EVALUATION OF ACTIVE-SET EVOLUTION STRATEGIES FOR OPTIMIZATION WITH KNOWN CONSTRAINTS

by

Zehao Ba

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
July 2020

Table of Contents

List of Tables

List of Figures

Abstract

Evolution strategy (ES) is most often used to solve unconstrained black-box problems, while active-set methods focus on solving constrained optimization problems. A recent algorithm combines (1+1)-ES and an active-set method to get an active-set evolution strategy to solve problems in which the objective function is considered a black-box, but the constraint functions are known explicitly. We observe that the previous active-set evolution strategies have some settings result in less than optimal performance, so we make some adjustments to the past algorithms. More importantly, we systematically evaluate the performances of the two previous active-set evolution strategies with our modified version on the spherically symmetric functions with mutually orthogonal linear constraints. We also compare the performances of the modified version with three deterministic algorithms and an evolutionary algorithm. The test set we use is from the IEEE Congress on Evolutionary Computation (CEC) Competitions in 2006.

List of Abbreviations Used

GECCO	Geneticand Evolutionary Computation Conference
PPSN	Geneticand Evolutionary Computation Conference
KKT	Karush-Kuhn-Tucker
ES	Evolution Strategie
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
QP	Quadratic Programming
SQP	Sequential Quadratic Programming
EA	Evolutionary Algorithms
HM	Homomorphous Map
ARCH	Adaptive Ranking based Constraint Handling
DE	Differential Evolution
CEC	Congress on Evolutionary Computation

Acknowledgements

I would like to express my sincere gratitude to Prof. Dirk V. Arnold for his support during my master study. His patience, carefulness and wisdom constantly urged me to keep up with him, prompting me to delve into every detail of my study. In addition, I would also like to thank my parents for their continuous support and encouragement. Their selfless love and dedication have enabled me to focus and work harder in my research.

Chapter 1

Introduction

Optimization is an essential tool in several areas, such as mathematics, finance, engineering and machine learning. When you optimize, it means you are looking for the best solution. However, the definition varies for "best". For example, if you are a vaccine researcher, you might want to maximize the efficacy of a vaccine, but minimize the side effects. Both maximizing and minimizing are optimization problems. To perform the optimization, we first need to identify the objective function $f : \mathbb{R}^n \to \mathbb{R}$, which is a measurable quantity representing the performance of the system. A system's performance is affected by properties and characteristics which can be adjusted, called variables. The optimization process is searching for an optimal solution or a set of optimal solutions that maximize or minimize the objective. In some cases, variables are constrained to certain acceptable values, which we call constraints. For instance, setting a cost limitation for each vaccine is a constraint to medical research. Identifying the objective, variables, and constraints for a given problem is called modelling. The construction of the model can have a significant impact on the effectiveness of the optimization process [32, page 9].

To solve an optimization problem, algorithms start from an initial guess of variables' settings and iteratively generate improved estimates, based on the measure of the objective. Ideally, algorithms will terminate after finding a valid solution that meets some acceptability criterion. Good optimization algorithms tend to have the following characteristics:

- Robustness, which means the algorithm should be effective on a large number of problems.
- Efficiency, which means running the algorithm on a computer should not take too much time or storage.
- Accuracy, which means the algorithm can find a solution that is optimal.

However, sometimes these properties may conflict. A robust algorithm may be slow; or a fast algorithm has high storage requirements [32, page 9]. It is important to balance these elements in a useful algorithm.

In many cases, optimization algorithms employ the objective function values, constraint function values, and the function's first and second derivatives information to optimize problems. An example of an algorithm using first derivatives of the objective function is the steepest descent optimization. The first-order gradient provides the direction in which the objective function increases most rapidly, then the steepest descent algorithm can iteratively proceed opposite to the direction of the gradient to find the minima of the objective function. Newton's method is an example of a second-order method, which derives the direction from the second-order Taylor series approximation. This method requires the first and second derivatives of the objective function [32, page 44]. However, derivatives of the objective function are not always available, as is the case in black-box optimization where the first and second order information of the objective function cannot be examined. Under this circumstance, strategies study the history of previous iterations to get the successful steps and the directions, and then evolve according to these information. Evolutionary algorithms are one approach to conduct black-box optimization.

Evolutionary algorithms are commonly applied in problems where the objective function is difficult or impossible to represent mathematically, or the gradient estimation is expensive or inaccurate [2]. In evolutionary algorithms, optimization occurs through an iterative process. In each iteration, new candidate solutions (referred to as offspring) are generated by combining or mutating the best candidate solutions of the last iteration (referred to as parents), and then evaluate the objective function value of each. The candidate solutions with the best function values will be selected as the parents of the next iteration. These principles are inspired by biological evolution [17].

Optimizing the objective function subject to constraints on the variables is called constrained optimization. Constraints are denoted as equality and inequality. Le Digabel and Wild [12] categorize constraints as quantifiable and nonquantifiable, relaxable and unrelaxable, a-priori and simulation-based, and known and hidden. Quantifiable and nonquantifiable means the feasibility and/or violation degree of the

constraint can be quantified. Relaxable indicates the constraints may not need to be satisfied to compute meaningful objective function values. A-priori and simulation-based constraints mean whether or not the constraint needs to be simulated by a computer to confirm feasibility. Simulation-based constraints are potentially costly due to launching computer simulations. Known and hidden constraints point out whether the constraints are explicitly identified. For example, consider the problem $\min\{\sqrt{x} : x \in \mathbb{R}\}$. The constraint $x \geq 0$ is a hidden constraint unless it is expressed in the problem.

1.1 Motivation

Evolution strategies most often focus on solving unconstrained continuous black-box optimization, while active-set methods are designed to solve constrained optimization problems. Arnold implemented an active-set approach in an evolution strategy for dealing with constrained optimization [2, 3]. A goal of this thesis is to adjust the algorithm of Arnold so that it solves more problems in a set of test functions [26] that expands on those considered by Arnold. A further goal is to evaluate and compare the performances of our modification with three deterministic algorithms and one evolutionary algorithm.

The active-set evolution strategy was proposed at Parallel Problem Solving from Nature (PPSN) Conference. It combines the active-set method with (1+1)-ES to solve constrained optimization. (1+1)-ES means that a single offspring is generated by one parent in each iteration, while the active-set method maintains a set of constraints that contains the constraints satisfied by successful offspring. Offspring is usually projected onto the space determined by active-set to avoid unnecessary step-size reduction. In [2], algorithm checks whether the after-projection offspring is better than the best candidate solution so far at each iteration. If it does, active-set will be replaced by the one that generates the current candidate solution. Otherwise, the algorithm keeps the current active-set unchanged and enters the next iteration.

As more inequality constraints are added into the active-set, the search space is reduced. Continuously doing this may reduce the search space to zero, in which case the algorithm will repeatedly yield the same point. Under this circumstances, Arnold suggested suspending the use of the whole active-set [2]. But this method

does not perform well for some problems, especially for those with a large amount of inequality constraints in the optimal active-set. The algorithm may have long stagnation when inequality constraints are in the active-set but are not supposed to be. Therefore, Arnold revised this algorithm [2] and proposed a new one at the Genetic and Evolutionary Computation Conference (GECCO) [3]. Rather than suspending the whole active-set, he suggested releasing individual active inequality constraints in a random order. This revised algorithm has better performances than the previous version overall on the Michalewicz and Schoenauer test set.

The Michalewicz and Schoenauer test set [28] contains 11 test problems, $g01$ to $g11$; and Liang et al. extend them to 24 test problems [26]. These problems have variable dimensions, from 2 to 24, and have different kinds of objective functions and constraint functions. We tested the algorithm [3] on the additional problems from Liang et al., and we observed that infinite loops may occur on some problems. Those loops are internal to the *fmincon* function in the Matlab Optimization toolbox. Thus, we are investigating adjustments to options in the *fmincon* function and the active-set evolution strategy [3] so that it will work well from $g12$ to $g24$.

1.2 Contribution

In our experiments we discovered elements that affect the performance of Arnold's algorithm, such as the conditions for triggers to release constraints and the options that the projection function chooses. A well designed release condition can improve the efficiency of the algorithm, as does the projection function settings. Thus, we revise the rules of which constraint should be released, and how to set up the projection function. More importantly, we systematically compare the performances of our modified algorithm with algorithms from the PPSN conference and the GECCO conference on different dimensional spheres with different numbers of linear constraints. We also evaluate the performances of our modified algorithm with three deterministic algorithms and one evolutionary algorithm on the test set gathered by Liang et al. [26]. Deterministic algorithms include active-set methods and interior-point methods. The evolutionary algorithm is LSHADE44.

In the first experiment on a spherically symmetric function with mutually orthogonal linear constraints, we record the median number of objective function evaluations

required, and plot the histograms and traces of the algorithms. During the second experiment, we analyze the performance of a selection of algorithms applied to test functions from Liang et al. The median number of objective function evaluations required and success rates are recorded as the standard to evaluate approaches. We also plot the empirical cumulative running time distribution which illustrates the fraction of function instances which were optimized to a target value. The more targets the algorithm achieves, the better its performance.

Overall, we find that our modified version performs the best among three active-set evolution strategies on the sphere experiment, especially on the higher dimension problems. Furthermore, it has a better performance than active-set methods, the interior-point method and LSHADE44. The modified active-set evolution strategy either has a higher success rate or requires fewer function evaluations to optimize a function, or sometimes both. It also reaches the target on more function instances than the other tested algorithms across the 24 problems.

1.3 Outline

This thesis is organized as follows: Chapter 2 introduces necessary terminology, background information and related work. Chapter 3 describes the modification we made to the active-set evolution strategy [3] and analysis on simple test problems. In Chapter 4, we explain the set up for the complete experiments and systematically compare the performance of algorithms on test sets. The results are shown, displaying the table of success rates and median function evaluations for each test function, and the plots of empirical cumulative running time distributions. Finally, Chapter 5 discusses conclusions derived from this work and considers the outlook for future work in this field.

Chapter 2

Background and Related Work

In this chapter, we review previous work on constrained optimization, including active-set methods, interior-point methods, and evolutionary algorithms. Active-set methods work by transforming the optimization problem into an easier subproblem to be solved, while interior-point methods add a penalty function and convert the constrained problem to an unconstrained problem [32, page 529, 565]. Evolutionary algorithms are inspired by biological evolution and are useful for addressing problems which are difficult to solve mathematically, such as those where gradient information is unavailable or inaccurate. Although these methods work differently, they can all solve constrained optimization problems.

This chapter is organized as follows. In Section 2.1, some terminologies will be given for better understanding of the process of optimization. Then, we will introduce evolution strategies for unconstrained optimization problems in Section 2.2. Gradient-based methods and evolutionary algorithms for constrained optimization will be discussed in Section 2.3. Last, we will review the benchmarks established for Special Sessions held in connection with the IEEE Congress of Evolutionary Computation [26].

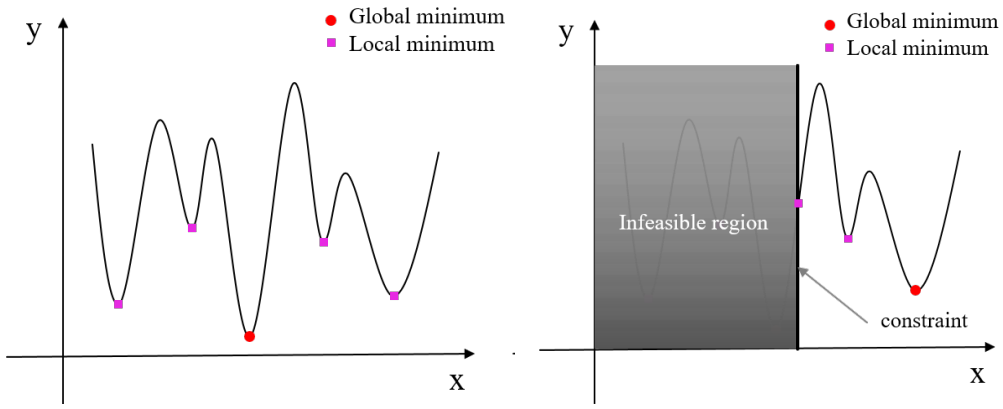
2.1 Terminology

In this section, we introduce some necessary information about optimization, including the concept of an optimization problem, and the tools that will be used in the following sections.

2.1.1 Global and Local Minima

An optimization problem usually describes the process of minimizing or maximizing an objective function. We will consider the minimization process in this thesis. In unconstrained optimization problems, algorithms look for the global minima of the

objective function f, which means a point $\mathbf{x}^*$ meets $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$, where n is the dimension of the search space. The global minima may be difficult to find because the objective function value is only known where it has been tested and may have a sharp dip. Algorithms are thus prone to finding local minima, being unaware of the global minima. A point $\mathbf{x}^*$ is a local minimum if there is an open neighbourhood $\mathcal{N}$ of $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{N}$ [32, page 13].



(a) Problem without constraint. (b) Problem with constraint.

Figure 2.1: An example of global minima and local minima of unconstrained and constrained optimization. The x-axis indicates the variables and the y-axis represents the objective function values. The red spots indicate the global minimum while the magenta spots indicate the local minima. The shaded area indicates the infeasible region.

In constrained optimization, the algorithm looks for the maxima or minima of the objective function subject to the equality and inequality constraints. The search space is reduced due to the constraints, and algorithms search for minimal or maximum function value within this reduced space. The global minima in constrained optimization must have the smallest function value for all points satisfying the constraints. Figure 2.1 displays a function with and without a constraint. In the Figure 2.1a, there are no constraints so the global minimum is at the position of the red spot. In the Figure 2.1b, a constraint appears and so the global minimum in the unconstrained objective function is not available. The algorithm has to search for the optimal solution in a reduced search space and the global minimum is relocated. Therefore, It can be seen that constrained optimization problems and unconstrained optimization problems may have different scenarios, which lead to different global

optima.

2.1.2 Constrained Optimization

As we have seen, optimization problems can be constrained, and there are several methods to solving these problems. Some methods transform constrained problems into a series of unconstrained problems; others may convert them to easier solved constrained problems. Generally speaking, a constrained minimization problem can be written as follows [32, page 304]:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \quad \text{subject to} \begin{cases} g_i(\mathbf{x}) \leq 0 & i \in [1, ..., l] \\ h_j(\mathbf{x}) = 0 & j \in [1, ..., m] \end{cases} \tag{2.1}$$

where $f(\mathbf{x})$ is the objective function, $h_j,\ j \in [1, ..., m]$ are the equality constraints and $g_i,\ i \in [1, ..., l]$ are the inequality constraints. The feasible set, denoted by Ω, is described as the set of points that satisfy all constraints, that is,

$$\Omega = \{\mathbf{x}\ \in \mathbb{R}^n \mid h_j(\mathbf{x}) = 0,\ j \in [1, ..., m]; g_i(\mathbf{x}) \leq 0, i \in [1, ..., l]\}. \tag{2.2}$$

It follows that (2.1) can be rewritten as

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}), \tag{2.3}$$

which means constrained optimization algorithms look for the optimal solution within Ω.

Le Digabel and Wild [12] categorize constraints as quantifiable and nonquantifiable, relaxable and unrelaxable, a-priori and simulation-based, and known and hidden.

- **Quantifiable versus nonquantifiable (Q/N):** A quantifiable constraint indicates that the degree of feasibility and/or violation of the constraint can be quantified. A nonquantifiable constraint means that the satisfaction or violation degree of a constraint is inaccessible. For example, if the running of a piece of code is less than 10 seconds, then this constraint is called quantifiable feasibility. The reason is we can get the time it takes for the code to run, and then know

how far we are from the 10-second limit. However, the run will be terminated if the run time exceeds the limit so we cannot know the degree to which the constraint was violated. An example of a nonquantifiable constraint is using a binary indicator for recording whether the constraint has been satisfied or violated.

- **Relaxable versus unrelaxable (R/U):** During the process of obtaining optimal solutions, a constraint is called relaxable if it does not need to be satisfied for the objective function to be evaluated. Generally, relaxable constraints are restrictions from outside of the model. For example, a budget or a weight limit only needs to be satisfied by the ultimate solution rather than throughout process. However, an unrelaxable constraint means we must always consider and satisfy the constraint during the entire optimization process.

- **A-Priori versus simulation-based (A/S):** To confirm an a priori constraint's feasibility, there is no need to run an expensive simulation. Simulation-based constraints on the other hand require that a potentially costly simulation be run. An example of an a-priori constraint is a one-side bound, such as a non-negativity constraint, because it is cheap to evaluate whether the solution violates the constraint.

- **Known versus hidden (K/H):** Known means that the solver knows the constraint information, while hidden constraint means the constraint is not explicitly identified. For example, if the optimization problem is $\min \log(x)$ for $x \in \mathbb{R}$, then $x > 0$ is a hidden constraint (as otherwise the objective function is not defined), but it is not expressed in the problem. A hidden constraint cannot be a-priori, quantifiable, or relaxable because we do not know the constraint and cannot quantify it or detect the violation.

In order to classify the constraint-handling techniques, the above classifications can be expressed by the combination of their initial letters. For example, QRSK represents the constraint-handling technique that can solve the quantifiable, relaxable, simulation-based, and known constraints.

2.1.3 Karush-Kuhn-Tucker (KKT) Conditions

Karush-Kuhn-Tucker (KKT) is a set of first-order necessary conditions for the optimal solution of nonlinear programming. To state the necessary conditions, we first need to introduce the Lagrange function. Consider the constrained minimization problem as (2.1), the Lagrange function is defined as below [32, page 321]:

$$\mathcal{L}(\mathbf{x},\ \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^{l} \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^{m} \lambda_{l+j} h_j(\mathbf{x}), \tag{2.4}$$

where $\boldsymbol{\lambda}$ is called the Lagrange multipliers, g_i are the inequality constraints and h_j are the equality constraints. The Lagrange function formulates the objective function, equality and inequality constraints into one equation, with the Lagrange multipliers as the coefficients of the constraints.

If $\mathbf{x}^*$ is a locally optimal solution of (2.1), and f, g_i and h_j are continuously differentiable at $\mathbf{x}^*$, then there exists a Lagrange multiplier vector $\boldsymbol{\lambda}^*$ which consists of $\lambda_i^*, i \in [1, ..., l+m]$ such that the following conditions are satisfied at $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ [32, page 321]

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{2.5a}$$

$$g_i(\mathbf{x}^*) \leq 0 \text{ for } i = 1, 2, ..., l \tag{2.5b}$$

$$h_j(\mathbf{x}^*) = 0 \text{ for } j = 1, 2, ..., m \tag{2.5c}$$

$$\lambda_i^* \geq 0 \text{ for } i = 1, 2, ..., l \tag{2.5d}$$

$$\lambda_i^* g_i(\mathbf{x}^*) = 0 \text{ for } i = 1, 2, ..., l \tag{2.5e}$$

These are the necessary conditions for searching for the optimal solution of the objective function $f(\mathbf{x})$ with constraints g_i and h_j.

(2.5a) states that at the locally optimal solution, the linear combination of the constraint gradients with the coefficients determined by λ^* cancel out the gradient of the objective function. (2.5b) and (2.5c) holds as the optimal solution is feasible. (2.5d) stipulates that the Lagrange multipliers for inequality constraints are either zero (if the constraint is not tight at $\mathbf{x}^*$) or strictly positive (if $\mathbf{x}^*$ lies on the constraint boundary). In the latter case, (2.5e) is satisfied as the value of the constraint function is zero.

2.1.4 Ill-conditioned Problems

Optimization problems may be ill-conditioned. For an optimization problem, if the condition number of the second partial derivatives matrix of the objective function is large, then the objective function is considered to be an ill-conditioned problem [7]. The condition number of a matrix is the ratio of the largest singular value to the smallest. The second partial derivatives matrix of the objective function f of n variables is also called the Hessian matrix. The Hessian matrix is given as follows:

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} & \frac{\partial^2}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} \\ \frac{\partial^2}{\partial x_2 \partial x_1} & \frac{\partial^2}{\partial x_2^2} & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} & \frac{\partial^2}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2}{\partial x_n^2} \end{bmatrix} f(\mathbf{x}) \tag{2.6}$$

To solve the ill-conditioned problem, we can apply Newton's method. Newton's method aims to construct a series of points from the initial guess to the minima of f by looking for a forward direction $\mathbf{p}$ iteratively. According to the second-order Taylor approximation, we have

$$\nabla f(\mathbf{x} + \mathbf{p}) \approx \nabla f(\mathbf{x}) + \nabla^2 f(\mathbf{x})\mathbf{p} = 0. \tag{2.7}$$

Solving (2.7) can obtain $\mathbf{p} = (-\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})$, where $(-\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})$ is called Newton's direction [32].

Figure 2.2 illustrates a problem with quadratic objective function, where thin black lines are contour lines of the objective function, and the optimal solution is at the center of the contour lines. This figure marks the negative of the gradient direction $-\nabla f(\mathbf{x})$ as a green line and Newton's direction $-(\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})$ as a red line. They both point in directions where the objective function value decreases, but it can be seen that the Newton's direction directly points to where the optimal solution is located, while the gradient direction does not immediately.

