76.659919	
	Min	1.494405	1.396753	
	Std. Dev.	62.590117	23.982527	
Srinivas	Average	1.554161	1.576505	0.623176
	Max	1.877404	1.884739	
	Min	1.386098	1.417530	
	Std. Dev.	0.158957	0.153774	
Tanaka	Average	0.016200	**0.006440**	0.000183
	Max	0.028136	0.007060	
	Min	0.009353	0.005995	
	Std. Dev.	0.005729	0.000354	
Tamaki	Average	0.257510	**0.108770**	0.011330
	Max	1.528184	0.113186	
	Min	0.104842	0.101753	
	Std. Dev.	0.446527	0.004243	

Table 5.3: Δ_2 results of $NSGA-II/SPM$ for $50,000$ function calls.

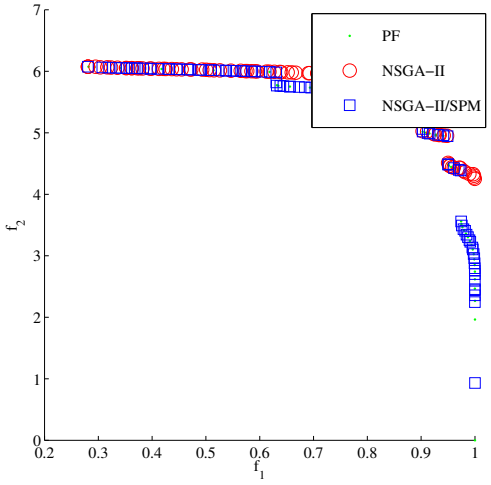


(a) ZDT1-C

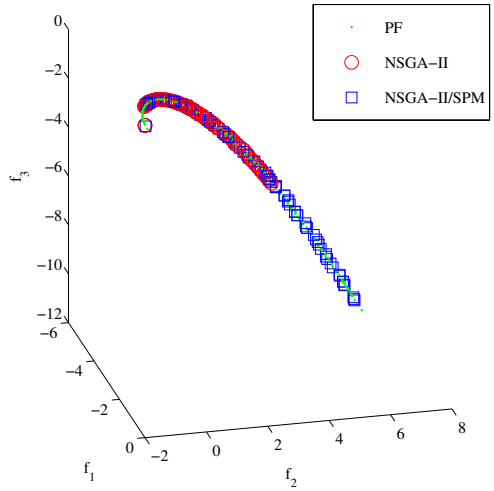
(b) ZDT2-C



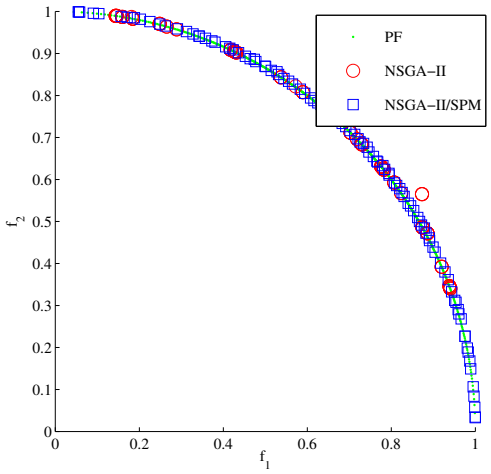
(c) ZDT3-C



(d) ZDT6-C

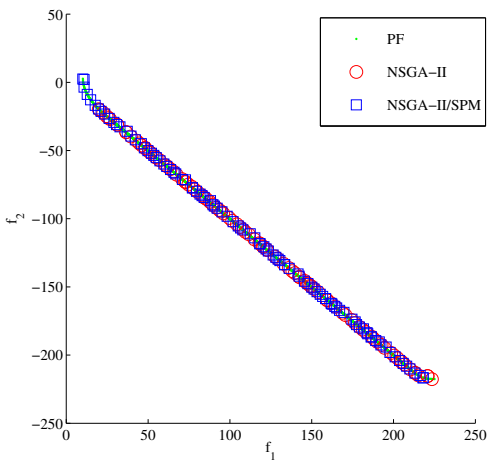


(e) Binh(4)

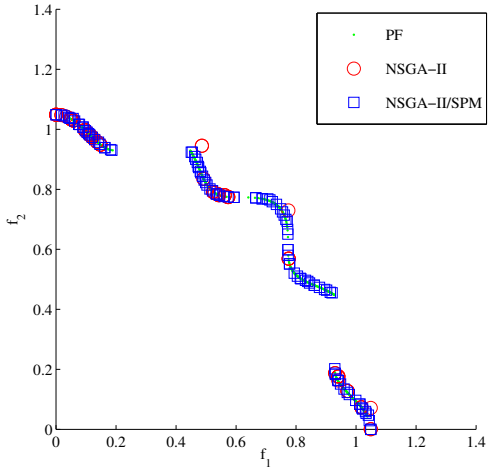


(f) Obayashi

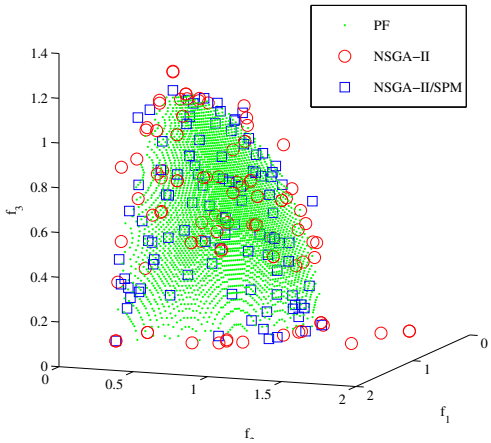
Figure 5.5: Comparison of the SPM PF at 15, 000 function calls



(a) Srinivas

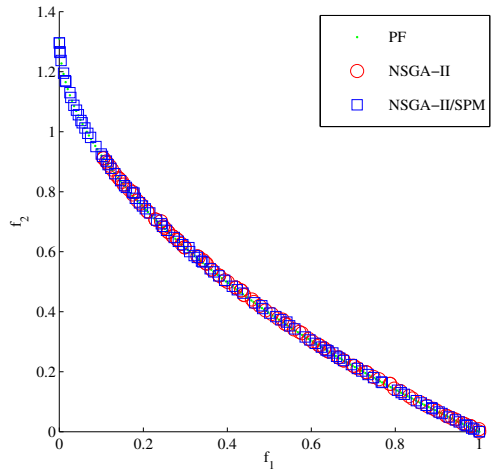


(b) Tanaka

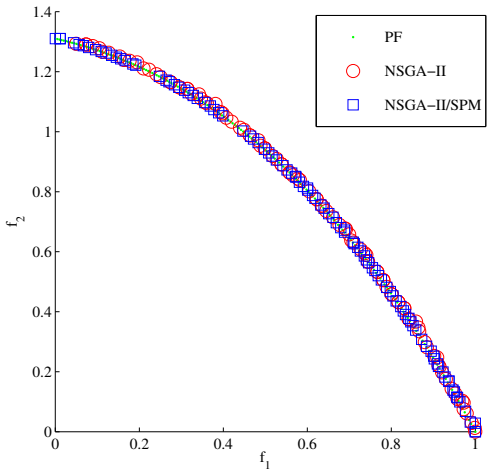


(c) Tamaki

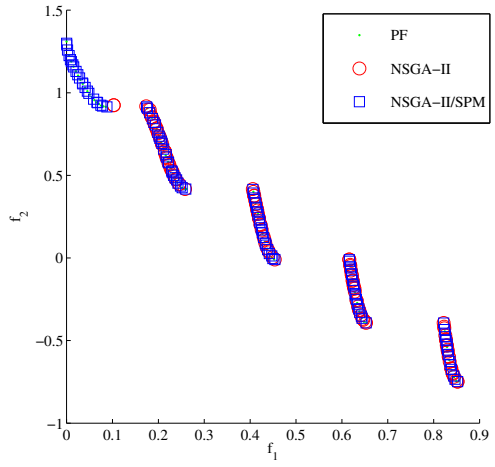
Figure 5.6: Cont'd Comparison of the SPM PF at 15, 000 function calls



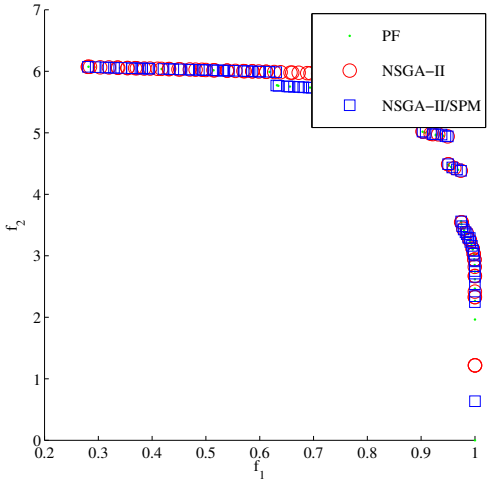
(a) ZDT1-C



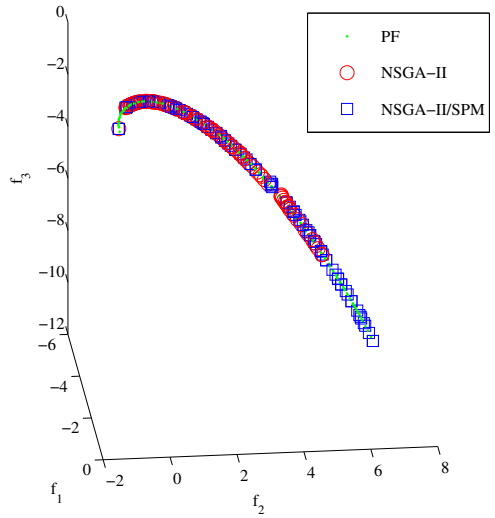
(b) ZDT2-C



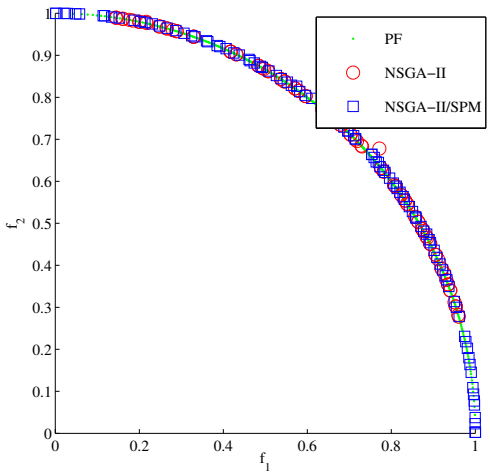
(c) ZDT3-C



(d) ZDT6-C

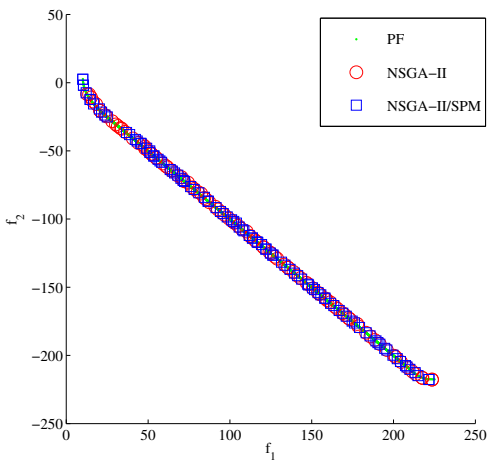


(e) Binh(4)

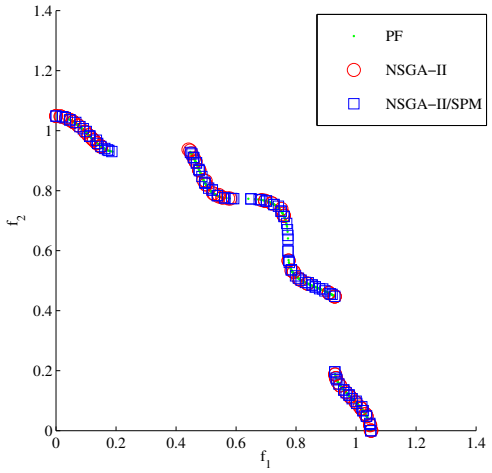


(f) Obayashi

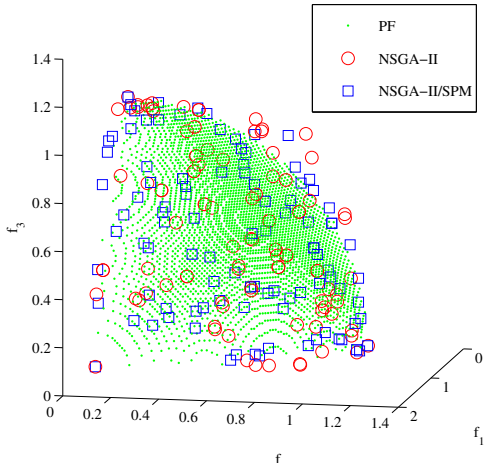
Figure 5.7: Comparison of the SPM PF at 50, 000 function calls



