# AN ACTIVE-SET EVOLUTION STRATEGY FOR MIXED-INTEGER BLACK-BOX OPTIMIZATION WITH EXPLICIT CONSTRAINTS

by

Yuan Hong

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

at

Dalhousie University
Halifax, Nova Scotia
April 2023

# Table of Contents

# List of Tables

# List of Figures

## Abstract

Many real-world applications involve the optimization of both continuous and discrete variables simultaneously. Evolution Strategies are stochastic black-box optimization strategies, which are most commonly applied to continuous optimization. Black-box optimization refers to tasks where the objective function values of specific points can be obtained, but no additional information is available. Gradients can only be approximated through finite differencing. In recent studies, the Active-Set Evolution Strategy (AS-ES) has been designed for constrained continuous optimization problems, which assumes that the objective function is a black box, but the constraint functions are explicit and computationally inexpensive to evaluate. This assumption allows the algorithm to evaluate the feasibility of constraints at multiple points before spending an objective function evaluation. In this thesis, we propose a Mixed-Integer Active-Set Evolution Strategy ($\text{AS}_{\text{mixInt}}$-ES) for solving black-box mixed-integer optimization problems with explicit constraints. In contrast to Branch-and-Bound techniques, the proposed algorithm does not require solving problems at each node to be optimal to prune the search tree. Instead, a heuristic is employed in place of the bounding mechanism to determine which node to propagate forward. After introducing the design of $\text{AS}_{\text{mixInt}}$-ES, we analyze the behaviour of the algorithm on a series of linearly constrained sphere problems. In addition, we conduct computational experiments to compare its performance against several algorithms designed for constrained optimization.

## Acknowledgements

I would like to express my deepest appreciation to my supervisor Dr. Arnold for his invaluable patience and feedback. He generously shared his knowledge and expertise with me. This thesis would not have been accomplished without his support and dedicated participation throughout the entire process. In addition, many thanks to my parents who provided generous support and encouragement. The trust they have in me has maintained my spirit and motivation at every stage of my Master's program.

# Chapter 1

# Introduction

Optimization is widely used in various fields, such as engineering, economics, mathematics, etc. Optimization aims to find the minimum or maximum value of a function in a domain of definition. Real-world optimization problems can vary widely in terms of their objective functions, constraints, and variable types in different areas. Two common types of optimization problems are continuous and discrete optimization. The focus of continuous optimization is finding optimal solutions for problems over a continuous domain. The problem variables are represented by real-valued variables. In discrete optimization, the problem variables are represented by integer, binary, or categorical values. Research in Evolutionary Computation is diverse, and some of it is tailored to solve purely continuous or discrete optimization problems. In practice, many academic and industrial problems require the optimization of both continuous and discrete variables simultaneously. These problems are classically referred to as mixed-discrete optimization problems. Such optimization problems frequently arise in various real-world problems and application fields such as engineering design, process industry, finance, automobile engineering, aircraft design, etc.

Black-box functions do not have explicit functional forms and can be non-smooth, discontinuous, or computationally expensive to evaluate. In a black-box setting, the cost of optimization is often measured in terms of the number of function evaluations required to locate an optimal solution. Since black-box optimization algorithms do not assume any specific structure or characteristics of the functions being optimized, they may require costly computer simulations to evaluate the functions at sampled points. Using as few objective function evaluations as possible to find optimal solutions can significantly reduce the overall computational cost of the optimization process. This is particularly important in cases where the evaluation of the objective function is computationally expensive. In contrast, gradient-based methods require the objectives and constraints to be continuous and differentiable. These methods

utilize information about the gradient or its approximation to direct the search for optimal solutions.

Mixed-discrete optimization problems are often subject to a set of linear or nonlinear constraints, which restrict the feasibility of the solutions in the search space. A taxonomy that categorizes constraints using four criteria has been proposed by Le Digabel and Wild [19], including Quantifiable or Non-quantifiable, Relaxable or Unrelaxable, A priori or Simulation-based, and Known or Hidden. The taxonomic notation QRAK implies that the constraint is quantifiable in that it can provide a meaningful numerical value to indicate degrees of its violation/feasibility; the constraint is relaxable in that it allows for the evaluation of the objective function at infeasible points, leading to meaningful results; the constraint is given a priori and its feasibility can be verified without running a simulation; and gradient information of the known constraint may be available or approximated at little cost. Non-quantifiable constraints provide Boolean values to reflect their violation/feasibility; for unrelaxable constraints, an infeasible point is not meaningfully interpreted by objective function; for simulation-based constraints, each candidate solution is evaluated by both the objective and constraint functions; hidden constraints are not given explicitly. In certain instances, the constraint functions of a problem may be considered black boxes and could be as computationally expensive as the objective function to evaluate. However, in other cases, constraint function values may be calculated at a negligible computational cost compared to the objective function evaluation. Under this taxonomy, we consider constraints to be quantifiable and a priori, which are referred to as explicit constraints.

Evolution strategies (ES) [63, 81] are stochastic black-box optimization strategies that are commonly used for continuous optimization. They are based on principles from biological evolution theory, including selection, recombination, and mutation. They are more likely to escape local optima and find the global optimum due to their stochastic nature. In recent years, several adaptive variants of ES have been proposed, each with its strengths for dealing with various types of constrained black-box optimization problems where constraints are given a priori, based on the taxonomy of Le Digabel and Wild [19]. These ES variants include Adaptive Ranking Constraint Handling (ARCH) [69, 70], Active-Set Evolution Strategies (AS-ES) [6, 7, 83], and

lcCMSA-ES [84]. They assume that constraint functions are computationally cheaper to calculate than the objective function and employ larger numbers of constraint function evaluations than objective function evaluations. Assuming constraints are given explicitly, the (1+1)-AS-ES as described by Spettel et al. [83] exhibits good performance on a set of 24 constrained continuous problems from the 2006 IEEE Congress on Evolutionary Computation (CEC) competition benchmark gathered by Liang et al. [53].

## 1.1 Motivation

There are several nature-inspired algorithms for mixed-integer black-box optimization. Those algorithms are considered stochastic black-box optimization algorithms and they do not make any assumptions about the structure or properties of the objective function and constraints present in a given problem. Those algorithms do not distinguish between the cost of evaluating the objective function and constraint functions. Assuming that the constraints are explicit, based on the taxonomy of Le Digabel and Wild [19], and that integrality constraints are relaxable, we argue that it is possible to achieve near-optimal solutions with significantly fewer objective function evaluations than those required by the aforementioned algorithms. In this thesis, we propose an evolutionary algorithm for constrained black-box mixed-integer optimization with explicit constraints. The Branch-and-Bound algorithm [48, 16] involves a tree search while utilizing specific rules that restrict the search to a sub-tree. Inspired by this method, we focus on implementing a branching mechanism for handling integrality constraints. By utilizing the explicitness of the constraints, we rely on the (1+1)-AS-ES algorithm by Spettel et al. [83], performing a mutation operation and employing active-set techniques to project infeasible candidate solutions onto the feasible search space. In addition, the performance of our proposed algorithm will be evaluated and compared with other comparable algorithms. The goal of this thesis is to demonstrate the advantages of the algorithm that exploits the explicit nature of constraints.

## 1.2 Contribution

In this thesis, we develop a new algorithm in the field of Evolutionary Computation called the Mixed-Integer Active-Set Evolution Strategy. Our algorithm aims to solve constrained mixed-integer optimization problems, where the objective function is treated as a black box and the constraint functions are explicitly given. To achieve this, we draw inspiration from the work of Dakin [16] and utilize a branching technique to handle binary variables only or both binary and integer-valued variables. To deal with the resulting continuous optimization problems, we use the (1+1)-AS-ES algorithm [83] as an underlying solver. However, due to the presence of nonlinearities and nonconvexities in objective and constraint functions, it may not be possible to guarantee that the optimization problem is solved to global optimality, and be challenging to determine effective bounds. Therefore, different from classical Branch-and-Bound techniques, the proposed algorithm does not require the continuous optimization problem at each node to be solved to optimality. Instead of the bounding scheme, we propose heuristics to determine when to split a problem into subproblems and then eliminate them from further consideration. The fundamental idea underlying the proposed method is to revisit the choice of a node to propagate forward, applying a single step of (1+1)-AS-ES at each iteration, to possibly minimize the required effort.

A benchmark of 57 constrained optimization problems motivated by industrial applications for the 2020 IEEE Congress on Evolutionary Computation competition has been gathered by Kumar et al. [47]. Their collection includes several mixed-integer problems. During our examination, we observed that several of these problems have undergone modifications since their original proposal, resulting in significant and often undesirable changes to their characteristics in some cases. In this thesis, we evaluate the performance of the proposed algorithm on two sets of test functions for constrained mixed-integer optimization [14, 47]. In addition to mixed-integer problems, we also examine several other problems from the CEC 2020 problem set [47] that do not involve any integrality constraints. These problems are of interest due to their real-world applicability. Another contribution of our work is that we have carefully examined and corrected a subset of test problems in the CEC 2020 benchmark.

## 1.3 Outline

The remainder of this thesis is organized as follows. Chapter 2 introduces some necessary terminology and background information required for understanding the proposed algorithm. Additionally, we highlight related research as well as benchmarks for evaluating the proposed algorithm. In Chapter 3, we describe our algorithm, which we refer to as Mixed-Integer Active-Set Evolution Strategy ($\text{AS}_{\text{mixInt}}$-ES), and analyze the performance on simple test problems. Experiments on constrained mixed-integer problems are conducted in Chapter 4, along with comparisons between the proposed algorithm and several other algorithms. Finally, Chapter 5 summarizes conclusions and provides open questions for future research.

# Chapter 2

# Background and Literature Review

In this chapter, we provide some necessary background information for the proposed algorithm in Chapter 3 and present a literature review for constrained continuous optimization and mixed-integer optimization. Section 2.1 reviews some relevant knowledge about mixed-integer optimization, including the concept of a mixed-integer optimization problem. Then, an introduction of the general Branch-and-Bound algorithm is accompanied by a detailed description of two important decisions in Branch-and-Bound. This is followed by other integrality constraint handling techniques. In Section 2.2, we survey comparable nature-inspired algorithms to constrained continuous optimization and constrained mixed-integer optimization. Section 2.3 gives a summary of active set evolution strategies for constrained continuous optimization, along with pseudo-code and descriptions of implementations. Last, we will highlight the benchmarks for evaluating the proposed algorithm in Section 2.4.

## 2.1 Background

In this section, some necessary background information is introduced for a better understanding of the process of mixed-integer optimization, which is used in the later chapters.

### 2.1.1 Mixed-Integer Programming

In general, optimization refers to a process of finding minimum or maximum values of a function. The problem to be optimized is defined by an objective function, subject to a set of constraints that define the feasible domain of optimization variables. Unless otherwise noted, we will focus on the minimization problem in the following sections.

The general form of constrained optimization problem is defined as:

$$\begin{array}{lll} (P) \quad \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 & \text{for } i \in \{1, 2, \ldots, l\} \\ & h_j(\mathbf{x}) = 0 & \text{for } j \in \{1, 2, \ldots, m\} \\ & \mathbf{x} \in \mathbb{R}^n & \end{array} \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function and $g_i(\mathbf{x})$ is referred to as inequality constraints and $h_j(\mathbf{x})$ as equality constraints. These constraints can be linear or non-linear. There are bound constraints on all variables $\mathbf{x}$, i.e., $\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$. We assume that lower and upper bounds may be finite or infinite. The feasible region is the set $D = \{\mathbf{x} \in \mathbb{R}^n \mid g_i(\mathbf{x}) \leq 0 \text{ for all } i \in \{1, 2, \ldots, l\}, h_j(\mathbf{x}) = 0 \text{ for all } j \in \{1, 2, \ldots, m\}\}$. An inequality constraint $g_i$ is said to be active at $\mathbf{x} \in D$ if $g_i(\mathbf{x}) = 0$; it is said to be inactive if $g_i(\mathbf{x}) < 0$. Equality constraints $h_j$ are always active. The active set $A(\mathbf{x}) \subseteq \{1, 2, \ldots, l\}$ is the set of the indices of all inequality constraints active at $\mathbf{x} \in D$.

A point $\mathbf{x}^*$ is called a *local minimizer* of objective function $f(\mathbf{x})$, if and only if there is an open neighbourhood $S(\mathbf{x}^*)$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in S(\mathbf{x}^*) \cap D$$

A *local minimum* is the function value associated with a local minimizer. A point $\mathbf{x}^*$ is called a *global minimizer*, if and only if

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in D$$

The function value of the point $\mathbf{x}^*$ is called a *global minimum*. *Multimodal* functions include multiple local minimizers, whereas *unimodal* functions only have one local minimizer.

Mixed-discrete programming problems are mathematical programs where some variables take on discrete values. A *Mixed-Integer Programming* (MIP) problem is a special case of mixed-discrete programming problems in which a specified subset of the variables are required to take on integer values. The set $I \subseteq \{1, \ldots, n\}$ is the index

set of integer variables. Variables that are restricted to the values of zero and one are commonly referred to as binary variables. We denote by $n_{\text{int}} = |I|$ the dimension of the integer space. We denote the subvector of integer variables by $\mathbf{x}_I \in \mathbb{Z}^{n_{\text{int}}}$. To distinguish between the constraints described in Eq. (2.1) and integrality constraints (i.e. all integer variables $\mathbf{x}_I \in \mathbb{Z}^{n_{\text{int}}}$ take integer values), we will use the term *feasible* to refer to points that satisfy the former, and *integer-feasible* to describe those points that satisfy all constraints, including integrality constraints. *Infeasible* refers to points in the search space that violate at least one constraint. We write $D_{\text{int}} \subseteq \mathbb{R}^n$ for the set of all integer-feasible points.

This general type of problem finds applications in various fields, such as chemical engineering, mechanical engineering, automobile engineering, and aerospace. If the MIP problem contains a linear objective function and all the constraints are linear in form, then the problem is termed a *Mixed-Integer Linear Programming* (MILP) problem. If the MIP problem involves nonlinearities in the objective function or constraints, the problem is termed a *Mixed-Integer Nonlinear Programming* (MINLP) problem, which includes both *Nonlinear Programming* (NLP) and *Mixed-Integer Linear Programming* (MILP) as subproblems. Solving such problems can be challenging. The discrete nature of integer variables results in the search space becoming discontinuous, potentially making it more difficult for optimization algorithms to locate the global optimum compared with continuous optimization problems. In addition, non-convexity increases the difficulty of finding a globally optimal solution.

### 2.1.2 Black-Box Optimization

In a black-box scenario, we assume no inside knowledge of the formulation of the optimization problem or how the objective function value is calculated. In particular, gradients of the objective function are not explicitly available. It is possible that obtaining an objective function value requires a computationally expensive simulation or even conducts a physical experiment. In constrained black-box optimization, the objective function is typically considered a black box because it may lack a mathematical representation. However, the constraint functions may or may not be black boxes. In some cases, such as bound constraints, these functions are known and can be evaluated at a relatively low cost. A taxonomy for constraint functions allowing for

the comparison of different algorithms for constrained optimization problems has been developed by Le Digabel and Wild [19]. It includes Quantifiable or Non-quantifiable, Relaxable or Unrelaxable, A priori or Simulation-based, and Known or Hidden.

- **Quantifiable/Non-quantifiable**: A quantifiable constraint (Q) provides a meaningful numerical value to indicate degrees of constraint violation or feasibility in the search space, whereas a non-quantifiable constraint (N) only returns a Boolean value to reflect the constraint's feasibility.

- **Relaxable/Unrelaxable**: During the optimization process, relaxable constraints (R) allow the evaluation of infeasible points on the objective function, while unrelaxable constraints (U) do not.

- **A priori/Simulation-based**: An a priori constraint (A) can evaluate the constraint's feasibility directly, such as linear equalities or bound constraints based on the type of variables involved (e.g., continuous, integer, binary, or discrete). A simulation-based constraint (S) cannot be evaluated analytically, and requires a simulation to verify if a candidate solution violates the constraint.

- **Known/Hidden**: Known constraints (K) are explicitly stated in the problem definition, while hidden constraints (H) are not defined explicitly or are not known to the solving algorithm. A hidden constraint cannot be quantifiable, relaxable, or a priori since we cannot detect the violation or feasibility.

The taxonomy utilizes four letters that classify constraint types. For instance, QRAK represents the constraint as quantifiable, relaxable, a priori, and known. Le Digabel and Wild [19] point out that this is the most common type of constraint found in nonlinear optimization. We use the term *explicit constraints* to refer to constraints that are quantifiable and a priori. Bounds and integrality constraints are common examples of explicit constraints, but other linear or nonlinear constraints can also be explicit.

### 2.1.3 Active-Set Method

The active-set method is a widely used numerical optimization technique to handle constrained optimization problems that involve both inequality and equality constraints. The active-set method aims to solve an optimization problem by transforming it into a sequence of quadratic programming (QP) subproblems. The QP subproblem involves minimizing a quadratic function subject to equality constraints and active inequality constraints, where the quadratic function is obtained by approximating the Lagrangian function quadratically. Solving a subproblem involves iteratively finding a search direction that converges to a local optimum. At each iteration, some of the inequality constraints are treated as equality constraints, and added to the working set $W$, while the remaining inequality constraints not in $W$ are disregarded. The active-set method then computes Lagrange multipliers to determine whether the solution $\mathbf{x}^*$ obtained for the working set $W$ satisfies the Karush-Kuhn-Tucker (KKT) conditions [61], then removes constraints with negative Lagrange multipliers. The working set at the optimum $\mathbf{x}^*$ is the active set $A(\mathbf{x}^*)$ of the solution. The active-set method is known for its effectiveness in identifying the optimal solution in finite iterations [61].

### 2.1.4 Branch-and-Bound

Many industrial problems have been formulated as MILP problems. The methods used to solve MILP problems differ significantly from those used to solve Linear Programming (LP) problems. To solve a MILP, the solution of the LP relaxation (obtained by relaxing the integrality constraints on the integer variables) is usually required at each step of the algorithm. Four classes of algorithms for MILP problems were outlined by Floudas [22]. Branch-and-Bound is one of the most commonly used methods for handling integer programming. The earliest work on Branch-and-Bound for MILP is by Land and Doig [48], dating back to the early 1960s. LP-based Branch-and-Bound proposed by Dakin [16] has become the standard paradigm for most modern solvers for solving MILPs. In this context, we will give an overview of the general idea behind the Branch-and-Bound method for MILPs, along with a detailed description of two selection schemes.

#### 2.1.4.1 General Branch-and-Bound Method Overview

In order to describe the algorithm it is necessary to introduce some notation and terminology. We use the term *node* or *subproblem* to denote the problem associated with a certain part of the feasible region of MILPs. Generally, a node stores a candidate solution, its corresponding objective function value, and a constraint set. Roughly speaking, Branch-and-Bound is a Divide-and-Conquer method that divides the original problem into a series of smaller subproblems and then recursively solves each subproblem. Each subproblem is either a continuous or mixed-integer optimization problem with potentially fewer integer variables than the original MILP problem. The entire procedure of a Branch-and-Bound method can be viewed as a tree search, with all variables regarded as continuous at the root node of the tree. The term *branch* refers to a constructed tree structure. The term *bound* refers to bounding the best feasible solution in the sub-tree and pruning the search tree to avoid complete enumeration if its bound indicates that it will not contain a better solution in the following sub-trees. The search tree is built in a recursive way to solve a relaxation of a candidate node to optimality (i.e. the relaxation obtained by relaxing the integrality restrictions on the variables). A candidate solution that is feasible for the continuous relaxation is said to be an *integer-feasible solution* if all integer variables $x_d$ take integer values at the solution. Moreover, the best integer-feasible solution found so far (a.k.a *incumbent solution*) $\mathbf{x}^* \in \mathbb{R}^n$ is stored globally. Set $\mathcal{N}$ keeps a list of active nodes that need to be explored further.

The search starts by solving the continuous relaxation ($P$) of the MILP problem to optimality (i.e., the relaxation obtained by simply ignoring the integrality requirements on variables $x_d \in \mathbb{Z}, \forall d \in I$). At each iteration, Branch-and-Bound selects a node from the set $\mathcal{N}$ of active nodes for exploration. The algorithm maintains the incumbent solution and the corresponding objective function value in the search. Assuming we consider the minimization problem, if an integer-feasible solution $\mathbf{x} \in \mathbb{R}^n$ with a better objective function value is found, i.e., $f(\mathbf{x}) < f(\mathbf{x}^*)$, the incumbent solution is then updated. Conversely, if the integer-feasible solution of a candidate node is not better than the best one found so far, the candidate node is terminated and removed from the set $\mathcal{N}$. On the other hand, if some of the integer variables are fractional at the optimal solution of a relaxed problem, then one can split the

problem into two new subproblems by selecting one of those integer variables $x_d \in \mathbb{Z}$ for some $d \in I$. In one of the subproblems, the constraint $x_d \leq \lfloor \bar{x_d} \rfloor$ is added to the associated constraint set, and in the other, the constraint $x_d \geq \lceil \bar{x_d} \rceil$ is added, where $\bar{x_d}$ represents the value of variable $x_d$. The parent node is removed and new nodes are added to the set of active nodes. One of the nodes from set $\mathcal{N}$ is selected and solved next. Note that an objective function value stored in a node is the optimal function value of the relaxation of the MILP problem given all of the constraints, including new bound constraints. After exploring all nodes of set $\mathcal{N}$, the algorithm terminates and the best incumbent solution in the search is returned. The complete Branch-and-Bound algorithm for solving MILPs is described in Algorithm 1.