2.2 Evolution Strategy

The principle of biological evolution inspires evolution strategies (ES) [15]. ES are evolutionary algorithms that date back to the 1960s. ES solve black-box optimization problems within continuous search spaces by doing mutation, recombination, and

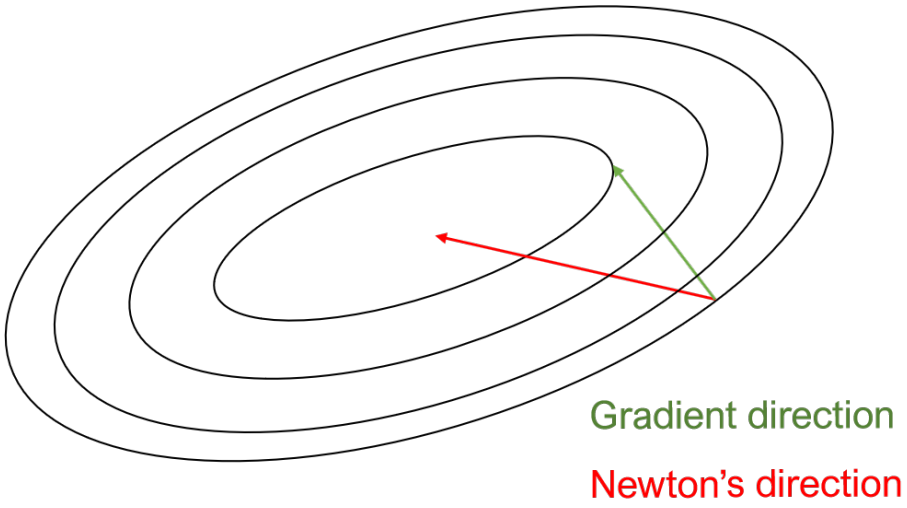


Figure 2.2: Ill-conditioned problem with gradient direction and Newton direction. Figure adapted from [6]

selection. In ES, individuals are denoted as $\mathbf{x} \in \mathbb{R}^n$, a population may contain one or more individuals. Generally, ES is an iterative process. People commonly use mnemonic notation to describe some features of the iteration [17]. The $(\mu/\rho \dagger, \lambda)$-ES, where μ is the number of parent individuals, ρ is the number of parent individuals for recombination used, and λ is the number of offspring generated in each iteration. The symbol $\dagger$ denotes the selection process. ES implements either the 'plus' or 'comma' selection. In each iteration of ES (see Figure 2.3), λ individuals are generated by mutation. The fitness of individuals in the population is determined by evaluating them with their objective function values. Selection is conducted according to each individual's fitness. In 'plus' selection, the strategy selects the μ best candidate solutions from μ parents and λ offspring individuals. In 'comma' selection, only the λ offspring are eligible for selection. Recombination generates a new offspring from ρ out of μ selected individuals, where $\rho \leq \mu$, but recombination is not always necessary. For example, in (1+1)-ES, we conduct the mutation operator based on one parent, generate one offspring, and then select the superior offspring as the parent of the next generation. In this section, we will focus on (1+1)-ES, $(\mu/\mu, \lambda)$-ES and $(\mu/\mu, \lambda)$-CMA-ES.

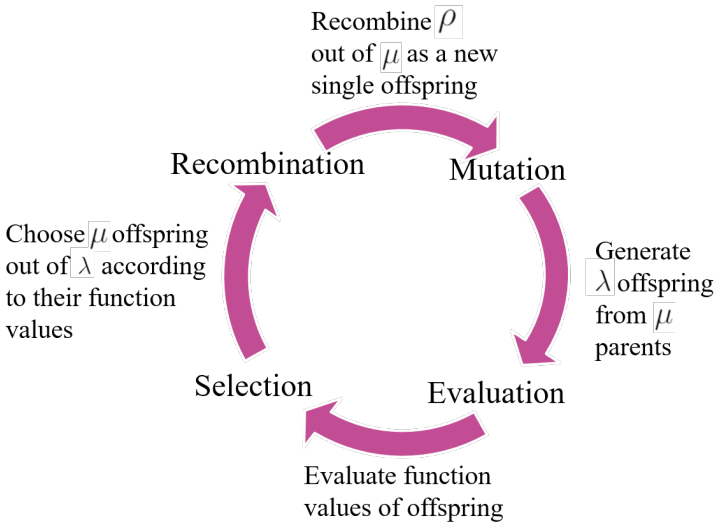


Figure 2.3: Evolution strategy iteration.

2.2.1 (1+1)-ES

(1+1)-ES is the simplest evolution strategy because there is only one offspring is generated by a mutation operator applied to a parent in each iteration. During the minimization process, new offspring $\mathbf{y}^{(k)}$updates by replacing its parent $\mathbf{x}^{(k)}$ when the offspring's function value is smaller than the parent's, where the superscript indicates the iteration number.

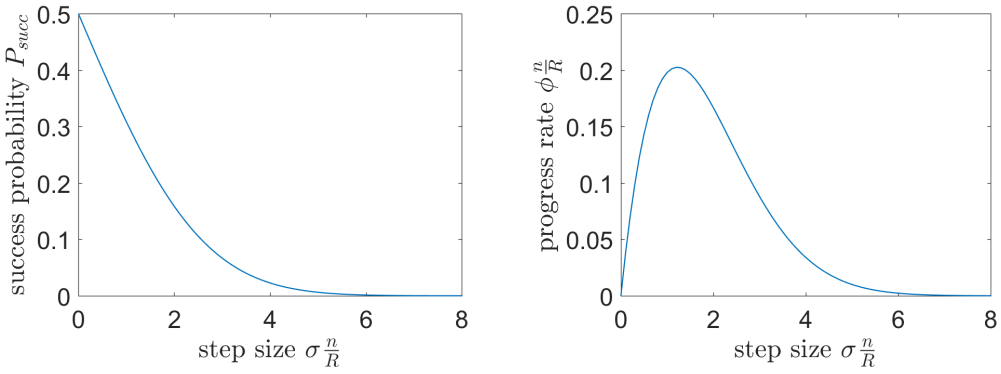
The mutation operator adds a symmetric point perturbation to the parent, and the perturbation comes from a multivariate normal distribution, $\mathcal{N}(\mathbf{0}, \mathbf{C})$, with zero mean value and covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$ [17]. The mutation operator in (1+1)-ES is spherical/isotropic, which means the covariance matrix $\mathbf{C}$ is proportional to the identity matrix $\mathbf{I}$. The mutation distribution follows $\sigma \times \mathcal{N}(\mathbf{0}, \mathbf{I})$ in (1+1)-ES where $\sigma \in \mathbb{R}^{+}$ represents step-size.

Tuning the step-size σ is important because it influences the performance of the ES. A small σ may lead to a high success probability, but it slows down the progress towards the solution. The success probability, denoted by P_{succ}, is the probability that an offspring's function value is better than a parent's. The progress rate, ϕ, is defined as the expected distance change of the parent towards the optimum point. Having too large of a σ decreases the probability of stepping towards the optimum [8]. Based on the fact that both P_{succ} and ϕ depend on σ, Rechenberg [34] develops

a step-size adaptation mechanism for (1+1)-ES by calculating the optimum success probability of the quadratic sphere:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \tag{2.8}$$

The success probability P_{succ} and progress rate ϕ calculated from the sphere model (2.8) are shown in Figure 2.4 when n approached ∞. It can be seen that the success probability curve decreases as the step-size increases. In contrast, the progress rate curve increases first and then decreases as the step-size increases. The x-axis is represented as $\sigma\frac{n}{R}$, where n indicates the dimension of the problem, and R implies the distance between the candidate solution to the optimal solution. The choice of step-size should be small because a smaller step-size can have a higher probability of success, but too small a step-size will lead to slow progress. Therefore, it is important to find a balance that makes a proper step-size for achieving a high progress rate. We can see that the progress rate reaches the optimal value when the corresponding success probability is ≈ 0.270. Other functions may have different optimal success



(a) Success probability. (b) Progress rate.

Figure 2.4: Progress rate and success probability of (1+1)-ES on sphere model.

probabilities, for example, the corridor function [34] which is defined as

$$f(\mathbf{x}) = \begin{cases} x_1 & \text{if } |x_i| < 1 \text{ for } i = 2, 3, ..., n \\ \infty & \text{otherwise.} \end{cases} \tag{2.9}$$

The optimal success probability of this function is ≈ 0.184 [34]. As a compromise between these two probabilities, Rechenberg proposed 0.2 as a target for success

probabilities in the (1+1)-ES. In summary, to gain nearly optimal performance in (1+1)-ES, we increase the step-size if $P_{succ} > 1/5$ and we decrease it when $P_{succ} < 1/5$. If $P_{succ} = 1/5$, the step-size does not change. This approach is also known as the 1/5th rule.

Algorithm 1 shows (1+1)-ES with the 1/5th rule [17]. $\sigma^{(k)}$ represents the step-size and $\mathbf{x}^{(k)}$ indicates the parent at the kth iteration, while $\mathbf{y}^{(k)}$ is the offspring of the kth iteration. Line 4 implements mutation, which ensures unbiased random mutation on $\mathbf{x}^{(k)}$ to generate offspring $\mathbf{y}^{(k)}$. Line 5 updates $\sigma^{(k)}$ by the 1/5th rule to become $\sigma^{(k+1)}$ in $(k+1)$th iteration. Kern et al. [22] proposed a simple implementation

Algorithm 1 (1+1)-ES with 1/5th rule [17]

1: given $n, k \in \mathbb{N}_+, d \approx \sqrt{n+1}$
2: Initialize $\sigma^{(0)} \in \mathbb{R}^+, \mathbf{x}^{(0)} \in \mathbb{R}^n$
3: **while** the termination criterion is not met **do**
4: $\quad \mathbf{y}^{(k)} \leftarrow \mathbf{x}^{(k)} + \sigma^{(k)} \times \mathcal{N}(\mathbf{0}, \mathbf{I})$ // mutation
5: $\quad \sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp^{1/d}(\mathbb{1}_{f(\mathbf{y}^{(k)}) \leq f(\mathbf{x}^{(k)})} - 1/5)$
6: $\quad$ **if** $f(\mathbf{y}^{(k)}) \leq f(\mathbf{x}^{(k)})$ **then**
7: $\quad\quad \mathbf{x}^{(k+1)} \leftarrow \mathbf{y}^{(k)}$
8: $\quad$ **else**
9: $\quad\quad \mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)}$
10: $\quad$ **end if**
11: $\quad k \leftarrow k + 1$
12: **end while**

of the 1/5th rule. Rather than estimating the success probability, they adapt the step-size by using $\sigma^{(k+1)} = \sigma^{(k)} \exp^{1/d}(\mathbb{1}_{f(\mathbf{y}^{(k)}) \leq f(\mathbf{x}^{(k)})} - 1/5)$ where $\mathbb{1}_{f(\mathbf{y}^{(k)}) \leq f(\mathbf{x}^{(k)})}$ is an indicator function. The indicator function equals to one if condition $f(\mathbf{y}^{(k)}) \leq f(\mathbf{x}^{(k)})$ is satisfied, which means the offspring mutated by the parent is successful, and the step-size increases by the factor of $e^{0.8/d}$. If not, $\mathbb{1}_{f(\mathbf{y}^{(k)}) \leq f(\mathbf{x}^{(k)})}$ equals zero, and σ decreases by the factor of $e^{-0.2/d}$. Lines 6-10 display the selection steps. The function values of the new candidate solution and the best solution found up to this point are compared. If the new candidate solution's function value is less than or equal to the current best solution, $\mathbf{y}^{(k)}$ replaces $\mathbf{x}^{(k)}$ as the parent of next iteration. If not, keep

the parent $\mathbf{x}^{(k)}$. Repeat Line 3-11 until the optimal solution is found or some other stop criterion is met.

2.2.2 $(\mu/\mu, \lambda)$-ES

The next type of ES we consider is the $(\mu/\mu, \lambda)$-ES [17]. Comparing with (1+1)-ES, $(\mu/\mu, \lambda)$-ES generates λ candidate solutions in each iteration and selects the μ best candidates according to their function values. In order to integrate the information of μ candidates, a recombination mechanism is introduced. This mechanism combines the μ best parents and produces a recombinant individual.

Apart from recombination, another step-size adaptation strategy is commonly used in $(\mu/\mu, \lambda)$-ES, called the cumulative step-size adaptation (CSA). CSA accumulates the relationship between past consecutive steps. If there is a positive correlation between consecutive steps, then the steps can be replaced by fewer and longer steps in the same direction. Algorithm 2 introduces $(\mu/\mu, \lambda)$-ES with CSA and the recombination mechanism.

Algorithm 2 $(\mu/\mu, \lambda)$-ES [17]

1: given $n, k \in \mathbb{N}_+, \lambda \geq 5n, \mu \approx \lambda/4 \in \mathbb{N}, \tau \approx 1/\sqrt{n}$
2: initialize $\mathbf{x}^{(0)} \in \mathbb{R}^n, \sigma^{(0)} \in \mathbb{R}^n_+$
3: **while** the termination criterion is not met **do**
4: **for** $m \in \{1, ..., \lambda\}$ **do**
5: $\mathbf{z}_m^{(k)} = \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: $\mathbf{y}_m^{(k)} = \mathbf{x}^{(k)} + \sigma_m^{(k)} \mathbf{z}_m^{(k)}$ // mutation
7: Compute $f(\mathbf{y}_m^{(k)})$
8: $\mathcal{P} = sel\_\mu\_best(\{(\mathbf{y}_m^{(k)}, \mathbf{z}_m^{(k)}, f(\mathbf{y}_m^{(k)})) \mid 1 \leq m \leq \lambda\})$
9: $\mathbf{s}_\sigma^{(k+1)} \leftarrow (1 - c_\sigma)\mathbf{s}_\sigma^{(k)} + \sqrt{c_\sigma(2 - c_\sigma)}\frac{\sqrt{\mu}}{\mu} \sum_{\mathbf{z}_m^{(k)} \in \mathcal{P}} \mathbf{z}_m^{(k)}$
10: $\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp^{c_\sigma/d_\sigma}(\frac{||\mathbf{s}_\sigma^{(k+1)}||}{\mathbb{E}||\mathcal{N}(\mathbf{0},\mathbf{I})||} - 1)$
11: $\mathbf{x}^{(k+1)} = \frac{1}{\mu} \sum_{\mathbf{y}_m^{(k)} \in \mathcal{P}} \mathbf{y}_m^{(k)}$
12: $k \leftarrow k + 1$
13: **end while**

In a single iteration, Algorithm 2 generates λ candidate offspring by mutating

from $\mathbf{x}^{(k)}$ in Lines 4-6, where $\mathbf{y}_m^{(k)}$ indicates the mth offspring at the kth iteration. Mutation distribution is still isotropic here. Each candidate offspring's function value is computed in Line 7. Then the μ best offspring from the λ candidates are selected based on their function values (Line 8). Lines 9-10 implement the cumulative step-size adaptation. $\mathbf{s}_\sigma$ is an exponentially fading record of mutation steps. This record accumulates a sequence of consecutive successful $\mathbf{z}_m^{(k)}$ steps, and is also called the search path. Accumulating successful steps into the search path allows for exploiting correlations between successive steps. For example, if two successful mutation steps are going in the same direction, $\mathbf{s}_\sigma^{(k+1)}$ will be comparatively long. If they are going in the opposite direction, $\mathbf{s}_\sigma^{(k+1)}$ will be comparatively short [18]. c_σ in Line 9 indicates the cumulative number of iterations of $\mathbf{s}_\sigma^{(k+1)}$. If $c_\sigma = 1$, then $(1 - c_\sigma)\mathbf{s}_\sigma^{(k)} = 0$, and so the prior information is not retained [18]; if $c_\sigma = 0$, then $\mathbf{s}_\sigma^{(k+1)} = \mathbf{s}_\sigma^{(k)}$, and no learning takes place. $\sqrt{c_\sigma(2 - c_\sigma)\mu}$ is a normalization constant to ensure that $s_\sigma^{(k+1)}$ is standard normally distributed if the selection of mutation vectors is random. The reason is if $\mathbf{z}$ are independently standard normally distributed, then their average is normally distributed. The normalization constant $\sqrt{c_\sigma(2 - c_\sigma)\mu}$ cancels out the μ, resulting in $\mathbf{s}_\sigma^{(k+1)}$ having standard normally distributed components. $\sigma^{(k+1)}$ is updated by comparing the length of search path $||\mathbf{s}_\sigma^{(k+1)}||$ with the expected length of $\mathcal{N}(\mathbf{0}, \mathbf{I})$ (Line 10). When $||\mathbf{s}_\sigma^{(k+1)}|| = \mathbb{E}||\mathcal{N}(\mathbf{0}, \mathbf{I})||$, $\exp^{c_\sigma/d_\sigma}(\frac{||\mathbf{s}_\sigma^{(k+1)}||}{\mathbb{E}||\mathcal{N}(\mathbf{0},\mathbf{I})||} - 1) = 1$, and $\sigma^{(k+1)} = \sigma^{(k)}$. If $||\mathbf{s}_\sigma^{(k+1)}|| < \mathbb{E}||\mathcal{N}(\mathbf{0}, \mathbf{I})||$, $\sigma^{(k+1)}$ is decreased; otherwise, $\sigma^{(k+1)}$ is increased. d_σ is a damping parameter to decide the scale change of $\sigma^{(k)}$ [16].

Recombination is conducted in Line 11. $\mathbf{x}^{(k+1)}$ at the $(k+1)$th iteration is obtained by averaging the current best children over μ [17]. The recombination used here is intermediate recombination, which computes the average value among all selected offspring.

2.2.3 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

So far, only isotropic mutation operators have been used. In covariance matrix adaptation evolution strategy (CMA-ES), offspring are sampled as $\mathbf{x} + \sigma \times \mathcal{N}(\mathbf{0}, \mathbf{C})$ where the covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$ determines the shape of the mutation ellipse. A symmetric and positive definite covariance matrix is proposed in CMA-ES so that the distribution of mutation can adjust with the shape of the objective function.

This is useful, especially in ill-conditioned problems [16]. For example, if the objective function contours are an ill-conditioned ellipsoid, the mutated distribution can adjust as the same shape to the objective function contours with a general covariance matrix.

Algorithm 3 $(\mu/\mu, \lambda)$-CMA-ES [17]

1: initialize $\mathbf{s}_\sigma^{(0)} = \mathbf{0}, \mathbf{s}_c^{(0)} = \mathbf{0}, \mathbf{C}^{(0)} = \mathbf{I}, \mathbf{x}^{(0)} \in \mathbb{R}^n, \sigma^{(0)} \in \mathbb{R}_+^n, k \in \mathbb{N}^+$
2: **while** the termination criterion is not met **do**
3: **for** $m \in \{1, ..., \lambda\}$ **do**
4: $\mathbf{z}_m^{(k)} = \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: $\mathbf{y}_m^{(k)} = \mathbf{x}^{(k)} + \sigma^{(k)}(\mathbf{C}^{(k)})^{1/2} \times \mathbf{z}_m^{(k)}$
6: $\mathcal{P} = sel\_\mu\_best(\{(\mathbf{y}_m^{(k)}, \mathbf{z}_m^{(k)}, f(\mathbf{x}_m^{(k)})) \mid 1 \leq m \leq \lambda\})$
7: $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \sigma^{(k)}(\mathbf{C}^{(k)})^{1/2} \sum_{\mathbf{z}_m^{(k)} \in \mathcal{P}} \mathbf{z}_m^{(k)}/\mu$
8: $\mathbf{s}_\sigma^{(k+1)} \leftarrow (1 - c_\sigma)\mathbf{s}_\sigma^{(k)} + \sqrt{c_\sigma(2 - c_\sigma)}\frac{\sqrt{\mu}}{\mu} \sum_{\mathbf{z}_m^{(k)} \in \mathcal{P}} \mathbf{z}_m^{(k)}$
9: $\mathbf{s}_c^{(k+1)} \leftarrow (1 - c_c)\mathbf{s}_c^{(k)} + \sqrt{c_c(2 - c_c)}\frac{\sqrt{\mu}}{\mu} \sum_{\mathbf{z}_m^{(k)} \in \mathcal{P}} (\mathbf{C}^{(k)})^{1/2}\mathbf{z}_m^{(k)}$
10: $\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp^{c_\sigma/d_\sigma}(\frac{||\mathbf{s}_\sigma^{(k+1)}||}{\mathbb{E}||\mathcal{N}(\mathbf{0},\mathbf{I})||} - 1)$
11: $\mathbf{C}^{(k+1)} \leftarrow (1 - c_{cov})\mathbf{C}^{(k)} + c_{cov} \cdot \mathbf{s}_c^{(k+1)} \cdot (\mathbf{s}_c^{(k+1)})^\intercal$
12: $k \leftarrow k + 1$
13: **end while**

The $(\mu/\mu, \lambda)$-CMA-ES algorithm is described in Algorithm 3. The CMA-ES algorithm described here has some differences from the one in [17] because they use different recombination operators. It samples λ candidate solutions at the beginning (Lines 3-5). $\mathbf{x}^{(k)}$ is the parent at the kth iteration and $\mathbf{y}^{(k)}$ are the offsprings. The algorithm selects the μ best offspring $\mathbf{y}_m^{(k)}$, corresponding mutation $\mathbf{z}_m^{(k+1)}$ and function values $f(\mathbf{y}_m^{(k)})$ from the λ offspring (Line 6). $\mathbf{x}^{(k+1)}$ (Line 7) is generated by covariance matrix $\mathbf{C}^{(k)}$ and the average of the selected $\mathbf{z}_m^{(k)}$ over μ.

Two search paths are updated as follows. The first path is $\mathbf{s}_\sigma^{(k+1)}$ (Line 8), which is the same as the $\mathbf{s}_\sigma^{(k+1)}$ in Algorithm 2. The second path $\mathbf{s}_c^{(k+1)}$ (Line 9) is a cumulation for $\mathbf{C}$ [16]. c_c works as the same as c_σ [18]. The update of $\sigma^{(k+1)}$ (Line 10) is the same as the $\sigma^{(k+1)}$ updating in Algorithm 2. Comparing $||\mathbf{s}_\sigma^{(k+1)}||$ with its expected length $\mathbb{E}||\mathcal{N}(\mathbf{0}, \mathbf{I})||$ and increasing $\sigma^{(k+1)}$ if $||\mathbf{s}_\sigma^{(k+1)}|| > \mathbb{E}||\mathcal{N}(\mathbf{0}, \mathbf{I})||$, decreasing $\sigma^{(k+1)}$

if $||\mathbf{s}_\sigma^{(k+1)}|| < \mathbb{E}||\mathcal{N}(\mathbf{0},\mathbf{I})||$. $\sigma^{(k+1)}$ does not change when $||\mathbf{s}_\sigma^{(k+1)}|| = \mathbb{E}||\mathcal{N}(\mathbf{0},\mathbf{I})||$ [16]. The parameter settings for c_σ and d_σ are given in [18]. Line 11 describes the adaptation of the covariance matrix $\mathbf{C}^{(k+1)}$. It takes place by means of the evolution path $\mathbf{s}_c^{(k+1)}$. The shape of the covariance matrix of the next generation, $\mathbf{C}^{(k+1)}$, is decided by $\mathbf{C}^{(k)}$ and $\mathbf{s}_c^{(k+1)}$. c_{cov} is the change rate of the covariance matrix, which has the same effect as c_c and c_σ [18].