(a) Srinivas

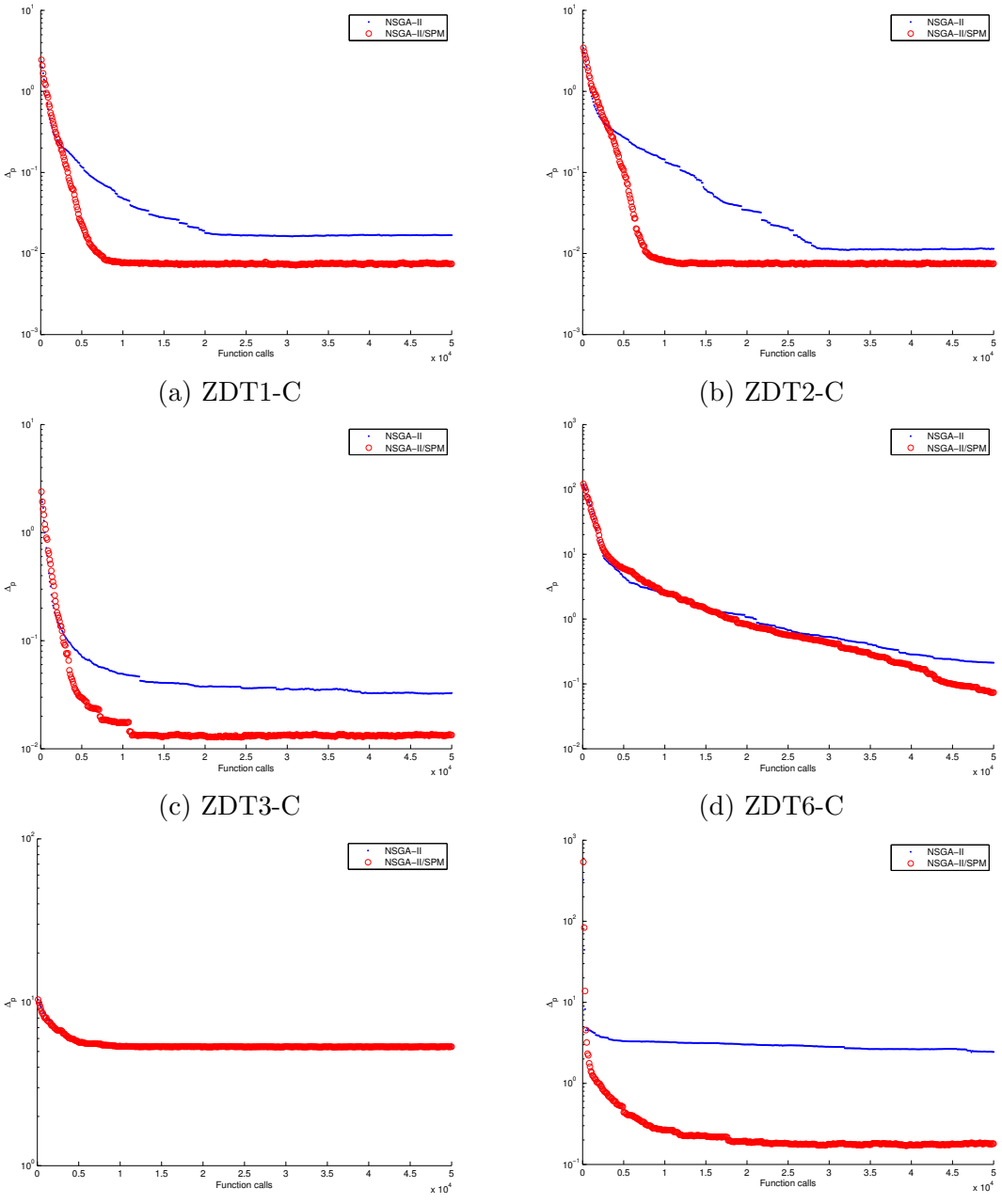


(b) Tanaka



(c) Tamaki

Figure 5.8: Cont'd Comparison of the SPM PF at 50, 000 function calls



(a) ZDT1-C

(b) ZDT2-C

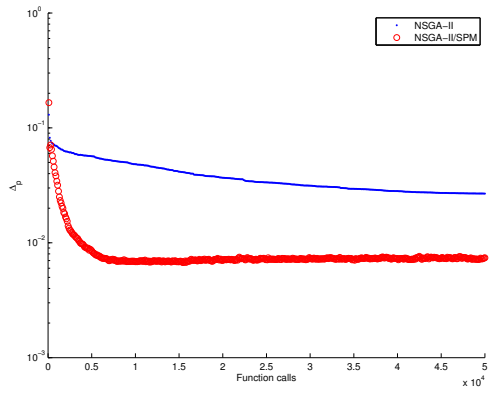
(c) ZDT3-C

(d) ZDT6-C

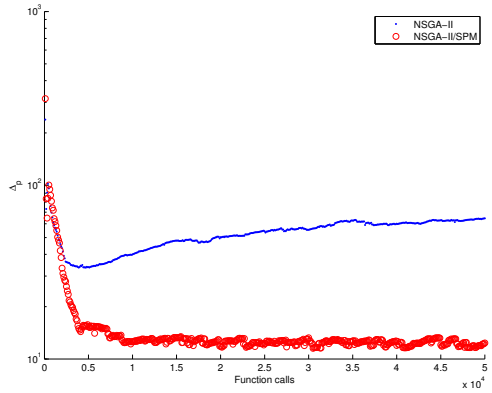
(e) Binh(2)

(f) Binh(4)

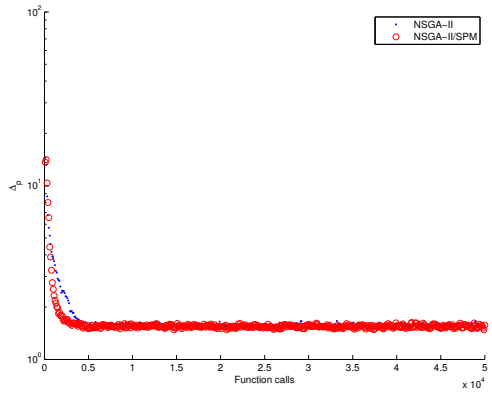
Figure 5.9: Convergence plot of the variants of the NSGA-II



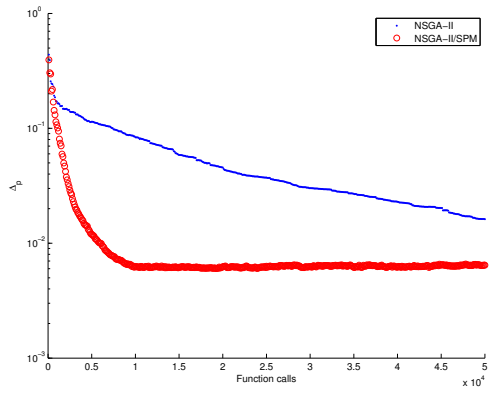
(a) Obayashi



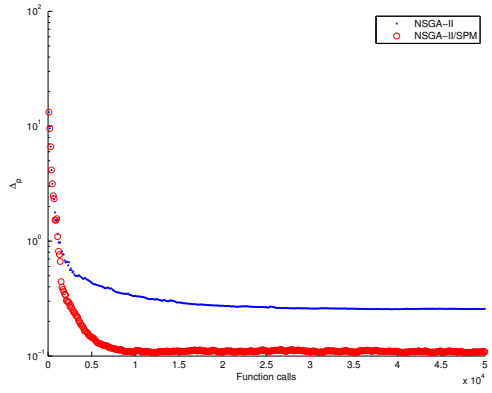
(b) Osyczka(2)



(c) Srinivas



(d) Tanaka



(e) Tamaki

Figure 5.10: Cont'd Convergence plot of the variants of the NSGA-II

of five problems.

The second type of problems used for this set of experiments where the ones with nonlinear inequality constraints. The statistical results show that the SPM operator does not lose on any of the adopted problems. Unfortunately, we expected that the number of significant winnings by the method was greater. Considering that the information required for the computation of direction ν comes for 'free' one could expect a great advantage of the operator. Unfortunately, there existed several points that modified the performance of the SPM. The first point to think about is the size of the δ value. In our computations, we fixed its value but the experiments showed that the values of δ require a mechanism to adapt it according to each problem.

The second point to be considered is how to detect the moment when the SPM has to be applied. If we remember the formulation of the SPM, we considered that a movement along a constraint leads us to a movement along the PF. Unfortunately, in practice, it is possible that this is not correct. For example, consider the Tanaka where the NSGA-II/SPM presents a great advantage over the NSGA-II. In such a problem the constraint defines the PF so the original proposition of the SPM is accomplished. But, there exist problems where the PF is not defined by the constraints. For such reasons, it is necessary to improve the detection mechanism of solutions where SPM needs to be applied.

To this end, the results obtained in these experiments show the advantage of applying the SPM operator. Using SPM, we accelerated the convergence rate in 5 of the test problem adopted. Moreover, in principle such improvement comes with no additional cost in terms of function calls. Finally, if we perform specialization changes (such as a correction step or the ones described above) to the operator it is quite possible that we can obtain an improvement of the statistical results obtained in this set of experiments.

Chapter 6

Conclusions and Future Work

In this thesis we described three different methods: the Discrete Directed Search, the Gradient Subspace Approximation algorithm and the Subspace Polynomial Mutation operator. These three algorithms presented several advantages when they were coupled with evolutionary algorithms. Such advantages included the increase of the convergence rate of its base evolutionary strategies. Furthermore, none of these three methods requires gradient information. Instead, they use information from the neighborhood of the given population to compute a new candidate solution.

Discrete Directed Search

The Discrete Directed Search (as the original Directed Search) is a method that uses a given direction $d \in \mathbb{R}^k$ to steer the search in objective space. DDS is a modification of the DS method which is gradient-free. DDS does not calculate an approximation of the gradient, instead, it computes an alternative direction ν using neighborhood information. Using such kind of information, DDS does not require any extra function calls to compute a search direction ν. This feature makes it a perfect candidate for coupling it with population-based strategies such as evolutionary algorithms.

According to the experiments, one of the decisions that highly affects the DDS method is the proper choice of r. It was shown that when more information was incorporated into the matrix $\mathcal{F}$, a more greedy step was obtained as a result. Unfortunately, the experiments also showed that if the number of individuals increased, it is possible that the system of equations becomes unstable. Another drawback of DDS is that it required approximately $0.4 \cdot n$ neighbors in order to construct a 'good'

approximation of the search direction ν, where n is the dimensionality of the domain.

We used the DDS method to construct a memetic algorithm with MOEA/D as our base algorithm. We performed a comparison between MOEA/D and two memetic algorithms. Such a comparison was performed using the Hypervolume and the Δ_2 indicators. The results showed that the memetic algorithms presented better results in comparison with the standalone approach. It is important to mention that we obtained competitive results in comparison with those obtained by the DS method. This illustrates the promising applicability of the DDS method for a problems where DS can not be applied due to the lack of gradient information.

As part of the future work for the DDS method, it is necessary to find a specialized step size control. In the experiments, the step size control consisted in a method that reduced the step size of the new computed solution until a non dominated one is found. Such condition caused that in some cases even when the direction ν steered the search in the d direction, the value of t gave a poor improvement of the candidate solution. Hence, it finds a value of t such that the maximal decay in direction d is desirable.

Another possible improvement for the MOEA/D/DDS is to find a more suitable technique that balances the resources between the local search and the evolutionary strategy. The balancing mechanism proposed in this work only applies the local search to a certain number of individuals on certain specific generations. It is clear that such mechanism is not optimal. An improvement in the balance mechanism could lead to the increase of the quality of the computed solutions. For example, if the mechanism detects that a good candidate solution is generated it can decide if the local search is applied more often.

Gradient Subspace Approximation

The GSA method was presented as a local search technique that uses the information of a population-based strategy to find a descent direction. Using this type of information, GSA tries to reduce the consumption of resources (in terms of function evaluations) that other local search operators have.

One of the features of GSA is that, for this approach, it is not so important how many neighbors exist around the candidate solution. In fact, in some cases even when the number of neighboring solutions is low, it is possible to find the most greedy direction only for the subspace constructed by these solutions. But if more neighboring solutions are incorporated, the size of the subspace increases, and, as a consequence, a more greedy direction can be obtain.

Another advantage of GSA is that it includes a formulation to handle equality and inequality constraints. Such realization avoids the need of an extra constraint handling technique. For the case of equality constrained problems, GSA constructs a direction using a linear approximation of the constraint such that the search direction moves along such approximation. For inequality constraints, GSA proposes to correct the direction such that it moves along the constraints.

The first experiments with GSA compared the algorithm with other local search strategies: pattern search and the Nelder-Mead algorithm. In these experiments, GSA yielded faster convergence rates in comparison with the other algorithms. This can be explained because when constraints were found, the other algorithms kept searching in all directions. But at such condition, GSA started the predictor-corrector step to move along the active constraints.

Using the MOS mechanism along with GSA, we constructed a memetic algorithm using Differential Evolution as our base algorithm. The comparison was performed using three different algorithms: DE, DE/GSA and DE/LS1. The results showed an improvement in terms of convergence rate. The GSA/DE algorithm presented better results than the other two algorithms in 13 out of 23 instances of the problems.

