Improved Placement of Local Solver Launch Points for Large-scale Global Optimization

by

Laurence Smith, MASc., BASc.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

April 2011

Your file Votre référence
ISBN: 978-0-494-81548-9
Our file Notre référence
ISBN: 978-0-494-81548-9

The undersigned recommend to
the Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Improved Placement of Local Solver Launch Points for Large-scale Global Optimization

submitted by

Laurence Smith, MASc., BASc.

in partial fulfillment of the requirements for

the degree of Doctor of Philosophy in Electrical and Computer Engineering

---

Chair, Howard Schwartz, Department of Systems and Computer Engineering

---

Thesis Supervisor, Victor Aitken, Department of Systems and Computer Engineering

---

Thesis Supervisor, John Chinneck, Department of Systems and Computer Engineering

---

External Examiner, Dominique Orban, Mathematics and Industrial Engineering
Department, Ecole Polytechnique de Montreal

Carleton University
April 2011

Abstract

Nonlinear programming (NLP) solvers are extremely sensitive to their starting location. NLP solvers find optima quickly when launched from certain locations, yet they fail or progress very slowly when launched from other locations in the same model. Furthermore, it is unknown if the optima returned by NLP solvers are local or global. Constraint Consensus (CC) is an algorithm that is used to quickly improve potential solver launch points. This thesis explores two avenues of research that use CC to explore the variable space in order to find promising NLP solver launch points for global optimization.

First, advances to the CC method are explored. New techniques for the feasibility and consensus vector calculations are proposed and the concept of augmentation is introduced. An investigation of successful launch points reveals other modifications that enhance the performance of the CC method.

Second, a heuristic multistart framework for global optimization is proposed. It uses CC to quickly search the variable space. Randomly sampled start points are concentrated near promising areas of the variable space with CC before a clustering algorithm is applied to select a set of solver launch points. The key to the process is a novel method for analyzing the inter-point distance distribution of the concentrated points in order to extract a critical distance for the clustering routine.

Numerical results are presented to show that the proposed methods are effective compared to the state of the art when applied to a variety of large-scale nonlinear models.

Acknowledgements

It is a pleasure to thank everyone who played a role in making this thesis possible. I owe my deepest gratitude to Vic Aitken and John Chinneck who supervised my thesis work from beginning to end. Vic helped guide me through many of the mathematical difficulties I encountered, and after only five short years John successfully taught me why suggesting the global optimum as a start point is so funny. I would like to thank the technical support staff: Naren, Danny, Daren, Stephen, and Jerry, for their help with my never ending computer problems and for returning my hockey stick in one piece after using it to poke holes in ceiling tiles that were collecting water after a storm. I am also grateful to the office staff, particularly Darlene, Coleen, and Jennifer, who have helped me since I started my thesis. I do not want to admit how many times they reminded me to register and pay tuition. I would like to thank my office mate, Daanish. Our conversations about economics and the valuation of mining sector companies motivated me to come into the office every day. More importantly, Daanish showed me the value of including spicy eggplant on a sandwich. I would like to thank the Snowflakes hockey organization for drafting me and giving me a chance, in particular, Greg and Steph who taught me a thing or two about hockey. I would also like to thank Jason for organizing the team and the socials after the games, and Hank, for patiently listening to Jason and me argue about the Renminbi, silver, and housing bubbles. Finally, I would like to thank all my friends (especially the Ottawans) and family for all their support. In particular, I would like to thank Roz for always being there when I needed her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over time engineers have learned to describe physical plants and industrial processes mathematically in terms of algebraic and/or differential equations known as models. Recent developments in computer technology and numerical methods have led to a variety of tools that allow engineers to solve models in order to simulate designs in a safe, fast, and relatively inexpensive manner without any physical construction. Computer simulation also allows engineers to try various sets of parameters in order to increase the effectiveness and efficiency of their products.

A natural extension to the simulation work is to determine how to change a model's parameters in order to optimize some objective. For instance, an aerospace engineer may be interested in adjusting a rocket's design in order to find the smallest vehicle that can fulfill a mission's requirements [92]. The results from optimization may offer new insight into the underlying plants or processes being studied.

Advancing technology allows engineers to consider increasingly larger applications in greater detail. Engineering models are often highly nonlinear, especially in their constraint sets, and it is extremely difficult to identify where a feasible region or point is. Consequently, optimization algorithms must continue to advance to keep up with industrial needs. This thesis investigates new methods for seeking feasibility, finding disjoint feasible regions, and solving global optimization models, specifically, large-scale highly-constrained continuous nonlinear programs.

Global optimization solvers try to determine the absolute minimum or maximum feasible solution to a given problem. This is an important distinction from nonlinear programming (NLP) solvers that calculate local optima. Global optimization algorithms are often divided into two categories: *deterministic* methods and *probabilistic* methods. Deterministic methods attempt to prove that a particular solution vector is the global optimum. They

do this more or less by ruling out all areas in the problem until only the global optima are left. For many problems this is a very time consuming process and often an exact global optimum is not actually required. In practice, a solution that is close to optimal is often all that is needed. For these models, probabilistic methods are used to quickly find a solution. Probabilistic methods often use heuristics to decrease the search time and do not guarantee that the solution found is the global optimum. The research presented in this thesis deals with probabilistic methods, specifically variations of *multistart*. Multistart algorithms launch a local solver from various locations throughout the search space.

Naive multistart is a basic probabilistic global optimization algorithm. Naive multistart simply launches a local solver repeatedly from randomly selected start points for a pre-specified number of iterations or until a time limit is reached. There are many reasons why this method is often ineffective in practice. Firstly, the random start points may be far enough from the feasible region to cause problems for the solver. Furthermore, the solver may be repeatedly launched within the same basin of attraction, hence, returning the same solution multiple times. Finally, due to the random assignment of the initial points there is no guarantee that the variable space will be adequately explored. Improved versions of the multistart method exist, for instance, MSNLP [56] and GLOBALm [85]. These algorithms employ techniques including advanced sampling, acceptance and rejection, and clustering methods in order increase the effectiveness of the multistart method.

The techniques proposed in this thesis increase the efficiency of the multistart method by providing a promising set of points from which a local solver is launched. The main difference between the proposed techniques and previous related research is how gradient-based repair methods are used to aid the search.

Constraint Consensus (CC) is a gradient-based repair method that is used to quickly find a point close to a feasible region. Ibrahim and Chinneck [42] show that a combined method (CC and a local solver) finds feasible solutions more often than launching local NLP solvers alone. More recently, MacLeod [58] combined CC and multistart with a voting routine into a feasibility seeking algorithm known as multistart CC. MacLeod's algorithm effectively increases the frequency at which local solvers find feasible solutions.

This thesis investigates a probabilistic optimization algorithm that combines a new variant of CC with a clustering method in order to conduct an efficient search for a global optimum solution. The method keeps track of a population of points, each of which is improved with CC. CC concentrates the points in areas of the variable space that minimize the constraint violations. These are promising locations because they are close to feasibility. A hierarchical clustering technique is applied to the concentrated set of points to identify

subsets of points likely to be in the same basins of attraction, which ultimately reduces the number of redundant NLP solver launches.

The techniques proposed herein are heuristics whose purpose is to speed up the process of finding high quality solutions to large-scale optimization problems where an exhaustive search is frequently impractical. These probabilistic algorithms are intended to work well in general and to be useful for a wide range of optimization models. Mathematical convergence proofs are of little practical value because it is the algorithms' performances on actual problems which matters. For this reason, the effectiveness of heuristic methods, including such popular metaheuristic optimizers as genetic algorithms [63] and particle swarm optimization [52], is normally proved by empirical testing on realistic problems. In the tradition of other major heuristics, this thesis uses extensive numerical experimentation to show that the new methods are often faster and more robust at finding solutions to large-scale global optimization problems than current state of the art algorithms.

1.1 Overview

Chapter 2 reviews nonlinear programming, its challenges, and the basic strategies commonly used to solve global optimization problems. The state of the art is presented in Chapter 3. Specifically, details about selecting an initial sample, seeking feasibility, clustering, and various multistart algorithms are addressed. The thesis statement is provided in Chapter 4. Chapters 5, 6, and 7 provide a detailed description of the improved CC algorithm used to quickly and effectively seek feasibility, a gradient-based concentration scheme for locating promising solver launch points, and a complete method for global optimization, respectively. Experimental results and an evaluation of the performances are also presented in these chapters. Finally, Chapter 8 summarizes the conclusions and academic contributions of this dissertation and proposes avenues for future research.

Chapter 2

A Brief Review of Global Optimization

This chapter introduces the basic ideas and definitions related to NLP and global optimization. The optimization problem is formally introduced along with many of its associated challenges. Common global optimization techniques are discussed, including complete and probabilistic methods. The final section covers various ways to deal with constraints.

2.1 Nonlinear Programming and Notation

The goal of mathematical programming is to find an optimal solution to a problem. In practice there is usually an objective function that must be minimized while a set of constraints is satisfied. More formally, the general NLP problem is

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) && (2.1)\\ \text{s. t.} \quad & g_i(\mathbf{x})\{\leq, =\}0, \ \ \forall i \in I \doteq \{1, 2, ..., m\} && (2.2)\\ & \ell_j \leq x_j \leq u_j, \ \ \forall j \in J \doteq \{1, 2, ..., n\} && (2.3) \end{aligned}$$

where $\mathbf{x} \in \mathcal{R}^n$ is an n dimensional solution vector. The *objective function* $f : \mathcal{R}^n \rightarrow \mathcal{R}$ is a continuously differentiable function. It is called the objective function because the objective of the problem is to minimize $f(\mathbf{x})$ with respect to $\mathbf{x}$. Minimizing an objective

function, $f(\mathbf{x})$, is equivalent to maximizing the negative of the objective function, i.e.,

$$\min f(\mathbf{x}) = -\max\ -f(\mathbf{x}). \tag{2.4}$$

The terms maximize and minimize are sometimes used interchangeably in general discussions of optimization. When possible, discussions herein will refer to finding the optimum instead of maximum or minimum points. Also note that all the variables are continuous. Models with this characteristic are considered *continuous models* and should not be confused with binary, mixed-integer, or any other kind of model where some or all of the variables are restricted to a set of integer values.

The m continuously differentiable functions $g_i(\mathbf{x}) : \mathcal{R}^n \to \mathcal{R}$ are the *constraints*. They are either equality or inequality constraints. For a model to be considered nonlinear, at least one of its constraints and/or the objective must be a nonlinear function. The *bound constraints* are listed in Eqn 2.3 where ℓ_j and u_j are the lower and upper bounds on x_j, respectively. If $I = \emptyset$ and all the variables are unbounded the model is *unconstrained*, if $I = \emptyset$ and at least one variable is bounded the model is *bound constrained*, and if $I \neq \emptyset$ the model is considered *constrained*, regardless of the variable bounds. The work presented in Chapters 5, 6, and 7 deals only with constrained models. Additionally, at least one constraint is nonlinear in each of the considered models.

The gradient of a constraint is represented by the vector $\nabla g_i(\mathbf{x}) \in \mathcal{R}^{1\times n}$ and the Hessian of a constraint is represented by the matrix $\nabla^2 g_i(\mathbf{x}) \in \mathcal{R}^{n\times n}$. The violation of an equality constraint is defined as the absolute value of the constraint

$$v_i \doteq |g_i(\mathbf{x})|. \tag{2.5}$$

The violation of an inequality constraint is the greater of the constraint function value and zero

$$v_i \doteq \max\{0, g_i(\mathbf{x})\}. \tag{2.6}$$

The *search space* $\mathcal{S}$ is defined as

$$\mathcal{S} \doteq \{\mathbf{x} \in \mathcal{R}^n \mid \forall j \in J:\ \ell_j \leq x_j \leq u_j\}, \tag{2.7}$$

and the *feasible region* $\mathcal{F}$ is defined as

$$\mathcal{F} \doteq \{\mathbf{x} \in \mathcal{S} \mid \forall i \in I:\ g_i(\mathbf{x})\{\leq, =\}0\}. \tag{2.8}$$

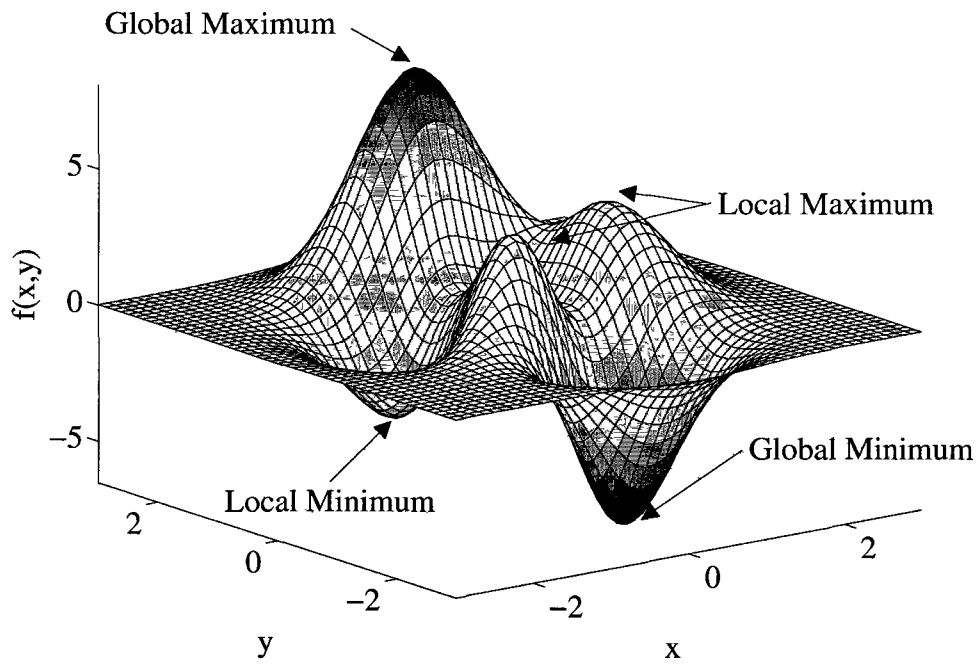


Figure 2.1: An example illustrating local and global optima. The figure was generated by the *peaks* function of Matlab.

Note that $\mathcal{F} \subseteq \mathcal{S}$ and that all optimal solution vectors $\mathbf{x}^*$ are in the feasible region, $\mathbf{x}^* \in \mathcal{F}$.

There are two types of optimal solutions: local and global. A local solution is the best solution in a sub-region of $\mathcal{F}$. Global solutions are the best solutions that exist in $\mathcal{F}$. Local solvers (also known as NLP solvers) are algorithms that attempt to find a locally optimal solution. The solution a local solver finds may or may not be the global optimum, however, a local solver cannot determine if a local solution is also a global solution. Feasibility seeking algorithms or repair methods ignore the objective function and attempt to find any solution vector in the feasible region. Global solvers attempt to find the global optimum. Complete global solvers try to identify with certainty the global optimum, while probabilistic global solvers cannot guarantee the solution is more than a local optimum. See Figure 2.1 for an illustration of various types of optimum points.

A *basin of attraction* is a region in which the launch of a particular local solver will lead to the same feasible region. The basins of attraction for a particular model may vary based on the solver. A *point of attraction* is a location in the variable space where a local minimum of the sum of the constraint violations exists. Care should be taken to distinguish

between these two related ideas. Furthermore, a solver launch is considered *redundant* if it returns the same solution that has already been found by an earlier launch.

Sometimes nonlinear programs are classified by their relative difficulty. A commonly used term to indicate tough problems is *large-scale*. There is no quantitative definition of what large-scale means in terms of NLP. It does imply, however, that the problem consists of many variables and/or constraints. Some large-scale problems are easy to solve so it is not necessarily a good measure of difficulty. For this thesis, the term *difficult* will be used to describe problems for which contemporary solvers cannot find optimum solutions in a relatively short period of time. This is not a very rigorous definition, nonetheless, it does not discriminate against lower dimensional problems or problems with few constraints. It is meant to include complex problems, large or small, that would be of interest to an industrial audience.