Compared to the (1+1)-ES, Algorithm 3 introduces two search paths, $\mathbf{s}_c$ and $\mathbf{s}_\sigma$. Maintaining these two search paths $\mathbf{s}_c$ and $\mathbf{s}_\sigma$ has two benefits: first, the paths can filter high-frequency information, which is most likely noisy. Second, they can adjust the shape of the mutation distribution adaptively according to the shape of the objective function, which makes CMA-ES perform well in ill-conditioned problems [18].

2.3 Deterministic Algorithms for Constrained Optimization

There are several deterministic algorithms for constrained optimization, such as interior-point methods, Lagrange multiplier methods, and active-set methods. In this section, we are going to introduce two deterministic methods that we used for comparison in our experiment. One is active-set methods, the other is an interior-point method.

2.3.1 Active-set Methods

Active-set methods have been widely used for solving constrained optimization problems since the 1970s. An active set $\mathcal{A}^{(k)}$ is a set to store active constraints at the kth iterate $\mathbf{x}^{(k)}$. The idea of active-set methods is to convert the optimization problem to an easier subproblem, maintain a set of active constraints, and find a step from one iterate to the next. Finding a step is conducted by solving the subproblem within the feasible region that the active set determines until an optimal point is found. For constrained optimization, equality constraints are always active, and inequality constraints become active when they are tight [32, page 467].

Active-set methods convert the objective function to quadratic programming (QP) subproblems, which is the process of solving a quadratic optimization problem in each step based on the approximation of the first three terms of the Taylor series. It

optimizes a quadratic function of multiple variables subjected to linear constraints of these variables. The general quadratic program can be denoted as

$$\begin{aligned} \min_{\mathbf{x}} \quad & q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\intercal \mathbf{H}\mathbf{x} + \mathbf{x}^\intercal \mathbf{c} \\ \text{subject to} \quad & \mathbf{a}_k^\intercal \, \mathbf{x} \; = b_k, \qquad k \in [1, ..., m], \\ & \mathbf{r}_t^\intercal \, \mathbf{x} \; \leq u_t, \qquad t \in [1, ..., l] \end{aligned} \tag{2.10}$$

where $\mathbf{H}$ is the $n \times n$ Hessian matrix (2.6), $\mathbf{c}, \mathbf{x}$ are vectors in $\mathbb{R}^n$, and $\mathbf{a}_k$ and $\mathbf{r}_t$ represent the coefficients of linear equality constraints and inequality constraints, respectively. If $\mathbf{H}$ is positive semi-definite ($\mathbf{x}^\intercal \mathbf{H}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$), we say that (2.10) is a convex problem. On the other hand, if $\mathbf{H}$ is an indefinite matrix ($\mathbf{x}^\intercal \mathbf{H}\mathbf{x} < 0$ for some $\mathbf{x} \in \mathbb{R}^n$), then (2.10) is a nonconvex problem, which means the $q(\mathbf{x})$ may have several local minima in multiple feasible regions or be indefinite, and it is difficult to find global optimal solution. For this section, we shall focus on convex quadratic programming.

The goal of active-set methods is to find a step from the current iterate to the next in the feasible region by solving a QP subproblem aiming to make the objective function value decrease. Therefore, defining the step as $\mathbf{p}$, the QP subproblems can be established with $\mathbf{p}$ as the variable. This method is also called active-set sequential quadratic programming (SQP) methods because obtaining the optimal solution of the objective function requires solving a series of quadratic programming subproblems. The nonlinear constrained problem is linearized in a QP subproblem as below:

$$\begin{aligned} \min_{\mathbf{p}} \quad & q(\mathbf{x}^{(k)} + \mathbf{p}^{(k)}) = \mathbf{l}^{(k)\intercal} \mathbf{p}^{(k)} + \frac{1}{2}\, \mathbf{p}^{(k)\intercal} \mathbf{H}^{(k)}\, \mathbf{p}^{(k)} \\ \text{subject to} \quad & \nabla h_j(\mathbf{x}^{(k)})^\intercal \mathbf{p}^{(k)} + h_j(\mathbf{x}^{(k)}) \; = 0, \qquad j \in [1, ..., m], \\ & \nabla g_i(\mathbf{x}^{(k)})^\intercal \, \mathbf{p}^{(k)} + g_i(\mathbf{x}^{(k)}) \; \leq 0, \qquad i \in [1, ..., l]. \end{aligned} \tag{2.11}$$

where $\mathbf{l}^{(k)} = \nabla f(\mathbf{x}^{(k)})$.

Figure 2.5 displays the procedure of active-set methods for a single iteration k. In each iteration, we solve (2.11) to get $\mathbf{p}^{(k)}$. If $\mathbf{p}^{(k)}$ is nonzero, the algorithm needs to define a step length $\alpha^{(k)}$ to decide how far $\mathbf{x}^{(k)}$ is going to move. Determining the value of $\alpha^{(k)}$ is ensuring that the new point $\mathbf{x}^{(k+1)}$ is in the feasible region. If $\mathbf{x}^{(k)} + \mathbf{p}^{(k)}$ is feasible with respect to all constraints, then $\alpha^{(k)} = 1$ and $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{p}^{(k)}$. Otherwise, $\alpha^{(k)}$ is a non-negative number between $[0, 1]$. It is possible

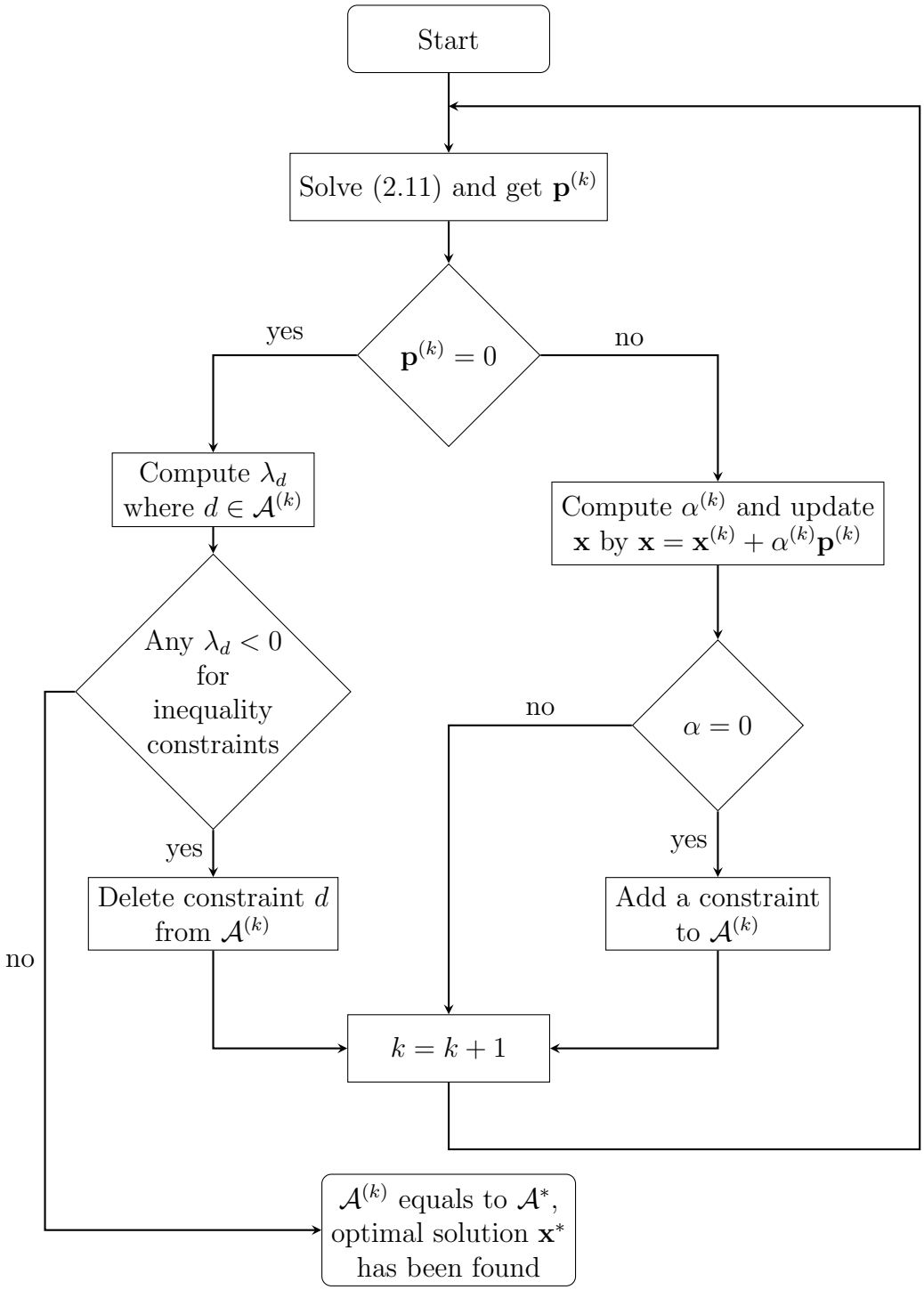


Figure 2.5: Flow chart of active-set methods.

for $\alpha^{(k)} = 0$, which means there are other constraints that should have been added to $\mathcal{A}^{(k)}$ but were not added at current iteration point $\mathbf{x}^{(k)}$. Under this circumstance, the algorithm should update $\mathcal{A}^{(k)}$ by adding a constraint to get a new active set

$\mathcal{A}^{(k+1)}$. Repeating this procedure until a point $\mathbf{x}^{(k)}$ which minimizes the quadratic programming subproblem is found with current active set $\mathcal{A}^{(k)}$. To verify that $\mathbf{x}^{(k)}$ is the global optimal solution $\mathbf{x}^*$ of the original objective function, the algorithm checks whether the corresponding $\lambda_d, d \in \mathcal{A}^{(k)}$ for each active inequality constraint satisfies the KKT conditions in (2.5) . If any $\lambda_d < 0$, the (2.5d) condition is not met, algorithm deletes constraint d from $\mathcal{A}^{(k)}$ and solves this new subproblem (2.11) to obtain a new step. If $\mathbf{x}^{(k)}$ satisfies all KKT conditions, then it is the global optimal solution $\mathbf{x}^*$ for the original problem (2.10), and $\mathcal{A}^{(k)}$ equals to $\mathcal{A}^*$, and the algorithm terminates.

2.3.2 Interior-point Method

Interior-point methods were first introduced in the 1960s [32, page 563]. They typically look for a solution from the inside of the feasible area. The interior-point method that we used in our experiment is the barrier method. The idea of the barrier method is to convert the constrained problem into a series of approximate minimization problems by adding a barrier function. The logarithm function is used here so that it prevents the algorithm from generating points that are too close to the boundary of the feasible region. Associating equation (2.1) with the barrier function, we have:

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{s}} f(\mathbf{x}) - \mu \sum_{i=1}^{l} \log(s_i) \\ &\text{subject to } \begin{cases} g_i(\mathbf{x}) + s_i = 0 & i \in [1, ..., l] \\ h_j(\mathbf{x}) = 0 & j \in [1, ..., m] \end{cases} \end{aligned} \tag{2.12}$$

where μ is a positive parameter. It can be seen that the inequality constraints $g_i(\mathbf{x}) \leq 0$ are transformed into equality constraints with a vector $\mathbf{s}$. Hidden constraints $s_i \geq 0$ are not included in (2.12) because minimization of $-\mu \sum_{i=1}^{l} \log(s_i)$ prevents s_i being too close to zero. The number of s_i is the same as the number of inequality constraints g_i. The value of s_i is strictly positive to ensure $\log(s_i)$ is valid.

Incorporating the constraints into the objective function by adding Lagrange multipliers, we have Lagrangian function

$$\mathcal{L}(\mathbf{x}, \mathbf{s}, \boldsymbol{\lambda}_\mathbf{y}, \boldsymbol{\lambda}_\mathbf{z}) = f(\mathbf{x}) + \mu \sum_{i=1}^{l} \log(s_i) + \sum_{i=1}^{l} \lambda_i (g_i(\mathbf{x}) + s_i) + \sum_{j=1}^{m} \lambda_{l+j} h_j(\mathbf{x}) \tag{2.13}$$

where $\boldsymbol{\lambda}_{\mathbf{y}} = (\lambda_{l+1}, \ldots, \lambda_{l+m})$ and $\boldsymbol{\lambda}_{\mathbf{z}} = (\lambda_1, \ldots, \lambda_l)$ are the corresponding Lagrange multipliers of equality and inequality constraints, respectively. The gradient of the Lagrangian in (2.13) with respect to $\mathbf{x}$ is

$$\nabla_{\mathbf{x}}\mathcal{L} = \nabla f(\mathbf{x}) + A_E^{\intercal}(\mathbf{x})\boldsymbol{\lambda}_{\mathbf{y}} + A_I^{\intercal}(\mathbf{x})\boldsymbol{\lambda}_{\mathbf{z}} \tag{2.14}$$

where $A_E(\mathbf{x})$ and $A_I(\mathbf{x})$ are the matrices of constraints gradients, that are

$$\begin{aligned} A_E^{\intercal}(\mathbf{x}) &= [\nabla h_1(\mathbf{x}), \nabla h_2(\mathbf{x}), ..., \nabla h_m(\mathbf{x})], \\ A_I^{\intercal}(\mathbf{x}) &= [\nabla g_1(\mathbf{x}), \nabla g_2(\mathbf{x}), ..., \nabla g_l(\mathbf{x})]. \end{aligned} \tag{2.15}$$

The partial derivative of the Lagrangian in (2.13) with respect to s_i is

$$\frac{\partial \mathcal{L}}{\partial s_i} = -\mu s_i + \lambda_i \tag{2.16}$$

Both (2.14) and (2.16) equal zero at the optimal solution $\mathbf{x}^*$, combining them with the equality and inequality constraints in (2.12), we have below equations at the optimal solution:

$$\nabla f(\mathbf{x}^*) + A_E^{\intercal}(\mathbf{x}^*)\boldsymbol{\lambda}_{\mathbf{y}}^* + A_I^{\intercal}(\mathbf{x}^*)\boldsymbol{\lambda}_{\mathbf{z}}^* = 0 \tag{2.17a}$$

$$S\boldsymbol{\lambda}_{\mathbf{z}}^* + \mu\mathbf{e} = 0 \tag{2.17b}$$

$$h_j(\mathbf{x}^*) = 0 \tag{2.17c}$$

$$g_i(\mathbf{x}^*) + s_i^* = 0. \tag{2.17d}$$

where S is the diagonal matrix of $\mathbf{s}$ and $\mathbf{e} = (1, 1, ..., 1)^{\intercal}$.

The Newton's method of determining root is applied here to find a direction forward at the current point. The search step can be denoted as $\mathbf{p}^{(k)} = (\mathbf{p}_{\mathbf{x}}^{(k)}, \mathbf{p}_{\mathbf{s}}^{(k)}, \mathbf{p}_{\boldsymbol{\lambda}_{\mathbf{y}}}^{(k)}, \mathbf{p}_{\boldsymbol{\lambda}_{\mathbf{z}}}^{(k)})$ [32, page 566]. However, Newton's method iterates with a fixed step length because it does not have a step length parameter. As such, appropriate step length α is introduced in the interior-point method for better performance. The choice of $\alpha \in (0, 1]$ should make $\mathbf{x}$ and $\mathbf{s}$ satisfy the corresponding constraints [32, page 567]. The new iterate $(\mathbf{x}^{(k+1)}, \mathbf{s}^{(k+1)}, \boldsymbol{\lambda}_{\mathbf{y}}{}^{(k+1)}, \boldsymbol{\lambda}_{\mathbf{z}}{}^{(k+1)})$ can be denoted as

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \boldsymbol{\alpha}_{\mathbf{s}}^{(k)}\mathbf{p}_x^{(k)}, \quad \mathbf{s}^{(k+1)} = \mathbf{s}^{(k)} + \boldsymbol{\alpha}_{\mathbf{s}}^{(k)}\mathbf{p}_{\mathbf{s}}^{(k)}, \\ \boldsymbol{\lambda}_{\mathbf{y}}{}^{(k+1)} &= \boldsymbol{\lambda}_{\mathbf{y}}{}^{(k)} + \boldsymbol{\alpha}_{\mathbf{z}}^{(k)}\mathbf{p}_{\boldsymbol{\lambda}_{\mathbf{y}}}^{(k)}, \quad \boldsymbol{\lambda}_{\mathbf{z}}^{(k+1)} = \boldsymbol{\lambda}_{\mathbf{z}}^{(k)} + \boldsymbol{\alpha}_{\mathbf{z}}^{(k)}\mathbf{p}_{\boldsymbol{\lambda}_{\mathbf{z}}}^{(k)}. \end{aligned} \tag{2.18}$$

Algorithm 4 Interior-point algorithm [32, page 568]

1: initialize $\mathbf{s}^{(0)} > 0$, set initial barrier parameter $\mu^{(0)} > 0$.
2: **while** the termination criterion is not met **do**
3: **if** $E^{(k)} \leq \epsilon$ **then**
4: Obtain the search direction $\mathbf{p}^{(k)}$;
5: Compute the step length $\boldsymbol{\alpha}^{(k)}$;
6: Compute new iterate by (2.18);
7: Update $\mu^{(k)}$
8: **end if**
9: **end while**

Basic steps of the interior-point algorithm are summarized in Algorithm 4 [32, page 565]. The new iterate is generated by $\mathbf{p}^{(k)}$ and $\alpha^{(k)}$. Updating the barrier parameter $\mu^{(k)}$ and repeating these procedures until the optimal solution is found.

We notice that Line 3 mentions an error function $E^{(k)}$. It is the condition of whether or not the current point satisfies the KKT conditions to some accuracy $\epsilon > 0$. The error function is defined as:

$$\begin{aligned} E^{(k)} = \max\{&||\nabla f(\mathbf{x}^{(k)}) + A_E^{\intercal}(\mathbf{x}^{(k)})\boldsymbol{\lambda}_{\mathbf{y}}^{(k)} + A_I^{\intercal}(\mathbf{x}^{(k)})\boldsymbol{\lambda}_{\mathbf{z}}^{(k)}||, \\ &||S\boldsymbol{\lambda}_{\mathbf{z}}^{(k)} - \mu^{(k)}\mathbf{e}||,\ ||h_j(\mathbf{x}^{(k)})||,\ ||g_i(\mathbf{x}^{(k)}) + \mathbf{s}^{(k)}||\} \end{aligned} \tag{2.19}$$

where $||\cdot||$ is the vector norm. The algorithm repeats until a point is found that satisfies all KKT conditions. The principle of updating the barrier parameter $\mu^{(k)}$ is to ensure the sequence of $\mu^{(k)}$ converges to zero so that the optimal point meets the KKT conditions of the original objective function.

Instead of swapping inequality constraints back and forth in active-set methods, interior-point methods search the optimum by gradually decreasing μ. The drawback of interior-point methods is they usually require a feasible initial point which is not easy to guarantee for every problem. Both approaches play a significant role in solving nonlinear optimization problems.

2.4 Evolutionary Algorithms for Constrained Optimization

There are some evolutionary algorithms (EA) for constrained optimization. EA aim to solve difficult objective functions, such as non-differentiable or discontinuous ones.

In order to be able to solve constrained optimization problems, constraint handling techniques need to be introduced. In this section, we are going to present some constraint-handling approaches that were introduced early in the literature [10], followed by a discussion of some recent constraint-handling evolution strategies. An adaptive version of the differential evolution is also considered, that is LSHADE44 [33], which we used in our experiment.

2.4.1 Pioneering Constraint-Handling Approaches

Some of the pioneering work on constraint-handling methods were created by Mezura-Montes and Coello Coello [27] who categorized these methods into four main groups: penalty functions, special representations and operators, repair algorithms and separation of objective function and constraints.

Penalty functions are common methods to solve a constrained problems and were first proposed by Courant in 1943 [11]. The normal approach of penalty functions is transforming a constrained problem to an unconstrained problem, and then measuring the degree to which a point violates the constraints. In mathematical programming, penalty methods are categorized as interior methods and exterior methods. In interior cases, when a point is far from the constraint boundary, the penalty function value is small, and when it is close to the constraint boundary, the penalty function value tends to infinity. Therefore, constraint boundaries are similar to barriers which prevent points from getting into the infeasible region. The method used in Section 2.3.2 is an example of an interior method. On the other hand, exterior methods design a function so that the penalty function value is equal to the original objective function value at a feasible point, but equals the original objective function value plus a large constraint positive at an infeasible point. The exterior penalty functions can be written as [27]:

$$\Phi(\mathbf{x}) = f(\mathbf{x}) + d(\mathbf{x}) \tag{2.20}$$

where $\Phi(\mathbf{x})$ is the new objective function with penalty elements $d(\mathbf{x})$. $d(\mathbf{x})$ can be calculated as follows:

$$d(\mathbf{x}) = \sum_{i=1}^{l} r_i \times \max\,[0, g_i(\mathbf{x})]^2 + \sum_{j=1}^{m} c_j \times |h_j(\mathbf{x})| \tag{2.21}$$

where r_i and c_j are penalty factors, which are normally positive constants during the entire evolutionary process. The equality and inequality constraints are treated differently because an equality constraint is violated only when $h_j(\mathbf{x}) \neq 0$, but an inequality constraint is violated only when $g_i(\mathbf{x}) > 0$. To make $d(\mathbf{x}) = 0$ when all constraints are met, (2.21) uses $(\max [0, g_i(\mathbf{x})])$. The difficulty of the penalty function is choosing the appropriate penalty factors. Penalty factors are highly problem-independent and need careful tuning for each problem.

The special representations and operator methods intend to map the feasible region onto a space with a different shape so that the problem is easier to be solved [27]. The space after-mapped is also called decoded space. The principle of mapping is each feasible solution in the search space must be included in the decoded space, and vice versa. Any small change in the search space leads to a small change in the decoded space as well. The homomorphous maps (HM) approach is proposed by Koziel and Michalewicz in 1998 [23], which mapped the feasible region onto a n-dimensional cube. However, HM are sometimes too complex to implement, and usually need a large number of objective function evaluations. Therefore, the special representations and operator approaches are rarely used [27].