**Algorithm 1** General Branch-and-Bound Algorithm

**Input:** initial objective function value $f^* = +\infty$, set $\mathcal{N}$ of active nodes
**Output:** optimal solution $\mathbf{x}^*$ of MILP

1: **while** $\mathcal{N} \neq \emptyset$ **do**
2: Select and remove a node $\mathcal{Q}$ from $\mathcal{N}$.
3: Solve a relaxation $(P)$ corresponding to the selected node $\mathcal{Q}$, and let $\mathbf{x}$ be an optimal solution of the relaxation and $f_{\mathbf{x}} = f(\mathbf{x})$ be its objective value.
4: **if** $\mathbf{x}$ is infeasible **then**
5: Jump to Line 2.
6: **end if**
7: **if** $\mathbf{x}$ is integer-feasible **then**
8: **if** $f^* > f_{\mathbf{x}}$ **then**
9: Let $\mathbf{x}^* \leftarrow \mathbf{x}$ and $f^* \leftarrow f_{\mathbf{x}}$. ▷ Update incumbent solution
10: Remove nodes from $\mathcal{N}$ whose objective function value is inferior to $f^*$.
11: **end if**
12: **else**
13: Branch on variable $x_d$ for $d \in I$ with a non-integer value.
14: Create node $\mathcal{L}$ with constraint $x_d \leq \lfloor \bar{x_d} \rfloor$.
15: Create node $\mathcal{R}$ with constraint $x_d \geq \lceil \bar{x_d} \rceil$.
16: Add nodes $\mathcal{L}$ and $\mathcal{R}$ to $\mathcal{N}$.
17: **end if**
18: **end while**

The complete search of a tree is not always necessary. It may be possible to prune a sub-tree (subproblem) in Line 7 if the solution of a node is infeasible, then no feasible solution in its sub-trees could be found. If the optimal objective function value of a node is greater than or equal to that of the incumbent solution in Line 8, then no improved solution in its sub-trees can be found and further branching from

this node is no longer required.

Compared to MILP problems, MINLP problems involve nonlinearities in the objective function or constraints. Algorithms for MINLP problems have a lot in common with methods for solving MILP problems. Branch-and-Bound algorithm can be extended to in a natural way to handle convex MINLP problems. Dakin [16] suggested that this Branch-and-Bound approach used for solving linear problems can also be applied to non-linear problems, although he did not provide any computational evidence. An implementation of the Branch-and-Bound approach for convex MINLP problems was also suggested by Gupta and Ravindran [27], which used the underlying NLP solver OPT, implemented with a generalized reduced gradient method. Leyffer [51] presented an integrated algorithm for convex MINLP problems which interlaces a Sequential Quadratic Programming method at a node with the Branch-and-Bound method tree search and applies an early branching heuristic as described by Borchers and Mitchell [10]. The MINLP problems are generally non-convex due to the presence of nonlinearities. Even when the integer variables are relaxed to be continuous, the feasible region may be non-convex. Handling non-convex problems can be difficult as the relaxed problems may have multiple locally optimal solutions and cannot be guaranteed to be solved to global optimality. Thus it may be possible to remove a node wrongly due to the bounding rules, cutting off the global optimum of the original problem. The search is terminated when all nodes in the set $\mathcal{N}$ are either explored or removed. A Spatial Branch-and-Bound method was proposed by McCormick [57], using quadratic under-estimating and over-estimating functions for non-convex MINLPs.

#### 2.1.4.2 Node Selection

There are two important decisions left unspecified in the above description of the Branch-and-Bound method. One of them is which node we select to process next in Line 2. In this subsection, we briefly discuss some popular strategies. The selection method for branching nodes may significantly affect the performance of Branch-and-Bound.

A common strategy for node selection is *depth-first search* (or *depth-first search with backtracking*), which selects the deepest node in the search tree (or the last node

that is added to the active node set). It was proposed by Dakin [16] and Little et al. [55] primarily due to the limited memory capacity of computers at that time. The depth-first search can exhibit inferior performance if some optimal solutions are close to the root and some long paths do not yield an optimal solution. It may choose those long paths and explore many nodes before it explores a path that leads to an optimal solution. Another popular strategy is *best-first search* proposed by Land and Doig [48]. As the name suggests, the candidate node which has the smallest objective function value is selected for branching. Lawler and Wood [49] argued that it has the advantage of minimizing the total amount of computation, under the assumption that the set of new bounding problems is uniquely determined for any given problem. However, this strategy requires significant storage space to keep the set of active nodes. Theoretical comparison of breadth-first search, depth-first search, and best-first search was done by Ibaraki [40]. Breadth-first search is the opposite of depth-first search, which selects the first node that is added to the active set. All nodes at one level of the search tree are processed before any new node at a higher level. The above strategies reflect a trade-off between saving memory usage and keeping the number of explored nodes in the search tree as small as possible.

The best-first search has some proposed variants. The first variant is the best-projection method [25], which evaluates a node by considering both its objective function value and the degree of integer infeasibility in its current solution. The second variant is the best-estimate method [9], which estimates the objective function value of the integer-feasible solution that can be obtained by exploring the subtree that originates from a node, by calculating pseudocost values.

#### 2.1.4.3 Branching Variable Selection

The other important decision left open in Algorithm 1 is which variable we decide to branch on in Line 13. As explained, we select an integer-constrained variable $x_d$ for $d \in I$ that takes a fractional value and create two smaller subproblems by adding the constraint $x_d \leq \lfloor \bar{x_d} \rfloor$ and $x_d \geq \lceil \bar{x_d} \rceil$ respectively. The choice of branching variable can likewise have significant effects on the performance of the Branch-and-Bound method. Some common strategies are listed below.

The simple and widely-used strategy is known as the *most fractional branching*

[13], which chooses the variable with the largest integer violation to branch on, in other words, branch on a variable whose fractional part is closest to 0.5. *Pseudocost branching* was first developed by Bénichou et al. [9] for MILPs, which keeps the history of the results of past branching decisions for every variable and utilizes the historical information to predict the per unit change in the objective function value by averaging the increase in the objective function value for each candidate branching variable. Bénichou et al. chose to branch on the variable that is expected to yield a significant change in the objective value. One difficulty is that past branching experience is not available at the start of the algorithm. Thus, the estimates of the change in the objective function value for each variable should be initialized in some way. *Strong branching* was first proposed by Applegate et al. [5], solving difficult instances of the traveling salesman problem. The main idea is to execute a few iterations of tentative branching on each candidate variable before performing actual branching, and then select the candidate variable with the greatest change in the objective function value. One drawback is the computation time is high. Several variants of combining pseudocost branching and strong branching have been proposed. One of the most popular ones is *reliability branching* [1]. Strong branching was applied early during the tree search by Achterberg et al. [1], and then pseudocost branching was turned to once accurate estimates were obtained.

### 2.1.5 MISQP Algorithm

Mixed Integer Sequential Quadratic Programming (MISQP) is a trust-region Sequential Quadratic Programming (SQP) algorithm designed for mixed-integer nonlinear programming problems proposed by Exler et al. [21, 20]. MISQP features a FORTRAN interface and operates as a standalone library. MISQP has the capability to solve both convex and nonconvex MINLP problems, but Exler and Schittkowski [21] state that the algorithm relies on the assumption that integer variables have a *smooth* influence on the model functions, i.e., an increment or decrement of an integer variable by one leads to a small change of function values. The algorithm is stabilized by an SQP-based trust region method with second-order corrections proposed by Yuan [90]. It models the objective function by constructing a quadratic approximation in every iteration. The Hessian of the Lagrangian function is approximated by a quasi-Newton

update formula (BFGS). The generated mixed-integer quadratic programming subproblems must be solved by the solver MIQL of Lehmann et al. [50], which uses the solver QL of Schittkowski [72] for solving continuous quadratic programming.

The algorithm assumes that integrality constraints are not relaxable, in other words, the objective function and constraint functions cannot be evaluated when the values of variable $x_d, \forall d \in I$ are fractional. Thus, Exler and Schittkowski [20] approximate partial derivatives of objective and constraint functions with respect to integer variables by a difference formula at neighbored grid points. When dealing with variables that are subject to bounds, they apply a forward or backward difference formula. This procedure may require additional function evaluations to obtain gradient approximations. The standard difference formula is used to calculate partial derivatives with regard to continuous variables. It is important to note that MISQP can solve relaxable mixed-integer programming problems as well if users provide gradients in analytical form. The MISQP algorithm rounds integer variables with non-integer values before evaluating them using the objective and constraint functions at each iteration, in order to ensure integer feasibility. However, even in the case of convex MINLP problems, Exler and Schittkowski [21] point out that the algorithm cannot guarantee a globally optimal solution.

### 2.1.6 Further Techniques

Besides rounding off the continuous solution to the nearest feasible integer points, the mixed-integer optimization problem can be converted into a continuous problem using nonlinear optimization techniques. For example, Li [30] proposed a global optimization algorithm where a binary variable $x_d \in \{0, 1\}$ can be replaced by a continuous variable $x_d{}' \in [0, 1]$, by introducing the constraint $x_d{}'(1 - x_d{}') = 0, 0 \leq x_d{}' \leq 1$, which enforces variable $x_d{}'$ to take a value of either 0 or 1. Li [30] found this operation to be more straightforward than the Branch-and-Bound method and implicit enumeration method. The newly introduced constraint function is a non-convex nonlinear function. Similarly, Li and Chou [31] investigated a mixed-discrete nonlinear programming problem, where some of the variables are continuous and others may be integer or categorical. For a discrete variable $x_d \in \{k_1, k_2, \ldots, k_m\}$, a non-convex equality constraint $(x_d - k_1)(x_d - k_2) \ldots (x_d - k_m) = 0$ can be formed as

a penalty term in the objective function. They referred to this as discrete condition to model discrete variables. The solutions found for non-convex NLP problems are often local optima, and they are not guaranteed to be the globally optimal solution. Li and Chou [31] then utilized a multi-level single linkage technique [64] to obtain globally optimal solutions.

## 2.2 Nature-Inspired Heuristics for Mixed-Integer Optimization

Nature-inspired heuristics are a class of optimization algorithms inspired by the process of natural selection. They are considered stochastic black-box optimization algorithms as they do not require any prior knowledge or assumptions about the structure or characteristics of the objective function present in a given problem. This type of algorithms is most commonly used for solving continuous optimization problems. The effectiveness for black box optimization problems has been extensively researched in the literature and demonstrated in various practical applications. Over the last few decades, nature-inspired heuristics have been applied to mixed-integer optimization problems, including but not limited to Genetic Algorithms, Partial Swarm Optimization, Ant Colony Optimization Algorithm, Differential Evolution, and Evolution Strategies. These algorithms have been used to solve a wide range of mixed-integer optimization problems in various domains, such as engineering design, finance, aerospace, and others. In this section, we provide a brief overview and discussion of some nature-inspired heuristics that will be used in the following chapters.

### 2.2.1 MIDACO Algorithm

Mixed Integer Distributed Ant Colony Optimization (MIDACO) [77] is a global optimization algorithm for black-box mixed-integer nonlinear problems. The nature-inspired heuristics utilized in MIDACO builds upon the ant colony optimization metaheuristic for continuous optimization developed by Socha and Dorigo [82] and was extended to mixed-integer optimization by Schlüter et al. [74]. MIDACO applies the Oracle Penalty Method [75] for constraint handling. It performs automatic restarts to escape from local solutions and to improve the best solution found so far. The algorithm does not require the relaxation of integer variables, which means

objective and constraint functions are evaluated only at integer values for integer variables. Similar to other heuristic algorithms, MIDACO does not provide a guarantee of finding the global optimal solution.

An extensive comparison of MIDACO against eight sequential quadratic programming (SQP) based algorithms using a large set of test problems in the MINLPlib (www.minlplib.org) [14] was conducted by Schlüter et al. [76]. They also provide references to real-world applications of MIDACO in areas such as chemical engineering, aerospace and space engineering, and robotics. Their study has revealed that MIDACO outperforms the SQP-based algorithms in terms of its ability to locate globally optimal solutions. Schlüter et al. [76] indicate that although MIDACO requires significantly larger numbers of function evaluations, it can achieve a very competitive performance in terms of CPU time while maintaining a high chance of global optimality if the function evaluations are not computationally expensive. The parallelization method discussed in [77] is targeted towards distributing function evaluations, specifically for complex industrial applications where the function evaluation is computationally expensive. Schlüter [77] points out that a reasonable amount of CPU time can be utilized to solve problems with effective parallelization of the problem function evaluation. Schlüter and Munetomo provide further information regarding the numerical analysis conducted using this method in [78, 79]. It is important to note that the target scenario of MIDACO differs significantly from our study, as we consider objective function evaluations to be expensive and constraints to be explicit.

### 2.2.2 GAMBIT Algorithm

GAMBIT [67] is a model-based evolutionary algorithm designed for optimizing black-box mixed-binary problems. GAMBIT treats both the objective function and the constraint functions as black boxes and utilizes a penalty function method for constraint handling. The penalty function value in their implementation [67] is the square of the total constraint violation value multiplied by the number of previous generations. The algorithm integrates the model-building and sampling ability of the Linkage Tree Genetic Algorithm (LTGA) [87] and the Incremental Adapted Maximum-Likelihood Gaussian Model Iterated Density Estimation Evolutionary Algorithm (iAMaLGaM)

[11] for handling discrete and continuous variables respectively. A clustering mechanism is employed by Sadowski et al. [67] to split the problem population into some sub-populations at the beginning of each generation. Subsequently, the authors optimize the sub-populations by utilizing instances of an integrated model-based mechanism on each sub-population. The resulting offspring solutions from each instance are combined, thereby replacing the previous population. The process is then repeated from the beginning. With a lack of problem information, the algorithm detailed in [67] utilizes two mechanisms to estimate variable dependencies.

The appropriate selection of population size parameters in GAMBIT was noted as a problem by Sadowski et al. [68]. In general, empirical data are used in population-based algorithm research to determine the optimal or somewhat adjusted population size for a given problem. Similarly, Sadowski et al. [68] indicate that determining the optimal number of clusters for a given problem before execution is often infeasible. GAMBIT has been extended by introducing a multi-restart scheme [68], making the algorithm more practical by eliminating the need to specify parameters such as problem population and cluster size. Sadowski et al. [67, 68] show that GAMBIT has proven to be effective in solving mixed-integer problems involving binary variables, even in the presence of constraints. However, this algorithm is incompatible with general mixed-integer problems, where discrete variables can take on more than two potential values.

### 2.2.3 Evolution Strategies

Evolution strategies (ES) are stochastic black-box optimization algorithms that are widely used to solve unconstrained optimization problems. Similar to differential evolution, evolution strategies are a type of evolutionary algorithm. The basic idea of ES is to generate a population of candidate solutions through mutation, evaluate their objective function values, and use a selection mechanism to choose the best individuals to generate offspring for the next generation. The process may also involve the recombination of multiple candidate solutions through arithmetic averaging. One of the most straightforward examples of evolution strategies, the (1+1)-ES does not employ recombination. It generates a single offspring candidate solution from a single parent candidate solution in each generation. The one with the better objective

function value is considered the parental candidate solution for the next generation. Rudolph [65] introduced a mutation operator for unbounded integer search spaces and Bäck and Schütz [8] presented a mixed-integer evolution strategy aimed at optimizing optical multilayer systems. Li et al. [52] utilized specialized mutation operators for different types of decision variables in mixed-integer evolution strategies (MIES). Additionally, they provided theoretical analysis on the optimality of adapting step sizes and mutation rates.

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [34] is an evolutionary algorithm for solving black-box continuous optimization problems. It is acknowledged as a powerful algorithm for complex optimization problems involving ill-conditioning, multi-modality, discontinuity, and non-separability. The CMA-ES has several invariance properties, as described by Hansen et al. [36], such as the invariance of rotation and scaling of the coordinate system. These properties are important in black-box optimization since they assume no other knowledge of problem structures. All hyperparameters in the CMA-ES [35] have default values based on thorough experiments and theoretical research. Given the dimension $n$, all the default values can be computed. Consequently, the CMA-ES [34, 35] is considered a quasi-parameter-free algorithm that does not require users to tune the hyperparameters, such as the learning rate parameters of the covariance matrix adaptation.

Hansen [32] presents an extension of CMA-ES for mixed-integer optimization, including boundary handling. For this extension, integer variables with fractional values are rounded when evaluating the objective $f$ and constraint functions $g_i$. To prevent the CMA-ES from being trapped on plateaus, integer mutations are introduced for the integer variables, in addition to regular mutations [32]. Moreover, Hansen disregards integer variables with a small standard deviation in the global step-size update all together. As pointed out in [32], this algorithm variant does not work effectively for binary variables and $k$-ary integers in $k < 10$. Hamano et al. [29] have revisited Hansen's algorithm to enable it to solve problems with binary variables, which is referred to as CMA-ES with Margin (CMA-ESwM). They use a penalty function that is the sum of squared values of constraint violations divided by the dimension $n$. The experimental results in [29] indicate that the CMA-ESwM remains robust even as the number of dimensions increases, and can locate the optimal solution

with fewer function evaluations than the existing method proposed by Hansen [32]. Both algorithm variants are designed for unconstrained or bound-constrained mixed-integer problems and thus are not applicable for general constrained mixed-integer optimization problems.

Another variant of the CMA-ES algorithm has been proposed by Kumar et al. [45] to deal with the limitations of CMA-ES in handling general mixed-integer optimization problems, which is referred to as sCMAgES. To address the nonlinear constraints of the optimization problems, sCMAgES utilizes a constraint handling technique, namely $\epsilon$-constrained based ranking scheme. In addition, to handle nonlinear equality constraints, sCMAgES employs a gradient-based repair method, which is capable of transforming infeasible solutions to feasible ones, especially when the solutions are close to the boundary of the feasible region. The authors [45] also incorporate a restart scheme in their algorithm to ensure that the solutions converge to globally optimal values. Empirical results in [45] have shown that the sCMAgES algorithm has demonstrated good performance and ranked third among all submissions to the IEEE Congress on Evolutionary Computation (CEC) 2020 competition on non-convex constrained optimization problems. However, sCMAgES obtains the optimal point by simply rounding the result of continuous optimization. Consequently, sCMAgES is not specifically intended for mixed-integer optimization.

### 2.2.4 Differential Evolution

Differential Evolution (DE) is a population-based evolutionary algorithm. It was first introduced by Storn and Price [85, 86], which has been successfully applied to optimization problems including non-convex, non-linear, non-differentiable, and multimodal functions. DE is inspired by natural selection and simulates the evolutionary process of biological systems. We use evolutionary terminology to describe the optimization process, where the iterations are referred to as *generations*, the collection of candidate solutions is referred to as a *population*, and *parents* and *offspring* are denoted existing and newly generated candidate solutions, respectively. DE operates on a population of candidate solutions within the search space and updates the population by applying mutation, recombination/crossover, and selection operators. The mutation operator creates a new trial vector by adding a scaled difference between

two randomly selected vectors from the population to the base vector. In general, the DE mutation operator is represented as a string of the form DE/$x$/$y$/$z$ where $x$ specifies the way that the base vector is selected from the population, $y$ refers to the number of difference vectors involved, and $z$ represents the parent vector considered during the mutation process. The recombination/crossover operator then generates a new offspring vector by incorporating the trial vector with the parent vector. Finally, the selection operator chooses the best vector among the parent and offspring vectors for the next iteration of the search process. Once the new population is formed in the next generation, the iterative processes of mutation, crossover, and selection are performed continuously until the termination criteria are met.

During the last two decades, various modifications have been proposed for DE, which has enabled modified variants of DE to perform well in almost all CEC competitions held during this time. The original DE algorithm is designed for handling continuous optimization problems. A modified differential evolution (MDE) algorithm has been extended by Angira and Babu [4] to solve process synthesis and design problems. These problems are difficult non-convex optimization problems with continuous and discrete variables. Before evaluating objective functions and constraints, MDE applies a truncation operation for integer variables with fractional values. Constraints are handled by using a penalty function method. COLSHADE [28], which is another variant of DE, has been ranked second among all submissions in the IEEE CEC 2020 competition for constrained optimization. COLSHADE is a modified version of the L-SHADE [62] that incorporates additional features such as adaptive Lévy flights and a constraint handling technique with a dynamic tolerance mechanism for equality constraints. The adaptive Lévy flight-based mutation (called levy/1/bin) as introduced in [28] is used to increase exploration in the search space and prevent loss of diversity in the population. During an optimization process, the current-to-$p$best/1/bin mutation of L-SHADE is incorporated to reduce the population size linearly over generations to accelerate convergence and exploit the best solutions obtained. In the context of mixed-integer optimization problems, COLSHADE rounds the non-integer values of the integer variables before evaluating trial vectors and handling constraint violations.