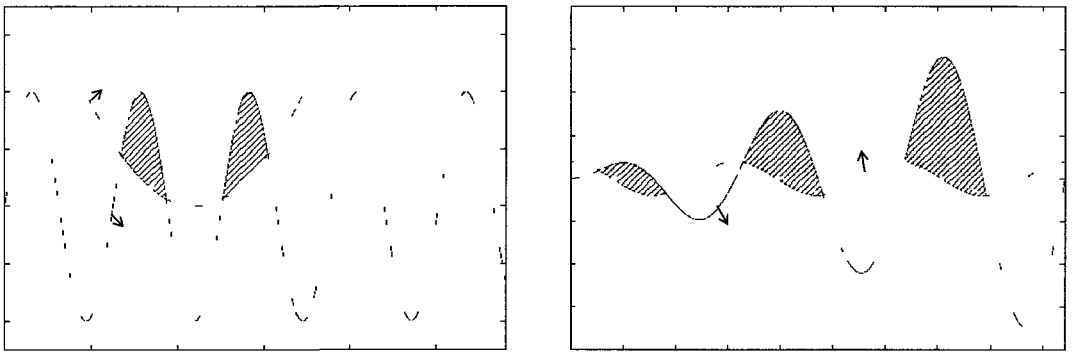
2.2 Challenges

Finding global optima of NLP models is a topic of great interest for both practitioners and academics. The value of global optimization to a plethora of industries covering a wide variety of topics continues to keep it relevant to practitioners. The difficulties intrinsically ingrained into the optimization problems due to their nonlinear elements provide academics with interesting research. Some specific challenges are listed below.

- **Disconnected Feasible Set**: Due to the nonlinear nature of the functions, the feasible region is not necessarily a connected set, hence, multiple feasible regions may exist adding to the difficulty in finding the global optimum. Refer to Figure 2.2 for illustrated examples.

- **Multiple Optima**: Another great challenge of NLP is that the problems can exhibit local optima. It is difficult to distinguish between local and global optima and computationally intensive to verify that a local optimum is a global optimum. Furthermore, local optimum points can lead solvers away from global optimum points. Issues pertaining to multiple and local optima are also present in the search for feasibility. For instance, algorithms that attempt to minimize constraint violations may be misled by local minima and become trapped in infeasible areas. An illustration of local and global optima is found in Figure 2.1.

- **Complicated Functions**: The set of nonlinear functions includes many that are computationally expensive to evaluate. For instance, simply taking the square root of



(a) Two feasible regions. (b) Multiple feasible regions.

Figure 2.2: Sets of constraints leading to multiple feasible regions (shaded areas).

a number is often avoided in many computer codes as it is computationally expensive. In graphics intensive games where the square root function is frequently needed in order to find distances between objects, approximation schemes such as Newton's method are often utilized. The cost associated with many complicated functions may limit the number of evaluations a solver can make in a particular problem. Also, it may be expensive to find other commonly used information such as derivatives.

- **Misleading Constraints**: In certain circumstances nonlinear constraints are very misleading. Consider the case where two constraints bend towards each other but never quite overlap. Feasibility seeking algorithms may get trapped since moving away from the current infeasible point may increase the total constraint violation. It is difficult for algorithms operating in regions like these to determine that there are no feasible points close by. Due to the variety of nonlinear functions, no solver is expected to work well for all models.

Due to the difficulties listed above there is no optimization or feasibility seeking method that outperforms all others over all problems (this is known as the *No Free Lunch Theorem* [101]). Instead there are many techniques that are designed for specific problem types. In fact, solvers designed for the general case often use various methods in hopes that at least one will perform well. For instance, Knitro [13, 99] is a general purpose commercial NLP solver that makes use of three different algorithms, namely: the interior-point direct, interior-point conjugate gradient, and active set algorithms.

2.3 Common Global Optimization Techniques

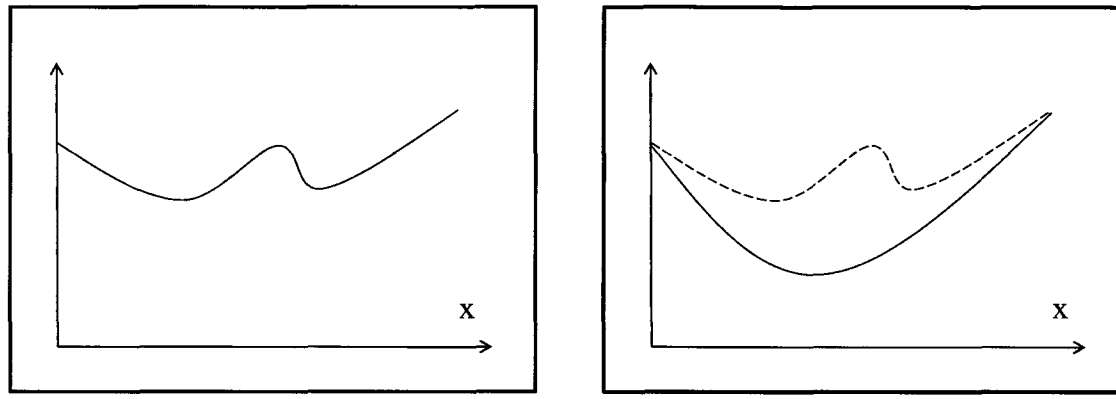
Many classes of algorithms exist to solve NLP models. The first major distinction is local versus global methods. Local solvers are commonly referred to as NLP solvers. Two examples are the commercial solver Knitro [13, 99], and the open source solver Ipopt [97, 98]. The goal of local solvers is to find local optima in NLP models. This is a huge distinction from global solvers, which attempt to find the absolute best solution. This thesis targets global optimization.

One way to classify global optimization algorithms is by the rigour with which they approach the goal. Two main categories are *deterministic* and *probabilistic*. Deterministic methods, also known as complete algorithms, guarantee to find an approximate global optimum, within prescribed tolerances, given an indefinitely long run time. This differs from probabilistic methods, which never verify if the solution found is the global optimum. Some problems require complete solutions; for instance, in safety verification problems, treating non-global extrema as worst cases may severely underestimate the true risk [69]. However, there are many practical applications where finding the global optimum is desirable but not essential. In fact, often a good feasible point is satisfactory since it is an improvement over what is available without optimization. For these problems, probabilistic search methods are employed. Probabilistic methods use heuristic search techniques to explore the variable space. They cannot recognize when they have found the global optimum and there is no guarantee that they will find the global optimum given an indefinite run time. Probabilistic algorithms are often used for large-scale problems where complete searches may not be feasible [69].

2.3.1 Deterministic Algorithms

As mentioned above, deterministic methods guarantee to find approximate global optimum points within some prescribed tolerances given an indefinitely long run time. This section covers some of the common techniques used for complete search. More thorough reviews are found in [40, 69].

Two basic deterministic global optimization strategies for unconstrained models are *covering* and *zooming*. Covering methods sequentially evaluate the objective function throughout the variable space. For instance, grid search evaluates the objective function at points on finer and finer grids [69]. Covering methods require that the rate of change of the objective function is bounded. Alternatively, the zooming strategy cycles through a two-stage process. A target value is set for the objective function and the model is updated to reflect



(a) Arbitrary function of x. (b) Underestimation.

Figure 2.3: An example of underestimation using a convex function.

that the solution must be better than the target value. Once the target objective function value is achieved, a new target is set. Updating the target value makes the algorithm *zoom in* on the global optimum. These techniques are impractical for large-scale optimization. For instance, setting stopping conditions for the zooming method is a challenge [28] and the covering method requires a number of samples that grows exponentially with the dimension of the problem.

A concept used often by deterministic global optimization algorithms is *relaxation*. To *relax* a model means to modify it such that the new model has a tractable solution and so that it gives some information about the location of the original model's global optima [69]. An example is *underestimation*: a new objective function is introduced that is less than or equal to the original objective function at each point. Underestimation is illustrated in Figure 2.3. Figure 2.3a shows an arbitrary function of x. Figure 2.3b depicts a convex function that is an underestimation of the initial function. The minimum of the relaxed function is found relatively easily because it is convex. Furthermore, the minimum of the underestimated function is a lower bound on the original function.

Usually relaxation is done such that the modified function is convex [69]. There are a couple of good reasons for this: a local optimum of a convex function is a global optimum, and if the relaxed model is infeasible (it does not contain any feasible solutions) then the original model must also be infeasible. If the relaxed model is feasible then its solution is a lower bound on the original model. For examples and further discussion of relaxation, refer to [30, 59, 60].

Another key concept for complete search methods is *branching*. Branching is the princi-

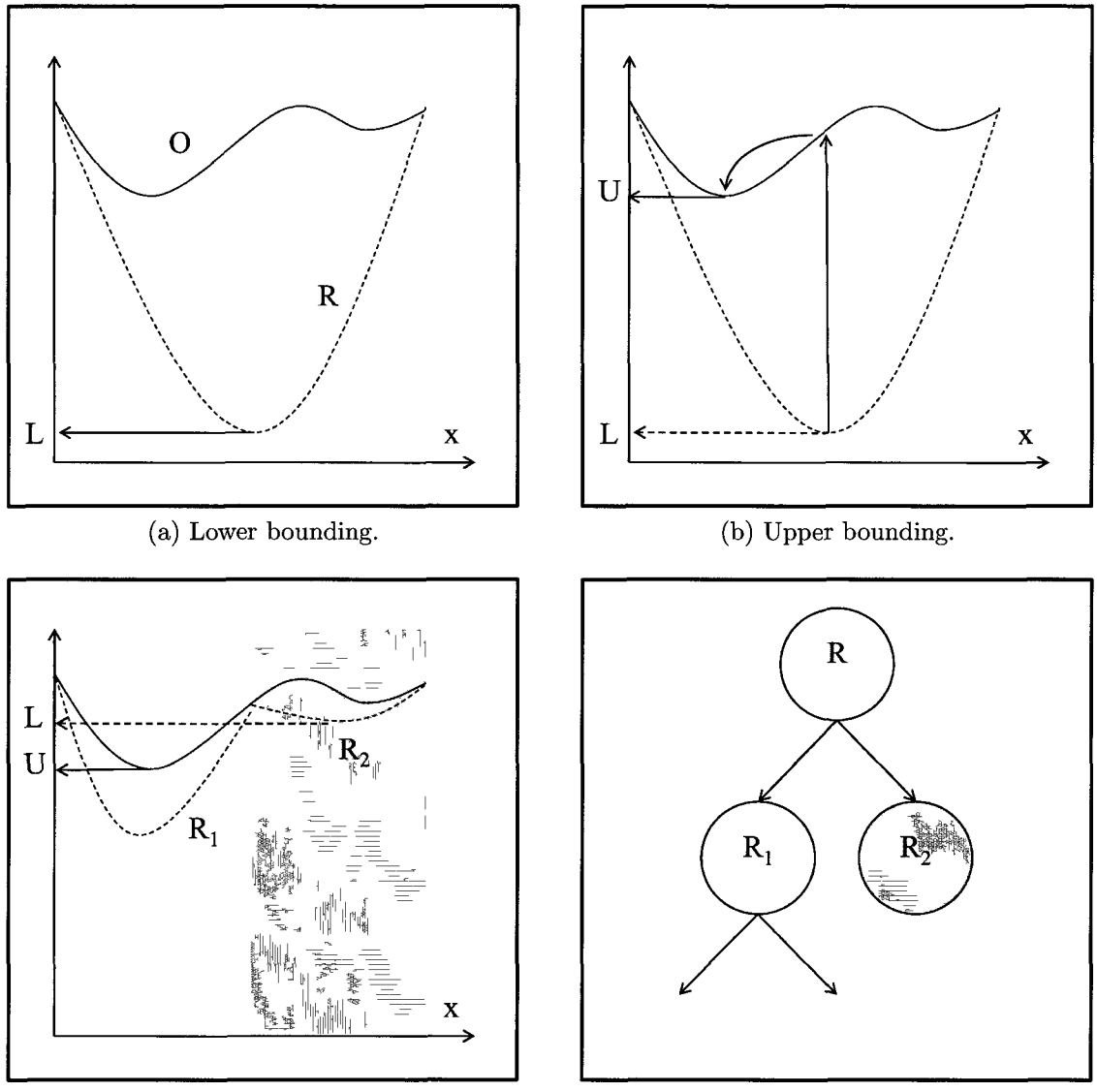
ple of recursively splitting the original problem into subproblems that are easier to solve (see Figure 2.4 for an example). There are two methods commonly used to increase the speed of the branching process: (i) Promising branches are split more frequently in hopes that they yield the best results; and (ii) Subproblem objective function bounds are compared in order to remove fruitless branches early in the search procedure.

Two global optimization software packages that use the concepts of branching and relaxation are The Branch and Reduce Optimization Navigator (BARON) [82, 89] and GlobSol [50]. BARON is a global optimization software based on the branch and bound method. Alternatively, the GlobSol software package is built upon an interval analysis routine. Both methods perform a complete search and can guarantee global optima to prescribed tolerances given an indefinitely long run time, nonetheless, both of these methods have weaknesses. Knowledge of the functions is necessary in order to get a good relaxation. Any solver that is based on relaxation will perform poorly when analytical information is limited, i.e., for black-box functions. Furthermore, the deterministic nature of the BARON and GlobSol solvers may lead to extremely long run times for certain models. For instance, some models may require BARON to search through a large number of branches in order to attain a solution. Large-scale models with many constraints and unbounded variables will emphasize this weakness. Comparing the two, GlobSol is less efficient than BARON with respect to time and solving capacity due to the level of mathematical rigour with which GlobSol attempts to solve the problems [70].

2.3.2 Probabilistic Algorithms

Probabilistic global search algorithms do not satisfy the complete search criteria. There is no guarantee that a probabilistic algorithm will find a global optimum. The goal of probabilistic methods is instead to find global optima as frequently as possible with the fewest computations possible. Many of the well known probabilistic algorithms are inspired by other fields of study such as metallurgy, genetics, and ornithology (the study of birds).

A connection between statistical mechanics and combinatorial optimization was first presented in 1983 by Kirkpatrick *et al.* in the seminal paper *Optimization by Simulated Annealing* [54]. The next year, Vanderbilt and Louie applied the simulated annealing (SA) method to continuous optimization problems [95]. The idea behind SA came from the process of annealing which refers to the controlled heating and cooling of materials to remove crystal defects and, in effect, change the strength and hardness of the material. This is done with the help of two processes. First, heating the material allows the atoms to move freely; then a cooling process gradually settles the atoms down into new, lower



(a) Lower bounding.

(b) Upper bounding.

(c) Domain subdivision.

(d) Search tree.

Figure 2.4: Using relaxation and branching for minimization: (a) Use the first convex relaxation, R, to find a lower bound, L, on the objective function, O. Start a local search from this value of x. (b) Use the result of the local search as an upper bound, U, on the objective function. (c) Divide into two subproblems each with its own convex relaxation. (d) Subdomain R_2 is removed from further search because its underestimated lower bound is greater than the objective function's upper bound. (Figure inspired by [89])

energy states.

SA improves upon an initial starting point by using a filtered random walk. Given a starting point, a second point is randomly generated from a transition distribution. If the objective function of the new point is lower than that of the current point, then the new point replaces the current point. If the new point has a higher objective function value than the current point, then it may or may not be accepted to replace the current point. The algorithm slowly decreases the probability that an inferior new point is accepted. Hence the method is allowed to search freely in the beginning but slowly becomes more constrained as it progresses, much like the annealing process. The change in the acceptance rate is known as a cooling schedule.

There are many variants of the SA algorithm. The differences often arise in how the methods choose the next candidate point and/or how the methods implement the cooling schedule. For example, Vanderbilt and Louie [95] proposed a method whereby a candidate point is generated as a random vector sampled from a hypercube distribution centered at the current point. This differs slightly from the method of Bohachevsky *et al.* [8] who proposed to generate a candidate point by sampling from a sphere centered at the current point. A popular implementation of adaptive simulated annealing is ASA [43].