Repair algorithms transform an infeasible solution into a feasible one, in other words, repair the infeasible candidate. Michalewicz proposed the GENOCOP in 1991 [29], but it was only used for solving linear constraints. Later, the GENOCOP III method [30] was introduced, which uses GENOCOP [29] to deal with one of its two populations. The first population is used to store candidate solutions in GENOCOP that only satisfy the linear constraints. A special operator is designed to convert these solutions into fully feasible solutions. The second population stores these feasible points. This process repairs only the points that satisfy the linear constraints to the points that satisfy all constraints so that they can be evaluated by the objective function.

The idea of separation of constraints and objective is opposite to the penalty function techniques. Hinterding and Michalewicz [20] introduced the idea of separating the optimization problem into two phases: the first one focuses on generating feasible solutions but ignores their objective function values. The second phase optimizes the objective function among at these candidate solutions. A disadvantage of this kind of

approach is that an algorithm may be biased because it concentrates on searching for the smaller constraint function values first, but ignoring the objective function value [27].

The stochastic ranking was originally proposed by Runarsson and Yao in 2000 [35]. It is designed to deal with the shortcoming of penalty function approaches, especially for those suffering from poorly chosen penalty factors. It introduces a user-defined parameter to decide how to compare the infeasible candidate points, either based on the objective function value of these points or the sum of constraint violations of them.

2.4.2 Current Constraint-Handling Approaches

Beyond the constraint-handling methods mentioned above, there are a set of recent constraint-handling techniques which are used more often in current research. Arnold and Porter [4] proposed the adaptive augmented Lagrangian constraint handling technique which converts a constrained problem to an unconstrained problem by adapting the augmented Lagrange function. This method utilizes a strategy for adapting the penalty parameter and Lagrange factor, which was first proposed for (1+1)-ES and extended to CMA-ES with a single constraint problem by Atamna, Auger and Hansen [5]. The taxonomy of the adaptive augmented Lagrangian constraint handling technique is QRSK, and it exhibits affine invariance. Other strategies reduce the dimension of the search space by forcing active inequality constraints to be equalities; such as active-set ES [2, 3]. The constraint taxonomy of active-set ES is Q*AK, it means Q, R/U, A, K. Linear constraint covariance matrix self-adaptation ES method [37] applies on a variant of CMA-ES, and it focuses on explicitly and linearly constrained problems. This algorithm solves the same type of constraints as active-set ES. In 2019, Sakamoto and Akimoto [36] proposed an adaptive ranking based constraint handling (ARCH) for CMA-ES with explicitly constrained optimization, it can also solve Q*AK constraints. Moreover, ARCH has both invariance properties.

The first invariance property is invariance with respect to strictly increasing transformations of the objective and constraint functions. It indicates the performance of the algorithm is not influenced by the increasing transformation of the objective and constraint functions. The second invariance property is the invariance to an affine

transformation of the search space. An affine transformation means that, a linear transformation is performed on a vector space, followed by a translation. Invariance to an affine transformation of the search space means the algorithm has the same ability to solve constrained problems on the original coordinate system as well as the affine transformed coordinate system. An algorithm with such an invariance can transform the shape of search space that is difficult to solve into a shape that is easier to solve [36].

In ARCH, the algorithm first repairs the infeasible points before they are evaluated. Then a total ranking is used in ARCH as the fitness value of candidate solutions instead of their function values. The total ranking is calculated by the weighted sum of candidate solutions' ranking based on their objective function value plus the constraint violations which are measured by the Mahalanobis distance with a ranking coefficient. The Mahalanobis distance is the distance between an infeasible point and the intersection of the violated constraints but is independent to the coordinate choices. Using the Mahalanobis distance allows the algorithm to obtain the invariant affine properties of the search space. The total ranking helps the algorithm maintain the invariance to element-wise increasing transformations of the objective and constraint functions. An adaptation of the ranking coefficient is employed during the calculatation of the total ranking, which is inspired by the weighted recombination in ES [1]. Overall, ARCH is a useful algorithm because of its invariance properties. Furthermore, the repair operator used when dealing with the infeasible points is very similar to the projection that was used to repair the infeasible point in active-set ES. We will discuss active-set ES in detail in chapter 3.

2.4.3 Differential Evolution

In this section, we mainly focus on differential evolution, or DE, algorithms because one of our comparison algorithms, LSHADE44, is an adaptive version of a DE algorithm.

Differential evolution was first introduced by Storn and Price in 1996 [39, 38]. It works by maintaining a population P of N individuals within the search space S. New trial points are generated by mutation, crossover and selection.

DE and ES have the similarities and differences. They both produce new points

from past iterations and do selection for the next generation. The differences are that they carry out different mutation and recombination/crossover strategies for individuals. In DE, each vector $\mathbf{x}_s^{(k)}$ is in the population P, where $s \in [1, 2, ..., N]$, and k indicates the generation number. In each iteration, mutation is conducted by adding the scaled differences between two existing vectors $\mathbf{x}_2^{(k)}$ and $\mathbf{x}_3^{(k)}$ to another distinct vector $\mathbf{x}_1^{(k)}$. That is $\mathbf{y}^{(k+1)} = \mathbf{x}_1^{(k)} + F(\mathbf{x}_2^{(k)} - \mathbf{x}_3^{(k)})$ where F is called the mutation factor which is a constant in $[0, 2]$, and $\mathbf{y}$ is called the trial vector. Whether crossover occurs depends on a probability CR in from $[0, 1]$ which is called the crossover probability. For each dimension $i \in [1, ..., n]$ in $\mathbf{x}_s$ where $s \in [1, 2, ..., N]$. If a random number in $[0, 1]$ is less than the value of CR, $y_{i,s}^{(k+1)} = x_{1,s}^{(k)} + F(x_{2,s}^{(k)} - x_{3,s}^{(k)})$; otherwise, $y_{i,s}^{(k+1)} = x_{i,s}^{(k)}$. The function value of $\mathbf{y}_s^{(k+1)}$ and $\mathbf{x}_s^{(k)}$ is evaluated and the vector with the lower function value will replace the other in the population. When all the individuals in the population have undergone mutation, crossover and selection, the algorithm will enter the next iteration with the updated population until the termination criterion is met.

DE can be modified for constrained optimization problems, Takahama and Sakai first proposed ϵDE in 2005 [40]. These DE are denoted by incorporating the ϵ method. This method defines an order relation, the ϵ level comparison, between the objective function value $f(\mathbf{x})$ and the constraint violation $g(\mathbf{x})$, and optimizes the constraint violation and the objective function separately. The constraint violation $g(\mathbf{x})$ indicates how much a point $\mathbf{x}$ violates the constraints. If $\mathbf{x}$ is in the feasible region, $g(\mathbf{x}) = 0$; otherwise, $g(\mathbf{x}) > 0$. Let f_1, f_2 and g_1, g_2 denote the function values and constraint violations at the point $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. For any $\epsilon \geq 0$, the ϵ level comparisons $<_\epsilon$ and $\leq_\epsilon$ between (f_1, g_1) and (f_2, g_2) are defined as:

$$(f_1, g_1) <_\epsilon (f_2, g_2) \Leftrightarrow \begin{cases} f_1 < f_2, \text{ if } g_1, g_2 \leq \epsilon \\ f_1 < f_2, \text{ if } g_1 = g_2 \\ g_1 < g_2, \text{ otherwise} \end{cases} \tag{2.22}$$

$$(f_1, g_1) \leq_\epsilon (f_2, g_2) \Leftrightarrow \begin{cases} f_1 \leq f_2, \text{ if } g_1, g_2 \leq \epsilon \\ f_1 \leq f_2, \text{ if } g_1 = g_2 \\ g_1 \leq g_2, \text{ otherwise} \end{cases} \tag{2.23}$$

In summary, ϵDE no longer compares the parent and child by their objective function value, but uses the ϵ level comparisons above. Applying the ϵ level comparisons can

obtain constraints violations and choose points that violate the constraints less often.

These ϵDE were improved by introducing a gradient-based mutation and elitism in 2006 [40]. The gradient-based mutation finds a feasible point by using the gradient of constraints at the infeasible point. Elitism preserves feasible points with less constraint violations as feasible elites. The improved ϵDE searches for vectors within both the original population and the elite population.

Another adaptive version of the DE algorithm is LSHADE44. It is initially used for solving bound-constrained optimization, but Poláková et al. modified it for solving constrained problems in 2016 [33]. It combines the L-SHADE algorithm with two mutation strategies and two crossover strategies to make the algorithm more efficient. In each iteration, LSHADE44 maintains a population set $P = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$. A trial point is generated from a point in P through mutation and crossover. The choice of strategies depends on the previous count of successes. The higher the number of successes, the more likely this strategy will be selected. Once a trial point is chosen, LSHADE44 evaluates the function value and constraint violation of the trial point. The algorithm tends to leave the feasible points with small function value or the infeasible point with small constraint violation in the population. After all points in P are processed as a parent, an updated population is generated. Finally, LSHADE44 applies a linearly decreasing population size mechanism. As the number of objective function evaluations increase, the population size decreases linearly generation by generation. This is because reducing the population size has been shown to improve the performance of the EA algorithm [9, 25]. These steps are repeated until the termination condition is met.

2.5 Test Functions

The test functions are chosen from IEEE Congress on Evolutionary Computation (CEC) competitions. They are the benchmark for the evolutionary algorithm since they provide the environment of constrained tests and are used for comparing and evaluating algorithms. CEC competitions proposed a benchmark for single objective optimization with constraints in CEC2006 and CEC2010.

To solve a constrained problem, it is unclear what characteristics make it difficult

to be solved by evolutionary algorithms. Any constrained problem may contain multiple parameters, such as the number of linear or nonlinear constraints, the number of equality or inequality constraints, and the type of the objective function. CEC2006 special session benchmark presented 24 distinct constrained functions with varying search space dimension between $n = 2$ and $n = 24$, as well as different types and numbers of constraints and different types of the objection functions. The constraints include linear equality and inequality constraints, and nonlinear equality and inequality constraints. The objective function types are linear, quadratic, cubic, polynomial, and nonlinear. The number of active constraints in the optimal active-set ranges from 0 to 24 [26].

CEC2010 special session benchmark defines 18 new constrained problems and have only one overlap functions with CEC2006 [41]. It focuses on eight distinct objective functions with their variations, including parameter translations and/or rotations. Some of the objective and constraint functions are collected from unconstrained problems, such as the Rosenbrock function, Schwefel's function, and other functions which can be found in [28] by the test-case generator. The dimension of the search space in CEC2010 benchmark is also scalable. Problems are demanded to be solved in dimension $n = 10$ and $n = 30$. Later, CEC2017 competition increased the dimension of the search space to $n = 50$, and $n = 100$ with 28 new benchmark problems.

In the problem definitions and evaluation criteria presented in CEC 2017, it was claimed that CEC2006 have been solved successfully [42]. However, the problems were solved with a large number of function evaluations which is not ideal. For example, in the process of solving the problems in CEC2006 with the ϵDE algorithm, the algorithm had the least function evaluation to solve the eighth problem among the 24 problems, which was 1,139. We will see that applying the interior-point method in Matlab to the constrained nonlinear optimization toolbox on the same problem, produces the median value of function evaluation as 52, which is 22 times faster than the result of ϵDE. There is clearly have room for improvement, so we continue to study the problems in CEC2006 special session benchmark.

Chapter 3

Method

As we have seen, the active-set method is applied to constrained optimization problems with both equality constraints and inequality constraints. We also saw that evolution strategies (ES) are algorithms for stochastic black-box optimization, which are widely used to solve unconstrained optimization problems. Arnold proposed to combine the active-set method with (1+1)-ES for optimization with Q*AK constraints at the Parallel Problem Solving from Nature (PPSN) conference [2] and then improved it in Genetic and Evolutionary Computation Conference (GECCO) [3]. He evaluated these two algorithms using the first 11 test functions in the Congress on Evolution Computing (CEC) competition test set from 2006. In our method, we revise his algorithms, adjust the projection setting and the decision to release constraints. We then examine the performances of our modified active-set ES algorithm at PPSN conference and at GECCO conference on spherically symmetric functions with mutually orthogonal linear constraints.

3.1 Related Work

In unconstrained (1+1)-ES, a candidate solution is generated by sampling a multivariate normal distribution centred on the best solution from past iterations. If the new candidate solution's function value is better than the previous values, the algorithm replaces the parent by the new candidate solution; otherwise, it keeps the parent. In the active-set method, it maintains a set of active inequality constraints and searches the result within the subspace determined by the active-set. Working set $\mathcal{W}$ stores the active constraints. If an inequality constraint is violated at the successful candidate solution, it becomes tight and will be added to the working set $\mathcal{W}$. Algorithm 5 describes the procedure of the active-set (1+1)-ES which was proposed by Arnold at PPSN conference in 2016 [2].

We recall the minimization problem as described in (2.1) on page 8, active-set

Algorithm 5 Active-set Evolution Strategy in PPSN Conference [2]

Initialize $\sigma \in \mathbb{R}^+, \mathbf{x} \in \mathbb{R}^n$
while the termination criterion is not met **do**
    Compute the dimension of the reduced search space at $\mathbf{x}$ by $n' = n - \text{rank}(N)$ where N is the matrix whose columns are normal vectors of constraints in the active-set.
    repeat
        $\mathbf{y} \leftarrow \mathbf{x} + \sigma \times \mathcal{N}(\mathbf{0}, \mathbf{I})$.
        If $n' = 0$, let κ be true. If $n' > 0$, let κ be true with probability p and false otherwise.
        If κ is true, project $\mathbf{y}$ onto the feasible region; otherwise, project $\mathbf{y}$ onto the intersection of the feasible region with the reduced search space at $\mathbf{x}$.
    until $\mathbf{y}$ is feasible
    if $f(\mathbf{y}) < f(\mathbf{x})$ **then**
        $\mathbf{x} \leftarrow \mathbf{y}$,
        if κ is false **then**
            $\sigma \leftarrow \sigma 2^{1/n'}$,
        end if
    else
        if κ is false **then**
            $\sigma \leftarrow \sigma 2^{-1/(4n')}$.
        end if
    end if
end while

ES generates new candidate solutions in a similar way as the (1+1)-ES method, and uses the 1/5th rule to adapt the step-size. The candidate solution is usually projected onto the reduced search space determined by active-set to avoid unnecessary step-size reduction. The search space is reduced because the search space dimension will be decreased when an inequality constraint is tight and is added to the working set. The point projection is accomplished by minimizing the function $d(\mathbf{w}) = ||\mathbf{w} - \mathbf{y}||^2$ subject

to the constraints

$$\begin{aligned} g_i(\mathbf{w}) &\leq 0 \quad \text{for } i \in \{1, ..., l\} \setminus \mathcal{W} \\ g_i(\mathbf{w}) &= 0 \quad \text{for } i \in \mathcal{W} \\ h_j(\mathbf{w}) &= 0 \quad \text{for } j \in \{1, ..., m\}, \end{aligned} \tag{3.1}$$

where $\mathbf{y}$ are the candidate solutions, g_i indicates the inequality constraints and h_j are the equality constraints. This minimization process aims to project the current candidate solution onto the intersection between the reduced search space determined by $\mathcal{W}$ and the feasible region. This does not require the algorithm to perform more than a single evaluation of the objective function per iteration because the objective function f is not used here. The minimization can be done by any constrained optimization algorithm, for instance the *fmincon* function in the Matlab optimization toolbox.

As more inequality constraints are added to $\mathcal{W}$, the algorithm needs a mechanism to release constraints from the working set. Arnold [2] introduced a Boolean flag κ to determine whether or not to use $\mathcal{W}$. If κ is true, the algorithm considers releasing all inequality constraints in $\mathcal{W}$, or in other words, suspending $\mathcal{W}$. Releasing a constraint indicates turning the active inequality constraints back to inequalities. On the other hand, if κ is false, then the algorithm keeps searching within the reduced search space. The Boolean flag κ is set to true either if the dimension of the reduced search space is zero, or a random number in $[0, 1]$ falls below a probability p.

The above method performs inadequately under certain circumstances. For instance, if there is a constraint that is relatively close to the optimal solution but it is not active at the optimal solution, then Algorithm 5 will have difficulty solving the problem. For example: consider the objective function subject to the linear inequality constraints [3]:

$$\begin{aligned} & f(\mathbf{x}) = \mathbf{x}^\intercal \mathbf{x} \\ \text{subject to} \quad & g_1(\mathbf{x}) = 100\, \mathbf{e}_1^\intercal \mathbf{x} - 1 \leq 0 \\ & g_i(\mathbf{x}) = \mathbf{e}_i^\intercal \mathbf{x} + 1 \leq 0 \qquad \text{for } i = 2, ..., l \end{aligned} \tag{3.2}$$

where $\mathbf{e}_i$ is the ith coordinate axis direction written as the unit column vector. The

optimal solution to this problem is

$$\mathbf{x}^* = (0, \underbrace{-1, ..., -1}_{l-1}, \underbrace{0, ..., 0}_{n-l})^\intercal \tag{3.3}$$

and the optimal active-set is

$$\mathcal{A}^* = (0, \underbrace{1, ..., 1}_{l-1}, \underbrace{0, ..., 0}_{n-l})^\intercal \tag{3.4}$$

where $l \leq n$. It can be seen that the optimal $\mathcal{A}^*$ is formed by all inequality constraints except the first one. However, the first constraint is much closer to the unconstrained optimum compare to other constraints due to the relatively large function coefficient, which makes it is easy to be added into $\mathcal{W}$.

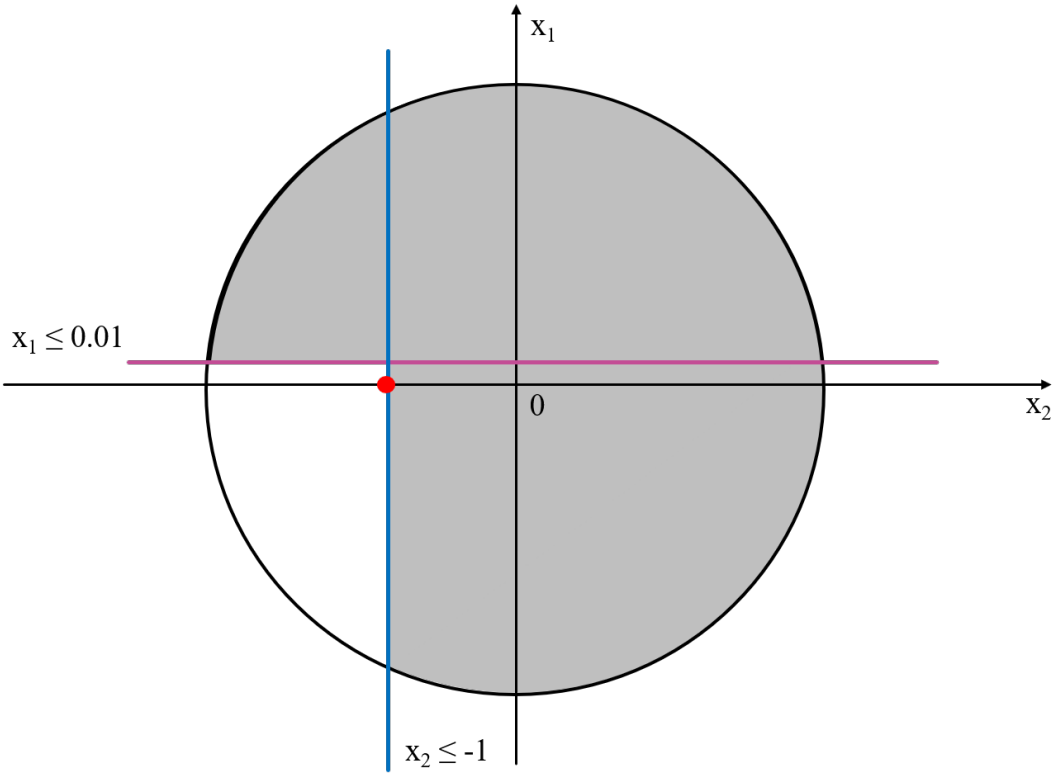


Figure 3.1: This figure shows a special case of equation (3.2). The objective function is a sphere function with the first two constraints. The shaded areas and the areas where $g_1 > 0$ and $g_2 > 0$ are the infeasible regions. The red spot is the optimal solution under these two constraints.

Figure 3.1 visualizes a special case of (3.2) with the first two constraints. The red spot indicates the global optimum. During the search process, if the first constraint

g_1 is added into $\mathcal{W}$ but the second constraint g_2 is not, then constraint g_1 will become tight and the algorithm will search for the optimal solution on the intersection of $x_1 = 0.01$ and $x_2 \leq -1$. It is impossible to find the red spot on this intersection because the optimal solution is not on the plane determined by g_1. A large number of iterations may be needed to improve the solution in a subspace that does not contain the optimal solution. At this point, the algorithm needs to release g_1 from the active-set. In order to release g_1, the candidate solution needs to be satisfied before the projection. However, the algorithm in PPSN conference can only suspend the whole active-set, it lets those constraints that should be active become inactive constraints, and algorithm will generate to inferior points. Therefore, the algorithm takes longer to locate constraint g_1 among so many constraints, and suspending the whole active-set will not help in this situation. The algorithm needs a mechanism to determine which individual constraints need to be released, and keep the other $l - 1$ constraints as active.

At the Genetic and Evolutionary Computation Conference (GECCO) in 2017 [3], Arnold came up with releasing the individual constraint in the active-set periodically. Rather than suspending the whole active-set, he added the steps of considering each active inequality constraint are determined which to release in random order after Line 3 in Algorithm 5. The rest steps of the revised algorithm remain the same. Algorithm 6 represents the added parts of the active-set ES from the GECCO conference.