Beside, GSA can save a considerable number of function evaluations. The experiments showed that in some of the problems, the percentage of function calls needed to reach the same quality level of solutions decreases up to 90 percent. On the other hand, the cases where the two other algorithms (DE and DE/LS1) spend less resources that the GSA were not so common.

We also proposed an extension of GSA to handle MOPs. The multi-objective GSA included two different techniques to compute a descent direction. The first approach used a scalarization function. Meanwhile, the second mechanism used Lara's

direction along with the approximations of gradient information given by GSA.

An experiment with the GSA was performed using two different multi-objective memetic algorithms: MOEA/D/GSA and IG-NSGA-II. On such an experiment we presented several comparison between the base algorithms and their memetic versions. The results showed that GSA improved the indicator results on the given test functions. In particular, we improved a 9 out of 11 of the unconstrained test problems and 16 out of 20 of the constraint test problems.

The future work proposed for the GSA algorithm is based on observations that were obtained during the experiments. The first possible improvement consists in the modification of the correction step. For some problems, solutions with large improvements were discarded because the constraints become infeasible by a certain threshold. If a good correction step is incorporated into the method it is possible that such solutions can be corrected. If the solution is returned to the feasible region, it is possible that such solution survives the selection process. By surviving, the recombination of these solutions can accelerate the convergence rate.

Another point of improvement consists in the mechanisms used for computing the initial step size. For some candidate solutions, if a large step size is taken, an overcost in the correction step can be triggered. The initial step size of this work was computed using the distance between the neighboring solution and the candidate solution, but in the state-of-the-art there exist several techniques that can improve such an idea.

Finally, we need the multi-objective version of the GSA for MOPs. For example, we suggest to test with some other scalarization functions. Moreover, we can incorporate a more sophisticate mechanism for the treatment of the MOPs. Besides, another future work is the improvement of the constraint handling techniques. The first improvement to be expected in such context is a mechanism to incorporate also equality constraints into multi-objective GSA. Besides, a different constraint-handling technique can be used, e.g., interior point methods to obtain a better convergence rate.

Subspace Polynomial Mutation

SPM is a mutation operator proposed for MOEAs to handle inequality constrained MOPs. A memetic algorithm was constructed using the NSGA-II method. An experiment was proposed to demonstrate the performance of the novel operator. The results showed that SPM improved the indicator results in comparison with the standalone NSGA-II. In particular, the gap between the two algorithms is clear when a problem contains only linear inequality constraints. For such kind of functions the results obtained a significant improvement in 4 out of 5 functions.

The second part of the experiments was performed using non-linear inequality constraints. For such kind of functions the results were promising. If some other specialized mechanism are introduced it is possible that it improves such results. The great advantage obtained in the experiments with the SPM operator is that it can steer the search along the border of the feasible set defined by the constraints. Besides, the computations of the SPM operator do not require explicitly gradient information.

Another advantages is that any computation performed by the PSM operator comes with no additional cost in terms of function calls. That is, all mechanism use information of the neighborhood of the candidate solution. Thus, our method can easily replace any other mutation operator of an existing MOEA.

As mentioned above, the SPM method requires several changes to improve its mechanisms. One of the most important changes that can be applied is the incorporation of a step size control mechanism. In the algorithm described in this work, the step size was proposed as a fixed value. But such value must be reduced as the algorithm starts to converge to the PF or increased if the solution requires it.

The second possible modification for the SPM is to improve the mechanism that controls the application of the operator. The actual mechanism applies SPM when it detects that there exists at least one active constraint. To improve the mechanism one can apply a mechanism that measures the performance of the SPM operator and such mechanism decides if the SPM has to be applied or not.

The SPM operator only incorporates inequality constraints. Until now, we can solve an equality constrained problem by transforming the constraints into inequal-

ity ones. But as future work it is necessary to incorporate the equality constraints directly into the constraint mechanism of the SPM such that this transformation is no longer required.

Appendix A: Single objective problems definition

Constrained problems

CEC 2006 contest for real parameter constrained optimization

In this Appendix we present the function definition for the 24 optimization problems for the 2006 contest on real constrained optimization [Liang et al., 2006].

g01

Minimize:

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i, \tag{6.1}$$

subject to:

$$\begin{aligned}
g_1(x) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
g_2(x) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
g_3(x) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
g_4(x) &= -8x_1 + x_{10} \leq 0 \\
g_5(x) &= -8x_2 + x_{11} \leq 0 \\
g_6(x) &= -8x_3 + x_{12} \leq 0 \\
g_7(x) &= -2x_4 - x_5 + x_{10} \leq 0 \\
g_8(x) &= -2x_6 - x_7 + x_{11} \leq 0 \\
g_9(x) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned} \quad , \tag{6.2}$$

where the bound are $0 \leq x_i \leq 1$, $i = 1, \ldots, 9$, $0 \leq x_i \leq 100$, $i = 10, 11, 12$ and $0 \leq x_{13} \leq 1$. The global minimum is $f(x^*) = -15$.

g02

Minimize:

$$f(x) = -\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i\, x_i^2}} \right| \tag{6.3}$$

subject to:

$$\begin{aligned} g_1(x) &= 0.75 - \prod_{i=1}^{n} x_i \leq 0 \\ g_2(x) &= \sum_{i=1}^{n} x_i - 7.5n \leq 0 \end{aligned}, \tag{6.4}$$

where $n = 20$ and $0 \leq x_i \leq 10$, $i = 1, \ldots, n$. The best value found so far is $f(x) = -0.80361910412559$.

g03

Minimize:

$$f(x) = -(\sqrt{(}n)^n) \prod_{i=1}^{n} x_i \tag{6.5}$$

subject to:

$$h_1(x) = \sum_{i=1}^{n} x_i^2 - 1 = 0, \tag{6.6}$$

where $n = 10$ and $0 \leq x_i \leq 1$, $i = 1, \ldots, n$. The global minimum is is $f(x^*) = -1.00050010001000$.

g04

Minimize:

$$f(x) = 5.3578547x_3^2 + 0.8356891\,x_1\,x_5 + 37.293239\,x_1 - 40792.141, \tag{6.7}$$

subject to:

$$\begin{aligned}
g_1(x) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0\\
g_2(x) &= -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5\\
g_3(x) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110\\
g_4(x) &= -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_23 + 90\\
g_5(x) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25\\
g_6(x) &= -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0
\end{aligned} \quad , \tag{6.8}$$

where $78 \leq x_1 \leq 102$, $33 \leq x_1 \leq 45$, $27 \leq x_i \leq 45$, $i = 3, 4, 5$. The global minimum is $f(x^*) = -3.066553867178332e^4$.

g05

Minimize:

$$f(x) = 3x_1 + 0.000001x_1^3 + 2x_2 + \frac{0.000002}{3}x_2^3, \tag{6.9}$$

subject to:

$$\begin{aligned}
g_1(x) &= -x_4 + x_3 - 0.55 \leq 0\\
g_2(x) &= -x_3 + x_4 - 0.55 \leq 0\\
h_1(x) &= 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0\\
h_2(x) &= 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0\\
h_3(x) &= 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0
\end{aligned} \quad , \tag{6.10}$$

where $0 \leq x_1 \leq 1200$, $0 \leq x_21200$ and $-0.55 \leq x_3, x_4 \leq 0.55$. The best value found so far is $f(x) = 5126.4967140071$.

g06

Minimize:

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3 \tag{6.11}$$

subject to:

$$\begin{aligned} g_1(x) &= -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\ g_2(x) &= -(x_1 - 6)^2 - (x_2 - 5)^2 - 82.81 \leq 0 \end{aligned}, \tag{6.12}$$

where $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. The global minimum is $f(x^*) = -6961.81387558015$.

g07

Minimize:

$$\begin{aligned} f(x) = x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\ + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7^2) + 45 \end{aligned}, \tag{6.13}$$

subject to:

$$\begin{aligned} g_1(x) &= -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0 \\ g_2(x) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \\ g_3(x) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\ g_4(x) &= 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \\ g_5(x) &= 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0 \\ g_6(x) &= x_1^2 + 2(x_2 - 2)^2 - 2x_1 x_2 + 14x_5 - 6x_6 \leq 0 \\ g_7(x) &= 0.5(x_1 - 8)^2 + 2(x_2 - 4)2 + 3x_5^2 - x_6 - 30 \leq 0 \\ g_8(x) &= -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0 \end{aligned}, \tag{6.14}$$

where the bound are $0 \leq x_i \leq 1$, $i = 1, \ldots, 9$, $0 \leq x_i \leq 100$, $i = 10, 11, 12$ and $0 \leq x_{13} \leq 1$. The global minimum is $f(x^*) = 24.30620906818$.

g08

Minimize:

$$f(x) = -\frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)}, \tag{6.15}$$

subject to:

$$\begin{aligned} g_1(x) &= x_1^2 - x_2 + 1 \leq 0 \\ g_2(x) &= 1 - x_1 + (x_2-3)^2 \leq 0 \end{aligned}, \tag{6.16}$$

where $0 \leq x_1, x_2 \leq 10$. The best value found so far is $f(x) = -0.0958250414180359..$

g09

Minimize:

$$f(x) = (x_1-10)^2+5(x_2-12)^2+4x_3+3(x_4-11)+10x_5^6+7x_6^2+x_7^4-4*x_6x_7-10x_6-8x_7 \tag{6.17}$$

subject to:

$$\begin{aligned} g_1(x) &= -127 + 2x_1^2 + 3x_2^2 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\ g_2(x) &= -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\ g_3(x) &= -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \\ g_4(x) &= 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \end{aligned}, \tag{6.18}$$

where $-10 \leq x_i \leq 10$, $i = 1, \ldots, 7$. The global minimum is $f(x^*) = 680.630057374402$.

g10

Minimize:

$$f(x) = x_1 + x_2 + x_3, \tag{6.19}$$

subject to:

$$\begin{aligned} g_1(x) &= -1 + 0.0025(x_4 + x_6) \leq 0 \\ g_2(x) &= -1 + 0.0025(x_5 + x_7 - x_4) \leq 0 \\ g_3(x) &= -1 + 0.01(x_8 - x_5) \leq 0 \\ g_4(x) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0 \\ g_5(x) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0 \\ g_6(x) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0 \end{aligned}, \tag{6.20}$$

where the bound are $100 \leq x_1 \leq 10000$, $10 \leq x_i \leq 1000$, $i = 4, \ldots, 8$. The global minimum is $f(x^*) = 7049.24802052867$.

g11

Minimize:

$$f(x) = x_1^2 + (x_2 - 1)^2 \tag{6.21}$$

subject to:

$$h(x) = x_2 - x_1^2 = 0, \tag{6.22}$$

where $-1 \leq x_1, x_2 \leq 1$. The global minimum is $f(x) = 0.7499$.

g12

Minimize:

$$f(x) = -\frac{100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2}{100}, \tag{6.23}$$

subject to:

$$g(x) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0, \tag{6.24}$$

where $0 \leq x_i \leq 10$, $i = 1, \ldots, 10$, $p, q, r = 1, \ldots, 9$. The global minimum is $f(x^*) = -1$.

g13

Minimize:

$$f(x) = e^{x_1 x_2 x_3 x_4 x_5}, \tag{6.25}$$

subject to:

$$\begin{aligned} h_1(x) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ h_2(x) &= x_2 x_3 - 5 x_4 x_5 = 0 \\ h_3(x) &= x_1^3 + x_2^3 + 1 = 0 \end{aligned} \quad , \tag{6.26}$$

where $-2.3 \leq x_1, x_2 \leq 2.3$ and $-3.2 \leq x_i \leq 3.2$, $i = 3, 4, 5$. The global minimum is $f(x^*) = 0.053941514041898$.

g14

Minimize:

$$f(x) = \sum_{i=1}^{10} x_i \left(c_i \ln \frac{x_i}{\sum_{j=1}^{1} 0 x_j} \right), \tag{6.27}$$

subject to:

$$\begin{aligned} h_1(x) &= x_1 + 2x_2 + +2x_3 + x_6 + x_1 0 - 2 = 0 \\ h_2(x) &= x_4 + 2x_5 + x_6 + x_7 - 1 = 0 \\ h_3(x) &= x_3 + x_7 + x_8 + 2x_9 + x_1 0 - 1 = 0 \end{aligned} \quad , \tag{6.28}$$

where $0 \leq x_i \leq 10$, $i = 1, \ldots, 10$, and $c_1 = -6.089$, $c_2 = -17.164$, $c_3 = -34.054$, $c_4 = -5.914$, $c_5 = -24.721$, $c_6 = -14.986$, $c_7 = -24.1$, $c_8 = -10.708$, $c_9 = -26.662$, $c_{10} = -22.179$. The global minimum is $f(x^*) = -47.7648884594915$.

g15

Minimize:

$$f(x) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3, \tag{6.29}$$

subject to:

$$\begin{aligned} h_1(x) &= x_1^2 + x_2^2 + x_3^2 - 25 = 0 \\ h_2(x) &= 8x_1 + 14x_2 + 7x_3 - 56 = 0 \end{aligned}, \tag{6.30}$$

where $0 \leq x_i \leq 10$, $i = 1, \ldots, 3$. The best known solution is $f(x^*) = 961.715022289961$.