Genetic algorithms (GAs) are population-based metaheuristic optimization methods. They use biologically inspired operators to manipulate a candidate solution population. Each member of the population is a candidate solution with a corresponding cost function value representing its relative *fitness* level as compared to other solutions or individuals in the population. The solution set is said to *evolve* as the operators are continually applied.

GAs are iterative algorithms that evolve a pool of candidate solutions of an optimization problem (or *individuals*) toward better solutions. Initially, the candidate solutions are generated either randomly or semi-randomly. At each iteration or *generation* a new subset of the candidate solutions is stochastically selected based on their fitness levels. The subset is modified by a set of biologically inspired operators such as *mutations* and/or *crossovers* to form a new population [24, 25, 63, 102]. The new population is then used in the next iteration of the algorithm. There are various stopping conditions for the algorithm. Some examples are capping the number of generations or termination when a satisfactory fitness level is reached.

Genocop (GEnetic algorithm for Numerical Optimization of COnstrained Problems) was first introduced and now maintained by Michalewicz *et al.* [64, 65, 66]. Originally, Genocop was designed to work with linear constraints only. The Genocop III release is an extension of the original system that is designed to work for both convex and nonconvex

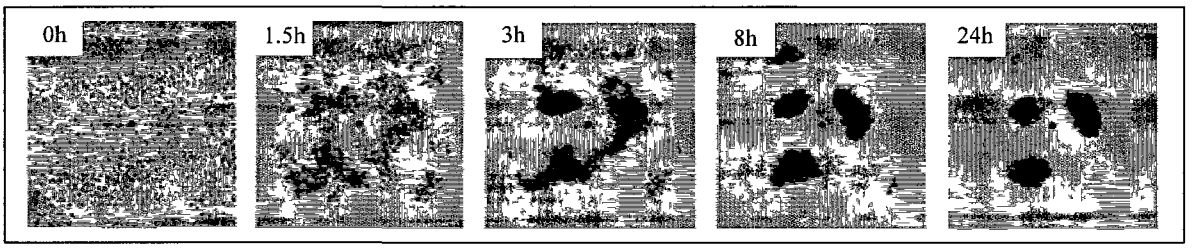


Figure 2.5: Spatio-temporal dynamics of corpse clustering by ants. The black dots are live ants and ant corpses. When left alone with the corpses, the live ants pile the corpses into groups. Time elapses from left (0h) to right (24h)(image from [32]).

sets of nonlinear constraints. Unlike the previous versions, Genocop III keeps track of two separate populations. The first population is feasible in the linear sense, that is, the individuals in the population satisfy all linear constraints. The individuals in this population are referred to as *search points*. The second population consists of fully feasible points that are called *reference points*. The algorithm attempts to repair the search points so that they are fully feasible by combining them with randomly selected reference points. The major problem with this algorithm is that it requires feasible points. For large-scale and/or difficult models feasible points may be hard to find.

Another popular population-based search algorithm for global optimization is particle swarm optimization (PSO) [21, 52, 57, 71, 75, 83]. PSO is inspired by swarm intelligence - the emergent collective intelligence of groups of simple agents. Swarm-based techniques are built upon the metaphor of social insects such as bees, ants, or termites, who as individuals perform rather simple tasks, yet as a whole complete large and complex tasks. For example, live ants are known to cluster the corpses of dead ants as shown in Figure 2.5. Strengths of swarm-based methods include: distribution, simplicity, flexibility, and robustness [9]. For instance, Schutte *et al.* present a parallel implementation of the PSO algorithm that can be run on a computer cluster [84].

PSO was first introduced as a method to optimize continuous nonlinear functions by Kennedy and Eberhart. Their seminal PSO paper *Particle Swarm Optimization* proposes a simple algorithm based on the social sharing of information that they claim is effective for optimizing a 'wide range' of functions [52]. The work is based on a flock of computer-simulated birds trying to find a cornfield. After initial experimentation, Kennedy and Eberhart found that a swarming behavior is more appropriate. They selected the term *particle* to describe the members of the populations instead of *point* because of the velocity parameter that is assigned to each member. This is similar to other research fields such

as computer graphics where the term *particle systems* is used to describe a method for modeling fuzzy objects such as fire, clouds, and water [78].

The basic idea behind the PSO algorithm is to use a set of particles to search the variable space with movements based on both *cognitive* and *social* aspects. The cognitive information is collected by the particle about its own path, while the social information is passed among the particles within the swarm. Only information from the most fit particle (the particle deemed by heuristic methods to be closest to the optimum location) is used as social information. The particles change their velocities based on both the cognitive and social information, hence the direction of search for each particle is biased in a promising direction [62].

There are two main ways to determine the social information and how it is shared. The first is a global best method in which all the particles communicate together in order to establish the most fit particle. The second method is a local routine in which the particles can only communicate with a subset of the population in order to determine the most fit particle locally. The second method emphasizes exploration of the variable space and reduces the chance of the algorithm getting stuck in a local minimum [51]. More details on various neighbourhood topologies are found in [44, 53, 62].

One of the unresolved issues with PSO is how to pick the parameter values. For instance, it is not clear how many particles a swarm should be composed of. The Tribes program [20] is a PSO method for unconstrained optimization in which the swarm adapts to the problem. Tribes are groups of individuals of variable size moving in an unknown environment in search of a *good* site [20]. In terms of PSO, the tribes are simply neighbourhoods of particles in which each particle has a direct connection to every other particle. The difference between the Tribes program and other PSO algorithms is that in Tribes the neighbourhoods are generated and modified automatically via creation, evolution, and removal operators.

The basic rule of the operators is that tribes that are composed of improving particles can decrease their populations while tribes that do not have improving members increase their populations. This heuristic diverts computation time from tribes that are progressing well to tribes that are having difficulty.

The PSwarm software also uses PSO. PSwarm is a hybrid method that uses both PSO and pattern search for bound constrained global optimization [96]. Pattern search is a derivative-free optimization routine that guarantees convergence to a stationary point [39]. PSwarm is a two-stage process. Initially PSO is used to explore the variable space. The best candidate solution found by the particles is then used to launch a pattern search.

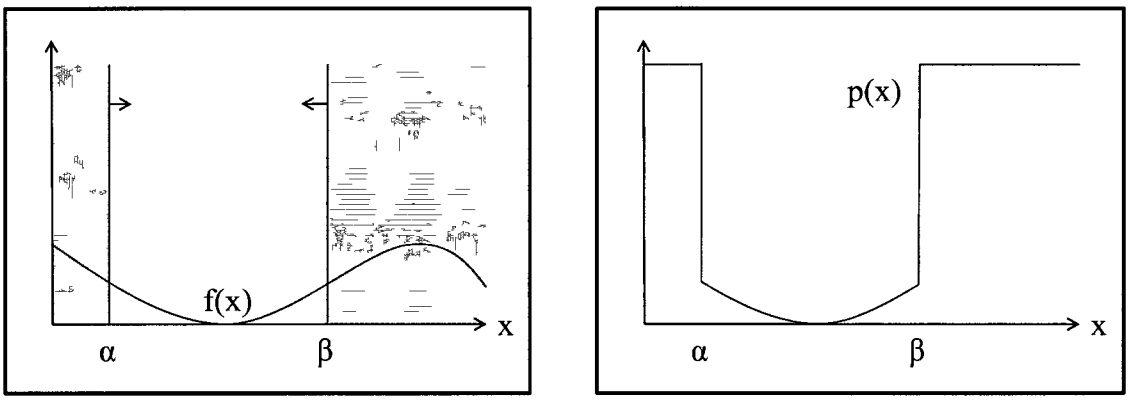
One of the major issues with population-based algorithms such as GA or PSO is that

they require a large number of function evaluations. For problems that have computationally expensive objective or constraint functions these methods may not be practical. A methodology for efficient global optimization of models with expensive functions was introduced by Jones *et al.* [47]. Their Efficient Global Optimization (EGO) algorithm is designed specifically for engineering problems that tend to have nonlinear and multimodal functions. They address the challenge by fitting response surfaces to data that is collected by evaluating the objective and constraint functions at a few points (they suggest using approximately $10n$, where n is the number of dimensions). Once the response surface is fit, the algorithm applies an iterative branch and bound process. More samples are taken in areas where expected improvement in the objective function is maximized. Every time additional samples are taken the response surface is re-estimated. The process is repeated until the expected improvement is less than 1% of the current best solution. Results for EGO on 4 small models including the 2D Branin model (see Chapter 6) are presented in [47]. The authors conclude that the method is viable for low-dimensional unconstrained problems.

2.3.3 Dealing with Constraints

A large part of the literature on global optimization to date is for unconstrained models. Therefore, a substantial effort in the field of constrained optimization is geared towards techniques to *manage* constraints and effectively convert constrained models into unconstrained models so that existing methods for solving unconstrained models can be used to solve constrained models. For instance, Particle Swarm Optimization solutions have included the penalty method [22, 73, 72], feasibility point superiority methods [37, 41, 77], and multiple swarm methods [103]. This section describes various constraint management techniques.

Penalty methods are used for NLP problems with equality and/or inequality constraints. The main idea is to convert the constrained problem into a sequence of equivalent unconstrained problems, then to use the algorithms developed for unconstrained optimization. The conversion is done by adding a term to the objective function that penalizes any violation of the constraints (see Figure 2.6 for an illustration). For instance, using a penalty method for minimization, constraint violations would result in a positive number being added to the objective function. The penalty function method is referred to as an exterior point method because it may generate a sequence of infeasible points (with respect to the original constrained problem). The penalty method is used extensively, particularly with population-based global search methods such as genetic algorithms where penalty functions



(a) Constrained model. (b) Unconstrained model.

Figure 2.6: Using a penalty function for minimization: (a) The initial constrained model is `min` $f(x)$ `s.t.` $\alpha \leq x \leq \beta$. (b) The constrained model is converted to be an unconstrained model that penalizes the objective function to account for the infeasible region. The new unconstrained model is simply `min` $p(x)$.

are convenient measures for the fitness levels of the individuals.

In terms of seeking feasibility, the penalty function method can be used to improve an initial point before passing it along to another solver. In this case the penalty function method is commonly referred to as a *Phase 1* method [18, 58]. The idea is that a feasible point is found in Phase 1, and then an optimal solution is sought in Phase 2. This is related to the Multistart method described in Chapter 3. A general weakness of the penalty method is that effective penalty functions can be very model specific. Furthermore, the magnitude of the penalty must be of sufficient size to enforce constraints without causing numerical conditioning problems [12].

Feasible point superiority algorithms are another approach for handling constraints that assume feasible solutions are always better than infeasible solutions. A simple yet inefficient way to implement this idea is to only use feasible candidate solutions. This is a feasible point superiority method known as *feasibility preservation.* For instance, Hu and Eberhart [41] proposed a modified particle swarm algorithm in which, during the initialization stage, particle positions are randomly generated and only feasible positions are kept. Infeasible positions are rejected and regenerated. Furthermore, the algorithm is altered such that only feasible candidate solutions are passed among the particles. Another approach to the feasible point superiority method that is applied to both particle swarm optimization [77, 104, 105] and genetic algorithms [23, 76] uses a set of rules that govern how candidate

solutions compare. Although the rules and implementations differ depending on the paper, the general idea is summarized as follows:

- Feasible solutions are preferred to infeasible solutions,
- Among feasible solutions, the one having the best objective function value is preferred, and
- Among infeasible solutions, the one with the smallest measure of constraint violation is preferred.

These rules are applied when an algorithm compares points. For instance, particle swarm algorithms keep track of the best location found. Using the rules above, if a feasible solution is found and the best solution in memory is infeasible, the feasible solution will replace it no matter what the objective function evaluates to at either point.

2.4 Summary

This chapter reviews the main NLP problem formulation for mathematical optimization. Section 2.1 covers the basic ideas relevant to the research presented herein. For instance, a set of large-scale problems is introduced in Chapter 5, seeking disjoint feasible regions is addressed in Chapter 6, and Chapter 7 is about finding global optimal solution vectors.

Global optimization is a difficult task, and as mentioned in the introduction, practical problems are continually getting harder. Consider the wire-routing problem for computer chips: 40 years ago a chip would have somewhere around two thousand transistors, now there are over two billion transistors on a chip. The optimization model describing the wire routing problem is much larger now. Many of the challenges of global optimization listed in Section 2.2 are highlighted throughout this document. For example, Chapter 4 describes weaknesses of the state of the art in global optimization software and Chapter 6 uses illustrations to identify challenging aspects of constrained nonlinear optimization problems.

Various techniques described in Section 2.3 surface again later within this dissertation. The next chapter focuses on the state of the art in multistart global optimization algorithms.

Chapter 3

The State of the Art in Multistart Global Optimization Algorithms

Multistart algorithms are probabilistic techniques that use a local solver for global optimization. Multistart is based on *pure random search* [11], an algorithm initially designed for optimization of unconstrained models. Pure random search proceeds by generating a random sequence of points in the feasible region. When a stopping criterion is met, the best point of the sequence is used as an approximation to the optimal solution. For unconstrained models the best point is the one with lowest objective function value. It was shown that if the objective function is continuous, then pure random search converges to the global optimum solution with probability one [80].

Multistart improves upon pure random search by launching a local solver from each randomly sampled point. Therefore, multistart is composed of two basic stages: *global* and *local*. During the global stage of multistart a sampling routine selects a candidate point. The local stage of multistart consists of a search routine that attempts to improve the candidate point. The two phases are repeated until user defined stopping conditions are met. The name *Multistart* is derived from the fact that a local solver is started from various locations.

This chapter addresses the main issues involved in using multistart algorithms for global optimization. Selection of the initial points is discussed first. This is an important part of multistart algorithms that is often overlooked: starting points greatly affect the outcome of local solvers [42]. Simple extensions to the multistart algorithm are also introduced, including clustering, acceptance-rejection, and repair techniques. In the final sections of this chapter, the state of the art of multistart software for global optimization is addressed in detail.

3.1 Initial Point Placement

A local solver's success in finding an optimal solution for an NLP is greatly influenced by the initial point provided by the modeler. A good initial point will allow the solver to find a feasible point or a global optimum with ease. In fact, the best initial point to start with would be the global optimum; of course this point is not known at the beginning of the search. Alternatively, given a poor initial point the NLP solver may not be able to find a feasible solution. It is also common that no initial point is even provided by a knowledgeable modeler. This section discusses various methods for placing initial points.

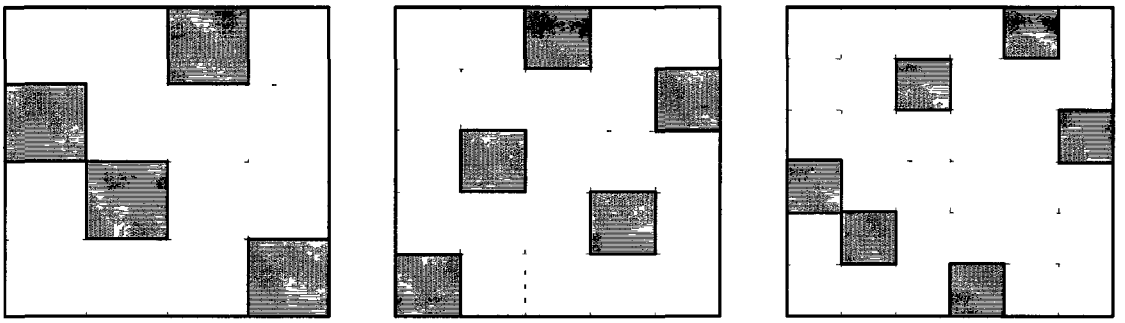
3.1.1 Standard Heuristics

Two of the simplest initial point selection methods are to use a random point or the origin. The problem with using random points is that it is unclear what one should do about unbounded variables. On the other hand, blindly using the origin can cause numerical problems, especially for models where the origin is not in the search space. To avoid these difficulties a commonly applied heuristic for selecting the initial point is the following:

- if the variable is double bounded, set it to the midpoint, x_m,
- if the variable is singly bounded, set it to the bound, x_b,
- if the variable is unbounded set it to zero.