A σ threshold is introduced by Arnold as another condition to release the active inequality constraints [3]. When the dimension of the reduced search space equals to zero, the algorithm reduces the step-size to $1/2$ of the previous step-size so that the algorithm can release the first constraint on the above problem (3.2) when $l = n$. If the step-size is too large, the algorithm may repeatedly go beyond the optimal solution in the first dimension and fail to release it. The algorithm then considers releasing every active inequality constraint in a random order from the working set and forms a temporary active-set. A point is generated from a normal distribution and is projected onto the intersection of the feasible region and the reduced search space determined by the temporary working set. This step is repeated until a feasible point that satisfies the constraints in the temporary working set is found and the constraint k is not tight at $\mathbf{y}$. If the objective function value of this candidate solution is better than the best

Algorithm 6 Revised releasing policy in GECCO conference [3]

if $n' = 0$ or $\sigma < \sigma_{thresh}$ **then**
  if $n' = 0$ **then**
    $\sigma \leftarrow \sigma/2$.
  end if
  for all $k \in \mathcal{W}$ **do**
    repeat
      $\mathbf{y} \leftarrow \mathbf{x} + \sigma \times \mathcal{N}(\mathbf{0}, \mathbf{I})$.
      Project $\mathbf{y}$ onto the intersection of the feasible region with $\mathcal{W} \setminus \{k\}$,
    until $\mathbf{y}$ is feasible and constraint k is not tight at $\mathbf{y}$.
    if $f(\mathbf{y}) < f(\mathbf{x})$ **then**
      $\mathbf{x} \leftarrow \mathbf{y}$
      $\mathcal{W} \leftarrow \mathcal{W} \setminus \{k\}$
    end if
  end for
  $\sigma_{thresh} \leftarrow \sigma_{thresh}/10$,
  $n' = n - \text{rank}(N)$
end if

solution found so far, it replaces the best solution and updates the working set to the temporary one. Otherwise, the algorithm abandons the temporary working set. In this way, all active inequality constraints are considered to be released and the dimension of reduced search space is recomputed. At the end of these releasing steps, the σ threshold is reduced to one-tenth of the previous threshold.

The test functions Arnold used are gathered by Michalewicz and Schoenauer [28] to evaluate both algorithms [2, 3]. The test set consists of 11 benchmark functions from $g01$ to $g11$ and Liang et al. expanded this 24 problems [26]. When testing the revised algorithm on problems $g12$ to $g24$, we found there are infinite loops that happen in the function *fmincon* during the projection of $g21$ and $g22$. Therefore, we consider to revise Algorithm 5 and 6 and systematically compare the performances of them.

3.2 Modified Active-set Evolution Strategies

In our modified active-set ES, we used the SQP method instead of the active-set method in *fmincon* to avoid the infinite loops. In addition, we combined the releasing policy of the active-set ES from the PPSN conference and the GECCO conference and increased the adapt factor of the step-size so that the modified active-set ES can be improved. Algorithm 7 describes the details of the modified active-set ES.

Modified active-set ES has the following adjustments compared to the previous algorithm:

- The releasing process is described in Lines 5-7. Rather than suspending the entire active-set as in Algorithm 5, we consider releasing only one constraint when κ is true. The settings of κ are the same as in Algorithm 5. Compared with Algorithm 6, we release the constraint at most once in each iteration instead of considering whether to release every constraint in the active-set, which makes the algorithm simpler. To decide which constraint the algorithm will release, we set a counter for each constraint to count the iteration number when they have been released. When κ is true, we choose the constraint that has not been released for the longest, which is the one with the smallest number in counter, to avoid always releasing a certain constraint. We also abandon σ_{thresh} because we observed it is too late as a condition for releasing constraints, the algorithm needs to consider releasing constraints before σ decreases to σ_{thresh}. In addition, reducing σ when $n' = 0$ makes it difficult for the algorithm to find a better candidate solution, and thus trapped at the local minimum. Therefore, we consider releasing the constraint if either $n' = 0$ or κ are true, but we do not modify σ here.

- The step-size updating is also different from Algorithm 5 and 6. The modified active-set ES employs the 1/5th rule [22] with larger factors compared with the algorithms from the PPSN conference and the GECCO conference. Regardless of whether we are releasing constraints, as long as the generated offspring is better than the parent, the algorithm will adjust the step-size. These operations of step-size updating can cause the step-size to adapt faster, thereby speeding up the process of the algorithm.

Algorithm 7 Modified Active-set Evolution Strategy

Initialize $\sigma \in \mathbb{R}^+, \mathbf{x} \in \mathbb{R}^n$
while the termination criterion is not met **do**
    Compute the dimension of the reduced search space at $\mathbf{x}$ by $n' = n - \text{rank}(N)$ where N is the matrix whose columns are normal vector of constraints in $\mathcal{W}$.
    If $n' = 0$, let κ be true. If $n' > 0$, let κ be true with probability p and false otherwise.
    if κ is True **then**
        Release the inequality constraint that has not been released for the longest, constraint m, $\mathcal{W} \leftarrow \mathcal{W} \backslash \{m\}$
    end if
    repeat
        $\mathbf{y} \leftarrow \mathbf{x} + \sigma \times \mathcal{N}(\mathbf{0}, \mathbf{I})$,
        project $\mathbf{y}$ onto the reduced search space that $\mathcal{W}$ determined,
    until $\mathbf{y}$ is feasible and constraint k is not tight at $\mathbf{y}$.
    if $f(\mathbf{y}) < f(\mathbf{x})$ **then**
        $\mathbf{x} \leftarrow \mathbf{y}$,
        $\sigma \leftarrow \sigma \exp^{0.8/\sqrt{(1+n')}}$,
    else
        $\sigma \leftarrow \sigma \exp^{-0.2/\sqrt{(1+n')}}$.
        if κ is True **then**
            $\mathcal{W} \leftarrow \mathcal{W} \cup \{m\}$,
        end if
    end if
end while

- During the process of point projection, the active-set method is used in the *fmincon* function of the Matlab optimization toolbox. We observe that fmincon has infinite loops when solving problems $g21$ and $g22$. We notice both of there methods have constraints with non-integer powers and with natural logarithms. When the base of the natural logarithm or the base of non-integer power is

negative, the value of this constraint will have complex value which makes the active-set method unable to continue to execute. One difference between the active-set method and the SQP method is their robustness to non-double results. The SQP method allows functions to return infinite values, values that are neither real or complex numbers, or complex numbers in the iteration; however, the active-set method does not allow there values. Therefore, using the SQP method when there is a complex value, for example, in problems $g21$ and $g22$, rather than the active-set method will avoid infinite loops in these problems.

3.3 Performance of Active-set Evolution Strategy

In order to systematically evaluate the active-set evolution strategies from the PPSN conference [2], the GECCO conference [3] and our modified version, we generate spherically symmetric functions with mutually orthogonal linear constraints, similar to the problem described in (3.2). These problems of the form such that:

$$\begin{aligned} & f(\mathbf{x}) = \mathbf{x}^\intercal \mathbf{x} \\ \text{subject to} \quad & g_i(\mathbf{x}) = 100\,\mathbf{e}_i^\intercal \mathbf{x} - 1 \leq 0 \qquad \text{for } i = 1, ..., m \\ & g_i(\mathbf{x}) = \mathbf{e}_i^\intercal \mathbf{x} + 1 \leq 0 \qquad \text{for } i = m+1, ..., l \end{aligned} \tag{3.5}$$

The dimension of the problem is $n \in \{10, 20\}$, and the constraint numbers are six and twelve in total, respectively. There are two experiments for each dimension of the problem, which are $m = 1$ and $m = l/2$. Therefore, we can observe the algorithm's ability of dealing with the problems with different dimensions and different numbers of active constraints. We use the same step-size adaptation for all three algorithms (see Algorithm 7 Line 14 and Line 16), and run the algorithms 500 times each. The algorithms terminate when an $\mathbf{x}$ which optimal solution satisfies $f(\mathbf{x}) < f^* + |f^* \times 10^{-8}|$ is found, given $\epsilon = 10^{-8}$. We consider 8,000 iterations here because we want all algorithms to be as successful as possible and not be affected by the limit of maximum iterations. A median number of function evaluations are recorded in Table 3.1 below.

It can be seen from the Table 3.1 that the modified active-set ES has the smallest median number of evaluations among all test function with all algorithms. The advantage of active-set ES is not particularly obvious in low-dimensional problems, but

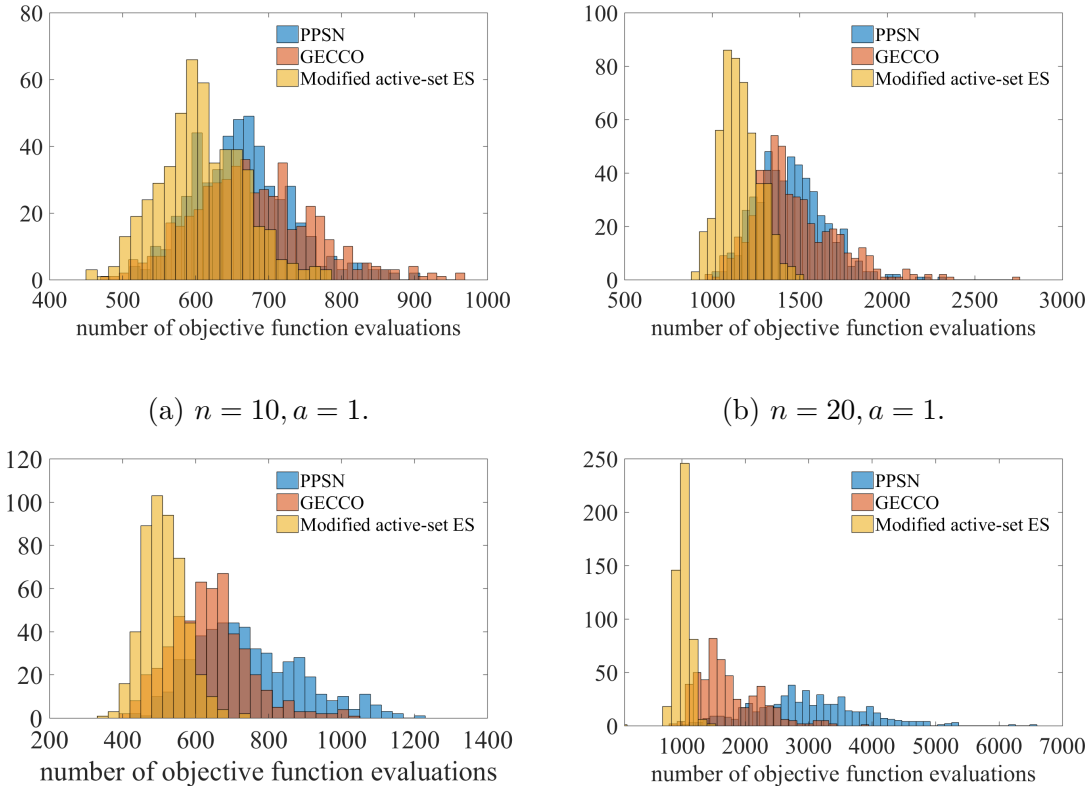
	Total constraints	m	PPSN [2]	GECCO [3]	Modified active-set ES
$n = 10$	6	1	661	671	603
$n = 10$	6	3	723	633	510
$n = 20$	12	1	1442	1410	1142
$n = 20$	12	6	2958	1618	1123

Table 3.1: Median number of objective function evaluations required of three active-set ES for solving sphere functions with linear constraints. n indicates the dimension of variables, and m means the number of active constraints in the optimal active-set.

it is more obvious in high-dimensional and more active constraint in optimal active-set. Look at the last row of this table, the median number of objective function evaluations required of the modified algorithm is only one-third of the algorithm in the PPSN conference.

Figure 3.2 shows the histograms of the number of objective function evaluations required to solve all problems. It can be seen that the distribution of runs in the first figure is very concentrated. As the number of active constraints gradually increases (see Figure 3.2(c) and (d)), the superiority of the performance of the modified algorithm is reflected. In Figure 3.2(d), the running distribution of algorithms from the PPSN conference and the GECCO conference are very decentralized compared with the modified algorithm. Moreover, the median number of objective function evaluations required of the modified algorithm is only one-third of the algorithm in the PPSN conference.

To see the differences between three algorithms in these two experiments clearly, we plot the traces of median number of objective function evaluations required which $n = 10, a = 3$ and $n = 20, a = 6$ in Figure 3.3. The plots in Figure 3.3(a) are the traces of three algorithms on the sphere function with six constraints when $n = 10$; half of the constraints are active at the optimal solution and half of them are not. The magenta triangle indicates the iteration where the optimal active-set has been found, and the black crosses show the iterations where the algorithm releases one or more constraints. It can be seen that the algorithm from the PPSN conference stagnates between iteration 424 and 605, and the step-size decreases at the same time until iteration 608 and then increases. The step-size decreases first because the algorithm has not get released the constraint at the beginning, and therefore no better candidate



(a) $n = 10, a = 1$.

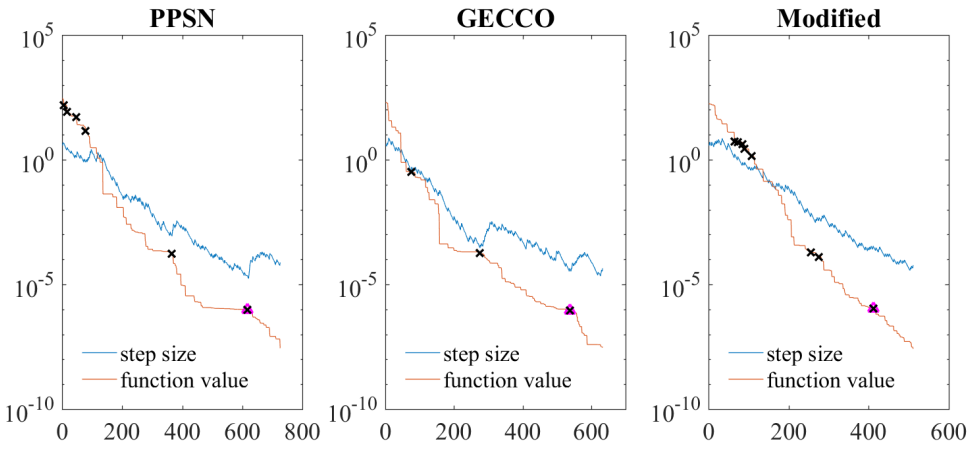
(b) $n = 20, a = 1$.

(c) $n = 10, a = 3$.

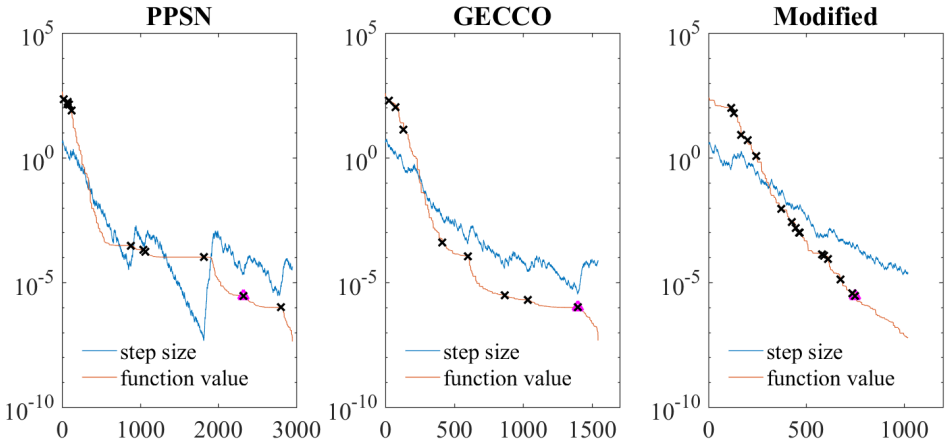
(d) $n = 20, a = 6$.

Figure 3.2: The histograms of numbers of the objective function evaluation for sphere function with constraints. The dimensions for (a) and (c) are both 10 and have six constraints; for (b) and (d) the dimensions are both $n = 20$ and have 12 constraints. a indicates the number of active constraints in the optimal active-set. There is only one active constraint at the optimal point in case (a) and (b), while for (c) and (d), half of the constraints are active at the optimal solution but the rest of the constraints are not.

solution can be found and the step size continues to decrease until it is small enough to locate the constraint that needs to be released. At 608 iterations, the constraints are released, and the optimal active-set is found. Thus the candidate solution can be further reduced, and therefore the step size is increased. The algorithm from the GECCO conference avoids the stagnation by releasing individual constraints three times and finding the optimal active-set at iteration 537. However, we notice that the step-size decreases quickly from the beginning until iteration 274 compared with the modified active-set ES, but the corresponding function value does not change a lot between iteration 200 and 274. The reason is that one of the conditions for releasing



(a) Sphere function with six constraints when $n = 10$.



(b) Sphere function with 12 constraints when $n = 20$.

Figure 3.3: From left to right are the the median number of objective function evaluations required traces of active-set ES in PPSN conference, GECCO conference and our modified version in the sphere function with different dimensions. In both problems, half of the constraints are active at the optimal solution, and the remaining half are not active. The magenta triangle indicates the iteration where active-set is optimal. The black crossing means the iterations where algorithm releases the constraint(s).

the constraint from the GECCO conference algorithm is when σ drops below σ_{thresh}. From the trace, we can observe that the algorithm wastes too many iterations to reduce σ so that it will satisfy $\sigma < \sigma_{thresh}$ between iteration 200 and 274, thus preventing the algorithm from releasing the constraints timely. In addition, too small a step-size will slow down the progress of the optimization process, which is improved in the modified algorithm. The modified active-set ES discards the setting of σ_{thresh} and allows the algorithm to consider when to release the constraints more frequently,

thereby improving the process rate of the algorithm.

The advantage of the modified active-set ES is more obvious when dealing with the problem of $n = 20$, see Figure (3.3(b)). From the traces of the algorithm from the PPSN conference, we notice that the function value starts to be stagnated from iterations 1075 to 1918, and the step-size decreases to 10^{-7} during these iterations until the constraints are released and the search can proceed. The situation improved when the algorithm in GECCO conference is applied to this problem, the function value no longer has a long stagnation, and the step-size does not need to be reduced to a very small value. Compared to the traces of the algorithm from the GECCO conference with the modified active-set ES, we find that the function value decreases slowly between iterations 418 to 593 and between iterations 1034 to 1395 in the algorithm from the GECCO conference. We find that the reason that the step-size has been decreasing is the step-size threshold is set too small which results in the conditions for releasing the constraints cannot be met and triggered. In order to satisfy the conditions of releasing the constraints, the step-size is even reduced further to below 10^{-5}. Compared with the modified active-set ES, the step-size has not been reduced to 10^{-5} for the whole process. Therefore, discarding σ_{thresh} as a condition for releasing constraints is very helpful to improve the performance of the algorithm.

Overall, we can say that the modified active-set ES performs the best in this experiment, followed by the algorithm from the GECCO conference [3]. The algorithm from the PPSN conference [2] performs the worst. The modified active-set ES has a greater advantage when the dimension of the problem and the number of active constraints become larger. In next chapter, we will compare the performance of modified active-set ES with other optimization algorithms.

Chapter 4

Experiments

To test the performance of our modified active-set evolution strategy, we compare it with the four other algorithms; three of which are deterministic algorithms from the optimization toolbox in Matlab, two active set methods and one interior-point method; and the forth one is the differential evolution, LSHADE44. We use these five algorithms to solve $g01$ to $g24$ gathered by Liang et al [26], and measure their performance by their success rate; the median number of objective function evaluations required, and empirical cumulative running time distribution.

4.1 Set-up

In the experiment of testing deterministic algorithms, we use the active-set, sequential quadratic programming, and interior-point method in the *fmincon* function in the Matlab optimization toolbox. The inputs of the *fmincon* function are the objective function, an initial point, lower bounds and upper bounds of the variables, linear equality and inequality constraints, nonlinear inequality and equality constraints, and some specified optimization options which will be introduced below. We initialize the starting point by sampling a point within the bound-constrained search space uniformly and randomly. For the optimization options, we choose the constraint tolerance as 10^{-9}, the constraint tolerance indicates the acceptable degree of constraint violation. In other words, the maximum value that a point can violate the constraint is 10^{-9}. The target accuracy ϵ is 10^{-8}, and the algorithm terminates either its objective function value $f(\mathbf{x}) < f^* + |f^* \times 10^{-8}|$, where f^* is the optimal function value, or the maximum function evaluation is reached but no optimal value has been found. All settings of the maximum function evaluation number are determined by the number of evaluations where the algorithm continues to run but cannot find the best value. We set the maximum function evaluation number as 4,000 for deterministic algorithms.

For the active-set evolution strategy, we initialize the starting point as we do in the deterministic algorithms, and then project it onto the feasible region to ensure its feasibility. The initial step-size is set as $0.2 \cdot \min\{u_i - l_i | i = 1, ..., n\}$, where u_i and l_i are the upper bounds and lower bounds of the search space in dimension i, respectively. The termination condition of this algorithm is either there is a f^* satisfies $f(\mathbf{x}) < f^* + |f^* \times 10^{-8}|$, or the maximum iteration limit is reached but the algorithm has not found a function value that meets $f(\mathbf{x}) < f^* + |f^* \times 10^{-8}|$. ϵ is set as 10^{-8} and the maximum iteration is set as 2,000. The constraint tolerance in *fmincon* is set as 10^{-9}.

The inputs of the LSHADE44 algorithm are the upper bounds and lower bounds of the variables, maximum function evaluations, target accuracy, and the constraints information. The initial point is sampled like previous methods. The maximum number of function iterations we set is 20,000, and the target accuracy is 10^{-8}. The maximum function iterations limit is quite large in this algorithm because the maximum function evaluation is $20,000 \times n$ for each test function, where n indicates the function dimension [33].

4.2 Test Function

The test functions we used are 24 test problems gathered by Liang et al [26] from $g01$ to $g24$ with variety of dimensions. Table 4.1 lists the dimensions of the objective functions, as well as the types of objective functions and constraint functions. It can be seen in Appendix that this test function set includes linear, nonlinear, quadratic, polynomial, and cubic functions and the constraint functions includes linear, quadratic, cubic, polynomial and nonlinear functions. The dimensions of the test functions range from 2 to 24. The diversity of test functions combines with different kinds of constraint functions which allows us to test the performance of all algorithms in various aspect, for example, the ability of algorithms to solve problems of different dimensions.