g16

Minimize:

$$\begin{aligned} f(x) = {} & 0.000117y_{14} + 0.1365 + 0.00002358y_{13} + 0.000001502y_{16} + 0.0321y_{12} \\ & + 0.004324y_5 + 0.0001\frac{c_15}{c_16} + 37.48\frac{y_2}{c_12} - 0.0000005843y_{17} \end{aligned}, \tag{6.31}$$

subject to:

$$
\begin{aligned}
g_1(x) &= \frac{0.28}{0.72}y_5 - y_4 \leq 0 \\
g_2(x) &= x_3 - 1.5x_2 \leq 0 \\
g_3(x) &= 3496\frac{y_{12}}{c_{12}} - 21 \leq 0 \\
g_4(x) &= 110.6 + y_1 - \frac{62212}{c_{17}} \leq 0 \\
g_5(x) &= 213.1 - y_1 \leq 0 \\
g_6(x) &= y_1 - 405.23 \leq 0 \\
g_7(x) &= 17.505 - y_2 \leq 0 \\
g_8(x) &= y_2 - 1053.6667 \leq 0 \\
g_9(x) &= 11.275 - y3 \leq 0 \\
g_{10}(x) &= y_3 - 35.03 \leq 0 \\
g_{11}(x) &= 214.228 - y_4 \leq 0 \\
g_{12}(x) &= y_4 - 665.585 \leq 0 \\
g_{13}(x) &= 7.458 - y_5 \leq 0 \\
g_{14}(x) &= y_5 - 584.463 \leq 0 \\
g_{15}(x) &= 0.961 - y_6 \leq 0 \\
g_{16}(x) &= y_6 - 265.916 \leq 0 \\
g_{17}(x) &= 1.612 - y_7 \leq 0 \\
g_{18}(x) &= y_7 - 7.046 \leq 0 \\
g_{19}(x) &= 0.146 - y_8 \leq 0 \\
g_{20}(x) &= y_8 - 0.222 \leq 0 \\
g_{21}(x) &= 107.99 - y_9 \leq 0 \\
g_{22}(x) &= y_9 - 273.366 \leq 0 \\
g_{23}(x) &= 922.693 - y_{10} \leq 0 \\
g_{24}(x) &= y_{10} - 1286.105 \leq 0 \\
g_{25}(x) &= 926.832 - y_{11} \leq 0 \\
g_{26}(x) &= y_{11} - 1444.046 \leq 0
\end{aligned}
\quad , \tag{6.32}
$$

$$
\begin{aligned}
g_{27}(x) &= 18.766 - y_{12} \leq 0 \\
g_{28}(x) &= y_{12} - 537.141 \leq 0 \\
g_{29}(x) &= 1072.163 - y_{13} \leq 0 \\
g_{30}(x) &= y_{13} - 3247.039 \leq 0 \\
g_{31}(x) &= 8961.448 - y_{14} \leq 0 \\
g_{32}(x) &= y_{14} - 26844.086 \leq 0 \\
g_{33}(x) &= 0.063 - y_{15} \leq 0 \\
g_{34}(x) &= y_{15} - 0.386 \leq 0 \\
g_{35}(x) &= 71084.33 - y_{16} \leq 0 \\
g_{36}(x) &= -140000 + y_{16} \leq 0 \\
g_{37}(x) &= 2802713 - y_{17} \leq 0 \\
g_{38}(x) &= y_{17} - 12146108 \leq 0
\end{aligned}
\quad ,
$$

where

$$
\begin{aligned}
y_1 &= x_2 + x_3 + 41.6 \\
c_1 &= 0.024x_4 - 4.62 \\
y_2 &= \frac{12.5}{c_1} + 12 \\
c_2 &= 0.0003535x_1^2 + 0.5311x_1 + 0.08705y_2x_1 \\
c_3 &= 0.052x_1 + 78 + 0.002377y_2x_1 \\
y_3 &= \frac{c_2}{c_3} \\
y_4 &= 19y_3 \\
c_4 &= 0.04782(x_1 - y_3) + \frac{0.1956(x_1 - y_3)^2}{x_2} + 0.6376y_4 + 1.594y_3 \\
c_5 &= 100x_2 \\
c_6 &= x_1 - y_3 - y_4 \\
c_7 &= 0.950 - \frac{c_4}{c_5} \\
y_5 &= c_6c_7 \\
y_6 &= x_1 - y_5 - y_4 - y_3
\end{aligned}
\quad , \qquad (6.33)
$$

$$
\begin{aligned}
c_8 &= (y_5 + y_4)0.995 \\
y_7 &= \frac{c_8}{y_1} \\
y_8 &= \frac{c_8}{3798} \\
c_9 &= y_7 - \frac{0.0663 y_7}{y_8} - 0.3153 \\
y_9 &= \frac{96.82}{c_9} + 0.321 y_1 \\
y_{10} &= 1.29 y_5 + 1.258 y_4 + 2.29 y_3 + 1.71 y_6 \\
y_{11} &= 1.71 x_1 - 0.452 y_4 + 0.580 y_3 \\
c_{10} &= \frac{12.3}{752.3} \\
c_{11} &= (1.75 y_2)(0.995 x_1) \\
c_{12} &= 0.995 y_{10} + 1998 \\
y_{12} &= c_{10} x_1 + \frac{c_{11}}{c_{12}} \\
y_{13} &= c_{12} - 1.75 y_2 \\
y_{14} &= 3623 + 64.4 x_2 + 58.4 x_3 + \frac{146312}{y_9 + x_5} \\
c_{13} &= 0.995 y_{10} + 60.8 x_2 + 48 x_4 - 0.1121 y_{14} - 5095 \\
y_{15} &= \frac{y_{13}}{c_{13}} \\
y_{16} &= 148000 - 331000 y_{15} + 40 y_{13} - 61 y_{15} y_{13} \\
c_{14} &= 2324 y_{10} - 28740000 y_2 \\
y_{17} &= 14130000 - 1328 y_{10} - 531 y_{11} + \frac{c_{14}}{c_{12}} \\
c_{15} &= \frac{y_{13}}{y_{15}} - \frac{y_{13}}{0.52} \\
c_{16} &= 1.104 - 0.72 y_{15} \\
c_{17} &= y_9 + x_5
\end{aligned}
\quad ,
$$

and the bounds are $704.4148 \leq x_1 \leq 906.3855$, $68.6 \leq x_2 \leq 288.88$, $0 \leq x_3 \leq 134.75$, $193 \leq x_4 \leq 287.0966$ and $25 \leq x_5 \leq 84.1988$. The best known value is $f(x^*) = -1.90515525853479$.

g17

Minimize:

$$f(x) = f_1(x_1) + f_2(x_2), \tag{6.34}$$

where

$$f_1(x_1) = \begin{cases} 30x_1 \; 0 \leq x_1 < 300 \\ 31x_1 \; 300 \leq x_1 < 400 \end{cases}$$
$$f_2(x_2) = \begin{cases} 28x_2 \; 0 \leq x_2 < 100 \\ 29x_2 \; 100 \leq x_2 < 200 \\ 30x_2 \; 200 \leq x_2 < 1000 \end{cases}, \tag{6.35}$$

subject to:

$$\begin{aligned} h_1(x) &= -x_1 + 300 - \frac{x_3 x_4}{131.078}\cos(1.48477 - x_6) + \frac{0.90798x_3^2}{131.078}\cos(1.47588) = 0 \\ h_2(x) &= -x_2 - \frac{x_3 x_4}{131.078}\cos(1.48477 + x_6) + \frac{0.90798x_4^2}{131.078}\cos(1.47588) = 0 \\ h_3(x) &= -x_5 - \frac{x_3 x_4}{131.078}\sin(1.48477 + x_6) + \frac{0.90798x_4^2}{131.078}\sin(1.47588) = 0 \\ h_4(x) &= 200 - \frac{x_3 x_4}{131.078}\sin(1.48477 + x_6) + \frac{0.90798x_3^2}{131.078}\sin(1.47588) = 0 \end{aligned}, \tag{6.36}$$

where the bounds are $0 \leq x_1 \leq 400$, $0 \leq x_2 \leq 1000$, $340 \leq x_3 \leq 420$, $340 \leq x_4 \leq 420$, $-1000 \leq x_5 \leq 1000$ and $0 \leq x_6 \leq 0.5236$. The best known solution is $f(x^*) = 8853.53967480648$.

g18

Minimize:

$$f(x) = -0.5(x_1 x_4 - x_2 x_3 + x_3 x_9 - x_5 x_9 + x_5 x_8 - x_6 x_7), \tag{6.37}$$

subject to:

$$\begin{aligned}
g_1(x) &= x_3^2 + x_4^2 - 1 \leq 0 \\
g_2(x) &= x_9^2 - 1 \leq 0 \\
g_3(x) &= x_5^2 + x_6^2 - 1 \leq 0 \\
g_4(x) &= x_1^2 + (x_2 - x_9)^2 - 1 \leq 0 \\
g_5(x) &= (x_1 - x_5)^2 + (x_2 - x_6)^2 - 1 \leq 0 \\
g_6(x) &= (x_1 - x_7)^2 + (x_2 - x_8)^2 - 1 \leq 0 \\
g_7(x) &= (x_3 - x_5)^2 + (x_4 - x_6)^2 - 1 \leq 0, \\
g_8(x) &= (x_3 - x_7)^2 + (x_4 - x_8)^2 - 1 \leq 0 \\
g_9(x) &= x_7^2 + (x_8 - x_9)^2 - 1 \leq 0 \\
g_{10}(x) &= x_2 x_3 - x_1 x_4 \leq 0 \\
g_{11}(x) &= -x_3 x_9 \leq 0 \\
g_{12}(x) &= x_5 x_9 \leq 0 \\
g_{13}(x) &= x_6 x_7 - x_5 x_8 \leq 0
\end{aligned} \tag{6.38}$$

where the bound are $100 - 10 \leq x_i \leq 10$, $i = 1, \ldots, 9$ and $0 \leq x_9 \leq 20$. The global minimum is $f(x^*) = -0.866025403784439$.

g19

Minimize:

$$f(x) = \sum_{j=1}^{5} \sum_{i=1}^{5} c_{ij} x_{(10+j)} + \sum_{j=1}^{5} d_j x_{(10+j)}^3 - \sum_{i=1}^{10} b_i x_i, \tag{6.39}$$

subject to:

$$g_j(x) = -2 \sum_{i=1}^{5} c_{ij} x_{(10+j)} - e_j + \sum_{j=1}^{10} a_{ij} x_i \leq 0, \; j = 1, \ldots, 5, \tag{6.40}$$

where $b = (-40, -2, -0.25, -4, -4, -1, -40, -60, 5, 1)^T$ and the remaining data is on 6.1. The best known solution is $f(x^*) = 32.6555929502463$.

Table 6.1: Parameters for g19

j	1	2	3	4	5
e_j	-15	-27	-36	-18	-12
c_{1j}	30	-20	-10	32	-10
c_{2j}	-20	39	-6	-31	32
c_{3j}	-10	-6	10	-6	-10
c_{4j}	32	-31	-6	39	-20
c_{5j}	-10	32	-10	-20	30
d_j	4	8	10	6	2
a_{1j}	-16	2	0	1	0
a_{2j}	0	-2	0	0.4	2
a_{3j}	-3.5	0	2	0	0
a_{4j}	0	-2	0	-4	-1
a_{5j}	0	-9	-2	1	-2.8
a_{6j}	2	0	-4	0	0
a_{7j}	-1	-1	-1	-1	-1
a_{8j}	-1	-2	-3	-2	-1
a_{9j}	1	2	3	4	5
a_{10j}	1	1	1	1	1

g20

Minimize:

$$f(x) = \sum_{i=1}^{24} a_i x_i, \tag{6.41}$$

subject to:

$$\begin{aligned}
g_i(x) &= \frac{(x_i + x_{(i+12)})}{\sum_{j=1}^{24} x_j + e_i} \leq 0,\ i = 1, 2, 3 \\
g_i(x) &= \frac{(x_{(i+3)} + x_{(i+15)})}{\sum_{j=1}^{24} x_j + e_i} \leq 0,\ i = 4, 5, 6 \\
h_i(x) &= \frac{x_{(i+12)}}{b_{(i+12)} \sum_{j=1}^{24} \frac{x_j}{b_j}} = 0,\ i = 1, \ldots, 12 \\
h_{13}(x) &= \sum_{i=1}^{24} x_i - 1 = 0 \\
h_{14}(x) &= \sum_{i=1}^{12} \frac{x_i}{d_i} + k \sum_{i=13}^{24} \frac{x_i}{b_i} - 1.671 = 0
\end{aligned} \tag{6.42}$$

where $k = (0.7320)(530)(\frac{14.7}{40})$ and the remaining data is on 6.2.

g21

Minimize:

$$f(x) = x_1, \tag{6.43}$$

subject to:

$$\begin{aligned}
g_1(x) &= -x_1 + 35x_2^{0.6} + 35x_3^{0.6} \leq 0 \\
h_1(x) &= -300x_3 + 7500x_5 - 7500x_6 - 25x_4x_5 + 25x_4x_6 + x_3x_4 = 0 \\
h_2(x) &= 100x_2 + 155.365x_4 + 2500x_7 - x_2x_4 - 25x_4x_7 - 15536.5 = 0 \\
h_3(x) &= -x_5 - \ln(x_4 + 900) = 0 \\
h_4(x) &= -x_6 + \ln(x_4 + 300) = 0 \\
h_5(x) &= -x_7 + \ln(-2x_4 + 700) = 0
\end{aligned} \tag{6.44}$$

where the bounds are $0 \leq x1 \leq 1000$, $0 \leq x_2, x_3 \leq 40$, $100 \leq x_4 \leq 300$, $6.3 \leq x_5 \leq 6.7$, $5.9 \leq x_6 \leq 6.4$ and $4.5 \leq x_7 \leq 6.25$. The best known solution is $f(x) = 193.724510070035$.