This standard heuristic method is not immune from numerical problems because it sets many coordinates to zero. This causes problems for any terms of the form $1/x$. Furthermore, this commonly applied heuristic may set many coordinates to the same value, for instance, when variables have the same bounds. This causes numerical difficulties for terms of the form $1/(x_1 - x_2)$. Of course, a small random perturbation could be added to each of the initial values to avoid these problems. The result is the randomized standard heuristic for selecting the initial point as proposed by Ibrahim and Chinneck [42]:

- if the variable is double bounded, set it to the midpoint plus a perturbation, $x_m + \delta$,
- if the variable is upper bounded, set it to the bound minus a perturbation, $x_b - \delta$,
- if the variable is lower bounded, set it to the bound plus a perturbation, $x_b + \delta$,
- if the variable is unbounded, set it to zero plus a perturbation, $0 + \delta$,



(a) Four samples in $\mathcal{R}^2$. (b) Five samples in $\mathcal{R}^2$. (c) Six samples in $\mathcal{R}^2$.

Figure 3.1: Two dimensional Latin hypercube samples.

where δ is a uniformly distributed random number between 0 and 1 (or some smaller range depending on the variable bounds). Ibrahim and Chinneck's results show that NLP solvers find feasible solutions more frequently when they are launched from the randomized version of the initial point placement heuristic rather than the standard version, the origin, or a randomly selected point.

3.1.2 Latin Hypercube Sampling

Multistart methods usually require a set of initial starting points. A simple solution is to generate a set of random starting points. A slightly more advanced set of starting points includes the randomized standard heuristic point. The problem with completely random points is that they may not be evenly distributed throughout the search space. Latin hypercube sampling (LHS) can be used to better distribute the initial sample [61].

In statistics, a Latin square is a two dimensional grid such that there is one sample in each row and column (for example see Figure 3.1). LHS is a generalization of the Latin square. Each dimension is divided into equally probable segments. A value is chosen randomly from each segment and samples are produced by randomly combining a set of values, each from a different dimension. This process can be completed sequentially (i.e. one point at a time) by recording previously sampled segments at each iteration so that they are never chosen more than once. Another advantage to this method is that the number of samples is independent of the number of dimensions.

3.1.3 Scatter Search

Scatter Search is a metaheuristic: it finds an optimal solution by iteratively improving a candidate solution set with regard to a given measure of quality. Scatter search was first introduced in 1977 as a heuristic for integer programming [34]. More recently, it has been applied to other applications including continuous optimization problems [55].

The basic idea driving Scatter Search is that information about the global optimum is stored in a diverse set of high-quality candidate solutions known as the *reference set.* Scatter Search linearly recombines samples from the reference set in an attempt to continually improve the candidate solutions. The result is a sampling routine that quickly determines a set of high-quality, yet diverse candidate solutions. However, Scatter Search is used in other ways. Some examples included using it as an *improvement method* to transform a trial solution into one or more enhanced candidate solutions, or a *solution combination method* to transform a given subset of solutions into one or more combined solution vectors [55]. A basic template for Scatter Search is presented in [35]. It is discussed further in reference to multistart and clustering routines in Section 3.4.1.

3.2 Feasibility Seeking Algorithms

A nonlinear program normally consists of an objective function and a set of constraints. The objective function is ignored in the *Feasibility Problem* considered here, which consists of finding a point that satisfies all of the constraints simultaneously:

$$\text{Find } \mathbf{x} \in \mathcal{F} \tag{3.1}$$

where the feasible region $\mathcal{F}$ is the intersection of the constraints (Eqn. 2.2) and the variable bounds (Eqn. 2.3). Due to the nonlinear functions involved, the feasible region may be a disjoint set. A *feasibility vector* is an estimate of a vector from an infeasible point to the closest feasible point for a given violated constraint. For a particular point $\mathbf{x} \notin \mathcal{F}$ there is a feasibility vector for each violated constraint. A *consensus vector* is the movement vector for a particular iteration of a feasibility seeking algorithm [17]. Consensus vectors are usually created by combining the feasibility vectors.

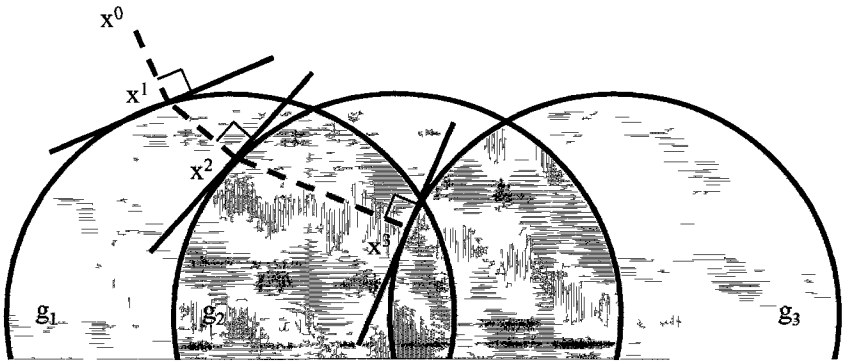


Figure 3 2 The method of successive orthogonal projections (figure inspired by [16], p 82)

3.2.1 Projection Algorithms for the Feasibility Problem

Many projection algorithms exist for solving the feasibility problem [16]. Projection algorithms calculate the orthogonal projection of the current infeasible point onto the closest point that satisfies the constraint Since this calculation is time consuming for nonlinear constraints, projection algorithms often use the feasibility vector for a violated constraint as an approximation. Sequential projection algorithms use only a single feasibility vector to form the consensus vector and each iteration of the algorithm uses the feasibility vector from a different violated constraint. An iteration of a projection algorithm comes to an end when the consensus vector is added to the initial point to generate a new point for the next iteration.

Constraint selection is governed by a control sequence. A variety of choices exist for the control sequence, including simple cyclic control (e.g. systematically rotate through all the constraints), and most violated constraint control which uses the longest feasibility vector at the current point. A scalar known as the *relaxation factor*, $\lambda > 0$, is sometimes used to adjust the length of the consensus vector.

The Sequential Orthogonal Projection method [36] is a basic form of the sequential projection algorithm and is illustrated in Fig. 3.2. Its control sequence is a simple cycle and the relaxation factor is set to 1. The feasibility vector for the next violated constraint in the cycle is used as the consensus vector at each iteration.

The calculation of an exact feasibility vector for a nonlinear constraint may require the solution of an entire nonlinear program and therefore can be computationally expensive. For this reason, projection algorithms often use a feasibility vector that is estimated based on the gradient of the violated constraint at the current point. Methods in this class include the Method of Cyclic Subgradient Projections [15], the Relaxation Method of Agmon *et al.* [2, 68], and Kaczmarz's Algorithm [81].

Simultaneous projection algorithms form the consensus vector by combining all of the feasibility vectors in a certain way. Block-iterative projection algorithms [4] are a particular type of simultaneous algorithm in which the consensus vector is constructed by combining only a subset of the feasibility vectors, often using a weighting system. Variations to block-iterative projection algorithms include how the feasibility vectors are selected and how the weights are determined.

3.2.2 Newton's Method

Newton's method [90] can be considered a sequential projection method for finding the root of an equation. For a function of a single variable having the form $g(x) = 0$, the iterative update step is

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} \tag{3.2}$$

where $g'(x)$ is the derivative of $g(x)$.

Newton's method can be adapted to handle multiple dimensions and multiple equations, however, it often requires a matrix inversion in place of the division by the derivative term in Eqn 3.2. For instance consider the pseudo-inverse (PIV) algorithm recently proposed by Chootinan and Chen as a repair method for genetic algorithms [19]. The PIV algorithm calculates a *repair vector* that is added to the current solution vector in order to direct infeasible solutions towards feasibility.

Define the Jacobian matrix

$$\mathbf{G}(\mathbf{x}) \doteq \begin{bmatrix} \nabla g_1(\mathbf{x}) \\ \vdots \\ \nabla g_m(\mathbf{x}) \end{bmatrix} \in \mathcal{R}^{m \times n}. \tag{3.3}$$

Chootinan and Chen suggest that the gradient of the constraints with respect to the solution vector determines the rate of change of constraint violation $\Delta\mathbf{v} \in \mathcal{R}^{m \times 1}$ for a unit change of the solution vector

$$\Delta\mathbf{v} = \mathbf{G}(\mathbf{x})\Delta\mathbf{x}. \tag{3.4}$$

Furthermore, the inverse of the Jacobian matrix $\mathbf{G}^{-1}(\mathbf{x})$ provides the rate of change of the solution vector with respect to the change in constraint violation

$$\Delta\mathbf{x} = \mathbf{G}^{-1}(\mathbf{x})\Delta\mathbf{v}. \tag{3.5}$$

Chootinan and Chen use the approximation

$$\Delta \mathbf{v} \approx \begin{bmatrix} v_1(\mathbf{x}) \\ \vdots \\ v_m(\mathbf{x}) \end{bmatrix} \in \mathcal{R}^{m \times 1}, \tag{3.6}$$

where v_i is the violation of the i^{th} constraint as defined in Section 2.1.

The main issue with the PIV algorithm is that the number of violated constraints is not generally equal to the number of variables in a model. Since $\mathbf{G}(\mathbf{x}) \in \mathcal{R}^{m \times n}$, the matrix $\mathbf{G}$ is not invertible if $m \neq n$. To circumvent this issue the *pseudo-inverse* matrix [74], $\mathbf{G}^+(\mathbf{x})$, is used when $m \neq n$. The pseudo-inverse repair equation is

$$\Delta \mathbf{x} = \mathbf{G}^+(\mathbf{x}) \Delta \mathbf{v}. \tag{3.7}$$

Pseudo-inverse repair may have to be applied more than once to a particular infeasible solution vector in order to move it into the feasible region because of the approximations made during the calculation. In practice the PIV repair method is repeatedly applied to successive solution vectors until the resulting solution vector is feasible, the repair vector is deemed insignificant, or a maximum number of iterations is reached. For specific details see [19].

One of the drawbacks of the PIV method, and any other method that requires matrix inversion, is the amount of computational effort it takes to invert a matrix. Inversion calculations do not scale well and quickly become intractable for problems with many constraints and variables. High quality local solvers (e.g. Ipopt [97] and Knitro [13]) do not invert large matrices, but instead solve linear systems at each of their iterations. This technique can also take a great deal of time when it is applied to large models.

3.2.3 Constraint Consensus Methods

CC methods [17, 18, 42] use various forms of projection algorithms as heuristic methods for solving the feasibility problem. CC can be applied to general sets of nonlinear constraints, including nonconvex constraints. The *Basic* method of CC [17] is a fully simultaneous algorithm with $\lambda = 1$. Consensus vectors are formed component-wise; each element of the consensus vector is formed as the average of the elements in the feasibility vectors for violated constraints at the current point that involve that variable. This is the same method employed by the Component Averaging projection method proposed by Censor et

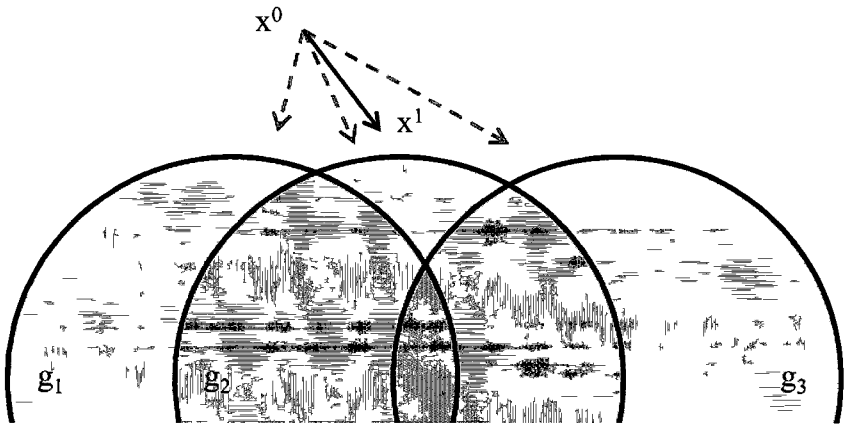


Figure 3.3: Example of Basic CC. The gray arrows represent feasibility vectors and the black arrow represents the corresponding consensus vector.

al. [14]. The Basic method of CC is illustrated in Fig. 3.3.

A variety of CC methods are available, each varying in the way it constructs the consensus vector [42]. The *Feasibility Distance far* (FDfar) algorithm chooses the longest feasibility vector as the consensus vector. The *Direction Based average* (DBavg) algorithm constructs the consensus vector component-wise. For each element, the direction of movement (positive or negative) is determined by the most common sign among the elements for that component in the feasibility vectors. The step size is calculated by averaging the magnitudes of the elements in the winning direction for that component in the feasibility vectors. *Direction Based maximum* (DBmax) is similar to DBavg except that the largest element determines the distance of movement in the winning direction in each component. Other consensus schemes exist, but empirical testing shows the superiority of the DBmax and FDfar variants [42].

State of the art CC uses a linear approximation to calculate feasibility vectors for nonlinear constraints known as *linear feasibility vectors*. This calculation is exact for linear constraints, but only an estimate for nonlinear constraints [17]. A linear feasibility vector moves a point in a direction parallel to the gradient of the violated constraint. The vector's magnitude is determined using a first order Taylor series expansion of the constraint function around the current location Consider a first order Taylor series expansion of some constraint function $g(\mathbf{x}) : \mathcal{R}^n \rightarrow \mathcal{R}$

$$g(\mathbf{x}_{k+1}) \approx g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{3.8}$$

The search direction is chosen to be parallel to the gradient of g at $\mathbf{x}_k$, therefore

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_\ell \nabla g(\mathbf{x}_k)^T \tag{3.9}$$

where α_ℓ is some step size and superscript T denotes transposition. The scalar α_ℓ will be negative for violated inequality constraints and for equality constraints with $g(\mathbf{x}_k) > 0$, but will take a positive value for violated equality constraints with $g(\mathbf{x}_k) < 0$. The step size is computed by setting $g(\mathbf{x}_{k+1}) = 0$ and solving for α_ℓ

$$\begin{aligned} 0 &= g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) \\ 0 &= g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\alpha_\ell \nabla g(\mathbf{x}_k)^T) \\ 0 &= g(\mathbf{x}_k) + \alpha_\ell ||\nabla g(\mathbf{x}_k)^T||^2 \\ \alpha_\ell &= \frac{-g(\mathbf{x}_k)}{||\nabla g(\mathbf{x}_k)^T||^2}. \end{aligned} \tag{3.10}$$

Therefore, the linear feasibility vector is

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) = \frac{-g(\mathbf{x}_k)}{||\nabla g(\mathbf{x}_k)^T||^2} \nabla g(\mathbf{x}_k)^T. \tag{3.11}$$

This feasibility vector calculation is proposed in the original CC algorithm [17], and previous projection methods [14, 81].