Problem	n	Type of objective function	Type of constraint functions
g01	13	quadratic	linear
g02	20	nonlinear	polynomial
g03	10	polynomial	quadratic
g04	5	quadratic	quadratic
g05	4	cubic	nonlinear
g06	2	cubic	quadratic
g07	10	quadratic	quadratic
g08	2	nonlinear	quadratic
g09	7	polynomial	polynomial
g10	8	linear	quadratic
g11	2	quadratic	quadratic
g12	3	quadratic	quadratic
g13	5	nonlinear	cubic
g14	10	nonlinear	linear
g15	3	quadratic	quadratic
g16	5	nonlinear	quadratic
g17	6	nonlinear	nonlinear
g18	9	quadratic	quadratic
g19	15	nonlinear	quadratic
g20	24	linear	nonlinear
g21	7	linear	nonlinear
g22	22	linear	nonlinear
g23	9	linear	quadratic
g24	2	linear	polynomial

Table 4.1: Table of details of 24 test functions [26]. n indicates the dimension of variables.

4.3 Success Rate and Median Function Evaluations

We set the success rate and the median function evaluations as the criterion for evaluating the algorithms. For each test function, we run each algorithm 30 times. We say that a run is successful if the function value hits the target accuracy within maximum iterations; and otherwise the run fails. We measure the performance of the algorithms by recording the median number of success function evaluations and the success rates over 30 runs. If all runs are successful, the success rate will be 1, see Table 4.2 for results. There are several useful observations in the table:

- It can be seen that the active-set evolution strategy can completely solve 11 test

Test Function	Active-set ES	Active-set method	SQP method	Interior-point method	LSHADE44 algorithm
success rate / the median number of objective function evaluations required					
g01	0.58/25	0.05/28	0.05/24	0.65/410	1.0/9,124
g02	0/-	0/-	0/-	0/-	0/-
g03	1.0/575	0.80/689	0.55/652	1.0/998	0/-
g04	1.0/22	1.0/28	1.0/30	1.0/60	1.0/23,810
g05	1.0/98	1.0/40	1.0/55	1.0/60	0/-
g06	1.0/4	1.0/33	1.0/37	1.0/104	1.0/7,159
g07	1.0/450	1.0/145	1.0/160	1.0/250	1.0/8,173
g08	0.74/203	0.05/44	0.10/82	0.80/52	0.91/1,874
g09	1.0/515	1.0/332	1.0/283	1.0/291	1.0/31,515
g10	1.0/200	0.99/319	0.95/387	0.70/1,153	0.92/88,536
g11	1.0/81	1.0/18	1.0/21	1.0/28	0/-
g12	0.97/195	0.10/14	0/-	0/-	1.0/4,184
g13	0.74/154	0.35/121	0.35/115	0.45/115	0/-
g14	0.94/1,164	1.0/376	1.0/382	1.0/328	0/-
g15	1.0/41	0.85/44	0.93/40	0.85/44	0/-
g16	1.0/16	0.05/96	1.0/193	0.95/772	1.0/25,563
g17	0.73/187	0/-	0.15/330	0/-	0/-
g18	0.71/48	0.70/95	0.60/120	0.85/385	0.97/71,744
g19	1.0/330	1.0/210	1.0/224	1.0/528	1.0/13,977
g20	0.77/77	0.25/525	0.75/425	0/-	0/-
g21	0.88/39	0.28/89	0.88/120	0.83/561	0/-
g22	0.70/79	0/-	0.10/2826	0.20/2,719	0/-
g23	0.90/87	0.90/150	0.90/160	0.90/307	0/-
g24	0.84/17	0.30/18	0.65/21	0.30/63	1.0/6,006

Table 4.2: Median number of objective function evaluations required and success rates for each problem for each algorithm.

problems out of 24. In comparison, the active-set method solves eight problems out of the total 24, and the SQP, interior-point method and LSHADE44 solve nine problems out of 24, respectively. If we also take into account problems that cannot be fully solved, LSHADE44 can only solve 50% among these problems while the active-set ES, active-set method, SQP method, and interior-point method solve 95.8% 87.5%, 91.7%, and 83.3%, respectively.

- The active-set, SQP and interior-point methods have either a higher success rate or a lower median number of objective function evaluations required compared with the active-set evolution strategy on $g04, g05, g07, g11, g14, g19$, and

$g24$. Among them, it is worth noting that active-set and SQP have great advantages on $g05, g07, g11$, and $g19$ because both methods can solve these problems completely and have smaller median numbers of objective function evaluations required compared with the active-set evolution strategy. The interior-point method has the same performances on $g05, g07, g11$.

- Problems for which no algorithm can solve completely, the active-set evolution strategy has either a higher success rate or a lower number of the median number of objective function evaluations required than other methods. Although LSHADE44 holds a higher success rate on some problems, such as $g08, g12, g18$, and $g24$, the median number of objective function evaluations required is about a maximum of 3,000 times more than the active-set evolution strategy has on $g18$, and a minimum of 14 times more on $g08$.

- It is worth noting that there is no single run that reaches the global minimum for $g02$ by any algorithm. The reason is $g02$ is a complex multimodal case that having a large number of local minima, which results in the algorithm has difficulty to converge to the global minima.

- For the high dimensional functions including problems $g20$ and $g22$ ($n \geq 22$), Active-set, SQP, and interior-point methods have relatively bad performances. While reviewing the objective functions we noticed that both functions also have a large number of constraints active at the optimal solution. The deterministic algorithms present a lower success rate with a higher median number of objective function evaluations required when solving these two problems, or they cannot find the optimal solution at all.

- The algorithm by Takahama and Sakai [40] is the best performing algorithm submitted to the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Using their algorithm to solve all of the problems with the median number of objective function evaluations required from 1,139 in $g08$ to 356,350 in $g19$. Overall, the active-set ES has greatly improved the median number of objective function evaluations require compared to the algorithm by Takahama and Sakai. Furthermore, their algorithm faces difficulties in solving $g20$ and $g22$

but the active-set ES does not. However, the advantage of their algorithm is they achieved 100% success rate on all other test functions except $g20$ and $g22$, even on $g02$.

4.4 Empirical Cumulative Running Time Distributions

To evaluate several algorithms tested by several test functions, we plotted the empirical cumulative running time distribution. For each problem, we create 20 different targets. The most strict target is set at $f(\mathbf{x}) + |(f^*) \times 10^{-8}|$, whereas the least strict target is determined by the median number of 100 random feasible solutions for each problem. The rest of the targets are generated to be logarithmically uniformly spaced between the most strict and the least strict targets. We then record the function evaluation number when each algorithm reaches every target. Figure 4.1 illustrates the distribution of the cumulative running time and aggregates the proportion of each test function that reaches all targets.

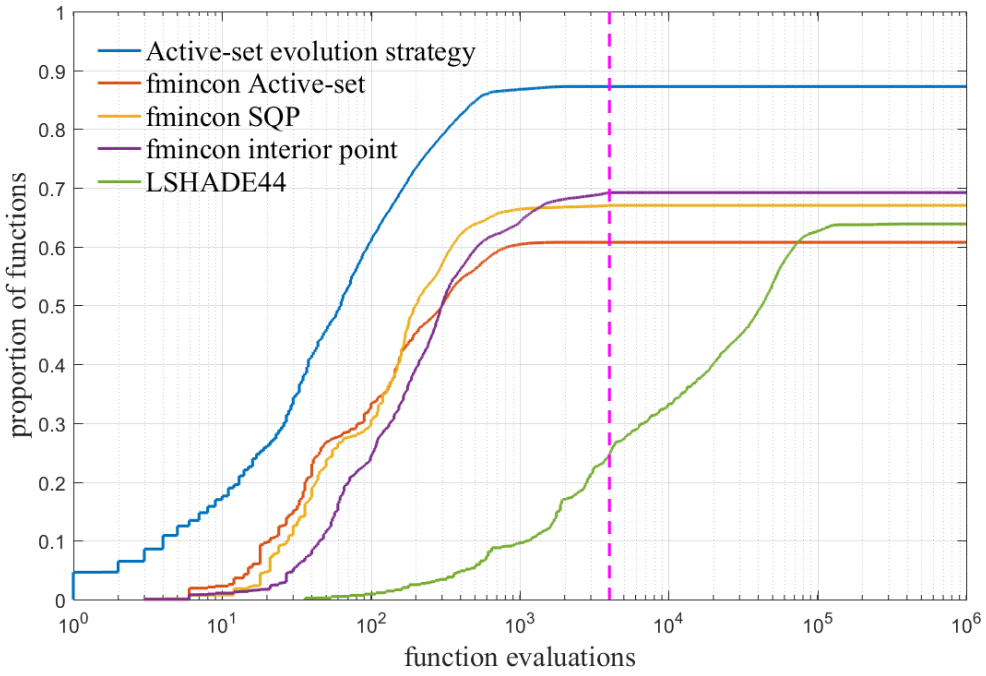
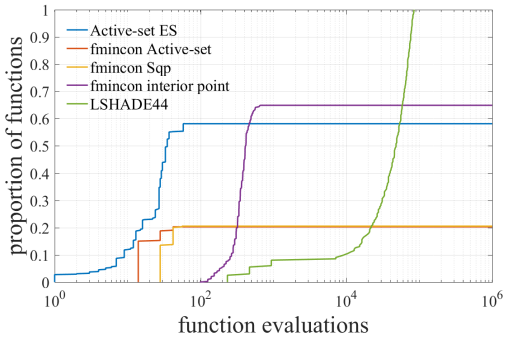
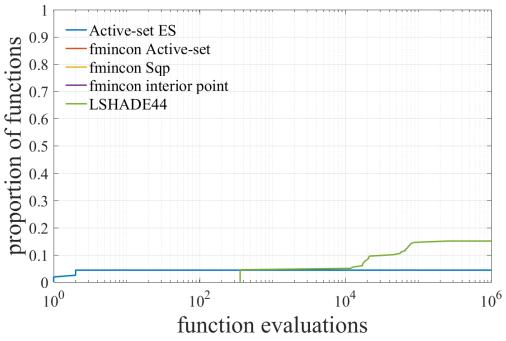


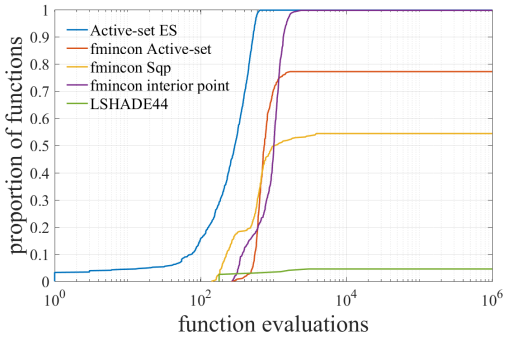
Figure 4.1: The empirical cumulative running time distributions of the active-set evolution strategy, active-set method, SQP method, interior-point method, and LSHADE44.



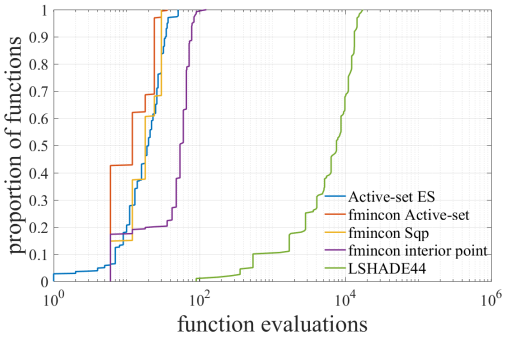
(a) g01



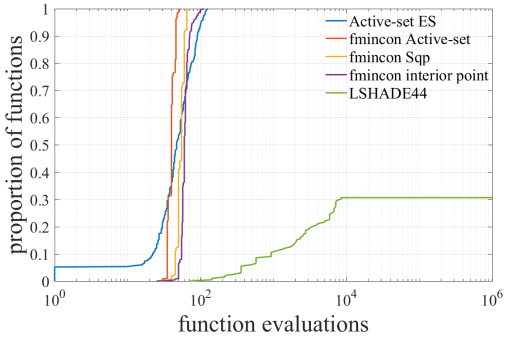
(b) g02



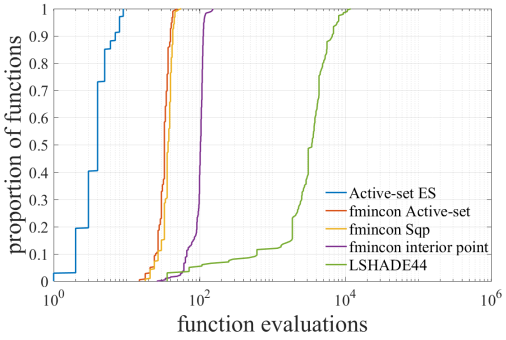
(c) g03



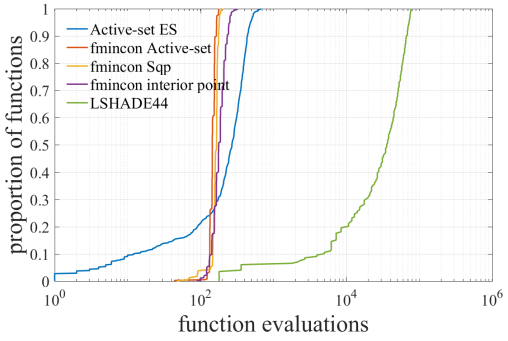
(d) g04



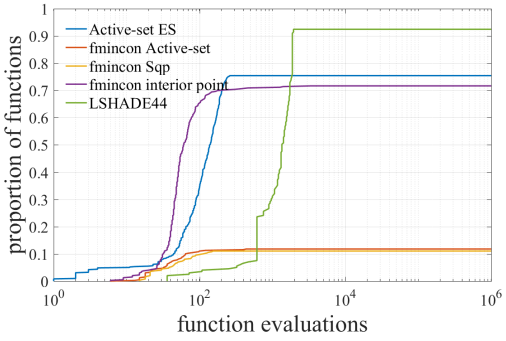
(e) g05



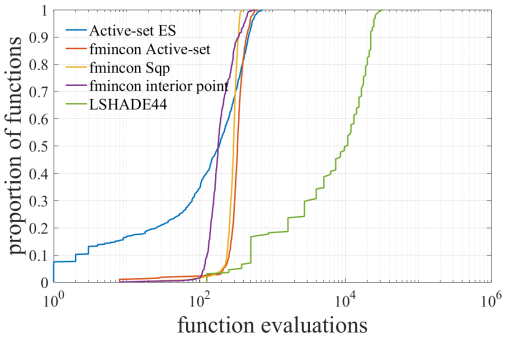
(f) g06



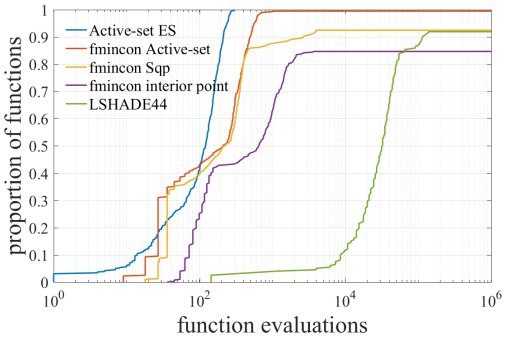
(g) g07



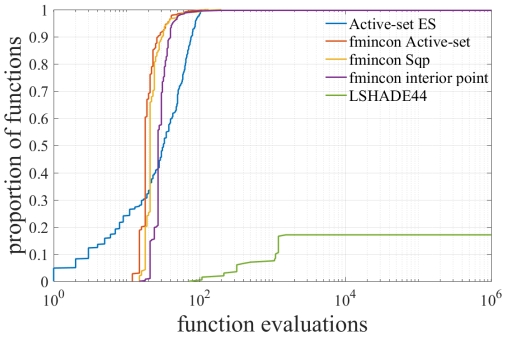
(h) g08



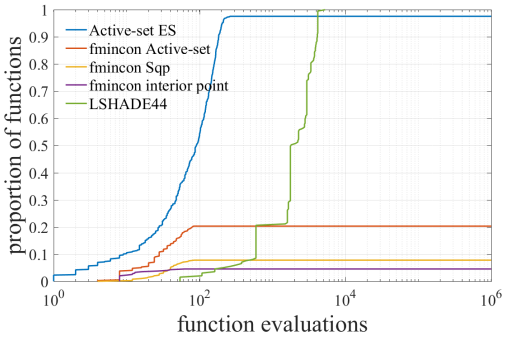
(i) g09



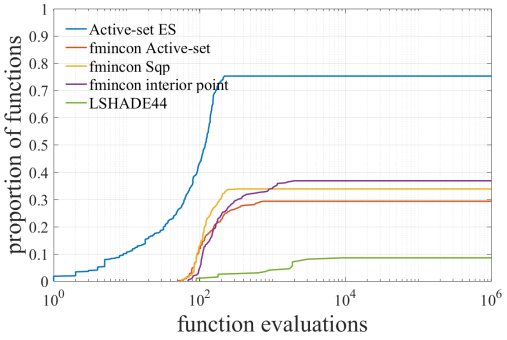
(j) g10



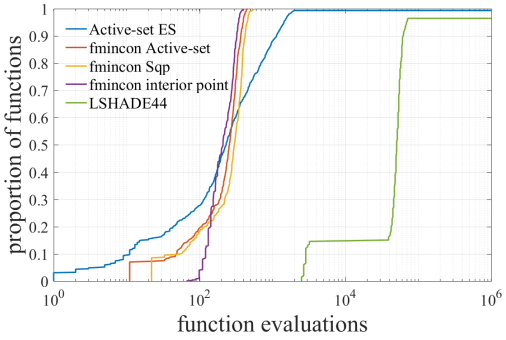
(k) g11



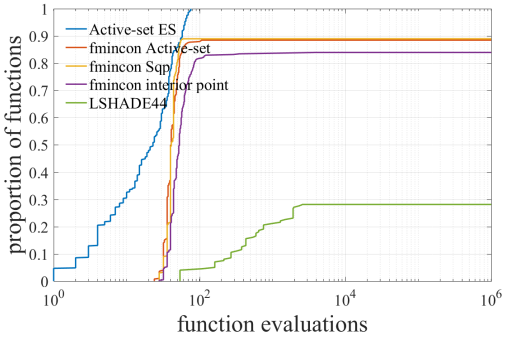
(l) g12



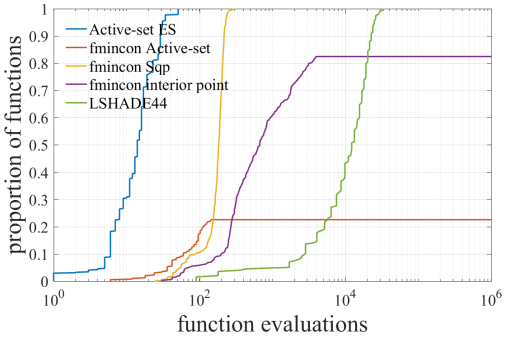
(m) g13



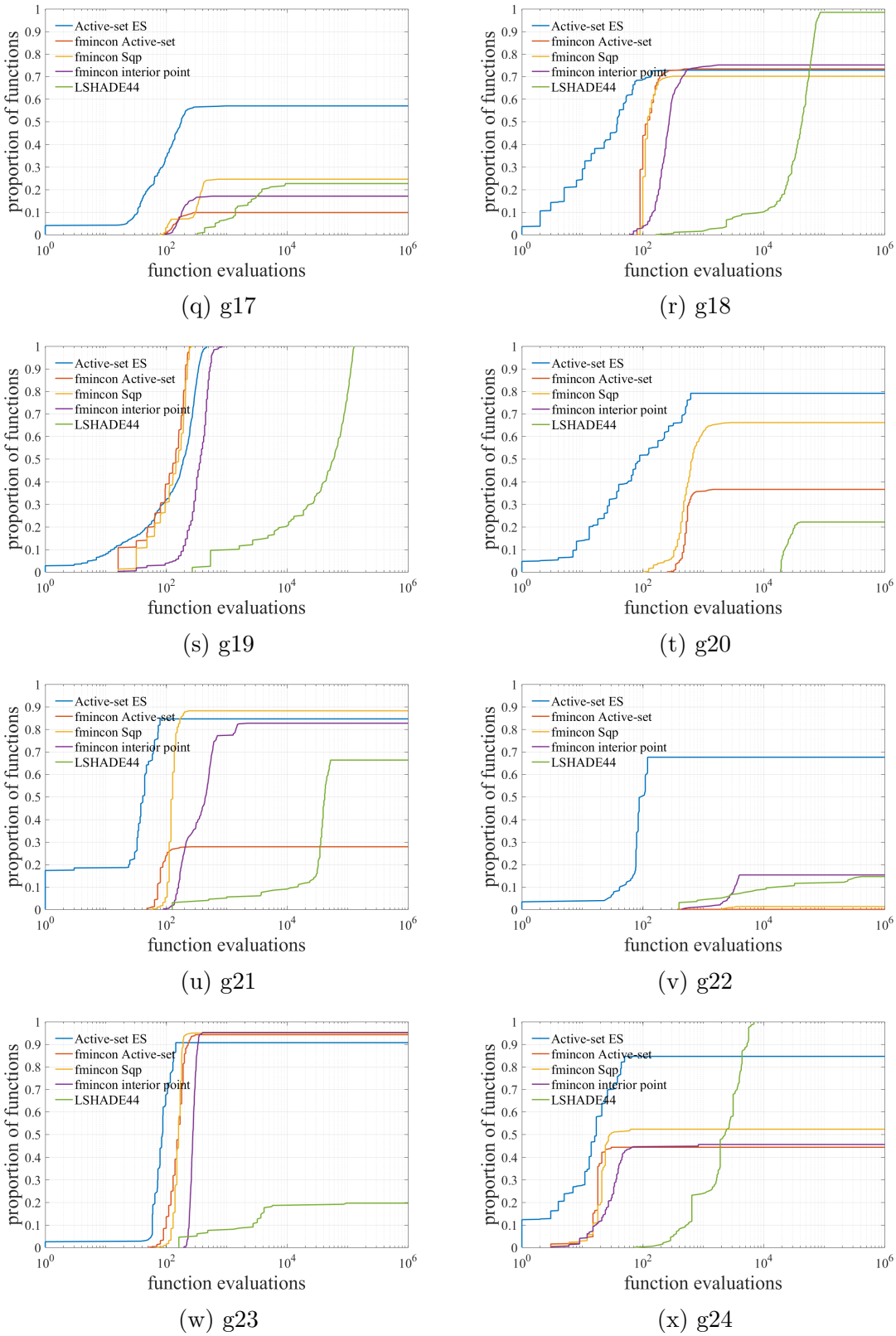
(n) g14



(o) g15



(p) g16



(q) g17 (r) g18

(s) g19 (t) g20

(u) g21 (v) g22

(w) g23 (x) g24

Figure 4.2: Individual comparison between the active-set ES method, *fmincon* algorithms and LSHADE44 algorithm for the test functions $g01$ to $g24$ empirical cumulative running time distribution.