Table 6.2: Parameters for g20

i	a_i	b_i	c_i	d_1	e_1
1	0.0693	44.094	123.7	31.244	0.1
2	0.0577	58.12	31.7	36.12	0.3
3	0.05	58.12	45.7	34.784	0.4
4	0.2	137.4	14.7	92.7	0.3
5	0.26	120.9	84.7	82.7	0.6
6	0.55	170.9	27.7	91.6	0.3
7	0.06	62.501	49.7	56.708	
8	0.1	84.94	7.1	82.7	
9	0.12	133.425	2.1	80.8	
10	0.18	82.507	17.7	64.517	
11	0.1	46.07	0.85	49.4	
12	0.09	60.097	0.64	49.1	
13	0.0693	44.094			
14	0.0577	58.12			
15	0.05	58.12			
16	0.2	137.4			
17	0.26	120.9			
18	0.55	170.9			
19	0.06	62.501			
20	0.1	84.94			
21	0.12	133.425			
22	0.18	82.507			
23	0.1	46.07			
24	0.09	60.097			

g22

Minimize:

$$f(x) = x_1, \tag{6.45}$$

subject to:

$$\begin{aligned}
g_1(x) &= -x_1 + x_2^0.6 + x_3 + x_4 \leq 0 \\
h_1(x) &= x_5 - 100000x_8 + 1 \times 10^7 = 0 \\
h_2(x) &= x_6 + 100000x_8 - 100000x_9 = 0 \\
h_3(x) &= x_7 + 100000x_9 - 5 \times 10^7 = 0 \\
h_4(x) &= x_5 + 100000x_{10} - 3.3 \times 10^7 = 0 \\
h_5(x) &= x_6 + 100000x_{11} - 4.4 \times 10^7 = 0 \\
h_6(x) &= x_7 + 100000x_{12} - 6.6 \times 10^7 = 0 \\
h_7(x) &= x_5 - 120x_2x_{13} = 0 \\
h_8(x) &= x_6 - 80x_3x_{14} = 0 \\
h_9(x) &= x_7 - 40x_4x_{15} = 0 \\
h_{10}(x) &= x_8 - x_{11} + x_{16} = 0 \\
h_{11}(x) &= x_9 - x_12 + x_{17} = 0 \\
h_{12}(x) &= -x_{18} + \ln(x_10 - 100) = 0 \\
h_{13}(x) &= -x_{19} + \ln(-x_8 + 300) = 0 \\
h_{14}(x) &= -x_{20} + \ln(x_{16}) = 0 \\
h_{15}(x) &= -x_{21} + \ln(-x_9 + 400) = 0 \\
h_{16}(x) &= -x_{22} + \ln(x_{17}) = 0 \\
h_{17}(x) &= -x_8 - x_{10} + x_{13}x_{18} - x_{13}x_{19} + 400 = 0 \\
h_{18}(x) &= x_8 - x_9 - x_{11} + x_{14}x_{20} - x_{14}x_{21} + 400 = 0 \\
h_{19}(x) &= x_9 - x_{12} - 4.60517x_{15} + x_{15}x_{22} + 100 = 0
\end{aligned} \quad , \tag{6.46}$$

where the bounds are $0 \leq x_1 \leq 20000$, $0 \leq x_2, x_3, x_4 \leq 1\times10^6$, $0 \leq x_5, x_6, x_7 \leq 4\times10^7$, $100 \leq x_8 \leq 299.99$, $100 \leq x_9 \leq 399.99$, $100.01 \leq x_{10} \leq 300$, $100 \leq x_{11} \leq 400, 100 \leq x_{12} \leq 600$, $0 \leq x_{13}, x_{14}, x_{15} \leq 500$, $0.01 \leq x_{16} \leq 300$, $0.01 \leq x_{17} \leq 400$, $-4.7 \leq$

$x_{18}, x_{19}, x_{20}, x_{21}, x_{22} \leq 6.25$. The best known solution is $f(x) = 236.430975504001$.

g23

Minimize:

$$f(x) = -9x_5 - 15x_8 + 6x_1 + 16x_2 - 10(x_6 + x_7), \tag{6.47}$$

subject to:

$$\begin{aligned} g_1(x) &= x_9x_3 + 0.02x_6 - 0.025x_5 \leq 0 \\ g_2(x) &= x_9x_4 + 0.02x_7 - 0.015x_8 \leq 0 \\ h_1(x) &= x_1 + x_2 - x_3 - x_4 = 0 \\ h_2(x) &= 0.03x_1 + 0.01x_2 - x_9(x_3 + x_4) = 0 \\ h_3(x) &= x_3 + x_6 - x_5 = 0 \\ h_4(x) &= x_4 + x_7 - x_8 = 0 \end{aligned}, \tag{6.48}$$

where the bounds are $0 \leq x_1, x_2, x_6 \leq 300$, $0 \leq x_3, x_5, x_7 \leq 100$, $0 \leq x_4, x_8 \leq 200$ and $0.01 \leq x9 \leq 0.03$. The best known solution is $f(x) = -400.055099999999584$.

g24

Minimize:

$$f(x) = -x_1 - x_2, \tag{6.49}$$

subject to:

$$\begin{aligned} g_1(x) &= -2x_1^4 + 8x_1^3 - 8x_1^2 + x_2 - 2 \leq 0 \\ g_2(x) &= -4x_1^4 + 32x_1^3 - 88x_1^2 + 96x_1 + x_2 - 36 \leq 0 \end{aligned}, \tag{6.50}$$

where the bounds are $0 \leq x1 \leq 3$ and $0 \leq x2 \leq 4$. The global minimum is $f(x) = -5.50801327159536$.

Appendix B: Multi-objective unconstrained problems definition

Unconstrained problems

Zitzler-Deb-Thiele (ZDT) benchmark

The test problems were original proposed in [Zitzler et al., 2000b].

ZDT1

Minimize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= g(x)\left(1 - \sqrt{\frac{f_1(x)}{g(x)}}\right), \end{aligned} \tag{6.51}$$

where

$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i, \tag{6.52}$$

and $n = 30$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 30$.

ZDT2

Minimize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= g(x)\left(1 - \left(\frac{f_1(x)}{g(x)}\right)^2\right), \end{aligned} \tag{6.53}$$

where

$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i, \tag{6.54}$$

and $n = 30$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 30$.

ZDT3

Minimize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= 1 - \sqrt{\frac{f_1(x)}{g(x)}} - \left(\frac{f_1(x)}{g(x)}\right) sin(10\pi f_1(x)) \end{aligned}, \tag{6.55}$$

where

$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \tag{6.56}$$

and $n = 30$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 30$.

ZDT4

Minimize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= 1 - g(x)\left(\frac{f_1(x)}{g(x)}\right)^2, \end{aligned} \tag{6.57}$$

where

$$g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}(x_i^2 - 10cos(4\pi x_i)) \tag{6.58}$$

and $n = 10$. The boundaries for the variables are defined as $-5 \leq x_i \leq 5$, $i = 1, \ldots, 10$.

ZDT6

Minimize:

$$\begin{aligned} f_1(x) &= 1 - \left(e^{-4x_1}\right) sin(6\pi x_1) \\ f_2(x) &= 1 - g(x)\left(\frac{f_1(x)}{g(x)}\right)^2 \end{aligned}, \tag{6.59}$$

where

$$g(x) = 1 + 9\left(\frac{\sum_{i=2}^{n} x_i}{n-1}\right)^{0.25}, \tag{6.60}$$

and $n = 10$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

Deb-Thiele-Laummans-Zitzler (DTLZ) benchmark

These test problems were originally proposed in [Deb et al., 2002b].

DTLZ1

Minimize:

$$\begin{aligned} f_1(x) &= \frac{1}{2}x_1(1 + g(x)) \\ f_2(x) &= \frac{1}{2}(1 - x_1)(1 + g(x)) \end{aligned}, \tag{6.61}$$

where

$$g(x) = 100\left(5 + \sum_{i=2}^{n}\left((x_i - 0.5)^2 - cos(20\pi(x_i - 0.5))\right)\right), \tag{6.62}$$

$n = 7$ and $k = 2$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

DTLZ2

Minimize:

$$\begin{aligned} f_1(x) &= (1 + g(x))cos\left(\frac{x_1\pi}{2}\right) \\ f_2(x) &= (1 + g(x))sin\left(\frac{x_1\pi}{2}\right) \end{aligned}, \tag{6.63}$$

where

$$g(x) = \sum_{i=2}^{n}(x_i - 0.5), \tag{6.64}$$

$n = 11$ and $k = 2$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

DTLZ3

Minimize:

$$\begin{aligned} f_1(x) &= (1 + g(x))cos\left(\frac{x_1\pi}{2}\right) \\ f_2(x) &= (1 + g(x))sin\left(\frac{x_1\pi}{2}\right) \end{aligned}, \tag{6.65}$$

where

$$g(x) = 100\left(10 + \sum_{i=2}^{n}\left((x_i - 0.5)^2 - cos(20\pi(x_i - 0.5))\right)\right) \tag{6.66}$$

$n = 11$ and $k = 2$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

DTLZ4

Minimize:

$$\begin{aligned} f_1(x) &= (1+g(x))cos\left(\frac{x_1^{100}\pi}{2}\right) \\ f_2(x) &= (1+g(x))sin\left(\frac{x_1^{100}\pi}{2}\right) \end{aligned}, \tag{6.67}$$

where

$$g(x) = \sum_{i=2}^{n}(x_i - 0.5) \tag{6.68}$$

$n = 11$ and $k = 2$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

DTLZ5

Minimize:

$$\begin{aligned} f_1(x) &= (1+g(x))cos\left(\frac{x_1\pi}{2}\right) cos\left(\left(\frac{\pi}{4(1+g(x))}\right)(1+2x_2g(x))\right) \\ f_2(x) &= (1+g(x))cos\left(\frac{x_1\pi}{2}\right) sin\left(\left(\frac{\pi}{4(1+g(x))}\right)(1+2x_2g(x))\right), \\ f_3(x) &= (1+g(x))sin\left(\frac{x_1\pi}{2}\right) \end{aligned} \tag{6.69}$$

where

$$g(x) = \sum_{i=3}^{n}(x_i - 0.5)^2 \tag{6.70}$$

$n = 10$ and $k = 3$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

DTLZ6

Minimize:

$$\begin{aligned} f_1(x) &= (1+g(x))cos\left(\frac{x_1\pi}{2}\right)cos\left(\left(\frac{\pi}{4(1+g(x))}\right)(1+2x_2g(x))\right) \\ f_2(x) &= (1+g(x))cos\left(\frac{x_1\pi}{2}\right)sin\left(\left(\frac{\pi}{4(1+g(x))}\right)(1+2x_2g(x))\right), \\ f_3(x) &= (1+g(x))sin\left(\frac{x_1\pi}{2}\right) \end{aligned} \tag{6.71}$$

where

$$g(x) = \sum_{i=3}^{n}(x_i - 0.5)^{0.1} \tag{6.72}$$

$n = 12$ and $k = 3$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

DTLZ7

Minimize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= x_2 \\ f_3(x) &= (1+g(x))h(x) \end{aligned} \quad , \tag{6.73}$$

where

$$g(x) = 1 + \frac{9}{8}\sum_{i=3}^{n} x_i, \tag{6.74}$$

$$h(x) = k - \frac{\sum_{i=1}^{k-1}(x_i sin\,(1+3\pi x_i))}{1+g(x)} \tag{6.75}$$

$n = 10$ and $k = 3$. The boundaries for the variables are defined as $0 \leq x_i \leq 1$, $i = 1, \ldots, 10$.

Other multi-objective test problems

CONV

Minimize:

$$\begin{aligned} f_1(x) &= (x_1 - 1)^4 + (x_2 - 1)^2 \\ f_2(x) &= (x_1 - 1)^2 + (x_2 - 1)^{2'} \end{aligned} \tag{6.76}$$

where $n = 2$ and $k = 2$. The boundaries for the variables are defined as $-5 \leq x_i \leq 5$, $i = 1, 2$.

Kursawe [Kursawe, 1990]

Minimize:

$$\begin{aligned} f_1(x) &= \sum_{i=1}^{2} \left(-10e^{-0.2\sqrt{x_i^2 + x_{i+1}^2}} \right) \\ f_2(x) &= \sum_{i=1}^{3} \left(|x_i|^{0.8} + 5sin\left(x_i^3\right) \right) \end{aligned}, \tag{6.77}$$

where $n = 3$ and $k = 2$. The boundaries for the variables are defined as $-5 \leq x_i \leq 5$, $i = 1, 2$.