The linear feasibility vector defined by Eqn 3.11 is applied to both linear and nonlinear constraints. For linear constraints Eqn 3.11 is exact, however, for nonlinear constraints it is only an approximation. For an illustration, consider the model

$$g_1(\mathbf{x}) = 4.32 - (x_1 + x_2) = 0 \tag{3.12}$$

$$g_2(\mathbf{x}) = (x_1 + 2)^2 + x_2^2 - (x_1 + 2)x_2 - 10 = 0 \tag{3.13}$$

where $\mathbf{x} = [x_1 \;\; x_2]^T$. The corresponding constraint gradient equations are

$$\nabla g_1(\mathbf{x}) = [-1 \quad -1] \tag{3.14}$$

$$\nabla g_2(\mathbf{x}) = [2(x_1 + 2) - x_2 \quad 2x_2 - (x_1 + 2)]. \tag{3.15}$$

Given an initial point $\mathbf{x}_0 = [8 \quad -8]^T$ the linear feasibility vector for the linear constraint

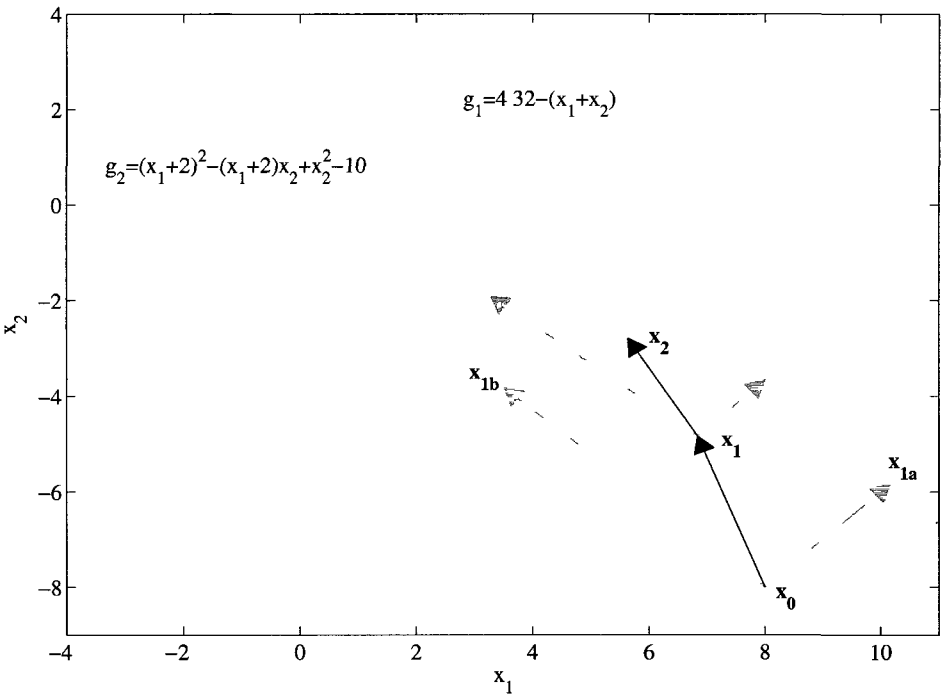


Figure 3.4: The linear feasibility vector calculation.

g_1 is

$$(\mathbf{x}_{1a} - \mathbf{x}_0) = \frac{-g_1(\mathbf{x}_0)}{||\nabla g_1(\mathbf{x}_0)^T||^2} \nabla g_1(\mathbf{x}_0)^T \tag{3.16}$$

$$= \frac{-4.32}{2} \begin{bmatrix} -1 \\ -1 \end{bmatrix} \tag{3.17}$$

$$= \begin{bmatrix} 2.160 \\ 2.160 \end{bmatrix}. \tag{3.18}$$

The linear feasibility vector for the nonlinear constraint g_2 is

$$(\mathbf{x}_{1b} - \mathbf{x}_0) = \frac{-g_2(\mathbf{x}_0)}{||\nabla g_2(\mathbf{x}_0)^T||^2} \nabla g_2(\mathbf{x}_0)^T \tag{3.19}$$

$$= \frac{-234}{1460} \begin{bmatrix} 28 \\ -26 \end{bmatrix} \tag{3.20}$$

$$= \begin{bmatrix} -4.488 \\ 4.167 \end{bmatrix}. \tag{3.21}$$

The feasibility vectors are illustrated in Fig. 3.4. The linear feasibility vector $(\mathbf{x}_{1b} - \mathbf{x}_0)$

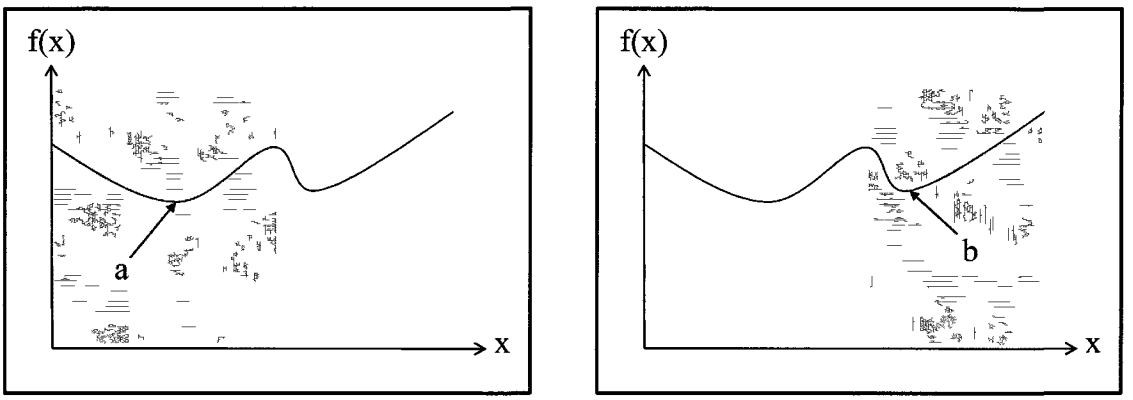
for the nonlinear constraint g_2 is only an estimate. The feasibility vector $(\mathbf{x}_{1a} - \mathbf{x}_0)$ for the linear constraint g_1 is exact, that is, it is the shortest path to the constraint from the current point. The consensus vector $(\mathbf{x}_1 - \mathbf{x}_0)$ is also shown in Fig. 3.4. It is formed using the *Basic* method for consensus.

Although the point $\mathbf{x}_1$ is closer to the feasible region than its predecessor, it still violates the constraints (note that there is a single feasible point near $\mathbf{x} = [1.216 \ 3.104]^T$ where the constraints intersect). The algorithm is repeated using $\mathbf{x}_1$ as the starting point for the next iteration. This pattern is repeated to find successive points that are closer to the feasible region. At the point $\mathbf{x}_2$ in Fig. 3.4 the violations of constraints g_1 and g_2 are 1.476 and 77.479, respectively. The violations at the initial point $\mathbf{x}_0$ are greater, specifically $v_1 = 4.320$ and $v_2 = 234.0$, respectively.

CC iterations continue until predefined stopping conditions are met. Three conditions often used are [42]: (i) stop when every feasibility vector is shorter than α (e.g. 10^{-6}), (ii) stop when the consensus vector is shorter than β (e.g. 10^{-4}), or (iii) stop when μ iterations have been completed (e.g. 500). Triggering the first stopping condition indicates successful termination because the constraint violation estimates are less than the user defined tolerance. The other two conditions indicate unsuccessful termination since they are triggered when convergence is considered too slow. A scheme to exit unsuccessfully when the consensus vectors are not shortening enough between iterations is also proposed [18, 58]. Slow convergence results from ill-conditioning such as a pair of constraints that are close to parallel that cause feasibility vectors to almost cancel out, resulting in a relatively short consensus vector. Slow convergence also results when conditions cause consensus vectors to zig-zag through the search space as they alternately satisfy one set of constraints and then another [17]. Another issue of concern is poor linear feasibility vector approximations to highly nonlinear constraints. Chapter 5 introduces new improved methods for calculating accurate feasibility vectors.

3.3 Clustering

Clustering is the process of dividing data into groups (clusters) for the purpose of improved understanding. Practical applications include a variety of subjects such as relating documents for browsing and information retrieval and providing a grouping of spatial locations prone to earthquakes [88]. The focus of this research is to use clustering to identify sets of promising launch points for a local solver in order to reduce the effort required for large-scale global optimization. As mentioned previously, a downfall of the multistart method



(a) Left basin of attraction. (b) Right basin of attraction.

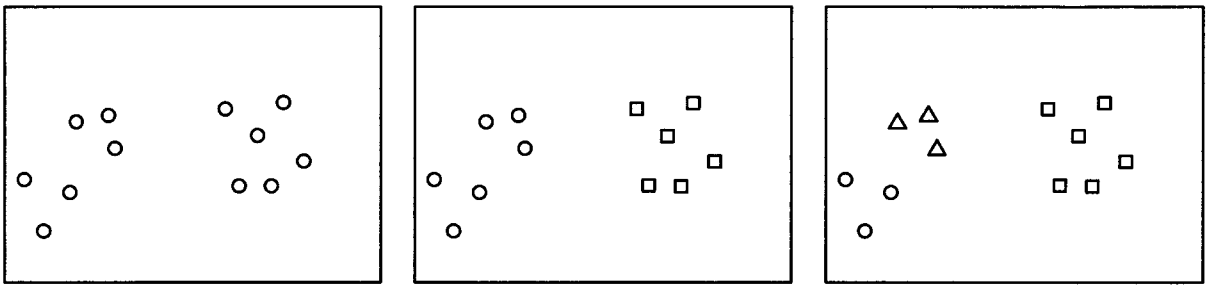
Figure 3.5: Example of basin of attractions in 2D: (a) A steepest descent local solver starting from any point in the left basin will return **a** as the minimum. (b) Alternatively, a steepest descent local solver launched from any point in the right basin will return **b** as the minimum. In general, **a** and **b** are different values.

is that many start points may lead to the same feasible region, and ultimately the same solution Clustering methods attempt to remedy this inefficiency by grouping prospective launch points. The local solver is launched only once from each group of points to eliminate redundant searches. See Figure 3.5 for an illustration of 2D basins of attraction.

The basic goal of clustering is to organize objects according to their characteristics such that similar objects belong to the same groups and unrelated objects belong to other groups. Clustering only uses information from the data, hence it is an unsupervised classification method. This is different from supervised classification whereby previously assigned class labels are utilized. One of the main issues for clustering is: what constitutes a cluster? For global optimization software that seeks basins of attraction, cluster definitions are commonly based on the relative spatial locations and objective values of the points [85].

Distinguishing clusters from the data is not necessarily an easy task. For example, Figure 3.6 shows how data can be classified into different numbers of clusters depending on how the clusters are defined. The measure of closeness between two points is a critical design detail of any clustering algorithm Different measures are required depending on the application. Commonly used measures are based on the Minkowski metric

$$L_r = dist_r(\mathbf{x}_a, \mathbf{x}_b) \doteq \left(\sum_{i=1}^{n} |x_{a_i} - x_{b_i}|^r \right)^{1/r}, \tag{3.22}$$



(a) One cluster. (b) Two clusters. (c) Three clusters.

Figure 3.6: Different groupings for a set of 12 objects. Shapes are used to distinguish clusters.

where x_{a_i} and x_{b_i} are the i^{th} components of the points $\mathbf{x}_a \in \mathcal{R}^n$ and $\mathbf{x}_b \in \mathcal{R}^n$, and r is a user defined parameter. Common cases are the L_1 norm or taxicab distance, the L_2 norm or Euclidean distance, and the L_∞ norm or Chebyshev distance. This thesis uses the L_2 norm because Hinneburg *et al.* showed that the concept of a nearest neighbour in high dimensional spaces is meaningless for L_r metrics with $r \geq 3$ [38].

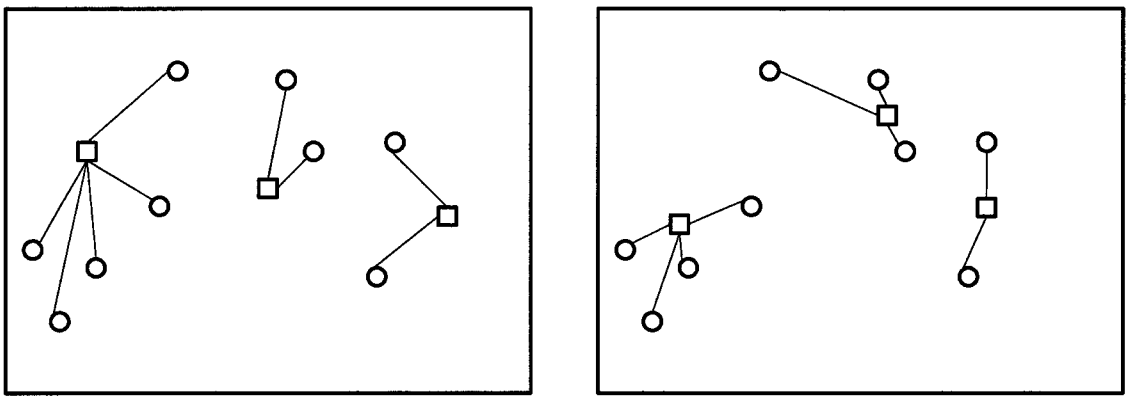
3.3.1 Hierarchical and Partitional Techniques

The two main approaches to clustering are hierarchical and partitional. Partitional methods divide data into the desired number of clusters for the duration of the algorithm. Divisive hierarchical methods start with all the data points in a single cluster and successively split the cluster and its offspring clusters until the desired number of clusters is produced or some other stopping criteria is satisfied. On the other hand, agglomerative hierarchical methods start with each data point assigned to its own cluster and then merge clusters together until a termination condition is met.

K-Means is a partitional algorithm that organizes data points into K clusters $\{\Lambda_1, \Lambda_2, ..., \Lambda_K\}$, each with mean $\mathbf{u}_i \in \mathcal{R}^n$. The algorithm attempts to minimize

$$\sum_{i=1}^{K} \sum_{\mathbf{x}_j \in \Lambda_i} ||\mathbf{x}_j - \mathbf{u}_i||^2 \tag{3.23}$$

by repeating two basic operations. First, each data element is assigned to the cluster with the closest mean. Second, the mean of each cluster is updated. An illustrative example of an iteration of the K-Means algorithm is presented in Figure 3.7.



(a) Points assigned to closest clusters. (b) Means updated and points reassigned.

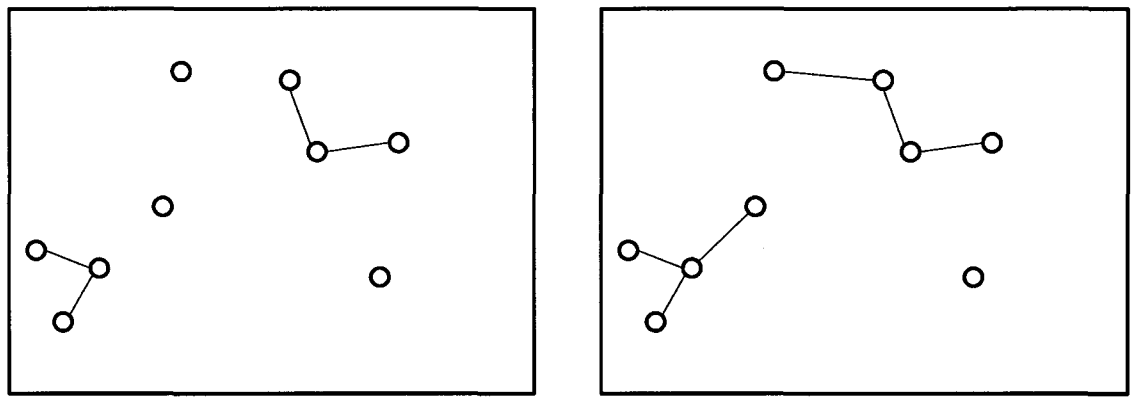
Figure 3.7: An iteration of the K-Means method with $K = 3$. Circles represent data points while squares represent cluster means.

The K-Means algorithm is a local search routine and different start points can lead to different solutions. The initial means can be assigned arbitrarily, however, other start techniques exist. An example is the K-Means++ algorithm that chooses the initial means using a random weighting scheme that favours points far away from each other [5]. Another practical consideration of the K-Means algorithm is how many clusters to search for. As shown in Figure 3.6, the number of clusters is not necessarily obvious. For the purpose of global optimization the number of clusters should be equal to the number of basins of attraction. The difficulty is that the number of basins of attraction is unknown.

Single linkage or nearest neighbour clustering is a hierarchical method for forming clusters that does not require the user to set how many clusters to search for [46]. Initially each data point is considered a cluster. During every iteration the clusters found to be the closest are linked to form a larger cluster. The process is repeated until the distance between each cluster is greater than a critical value. See Figure 3.8 for an example. Similar hierarchical clustering methods that use different linking strategies have been proposed. Examples include the complete link and group average algorithms [88]. One of the advantages of the hierarchical algorithms is that the number of clusters is not set before the algorithm runs. Instead, hierarchical algorithms require a critical distance.

3.3.2 The Grid Method

The grid method is a basic density-based approach to clustering [88]. Initially the space is divided into hypercells (see Figure 3.9), then adjacent high-density cells are combined



(a) 4 links, 5 clusters.