## 2.3 Active-Set Evolution Strategies

Building on previous research by Arnold [6, 7], Spettel et al. [83] assume that constraints are given explicitly based on the taxonomy of Le Digabel and Wild [19] and implement the active-set evolution strategies for evolutionary constrained black-box optimization, which is referred to as (1+1)-AS-ES. A fundamental assumption of the (1+1)-AS-ES algorithm is that constraint function evaluations are significantly less expensive than objective function evaluations. A single iteration of (1+1)-AS-ES is provided in Algorithm 2, taken directly from Spettel et al. [83]. The (1+1)-AS-ES performs mutation and projects infeasible candidate solutions onto the feasible search space. The (1+1)-AS-ES algorithm employs Lagrange multipliers to determine whether a constraint is active. The algorithm uses a working set $W \subseteq \{1, 2, \ldots, l\}$ of indices of inequality constraints that are treated as equality constraints. $S(W)$ is the subset of the feasible region where all of the constraints in the working set are active. These active constraints reduce the feasible region and consequently reduce the degrees of freedom of the optimization problem. We denote $S(W)$ the reduced search space. If the offspring candidate solution improves the objective function value of the parental candidate solution, then it replaces the latter. In this case, any constraints that are active at the offspring candidate solution and not yet present in the working set $W$ are added to it. However, during the optimization process, it is possible for constraints that are not in the optimal active set to be added to the working set. To address this, the (1+1)-AS-ES releases a constraint temporarily from the working set and generates an offspring candidate solution where the constraint is not active. The process of releasing a constraint refers to converting the active inequality constraints back to inequalities. If the offspring candidate solution outperforms the parental one, the constraint is removed from the working set. Additionally, the update of step size $\sigma$ employs the implementation of the 1/5th-rule [41] based on the dimension of the reduced search space. According to the study conducted on the CEC 2006 test set [53], it appears that the (1+1)-AS-ES algorithm proposed by Spettel et al. [83] has higher success rates for some problems than the SQP and interior-point methods implemented in MATLAB's fmincon, with comparable numbers of objective function evaluations.

**Algorithm 2** Single Iteration of (1+1)-AS-ES

**Input:**

- candidate solution $\mathbf{x} \in \mathbb{R}^n$ and its objective function value $f_x = f(\mathbf{x})$
- step size parameter $\sigma \in \mathbb{R}$
- working set $W$
- release times $\mathbf{t} \in \mathbb{N}^l$ and iteration number $t \in \mathbb{N}$

Computer the effective dimension $n_{\text{eff}}$ of the reduced search space.
**if** $W \neq \emptyset$ and ($n_{\text{eff}} = 0$ or $U_{0,1} < 0.2$) **then**
    Let $k \in W$ be the index of a constraint with $t_k = \min_{l \in W} t_l$.
**else**
    Let $k = 0$.
**end if**
**repeat**
    Sample $\mathbf{z} \in \mathbb{R}^n$ from a standard normal distribution.
    Let $\mathbf{y}$ be the projection of $\mathbf{x} + \sigma\mathbf{z}$ onto $S(W \backslash \{k\})$.
**until** $\mathbf{y}$ is feasible and (either $k = 0$ or $g_k(\mathbf{y}) < 0$).
Obtain $f_y = f(\mathbf{y})$.
**if** $f_y < f_x$ **then**
    Let $\mathbf{x} \leftarrow \mathbf{y}$ and $f_x \leftarrow f_y$. ▷ success
    Add to $W$ any newly tight constraints.
    **if** $k = 0$ **then**
        Let $\sigma = \sigma e^{1/\sqrt{1+n_{\text{eff}}}}$.
    **else**
        Let $W \leftarrow W \backslash \{k\}$.
    **end if**
**else if** $k = 0$ **then**
    Let $\sigma = \sigma e^{-0.25/\sqrt{1+n_{\text{eff}}}}$. ▷ failure
**end if**
**if** $k > 0$ **then**
    Let $t_k \leftarrow t$.
**end if**
Let $t \leftarrow t + 1$.

## 2.4 Benchmarks

MINLPlib (www.minlplib.org) [14] is a well-known library of over 1500 mixed-integer and continuous nonlinear programming test problem instances. These problems arise from a broad range of industrial applications. It has been used in the past to develop and test mixed-integer programming algorithms and solvers. There are several researchers who have conducted the performance analysis and comparisons of their algorithms using MINLPlib. For example, Li et al. [52] tested their algorithm on artificial unconstrained benchmarks and presented six mixed-integer constrained problems, three of which are included in MINLPlib. To make the results of algorithm comparisons more easily understood, Schittkowski [73] and Schlüter and Munetomo [80] typically concentrate on smaller test sets. They utilized a set of 200 test instances from MINLPlib to evaluate and compare their algorithms. We select a small set of constrained optimization problems in MINLPlib [14] with published experimental results for comparable algorithms, choosing problems of diverse difficulties and varying dimensions to investigate algorithm performance.

In the field of Evolutionary Computation, several much smaller test problem sets have been selected by Deep et al [17] and Liao [54] that have similar characteristics with MINLPlib. The test function sets that are specified for the IEEE Congress on Evolutionary Computation (CEC) competitions on single-objective constrained real-parameter optimization are widely used as benchmark test collections for stochastic search algorithms. A set of test problems for constrained evolutionary optimization was assembled by Michalewicz and Schoenauer [58], building on earlier collections by Hock and Schittkowski [39], as well as Floudas and Pardalos [24]. Subsequently, Liang et al. [53] expanded this set to 24 test problems and used it as the basis for a competition held in connection with the 2006 IEEE Congress on Evolutionary Computation. A set of 57 optimization problems for the 2020 IEEE CEC competition with the goal of ensuring that the test instances reflect real-world challenges has been gathered by Kumar et al. [47]. The collection includes various types of optimization problems, including mixed-integer problems. However, since the original references of these problems date back several decades, errors have been introduced by some authors and then copied by others. Bound constraints were imposed by Kumar et al. [47] on certain problems, resulting in changes to the properties of the problems, which

can significantly affect the optimization problem. These constraints can effectively prevent algorithms from converging to local optima, which have been previously identified as major obstacles in locating globally optimal solutions. On the other hand, some constraints are active at optimal solutions despite lacking physical significance.

# Chapter 3

# Algorithm

Several challenges arise when employing evolution strategies to solve mixed-integer optimization. The presence of integer variables alters the nature of the search space. Additionally, the application of traditional operators in evolution strategies, such as mutation and recombination, may generate infeasible solutions that violate the constraints. Active-set evolution strategy [83] is designed for solving constrained continuous optimization problems with both equality constraints and inequality constraints. Branch-and-Bound method [16] is one of the most commonly used method for handling mixed-integer programming problems. This chapter presents an algorithm for evolutionary mixed-integer optimization with explicit constraints, which we refer to as $\text{AS}_{\text{mixInt}}$-ES, incorporating (1+1)-AS-ES as described by Spettel et al. [83] with a branching technique to handle integrality constraints. However, the black-box nature of the objective function and the stochastic nature of the evolution strategy make it difficult to determine useful bounds for the optimization problem. Thus, we introduce heuristics to determine when to split a problem into subproblems and subsequently eliminate subproblems from further consideration, rather than pruning the search tree simply based on the bounds. The remainder of this chapter describes the proposed algorithm in detail.

## 3.1 Evolutionary Mixed-Integer Optimization with Explicit Constraints

A flowchart illustrating the process of the proposed algorithm is provided in Figure 3.1. The $\text{AS}_{\text{mixInt}}$-ES keeps a set $\mathcal{N}$ of active nodes that need to be explored further. Each node stores a candidate solution that satisfies its bound constraints as well as other linear and nonlinear constraints, but that may violate some integrality constraints. In addition, each node contains a step size, a working set of indices of inequality constraints that are treated as equalities, lower bounds, and upper bounds.

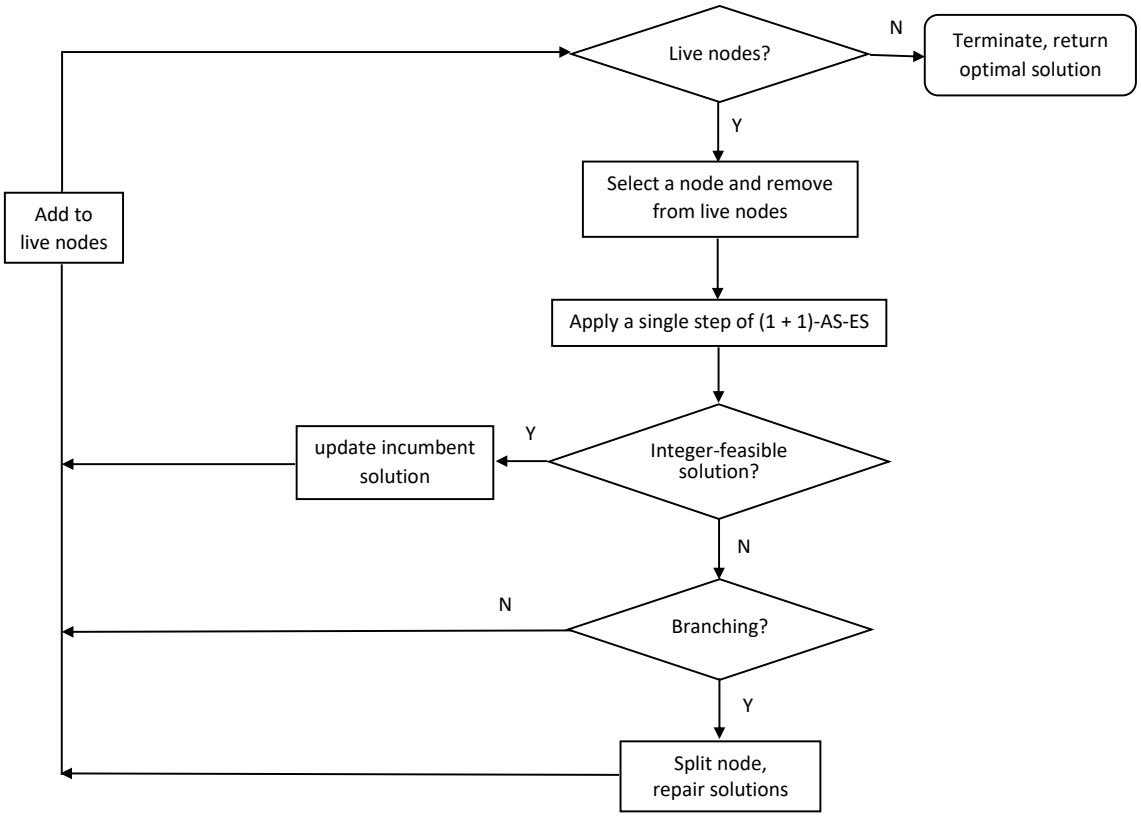


Figure 3.1: Flowchart of $\mathrm{AS}_{\mathrm{mixInt}}$-ES.

In each iteration of the algorithm, an active node is selected randomly, but the selection process is weighted to choose nodes whose candidate solutions have the minimum objective function values. The algorithm applies a single (1 + 1)-AS-ES step [83] to the selected node. If the integer-feasible solution improves on the incumbent solution, then the incumbent solution value is replaced by the former. If there exists an integer-constrained variable which takes a non-integer value, and the step size associated with the node drops below a threshold, then the node is split into two new nodes by selecting the integer variable and replacing the old node with the new ones. The two new nodes are updated bounds to the old node that make its candidate solution infeasible, while still ensuring that any integer-feasible solution is feasible for one of the new nodes. Only a node that generates a candidate solution satisfying the updated bound constraints can be added to the set $\mathcal{N}$. We repeat the whole procedure

until there are no live nodes left in $\mathcal{N}$ or the maximum objective function evaluation limit is reached.

The state of the algorithm is given by a set $\mathcal{N}$ of active nodes, the best integer-feasible candidate solution located so far along with its objective function value, and the current iteration number $t$. A node is a septuple $(\mathbf{x}, f_{\mathbf{x}}, \sigma, W, \mathbf{t}, \mathbf{l}, \mathbf{u})$ comprising a candidate solution $\mathbf{x} \in \mathbb{R}^n$ that satisfies all equality and inequality constraints but that may violate integrality constraints, its objective function value $f_{\mathbf{x}} \in \mathbb{R}$, step size parameter $\sigma > 0$, a working set $W \subseteq \{1, 2, \ldots, l\}$ of indices of inequality constraints that are considered to be equality constraints, an array $\mathbf{t} = (t_1, t_2, \ldots, t_l)$ that for each inequality constraint stores the iteration number when it was most recently considered for removal from the working set, lower bounds $\mathbf{l} \in \mathbb{R}^n$, and upper bounds $\mathbf{u} \in \mathbb{R}^n$. The initialization of $\mathbf{x}$ and $\sigma$ depends on the particular problem. When the initial candidate solution is infeasible, it is projected onto the feasible region using the method outlined by Spettel et al. [83]. The set $\mathcal{N}$ of active nodes is initialized to contain the root node. $W$ is initialized to the active set $A(\mathbf{x})$. $\mathbf{t}$ is initialized to hold all zeros. The incumbent solution $\mathbf{x}^*$ is initialized to have an objective function value $f^*$ of infinity.

A single iteration of the $\text{AS}_{\text{mixInt}}$-ES is summarized in Algorithm 3 and discussed in what follows. Lines 1 through 6 determine which node is selected in $\mathcal{N}$, i.e. which candidate subproblem should be processed next. A candidate node $\mathcal{Q}$ is selected with equal probability by either choosing the node with the lowest objective function value $f_{\mathbf{x}}$, or by randomly selecting a node from the set $\mathcal{N}$. Node $\mathcal{Q}$ is then temporarily removed from the set of active nodes in Line 7 and applied a single step of the (1 + 1)-AS-ES as described by Spettel et al. [83] in Line 8. The (1 + 1)-AS-ES performs mutation, projects the resulting point onto the feasible region, ignoring the integrality constraints, and then updates the candidate solution $\mathbf{x}$, its corresponding objective function value $f_{\mathbf{x}}$, the working set to account for any newly active or inactive inequality constraints. The step size is updated based on the 1/5th-rule. Line 9 computes the maximum integrality constraint violation $\Delta$. Lines 10 through 29 determine whether or not node $\mathcal{Q}$ from $\mathcal{N}$ is to be considered for permanent removal. As described in Lines 10 through 12, if an integer-feasible solution is better than the current incumbent solution, then one would update the incumbent solution and its

**Algorithm 3** Single Iteration of $\text{AS}_{\text{mixInt}}$-ES

**Input:**

- set $\mathcal{N}$ of active nodes
- incumbent solution $\mathbf{x}^*$, objective function value $f^* = f(\mathbf{x}^*)$
- iteration number $t \in \mathbb{N}$

1: Sample $p$ uniformly at random in $[0, 1)$.
2: **if** $p < 0.5$ **then**
3: Select node $\mathcal{Q} \in \mathcal{N}$ with the minimal objective function value.
4: **else**
5: Select node $\mathcal{Q} \in \mathcal{N}$ uniformly at random.
6: **end if**
7: Let $\mathcal{N} \leftarrow \mathcal{N} \backslash \{\mathcal{Q}\}$.
8: Let $(\mathbf{x}, f_{\mathbf{x}}, \sigma, W, \mathbf{t}, \mathbf{l}, \mathbf{u}) \leftarrow (1{+}1)\text{-AS-ES}_{\text{step}}(\mathcal{Q}, t)$.
9: Let $\Delta \leftarrow \|\mathbf{x}_I - \text{round}(\mathbf{x}_I)\|_\infty$ .
10: **if** $\Delta < \delta$ and $f_{\mathbf{x}} < f^*$ **then**
11: Let $\mathbf{x}^* \leftarrow \mathbf{x}$ and $f^* \leftarrow f_{\mathbf{x}}$. ▷ Update incumbent solution
12: **end if**
13: **if** $\sigma < 10^{-8}$ **then**
14: Jump to Line 30.
15: **end if**
16: Compute the effective dimension $n_{\text{eff}}$ of the reduced search space at $\mathbf{x}$.
17: **if** $\Delta > \delta$ and ($n_{\text{eff}} = 0$ or $\sigma < 0.1$) **then**
18: Branch on $x_d$ with the maximum integrality constraint violation.
19: Let $\mathbf{l}' \leftarrow \langle l_1, \ldots, l_{d-1}, \lceil x_d \rceil, l_{d+1}, \ldots, l_n \rangle$.
20: **if** $(\mathbf{y}, W') \leftarrow \text{repair}(\mathbf{x}, W, \mathbf{l}', \mathbf{u})$ is successful **then**
21: Let $\mathcal{N} \leftarrow \mathcal{N} \cup \{(\mathbf{y}, f_{\mathbf{y}}, \sigma, W', \mathbf{t}, \mathbf{l}', \mathbf{u})\}$.
22: **end if**
23: Let $\mathbf{u}' \leftarrow \langle u_1, \ldots, u_{d-1}, \lfloor x_d \rfloor, u_{d+1}, \ldots, u_n \rangle$.
24: **if** $(\mathbf{y}, W') \leftarrow \text{repair}(\mathbf{x}, W, \mathbf{l}, \mathbf{u}')$ is successful **then**
25: Let $\mathcal{N} \leftarrow \mathcal{N} \cup \{(\mathbf{y}, f_{\mathbf{y}}, \sigma, W', \mathbf{t}, \mathbf{l}, \mathbf{u}')\}$.
26: **end if**
27: **else**
28: Let $\mathcal{N} \leftarrow \mathcal{N} \cup \{(\mathbf{x}, f_{\mathbf{x}}, \sigma, W, \mathbf{t}, \mathbf{l}, \mathbf{u})\}$.
29: **end if**
30: Let $t \leftarrow t + 1$.

corresponding objective function value. If the step size falls below $10^{-8}$ as a result of the (1+1)-AS-ES step, the remaining steps in the iteration are skipped, and the node is permanently removed from consideration. Line 16 computes the effective dimension $n_{\text{eff}}$ of the reduced search space by subtracting the dimension of the span of the normal vectors of the equality constraints and the inequality constraints in the working set evaluated at $\mathbf{x}$ from the dimension $n$. In Line 17, we detect any violations of the integrality constraints, i.e. an integer variable $x_d \in \mathbb{Z}$ takes a non-integer value for some $d \in I$. If any violations exist, node $\mathcal{Q}$ is split if the effective dimension $n_{\text{eff}}$ of the reduced search space is zero or the step size drops below 0.1. If neither of these conditions are met, node $\mathcal{Q}$ is not split and is instead added back to the set $\mathcal{N}$ in Line 28. Lines 18 through 26 implement the branching procedure for the selected node $\mathcal{Q}$. Line 18 selects $x_d$ with the maximum integrality constraint violation as the branching variable. Based on the idea of branching operation [16], node $\mathcal{Q}$ is split into two new nodes. The new nodes should inherit all information of parent node $\mathcal{Q}$. Two bounds are updated correspondingly in Lines 19 and 23, one with lower bound for variable $x_d$ to $\lceil x_d \rceil$, and the other with upper bound for variable $x_d$ to $\lfloor x_d \rfloor$, where $\lceil x_d \rceil$ and $\lfloor x_d \rfloor$ denote rounding up and down the value of $x_d$ for $d \in I$ respectively. Clearly, solution $\mathbf{x}$ of parent node $\mathcal{Q}$ is infeasible for either of the new nodes. Lines 20 and 24 involve a repair operation that projects the candidate solution $\mathbf{x}$ onto the feasible regions of the corresponding node, using the working set $W$. Projection is implemented as detailed by Spettel et al. [83]. If no feasible solution is found, up to forty points are uniformly generated at random within the bound constraints of the node, and then projected with an empty working set. The first feasible solution that is obtained, along with its working set, is returned. The associated node is added to $\mathcal{N}$ in Lines 21 and 25. If no feasible solution is located, the node is not added in the set $\mathcal{N}$ of active nodes, and is therefore terminated. Line 30 updates the iteration number.

It should be noted that each iteration of the algorithm involves between one and three objective function evaluations. This includes one evaluation executed by the (1+1)-AS-ES in Line 8, and up to two additional evaluations if a node is split and feasible solutions are obtained for the newly created nodes in Lines 20 and 24. Additionally, projection onto the feasible region frequently requires a significantly

larger number of constraint function evaluations.

It is worth noting that the node selection process in Line 8 comes with a trade-off. Preliminary experiments demonstrated that it was not possible to reliably estimate the potential of a node that had undergone a limited number of AS-ES steps. Always selecting a node with the smallest objective function value is likely to result in the overlooking of the node with a larger objective function, which would lead to the optimal solution. Generally, advancing a node with a larger objective value in a small number of objective function evaluations may lead to a better solution in the end. Thus, we introduce a probability-based approach to selecting nodes, which takes into account either the node's objective function value or random sampling. At the branching stage in Line 17, we set a small threshold for the distance from an integer and the step size of a candidate node, which prevents early and frequent branching and allows integer variables to converge to an integral solution instead. Furthermore, $n_{\text{eff}} = 0$ effectively prevents the algorithm from exploring a node indefinitely for which no search directions are available.