The horizontal axis in Figure 4.1 indicates the number of function evaluations while the vertical axis is the proportion of targets that hit by the algorithms. It can be seen that 87.2% of the problems hit the targets among $g01$ to $g24$ with the active-set evolution strategy. At the same time, the active-set method solves 60.74%, the SQP method solves 67.02%, the interior-point solves 69.21%, and LSHADE44 solves 63.87% among the 24 problems. Setting a fixed evaluation number is also helpful for analyzing, see the magenta line in Figure 4.1 on page 52. We include a line when the function evaluation reaches 4,000, which makes it clear that the active-set evolution strategy already has 87.2% problems reaching all targets at this point, while LSHADE44 only has 25%. Therefore, this leads to the conclusion that the active-set evolution strategy converges faster than the other algorithms considered. This may be due to other algorithms needing to take several iterations at the beginning to have function values hit the targets, whereas the active-set evolution strategy does not need to this because it always projects the candidate solution onto the feasible region. This makes approximately 4.5% of the function values reach some targets although the function evaluation is 1.

The individual empirical cumulative running time distribution from $g01$ to $g24$ are shown in Figure 4.2. The axis information is the same as Figure 4.1, as well as the target values. There are several phenomena that can be observed:

- Problems $g01, g02, g08, g13, g21, g22$, and $g23$ have more than one local optima; therefore, there is a chance that the function value does not completely satisfy all targets.

- Not all runs are successful in $g12$ and $g24$ because of the existence of the disjointed feasible regions. Problem $g12$ has a feasible region that consists of 9^3 disjointed spheres, and the feasible region of $g24$ consists of two disconnected sub-regions. These disjointed feasible regions prevent the algorithm from locating the global optimum.

- Although there is no single run for $g02$ that located the global optimum, there are some function evaluations that did reach targets. This is because a small number of function evaluations hit the large targets before they fall into the local minima.

- We also notice that the active-set, SQP, and interior-point methods converge faster than the active-set evolution strategy in problems $g04, g07, g11, g15, g18$. The reason is that the objective functions of the above problems are quadratic; the principle of deterministic algorithms is modeling the objective function as a quadratic subproblem, which gives these algorithms an advantage in solving problems with quadratic objective functions.

Chapter 5

Conclusion

In this section, we present a summary of our method and experiments, and provide potential future work.

5.1 Summary

Arnold [2] combines the active-set method with (1+1)-ES in 2016. When too many inequality constraints have been added into the active-set, he proposes to suspend from using the whole active-set to avoid the algorithm wasting too much time in a non-optimal active-set. This method works well on the test problems $g01$ to $g11$ gathered by Michalewicz and Schoenauer [28] but is stagnated under some circumstances, so he modifies it and suggests a revised policy for considering to release the constraints [3]. Instead of suspending the whole active-set, he proposes to periodically release the individual constraints in the active-set, which significantly improves the algorithm performances on some problems among $g01$ to $g11$.

In our study, we adjusted some details based on the active-set ES from the PPSN conference [2] and the GECCO conference [3] so that it works well in the rest of the test functions summarized by Liang et al. [26]. We also tailored the releasing condition and the way to release the inequality constraints, speed up the factor of the step-size adaptation, and adjusted the options of the projection function. We then systematically compared the performances of both of the conferences with our modified version. We created a sphere function with different numbers of active constraints in different dimensions. The modified active-set ES has the best performance, especially in the high dimensional sphere with more active constraints.

To evaluate the performance of modified active-set ES, we compared it with three deterministic algorithms and one evolutionary algorithm on the test function set $g01$ to $g24$ which was held at the Congress on Evolutionary Computation Competitions in 2006. We used each algorithm to solve each problem 30 times and recorded the median

number of objective function evaluations required and the success rates required by the algorithm. We also created several function targets based on the feasible points' function value and plotted the proportion of algorithms that hit the targets versus the number of function evaluations for both individual function and the summation over 24 functions. From these experiments, we observed that the active-set and sequential quadratic programming algorithms converge faster on some quadratic problems, but it depends on the type of constraint functions. The LSHADE44 algorithm has a higher success rate on some problems but it is time-consuming. Among these five algorithms, the active-set ES can solve the most test problems. It has either a smaller function evaluation or a higher success rate, or even both. However, the active-set ES does not converge as quickly as the active-set and SQP algorithm on quadratic problems.

In summary, the active-set ES performs the best on the test functions from the Congress on Evolutionary Computation Competitions in 2006 when compared to the three deterministic algorithms and the LSHADE44 algorithm. The active-set ES shows a fair success rate and the median number of objective function evaluations required.

5.2 Future Work

Many open problems and work that can be done to improve our algorithm. Categorizing the test functions could be one potential work. Observing which kind of problem that each method is good at, and summarizing the advantages and weakness for each algorithm systematically will be beneficial. Moreover, it will be valuable to compare the active-set ES with other algorithms. For example, Sakamoto and Akimoto [36] proposed an adaptive ranking based constraint handling (ARCH) in 2019 which has similarities with active-set ES, and it is noteworthy to compare it with the active-set ES on the test functions which were held at the Congress on Evolutionary Computation Competitions in 2006.

We introduced a method to count the number of iterations for which the constraint has not been released, it will be useful to think about other ways to choose which constraints to release. Determining the relationship between the constraints may be helpful because releasing one constraint may also affect other constraints. An active-set ES that can solve $g02$ and also works for other functions simultaneously is

desirable.

Bibliography

[1] Youhei Akimoto, Anne Auger, and Nikolaus Hansen. Quality Gain Analysis of the Weighted Recombination Evolution Strategy on General Convex Quadratic Functions. *Theoretical Computer Science*, 832:42–67, 2018.

[2] Dirk V Arnold. An Active-set Evolution Strategy for Optimization with Known Constraints. In *International Conference on Parallel Problem Solving from Nature*, pages 192–202. Springer, 2016.

[3] Dirk V Arnold. Reconsidering Constraint Release for Active-set Evolution Strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 665–672. ACM, 2017.

[4] Dirk V Arnold and Jeremy Porter. Towards an Augmented Lagrangian Constraint Handling Approach for the (1+1)-ES. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 249–256, 2015.

[5] Asma Atamna, Anne Auger, and Nikolaus Hansen. Augmented Lagrangian Constraint Handling for CMA-ES Case of a Single Linear Constraint. In *International Conference on Parallel Problem Solving from Nature*, pages 181–191. Springer, 2016.

[6] Anne Auger and Nikolaus Hansen. Tutorial CMA-ES: Evolution Strategies and Covariance Matrix Adaptation. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 827–848, 2012.

[7] Anne Auger and Nikolaus Hansen. Introduction to Randomized Continuous Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 319–334, 2017.

[8] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution Strategies – A Comprehensive Introduction. *Natural Computing*, 1(1):3–52, 2002.

[9] Janez Brest and Mirjam Sepesy Maučec. Population Size Reduction for the Differential Evolution Algorithm. *Applied Intelligence*, 29(3):228–247, 2008.

[10] Carlos A Coello Coello. Constraint-handling Techniques Used with Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 675–701, 2017.

[11] Richard Courant. *Variational Methods for the Solution of Problems of Equilibrium and Vibrations*. Verlag nicht ermittelbar, 1943.

[12] Sébastien Le Digabel and Stefan M Wild. A Taxonomy of Constraints in Simulation-based Optimization. *arXiv preprint arXiv:1505.07881*, 2015.

[13] Thomas Epperly, Ross E Swaney, et al. Global Optimization Test Problems with Solutions, 1996.

[14] Christodoulos A Floudas and Panos M Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455. Springer Science & Business Media, 1990.

[15] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. Artificial Intelligence through Simulated Evolution. 1966.

[16] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[17] Nikolaus Hansen, Dirk V Arnold, and Anne Auger. Evolution Strategies. In *Springer Handbook of Computational Intelligence*, pages 871–898. Springer, 2015.

[18] Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[19] David M Himmelblau et al. *Applied Nonlinear Programming*. McGraw-Hill, 1972.

[20] Robert Hinterding and Zbigniew Michalewicz. Your Brains and My Beauty: Parent Matching for Constrained Optimisation. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*, pages 810–815. IEEE, 1998.

[21] Willi Hock and Klaus Schittkowski. Test Examples for Nonlinear Programming Codes. *Journal of optimization theory and applications*, 30(1):127–129, 1980.

[22] Stefan Kern, Sibylle D Müller, Nikolaus Hansen, Dirk Büche, Jiri Ocenasek, and Petros Koumoutsakos. Learning Probability Distributions in Continuous Evolutionary Algorithms – A Comparative Review. *Natural Computing*, 3(1):77–112, 2004.

[23] Slawomir Koziel and Zbigniew Michalewicz. A Decoder-based Evolutionary Algorithm for Constrained Parameter Optimization Problems. In *International Conference on Parallel Problem Solving from Nature*, pages 231–240. Springer, 1998.

[24] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary computation*, 7(1):19–44, 1999.

[25] Juan Luís J Laredo, Carlos Fernandes, Juan Julián Merelo, and Christian Gagné. Improving Genetic Algorithms Performance via Deterministic Population Shrinkage. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 819–826, 2009.

[26] Jing Liang, Thomas Philip Runarsson, Efrén Mezura-Montes, Maurice Clerc, PN Suganthan, Carlos A Coello Coello, and K Deb. Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-parameter Optimization. *Nangyang Technological University, Singapore, Tech. Rep*, 41, 2006.

[27] Efrén Mezura-Montes and Carlos A. Coello Coello. Constraint-handling in Nature-inspired Numerical Optimization: Past, Present and Future. *Swarm and Evolutionary Computation*, 1(4):173–194, 2011.

[28] Zbigniew Michalewicz, Kalyanmoy Deb, Martin Schmidt, and Thomas Stidsen. Test-case Generator for Nonlinear Continuous Parameter Optimization Techniques. *IEEE Transactions on Evolutionary Computation*, 4(3):197–215, 2000.

[29] Zbigniew Michalewicz and Cezary Z Janikow. Handling Constraints in Genetic Algorithms. In *International Conference on Genetic Algorithms*, pages 151–157, 1991.

[30] Zbigniew Michalewicz and Girish Nazhiyath. Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 647–651. IEEE, 1995.

[31] Zbigniew Michalewicz, Girish Nazhiyath, and Maciej Michalewicz. A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems. *Evolutionary programming*, 5(1):305–312, 1996.

[32] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[33] Radka Poláková, Josef Tvrdík, and Petr Bujok. Evaluating the Performance of L-SHADE with Competing Strategies on CEC2014 Single Parameter-operator Test Suite. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 1181–1187. IEEE, 2016.

[34] Ingo Rechenberg. *Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. PhD thesis, TU Berlin, 1970.

[35] Thomas P. Runarsson and Xin Yao. Stochastic Ranking for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[36] Naoki Sakamoto and Youhei Akimoto. Adaptive Ranking based Constraint Handling for Explicitly Constrained Black-box Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 700–708, 2019.

[37] Patrick Spettel, Hans-Georg Beyer, and Michael Hellwig. A Covariance Matrix Self-adaptation Evolution Strategy for Optimization under Linear Constraints. *IEEE Transactions on Evolutionary Computation*, 23(3):514–524, 2018.

[38] Rainer Storn. On the Usage of Differential Evolution for Function Optimization. In *Proceedings of North American Fuzzy Information Processing*, pages 519–523. IEEE, 1996.

[39] Rainer Storn and Kenneth Price. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[40] Tetsuyuki Takahama and Setsuko Sakai. Constrained Optimization by the ε Constrained Differential Evolution with Gradient-based Mutation and Feasible Elites. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1–8. IEEE, 2006.

[41] Ke Tang, Xiaodong Li, Ponnuthurai N Suganthan, Zhenyhu Yang, and Thomas Weise. Benchmark Functions for the CEC 2010 Special Session and Competition on Large Scale Global Optimization. *Technical report, University of Science and Technology of China*, 2010.

[42] Guohua Wu, Rammohan Mallipeddi, and Ponnuthurai N Suganthan. Problem Definitions and Evaluation Criteria for the CEC 2017 Competition on Constrained Real-parameter Optimization. *National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report*, 2017.

[43] Quanshi Xia. Global Optimization Test Problems, 1996.

Appendix A

Test functions

The test functions are gathered by Liang et al. [26]. The original references are given for each problem as well. The objective function, constraint functions and the optimal function value are listed.

$g01$ [14]

Minimize:

$$f(\mathbf{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} {x_i}^2 - \sum_{i=5}^{13} x_i \tag{A.1}$$

subject to:

$$\begin{aligned}
\mathrm{g}_1(\mathbf{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
\mathrm{g}_2(\mathbf{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
\mathrm{g}_3(\mathbf{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
\mathrm{g}_4(\mathbf{x}) &= -8x_1 + x_{10} \leq 0 \\
\mathrm{g}_5(\mathbf{x}) &= -8x_2 + x_{11} \leq 0 \\
\mathrm{g}_6(\mathbf{x}) &= -8x_3 + x_{12} \leq 0 \\
\mathrm{g}_7(\mathbf{x}) &= -2x_4 - x_5 + x_{10} \leq 0 \\
\mathrm{g}_8(\mathbf{x}) &= -2x_6 - x_7 + x_{11} \leq 0 \\
\mathrm{g}_9(\mathbf{x}) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned} \tag{A.2}$$

where the bounds are $0 \leq x_i \leq 1$ $(i = 1, \ldots, 9), 0 \leq x_i \leq 100$ $(i = 10, 11, 12)$ and $0 \leq x_{13} \leq 1$. $f(\mathbf{x}^*) = -15$.

$g02$ [24]

Minimize:

$$f(\mathbf{x}) = -\left|\frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}}\right| \tag{A.3}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= 0.75 - \prod_{i=1}^{n} x_i \leq 0 \\ \mathrm{g}_2(\mathbf{x}) &= \sum_{i=1}^{n} x_i - 7.5n \leq 0 \end{aligned} \tag{A.4}$$

where $n = 20$ and $0 < x_i \leq 10$ $(i = 1, \ldots, n)$. $f(\mathbf{x}^*) = -0.803619104125199$.

$g03$ [31]

Minimize:

$$f(\mathbf{x}) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i \tag{A.5}$$

subject to:

$$\mathrm{h}_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0 \tag{A.6}$$

where $n = 10$ and $0 \leq x_i \leq 1$ $(i = 1, \ldots, n)$. $f(\mathbf{x}^*) = -1$.

$g04$ [19]

Minimize:

$$f(\mathbf{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \tag{A.7}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0 \\ \mathrm{g}_2(\mathbf{x}) &= -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0 \\ \mathrm{g}_3(\mathbf{x}) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0 \\ \mathrm{g}_4(\mathbf{x}) &= -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0 \\ \mathrm{g}_5(\mathbf{x}) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0 \\ \mathrm{g}_6(\mathbf{x}) &= -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0 \end{aligned} \tag{A.8}$$

where $78 \leq \mathbf{x}_1 \leq 102, 33 \leq \mathbf{x}_2 \leq 45$ and $27n \leq \mathbf{x}_i \leq 45$ $(i = 3, 4, 5)$. $f(\mathbf{x}^*) = -30665.53867178332$.

$g05$ [21]

Minimize:

$$f(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3 \tag{A.9}$$

subject to:

$$\begin{aligned}
&\mathrm{g}_1(\mathbf{x}) = -x_4 + x_3 - 0.55 \leq 0 \\
&\mathrm{g}_2(\mathbf{x}) = -x_3 + x_4 - 0.55 \leq 0 \\
&\mathrm{h}_3(\mathbf{x}) = 1000\sin\left(-x_3 - 0.25\right) + 1000\sin\left(-x_4 - 0.25\right) + 894.8 - x_1 = 0 \\
&\mathrm{h}_4(\mathbf{x}) = 1000\sin\left(x_3 - 0.25\right) + 1000\sin\left(x_3 - x_4 - 0.25\right) + 894.8 - x_2 = 0 \\
&\mathrm{h}_5(\mathbf{x}) = 1000\sin\left(x_4 - 0.25\right) + 1000\sin\left(x_4 - x_3 - 0.25\right) + 1294.8 = 0
\end{aligned} \tag{A.10}$$

where $0 \leq x_1 \leq 1200, 0 \leq x_2 \leq 1200, -0.55 \leq x_3 \leq 0.55$ and $-0.55 \leq x_4 \leq 0.55$. $f\left(\mathbf{x}^*\right) = 5126.498109595271$.

$g06$ [14]

Minimize:

$$f(\mathbf{x}) = \left(x_1 - 10\right)^3 + \left(x_2 - 20\right)^3 \tag{A.11}$$

subject to:

$$\begin{aligned}
&\mathrm{g}_1(\mathbf{x}) = -\left(x_1 - 5\right)^2 - \left(x_2 - 5\right)^2 + 100 \leq 0 \\
&\mathrm{g}_2(\mathbf{x}) = \left(x_1 - 6\right)^2 + \left(x_2 - 5\right)^2 - 82.81 \leq 0
\end{aligned} \tag{A.12}$$

where $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. $f\left(\mathbf{x}^*\right) = -6961.813875580147$.

$g07$ [21]

Minimize:

$$\begin{aligned}
f(\mathbf{x}) = {} & x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + \left(x_3 - 10\right)^2 + 4\left(x_4 - 5\right)^2 + \left(x_5 - 3\right)^2 \\
& + 2\left(x_6 - 1\right)^2 + 5x_7^2 + 7\left(x_8 - 11\right)^2 + 2\left(x_9 - 10\right)^2 + \left(x_{10} - 7\right)^2 + 45
\end{aligned} \tag{A.13}$$

subject to:

$$\begin{aligned}
&\mathrm{g}_1(\mathbf{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0 \\
&\mathrm{g}_2(\mathbf{x}) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \\
&\mathrm{g}_3(\mathbf{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\
&\mathrm{g}_4(\mathbf{x}) = 3\left(x_1 - 2\right)^2 + 4\left(x_2 - 3\right)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \\
&\mathrm{g}_5(\mathbf{x}) = 5x_1^2 + 8x_2 + \left(x_3 - 6\right)^2 - 2x_4 - 40 \leq 0 \\
&\mathrm{g}_6(\mathbf{x}) = x_1^2 + 2\left(x_2 - 2\right)^2 - 2x_1 x_2 + 14x_5 - 6x_6 \leq 0 \\
&\mathrm{g}_7(\mathbf{x}) = 0.5\left(x_1 - 8\right)^2 + 2\left(x_2 - 4\right)^2 + 3x_5^2 - x_6 - 30 \leq 0 \\
&\mathrm{g}_8(\mathbf{x}) = -3x_1 + 6x_2 + 12\left(x_9 - 8\right)^2 - 7x_{10} \leq 0
\end{aligned} \tag{A.14}$$

where $-10 \leq x_i \leq 10\ (i = 1, \ldots, 10)$. $f(\mathbf{x}^*) = 24.306209068179840$.

$g08$ [24]

Minimize:

$$f(\mathbf{x}) = -\frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)} \tag{A.15}$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= x_1^2 - x_2 + 1 \leq 0 \\ g_2(\mathbf{x}) &= 1 - x_1 + (x_2 - 4)^2 \leq 0 \end{aligned} \tag{A.16}$$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. $f(\mathbf{x}^*) = -0.095825041418033$.

$g09$ [21]

Minimize:

$$\begin{aligned} f(\mathbf{x}) = &(x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 \\ &+10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \end{aligned} \tag{A.17}$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\ g_2(\mathbf{x}) &= -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\ g_3(\mathbf{x}) &= -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \\ g_4(\mathbf{x}) &= 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \end{aligned} \tag{A.18}$$

where $-10 \leq x_i \leq 10$ for $(i = 1, \ldots, 7)$. $f(\mathbf{x}^*) = 680.6300573744027$.

$g10$ [21]

Minimize:

$$f(\mathbf{x}) = x_1 + x_2 + x_3 \tag{A.19}$$

subject to:

$$
\begin{aligned}
g_1(\mathbf{x}) &= -1 + 0.0025\,(x_4 + x_6) \leq 0 \\
g_2(\mathbf{x}) &= -1 + 0.0025\,(x_5 + x_7 - x_4) \leq 0 \\
g_3(\mathbf{x}) &= -1 + 0.01\,(x_8 - x_5) \leq 0 \\
g_4(\mathbf{x}) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0 \\
g_5(\mathbf{x}) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0 \\
g_6(\mathbf{x}) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0
\end{aligned}
\tag{A.20}
$$

where $100 \leq x_1 \leq 10000, 1000 \leq x_i \leq 10000\ (i = 2, 3)$ and $10 \leq x_i \leq 1000\ (i = 4, \ldots, 8)$. $f(\mathbf{x}^*) = 7049.248020528665$.

$g11$ [24]

Minimize:

$$f(\mathbf{x}) = x_1^2 + (x_2 - 1)^2 \tag{A.21}$$

subject to:

$$\mathrm{h}(\mathbf{x}) = x_2 - x_1^2 = 0 \tag{A.22}$$

where $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. $f(\mathbf{x}^*) = 0.7500$.

$g12$ [24]

Minimize:

$$f(\mathbf{x}) = -(100 - \textstyle\sum_{i=1}^{3}(x_i - 5)^2/100 \tag{A.23}$$

subject to:

$$g(\mathbf{x}) = \sum_{i=1}^{3}(x_i - (\min(9, (\max(1, \lfloor \mathbf{x} \rceil))))^2 - 0.0625 \leq 0 \tag{A.24}$$

where $0 \leq x_i \leq 10\ (i = 1, 2, 3)$. $f(\mathbf{x}^*) = -1$.

$g13$ [21]

Minimize:

$$f(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5} \tag{A.25}$$

subject to:

$$\begin{aligned} \mathrm{h}_1(\mathbf{x}) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ \mathrm{h}_2(\mathbf{x}) &= x_2 x_3 - 5 x_4 x_5 = 0 \\ \mathrm{h}_3(\mathbf{x}) &= x_1^3 + x_2^3 + 1 = 0 \end{aligned} \tag{A.26}$$

where $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$. $f(\mathbf{x}^*) = 0.053949847770272$.