(b) 6 links, 3 clusters.

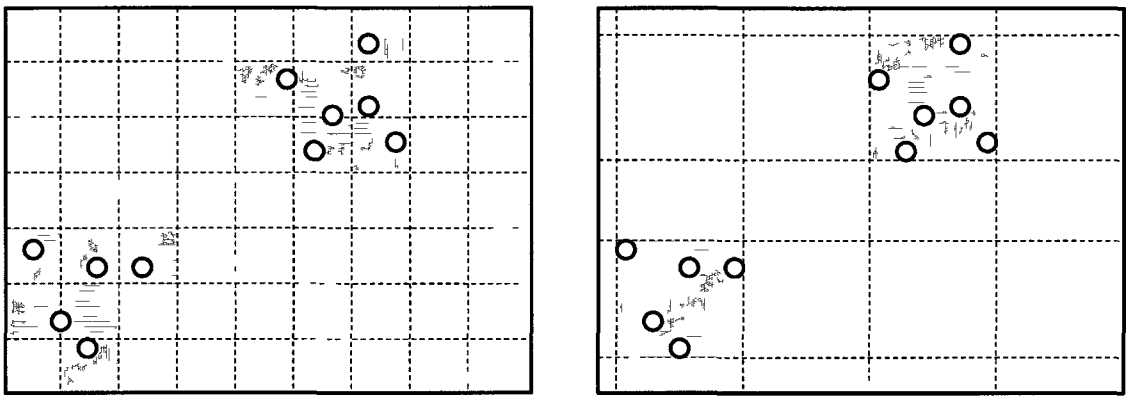
Figure 3.8: Single linkage cluster examples. The critical distance used to form the clusters is longer in (b) than (a), therefore example (b) has fewer clusters.

to form clusters. The geometry of the hypercells may not fit the shape of the clusters well since the hypercells are in fact hypercubes. This problem is mitigated at the cost of additional computation by decreasing the size of the hypercells. A greater concern is how the grid-based method performs on high-dimensional data because the number of hypercells increases exponentially with the dimension of the data.

CLIQUE [3] is a grid-based algorithm for clustering high-dimensional data. CLIQUE exploits the fact that a region that is dense in a certain space must also be dense when projected into a lower dimensional space. The CLIQUE algorithm projects dense regions onto lower-dimensional subspaces and then it uses the projections to predict clusters in the full-dimensional space. Density in a lower-dimensional space does not imply density in a higher-dimensional space, however. The CLIQUE method inspects predicted regions to confirm they are indeed clusters (see Figure 3.10).

3.4 Multistart

NLP solvers are sensitive to their start point and may return various results for the same model when launched from different start points. For global optimization it may be beneficial to run a local solver from different initial points and then pick the best solution. This is the basis of the multistart technique for global optimization. The multistart method is a modification to pure random search whereby a local search procedure is launched from every sample point. The algorithm is usually run for a predetermined number of sample



(a) Uniform grid. (b) Non-uniform grid.

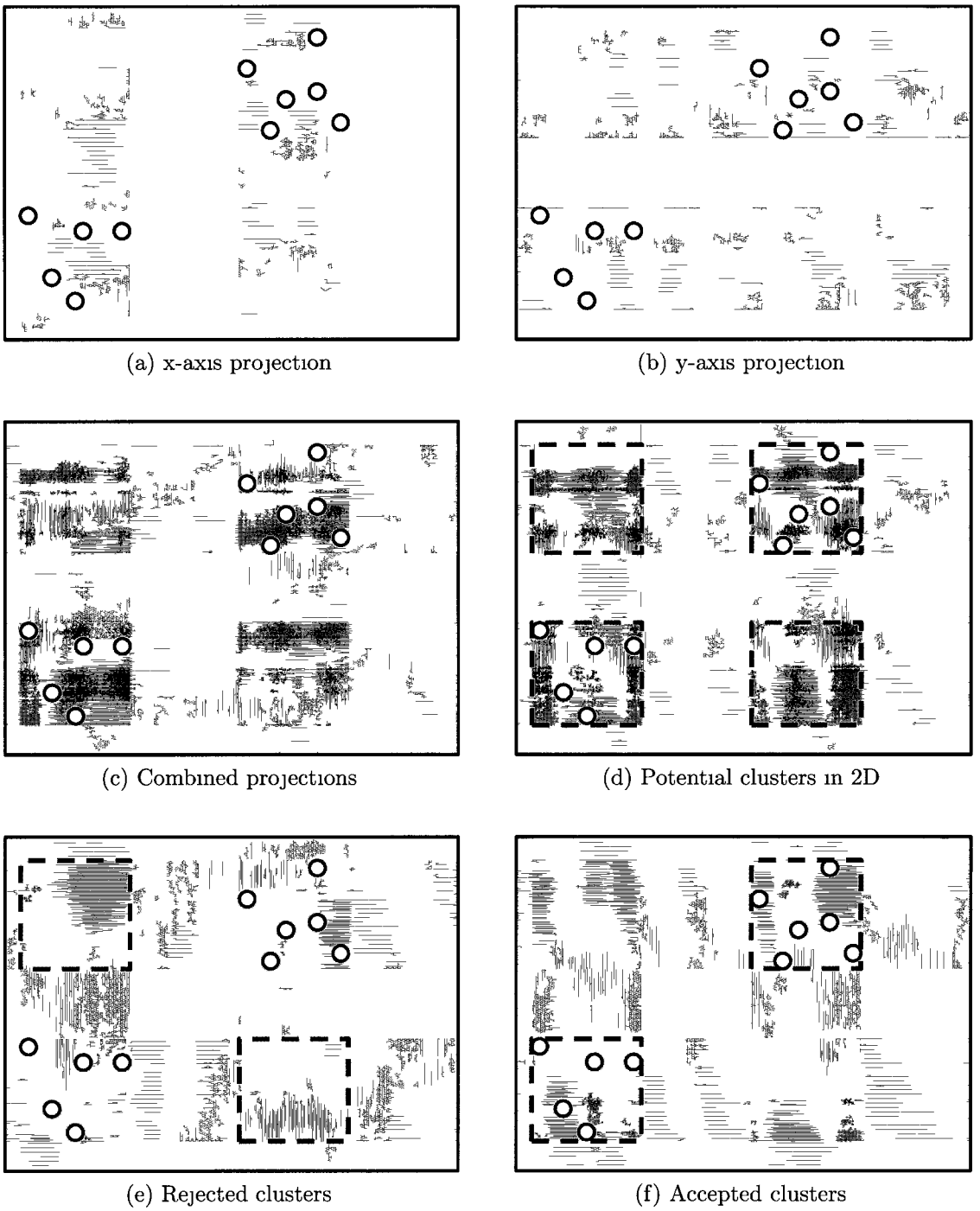
Figure 3.9: Different density-based clusters (shaded regions) using uniform and non-uniform grid methods: (a) A uniform grid where each of the divisions is equally spaced. (b) A non-uniform grid where the divisions are not spaced equally.

points or until a time limit is reached.

Although basic multistart is an improvement over pure random search, it still has inefficiencies. For instance, the multistart method may find the same local optima multiple times. If possible a solver should never be invoked more than once from each basin of attraction. A method that tries to remedy this issue is multistart with clustering as discussed in Section 3.3 and Chapter 6.

An example of a contemporary version of the multistart algorithm is the Greedy Randomized Adaptive Search Procedure (GRASP) [79]. GRASP is a two phase heuristic process designed for combinatorial optimization problems. During the construction phase a restricted list of the best elements is kept. A partial solution is generated by randomly selecting elements from the list. The second phase of GRASP is a local search that improves the constructed solution.

The repulsion algorithm is another form of the multistart algorithm [86]. Repulsion attempts to generate points that are in different basins of attraction so that local searches do not converge to the same local optimum points. The algorithm works by randomly generating a point and then transforming it based on the locations of previous samples. Through the transformation, the older samples repel the new point to an unexplored region.



(a) x-axis projection

(b) y-axis projection

(c) Combined projections

(d) Potential clusters in 2D

(e) Rejected clusters

(f) Accepted clusters

Figure 3 10 A 2D example of the CLIQUE algorithm (a,b) Initially density clusters are projected onto the x and y axes (c,d) The results are combined to form 4 potential clusters (e,f) The algorithm confirms the density of the possible clusters to accept or reject them

3.4.1 Applying Clustering to Optimization

There are three basic steps for incorporating clustering into the multistart method for global optimization [85]:

1. A set of candidate points is selected. The set is processed either by *reduction* or *concentration* to determine the set of *seed* points. A reduction routine determines the most promising subset of candidate points [6]. For an unconstrained minimization model the points with the highest objective function values would be removed. For constrained models a metric designed to compare the relative promise of the points (i.e., fitness in Section 2.3.2) is used. Alternatively, concentration is achieved by iterating each point through a few steps of a local optimization routine [91]. Unlike the reduction method, during concentration no points are eliminated from the search.

2. The seed points are used to build clusters.

3. Launch points are calculated for a local solver. Some possible launch points are the means of each cluster and the point in each cluster with the lowest sum of constraint violations.

In Scatter Search (Section 3.1.3), groups of candidate solutions are organized to form weighted centers in various regions throughout the search space. Subsets of these solutions are selected to be reference solutions. The reference solutions are then modified in linear combinations to produce new solutions that lay both within and outside of the regions spanned by the reference solutions. The best solutions are then identified and used to create a new set of candidate solutions and the whole process is repeated. Implementations of the Multistart Nonlinear Programming (MSNLP) method discussed in Section 3.5.2 use Scatter Search to determine candidate solutions.

An algorithm similar to MSNLP is multi-level single linkage (MLSL)][48, 49]. MLSL is a multistart routine, designed for unconstrained models, that uses simple rules to prevent a local solver from being launched from every randomly selected candidate point. Initially, the objective function is evaluated at each point in a random sample of N points in the feasible region. The points are sorted by their respective objective function values. The set of points is reduced by keeping the qN points with the lowest objective function values, where q is a parameter between 0 and 1. In the next stage, a local solver is launched from some of the remaining candidate points. In order for a candidate point to be accepted as a launch point, it must not be within a critical distance of another candidate point that has a lower objective function value. Furthermore, the candidate point will not be a launch

point if it is too close to previously discovered local minimum. When all the candidate points are exhausted, N additional points are selected and the process is repeated on the union of these points and those retained from the previous iterations. The critical distance is decreased each time a new set of sample points is added. The formula that determines the critical distance listed in [48] is similar to the formula used by the GLOBALm software discussed in Section 3.5.3.

A drawback of clustering methods is that their performance may depend on the dimension of the problem [29]. Bellman [7] wrote about the *curse of dimensionality* after trying to use brute force search methods for optimizing a function of many variables. The curse of dimensionality applies to any problem that suffers an exponential increase in volume of a mathematical space when the number of dimensions is increased. For instance, a fixed number of sample points becomes increasingly sparse as the dimension of a space increases. This can have negative effects on clustering algorithms, specifically on the distance or similarity measure, since in a high dimensional space certain similarity measures may not produce meaningful clusters for a relatively sparse set of data. For instance, Hinneburg *et al.* showed that the L_r metric with $r \geq 3$ is a meaningless distance measure in high-dimensional spaces [38]. Others have proposed various analysis methods that can be used to gain an understanding of the geometry of a data set in high-dimensional spaces that are difficult to visualize. For example, Brin[1] proposed using the distributions of the distances between the data [10]. Chapter 6 outlines a clustering method based on the observation that clusters of data points that are separable via a distance measure will appear as distinct peaks in the distribution of inter-point distances. Other techniques like projected clustering [1] attempt to reduce the effects of the curse of dimensionality by projecting data onto lower dimensional spaces.

3.4.2 Acceptance-Rejection

The acceptance-rejection method is another way to modify the multistart algorithm. Acceptance-rejection uses ideas from rejection sampling and statistical mechanics in order to improve the basic multistart algorithm [28]. As suggested by the name, the strategy of acceptance-rejection methods is to only start local searches from randomly generated points that pass certain criteria. The acceptance criteria are based on a probability calculation that involves a variable that is analogous to temperature in statistical mechanics [28] (also similar to simulated annealing, see Section 2.3.2). When the temperature variable is

[1] Co-founder of Google, Inc.

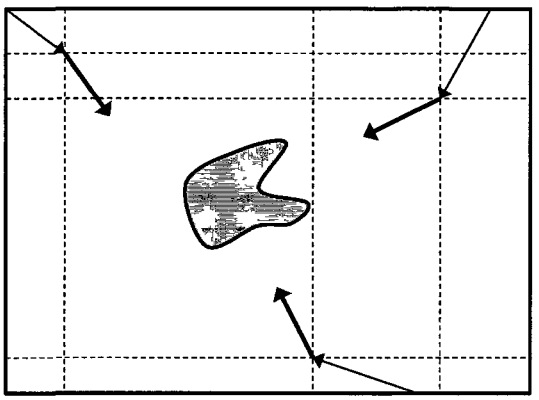


Figure 3.11: Defining bins for voting. The tails of the final consensus vectors are used to define axis-parallel divisions illustrated as dashed lines. The resulting grid divides the search space into a set of hypercells called bins.

high many of the points are accepted as starting points for the local search; however, if the temperature is decreased fewer points are accepted as initial start points for the local search. Higher quality starting points result due to the increasing severity of the filter. As the temperature variable approaches zero the algorithm is terminated. The MSNLP routine described in Section 3.5.2 uses a form of acceptance-rejection.

3.5 Recent Software

3.5.1 Multistart Constraint Consensus and Voting

A recent avenue of research studies how to extract information from unsuccessful search attempts. Methods like LHS are used to initially sample the search space, however, MacLeod [58] demonstrates that information collected by CC runs may help select promising areas for future search even if CC fails to find any feasible regions.

In his Master's thesis, MacLeod [58] presents a voting method that uses information from a set of CC runs to determine a new set of promising sample points. The voting method is similar to the projection [1] and grid-based [3] clustering methods where data points are projected onto lower-dimensional spaces. MacLeod proposes using the voting method in conjunction with a multistart method and a local solver (Knitro) for finding feasible regions in difficult nonlinear programs. The algorithm is called Multistart Constraint Consensus (MCC).

The voting method presented by MacLeod uses LHS to select a set of initial starting

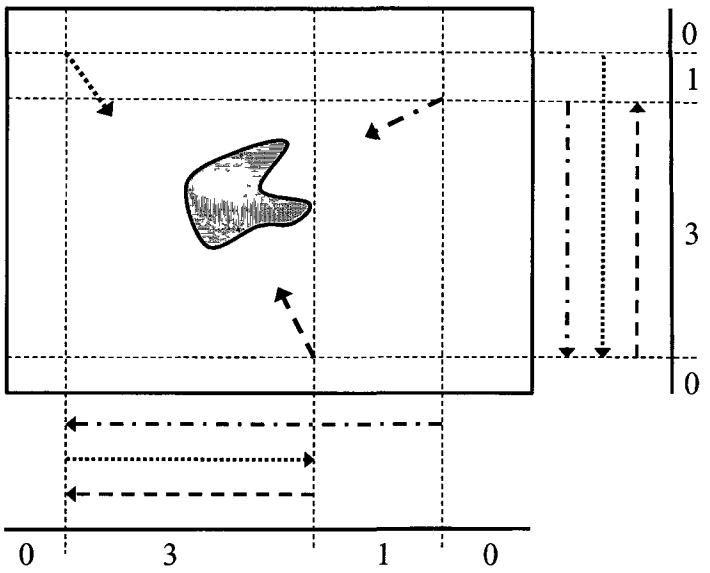


Figure 3.12: The voting process. The consensus vectors vote once in each dimension. The numbers above sum the votes in each bin. The vote sums form a distribution that is used to select the next solver launch point.

points that are run through the CC method. If any of the CC runs finds a feasible region then the MCC algorithm exits successfully. Otherwise, the last consensus vector from each of the runs is saved as it may store valuable information about the search space. Once all the vectors are determined, and if a feasible region is still not found, the vectors are used as a voting mechanism with the intention of finding promising areas for a future search. It is assumed that the last consensus vector should be pointing somewhat towards a feasible region, hence, given a set of consensus vectors one should be able to make an educated guess as to where a feasible region is located. A vote is completed in each dimension of the search space and the results are combined afterwards to determine a single new launch point. The launch point is passed to a local solver that runs in feasibility mode. If the local solver finds a feasible region the algorithm terminates successfully, else, the lowest quality point is replaced, and another vote is conducted (see [58] for details). This is repeated for a predetermined maximum number of solver launches.