## 3.2 Performance of $\text{AS}_{\text{mixInt}}$-ES

This section aims to investigate the impact of different problem characteristics on the performance of $\text{AS}_{\text{mixInt}}$-ES, including the number of integer variables, the number of constraints, and the scaling factors of constraint functions. To analyze their influence, we first consider $n$-dimensional constrained sphere functions with different numbers of constraints. The objective function is

$$\text{minimize}\, f(\mathbf{x}) = (\mathbf{x} - \mathbf{c})^T(\mathbf{x} - \mathbf{c}) \tag{3.1}$$

where the elements of $\mathbf{c}$ are uniformly distributed numbers between 0 and 1. The linear inequalities constraints are of the form

$$g_i(\mathbf{x}) = \mathbf{A}^T\mathbf{x} \leq \mathbf{b}$$

where $\mathbf{A}$ is an $l \times n$ matrix and $\mathbf{b}$ is an $l$-dimensional column vector. Each element of matrix $\mathbf{A}$ and vector $\mathbf{b}$ are independently sampled from a normal distribution with

mean zero and unit variance. To study the effect of both the number of constraints and integer variables, we randomly generated a set of problem instances with $n = 10$ and $l \in \{1, 5, 9\}$. The number of integer variables of test problems ranges from zero to $n$. The problem instances used in this study do not have any bound constraints on their continuous and integer variables and are characterized by convexity and unimodality. Despite their relative simplicity, they allow us to explore how the algorithm performs under different numbers of constraints and integrality constraints. Examining the algorithm's behaviour on such problems can provide insights into its performance on more complex objective functions. In practice, if an optimization algorithm fails on a simple objective function, it is not likely to work well on more complicated ones.

To observe the scaling behaviour of the algorithm on linearly constrained spheres, two additional sets of sphere functions are considered, which have increased (referred to as large $A$) and decreased (referred to as small $A$) scaling factors of random constraint functions. Each component of matrix $\mathbf{A}$ is randomly sampled from a normal distribution with a mean of 0 and a variance of $10^4$ and $10^{-2}$ respectively in large $A$ and small $A$ problems. The modifications made to the test problems involve changing only the relevant scaling factor while leaving the problem formulations themselves unchanged. In order to ensure that problem dimensions do not impact algorithm performance, we keep the size of the test problems constant. The optimal solutions for large $A$ and middle $A$ spheres are expected to be concentrated in a narrow range near the origin; however, small $A$ values may cause the optimal solutions to be located farther away.

In addition to analyzing the algorithm's performance, we also compare it with the Mixed Integer Distributed Ant Colony Optimization (MIDACO) by Schlüter [77], which was described in Section 2.2.1. MIDACO extends the ant colony optimization algorithm and incorporates the oracle penalty function for constrained handling. Schlüter [77] demonstrated that MIDACO is competitive with several established MINLP software on comprehensive MINLP benchmark sets in terms of the number of globally optimal solutions found. We have conducted 101 runs of both $\text{AS}_{\text{mixInt}}$-ES and MIDACO. A problem is considered solved successfully if a solution with an objective function value $f(\mathbf{x}) < f^* + \epsilon|f^*|$ is obtained, where $f^*$ is the optimal value of the problem and it is feasible within the tolerance of constraints $\delta > 0$, which

indicates the maximum degree of constraint violation. Parameter $\epsilon > 0$ is referred to as the target accuracy. If no solution meeting the termination criterion has been obtained after a maximal budget of objective function evaluations, then the run is declared unsuccessful.

The initialization of starting points involves uniform sampling from the interval $[-2, 2]^n$, with an initial step size of $\sigma = 1$ and constraint tolerance of $\delta = 10^{-8}$. Runs are terminated after either a solution with an objective function value within a factor of $1 + 10^{-8}$ of the optimal value is found, or $10^5$ iterations have been spent without finding a solution that satisfies the termination criterion. Runtimes for successful runs are measured as the number of objective function evaluations required to solve the problem.

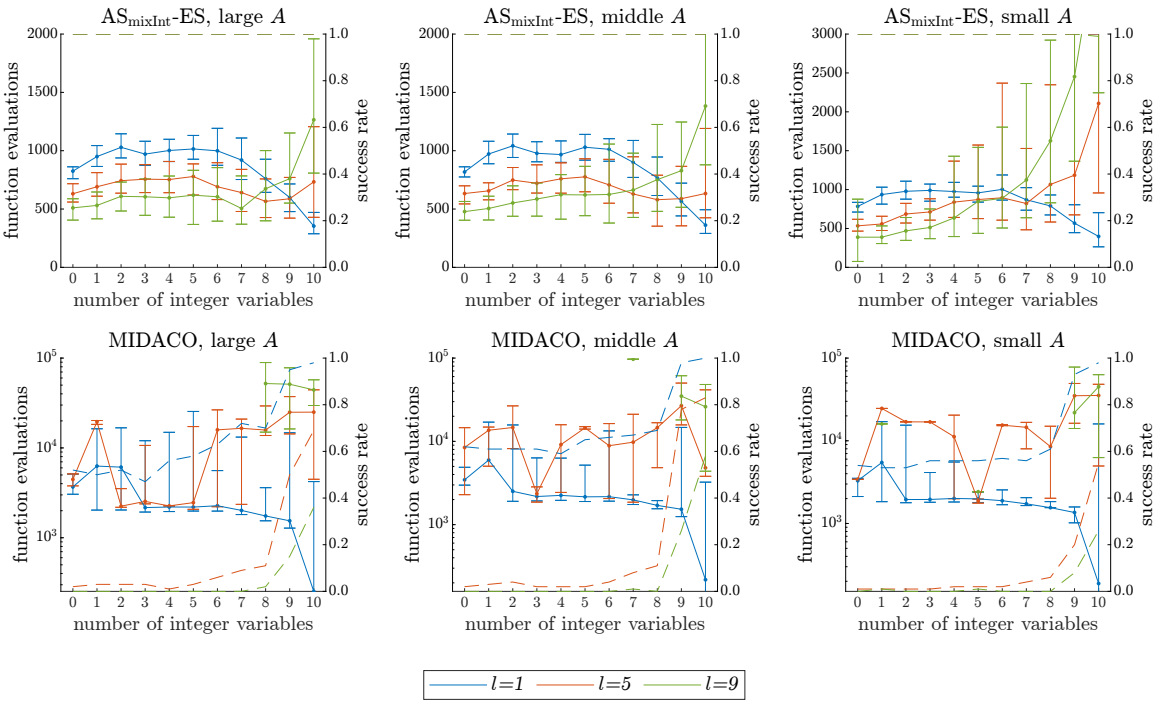


Figure 3.2: Performance on sphere functions with randomly linear constraints $l \in \{1, 5, 9\}$ for different coefficients in the constraint matrix $\mathbf{A}$ in 101 runs. The top row presents results for the AS$_{\text{mixInt}}$-ES and the bottom row presents results for MIDACO. The columns from left to right show results for large $A$, middle $A$, and small $A$ spheres. Left y-axis: the solid lines connect the median numbers of objective function evaluations required to locate the optimal solutions within the required accuracy; the error bars encompass the range of 25th and 75th percentiles observed for each algorithm. Right y-axis: the dashed lines represent success rates.

The median number of objective function evaluations required to achieve a termination accuracy $\epsilon = 10^{-8}$ and success rates for the $\text{AS}_{\text{mixInt}}$-ES and MIDACO algorithms against the number of integer variables is shown in Figure 3.2. The solid lines connect the median values over successful runs. The error bars illustrate the range from the 25th to 75th percentile values observed across successful runs. The dashed lines represent success rates. The top row of Figure 3.2 reveals a similar trend among different $A$ spheres with $l = 1$. However, with $l = 9$, the number of median objective function evaluations required has an increasing trend as the number of integer variables grows. It can be observed that for $l = 5$, as the number of integer variables increases, there is a similar trend in the behaviour of large and middle $A$ spheres. However, for small $A$ spheres, there is a rapid increase in the number of required function evaluations. It is apparent that the number of linear constraints has an effect on the runtime. The $\text{AS}_{\text{mixInt}}$-ES exhibits an advantage in handling problems that possess a higher number of linear constraints and a lower number of integer variables. This could be contributed to the fact that more linear constraints lead to decreased unconstrained dimensions, and the active-set ES can benefit from the reduced search space. However, as the integer variables begin to exceed half of the total variables, this benefit gradually diminishes. Moreover, in terms of different scaling factors $\mathbf{A}$, the algorithm's performance on large and middle $A$ problems with varying numbers of integrality constraints is similar, solving all problem instances in all of the runs. However, compared to large and middle $A$ spheres, the $\text{AS}_{\text{mixInt}}$-ES requires a higher number of objective function evaluations on the small $A$ problems with $l = 5$ and $l = 9$.

As expected, Figure 3.3 indicates that on spheres with $l = 5$ and $l = 9$, the $\text{AS}_{\text{mixInt}}$-ES requires visiting a larger number of nodes during the branching stage as the number of integrality constraints increases.

The results obtained with MIDACO indicate a different trend as illustrated in the bottom row of Figure 3.2, in contrast to the performance of $\text{AS}_{\text{mixInt}}$-ES. The missing data in the figure reflect that MIDACO fails to achieve the termination accuracy on the corresponding problems in any of the runs. Notably, MIDACO performs well when the number of integer variables exceeds half of the total variables and achieves a success rate of 100 percent on pure integer problems with $l = 1$, where all variables

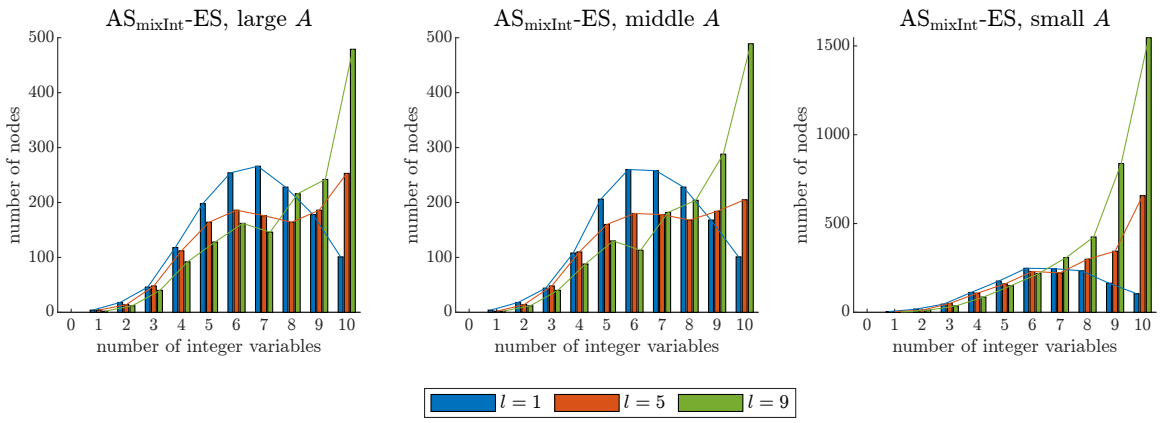


Figure 3.3: Median numbers of nodes required to solve sphere functions with randomly linear constraints $l \in \{1, 5, 9\}$ for different coefficients in the constraint matrix $\mathbf{A}$ in 101 runs. The columns from left to right show results for large $A$, middle $A$, and small $A$ spheres.

of problem instances take on integer values. However, it is observed that MIDACO is less effective in problems with more linear constraints, regardless of the number of integer variables and the size of the scaling factor $\mathbf{A}$. The comparison of the results depicted from top to bottom of Figure 3.2 demonstrates that MIDACO requires a higher median number of objective function evaluations and has lower success rates than the AS$_{\text{mixInt}}$-ES for all test instances. The AS$_{\text{mixInt}}$-ES locates globally optimal solutions in 99 percent of runs for target accuracy $\epsilon = 10^{-8}$, while MIDACO only achieves a corresponding rate of 28 percent. For MIDACO, the corresponding rates on $l = 1$, $l = 5$, and $l = 9$ are 66 percent, 14 percent, and 5 percent respectively; the rates on middle $A$, large $A$, and small $A$ spheres are 32 percent, 28 percent, and 25 percent respectively. It can be observed that the capability of MIDACO to solve the problem decreases with an increase in the number of constraints and an increase in the range of optimal solutions for integer variables. It is important to keep in mind that MIDACO considers the constraint functions as black boxes while the AS$_{\text{mixInt}}$-ES assumes knowledge of constraint functions.

To have a better insight into the overall performance of two algorithms, we present empirical cumulative distribution function (ECDF) plots as described by Hansen et al. [33] in Figure 3.4, which are known as data profiles [60]. We select twenty logarithmically uniformly distributed values between $f^* + 10^{-8}\,|f^*|$ and $f^* + 10^{2}\,|f^*|$ as targets for each problem, where $f^*$ is the optimal solution value of the problem. For

each algorithm, the plots show the percentage of test problems that reached objective function values within the required accuracy against running time. A measure of its performance is the ratio of the number of targets achieved from the set by integer-feasible solutions. We limit each run to a maximum of $10^5$ objective function evaluations for both algorithms. In Figure 3.4, it is evident from the first to the third row that the $\text{AS}_{\text{mixInt}}$-ES successfully meets all targets in all runs, with an expected increase in the proportion of met targets along with runtimes. It can be observed that small $A$ problems require more time to achieve the targets than problems with larger scaling values $\mathbf{A}$. Although all test spheres are defined for any $\mathbf{x} \in \mathbb{R}^n$, middle and large $A$ spheres have their optimal solutions for the integer variables fall between -3 and 3, whereas for small $A$ spheres, optimal solutions are found in a broader range of integer variables within $[-5, 5]$, $[-10, 10]$, and $[-20, 20]$ for $l = 1$, $l = 5$, and $l = 9$ respectively. This difference in optimal solutions requires more computational effort in the branching step of the $\text{AS}_{\text{mixInt}}$-ES algorithm as illustrated in Figure 3.3, leading to much longer runtimes for solving small $A$ spheres.

MIDACO reaches less than half of the targets for the $n_{\text{int}} \in \{2, 5\}$ and $l \in \{5, 9\}$ combination cases, while it achieves a higher percentage of targets (around 89 percent, 86 percent, and 84 percent respectively) for middle $A$, large $A$, and small $A$ spheres with $n_{\text{int}} = 8$ and $l = 1$. MIDACO can be observed to reach a small number of targets than the $\text{AS}_{\text{mixInt}}$-ES does at the beginning. This may be because MIDACO only evaluates objective and constraint functions for integer variables at integer points, while the $\text{AS}_{\text{mixInt}}$-ES does not enforce the integrality requirement for integer variables when evaluating functions. After that, the $\text{AS}_{\text{mixInt}}$-ES appears to converge significantly faster than MIDACO. The $\text{AS}_{\text{mixInt}}$-ES eventually achieves all of the targets with different numbers of linear constraints $l \in \{1, 5, 9\}$. MIDACO achieves 52 percent of targets on $n_{\text{int}} = 8$, 46 percent on $n_{\text{int}} = 5$, 43 percent on $n_{\text{int}} = 2$. We can see that the efficiency of MIDACO in solving problems increases with decreasing number of continuous variables.

The objective function definition for the mixed-integer problems discussed in this section is relatively straightforward, but the results show that the number of integer variables and constraint functions has a significant impact on the algorithm's running time. The effects can contribute to the complexity of the problems to some extent.

The $\text{AS}_{\text{mixInt}}$-ES can deal with such mixed-integer problems using an integrality constraint handling approach. Additionally, we observe that different scaling values for the coefficient matrix $\mathbf{A}$ have a strong effect on the algorithm's performance, with the range of optimal solutions on integer variables being a key factor. As the number of integrality constraints increases, small $A$ spheres become more challenging for $\text{AS}_{\text{mixInt}}$-ES, reducing its efficiency. In the next chapter, we will evaluate the computational performance of $\text{AS}_{\text{mixInt}}$-ES and compare it with other related algorithms on relatively complex problems.

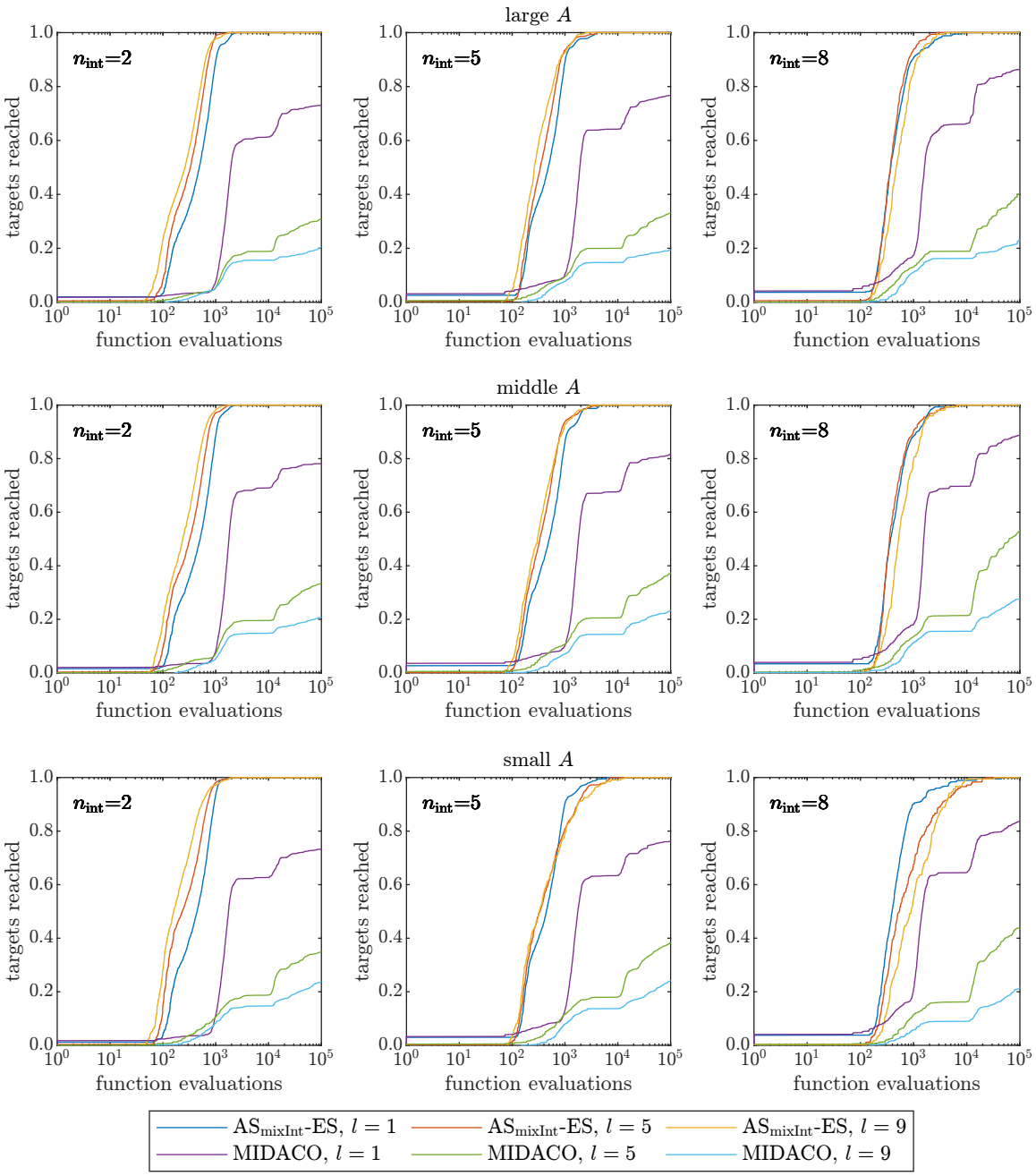


Figure 3.4: Empirical cumulative running time distributions for AS$_{\text{mixInt}}$-ES and MIDACO on linearly constrained spheres for $l \in \{1, 5, 9\}$ with different coefficients in the constraint matrix $\mathbf{A}$. The rows from top to bottom show results for the AS$_{\text{mixInt}}$-ES and the MIDACO on large $A$, middle $A$, and small $A$ spheres. The columns from left to right present results with the number of integer variables $n_{\text{int}} \in \{2, 5, 8\}$ for two algorithms.

# Chapter 4

# Experiments

The presence of both discrete and continuous variables introduces new optimization challenges. The main focus here is to illustrate the applicability and efficiency of the proposed algorithm for mixed-integer optimization. This chapter evaluates the performance of the proposed $\text{AS}_{\text{mixInt}}$-ES on two sets of test problems. In the first experiment, we consider several of the algorithms surveyed in Chapter 2. These algorithms' implementations are not accessible. They have been evaluated on problems included in the MINLPlib set, and we compare the performance data of $\text{AS}_{\text{mixInt}}$-ES to those of other algorithms that have been reported in the literature. The second experiment utilizes a subset of non-convex constrained problems collected by Kumar et al. [47] that we refer to as the CEC 2020 Benchmark. Detailed information about these problems is given in Appendix A. We present numerical results of the proposed $\text{AS}_{\text{mixInt}}$-ES and compare them with several algorithms submitted to the CEC 2020 competition. A further investigation that we would like to conduct is to confirm whether the (1+1)-AS-ES algorithm's strong performance reported by Spettel et al. [83] on the CEC 2006 test problem set, which was a key component in the development of the $\text{AS}_{\text{mixInt}}$-ES, extends to real-world problems.