$g14$ [19]

Minimize:

$$f(\mathbf{x}) = \sum_{i=1}^{10} x_i \left(c_i + \ln \frac{x_i}{\sum_{j=1}^{10} x_j} \right) \tag{A.27}$$

subject to:

$$\begin{aligned} \mathrm{h}_1(\mathbf{x}) &= x_1 + 2x_2 + +2x_3 + x_6 + x_{10} - 2 = 0 \\ \mathrm{h}_2(\mathbf{x}) &= x_4 + 2x_5 + x_6 + x_7 - 1 = 0 \\ \mathrm{h}_3(\mathbf{x}) &= x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 = 0 \end{aligned} \tag{A.28}$$

where the bounds are $0 < x_i \leq 10 (i = 1, \ldots, 10)$, and $c_1 = -6.089, c_2 = -17.164, c_3 = -34.054$ $c_4 = -5.914, c_5 = -24.721, c_6 = -14.986, c_7 = -24.1, c_8 = -10.708, c_9 = -26.662, c_{10} = -22.179$. $f(\mathbf{x}^*) = -47.761090859346020$.

$g15$ [19]

Minimize:

$$f(\mathbf{x}) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1 x_2 - x_1 x_3 \tag{A.29}$$

subject to:

$$\begin{aligned} \mathrm{h}_1(\mathbf{x}) &= x_1^2 + x_2^2 + x_3^2 - 25 = 0 \\ \mathrm{h}_2(\mathbf{x}) &= 8x_1 + 14x_2 + 7x_3 - 56 = 0 \end{aligned} \tag{A.30}$$

where the bounds are $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$. $f(\mathbf{x}^*) = 961.7151721300521$.

$g16$ [19]

Minimize:

$$\begin{aligned} f(\mathbf{x}) = 0.000117 y_{14} + 0.1365 + 0.00002358 y_{13} + 0.000001502 y_{16} + 0.0321 y_{12} \\ +0.004324 y_5 + 0.0001 \tfrac{c_{15}}{c_{16}} + 37.48 \tfrac{y_2}{c_{12}} - 0.0000005843 y_{17} \end{aligned} \tag{A.31}$$

subject to:

$$
\begin{aligned}
& g_1(\mathbf{x}) = \frac{0.28}{0.72} y_5 - y_4 \leq 0 \\
& \mathrm{g}_2(\mathbf{x}) = x_3 - 1.5 x_2 \leq 0 \\
& g_3(\mathbf{x}) = 3496 \frac{y_2}{c_{12}} - 21 \leq 0 \\
& g_4(\mathbf{x}) = 110.6 + y_1 - \frac{62212}{c_{17}} \leq 0 \\
& \mathrm{g}_5(\mathbf{x}) = 213.1 - y_1 \leq 0 \\
& g_6(\mathbf{x}) = y_1 - 405.23 \leq 0 \\
& g_7(\mathbf{x}) = 17.505 - y_2 \leq 0 \\
& g_8(\mathbf{x}) = y_2 - 1053.6667 \leq 0 \\
& g_9(\mathbf{x}) = 11.275 - y_3 \leq 0 \\
& \mathrm{g}_{10}(\mathbf{x}) = y_3 - 35.03 \leq 0 \\
& \mathrm{g}_{11}(\mathbf{x}) = 214.228 - y_4 \leq 0 \\
& \mathrm{g}_{12}(\mathbf{x}) = y_4 - 665.585 \leq 0 \\
& \mathrm{g}_{13}(\mathbf{x}) = 7.458 - y_5 \leq 0 \\
& g_{14}(\mathbf{x}) = y_5 - 584.463 \leq 0 \\
& \mathrm{g}_{15}(\mathbf{x}) = 0.961 - y_6 \leq 0 \\
& \mathrm{g}_{16}(\mathbf{x}) = y_6 - 265.916 \leq 0 \\
& g_{17}(\mathbf{x}) = 1.612 - y_7 \leq 0 \\
& \mathrm{g}_{18}(\mathbf{x}) = y_7 - 7.046 \leq 0 \\
& g_{19}(\mathbf{x}) = 0.146 - y_8 \leq 0 \\
& \mathrm{g}_{20}(\mathbf{x}) = y_8 - 0.222 \leq 0 \\
& \mathrm{g}_{21}(\mathbf{x}) = 107.99 - y_9 \leq 0 \\
& \mathrm{g}_{22}(\mathbf{x}) = y_9 - 273.366 \leq 0 \\
& \mathrm{g}_{23}(\mathbf{x}) = 922.693 - y_{10} \leq 0 \\
& \mathrm{g}_{24}(\mathbf{x}) = y_{10} - 1286.105 \leq 0 \\
& \mathrm{g}_{25}(\mathbf{x}) = 926.832 - y_{11} \leq 0
\end{aligned}
\tag{A.32}
$$

$$
\begin{aligned}
&g_{26}(\mathbf{x}) = y_{11} - 1444.046 \leq 0 \\
&g_{27}(\mathbf{x}) = 18.766 - y_{12} \leq 0 \\
&g_{28}(\mathbf{x}) = y_{12} - 537.141 \leq 0 \\
&g_{29}(\mathbf{x}) = 1072.163 - y_{13} \leq 0 \\
&g_{30}(\mathbf{x}) = y_{13} - 3247.039 \leq 0 \\
&g_{31}(\mathbf{x}) = 8961.448 - y_{14} \leq 0 \\
&g_{32}(\mathbf{x}) = y_{14} - 26844.086 \leq 0 \\
&g_{33}(\mathbf{x}) = 0.063 - y_{15} \leq 0 \\
&g_{34}(\mathbf{x}) = y_{15} - 0.386 \leq 0 \\
&g_{35}(\mathbf{x}) = 71084.33 - y_{16} \leq 0 \\
&g_{36}(\mathbf{x}) = -140000 + y_{16} \leq 0 \\
&g_{37}(\mathbf{x}) = 2802713 - y_{17} \leq 0 \\
&g_{38}(\mathbf{x}) = y_{17} - 12146108 \leq 0
\end{aligned}
$$

where:

$$
\begin{aligned}
&y_1 = x_2 + x_3 + 41.6 \\
&c_1 = 0.024x_4 - 4.62 \\
&y_2 = \frac{12.5}{c_1} + 12 \\
&c_2 = 0.0003535x_1^2 + 0.5311x_1 + 0.08705y_2x_1 \\
&c_3 = 0.052x_1 + 78 + 0.002377y_2x_1 \\
&y_3 = \frac{c_2}{c_3} \\
&y_4 = 19y_3 \\
&c_4 = 0.04782\left(x_1 - y_3\right) + \frac{0.1956\left(x_1 - y_3\right)^2}{x_2} + 0.6376y_4 + 1.594y_3 \\
&c_5 = 100x_2 \\
&c_6 = x_1 - y_3 - y_4 \\
&c_7 = 0.950 - \frac{c_4}{c_5}
\end{aligned}
\tag{A.33}
$$

$$
\begin{aligned}
y_5 &= c_6 c_7 \\
y_6 &= x_1 - y_5 - y_4 - y_3 \\
c_8 &= (y_5 + y_4)\, 0.995 \\
y_7 &= \frac{c_8}{y_1} \\
y_8 &= \frac{c_8}{3798} \\
c_9 &= y_7 - \frac{0.0663 y_7}{y_8} - 0.3153 \\
y_9 &= \frac{96.82}{c_9} + 0.321 y_1 \\
y_{10} &= 1.29 y_5 + 1.258 y_4 + 2.29 y_3 + 1.71 y_6 \\
y_{11} &= 1.71 x_1 - 0.452 y_4 + 0.580 y_3 \\
c_{10} &= \frac{12.3}{752.3} \\
c_{11} &= (1.75 y_2)\,(0.995 x_1) \\
c_{12} &= 0.995 y_{10} + 1998 \\
y_{12} &= c_{10} x_1 + \frac{c_{11}}{c_{12}} \\
y_{13} &= c_{12} - 1.75 y_2 \\
y_{14} &= 3623 + 64.4 x_2 + 58.4 x_3 + \frac{146312}{y_9 + x_5} \\
c_{13} &= 0.995 y_{10} + 60.8 x_2 + 48 x_4 - 0.1121 y_{14} - 5095 \\
y_{15} &= \frac{y_{13}}{c_{13}} \\
y_{16} &= 148000 - 331000 y_{15} + 40 y_{13} - 61 y_{15} y_{13} \\
c_{14} &= 2324 y_{10} - 28740000 y_2 \\
y_{17} &= 14130000 - 1328 y_{10} - 531 \\
& y_{11} + \frac{c_{14}}{c_{12}} \\
c_{15} &= \frac{y_{13}}{y_{15}} - \frac{y_{13}}{0.52} \\
c_{16} &= 1.104 - 0.72 y_{15} \\
c_{17} &= y_9 + x_5
\end{aligned}
\tag{A.34}
$$

and where 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 $25 \leq x_5 \leq 84.1988$. $f(\mathbf{x}^*)$ = -1.905155258534784.

$g17$ [19]

Minimize:

$$f(\mathbf{x}) = f(x_1) + f(x_2) \tag{A.35}$$

where:

$$f_1(x_1) = \begin{cases} 30x_1 & 0 \leq x_1 < 300 \\ 31x_1 & 300 \leq x_1 < 400 \end{cases}$$
$$f_2(x_2) = \begin{cases} 28x_2 & 0 \leq x_2 < 100 \\ 29x_2 & 100 \leq x_2 < 200 \\ 30x_2 & 200 \leq x_2 < 1000 \end{cases} \tag{A.36}$$

subject to:

$$\begin{aligned} \mathrm{h}_1(\mathbf{x}) &= -x_1 + 300 - \tfrac{x_3 x_4}{131.078}\cos(1.48477 - x_6) + \tfrac{0.90798 x_3^2}{131.078}\cos(1.47588) \\ \mathrm{h}_2(\mathbf{x}) &= -x_2 - \tfrac{x_3 x_4}{131.078}\cos\Big((1.48477 + x_6) + \tfrac{0.90798 x_4^2}{131.078}\cos(1.47588) \\ \mathrm{h}_3(\mathbf{x}) &= -x_5 - \tfrac{x_3 x_4}{131.078}\sin\Big((1.48477 + x_6) + \tfrac{0.90798 x_4^2}{131.078}\sin(1.47588) \\ \mathrm{h}_4(\mathbf{x}) &= 200 - \tfrac{x_3 x_4}{131.078}\sin\Big((1.48477 - x_6) + \tfrac{0.90798 x_3^2}{131.078}\sin(1.47588) \end{aligned} \tag{A.37}$$

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 $0 \leq x_6 \leq 0.5236$. $f(\mathbf{x}^*) = 8853.539891329588$.

$g18$ [19]

Minimize:

$$f(\mathbf{x}) = -0.5\left(x_1 x_4 - x_2 x_3 + x_3 x_9 - x_5 x_9 + x_5 x_8 - x_6 x_7\right) \tag{A.38}$$

subject to:

$$
\begin{aligned}
g_1(\mathbf{x}) &= x_3^2 + x_4^2 - 1 \leq 0 \\
g_2(\mathbf{x}) &= x_9^2 - 1 \leq 0 \\
g_3(\mathbf{x}) &= x_5^2 + x_6^2 - 1 \leq 0 \\
g_4(\mathbf{x}) &= x_1^2 + (x_2 - x_9)^2 - 1 \leq 0 \\
g_5(\mathbf{x}) &= (x_1 - x_5)^2 + (x_2 - x_6)^2 - 1 \leq 0 \\
g_6(\mathbf{x}) &= (x_1 - x_7)^2 + (x_2 - x_8)^2 - 1 \leq 0 \\
g_7(\mathbf{x}) &= (x_3 - x_5)^2 + (x_4 - x_6)^2 - 1 \leq 0 \\
g_8(\mathbf{x}) &= (x_3 - x_7)^2 + (x_4 - x_8)^2 - 1 \leq 0 \\
g_9(\mathbf{x}) &= x_7^2 + (x_8 - x_9)^2 - 1 \leq 0 \\
g_{10}(\mathbf{x}) &= x_2 x_3 - x_1 x_4 \leq 0 \\
g_{11}(\mathbf{x}) &= -x_3 x_9 \leq 0 \\
g_{12}(\mathbf{x}) &= x_5 x_9 \leq 0 \\
g_{13}(\mathbf{x}) &= x_6 x_7 - x_5 x_8 \leq 0
\end{aligned}
\tag{A.39}
$$

where the bounds are $-10 \leq x_i \leq 10\ (i = 1, \ldots, 8)$ and $0 \leq x_9 \leq 20$. $f(\mathbf{x}^*) = -0.866025403784439$.

$g19$ [19]

Minimize:

$$
f(\mathbf{x}) = \sum_{j=1}^{5} \sum_{i=1}^{5} c_{ij} x_{(10+i)} x_{(10+j)} + 2 \sum_{j=1}^{5} d_j x_{(10+j)}^3 - \sum_{i=1}^{10} b_i x_i \tag{A.40}
$$

subject to:

$$
g_j(\mathbf{x}) = -2 \sum_{i=1}^{5} c_{ij} x_{(10+i)} - 3 d_j x_{(10+j)}^2 - e_j + \sum_{i=1}^{10} a_{ij} x_i \leq 0 \quad j = 1, \ldots, 5 \tag{A.41}
$$

where $\vec{b} = [-40, -2, -0.25, -4, -4, -1, -40, -60, 5, 1]$ and the remaining data is on Table 1. The bounds are $0 \leq x_i \leq 10\ (i = 1, \ldots, 15)$. $f(\mathbf{x}^*) = 32.655592950246340$.

$g20$ [19]

Minimize:

$$
f(\mathbf{x}) = \sum_{i=1}^{24} a_i x_i \tag{A.42}
$$

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

Table A.1: Data set for test problem $g19$.

subject to:

$$\begin{aligned}
&g_i(\mathbf{x}) = \frac{\left(x_i + x_{(i+12)}\right)}{\sum_{j=1}^{24} x_j + e_i} \leq 0 \quad i = 1, 2, 3 \\
&\mathrm{g}_i(\mathbf{x}) = \frac{\left(x_{(i+3)} + x_{(i+15)}\right)}{\sum_{j=1}^{24} x_j + e_i} \leq 0 \quad i = 4, 5, 6 \\
&\mathrm{h}_i(\mathbf{x}) = \frac{x_{(i+12)}}{b_{(i+12)} \sum_{j=13}^{24} \frac{x_j}{b_j}} - \frac{c_i x_i}{40 b_i \sum_{j=1}^{12} \frac{x_j}{b_j}} = 0 \quad i = 1, \ldots, 12 \\
&\mathrm{h}_{13}(\mathbf{x}) = \sum_{i=1}^{24} x_i - 1 = 0 \\
&\mathrm{h}_{14}(\mathbf{x}) = \sum_{i=1}^{12} \frac{x_i}{d_i} + k \sum_{i=13}^{24} \frac{x_i}{b_i} - 1.671 = 0
\end{aligned} \tag{A.43}$$

where $k = (0.7302)(530)\left(\frac{14.7}{40}\right)$ and the data set is detailed on Table 2. The bounds are $0 \leq x_i \leq 10$ $(i = 1, \ldots, 24)$. $f(\mathbf{x}^*) = 0.147466071547197$.

$g21$ [13]

Minimize:

$$f(\mathbf{x}) = x_1 \tag{A.44}$$

subject to:

i	a_i	b_i	c_i	d_i	e_i
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			

Table A.2: Data set for test problem $g20$

$$
\begin{aligned}
&g_1(\mathbf{x}) = -x_1 + 35x_2^{0.6} + 35x_3^{0.6} \leq 0 \\
&\mathrm{h}_1(\mathbf{x}) = -300x_3 + 7500x_5 - 7500x_6 - 25x_4x_5 + 25x_4x_6 + x_3x_4 = 0 \\
&\mathrm{h}_2(\mathbf{x}) = 100x_2 + 155.365x_4 + 2500x_7 - x_2x_4 - 25x_4x_7 - 15536.5 = 0 \\
&\mathrm{h}_3(\mathbf{x}) = -x_5 + \ln(-x_4 + 900) = 0 \\
&\mathrm{h}_4(\mathbf{x}) = -x_6 + \ln(x_4 + 300) = 0 \\
&\mathrm{h}_5(\mathbf{x}) = -x_7 + \ln(-2x_4 + 700) = 0
\end{aligned}
\tag{A.45}
$$

where the bounds are $0 \leq x_1 \leq 1000, 0 \leq x_2, x_3 \leq 40, 100 \leq x_4 \leq 300, 6.3 \leq x_5 \leq 6.7, 5.9 \leq x_6 \leq 6.4$ and $4.5 \leq x_7 \leq 6.25$. $f(\mathbf{x}^*) = 193.7881988317070$.

$g22$ [13]

Minimize:

$$f(\mathbf{x}) = x_1 \tag{A.46}$$

subject to:

$$\begin{aligned}
&\mathrm{g}_1(\mathbf{x}) = -x_1 + x_2^{0.6} + x_3^{0.6} + x_4^{0.6} \leq 0 \\
&\mathrm{h}_1(\mathbf{x}) = x_5 - 100000x_8 + 1 \times 10^7 = 0 \\
&\mathrm{h}_2(\mathbf{x}) = x_6 + 100000x_8 - 100000x_9 = 0 \\
&\mathrm{h}_3(\mathbf{x}) = x_7 + 100000x_9 - 5 \times 10^7 = 0 \\
&\mathrm{h}_4(\mathbf{x}) = x_5 + 100000x_{10} - 3.3 \times 10^7 = 0 \\
&\mathrm{h}_5(\mathbf{x}) = x_6 + 100000x_{11} - 4.4 \times 10^7 = 0 \\
&\mathrm{h}_6(\mathbf{x}) = x_7 + 100000x_{12} - 6.6 \times 10^7 = 0 \\
&\mathrm{h}_7(\mathbf{x}) = x_5 - 120x_2x_{13} = 0 \\
&\mathrm{h}_8(\mathbf{x}) = x_6 - 80x_3x_{14} = 0 \\
&\mathrm{h}_9(\mathbf{x}) = x_7 - 40x_4x_{15} = 0 \\
&\mathrm{h}_{10}(\mathbf{x}) = x_8 - x_{11} + x_{16} = 0 \\
&\mathrm{h}_{11}(\mathbf{x}) = x_9 - x_{12} + x_{17} = 0 \\
&\mathrm{h}_{12}(\mathbf{x}) = -x_{18} + \ln(x_{10} - 100) = 0 \\
&\mathrm{h}_{13}(\mathbf{x}) = -x_{19} + \ln(-x_8 + 300) = 0 \\
&\mathrm{h}_{14}(\mathbf{x}) = -x_{20} + \ln(x_{16}) = 0 \\
&\mathrm{h}_{15}(\mathbf{x}) = -x_{21} + \ln(-x_9 + 400) = 0 \\
&\mathrm{h}_{16}(\mathbf{x}) = -x_{22} + \ln(x_{17}) = 0 \\
&\mathrm{h}_{17}(\mathbf{x}) = -x_8 - x_{10} + x_{13}x_{18} - x_{13}x_{19} + 400 = 0 \\
&\mathrm{h}_{18}(\mathbf{x}) = x_8 - x_9 - x_{11} + x_{14}x_{20} - x_{14}x_{21} + 400 = 0 \\
&\mathrm{h}_{19}(\mathbf{x}) = x_9 - x_{12} - 4.60517x_{15} + x_{15}x_{22} + 100 = 0
\end{aligned} \tag{A.47}$$

where the bounds are $0 \leq x_1 \leq 20000, 0 \leq x_2, x_3, x_4 \leq 1 \times 10^6, 0 \leq x_5, x_6, x_7 \leq 4 \times 10^7, 100 \leq x_8 \leq 299.99, 100 \leq x_9 \leq 399.99, 100.01 \leq x_{10} \leq 300, 100 \leq x_{11} \leq 400, 100 \leq x_{12} \leq 600, 0 \leq x_{13}, x_{14}, x_{15} \leq 500, 0.01 \leq x_{16} \leq 300, 0.01 \leq x_{17} \leq 400, -4.7 \leq x_{18}, x_{19}, x_{20}, x_{21}, x_{22} \leq 6.25$. $f(\mathbf{x}^*) = 236.3703133145661$.

$g23$ [14]

Minimize:

$$f(\mathbf{x}) = -9x_5 - 15x_8 + 6x_1 + 16x_2 + 10(x_6 + x_7) \tag{A.48}$$

subject to:

$$\begin{aligned}
\mathrm{g}_1(\mathbf{x}) &= x_9 x_3 + 0.02x_6 - 0.025x_5 \leq 0 \\
\mathrm{g}_2(\mathbf{x}) &= x_9 x_4 + 0.02x_7 - 0.015x_8 \leq 0 \\
\mathrm{h}_1(\mathbf{x}) &= x_1 + x_2 - x_3 - x_4 = 0 \\
\mathrm{h}_2(\mathbf{x}) &= 0.03x_1 + 0.01x_2 - x_9\left(x_3 + x_4\right) = 0 \\
\mathrm{h}_3(\mathbf{x}) &= x_3 + x_6 - x_5 = 0 \\
\mathrm{h}_4(\mathbf{x}) &= x_4 + x_7 - x_8 = 0
\end{aligned} \tag{A.49}$$

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 $0.01 \leq x_9 \leq 0.03$. $f\left(\mathbf{x}^*\right) = -400$.

$g24$ [43]

Minimize:

$$f(\mathbf{x}) = -x_1 - x_2 \tag{A.50}$$

subject to:

$$\begin{aligned}
g_1(\mathbf{x}) &= -2x_1^4 + 8x_1^3 - 8x_1^2 + x_2 - 2 \leq 0 \\
g_2(\mathbf{x}) &= -4x_1^4 + 32x_1^3 - 88x_1^2 + 96x_1 + x_2 - 36 \leq 0
\end{aligned} \tag{A.51}$$

where the bounds are $0 \leq x_1 \leq 3$ and $0 \leq x_2 \leq 4$. $f\left(\mathbf{x}^*\right) = -5.508013271595251$.