Appendix C: Multi-objective constrained problems definition

Constrained problems

CEC 2009 contest

These test problems were originally proposed in [Zhang et al., 2008].

UF1

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}(x_j - \sin(6\pi x_1 + \frac{j\pi}{n}))^2 \\ f_2(x) &= 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2}(x_j - \sin(6\pi x_1 + \frac{j\pi}{n}))^2 \end{aligned}, \tag{6.78}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$ and $n = 30$. The variable bounds are $0 \leq x_1 \leq 1$ and $-1 \leq x_i \leq 2$, $i = 1, \ldots, n$.

UF2

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \\ f_2(x) &= 1 - \sqrt{x_1} + \frac{2}{|J_2|} y_j^2 \end{aligned}, \tag{6.79}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 30$ and

$$y_j = \begin{cases} x_j - (0.3x_1^2 \cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1)\cos(6\pi x_1 + \frac{j\pi}{n}), & j \in J_1 \\ x_j - (0.3x_1^2 \cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1)\sin(6\pi x_1 + \frac{j\pi}{n}), & j \in J_2 \end{cases} \tag{6.80}$$

The variable bounds are $0 \leq x_1 \leq 1$ and $-1 \leq x_i \leq 1$, $i = 2, \ldots, n$.

UF3

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\left(4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}} + 2\right)\right) \\ f_2(x) &= 1 - \sqrt{x_1} + \frac{2}{|J_2|}\left(4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos\left(\frac{20y_j\pi}{\sqrt{j}} + 2\right)\right) \end{aligned}, \tag{6.81}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 30$ and

$$y_j = x_j - x_1^{0.5\left(1+\frac{3(j-2)}{n-2}\right)},\ j = 2, \ldots, n \tag{6.82}$$

The variable bounds are $0 \leq x_i \leq 1$, $i = 1, \ldots, n$.

UF4

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j) \\ f_2(x) &= 1 - x_1^2 + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) \end{aligned}, \tag{6.83}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 30$,

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right),\ j = 2, \ldots, n, \tag{6.84}$$

and

$$h(t) = \frac{|t|}{|+e^{2|t|}}. \tag{6.85}$$

The variable bounds are $0 \leq x_1 \leq 1$ and $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

UF5

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \left(\frac{1}{2N} + \epsilon\right) |\sin(2N\pi x_1)| + \frac{2}{|J_1|} \sum_{j \in J_1} h(y_j) \\ f_2(x) &= 1 - x_1 + \left(\frac{1}{2N} + \epsilon\right) |\sin(2N\pi x_1)| + \frac{2}{|J_2|} \sum_{j \in J_2} h(y_j) \end{aligned}, \tag{6.86}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 30$, $N = 10$, $\epsilon = 0.1$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right), \; j = 2, \ldots, n, \tag{6.87}$$

and

$$h(t) = 2t^2 - \cos(4\pi t) + 1. \tag{6.88}$$

The variable bounds are $0 \leq x_1 \leq 1$ and $-1 \leq x_i \leq 1$, $i = 2, \ldots, n$.

UF6

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \max(0, 2\left(\frac{1}{2N} + \epsilon\right) \sin(2N\pi x_1)) + \frac{2}{|J_1|} \left(4 \sum_{j \in J_1} y_j^2 - 2 \prod_{j \in J_1} \cos\left(\frac{20 y_j \pi}{\sqrt{j}} + 2\right)\right) \\ f_2(x) &= 1 - x_1 + \max(0, 2\left(\frac{1}{2N} + \epsilon\right) \sin(2N\pi x_1)) + \frac{2}{|J_2|} \left(4 \sum_{j \in J_2} y_j^2 - 2 \prod_{j \in J_2} \cos\left(\frac{20 y_j \pi}{\sqrt{j}} + 2\right)\right) \end{aligned}, \tag{6.89}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 30$, $N = 2$, $\epsilon = 0.1$ and

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right), \; j = 2, \ldots, n, \tag{6.90}$$

The variable bounds are $0 \leq x_1 \leq 1$ and $-1 \leq x_i \leq 1$, $i = 2, \ldots, n$.

UF7

Minimize:

$$\begin{aligned} f_1(x) &= \sqrt[5]{x_1} + \frac{2}{|J_1|} \sum_{j \in J_1} y_j^2 \\ f_2(x) &= 1 - \sqrt[5]{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} y_j^2 \end{aligned}, \tag{6.91}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 30$,

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right), \; j = 2, \ldots, n, \tag{6.92}$$

and

$$h(t) = \frac{|t|}{|+e^{2|t|}}. \tag{6.93}$$

The variable bounds are $0 \leq x_1 \leq 1$ and $-1 \leq x_i \leq 1$, $i = 2, \ldots, n$.

UF8

Minimize:

$$\begin{aligned} f_1(x) &= \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|} \sum_{j \in J_1} \left(x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2 \\ f_2(x) &= \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|} \sum_{j \in J_2} \left(x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2, \\ f_3(x) &= \sin(0.5x_1\pi) + \frac{2}{|J_3|} \sum_{j \in J_3} \left(x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2 \end{aligned} \tag{6.94}$$

where $J_1 = \{j : 3 \leq j \leq n, \text{ and } j-1 \text{ is a multiplication of } 3\}$, $J_2 = \{j : 3 \leq j \leq n, \text{ and } j-2 \text{ is a multiplication of } 3\}$, $J_3 = \{j : 3 \leq j \leq n, \text{ and } j \text{ is a multiplication of } 3\}$ and $n = 30$. The variable bounds are $0 \leq x_1 \leq 1$ and $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

UF9

Minimize:

$$\begin{aligned} f_1(x) &= 0.5\left(\max\left(0,(1+\epsilon)(|-4(2x_1-1)^2)\right)+2x_1\right)x_2+\frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j-2x_2\sin\left(2\pi x_1+\frac{j\pi}{n}\right)\right)^2 \\ f_2(x) &= 0.5\left(\max\left(0,(1+\epsilon)(|-4(2x_1-1)^2)\right)+2x_1\right)x_2+\frac{2}{|J_2|}\sum_{j\in J_2}\left(x_j-2x_2\sin\left(2\pi x_1+\frac{j\pi}{n}\right)\right)^2, \\ f_3(x) &= 1-x_2+\frac{2}{|J_3|}\sum_{j\in J_3}\left(x_j-2x_2\sin\left(2\pi x_1+\frac{j\pi}{n}\right)\right)^2 \end{aligned} \tag{6.95}$$

where $J_1 = \{j : 3 \leq j \leq n,\ \text{and } j-1 \text{ is a multiplication of } 3\}$, $J_2 = \{j : 3 \leq j \leq n,\ \text{and } j-2 \text{ is a multiplication of } 3\}$, $J_3 = \{j : 3 \leq j \leq n,\ \text{and } j \text{ is a multiplication of } 3\}$ and $n = 30$. The variable bounds are $0 \leq x_1 \leq 1$ and $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

UF10

Minimize:

$$\begin{aligned} f_1(x) &= \cos(0.5\pi x_1)\cos(0.5\pi x_2)+\frac{2}{|J_1|}\sum_{j\in J_1}\left(4y_j-cos(8\pi y_j)+1\right) \\ f_2(x) &= \cos(0.5\pi x_1)\sin(0.5\pi x_2)+\frac{2}{|J_2|}\sum_{j\in J_2}\left(4y_j-cos(8\pi y_j)+1\right), \\ f_3(x) &= \sin(0.5\pi x_1)+\frac{2}{|J_3|}\sum_{j\in J_3}\left(4y_j-cos(8\pi y_j)+1\right) \end{aligned} \tag{6.96}$$

where $J_1 = \{j : 3 \leq j \leq n,\ \text{and } j-1 \text{ is a multiplication of } 3\}$, $J_2 = \{j : 3 \leq j \leq n,\ \text{and } j-2 \text{ is a multiplication of } 3\}$, $J_3 = \{j : 3 \leq j \leq n,\ \text{and } j \text{ is a multiplication of } 3\}$, $n = 30$ and

$$y_j = x_j - 2x_2\sin\left(2\pi x_1+\frac{j\pi}{n}\right),\ 3,\ldots,n. \tag{6.97}$$

The variable bounds are $0 \leq x_1 \leq 1$ and $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

Constrained problems

CEC 2009 constest

These test problems were originally proposed in [Zhang et al., 2008].

CF1

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - x_1^{0.5\left(1+\frac{3(j-2)}{n-2}\right)}\right)^2 \\ f_2(x) &= 1 - x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - x_1^{0.5\left(1+\frac{3(j-2)}{n-2}\right)}\right)^2, \end{aligned} \tag{6.98}$$

subject to:

$$g(x) = f_1(x) + f_2(x) - a\left|\sin\left(N\pi(f_1(x) - f_2(x) + 1) - 1\right)\right| \geq 0, \tag{6.99}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 30$, $N = 10$ and $a = 1$. The variable bounds are $0 \leq x_i \leq 1$, $i = 1, \ldots, n$.

CF2

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right)\right)^2 \\ f_2(x) &= 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2}\left(x_j - \cos\left(6\pi x_1 + \frac{j\pi}{n}\right)\right)^2, \end{aligned} \tag{6.100}$$

subject to:

$$g(x) = \frac{t(x)}{1 + e^{4|t|}} \geq 0, \tag{6.101}$$

where

$$t(x) = f_2(x) + \sqrt{f_1(x)} - a\sin\left(N\pi(\sqrt{f_1(x)} - f_2(x) + 1) - 1,\right), \tag{6.102}$$

$J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 10$, $N = 2$ and $a = 1$. The variable bounds are $0 \leq x_1 \leq 1$, $-1 \leq x_i \leq 1$, $i = 2, \ldots, n$.

CF3

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\left(4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right) \\ f_2(x) &= 1 - x_1^2 + \frac{2}{|J_2|}\left(4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right) \end{aligned}, \tag{6.103}$$

where

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right),\ j = 2, \ldots, n, \tag{6.104}$$

subject to:

$$g(x) = f_2(x) + f_1(x)^2 - -a\sin\left(N\pi(f_1(x)^2 - f_2(x) + 1) - 1\right) \geq 0, \tag{6.105}$$

where $J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$, $n = 10$, $N = 2$ and $a = 1$. The variable bounds are $0 \leq x_1 \leq 1$, $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

CF4

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \sum_{j\in J_1} h_j(y_j) \\ f_2(x) &= 1 - x_1 + \sum_{j\in J_2} h_j(y_j) \end{aligned}, \tag{6.106}$$

where

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right),\ j = 2, \ldots, n, \tag{6.107}$$

$$h_2(t) = \left\{ |t| \text{ if } t < \tfrac{3}{2}\left(1 - \tfrac{\sqrt{2}}{2}\right) 0.125 + (t-1)^2, \text{ otherwise} \right. \tag{6.108}$$

and

$$h_j(t) = t^2,\ j = 3, \ldots, n \tag{6.109}$$

Subject to:

$$g(x) = \frac{s(x)}{1 + e^{4|s(x)|}} \geq 0, \tag{6.110}$$

where

$$s(x) = x_2 - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right) - 0.5x_1 + 0.25, \tag{6.111}$$

$J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$ and $n = 10$. The variable bounds are $0 \leq x_1 \leq 1$, $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

CF5

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \sum_{j \in J_1} h_j(y_j) \\ f_2(x) &= 1 - x_1 + \sum_{j \in J_2} h_j(y_j) \end{aligned}, \tag{6.112}$$

where

$$y_j = \left\{ x_j - 0.8x_1 \cos\left(6\pi x_1 + \tfrac{j\pi}{n}\right),\ \text{if } j \in J_1 x_j - 0.8x_1 \sin\left(6\pi x_1 + \tfrac{j\pi}{n}\right),\ \text{if } j \in J_2 \right. \tag{6.113}$$

$$h_2(t) = \left\{ |t| \text{ if } t < \tfrac{3}{2}\left(1 - \tfrac{\sqrt{2}}{2}\right) 0.125 + (t-1)^2, \text{ otherwise} \right. \tag{6.114}$$

and

$$h_j(t) = 2t^2 - \cos(4\pi t) + 1,\ j = 3, \ldots, n \tag{6.115}$$

Subject to:

$$g(x) = x_2 - 0.8x_1 \sin\left(6\pi x_1 + \frac{2\pi}{n}\right) - 0.5x_1 + 0.25 \geq 0, \tag{6.116}$$

$J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$ and $n = 10$. The variable bounds are $0 \leq x_1 \leq 1$, $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

CF6

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \sum_{j \in J_1} y_j^2 \\ f_2(x) &= (1 - x_1)^2 + \sum_{j \in J_2} y_j^2 \end{aligned}, \tag{6.117}$$

where

$$y_j = \begin{cases} x_j - 0.8x_1 \cos\left(6\pi x_1 + \frac{j\pi}{n}\right), & \text{if } j \in J_1 \\ x_j - 0.8x_1 \sin\left(6\pi x_1 + \frac{j\pi}{n}\right), & \text{if } j \in J_2 \end{cases} \tag{6.118}$$