The performance of the $\text{AS}_{\text{mixInt}}$-ES on a set of MINLPlib test problems and compares the results to one deterministic algorithm (named MISQP) and two stochastic algorithms (named MIDACO and GAMBIT respectively) is investigated in Section 4.1. MIDACO builds upon the ant colony optimization and GAMBIT is a model-based evolutionary algorithm. In Section 4.2, extensive experiments are carried out on a set of the CEC 2020 benchmark to evaluate the performance of the proposed algorithm. Section 4.2.1 highlights several comparable algorithms that are used on constrained optimization problems, Section 4.2.2 describes the test environment, and Section 4.2.3 presents experimental results.

## 4.1 Numerical Tests on MINLPlib

### 4.1.1 Algorithms

We compare the performance of the $AS_{mixInt}$-ES with three other algorithms:

- MISQP by Exler and Schittkowski [21], as described in Section 2.1.5. MISQP makes use of pre-defined initial starting points in the source code provided by Schittkowski [73]. Default tolerances and parameter settings are applied with target accuracy $\epsilon = 10^{-6}$ and constraint tolerance $\delta = 10^{-6}$. As stated in [73], the maximum number of iterations is set to 100, and the maximum number of iterations without improvements is set to 10.

- MIDACO Version 6.0 by Schlüter [77], as outlined in Section 2.2.1, conducting 21 independent runs for each problem instance. We use the same starting points as Schittkowski did in [73] for all problem instances. Additionally, we set the constraint tolerance to $\delta = 10^{-6}$, matching the tolerances used in [73]. Runs are terminated when either a solution with an objective function value within a factor of $1 + 10^{-6}$ of the optimal value is found, or $10^5$ iterations have been spent without finding a solution that satisfies the termination criterion. We do not consider the parallelization of MIDACO in the following experiments.

- GAMBIT, which extends MIES [52] to solve constrained MINLP by exploiting a dynamic penalty method as a means of constraint handling and the clustering mechanism, as outlined in Section 2.2.2. Identical starting points as those in MIES [52] are used. The maximum population size is set to 2500 [67]. Success criteria are declared if the optimal value is achieved with a precision of $10^{-5}$ and a constraint tolerance of $10^{-10}$. The optimum is reached in at least 19 out of 20 runs.

### 4.1.2 Test Environment

Three problems in MINLPlib [59] have been used to evaluate their algorithms by Sadowski et al. [67]. The integer variables of these three problems only involve a value of zero or one. We refer to this type of problem as a mixed-binary problem. To evaluate the generality of the proposed algorithm, we select additional test problems

from [59] in several different fields. These problems contain binary and integer-valued variables, thereby adding to the diversity of the test set. These are all mixed-integer problems with various types of objective functions, including linear and non-linear, as well as different types of constraints, such as linear inequalities, non-linear inequalities, linear equalities, and non-linear equalities. The problem dimensions range from 2 to 17, with the number of integer variables ranging from 1 to 12 and up to 13 constraints. The first class of problems is composed of seven mixed-binary problems, such as problems EX1223A, EX1226, and ST_E15. The other seven test problems are classified as general mixed-integer problems. It is necessary to collect a wide range of test problems in a comparative study. The dimension of a problem partly determines its overall complexity. Moreover, the complexity of a problem is also influenced by factors such as the types of integer variables, the number of constraints and integer variables, as well as the presence of inequality and equality constraints.

A summary of the selected problems is listed in Table 4.1. The first column gives the name of the problems, followed by four columns that detail the dimensions, number of integer variables, inequalities, and equalities constraints. The types of objective functions are given in column six, followed by the types of integer variables and their ranges in the last column. All constraint functions include general nonlinear constraints. These problems are collected from a range of real-world applications. Note that while most of the test problems have both inequality and equality constraints, some exceptions exist. For instance, problem WINDFAC does not have inequality constraints, and problems EX1223A, NVS08, and PROB10 do not have equality constraints. Some problems in [59] do not have lower or upper bound constraints defined for some continuous variables, while other problems have both bound constraints. However, Schittkowski [73] provides lower and upper bounds for all variables in all test problems. Therefore, we use the same bounds as in [73] for consistency.

Initial starting points of the proposed algorithm $\text{AS}_{\text{mixInt}}$-ES are the same as those provided in [73]. The step size parameter $\sigma$ is initialized to one-fifth of the minimum extent of the interval width for all variables. All test problems are provided with globally optimal objective function values $f^*$, which are the same as the known ones reported in the MINLPlib [59]. In the following experiments, we conduct 21 independent runs of $\text{AS}_{\text{mixInt}}$-ES and MIDACO for each test problem considered and use the

| Problem | $n$ | $n_{\text{int}}$ | $l$ | $m$ | Type of objective function | Type of integer variables | Ranges for integer variables |
|---|---|---|---|---|---|---|---|
| EX1223A | 7 | 4 | 9 | 0 | non-linear | B | $\{0,1\}^4$ |
| EX1225 | 8 | 6 | 8 | 2 | linear | B | $\{0,1\}^6$ |
| EX1226 | 5 | 3 | 4 | 1 | linear | B | $\{0,1\}^3$ |
| GKOCIS | 11 | 3 | 3 | 5 | linear | B | $\{0,1\}^3$ |
| PROCSEL | 10 | 3 | 3 | 4 | linear | B | $\{0,1\}^3$ |
| ST_E15 | 5 | 3 | 3 | 2 | linear | B | $\{0,1\}^3$ |
| SYNTHES2 | 11 | 5 | 13 | 1 | non-linear | B | $\{0,1\}^5$ |
| NVS01 | 3 | 2 | 2 | 1 | non-linear | I | $[0,200]^2$ |
| NVS05 | 8 | 2 | 5 | 4 | non-linear | I | $[1,200]^2$ |
| NVS08 | 3 | 2 | 3 | 0 | non-linear | I | $[0,200]^2$ |
| PROB10 | 2 | 1 | 2 | 0 | non-linear | I | $[0,10]$ |
| SPRING | 17 | 12 | 3 | 5 | non-linear | both | $\{0,1\}^{11} \times [1,10]$ |
| ST_E36 | 2 | 1 | 1 | 1 | non-linear | I | $[15,25]$ |
| WINDFAC | 14 | 3 | 0 | 13 | non-linear | I | $[1,10] \times [1,100]^2$ |

Table 4.1: Summary of characteristics of MINLPlib test problems. $n$ denotes the dimension of the problem, $n_{\text{int}} \leq n$ denotes the number of integer variables including binary variables, $l$ denotes the number of inequality constraints, and $m$ denotes the number of equality constraints. Types of integer variables contain zero-one binary variables (B), integer-valued variables (I), or a mixture of both. All constraint functions include general nonlinear constraints.

same termination criteria as Schittkowski [73]. The evaluation metrics include the median number of objective function evaluations needed by each algorithm to meet the target accuracy over successful runs and the success rate, obtained as the ratio of the number of successful runs to the total number of runs.

### 4.1.3 Results

The experimental results are presented in Table 4.2 where, for each problem, we display the problem name, the median number of objective function evaluations across successful runs, and success rates. To bring into comparisons, the previously reported data of two optimization algorithms are also summarized in Table 4.2. Data for MISQP have been taken from Schittkowski [73]. Data for GAMBIT stem from Sadowski et al. [67] and for each problem represent the best algorithm variant considered there. There is no available data for GAMBIT on the problems marked with a dash. It should be noted that making a direct and fair comparison is difficult, as the

data have been collected from various sources that used different termination criteria and evaluation measures. Also, GAMBIT uses a different initialization condition than the other algorithms. Moreover, GAMBIT and MIDACO treat the constraint functions as black boxes, while we do not. MISQP by Exler and Schittkowski [21] applies finite differences and computes partial derivatives of objective functions and constraints to variables.

| | $AS_{mixInt}$-ES | MISQP | MIDACO | GAMBIT |
|---|---|---|---|---|
| EX1223A | 50/1.00 | 150/1.00 | 9010/1.00 | 20155/1.00 |
| EX1225 | 59/1.00 | 80/1.00 | 5773/0.57 | - |
| EX1226 | 13/1.00 | 21/1.00 | 329/1.00 | 2996/1.00 |
| GKOCIS | 144/1.00 | 373/1.00 | –/0.00 | - |
| PROCSEL | 133/1.00 | 362/1.00 | –/0.00 | - |
| ST_E15 | 12/1.00 | 34/1.00 | 5730/0.76 | 4001/1.00 |
| SYNTHES2 | 171/1.00 | 623/1.00 | 90962/0.05 | - |
| NVS01 | 21/1.00 | 140/1.00 | 56923/0.10 | - |
| NVS05 | 139/1.00 | 770/1.00 | –/0.00 | - |
| NVS08 | 42/1.00 | 138/1.00 | 2818/1.00 | - |
| PROB10 | 40/1.00 | 20/1.00 | 594/1.00 | - |
| SPRING | 61/1.00 | 965/1.00 | –/0.00 | - |
| ST_E36 | 58/1.00 | 188/1.00 | –/0.00 | - |
| WINDFAC | 10/1.00 | 3321/1.00 | –/0.00 | - |

Table 4.2: Median numbers of objective function evaluations required to solve problems across the successful runs and success rates within the required target accuracy $\epsilon = 10^{-6}$ and constraint tolerance $\delta = 10^{-6}$. – represents no optimal solutions found and - represents no data reported.

As it can be seen from Table 4.2, $AS_{mixInt}$-ES appears to perform at least as well, even better than MISQP and locates the global optima in all test instances. The median number of required objective function evaluations ranges between 10 for WINDFAC and 171 for SYNTHES2. Problem size alone does not determine the runtimes as illustrated in Table 4.2. For instance, $AS_{mixInt}$-ES takes similar median objective function evaluations to solve problems ST_E36 and SPRING. Problem ST_E36 comprises two variables, one of which takes a value ranging from 15 to 25, and two constraints. In contrast, problem SPRING consists of 17 variables, including eleven binary variables and one integer variable, and it is subject to eight constraints. Both problems involve nonlinear objective functions. On the other hand, problem NVS05, which requires a relatively large number of function evaluations to solve,

contains eight variables and nine constraints. The objective function of the problem is nonlinear, and there are two integer variables that can take values in a wider range between 1 and 200.

MISQP converges to globally optimal solutions within default tolerance for all problems reported in [73]. The number of objective function evaluations required ranges from 20 for PROB10 to 3321 for WINDFAC. Comparing the results of $AS_{mixInt}$-ES, MISQP achieves a higher number of objective function evaluations, with the exception of PROB10. MISQP computes gradients of both the objective function and constraints, which can be computationally expensive for some problems. In contrast, the $AS_{mixInt}$-ES assumes that constraint function evaluations have negligible cost compared to objective function evaluations and focuses solely on the number of objective function evaluations. The $AS_{mixInt}$-ES likely employs a far larger number of constraint function evaluations than MISQP.

The stochastic algorithm MIDACO successfully solves four of the fourteen problem instances in all runs. For the four problems (NVS05, ST_E36, SPRING, and WINDFAC) containing integer-valued variables, MIDACO cannot locate the optimal solution, even though the problem dimension is small. The results indicate that MIDACO presents a higher success rate with a relatively lower median number of objective function evaluations required when solving those with only zero-one binary variables, while its performance is poor with general integrality constraints. Additionally, for mixed-binary problems, MIDACO performs relatively well in problems with low-dimensional search spaces, and conversely, it exhibits inferior performance in high-dimensional problems.

The number of objective function evaluations required of GAMBIT is significantly larger than those of $AS_{mixInt}$-ES for all problems. It is important to note that the starting point of $AS_{mixInt}$-ES, MISQP, and MIDACO for each problem is determined using a priori-defined initial point given in [73], while GAMBIT does not utilize the pre-defined starting points. Furthermore, one has to keep in mind that MIDACO and GAMBIT are designed to work without any prior knowledge of constraint functions, therefore treating constraints as black boxes. However, if these algorithms exploit known problem characteristics, such as constraint functions, there would be potential for improved performance in such cases.

The above results indicate the potential of the proposed $AS_{mixInt}$-ES for solving such mixed-integer problems in terms of objective function evaluations. $AS_{mixInt}$-ES takes advantage of the explicit nature of the constraints. MISQP is comparable with the proposed $AS_{mixInt}$-ES, whereas the other algorithms are not.

## 4.2 Numerical Tests on CEC 2020 Benchmark

### 4.2.1 Algorithms

In this section, we validate the effectiveness of the proposed algorithm, conducting extensive experiments on CEC 2020 benchmark suite and comparing the performance with four other optimization algorithms in the literature. Except for MIDACO, the other three selected algorithms are the top three algorithms submitted to the CEC 2020 Competition on Real-World Single Objective Constrained Optimization. They are a self-adaptive spherical search algorithm (SASS) [46], a differential evolution variant (COLSHADE) [28], and a CMA-ES variant (sCMAgES) [45]. Both SASS and sCMAgES employ a gradient-based repair method and analytically compute gradients of constraint functions, while COLSHADE treats constraint functions as black boxes. None of these algorithms are intended for mixed-integer optimization and instead rely on rounding the values of integer variables with non-integer values before computing objective and constraint function values. We run these algorithms using code from the authors[1]. To avoid misunderstandings or incorrect parameter tuning, the parameters of these comparative algorithms are identical to default settings in the literature. Furthermore, we consider the SQP and interior-point methods implemented in MATLAB's fmincon in the comparison to confirm the efficiency of the (1+1)-AS-ES algorithm [83] on real-world continuous constrained problems.

### 4.2.2 Test Environment

A set of 14 benchmark problems are chosen from CEC 2020 real-world single-objective constrained optimization competition benchmarks collected by Kumar et al. [47]. These test problems arise from the area of chemical engineering and represent difficult non-convex optimization problems. The brief properties of these test problems

[1]The source codes of SASS, COLSHADE, and sCMAgES are available at https://github.com/P-N-Suganthan/2020-RW-Constrained-Optimisation.

are reported in Table 4.3. The test problem names are followed by the original citation sources. Some optimization problems in CEC 2020 benchmarks were chosen to be altered by Kumar et al. [47]. They imposed bound constraints on some problems and changed problems' properties, which can prevent algorithms from converging to local optima and lost physical significance. To ensure the comparability of the experiments, we have used formulations of the optimization problems that are either directly from the original references as provided in Table 4.3, or if the problem has been significantly reformulated before being included in the test set by Kumar et al. [47], we use a reference that preserves the problem's essential characteristics and optimal solution from the original reference[2]. We have ensured not to impose bound constraints that alter the characteristics of the problems significantly. Full details on these problems are given in Appendix A. These problems have different characteristics of objective functions (linear or non-linear), constraints (linear or non-linear, equalities or inequalities), and variables (continuous, zero-one binary, integer-valued). There are no more than 48 continuous variables and 4 integer variables. Moreover, there are up to 38 equality constraints and 14 inequality constraints. They fall into two broad classes. Problems RC01-RC07 are continuous constrained optimization problems in the field of industrial chemical processes. Problems RC08-RC14 are mixed-integer constrained optimization problems in the field of process synthesis and design. The variety of test problems enables us to evaluate the performance of all algorithms from various aspects, such as their capability to solve problems with different dimensions and constraint functions.

In order to succinctly present the results of this computational study, we use a fixed-target approach [33] for comparing algorithms. A successful run for each problem refers to the algorithm finding the optimal solution within target accuracy $\epsilon > 0$ and the solution being feasible within the constraint tolerance $\delta$. Unless otherwise noted, experiments are run with constraint tolerance $\delta = 10^{-8}$, and computations are carried out with relative termination accuracy. We generate a set of targets for each problem by choosing twenty values that are logarithmically uniformly distributed between $f^* + 10^{-8}\,|f^*|$ and $f^* + 10^{0}\,|f^*|$, where $f^*$ is the value of an optimal solution to

[2]According to our investigation, the source reference for problem RC13 by J. Wong cannot be found. Nevertheless, it seems that the problem has been derived from Himmelblau's problem [38] by removing three constraints and imposing two new integrality constraints.

| Problem | Ref | $n$ | $n_{\text{int}}$ | $l$ | $m$ | Type of objective function | Type of integer variables | Ranges for integer variables |
|---|---|---|---|---|---|---|---|---|
| RC01 | [88] | 3 | 0 | 0 | 2 | non-linear | | |
| RC02 | [88] | 5 | 0 | 0 | 3 | linear | | |
| RC03 | [71] | 7 | 0 | 14 | 0 | non-linear | | |
| RC04 | [56] | 6 | 0 | 1 | 4 | linear | | |
| RC05 | [37] | 9 | 0 | 2 | 4 | linear | | |
| RC06 | [23] | 38 | 0 | 0 | 32 | linear | | |
| RC07 | [3] | 48 | 0 | 0 | 38 | non-linear | | |
| RC08 | [43] | 2 | 1 | 2 | 0 | linear | B | $\{0,1\}^1$ |
| RC09 | [42] | 3 | 1 | 1 | 1 | linear | B | $\{0,1\}^1$ |
| RC10 | [22] | 3 | 1 | 3 | 0 | non-linear | B | $\{0,1\}^1$ |
| RC11 | [44] | 8 | 2 | 4 | 4 | linear | B | $\{0,1\}^2$ |
| RC12 | [89] | 7 | 4 | 9 | 0 | non-linear | B | $\{0,1\}^4$ |
| RC13 | [15] | 5 | 2 | 3 | 0 | non-linear | I | $[78,102] \times [33,45]$ |
| RC14 | [26] | 10 | 3 | 13 | 0 | non-linear | I | $[1,3]^3$ |

Table 4.3: Summary of characteristics of CEC 2020 problems RC01-RC14. $n$ denotes the dimension of the problem, $n_{\text{int}} \leq n$ is the number of integer variables, $l$ details the number of inequality constraints, and $m$ gives the number of equality constraints. Types of integer variables contain zero-one binary variables (B) or integer-valued variables (I). All constraint functions include general nonlinear constraints.

the problem. For each algorithm, the empirical cumulative running time distribution function (ECDF) plots show the fraction of reached objective function value targets for each problem across 21 runs against the number of objective function evaluations. In order for a target to be considered as reached, the corresponding solution generated by the algorithm needs to be feasible within tolerance $\delta$. To ensure a fair comparison between the algorithms and reduce potential biases, we use identical initialization methods and termination criteria across all experiments. Initial starting points of all algorithms are generated by sampling uniformly and randomly within the bound-constrained region of the test problems. The step size parameter $\sigma$ of the proposed algorithm $\text{AS}_{\text{mixInt}}$-ES is initialized to one-fifth of the minimum extent of the interval width for all variables in all runs. The maximum number of objective function evaluations for all algorithms is set to $10^5$. To ensure that the SQP and interior-point algorithms do not terminate before finding a feasible solution for the desired target accuracy, we specify the optimality tolerance, step tolerance, and function tolerance parameters as zero, and employ an output function to halt the algorithms when the termination criteria outlined above are met. Except for sCMAgES and MIDACO, we do not consider restart schemes in our experiments.

### 4.2.3 Results

The numerical results of testing the algorithms implemented in MATLAB's fmincon function on a set of CEC 2020 benchmark problems are summarized in Table 4.4. For each problem considered, we have conducted 21 runs of the above-mentioned algorithms for target accuracies $\epsilon \in \{10^{-4}, 10^{-8}\}$ and constraint tolerance $\delta = 10^{-8}$. Each cell in the table displays results for target accuracy $\epsilon = 10^{-4}$ on the left and for $\epsilon = 10^{-8}$ on the right. In each cell, the upper row shows the median number of objective function evaluations required to reach termination accuracy for all successful runs of each algorithm. The lower row shows the percentage of successful runs out of all runs. The table does not include data for SASS, COLSHADE, and sCMAgES due to their high computational cost, which requires a number of objective function evaluations ranging from a few thousand to tens of thousands. In the table, problems RC06 through RC07 for some algorithms are marked with dashed lines indicating that no optimal solutions could be located for these problems. The SQP and interior-point algorithms have no values in the cells from problems RC08 to RC14 due to the presence of integrality constraints that make the algorithms inapplicable. It is worth noting that when there are no integrality constraints, the $\text{AS}_{\text{mixInt}}$-ES is equivalent to the (1+1)-AS-ES described by Spettel et al. [83].

It can be observed that none of the algorithms can solve all problems from RC01 to RC07 in all runs from Table 4.4. The $\text{AS}_{\text{mixInt}}$-ES solves seven out of the 14 problems within both target accuracies $\epsilon = 10^{-4}$ and $\epsilon = 10^{-8}$ in all runs. The $\text{AS}_{\text{mixInt}}$-ES fails to solve problems RC01, RC04, RC05, RC06, RC07, RC08, and RC11 in some of the runs, which are multimodal. In such cases, the $\text{AS}_{\text{mixInt}}$-ES converges to locally optimal solutions, which have been noted in several references [66, 23, 3, 44]. Incorporating $\text{AS}_{\text{mixInt}}$-ES into a straightforward restart scheme would significantly increase the probability of finding globally optimal solutions.