The consensus vectors vote once in each dimension using their respective component in that particular dimension. Figures 3.11-3.13 illustrate a fictional problem in order to help explain how MacLeod's voting method works. The first figure in the series, Figure 3.11, shows a feasible region (an odd shape with gray fill) surrounded by three CC attempts. The last two iterations of each CC attempt are shown. The tail of the last consensus vector

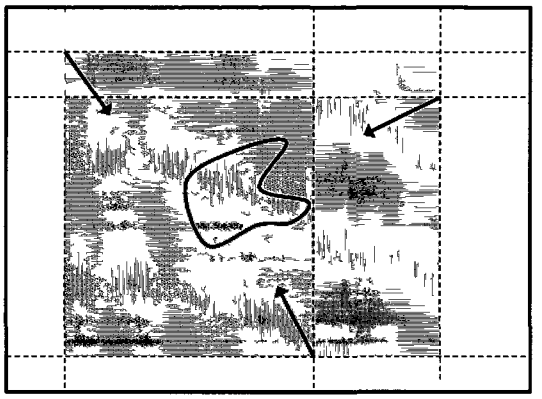


Figure 3.13: Voting results. The shaded areas represent the relative probabilities of a point being selected in a particular grid region. Darker shades indicate higher probabilities. The probabilities are determined from the bins' respective vote totals shown in Figure 3.12.

represents a location that is referred to as a *marker point*. Axis-parallel lines are drawn through each of the marker points. These lines divide the space into hypercells called *bins*. The voting method eventually assigns a probability to each of the bins. The weighted bins are used to select the point from which the local solver is launched.

Figure 3.12 illustrates how the votes are distributed in each dimension. Each consensus vector gets one vote per dimension The vote starts at the tail of the final consensus vector and continues until it reaches the marker point of another consensus vector that is voting in the opposite direction. The votes are tallied in each dimension. It is assumed that more votes indicate more promise. Similar to the grid-based clustering methods presented in Section 3.3.2, the votes are projected onto the axes to avoid an exponential explosion of possible bins.

Figure 3.13 depicts the final probability distribution of the bins. The darker gray areas represent a higher probability of a bin being selected than the lighter gray areas. A new launch point is determined from the voting results by selecting a random number from each dimension, weighted by the vote totals of the bins. MacLeod suggests squaring the vote totals to exaggerate the relative differences between each bin Furthermore, regions that do not receive any votes still retain a small probability of being chosen for the next start point.

3.5.2 MSNLP

MSNLP is a heuristic multistart algorithm for global optimization [56, 93], available commercially from Optimal Methods, Inc. MSNLP is a two-stage method. The first stage generates a set of starting solutions using a random search routine such as Scatter Search (Section 3.4.1). The points are stored in memory along with their measure of *merit* (as described below). The point deemed to be of the highest quality is used first in stage 2.

Distance and merit filters are used in the second stage of MSNLP. The distance filter ensures that the starting points are diverse, i.e., that none of the starting points are within a basin of attraction of the same local optimum. Meanwhile, the merit filter ensures that points meet a certain quality level. This is achieved via an acceptance-rejection routine. If a point makes it through the filters it is passed to a gradient-based local NLP solver. The solver's output is processed and distinct candidate solutions are stored in a *list of locals.* Distinct points are those that are a certain distance away from any other stored point. In the end, the point in the list of locals with the best objective function value is chosen as the optimum point.

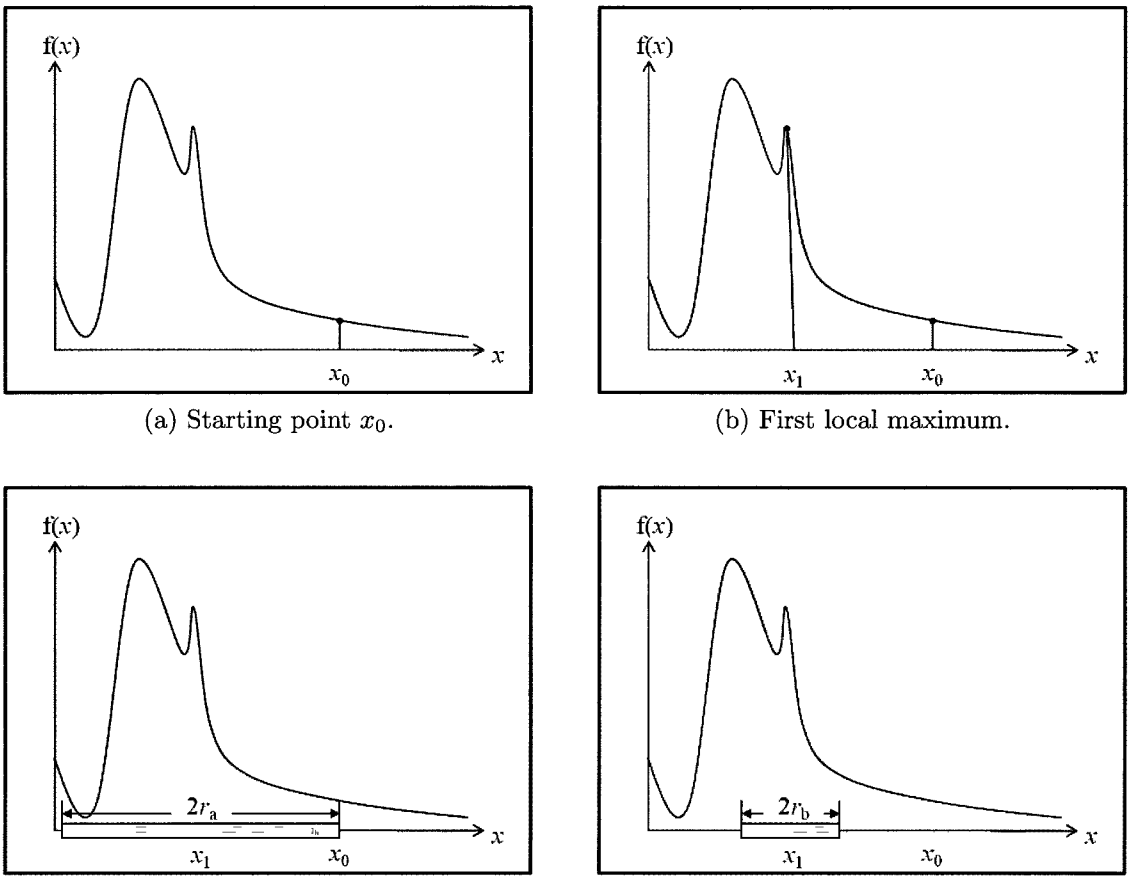
The merit filter for minimization makes use of an L_1 penalty function

$$P(\mathbf{x}, \mathbf{w}) = f(\mathbf{x}) + \sum_{i=1}^{m} w_i v_i(\mathbf{x}) \tag{3.24}$$

where $\mathbf{x} \in \mathcal{R}^n$ is a candidate solution, $\mathbf{w} \in \mathcal{R}^m$ is a vector of positive penalty weights, m is the number of constraints, and $v_i(\mathbf{x})$ is the violation of constraint i. Only points whose penalty values are less than the threshold value are accepted by the merit filter. The initial threshold value is set to the lowest penalty value found in the first stage of the algorithm. Every time a new point is accepted, the threshold is reset to the penalty value of the accepted point. The threshold is increased if too many points are rejected consecutively.

Like clustering, the distance filter attempts to determine points that are close together because they might lead local solvers to the same solutions. MSNLP rejects points that are deemed likely to be in a previously discovered basin of attraction before they are passed to a local NLP solver. Hyperspheres that define basins of attraction are centered around accepted points in order to form a filter. New points that fall within these boundaries are rejected since it is assumed all launch points within a hypershpere will lead a local NLP solver to the same solution. Hypersphere basin formation is illustrated in Fig 3.14.

A new point is rejected if it is within the hypersphere that approximates the basin for any local solution. Furthermore, the basin radius is dynamically decreased if too many



(a) Starting point x_0.

(b) First local maximum.

(c) Initial hypersphere basin.

(d) Basin with reduced radius.

Figure 3.14: MSNLP's distance filter. The object in this example is to maximize $f(x)$. (a) The search is started at x_0. (b) The first local optimum is found at x_1. (c) The basin of attraction is approximated with a hypersphere of radius $r_a = x_0 - x_1$ (in one dimension the hypersphere is an interval). Initially, all samples in the interval $[x_1 - r_a,\ x_1 + r_a]$ are rejected. (d) If too many samples are rejected the radius of the hypersphere is decreased to r_b by multiplying r_a by a user defined scalar between 0 and 1.

points are being rejected (see Fig 3.14d). The initial radius is the distance from a point's starting position to its ending position. A new local point forms an additional basin if it is not within a previously established basin. If a new local point is within an existing basin, the radius of the basin in question is adjusted such that it is the greater of the two possible radii. Finally, any hyperspheres that are found to overlap are scaled back. Specific details of MSNLP are oulined in [56].

3.5.3 GLOBALm

GLOBALm is a recently proposed multistart clustering algorithm for constrained global optimization problems [85]. The method consists of a global phase and a local phase. The algorithm starts the global phase by taking a uniform random sample of points from the variable space. The points are assigned fitness values with an L_1 penalty function (as presented in the previous section). The initial points with the worst fitness levels are rejected. The remaining points are clustered using a single linkage clustering technique.

Clustering is conducted in order to try and find basins of attraction. The single linkage method was chosen so that the basins are free to form in any geometrical shape (unlike the aforementioned MSNLP algorithm that forces spherical basins). The critical distance of the clustering routine depends on the number of samples and the dimension of the search space. The local phase of the GLOBALm algorithm consists of running a local solver once from each point that does not get assigned to a cluster. The algorithm starts out by selecting the initial sample and reducing it. A point is then selected from the reduced set. If it gets assigned to an existing cluster, another point is selected from the reduced set. If it does not get assigned to an existing cluster, it is used to launch a local solver before another point is selected. The clustering process is repeated for a predetermined set of iterations.

3.5.4 Knitro Multistart

Knitro is a local solver designed to solve convex problems and is produced commercially by Ziena Optimization, Inc. Knitro combines complementary approaches to nonlinear optimization to achieve robust performance over a wide range of application requirements [13]. Knitro is designed specifically for large-scale, smooth nonlinear programming problems, nonetheless, it can also be used for unconstrained optimization, nonlinear systems of equations, least squares, and linear and quadratic programming. Here, the term large-scale refers to dimensions into the hundreds of thousands.

The Knitro solver incorporates three algorithms for efficiently solving large-scale prob-

lems, including two interior methods and an active-set method. More specifically, the algorithms are the Interior-point Direct algorithm, the Interior-point Conjugate Gradient algorithm, and the Active Set algorithm [99]. Details and pseudo code for the algorithms of Knitro are provided by Bryd *et al.* in [13].

For nonconvex problems Knitro can run in a multistart mode. To run Knitro in multistart mode each variable in the model must be given a finite upper and lower bound. Knitro randomly selects a predefined number of start points from the search space and launches Knitro from each. The best local solution is returned to the user.

3.6 Summary

This chapter introduces the basics of the multistart method and the current state of the art in using multistart for global optimization of constrained models. The first part of every optimization problem is to determine a starting point. Although initial point placement seems relatively trivial it is actually a vital part of the optimization process. As my supervisor John Chinneck often jokes, the best initial point is the optimum. Methods for seeking feasibility are an extension to the initial point placement. Their goal is often to find a promising launch point for another more computationally expensive solver (see Chapter 5). Although finding feasibility seems like a sub-problem to the optimization problem, both problems are actually equivalent. Solvers designed for optimization can be used to find feasible regions by replacing the original objective by the objective of minimizing the sum of the constraint violations (when this reaches zero a feasible point has been found). Conversely, the objective function is converted to a constraint and continually tightened in order to use a feasibility seeking algorithm to solve for optimality. Specific details are found in Chapter 6 and on page 5 of [58].

New variations to the CC algorithm are proposed in Chapter 5 and are one of the academic contributions of this thesis. Specifically, alterations are proposed for the feasibility and consensus vector calculations of CC. Care should be taken not to confuse the various types of feasibility and consensus vector calculations. As described in Section 3.2, the consensus vector is constructed from the feasibility vectors. Different types of feasibility vectors can be used interchangeably with the available types of consensus. For instance, one variant of CC could use linear feasibility vectors and the FDfar method of consensus.

Basic multistart can be very effective for nonconvex problems. Many of the improvements to the multistart method are combined in different ways to create other algorithms. For instance, the MSNLP and GLOBALm algorithms both use versions of an acceptance-

rejection technique.

Chapter 4

Thesis Statement

The main goal of this thesis is to develop an improved multistart heuristic framework for solving large-scale global optimization problems of the form presented in Section 2.1. The new method is to be effective and efficient, finding global optima frequently and quickly. There is a particular focus on models having complex feasible regions. For instance, a set of nonlinear constraints may have multiple discontiguous feasible regions or the feasible region may be relatively small and difficult to locate. As a result, the secondary goal of this thesis is to devise improved methods for quickly seeking feasibility in large-scale nonlinear programs. Two main developments in this thesis support these goals:

1. *Improvements to the CC algorithm for seeking feasibility.* Novel ways to calculate the feasibility and consensus vectors are presented and the concept of augmentation is introduced. Improved methods for constraint selection and monitoring progress are also presented.

2. *Techniques that use multiple CC runs to quickly explore the variable space and find promising launch points for expensive local solvers.* An algorithm for avoiding redundant and ineffective launches of the expensive local solver is presented. It relies on an original method that automatically determines a critical distance for the single link clustering routine. The method calculates a critical distance by analyzing the inter-point distance distribution of a set of points after they are concentrated towards promising areas of the variable space by CC.

This is not the first attempt to use CC for concentration in a multistart framework. MacLeod presented a multistart/voting method that uses the CC algorithm in his Master's thesis (see Section 3.5.1). The methodology proposed herein is unique in the way the results of the CC runs are analyzed and used to determine promising regions for the global

optimization models. Similarly, the idea of using clustering for optimization is not novel, however, the method for finding the critical distance from the inter-point frequency distribution is. Finally, the techniques proposed to enhance the performance of CC are original and the combination of all the improvements mentioned above results in a promising multistart technique for global optimization. Although global optimization is the subject of intense research, there are still many outstanding issues. A glaring hole in the current state of the art is large-scale global optimization of nonlinear programs. Many of the proposed global optimization algorithms may not work well, if at all, for higher dimensional problems. For instance, a large number of dimensions may lead to slow progress for branching methods such as BARON and GlobSol (Section 2.3.1) as there are a large number of branches to search through. Unbounded problems and problems with particularly large variable spaces emphasize this weakness.

Due to the increased number of variables, large-scale problems can also have objective functions or constraints that require substantial resources to evaluate. The population methods GA and PSO mentioned in Section 2.3.2 that depend upon the ability to evaluate the objective and constraint functions many times may not perform well for such problems. Furthermore, the output from a penalty function might not accurately model problems with many nonlinear constraints. Consider the results published by Parsopoulos and Vrahatis [72]. PSO is applied to 6 relatively small (each has 7 variables or less) nonlinear constrained optimization models using a penalty function. Even for this small set they required two different penalty functions to achieve useful results.

Another unresolved issue is how to deal with multiple feasible regions. This is particularly problematic for probabilistic algorithms such as PSO or SA (and most others) that spend time converging to, and possibly getting stuck in, local optima. Multistart techniques are often used to compensate, but when used rashly the multistart method introduces problems of its own. A major drawback of the basic multistart method is that the local solver may redundantly find the same solution multiple times. This is an inefficient use of search time.