Subject to:

$$\begin{aligned} g_1(x) &= x_2 - 0.8x_1 \sin\left(6\pi x_1 + \frac{2\pi}{n}\right) - \text{sign}\left(0.5(1 - x_1) - (1 - x_1)^2\right) \sqrt{|0.5(1 - x_1) - (1 - x_1)^2|} \geq 0 \\ g_2(x) &= x_4 - 0.8x_1 \sin\left(6\pi x_1 + \frac{4\pi}{n}\right) - \text{sign}\left(0.25\sqrt{1 - x_1} - 0.5(1 - x_1)\right) \sqrt{|0.25\sqrt{1 - x_1} - 0.5(1 - x_1)|} \geq 0 \end{aligned}, \tag{6.119}$$

$J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$ and $n = 10$. The variable bounds are $0 \leq x_1 \leq 1$, $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

CF7

Minimize:

$$\begin{aligned} f_1(x) &= x_1 + \sum_{j \in J_1} h_j(y_j) \\ f_2(x) &= (1 - x_1)^2 + \sum_{j \in J_2} h_j(y_j) \end{aligned}, \tag{6.120}$$

where

$$y_j = \begin{cases} x_j - \cos\left(6\pi x_1 + \frac{j\pi}{n}\right), & \text{if } j \in J_1 \\ x_j - \sin\left(6\pi x_1 + \frac{j\pi}{n}\right), & \text{if } j \in J_2 \end{cases} \tag{6.121}$$

$$h_2(t) = h_4(t) = t^2, \tag{6.122}$$

and

$$h_j(t) = 2t^2 - \cos(4\pi t) + 1, i = 3, 5, 6 \ldots, n. \tag{6.123}$$

Subject to:

$$\begin{aligned} g_1(x) &= x_2 - x_1 \sin\left(6\pi x_1 + \frac{2\pi}{n}\right) - \text{sign}\left(0.5(1 - x_1) - (1 - x_1)^2\right) \sqrt{|0.5(1 - x_1) - (1 - x_1)^2|} \geq 0 \\ g_2(x) &= x_4 - x_1 \sin\left(6\pi x_1 + \frac{4\pi}{n}\right) - \text{sign}\left(0.25\sqrt{1 - x_1} - 0.5(1 - x_1)\right) \sqrt{|0.25\sqrt{1 - x_1} - 0.5(1 - x_1)|} \geq 0 \end{aligned}, \tag{6.124}$$

$J_1 = \{j : j \text{ is odd and } 2 \leq j \leq n\}$, $J_2 = \{j : j \text{ is even and } 2 \leq j \leq n\}$ and $n = 10$. The variable bounds are $0 \leq x_1 \leq 1$, $-2 \leq x_i \leq 2$, $i = 2, \ldots, n$.

CF8

Minimize:

$$f_1(x) = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2$$
$$f_2(x) = \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2, \quad (6.125)$$
$$f_3(x) = \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2$$

subject to:

$$g_1(x) = \frac{f_1(x)^2 + f_2(x)^2}{1 - f_3(x)^2} - a\sin\left(N\pi\left(\frac{f_1(x)^2 + f_2(x)^2}{1 - f_3(x)^2} + 1\right)\right) - 1 \geq 0, \quad (6.126)$$

where $J_1 = \{j : 3 \leq j \leq n, \text{ and } j-1 \text{ is a multiplication of } 3\}$, $J_2 = \{j : 3 \leq j \leq n, \text{ and } j-2 \text{ is a multiplication of } 3\}$, $J_3 = \{j : 3 \leq j \leq n, \text{ and } j \text{ is a multiplication of } 3\}$, $n = 30$, $N = 2$ and $a = 4$. The variable bounds are $0 \leq x_1, x_2 \leq 1$ and $-4 \leq x_i \leq 4$, $i = 3, \ldots, n$.

CF9

Minimize:

$$f_1(x) = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2$$
$$f_2(x) = \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2, \quad (6.127)$$
$$f_3(x) = \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right)\right)^2$$

subject to:

$$g_1(x) = \frac{f_1(x)^2 + f_2(x)^2}{1 - f_3(x)^2} - a\sin\left(N\pi\left(\frac{f_1(x)^2 + f_2(x)^2}{1 - f_3(x)^2} + 1\right)\right) - 1 \geq 0, \quad (6.128)$$

where $J_1 = \{j : 3 \leq j \leq n, \text{ and } j-1 \text{ is a multiplication of } 3\}$, $J_2 = \{j : 3 \leq j \leq n, \text{ and } j-2 \text{ is a multiplication of } 3\}$, $J_3 = \{j : 3 \leq j \leq n, \text{ and } j \text{ is a multiplication of } 3\}$, $n = 30$, $N = 2$ and $a = 3$. The variable bounds are $0 \leq x_1, x_2 \leq 1$ and $-4 \leq x_i \leq 4$, $i = 3, \ldots, n$.

CF10

Minimize:

$$\begin{aligned} f_1(x) &= \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left(4y_j^2 - \cos(8\pi y_j) + 1\right) \\ f_2(x) &= \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left(4y_j^2 - \cos(8\pi y_j) + 1\right), \\ f_3(x) &= \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left(4y_j^2 - \cos(8\pi y_j) + 1\right) \end{aligned} \tag{6.129}$$

where

$$y_j(x) = x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{n}\right). \tag{6.130}$$

Subject to:

$$g_1(x) = \frac{f_1(x)^2 + f_2(x)^2}{1 - f_3(x)^2} - a\sin\left(N\pi\left(\frac{f_1(x)^2 + f_2(x)^2}{1 - f_3(x)^2} + 1\right)\right) - 1 \geq 0, \tag{6.131}$$

where $J_1 = \{j : 3 \leq j \leq n, \text{ and } j-1 \text{ is a multiplication of } 3\}$, $J_2 = \{j : 3 \leq j \leq n, \text{ and } j-2 \text{ is a multiplication of } 3\}$, $J_3 = \{j : 3 \leq j \leq n, \text{ and } j \text{ is a multiplication of } 3\}$, $n = 30$, $N = 2$ and $a = 1$. The variable bounds are $0 \leq x_1, x_2 \leq 1$ and $-4 \leq x_i \leq 4$, $i = 3, \ldots, n$.

Other constrained multi-objective problems

These test problems were taken from [Coello et al., 2007].

Belegundu

Minimize:

$$\begin{aligned} f_1(x) &= -x_1 + x_2 - 1 \leq 0 \\ f_2(x) &= x_1 + x_2 - 7 \leq 0 \end{aligned}, \tag{6.132}$$

subject to:

$$\begin{aligned} -x_1 + x_2 - 1 &\leq 0 \\ x_1 + x_2 - 7 &\leq 0 \\ 0 \leq x_1 &\leq 5 \\ 0 \leq x_2 &\leq 3 \end{aligned}, \tag{6.133}$$

where $n = 2$.

Binh(2)

Minimize:

$$\begin{aligned} f_1(x) &= 4x_1^2 + 4x_2^2 \\ f_2(x) &= (x_1 - 5)^2 + (x_2 - 5)^2 \end{aligned}, \tag{6.134}$$

subject to:

$$\begin{aligned} (x_1 - 5)^2 + x_2^2 - 25 &\leq 0 \\ -(x_1 - 8)^2 - (x_2 + 3)^3 + 7.7 &\leq 0 \\ 0 \leq x_1 &\leq 5 \\ 0 \leq x_2 &\leq 3 \end{aligned}, \tag{6.135}$$

where $n = 2$.

Binh(4)

Minimize:

$$\begin{aligned} f_1(x) &= 1.5 - x_1(1 - x_2) \\ f_2(x) &= 2.25 - x_1(1 - x_2^2) \end{aligned}, \tag{6.136}$$

subject to:

$$\begin{aligned} -x_1^2 - (x_2 - 0.5)^2 + 9 &\leq 0 \\ (x_1 - 1)^2 + (x_2 - 0.5)^2 - 6.25 &\leq 0 \\ -10 \leq x_1 &\leq 10 \\ -10 \leq x_2 &\leq 10 \end{aligned}, \tag{6.137}$$

where $n = 2$.

Obayashi

Maximize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= x_2 \end{aligned}, \tag{6.138}$$

subject to:

$$\begin{aligned} x_1^2 + x_2^2 &\leq 1 \\ 0 \leq x_1 &\leq 1, \\ 0 \leq x_2 &\leq 1 \end{aligned} \tag{6.139}$$

where $n = 2$.

Osyczka

Maximize:

$$\begin{matrix} f_1(x) = x_1 + x_2^2 \\ f_2(x) = x_1^2 + x_2 \end{matrix}, \tag{6.140}$$

subject to:

$$\begin{matrix} 12 - x_1 - x_2 \geq 0 \\ x_1^2 + 10x_1 - x_2^2 + 16x_2 - 80 \geq 0 \\ 2 \leq x_1 \leq 7 \\ 5 \leq x_2 \leq 10 \end{matrix}, \tag{6.141}$$

where $n = 6$.

Osyczka(2)

Minimize:

$$\begin{matrix} f_1(x) = x_1 + x_2^2 \\ f_2(x) = x_1^2 + x_2 \end{matrix}, \tag{6.142}$$

subject to:

$$\begin{matrix} x_1 + x_2 - 2 \geq 0 \\ 6 - x_1 - x_2 \geq 0 \\ 2 - x_2 + x_1 \geq 0 \\ 2 - x_1 + 3x_2 \geq 0 \\ 4 - (x_3 - 3)^2 - x_4 \geq 0 \\ (x_5 - 3)^3 + x_6 - 4 \geq 0 \\ 0 \leq x_1, x_2, x_6 \leq 10 \\ 1 \leq x_3, x_5 \leq 5 \\ 0 \leq x_4 \leq 6 \end{matrix}, \tag{6.143}$$

where $n = 6$.

Srinivas

Minimize:

$$\begin{aligned} f_1(x) &= (x_1 - 2)^2 + (x_2 - 1)^2 + 2 \\ f_2(x) &= 9x_1 - (x_2 - 1)^2 \end{aligned}, \tag{6.144}$$

subject to:

$$\begin{aligned} x_1^2 + x_2^2 - 225 &\leq 0 \\ x_1 - 3x_2 + 10 &\leq 0 \\ -20 \leq x_1, x_2 &\leq 20 \end{aligned}, \tag{6.145}$$

where $n = 2$.

Tamaki

Maximize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= x_2, \\ f_3(x) &= x_3 \end{aligned} \tag{6.146}$$

subject to:

$$\begin{aligned} x_1^2 + x_2^2 + x_3 &\leq 1 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}', \tag{6.147}$$

where $n = 2$.

Tanaka

Minimize:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= x_2 \end{aligned}, \tag{6.148}$$

subject to:

$$\begin{aligned} -x_1^2 - x_2^2 + 1 + 0.1cos\left(16\arctan(\frac{x_1}{x_2})\right) &\leq 0 \\ (x_1 - 0.5)^2 + (x_2 - 0.5)^2 &\leq 0.5' \\ 0 < x_1, x_2 &\leq \pi \end{aligned} \tag{6.149}$$

where $n = 2$.

Viennet(4)

Minimize:

$$\begin{aligned} f_1(x) &= \frac{(x_1 - 2)^2}{2} + \frac{(x_2 + 1)^2}{13} + 3 \\ f_2(x) &= \frac{(x_1 + x_2 - 3)^2}{175} + \frac{(2x_2 - x_1)^2}{17} - 13 \\ f_3(x) &= \frac{(3x_1 - 2x_2 + 4)^2}{8} + \frac{(x_1 - x_2 + 1)^2}{27} + 15 \end{aligned}, \tag{6.150}$$

subject to:

$$\begin{aligned} -4x_1 - x_2 + 4 &\geq 0 \\ x_1 + 1 &\geq 0 \\ x_2 - x_1 + 2 &\geq 0 \\ -4 \leq x_1, x_2 &\leq 4 \end{aligned}, \tag{6.151}$$

where $n = 2$.

Bibliography

[Ackley, 1987] D.H. Ackley, An empirical study of bit vector function optimization, *Genetic algorithms and simulated annealing*, vol. 1, pp. 170–204, Pitman, London, 1987.

[Alvarado et al., 2016] S. Alvarado, A. Lara, V. Sosa and O. Schütze, An effective mutation operator to deal with multi-objective constrained problems: SPM, in *2016 13th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pp. 1–6, Sept 2016.

[Bao et al., 2013] Y. Bao, Z. Hu and T. Xiong, A PSO and Pattern Search based Memetic Algorithm for SVMs Parameters Optimization, *Neurocomputing*, Elsevier, 2013.

[Beume et al., 2007] N. Beume, B. Naujoks and M. Emmerich, Sms-emoa: Multiobjective selection based on dominated hypervolume, *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, Elsevier, 2007.

[Brown and Smith, 2005a] M. Brown and R.E. Smith, Directed multi-objective optimisation, *International Journal of Computers, Systems and Signals*, vol. 6, no. 1, pp. 3–17, 2005a.

[Brown and Smith, 2005b] M. Brown and R.E. Smith, Directed multi-objective optimization, *International Journal of Computers, Systems, and Signals*, vol. 6, no. 1, pp. 3–17, IEE, 2005b.