The algorithm exhibits relatively lower success rates in problems RC06 and RC07, solving only around 15 out of 21 runs. These two problems have the highest dimension and the largest number of constraints active at the optimal solution. In the unsuccessful runs, the algorithm runs into local optimizers, which prevents it from converging to the optimal solution. Nevertheless, $\text{AS}_{\text{mixInt}}$-ES achieves among the highest success rates for RC06 and RC07 of all algorithms considered. The interior-point method

| | $AS_{mixInt}$-ES | *active-set* | *sqp* | *interior-point* |
|---|---|---|---|---|
| RC01 | 14/13<br>0.95/0.95 | 24/24<br>0.76/0.86 | 35/41<br>0.90/0.90 | 127/181<br>0.95/0.90 |
| RC02 | 101/181<br>1.00/1.00 | 316/341<br>0.86/0.86 | 230/229<br>1.00/1.00 | 1886/1716<br>1.00/1.00 |
| RC03 | 194/321<br>1.00/1.00 | 175/178<br>0.95/1.00 | 215/399<br>1.00/1.00 | 730/677<br>1.00/0.95 |
| RC04 | 19/85<br>0.90/0.95 | 1183/1624<br>0.71/0.57 | 114/108<br>0.57/0.57 | 198/213<br>1.00/1.00 |
| RC05 | 60/42<br>0.95/0.90 | 175/165<br>0.86/0.95 | 161/161<br>0.90/1.00 | 252/261<br>0.76/1.00 |
| RC06 | 61/66<br>0.67/0.67 | 24934/19826<br>0.38/0.38 | 2383/2577<br>0.62/0.71 | –/–<br>0.00/0.00 |
| RC07 | 90/115<br>0.76/0.71 | –/–<br>0.00/0.00 | 4809/4411<br>0.62/0.43 | –/10686<br>0.00/0.14 |
| RC08 | 7/10<br>0.95/0.95 | -/- | -/- | -/- |
| RC09 | 5/7<br>1.00/1.00 | -/- | -/- | -/- |
| RC10 | 17/26<br>1.00/1.00 | -/- | -/- | -/- |
| RC11 | 65/142<br>0.81/0.86 | -/- | -/- | -/- |
| RC12 | 81/63<br>1.00/1.00 | -/- | -/- | -/- |
| RC13 | 14/21<br>1.00/1.00 | -/- | -/- | -/- |
| RC14 | 109/104<br>1.00/1.00 | -/- | -/- | -/- |

Table 4.4: Each cell in the table contains four data points. The top row of the cell shows the median number of objective function evaluations required to solve problems across the successful runs on the CEC 2020 benchmarks, for two different target accuracies: $10^{-4}$ on the left and $10^{-8}$ on the right. The bottom row of the cell displays success rates for the corresponding target accuracies $10^{-4}$ and $10^{-8}$.

has encountered singular matrices when solving problem RC07 and failed to converge to a solution. Comparing the median numbers of objective function evaluations required across successful runs, the $\text{AS}_{\text{mixInt}}$-ES is the fastest algorithm for most of those problems tested. However, for problem RC03, the *active-set* method exhibits superior performance. A single run of the algorithm on problem RC04 is unsuccessful. Problem RC04 exhibits multi-modality and contains two local minima, with one of them having an objective function value close to the global optimal function value. *Interior-point* method is the only algorithm to solve problems RC04 and RC05 in all runs within target accuracy $\epsilon = 10^{-8}$.

The $\text{AS}_{\text{mixInt}}$-ES locates globally optimal solutions in 89 percent of runs for target accuracy $\epsilon = 10^{-4}$ and in 88 percent of runs for $\epsilon = 10^{-8}$ across all continuous problems RC01 through RC07. The corresponding rates for *active-set* are 65 percent and 66 percent, for *sqp* are 80 percent for both target accuracies, and for *interior-point* are 67 percent and 71 percent. As shown in Figure B.1 of Appendix B, for SASS are 43 percent and 21 percent, for COLSHADE are 15 percent and 16 percent, and for sCMAgES are 27 percent and 16 percent. MIDACO cannot achieve globally optimal solutions for these problems within both target accuracies and the tightest constraint tolerance even in a single run.

It can be noted that the performance of the $\text{AS}_{\text{mixInt}}$-ES is consistent with the results reported by Spettel et al. [83] from the data in Table 4.4. With regard to all constrained mixed-integer problems from RC08 to RC14, the $\text{AS}_{\text{mixInt}}$-ES reaches globally optimal solutions in 97 percent of runs for both target accuracies. The SQP and interior-point algorithms are not capable of solving mixed-integer optimization problems. The corresponding rates within both target accuracies shown in Figure B.1 of Appendix B for SASS are 71 percent, for sCMAgES are 71 percent, for COLSHADE are 53 percent, and for MIDACO are 58 percent of runs and 56 percent of runs with $\epsilon = 10^{-4}$ and $\epsilon = 10^{-8}$ respectively. It is noteworthy that the relatively poor performance of COLSHADE and MIDACO may be attributed to a strict constraint tolerance $\delta$ used in the conducted experiments. COLSHADE and MIDACO do not require any assumptions on the constraint functions and treat constraints as black boxes, while other algorithms assume knowledge of the constraint functions. As shown in Figure B.2 of Appendix B, with a higher constraint tolerance of $\delta = 10^{-4}$ used by

Schlüter [77], MIDACO can locate globally optimal solutions in 62 percent of runs and 64 percent of runs for $\epsilon = 10^{-4}$ and $\epsilon = 10^{-8}$ respectively. The corresponding rates with $\delta = 10^{-2}$ for MIDACO are 66 percent and 69 percent respectively.

Empirical cumulative running time distributions for all fourteen test problems are displayed in Figure 4.1. For each algorithm, the plots show the percentage of runs that reached objective function values within the required accuracy and tolerance $\delta$ against running time. A measure of its performance is the ratio of the number of targets achieved from the set. With the exception of problem RC03, the $\text{AS}_{\text{mixInt}}$-ES demonstrates the quickest convergence to globally optimal solutions among all problems in Figure 4.1. For problem RC03, the *active-set* method requires fewer objective function evaluations to reach the hardest target accuracy. Except for $\text{AS}_{\text{mixInt}}$-ES, all other algorithms achieve all targets within tolerance $\delta$ on problem RC08, where one of the 21 runs converges to a local optimum. On problem RC11, none of the algorithms are able to reach all targets. $\text{AS}_{\text{mixInt}}$-ES achieves 88 percent of the targets, MIDACO 30 percent, sCMAgES 8 percent, and COLSHADE 6 percent. The introduction of a restart technique can possibly prevent the $\text{AS}_{\text{mixInt}}$-ES from getting stuck on the local optimum and achieving better performance. $\text{AS}_{\text{mixInt}}$-ES is the only algorithm to meet all targets on the 10-dimensional problem RC14 which contains three integer variables. Empirical cumulative running time distributions for MIDACO across all test problems within different constraint tolerances and target accuracies are provided in Figure B.3. In certain cases, it has been observed that MIDACO achieves a higher number of targets with the strictest tolerance and target accuracy compared to other settings within the same number of function evaluations.

Furthermore, compared to the other algorithms, $\text{AS}_{\text{mixInt}}$-ES is capable of reaching more targets with a single objective function evaluation due to projecting feasible candidate solutions onto the feasible region, even in several constrained mixed-integer test problems. As mentioned previously, the integer variables in the implementations provided by Kumar et al. [47] are rounded before applying objective or constraint functions. Thus, SASS, COLSHADE, and sCMAgES are not specifically tailored for mixed-integer optimization. In contrast, MIDACO exhibits better performance in mixed-integer test problems compared to problems with continuous search spaces.

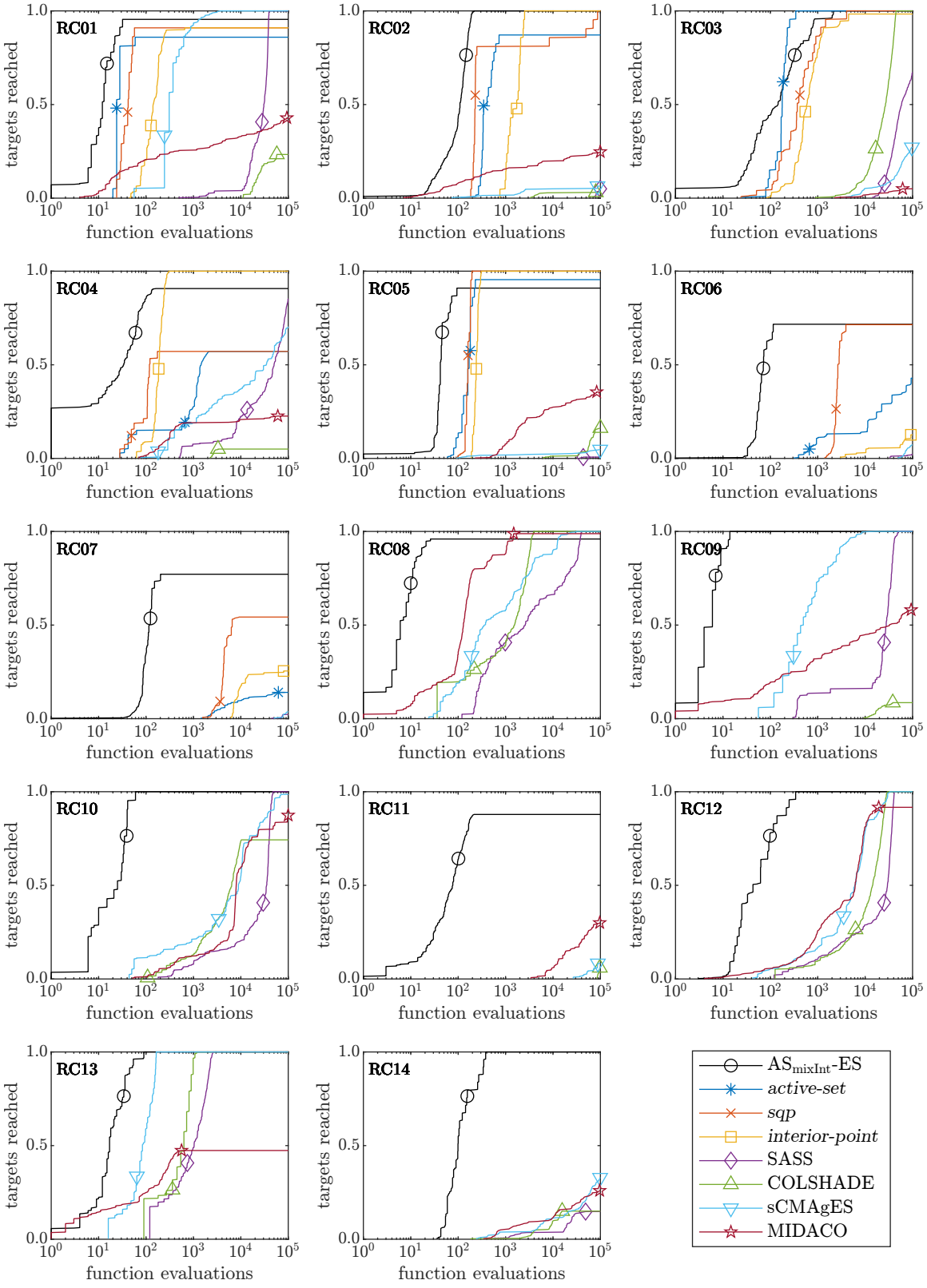


Figure 4.1: Empirical cumulative running time distributions on the CEC 2020 problems.

# Chapter 5

# Conclusion

In this chapter, we summarize the main contributions of this thesis, and provide potential directions for future research.

## 5.1 Summary

In this thesis, we have proposed $AS_{mixInt}$-ES as an evolutionary algorithm for solving mixed-integer problems with explicit constraints. Previous studies on active-set evolution strategies [6, 7, 83] for constrained optimization have indicated that employing active-set techniques to enforce active constraints to be equality constraints can reduce the search space dimensions. According to the results of the CEC 2006 test set [53], the (1+1)-AS-ES algorithm proposed by Spettel et al. [83] appears to have higher success rates for some problems than the SQP and interior-point methods implemented in MATLAB's fmincon, while using a comparable number of objective function evaluations. The (1+1)-AS-ES algorithm [83] is designed for continuous black-box optimization with explicit constraints. To expand its applicability to optimize mixed-integer constrained problems, a branching technique is incorporated to prevent the optimization algorithm from converging to solutions that violate any integrality constraints. In this thesis, we focus on explicit constraints, where the constraint functions are computationally cheap to evaluate.

In contrast to other solvers, we do not utilize the bounding scheme of Branch-and-Bound methods to prune the search tree. Our proposed algorithm avoids solving problems to optimality at each node. We use heuristics instead to decide which node to propagate forward, when to split a problem into subproblems, and when to eliminate them from further consideration. For our proposed algorithm, we have conducted multiple runs on linearly constrained random spheres and collected experimental data for analysis. Empirical studies show that the $AS_{mixInt}$-ES can converge to the global optimum with different numbers of integrality constraints and linear constraints. The

results suggest that in terms of objective function evaluations, the algorithm's performance is impacted by the number of integer variables and constraint functions. These effects can contribute to the complexity of the problems to a certain extent. In addition, the range of optimal solutions on integer variables is a factor to affect the algorithm's effectiveness.

To validate the applicability and efficiency of $\text{AS}_{\text{mixInt}}$-ES, we have applied it to a set of 28 constrained optimization problems, half of which have been taken from MINLPlib [14], half of which have been adapted from the CEC 2020 test set of Kumar et al. [47]. Our experiments demonstrate that $\text{AS}_{\text{mixInt}}$-ES can be applied to linearly as well as non-linearly constrained mixed-integer problems. In most cases, the $\text{AS}_{\text{mixInt}}$-ES requires fewer objective function evaluations to solve constrained mixed-integer problems from MINLPlib compared to MISQP, a mixed-integer trust-region sequential quadratic programming algorithm. Both MISQP and $\text{AS}_{\text{mixInt}}$-ES compute constraint gradients, but the $\text{AS}_{\text{mixInt}}$-ES assumes that constraint functions are explicit and requires more constraint function evaluations than objective function evaluations. This assumption allows the $\text{AS}_{\text{mixInt}}$-ES to succeed faster in black-box optimization scenarios compared to algorithms that do not assume explicit constraints.

Among CEC 2020 problem sets considered, seven problems do not have integrality constraints. For these, the $\text{AS}_{\text{mixInt}}$-ES and the (1+1)-AS-ES algorithm perform exactly the same. We compare the empirical cumulative running time distributions of $\text{AS}_{\text{mixInt}}$-ES to those of the SQP and interior-point algorithms in MATLAB's fmincon function, the top-three algorithms from the CEC 2020 competition, and MIDACO version 6.0. Our studies confirm the observations made by Spettel et al. [83] that the active-set evolution strategy outperforms other algorithms for most problems. However, some of the other algorithms achieve higher success rates for some test problems when the active-set evolution strategy converges to the local optimum. Embedding a restart scheme would improve the chances to obtain the globally optimal solutions without exceeding its computational budget. The active-set evolution strategy treats constraint functions explicitly, employing more constraint function evaluations than objective function evaluations, which contributes to its lower computational cost. For those problems with integrality constraints considered in the CEC 2020 set, the $\text{AS}_{\text{mixInt}}$-ES also dominates the empirical cumulative running time distributions of the

top-three algorithms in the CEC 2020 competition and MIDACO. The $\text{AS}_{\text{mixInt}}$-ES is approximately two or three orders of magnitude faster than those algorithms. In some low dimensional cases, the $\text{AS}_{\text{mixInt}}$-ES only requires tens of objective function evaluations. This is because other algorithms do not assume that constraint function evaluations have negligible cost compared with objective function evaluations. In fact, if constraints are defined explicitly, algorithms can exploit this information and potentially improve their performance.

## 5.2 Future Research

While the experimental results of the algorithm are encouraging, it is important to acknowledge that the algorithm still has some limitations and potential directions for further investigation. Studies conducted on linearly constrained sphere functions have shown that the algorithm's performance appears to deteriorate with a large number of constraints and integrality constraints. The range of optimal solutions on integer variables can also pose challenges for the algorithm. However, these behaviours were not observed in the real-world test problems considered in our experiments. This is likely due to the fact that these test problems have no more than twelve integrality constraints, and in all but one problem, the number of integrality constraints is six or fewer. In general, larger numbers of integer variables may lead to combinatorial difficulties resulting in more computational effort in the branching process. The simple biased sampling of a live node to propagate may not be sufficient to handle such challenges. Considering variants of best-first search that focus on node selection could be a potential step to address this issue. Evaluating the $\text{AS}_{\text{mixInt}}$-ES on a broader range of real-world optimization problems with more than ten integer variables is also an important future direction. In addition, the algorithm can be extended to address multimodal and ill-conditioned optimization problems, which are areas of interest for future research. Finally, the $\text{AS}_{\text{mixInt}}$-ES requires the mixed-integer problems must be relaxable subject to integrality constraints based on the branching mechanism and does not consider categorical variables. A potential future target is to develop an algorithm variant that does not require making the assumption mentioned above, as the assumption may be suitable for certain problems, but it may not be applicable to others.

# Appendix A

## CEC 2020 Test Functions

The corrected test problems used in Section 4.2 are given. The original references, the objective function, constraint functions and the optimal function value are listed.

**Problem RC01** (Westerberg and Shah [88])

Kumar et al. [47] manipulated the constraint equations algebraically and added bound constraints active at the optimal solution. They simplified the problem by excluding locally optimal solutions. Ryoo and Sahinidis [66] (example 11) used less restrictive bound constraints and only included a single weakly active physical constraint at the optimal solution. We use their formulation to represent the original real-world problem.

Minimize:

$$f(\mathbf{x}) = 35x_1^{0.6} + 35x_2^{0.6}$$

subject to:

$$h_1(\mathbf{x}) = 600x_1 - 50x_3 - x_1x_3 + 5000 = 0$$
$$h_2(\mathbf{x}) = 600x_2 + 50x_3 - 15000 = 0$$

with bounds:

$$(0, 0, 100) \leq \mathbf{x} \leq (34, 17, 300)$$

$f(\mathbf{x}^*) = 1.893116296866205$.

**Problem RC02** (Westerberg and Shah [88])

Similar to Problem RC01, we refer to the problem formulation by Ryoo and Sahinidis [66] (example 5).

Minimize:

$$f(\mathbf{x}) = x_1 + x_2 + x_3$$

subject to:

$$\begin{aligned} h_1(\mathbf{x}) &= 100000\,(x_4 - 100) - 120x_1\,(300 - x_4) = 0 \\ h_2(\mathbf{x}) &= 100000\,(x_5 - x_4) - 80x_2\,(400 - x_5) = 0 \\ h_3(\mathbf{x}) &= 100000\,(500 - x_5) - 40x_3(600 - 500) = 0 \end{aligned}$$

with bounds:

$$(0, 0, 0, 100, 100) \leq \mathbf{x} \leq (15834, 36, 250, 10000, 300, 400)$$

$f(\mathbf{x}^*) = 7049.249272475995.$

**Problem RC03** (Sauer et al. [71])

Kumar et al. [47] used bound constraints that are not present in earlier references [71, 12, 18]. However, their optimal solution is physically infeasible. The formulation by Adjiman et al. [2] appears to be the most relevant to the original problem among all formulations.