[Coello et al., 2007] C.A. Coello, G.B. Lamont and D.A.V. Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd Edition, Springer, USA, 2007.

[Coello and Cortés, 2005] C.A.C. Coello and N.C. Cortés, Solving multiobjective optimization problems using an artificial immune system, *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 163–190, Springer, 2005.

[Das and Dennis, 1997] I. Das and J.E. Dennis, A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems, *Structural optimization*, vol. 14, no. 1, pp. 63–69, 1997, ISSN 1615-1488.

[Das and Dennis, 1998] I. Das and J.E. Dennis, Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems, *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 631–657, August 1998.

[Dawkins, 1989] R. Dawkins, The Selfish Gene, Popular Science, Oxford University Press, 1989, ISBN 9780192860927.

[Deb, 2001] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, 1st Edition, John Wiley and Sons, USA, 2001.

[Deb and Goyal, 1996] K. Deb and M. Goyal, A combined genetic adaptive search (geneas) for engineering design, *Computer Science and informatics*, vol. 26, pp. 30–45, Citeseer, 1996.

[Deb et al., 2002a] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective optimization genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002a.

[Deb et al., 2002b] K. Deb, L. Thiele, M. Laumanns and E. Zitzler, Scalable Multi-Objective Optimization Test Problems, in *Congress on Evolutionary Computation (CEC 2002)*, pp. 825–830, 2002b.

[Debye, 1909] P. Debye, Näherungsformeln für die zylinderfunktionen für große werte des arguments und unbeschränkt veränderliche werte des index, *Mathematische Annalen*, vol. 67, no. 4, pp. 535–558, Springer, 1909.

[Durillo et al., 2010] J. Durillo, A. Nebro, C.A.C. Coello, J. Garcia-Nieto, F. Luna and E. Alba, A study of multiobjective metaheuristics when solving parameter scalable problems, *IEEE Trans. Evol. Comput.*, vol. 14, no. 4, pp. 618–635, Aug 2010.

[Edgeworth, 1881] F.Y. Edgeworth, Mathematical Psychics; An Essay on the Application of Mathematics to the Moral Sciences, P. Keagan, England, 1881.

[Eiben et al., 2003] A.E. Eiben, J.E. Smith et al., Introduction to evolutionary computing, vol. 53, Springer, 2003.

[Fermat, 1629] P.D. Fermat, Fermat theorem for stationary points, 1629.

[Fletcher, 1975] R. Fletcher, An ideal penalty function for constrained optimization, *IMA Journal of Applied Mathematics*, vol. 15, no. 3, pp. 319–342, IMA, 1975.

[Fletcher and Reeves, 1964] R. Fletcher and C.M. Reeves, Function minimization by conjugate gradients, *The computer journal*, vol. 7, no. 2, pp. 149–154, Br Computer Soc, 1964.

[Gass and Saaty, 1955] S.I. Gass and T.L. Saaty, Parametric objective function (part 2)-generalization, *Journal of the Operations Research Society of America*, vol. 3, no. 4, pp. 395–401, INFORMS, 1955.

[Gómez and Coello, 2013] R.H. Gómez and C.A.C. Coello, MOMBI: A new metaheuristic for many-objective optimization based on the r2 indicator, in *2013 IEEE Congress on Evolutionary Computation*, pp. 2488–2495, IEEE, 2013.

[Haimes et al., 1971] Y.Y. Haimes, L. Ladson and D.A. Wismer, On a bicriterion formulation of the problems of integrated system identification and system optimization, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-1, no. 3, pp. 296–297, July 1971, ISSN 0018-9472, doi:10.1109/TSMC.1971.4308298.

[Hooke and Jeeves, 1961] R. Hooke and T.A. Jeeves, "Direct Search" Solution of Numerical and Statistical Problems, *J. ACM*, vol. 8, no. 2, pp. 212–229, ACM, New York, NY, USA, Apr. 1961, ISSN 0004-5411.

[Karush, 1939] W. Karush, Minima of Functions of Several Variables with Inequalities as Side Constraints, Master's thesis, Department of Mathematics, University of Chicago, 1939.

[Kuhn and Tucker, 1951] H.W. Kuhn and A.W. Tucker, Nonlinear programming, in *Proceedings of the 2nd Berkley Symposium on Mathematical Statics and Probability*, University of California Press, 1951.

[Kukkonen and Lampinen, 2006] S. Kukkonen and J. Lampinen, Constrained real-parameter optimization with generalized differential evolution, in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pp. 207–214, IEEE, 2006.

[Kursawe, 1990] F. Kursawe, A variant of evolution strategies for vector optimization, in *International Conference on Parallel Problem Solving from Nature*, pp. 193–197, Springer, 1990.

[Lara et al., 2013] A. Lara, S. Alvarado, S. Salomon, G. Avigad, C.A. Coello and O. Schütze, The gradient free directed search method as local search within multi-objective evolutionary algorithms, in *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation (EVOLVE II)*, pp. 153–168, 2013.

[LaTorre, 2009] A. LaTorre, A framework for hybrid dynamic evolutionary algorithms: multiple offspring sampling (MOS), Ph.D. thesis, Informatica, 2009.

[LaTorre et al., 2011] A. LaTorre, S. Muelas and J.M. Pea, A mos-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test, *Soft Computing*, vol. 15, no. 11, pp. 2187–2199, Springer-Verlag, 2011.

[Liang et al., 2006] J.J. Liang, T.P. Runarsson, E. Mezura-Montes, M. Clerc, P.N. Suganthan, C.A.C. Coello and K. Deb, Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization, , Nanyang Technological University, Singapore, March 2006.

[Liu et al., 2016a] J. Liu, K.L. Teo, X. Wang and C. Wu, An exact penalty function-based differential search algorithm for constrained global optimization, *Soft Computing*, vol. 20, no. 4, pp. 1305–1313, Springer, 2016a.

[Liu et al., 2016b] T. Liu, X. Gao and Q. Yuan, An improved gradient-based NSGA-II algorithm by a new chaotic map model, *Soft Computing*, pp. 1–15, Springer, 2016b.

[López et al., 2010] A.L. López, C.A.C. Coello and O. Schütze, A painless gradient-assisted multi-objective memetic mechanism for solving continuous bi-objective optimization problems, in *IEEE congress on evolutionary computation*, pp. 1–8, IEEE, 2010.

[Miettinen, 2012] K. Miettinen, Nonlinear multiobjective optimization, vol. 12, Springer Science & Business Media, 2012.

[Nelder and Mead, 1965] J.A. Nelder and R. Mead, A simplex method for function minimization, *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[Nocedal and Wright, 2006] J. Nocedal and S. Wright, Numerical Optimization, Springer Series in Operations Research and Financial Engineering, Springer, 2006.

[Olsen, 1994] A.L. Olsen, Penalty functions and the knapsack problem, in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pp. 554–558, IEEE, 1994.

[Pareto, 1896] V. Pareto, Cours DEconomie Politique, F. Rouge, Switzerland, 1896.

[Rosenbrock, 1960] H. Rosenbrock, An automatic method for finding the greatest or least value of a function, *The Computer Journal*, vol. 3, no. 3, pp. 175–184, Br Computer Soc, 1960.

[Rudolph et al., 2016] G. Rudolph, O. Schütze, C. Grimme, C. Domínguez-Medina and H. Trautmann, Optimal averaged hausdorff archives for bi-objective problems: theoretical and numerical results, *Computational Optimization and Applications*, vol. 64, no. 2, pp. 589–618, Springer, 2016.

[Schütze et al., 2012] O. Schütze, X. Esquivel, A. Lara and C.A. Coello, Using the Averaged Hausdorff Distance as a Performance Measure in Evolutionary Multiobjective Optimization, *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 504–522, Aug 2012, ISSN 1089-778X, doi:10.1109/TEVC.2011.2161872.

[Schütze et al., 2016] O. Schütze, V.A.S. Hernández, S.J.A. García and A. Lara, On the choice of neighborhood sampling in evolutionary multi-objective optimization, 2016.

[Schütze et al., 2016a] O. Schütze, V.A.S. Hernández, H. Trautmann and G. Rudolph, The hypervolume based directed search method for multi-objective optimization problems, *Journal of Heuristics*, pp. 1–28, Springer, 2016a.

[Schütze et al., 2010] O. Schütze, A. Lara and C.A. Coello, The directed search method for unconstrained multi-objective optimization problems, TR-OS-2010-01, CINVESTAV, January 2010.

[Schütze et al., 2011] O. Schütze, A. Lara and C.A. Coello, On the influence of the Number of objectives on the hardness of a multiobjective optimization problem, *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 4, pp. 444–455, August 2011.

[Schütze et al., 2016b] O. Schütze, A. Martín, A. Lara, S. Alvarado, E. Salinas and C.A.C. Coello, The directed search method for multi-objective memetic algorithms, *Computational Optimization and Applications*, vol. 63, no. 2, pp. 305–332, 2016b, ISSN 1573-2894, doi:10.1007/s10589-015-9774-0.

[Schwefel, 1993] H.P.P. Schwefel, Evolution and optimum seeking: the sixth generation, John Wiley & Sons, Inc., 1993.

[Shewchuk, 1994] J.R. Shewchuk, An introduction to the conjugate gradient method without the agonizing pain, 1994.

[Shukla, 2007] P.K. Shukla, Evolutionary Multi-Criterion Optimization: 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007. Proceedings, chap. On Gradient Based Local Search Methods in Unconstrained Evolutionary Multi-objective Optimization, pp. 96–110, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Storn and Price, 1995] R. Storn and K. Price, Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces, vol. 3, ICSI Berkeley, 1995.

[Storn and Price, 1997a] R. Storn and K. Price, Differential evolution a simple and efficient heuristic for global optimization over continuous spaces, *J. of Global Optimization*, vol. 11, no. 4, pp. 341–359, Kluwer Academic Publishers, Hingham, MA, USA, Dec. 1997a, ISSN 0925-5001.

[Storn and Price, 1997b] R. Storn and K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, Springer, 1997b.

[Trautmann et al., 2013] H. Trautmann, T. Wagner and D. Brockhoff, Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers, chap. R2-EMOA: Focused Multiobjective Search Using R2-Indicator-Based Selection, pp. 70–74, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, ISBN 978-3-642-44973-4.

[Tseng and Chen, 2008] L.Y. Tseng and C. Chen, Multiple trajectory search for large scale global optimization, in *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 3052–3059, June 2008.

[Van Veldhuizen, 1999] D.A. Van Veldhuizen, Multiobjective evolutionary algorithms: classifications, analyses, and new innovations, , DTIC Document, 1999.

[Wierzbicki, 1980] A.P. Wierzbicki, The use of reference objectives in multiobjective optimization, in *Multiple criteria decision making theory and application*, pp. 468–486, Springer, 1980.

[Wolfe, 1969] P. Wolfe, Convergence conditions for ascent methods, *SIAM review*, vol. 11, no. 2, pp. 226–235, SIAM, 1969.

[Zapotecas Martínez and Coello, 2012] S. Zapotecas Martínez and C.A.C. Coello, A Direct Local Search Mechanism for Decomposition-based Multi-Objective Evolutionary Algorithms, in *2012 IEEE Congress on Evolutionary Computation (CEC'2012)*, IEEE Press, pp. 3431–3438, Brisbane, Australia, June 2012.

[Zapotecas-Martnez and Coello, 2016] S. Zapotecas-Martnez and C.A.C. Coello, Monss: A multi-objective nonlinear simplex search approach, *Engineering Optimization*, vol. 48, no. 1, pp. 16–38, 2016, doi:10.1080/0305215X.2014.992889.

[Zhang et al., 2009a] Q. Zhang, W. Liu and H. Li, The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances, in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pp. 203–208, IEEE, 2009a.

[Zhang and Li, 2006] Q. Zhang and H. Li, A multi-objective evolutionary algorithm based on decomposition, , University of Essex, May 2006.

[Zhang et al., 2008] Q. Zhang, A. Zhou, S. Zhao, P.N. Suganthan, W. Liu and S. Tiwari, Multiobjective optimization test instances for the cec 2009 special session and competition, *University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report*, vol. 264, 2008.

[Zhang et al., 2009b] Q. Zhang, A. Zhou, S. Zhao, P.N. Suganthan, W. Liu and S. Tiwari, Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition, CES-487, University of Essex and Nanyang Technological University, April 2009b.

[Zitzler et al., 2000a] E. Zitzler, K. Deb and L. Thiele, Comparison of multiobjective evolutionary algorithms: Empirical results, *Evolutionary computation*, vol. 8, no. 2, pp. 173–195, MIT Press, 2000a.

[Zitzler et al., 2000b] E. Zitzler, K. Deb and L. Thiele, Comparison of Multiobjective Evolutionary Algorithms: Empirical Results, *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000b.

[Zitzler and Künzli, 2004] E. Zitzler and S. Künzli, Indicator-based selection in multiobjective search, in *International Conference on Parallel Problem Solving from Nature*, pp. 832–842, Springer, 2004.

[Zitzler and Thiele, 1999] E. Zitzler and L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, *IEEE transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, IEEE, 1999.