Minimize:

$$f(\mathbf{x}) = -0.035x_1x_6 - 1.715x_1 - 10x_2 - 4.0565x_3 + 0.063x_3x_5$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= 0.0059553571x_6^2x_1 + 0.88392857x_3 - 0.1175625x_6x_1 - x_1 \leq 0 \\ g_2(\mathbf{x}) &= 1.1088x_1 + 0.1303533x_1x_6 - 0.0066033x_1x_6^2 - x_3 \leq 0 \\ g_3(\mathbf{x}) &= 6.66173269x_6^2 - 56.596669x_4 + 172.39878x_5 - 10000 \\ &\qquad - 191.20592x_6 \leq 0 \\ g_4(\mathbf{x}) &= 1.08702x_6 - 0.03762x_6^2 + 0.32175x_4 + 56.85075 - x_5 \leq 0 \\ g_5(\mathbf{x}) &= 0.006198x_7x_4x_3 + 2462.3121x_2 - 25.125634x_2x_4 - x_3x_4 \leq 0 \end{aligned}$$

$$g_6(\mathbf{x}) = 161.18996x_3x_4 + 5000x_2x_4 - 489510x_2 - x_3x_4x_7 \leq 0$$
$$g_7(\mathbf{x}) = 0.33x_7 + 44.333333 - x_5 \leq 0$$
$$g_8(\mathbf{x}) = 0.022556x_5 - 1.0 - 0.007595x_7 \leq 0$$
$$g_9(\mathbf{x}) = 0.00061x_3 - 1.0 - 0.0005x_1 \leq 0$$
$$g_{10}(\mathbf{x}) = 0.819672x_1 - x_3 + 0.819672 \leq 0$$
$$g_{11}(\mathbf{x}) = 24500x_2 - 250x_2x_4 - x_3x_4 \leq 0$$
$$g_{12}(\mathbf{x}) = 1020.4082x_4x_2 + 1.2244898x_3x_4 - 100000x_2 \leq 0$$
$$g_{13}(\mathbf{x}) = 6.25x_1x_6 + 6.25x_1 - 7.625x_3 - 100000 \leq 0$$
$$g_{14}(\mathbf{x}) = 1.22x_3 - x_6x_1 - x_1 + 1.0 \leq 0$$

with bounds:

$$1 \leq x_1 \leq 2000$$
$$1 \leq x_2 \leq 120$$
$$1 \leq x_3 \leq 5000$$
$$85 \leq x_4 \leq 93$$
$$90 \leq x_5 \leq 95$$
$$3 \leq x_6 \leq 12$$
$$145 \leq x_7 \leq 162$$

$f(\mathbf{x}^*) = -1766.365179377954.$

**Problem RC04** (Manousiouthakis and Sourla [56])

Minimize:

$$f(\mathbf{x}) = -x_4$$

subject to:

$$g_1(\mathbf{x}) = x_5^{0.5} + x_6^{0.6} - 4 \leq 0$$
$$h_1(\mathbf{x}) = k_1x_5x_1 + x_1 - 1 = 0$$

$$h_2(\mathbf{x}) = k_3 x_5 x_3 + x_3 + x_1 - 1 = 0$$
$$h_3(\mathbf{x}) = k_2 x_6 x_2 - x_1 + x_2 = 0$$
$$h_4(\mathbf{x}) = k_4 x_6 x_4 + x_2 - x_1 + x_4 - x_3 = 0$$

where $k_3 = 0.0391908, k_4 = 0.9k_3, k_1 = 0.09755988,$ and $k_2 = 0.99k_1$

with bounds:

$$(0, 0, 0, 0, 0, 0) \leq \mathbf{x} \leq (1, 1, 1, 1, 16, 16)$$

$f(\mathbf{x}^*) = -0.3888114342920.$

**Problem RC05** (Haverly [37])

Minimize:

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

subject to:

$$g_1(\mathbf{x}) = x_3 x_9 + 2x_6 - 2.5x_5 \leq 0$$
$$g_2(\mathbf{x}) = x_4 x_9 + 2x_7 - 1.5x_8 \leq 0$$
$$h_1(\mathbf{x}) = 3x_1 + x_2 - x_9(x_3 + x_4) = 0$$
$$h_2(\mathbf{x}) = x_1 + x_2 - x_3 - x_4 = 0$$
$$h_3(\mathbf{x}) = x_3 - x_5 + x_6 = 0$$
$$h_4(\mathbf{x}) = x_4 + x_7 - x_8 = 0$$

with bounds:

$$(0, 0, 0, 0, 0, 0, 0, 0, 1) \leq \mathbf{x} \leq (300, 300, 100, 200, 100, 100, 200, 200, 3)$$

$f(\mathbf{x}^*) = -400.$

**Problem RC06** (Floudas and Aggarwal [23])

Kumar et al. [47] also impose bound constraints. We refer to the original formulation in [23].

Minimize:

$$f(\mathbf{x}) = 0.9979 + 0.00432x_5 + 0.01517x_{13}$$

subject to:

$$h_1(\mathbf{x}) = x_4 + x_3 + x_2 + x_1 - 300 = 0$$
$$h_2(\mathbf{x}) = x_6 - x_8 - x_7 = 0$$
$$h_3(\mathbf{x}) = x_9 - x_{11} - x_{10} - x_{12} = 0$$
$$h_4(\mathbf{x}) = x_{14} - x_{16} - x_{17} - x_{15} = 0$$
$$h_5(\mathbf{x}) = x_{18} - x_{20} - x_{19} = 0$$
$$h_6(\mathbf{x}) = x_5x_{21} - x_6x_{22} - x_9x_{23} = 0$$
$$h_7(\mathbf{x}) = x_5x_{24} - x_6x_{25} - x_9x_{26} = 0$$
$$h_8(\mathbf{x}) = x_5x_{27} - x_6x_{28} - x_9x_{29} = 0$$
$$h_9(\mathbf{x}) = x_{13}x_{30} - x_{14}x_{31} - x_{18}x_{32} = 0$$
$$h_{10}(\mathbf{x}) = x_{13}x_{33} - x_{14}x_{34} - x_{18}x_{35} = 0$$
$$h_{11}(\mathbf{x}) = x_{13}x_{36} - x_{14}x_{37} - x_{18}x_{38} = 0$$
$$h_{12}(\mathbf{x}) = \frac{1}{3}x_1 + x_{15}x_{31} - x_5x_{21} = 0$$
$$h_{13}(\mathbf{x}) = \frac{1}{3}x_1 + x_{15}x_{34} - x_5x_{24} = 0$$
$$h_{14}(\mathbf{x}) = \frac{1}{3}x_1 + x_{15}x_{37} - x_5x_{27} = 0$$
$$h_{15}(\mathbf{x}) = \frac{1}{3}x_2 + x_{10}x_{23} - x_{13}x_{30} = 0$$
$$h_{16}(\mathbf{x}) = \frac{1}{3}x_2 + x_{10}x_{26} - x_{13}x_{33} = 0$$
$$h_{17}(\mathbf{x}) = \frac{1}{3}x_2 + x_{10}x_{29} - x_{13}x_{36} = 0$$
$$h_{18}(\mathbf{x}) = \frac{1}{3}x_3 + x_7x_{22} + x_{11}x_{23} + x_{16}x_{31} + x_{19}x_{32} - 30 = 0$$
$$h_{19}(\mathbf{x}) = \frac{1}{3}x_3 + x_7x_{25} + x_{11}x_{26} + x_{16}x_{34} + x_{19}x_{35} - 50 = 0$$
$$h_{20}(\mathbf{x}) = \frac{1}{3}x_3 + x_7x_{28} + x_{11}x_{29} + x_{16}x_{37} + x_{19}x_{38} - 30 = 0$$
$$h_{21}(\mathbf{x}) = x_{21} + x_{24} + x_{27} - 1 = 0$$
$$h_{22}(\mathbf{x}) = x_{22} + x_{25} + x_{28} - 1 = 0$$

$$h_{23}(\mathbf{x}) = x_{23} + x_{26} + x_{29} - 1 = 0$$
$$h_{24}(\mathbf{x}) = x_{30} + x_{33} + x_{36} - 1 = 0$$
$$h_{25}(\mathbf{x}) = x_{31} + x_{34} + x_{37} - 1 = 0$$
$$h_{26}(\mathbf{x}) = x_{32} + x_{35} + x_{38} - 1 = 0$$
$$h_{27}(\mathbf{x}) = x_{25} = 0$$
$$h_{28}(\mathbf{x}) = x_{28} = 0$$
$$h_{29}(\mathbf{x}) = x_{23} = 0$$
$$h_{30}(\mathbf{x}) = x_{37} = 0$$
$$h_{31}(\mathbf{x}) = x_{32} = 0$$
$$h_{32}(\mathbf{x}) = x_{35} = 0$$

with bounds:

$$0 \leq x_i \leq 300, i = 1, 2, \ldots, 20$$
$$0 \leq x_i \leq 1, i = 21, 22, \ldots, 38$$

$f(\mathbf{x}^*) = 1.8639$.

**Problem RC07** (Aggarwal and Floudas [3])

Minimize:

$$f(\mathbf{x}) = c_{11} + (c_{21} + c_{31}x_{24} + c_{41}x_{28} + c_{51}x_{33} + c_{61}x_{34})x_5 + c_{12} + (c_{22} + c_{32}x_{26} + c_{42}x_{31} + c_{52}x_{38} + c_{62}x_{39})x_{13}$$

where

| $c$ | $i = 1$ | $i = 2$ |
|---|---|---|
| $c_{1i}$ | 0.23947 | 0.75835 |
| $c_{2i}$ | -0.0139904 | -0.0661588 |
| $c_{3i}$ | 0.0093514 | 0.0338147 |
| $c_{4i}$ | 0.0077308 | 0.0373349 |
| $c_{5i}$ | -0.0005719 | 0.0016371 |
| $c_{6i}$ | 0.004256 | 0.0288996 |

subject to:

$$h_1(\mathbf{x}) = x_4 + x_3 + x_2 + x_1 - 300 = 0$$
$$h_2(\mathbf{x}) = x_6 - x_8 - x_7 = 0$$
$$h_3(\mathbf{x}) = x_9 - x_{11} - x_{10} - x_{12} = 0$$
$$h_4(\mathbf{x}) = x_{14} - x_{16} - x_{17} - x_{15} = 0$$
$$h_5(\mathbf{x}) = x_{18} - x_{20} - x_{19} = 0$$
$$h_6(\mathbf{x}) = x_6 x_{21} - x_{24} x_{25} = 0$$
$$h_7(\mathbf{x}) = x_{14} x_{22} - x_{26} x_{27} = 0$$
$$h_8(\mathbf{x}) = x_9 x_{23} - x_{28} x_{29} = 0$$
$$h_9(\mathbf{x}) = x_{18} x_{30} - x_{31} x_{32} = 0$$
$$h_{10}(\mathbf{x}) = x_{25} - x_5 x_{33} = 0$$
$$h_{11}(\mathbf{x}) = x_{29} - x_5 x_{34} = 0$$
$$h_{12}(\mathbf{x}) = x_{35} - x_5 x_{36} = 0$$
$$h_{13}(\mathbf{x}) = x_{37} - x_{13} x_{38} = 0$$
$$h_{14}(\mathbf{x}) = x_{27} - x_{13} x_{39} = 0$$
$$h_{15}(\mathbf{x}) = x_{32} - x_{13} x_{40} = 0$$
$$h_{16}(\mathbf{x}) = x_{25} - x_6 x_{21} - x_9 x_{41} = 0$$
$$h_{17}(\mathbf{x}) = x_{29} - x_6 x_{42} - x_9 x_{23} = 0$$
$$h_{18}(\mathbf{x}) = x_{35} - x_6 x_{43} - x_9 x_{44} = 0$$
$$h_{19}(\mathbf{x}) = x_{37} - x_{14} x_{45} - x_{18} x_{46} = 0$$
$$h_{20}(\mathbf{x}) = x_{27} - x_{14} x_{22} - x_{18} x_{47} = 0$$
$$h_{21}(\mathbf{x}) = x_{32} - x_{14} x_{48} - x_{18} x_{30} = 0$$
$$h_{22}(\mathbf{x}) = \frac{1}{3} x_1 + x_{15} x_{45} - x_{25} = 0$$
$$h_{23}(\mathbf{x}) = \frac{1}{3} x_1 + x_{15} x_{22} - x_{29} = 0$$
$$h_{24}(\mathbf{x}) = \frac{1}{3} x_1 + x_{15} x_{48} - x_{35} = 0$$
$$h_{25}(\mathbf{x}) = \frac{1}{3} x_2 + x_{10} x_{41} - x_{37} = 0$$

$$h_{26}(\mathbf{x}) = \frac{1}{3}x_2 + x_{10}x_{23} - x_{27} = 0$$
$$h_{27}(\mathbf{x}) = \frac{1}{3}x_2 + x_{10}x_{44} - x_{32} = 0$$
$$h_{28}(\mathbf{x}) = \frac{1}{3}x_3 + x_7x_{21} + x_{11}x_{41} + x_{16}x_{45} + x_{19}x_{46} - 30 = 0$$
$$h_{29}(\mathbf{x}) = \frac{1}{3}x_3 + x_7x_{42} + x_{11}x_{23} + x_{16}x_{22} + x_{19}x_{47} - 50 = 0$$
$$h_{30}(\mathbf{x}) = \frac{1}{3}x_3 + x_7x_{43} + x_{11}x_{44} + x_{16}x_{48} + x_{19}x_{30} - 30 = 0$$
$$h_{31}(\mathbf{x}) = x_{33} + x_{34} + x_{36} - 1 = 0$$
$$h_{32}(\mathbf{x}) = x_{21} + x_{42} + x_{43} - 1 = 0$$
$$h_{33}(\mathbf{x}) = x_{41} + x_{23} + x_{44} - 1 = 0$$
$$h_{34}(\mathbf{x}) = x_{38} + x_{39} + x_{40} - 1 = 0$$
$$h_{35}(\mathbf{x}) = x_{45} + x_{22} + x_{48} - 1 = 0$$
$$h_{36}(\mathbf{x}) = x_{46} + x_{47} + x_{30} - 1 = 0$$
$$h_{37}(\mathbf{x}) = x_{43} = 0$$
$$h_{38}(\mathbf{x}) = x_{46} = 0$$

with bounds:

$$0 \leq x_i \leq 300, i = 1, 2, \ldots, 20, 25, 27, 29, 32, 35, 37$$
$$0.85 \leq x_i \leq 1, i = 24, 26, 28, 31$$
$$0 \leq x_i \leq 1, i = 21, 22, 23, 30, 33, 34, 36, 38, 39, \ldots, 48$$

$f(\mathbf{x}^*) = 1.567072$.

**Problem RC08** (Kocis and Grossmann [43])

Minimize:

$$f(\mathbf{x}) = x_2 + 2x_1$$

subject to:

$$g_1(\mathbf{x}) = -x_1^2 - x_2 + 1.25 \leq 0$$
$$g_2(\mathbf{x}) = x_1 + x_2 \leq 1.6$$

with bounds:

$$0 \leq x_1 \leq 1.6$$
$$x_2 \in \{0, 1\}$$

$f(\mathbf{x}^*) = 2.0$.

**Problem RC09** (Kocis and Grossmann [42])

Minimize:

$$f(\mathbf{x}) = -x_3 + x_2 + 2x_1$$

subject to:

$$g_1(\mathbf{x}) = x_2 - x_1 + x_3 \leq 0$$
$$h_1(\mathbf{x}) = -2e^{-x_2} + x_1 = 0$$

with bounds:

$$0.5 \leq x_1 \leq 1.4$$
$$0 \leq x_2 \leq 1.4$$
$$x_3 \in \{0, 1\}$$

$f(\mathbf{x}^*) = 2.124467584550870$.

**Problem RC10** (Floudas [22])

Minimize:

$$f(\mathbf{x}) = -0.7x_3 + 0.8 + 5(0.5 - x_1)^2$$

subject to:

$$g_1(\mathbf{x}) = -e^{x_1 - 0.2} - x_2 \leq 0$$
$$g_2(\mathbf{x}) = x_2 + 1.1x_3 + 1 \leq 0$$
$$g_3(\mathbf{x}) = x_1 - 1.2x_3 - 0.2 \leq 0$$

with bounds:

$$0.2 \leq x_1 \leq 1$$
$$-2.22554 \leq x_2 \leq -1$$
$$x_3 \in \{0, 1\}$$

$f(\mathbf{x}^*) = 1.076543083332262$.

**Problem RC11** (Kocis and Grossmann [44])

Minimize:

$$f(\mathbf{x}) = 7.5x_7 + 5.5x_8 + 7x_5 + 6x_6 + 5(x_1 + x_2)$$

subject to:

$$g_1(\mathbf{x}) = x_5 - 10x_7 \leq 0$$
$$g_2(\mathbf{x}) = x_6 - 10x_8 \leq 0$$
$$g_3(\mathbf{x}) = x_1 - 20x_7 \leq 0$$
$$g_4(\mathbf{x}) = x_2 - 20x_8 \leq 0$$
$$h_1(\mathbf{x}) = x_7 + x_8 - 1 = 0$$
$$h_2(\mathbf{x}) = x_3 + x_4 - 10 = 0$$
$$h_3(\mathbf{x}) = x_3 - 0.9(1 - e^{-0.5x_5})x_1 = 0$$
$$h_4(\mathbf{x}) = x_4 - 0.8(1 - e^{-0.4x_6})x_2 = 0$$
$$h_5(\mathbf{x}) = x_3x_7 + x_4x_8 - 10 = 0$$

with bounds:

$$0 \leq x_i \leq 100, i = 1, 2, \ldots, 6$$
$$x_7, x_8 \in \{0, 1\}$$

$f(\mathbf{x}^*) = 99.239635053646964$.

**Problem RC12** (Yuan et al. [89])

Minimize:

$$f(\mathbf{x}) = (1 - x_4)^2 + (2 - x_5)^2 + (1 - x_6)^2 - ln(1 + x_7) \\ +(1 - x_1)^2 + (2 - x_2)^2 + (3 - x_3)^2$$

subject to:

$$g_1(\mathbf{x}) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 - 5 \leq 0$$
$$g_2(\mathbf{x}) = x_6^2 + x_1^2 + x_2^2 + x_3^2 - 5.5 \leq 0$$
$$g_3(\mathbf{x}) = x_1 + x_4 - 1.2 \leq 0$$
$$g_4(\mathbf{x}) = x_2 + x_5 - 1.8 \leq 0$$
$$g_5(\mathbf{x}) = x_3 + x_6 - 2.5 \leq 0$$
$$g_6(\mathbf{x}) = x_1 + x_7 - 1.2 \leq 0$$
$$g_7(\mathbf{x}) = x_5^2 + x_2^2 - 1.64 \leq 0$$
$$g_8(\mathbf{x}) = x_6^2 + x_3^2 - 4.25 \leq 0$$
$$g_9(\mathbf{x}) = x_5^2 + x_3^2 - 4.64 \leq 0$$

with bounds:

$$0 \leq x_1 \leq 1.2$$
$$0 \leq x_2 \leq 1.8$$
$$0 \leq x_3 \leq 2.5$$
$$x_4, x_5, x_6, x_7 \in \{0, 1\}$$

$f(\mathbf{x}^*) = 4.579582402436706.$

**Problem RC13** (Cardoso et al. [15], Himmelblau [38])

Minimize:

$$f(\mathbf{x}) = 5.357854x_1^2 - 40792.141 + 37.29329x_4 + 0.835689x_4x_3$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= a_3x_4x_2 + a_1 + a_2x_5x_4 - a_4x_1x_3 - 92 \leq 0 \\ g_2(\mathbf{x}) &= a_7x_4x_5 + a_5 + a_6x_5x_3 + a_8x_1^2 - 110 \leq 0 \\ g_3(\mathbf{x}) &= a_9 + a_{11}x_4x_1 + a_{10}x_1x_3 + a_{12}x_1x_2 - 25 \leq 0 \end{aligned}$$

where

| | | |
|---|---|---|
| $a_1 = 85.334407$ | $a_5 = 80.51249$ | $a_9 = 9.300961$ |
| $a_2 = 0.0056858$ | $a_6 = 0.0071317$ | $a_{10} = 0.0047026$ |
| $a_3 = 0.0006262$ | $a_7 = 0.0029955$ | $a_{11} = 0.0012547$ |
| $a_4 = 0.0022053$ | $a_8 = 0.0021813$ | $a_{12} = 0.0019085$ |

with bounds:

$$\begin{aligned} &27 \leq x_1, x_2, x_3 \leq 45 \\ &x_4 \in \{78, 79, \ldots, 102\} \\ &x_5 \in \{33, 34, \ldots, 45\} \end{aligned}$$

$f(\mathbf{x}^*) = -32217.4310371.$

**Problem RC14** (Grossmann and Sargent [26])

Minimize:

$$f(\mathbf{x}) = 250(x_1x_4^{0.6} + x_2x_5^{0.6} + x_3x_6^{0.6})$$

subject to:

$$
\begin{aligned}
g_1(\mathbf{x}) &= \frac{40000x_7}{x_9} + \frac{20000x_8}{x_{10}} - 6000 \leq 0 \\
g_2(\mathbf{x}) &= 8 - x_1x_7 \leq 0 \\
g_3(\mathbf{x}) &= 20 - x_2x_7 \leq 0 \\
g_4(\mathbf{x}) &= 8 - x_3x_7 \leq 0 \\
g_5(\mathbf{x}) &= 16 - x_1x_8 \leq 0 \\
g_6(\mathbf{x}) &= 4 - x_2x_8 \leq 0 \\
g_7(\mathbf{x}) &= 4 - x_3x_8 \leq 0 \\
g_8(\mathbf{x}) &= -x_4 + 2x_9 \leq 0 \\
g_9(\mathbf{x}) &= -x_5 + 3x_9 \leq 0 \\
g_{10}(\mathbf{x}) &= -x_6 + 4x_9 \leq 0 \\
g_{11}(\mathbf{x}) &= -x_4 + 4x_{10} \leq 0 \\
g_{12}(\mathbf{x}) &= -x_5 + 6x_{10} \leq 0 \\
g_{13}(\mathbf{x}) &= -x_6 + 3x_{10} \leq 0
\end{aligned}
$$

with bounds:

$$
\begin{aligned}
& x_1, x_2, x_3 \in \{1, 2, 3\} \\
& 250 \leq x_4, x_5, x_6 \leq 2500 \\
& 6 \leq x_7 \leq 20 \\
& 4 \leq x_8 \leq 16 \\
& 40 \leq x_9 \leq 700 \\
& 10 \leq x_{10} \leq 450
\end{aligned}
$$

$f(\mathbf{x}^*) = 38499.46511672663$.

# Appendix B

## Additional Experimental Figures

This section provides supplementary figures for the experimental results discussed in Chapter 4.

Figure B.1 shows for each algorithm, the proportion of successful runs across 21 runs on a set of CEC 2020 problems within tolerance $\delta = 10^{-8}$ and target accuracies $\epsilon \in \{10^{-4}, 10^{-8}\}$. The left plot of Figure B.1 shows that across seven continuous optimization problems RC01 through RC07, SASS locates globally optimal solutions in 43 percent of runs for target accuracy $\epsilon = 10^{-4}$ and in 21 percent of runs for $\epsilon = 10^{-8}$. The corresponding rates for COLSHADE are 15 percent and 16 percent, and for sCMAgES are 27 percent and 16 percent. MIDACO cannot achieve globally optimal solutions for any continuous problems within both target accuracies and the tightest constraint tolerance even in a single run. The right plot of Figure B.1 shows the percentage for each algorithm across seven mixed-integer optimization problems RC08 through RC14. The corresponding rates within both target accuracies for SASS are 71 percent, for sCMAgES are 71 percent, for COLSHADE are 53 percent, and for MIDACO are 58 percent of runs and 56 percent of runs with $\epsilon = 10^{-4}$ and $\epsilon = 10^{-8}$ respectively.

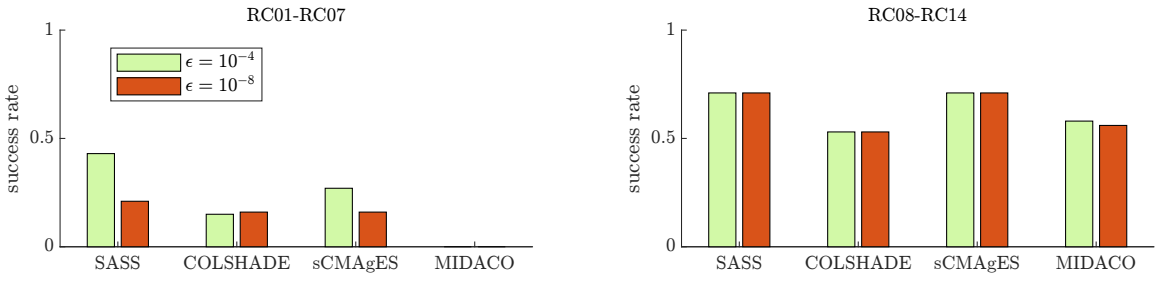


Figure B.1: Percentage of runs to reach globally optimum on the CEC 2020 problems for SASS, COLSHADE, sCMAgES, and MIDACO with constraint tolerance $\delta = 10^{-8}$.

Figure B.2 indicates the performance of MIDACO varies with different constraint

tolerances and target accuracies. MIDACO performs better for both continuous problems and mixed-integer problems when the tolerance for constraints is not very strict, compared to when the tolerance is set to be very small. One possible reason is that smaller tolerances can potentially lead to smaller search spaces, making it more difficult for MIDACO to solve these problems, and vice versa.

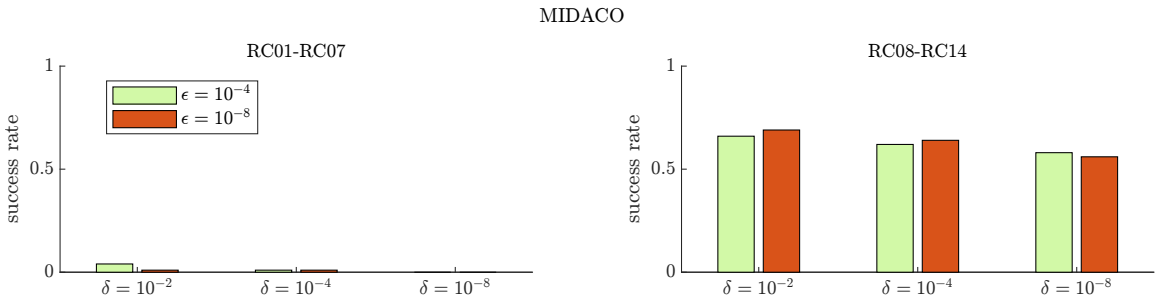


Figure B.2: Percentage of runs to reach globally optimum on the CEC 2020 problems for MIDACO with constraint tolerances $\delta \in \{10^{-2}, 10^{-4}, 10^{-8}\}$ and target accuracies $\epsilon \in \{10^{-4}, 10^{-8}\}$.

ECDF plots are given in Figure B.3. We select twenty logarithmically uniformly distributed objective function values between $f^* + 10^{-8}\,|f^*|$ and $f^* + 10^{0}\,|f^*|$ as targets for each problem, where $f^*$ is the optimal solution value of the problem. The plots show the percentage of reached objective function value targets for each problem across 21 runs against the number of objective function evaluations. In order for a target to be considered as reached, the corresponding solution generated by the algorithm needs to be feasible within tolerance $\delta$. In some cases, MIDACO can be observed to achieve a larger number of targets with the strictest tolerance and target accuracy compared to other settings within the same number of function evaluations. MIDACO does not reach any targets with the maximum function evaluation limit for the high-dimensional problems RC06 and RC07, regardless of the tolerance settings used. Moreover, the initial slow increase in the fraction of reached targets and the rapid increase afterward may indicate that exploiting a restart scheme could be helpful in exploring different regions of the search space and improving the performance. By generating new initial points, MIDACO may be able to find better solutions more quickly.

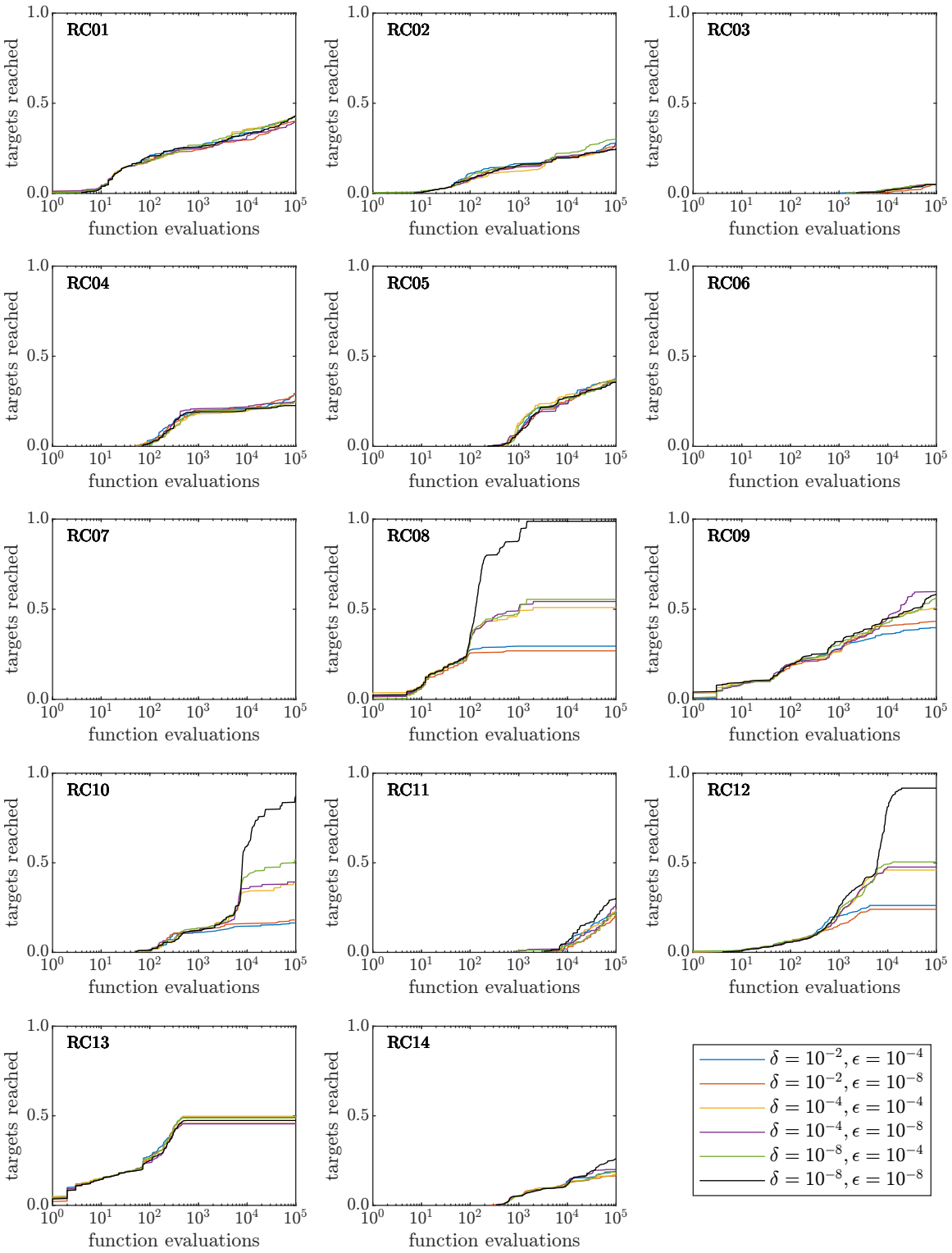


Figure B.3: ECDF plots on the CEC 2020 problems for MIDACO with constraint tolerances $\delta \in \{10^{-2}, 10^{-4}, 10^{-8}\}$.

## Bibliography


[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A global optimization method, $\alpha$BB, for general twice-differentiable constrained NLPs—I. Theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998.

[3] A. Aggarwal and C. A. Floudas. Synthesis of general distillation sequences—nonsharp separations. *Computers & Chemical Engineering*, 14(6):631–653, 1990.

[4] R. Angira and B. V. Babu. Optimization of process synthesis and design problems: A modified differential evolution approach. *Chemical Engineering Science*, 61(14):4707–4721, 2006.

[5] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP. Technical report, DIMACS, 1995.

[6] D. V. Arnold. An active-set evolution strategy for optimization with known constraints. In *Parallel Problem Solving from Nature–PPSN XIV: 14th International Conference*, pages 192–202. Springer, 2016.

[7] D. V. Arnold. Reconsidering constraint release for active-set evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 665–672, 2017.

[8] T. Bäck and M. Schütz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 33–51. MIT Press, 1995.

[9] M. Bénichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[10] B. Borchers and J. E. Mitchell. An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers & Operations Research*, 21(4):359–367, 1994.

[11] P. A. Bosman, J. Grahl, and D. Thierens. Benchmarking parameter-free amalgam on functions with and without noise. *Evolutionary Computation*, 21(3):445–469, 2013.

[12] J. Bracken and G. P McCormick. Selected applications of nonlinear programming. Technical report, Wiley, 1968.

[13] R. Breu and C. A. Burdet. Branch and bound experiments in zero-one programming. In M. L. Balinski, editor, *Approaches to Integer Programming*, volume 2, pages 1–50. Springer Berlin Heidelberg, 1974.

[14] M. R. Bussieck, A. S. Drud, and A. Meeraus. MINLPLib—A collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.

[15] M. Cardoso, R. Salcedo, S. Feyo De Azevedo, and D. Barbosa. A simulated annealing approach to the solution of MINLP problems. *Computers & Chemical Engineering*, 21(12):1349–1364, 1997.

[16] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.

[17] K. Deep, K. P. Singh, M. L. Kansal, and C. Mohan. A real coded genetic algorithm for solving integer and mixed integer optimization problems. *Applied Mathematics and Computation*, 212(2):505–518, 2009.

[18] R. S. Dembo. A set of geometric programming test problems and their solutions. *Mathematical Programming*, 10(1):192–213, 1976.

[19] S. L. Digabel and S. M. Wild. A taxonomy of constraints in simulation-based optimization. *arXiv:1505.07881*, 2015.

[20] O. Exler, T. Lehmann, and K. Schittkowski. MISQP: A Fortran implementation of a trust region SQP algorithm for mixed-integer nonlinear programming-user's guide. Technical report, Department of Computer Science, University of Bayreuth, Germany, 2010.

[21] O. Exler and K. Schittkowski. A trust region SQP algorithm for mixed-integer nonlinear programming. *Optimization Letters*, 1(3):269–280, 2007.

[22] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, 11 1995.

[23] C. A. Floudas and A. Aggarwal. A decomposition strategy for global optimum search in the pooling problem. *ORSA Journal on Computing*, 2(3):225–235, 1990.

[24] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Springer Verlag, Berlin, Heidelberg, 1990.

[25] J. Forrest, J. Hirst, and J. A. Tomlin. Practical solution of large mixed integer programming problems with umpire. *Management Science*, 20(5):736–773, 1974.

[26] I. E. Grossmann and R. W. Sargent. Optimum design of multipurpose chemical plants. *Industrial & Engineering Chemistry Process Design and Development*, 18(2):343–348, 1979.

[27] O. K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[28] J. Gurrola-Ramos, A. Hernàndez-Aguirre, and O. Dalmau-Cedeño. COLSHADE for real-world single-objective constrained optimization problems. In *2020 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2020.

[29] R. Hamano, S. Saito, M. Nomura, and S. Shirakawa. CMA-ES with margin: Lower-bounding marginal probability for mixed-integer black-box optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 639–647, 2022.

[30] L. Han-Lin. An approximate method for local optima for nonlinear mixed integer programming problems. *Computers & Operations Research*, 19(5):435–444, 1992.

[31] L. Han-Lin and C. Chih-Tan. A global approach for nonlinear mixed discrete programming in design optimization. *Engineering Optimization*, 22(2):109–122, 1993.

[32] N. Hansen. A CMA-ES for Mixed-Integer Nonlinear Optimization. Research Report RR-7751, INRIA, October 2011.

[33] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[34] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317. IEEE, 1996.

[35] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[36] N. Hansen, R. Ros, N. Mauny, M. Schoenauer, and A. Auger. Impacts of invariance in search: When CMA-ES and PSO face ill-conditioned and non-separable problems. *Applied Soft Computing*, 11(8):5755–5769, 2011.

[37] C. A. Haverly. Studies of the behavior of recursion for the pooling problem. *SIGMAP Bulletin*, (25):19–28, Dec 1978.

[38] D. M. Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill, 1972.

[39] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*. Springer Verlag, Berlin, Heidelberg, 1981.

[40] T. Ibaraki. Theoretical comparisons of search strategies in branch-and-bound algorithms. *International Journal of Computer & Information Sciences*, 5:315–344, 1976.

[41] S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Natural Computing*, 3:77–112, 2004.

[42] G. R. Kocis and I. E. Grossmann. Relaxation strategy for the structural optimization of process flow sheets. *Industrial & Engineering Chemistry Research*, 26(9):1869–1880, 1987.

[43] G. R. Kocis and I. E. Grossmann. Global optimization of nonconvex mixed-integer nonlinear programming (MINLP) problems in process synthesis. *Industrial & Engineering Chemistry Research*, 27(8):1407–1421, 1988.

[44] G. R. Kocis and I. E. Grossmann. A modelling and decomposition strategy for the MINLP optimization of process flowsheets. *Computers & Chemical Engineering*, 13(7):797–819, 1989.

[45] A. Kumar, S. Das, and I. Zelinka. A modified covariance matrix adaptation evolution strategy for real-world constrained optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 11–12, 2020.

[46] A. Kumar, S. Das, and I. Zelinka. A self-adaptive spherical search algorithm for real-world constrained optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 13–14, 2020.

[47] A. Kumar, G. Wu, M. Z. Ali, R. Mallipeddi, Ponnuthurai N. Suganthan, and S. Das. A test-suite of non-convex constrained optimization problems from the real-world and some baseline results. *Swarm and Evolutionary Computation*, 56:100693, 2020.

[48] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[49] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[50] T. Lehmann, K. Schittkowski, and T. Spickenreuther. MIQL: A Fortran subroutine for convex mixed-integer quadratic programming-user's guide. Technical report, Department of Computer Science, University of Bayreuth, Germany, 2010.

[51] S. Leyffer. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, 18:295–309, 2001.

[52] R. Li, M. T. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):29–64, 2013.

[53] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. C. Coello, and K. Deb. Problem definitions and evaluation criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. *Journal of Applied Mechanics*, 41(8):8–31, 2006.

[54] T. W. Liao. Two hybrid differential evolution algorithms for engineering design optimization. *Applied Soft Computing*, 10(4):1188–1199, 2010.

[55] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989, 1963.

[56] M. Manousiouthakis and D. Sourlas. A global optimization approach to rationally constrained rational programming. *Chemical Engineering Communications*, 115(1):127–147, 1992.

[57] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976.

[58] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[59] MINLPLib. A library of mixed-integer and continuous nonlinear programming instances. https://www.minlplib.org/. Last Accessed on December 22, 2022.

[60] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[61] J. Nocedal and S. Wright. *Numerical Optimization*. Springer New York, 2006.

[62] R. Polakova. L-SHADE with competing strategies applied to constrained optimization. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1683–1689. IEEE, 2017.

[63] I. Rechenberg. *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Friedrich Frommann Verlag, 1973.

[64] K. A. Rinnooy. Towards global optimization methods (I and II). *Mathematical Programming*, 39:27–78, 1987.

[65] G. Rudolph. An evolutionary algorithm for integer programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature — PPSN III*, pages 139–148. Springer Berlin Heidelberg, 1994.

[66] H. S. Ryoo and N. V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19(5):551–566, 1995.

[67] K. L. Sadowski, P. A. Bosman, and D. Thierens. A clustering-based model-building EA for optimization problems with binary and real-valued variables. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 911–918, 2015.

[68] K. L. Sadowski, D. Thierens, and P. A. Bosman. GAMBIT: A parameterless model-based evolutionary algorithm for mixed-integer problems. *Evolutionary Computation*, 26(1):117–143, 2018.

[69] N. Sakamoto and Y. 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.

[70] N. Sakamoto and Y. Akimoto. Adaptive ranking-based constraint handling for explicitly constrained black-box optimization. *Evolutionary Computation*, 30(4):503–529, 2022.

[71] R. Sauer, A. Colville, and C. Burwick. Computer points way to more profits. *Hydrocarbon Processing*, 84(2), 1964.

[72] K. Schittkowski. QL: A Fortran code for convex quadratic programming-user's guide. Technical report, Department of Computer Science, University of Bayreuth, Germany, 2005.

[73] K. Schittkowski. A collection of 200 test problems for nonlinear mixed-integer programming in Fortran-user's guide. Technical report, Department of Computer Science, University of Bayreuth, Germany, 2015.

[74] M. Schlüter, J. A. Egea, and J. R. Banga. Extended ant colony optimization for non-convex mixed integer nonlinear programming. *Computers & Operations Research*, 36(7):2217–2229, 2009.

[75] M. Schlüter and M. Gerdts. The oracle penalty method. *Journal of Global Optimization*, 47:293–325, 2010.

[76] M. Schlüter, M. Gerdts, and J. J. Rückmann. A numerical study of MIDACO on 100 MINLP benchmarks. *Optimization*, 61(7):873–900, 2012.

[77] M. Schlüter. *Nonlinear mixed integer based optimization technique for space applications*. PhD thesis, University of Birmingham, 2012.

[78] M. Schlüter and M. Munetomo. Parallelization strategies for evolutionary algorithms for MINLP. In *2013 IEEE Congress on Evolutionary Computation*, pages 635–641. IEEE, 2013.

[79] M. Schlüter and M. Munetomo. Numerical assessment of the parallelization scalability on 200 MINLP benchmarks. In *2016 IEEE Congress on Evolutionary Computation*, pages 830–837. IEEE, 2016.

[80] M. Schlüter and M. Munetomo. MIDACO parallelization scalability on 200 MINLP benchmarks. *Journal of Artificial Intelligence and Soft Computing Research*, 7(3):171–181, 2017.

[81] H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Birkhäuser Basel, 1977.

[82] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173, 2008.

[83] P. Spettel, Z. Ba, and D. V. Arnold. Active sets for explicitly constrained evolutionary optimization. *Evolutionary Computation*, 30(4):531–553, 04 2022.

[84] P. Spettel, H.-G. Beyer, and M. Hellwig. A covariance matrix self-adaptation evolution strategy for optimization under linear constraints. *IEEE Transactions on Evolutionary Computation*, 23(3):514–524, 2018.

[85] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of North American Fuzzy Information Processing*, pages 519–523. IEEE, 1996.

[86] R. Storn and K. Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341, 1997.

[87] D. Thierens. The linkage tree genetic algorithm. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, pages 264–273, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[88] A. W. Westerberg and J. V. Shah. Assuring a global optimum by the use of an upper bound on the lower (dual) bound. *Computers & Chemical Engineering*, 2(2-3):83–92, 1978.

[89] X. Yuan, S. Zhang, .L Pibouleau, and S. Domenech. A mixed-integer nonlinear-programming method for process design. *RAIRO-Recherche Opérationnelle-Operations Research*, 22(4):331–346, 1988.

[90] Y. X. Yuan. On the convergence of a new trust region algorithm. *Numerische Mathematik*, 70(4):515–539, 1995.