The MCC algorithm described in Section 3.5.1 is a multistart routine that uses CC to find promising launch points for local solvers. The method makes the assumption that the last consensus vector points in the direction of a feasible region. If this is true then a simple line search should be able to find the feasible region; however, this is not usually the case. The voting routine of the MCC method only operates on one dimension at a time. When there is more than one feasible region, the MCC method is likely to act as an averaging scheme and the launch points may very well end up being selected between

promising regions rather than in the regions. This is because density in higher dimensional space implies density in lower dimensional space, but not vice versa. Furthermore, MCC makes no effort to distinguish between possible basins of attraction. As a result the local solver may be launched in the same basin of attraction multiple times.

The multistart algorithm MSNLP (Section 3.5.2) uses a distance filter to identify basins of attraction as hyperspheres. The issue with this method is that sets of nonlinear constraints frequently result in basins of attraction that are not spherical. Consequently, promising points may be rejected by the algorithm before they are processed. The models MSNLP was tested on are relatively small and it is not clear if the method will work well on larger models.

Similarly, GLOBALm was only tested on a set of relatively small models and it is not clear how well the method will scale. Although GLOBALm uses single linkage clustering to allow basins of attraction to take any geometrical shape, the formula used to calculate the critical distance may not be robust. GLOBALm uses a formula based on the number of samples chosen by the operator and the number of dimensions of the problem. It is unclear if this calculation will work with larger problems. In higher-dimensional problems it is also unclear how many initial samples are needed in order to get meaningful clusters from GLOBALm's initial acceptance-rejection routine.

Knitro (Section 3.5.4) is a highly regarded commercial solver for nonlinear programs. Recent releases of Knitro include a global optimization mode. When operating in this mode the Knitro solver is launched from a series of points and the best result is returned as the solution. No effort is made to avoid redundant launches, so efficiency can be improved by the techniques developed in this thesis. This hypothesis is tested in Section 7.3.1.

Global optimization of constrained nonlinear models is an extremely difficult task with many applications. Modelers continue to create larger and more complicated models that require optimization for a variety of reasons. Specific examples of applications are [87]: Chemical engineering (pooling and blending, separation, reactor network synthesis), computational chemistry (molecular design), Civil engineering (stability analysis), Electrical/Systems engineering (robotics, VLSI chip design), and Economics/Business (Nash equilibrium, multicommodity network flow, portfolio optimization, traffic assignment). It is important that the efficiency and effectiveness of algorithms for global optimization of complex large-scale models improve in order to meet the continually growing demand.

Chapter 5

Advances to the Constraint Consensus Method

As described in Section 3.2.3, CC is an algorithm designed to quickly find approximately feasible solution vectors to systems of equations and inequalities. Applying CC to the set of constraints of an optimization problem is a comparatively quick method of approximately locating feasible regions. An iteration of the CC method is relatively fast compared to more elaborate algorithms such as those based on Newton's method because CC does not require the inversion of large matrices or use line or filter searches to adjust step lengths. This chapter explores novel techniques for calculating feasibility and consensus vectors for CC. Experimental results that quantize the advantage of using CC to select launch points for the NLP solver Ipopt are presented and the newly proposed CC variants are shown to outperform their predecessors. Furthermore, the NLP solver launch points found by the CC methods are analyzed in order to determine the general characteristics of a successful point.

5.1 New Constraint Consensus Algorithm Variants

5.1.1 Augmentation

The augmented feasibility vector calculation uses a predictor-corrector approach to adjust the length of the previous consensus vector. By design, augmented iterations of CC do not require gradient calculations at the current iterate. Instead, the step length is calculated using the values of the constraint functions at the current and previous iterates (Eqn. 5.7). Here, the next consensus vector is denoted $(\mathbf{x}_{k+2} - \mathbf{x}_{x+1})$ and the previous consensus vector,

calculated with a standard CC step as shown in Section 3.2.3, is denoted $(\mathbf{x}_{k+1} - \mathbf{x}_k)$.

The augmented feasibility vector calculation resembles the Secant method for root finding [90]. The violated constraint is initially approximated with a first order Taylor series expansion

$$g(\mathbf{x}_{k+2}) \approx g(\mathbf{x}_{k+1}) + \nabla g(\mathbf{x}_{k+1})(\mathbf{x}_{k+2} - \mathbf{x}_{k+1}). \tag{5.1}$$

The previous consensus vector $(\mathbf{x}_{k+1} - \mathbf{x}_k)$ is re-used as the search direction, but the relaxation value α_{ag} is applied as the step length multiplier, i.e.,

$$(\mathbf{x}_{k+2} - \mathbf{x}_{k+1}) = \alpha_{ag}(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{5.2}$$

Substituting Eqn. 5.2 into Eqn. 5.1, the equation approximating the violated constraint becomes

$$g(\mathbf{x}_{k+2}) \approx g(\mathbf{x}_{k+1}) + \alpha_{ag}\nabla g(\mathbf{x}_{k+1})(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{5.3}$$

Start again with the Taylor series expansion about $\mathbf{x}_{k+1}$,

$$g(\mathbf{x}_k) \approx g(\mathbf{x}_{k+1}) + \nabla g(\mathbf{x}_{k+1})(\mathbf{x}_k - \mathbf{x}_{k+1}), \tag{5.4}$$

and rearrange such that

$$\nabla g(\mathbf{x}_{k+1})(\mathbf{x}_{k+1} - \mathbf{x}_k) \approx g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k), \tag{5.5}$$

then substitute the result into Eqn. 5.3 to give

$$g(\mathbf{x}_{k+2}) \approx g(\mathbf{x}_{k+1}) + \alpha_{ag}(g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k)). \tag{5.6}$$

Feasibility vectors approximate the length and direction of a step to the root of the violated constraint function, i.e., $g(\mathbf{x}_{k+2}) = 0$. Therefore, the step size is determined by setting Eqn. 5.6 to zero and solving:

$$\alpha_{ag} = \frac{-g(\mathbf{x}_{k+1})}{g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k)}. \tag{5.7}$$

The next feasibility vector for constraint g is

$$(\mathbf{x}_{k+2} - \mathbf{x}_{k+1}) = \frac{-g(\mathbf{x}_{k+1})}{g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k)}(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{5.8}$$

After an augmented feasibility vector (Eqn. 5.8) is calculated for each violated constraint, the consensus vector is formed using one of the consensus methods described in

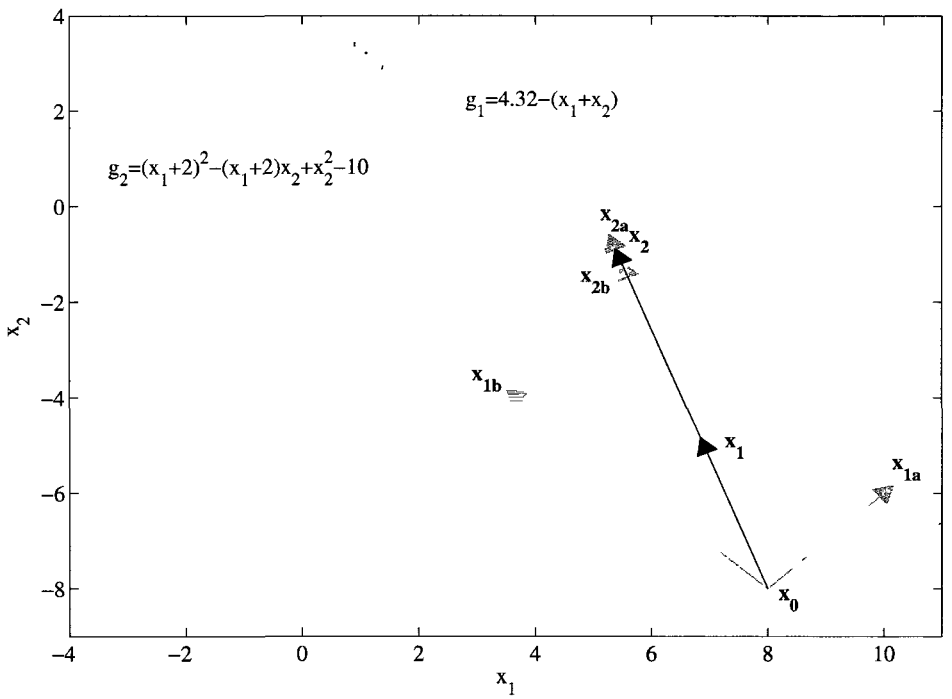


Figure 5.1: Two CC iterations: the first uses linear feasibility vectors, and the second uses augmented feasibility vectors. Both iterations use Basic consensus.

Section 3.2.3. Results from using augmentation with various forms of consensus are reported in Chapter 5.

The denominator of Eqn 5.7 is precarious because it causes the step size to be abnormally large when $g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k) \approx 0$. The algorithmic framework of CC has multiple fail-safes to deal with these circumstances. The feasibility tolerance eliminates short feasibility vectors from contention in the consensus decision and the movement tolerance ensures that CC exits if the consensus vector is not of sufficient length. Furthermore, CC is restrained such that if the intermediate search point at any iteration is outside of the search space it is immediately reset such that it satisfies the model's bounds. Therefore large steps caused by augmentation, or any other reason, do not cause the method to fail.

An augmented feasibility vector calculation for the model presented in Section 3.2.3 is illustrated in Fig. 5.1. This example shows how augmentation is applied to the original CC algorithm that uses Basic consensus and linear feasibility vector calculations. The first step shown, $(\mathbf{x}_1 - \mathbf{x}_0)$, is the consensus vector calculated previously in Section 3.2.3. The second step, $(\mathbf{x}_2 - \mathbf{x}_1)$, is a consensus vector formed using the augmented feasibility vector calculations. For example, given points $\mathbf{x}_0$ and $\mathbf{x}_1$, the augmented step size for violated

constraint g_2 is

$$\alpha_{ag} = \frac{-g_2(\mathbf{x}_1)}{g_2(\mathbf{x}_1) - g_2(\mathbf{x}_0)} \tag{5.9}$$

$$= \frac{-134.2}{134.2 - 234.0} \tag{5.10}$$

$$= 1.345 \tag{5.11}$$

and the augmented feasibility vector is

$$(\mathbf{x}_{2a} - \mathbf{x}_1) = \alpha_{ag}(\mathbf{x}_1 - \mathbf{x}_0) \tag{5.12}$$

$$= \begin{bmatrix} -1.565 \\ 4.254 \end{bmatrix}. \tag{5.13}$$

The result is shown in Fig. 5.1 as the vector $(\mathbf{x}_{2a} - \mathbf{x}_1)$. The same operation is applied to the other violated constraint g_1 and the resulting vector $(\mathbf{x}_{2b} - \mathbf{x}_1)$ is also shown in Fig. 5.1. In this case, the consensus vector is formed using Basic consensus. The result is illustrated as the vector $(\mathbf{x}_2 - \mathbf{x}_1)$ in Fig. 5.1. After the two iterations depicted, the violations of the constraints g_1 and g_2 at $\mathbf{x}_2$ are 0.185 and 51.653, respectively.

A key parameter for implementing the augmentation method is the recurrence period (T), that determines the frequency at which the augmented calculation is performed. For instance, if $T = 2$ then augmentation is applied to every second iteration. If $T = 3$ then augmentation is applied to one of every three iterations. The minimum recurrence period is $T_{min} = 2$ because augmentation requires the previous consensus vector as a search direction. Theoretically, augmentation could be applied to the same consensus vector more than once. In this case the algorithm would be restricted to searching along the same line for multiple iterations.

To be consistent, for all the experiments reported herein the augmented calculations took place in the second iteration of a cycle, even when $T > 2$.

5.1.2 Quadratic Feasibility Vectors

The original CC algorithm is explained in Section 3.2.3. Specifically, the linear feasibility vector formula (Eqn. 3.8) is derived from the first order Taylor series expansion of the constraint function g at the location $\mathbf{x}_k$. It is natural to question whether higher order approximations afford CC any advantages. This section outlines the quadratic feasibility

vector calculation based on the second order Taylor series expansion:

$$g(\mathbf{x}_{k+1}) \approx g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x}_{k+1} - \mathbf{x}_k)^T \nabla^2 g(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{5.14}$$

As in Section 3.2.3, the desired movement direction is parallel to the gradient of the violated constraint at $\mathbf{x}_k$. Therefore, the quadratic feasibility vector is defined

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_q \nabla g(\mathbf{x}_k)^T, \tag{5.15}$$

where α_q is the step size. Using the quadratic feasibility vector definition $g(\mathbf{x}_{k+1})$ is approximately

$$g(\mathbf{x}_{k+1}) \approx g(\mathbf{x}_k) + \alpha_q ||\nabla g(\mathbf{x}_k)^T||^2 + \frac{\alpha_q^2}{2} \nabla g(\mathbf{x}_k) \nabla^2 g(\mathbf{x}_k) \nabla g(\mathbf{x}_k)^T. \tag{5.16}$$

The quadratic feasibility vector step size is determined by setting $g(\mathbf{x}_{k+1}) = 0$ and solving

$$\alpha_q = \frac{-b \pm [b^2 - 4ac]^{\frac{1}{2}}}{2a}, \tag{5.17}$$

where

$$a = \frac{1}{2} \nabla g(\mathbf{x}_k) \nabla^2 g(\mathbf{x}_k) \nabla g(\mathbf{x}_k)^T \tag{5.18}$$

$$b = ||\nabla g(\mathbf{x}_k)^T||^2 \tag{5.19}$$

$$c = g(\mathbf{x}_k). \tag{5.20}$$

Due to the quadratic nature of Eqn 5.17, two distinct step sizes or a complex step size may result. Both issues are related to the discriminant

$$\gamma = b^2 - 4ac. \tag{5.21}$$

Three possibilities exist for the step size

1. $\gamma > 0$: There are two distinct real solutions because the quadratic approximation to the constraint function (Eqn. 5.16) is equal to zero at two real points. For instance, the parabola $f(x) = x^2 - 1$ crosses zero at $x = \pm 1$.

2. $\gamma = 0$: There is one distinct real solution because the quadratic approximation to the constraint function is equal to zero at only one real point. An example is the

parabola $f(x) = x^2$ which equals zero only at $x = 0$.

3. $\gamma < 0$: There are two distinct complex solutions because the quadratic approximation to the constraint function does not equal zero in the real space. An example is the parabola $f(x) = x^2 + 1$.

When $\gamma > 0$ the smaller of the two real roots is chosen to reduce the possibility of cycling between multiple solutions. When $\gamma < 0$ there are no real roots $\mathbf{x}_{k+1}$ such that $g(\mathbf{x}_{k+1}) = 0$ and a critical point of the constraint function $g(\mathbf{x}_{k+1})$ is used instead. This technique is realized by setting the derivative of Eqn. 5.16 with respect to the step size to zero

$$0 = ||\nabla g(\mathbf{x}_k)^T||^2 + \alpha_q \nabla g(\mathbf{x}_k) \nabla^2 g(\mathbf{x}_k) \nabla g(\mathbf{x}_k)^T, \tag{5.22}$$

and solving

$$\alpha_q = \frac{-||\nabla g(\mathbf{x}_k)^T||^2}{\nabla g(\mathbf{x}_k) \nabla^2 g(\mathbf{x}_k) \nabla g(\mathbf{x}_k)^T}. \tag{5.23}$$

Further inspection reveals that this technique offers the same result as simply taking the real part of Eqn. 5.17 when $\gamma < 0$.

Linear feasibility vector calculations are exact for linear constraints, so there is no advantage gained by using an additional quadratic term while calculating the feasibility vector of linear constraints. For quadratic constraints, the quadratic feasibility vector calculation is exact and therefore offers an accuracy advantage over the linear calculation. However, the quadratic term comes with additional computational cost. Section 5.3 explores the benefits and costs of the quadratic feasibility vector method in more detail.

The following example demonstrates the possible advantage of using the quadratic feasibility calculation. Consider again the model described in section 3.2.3. The Hessian matrix for the nonlinear constraint is

$$\nabla^2 g_2(\mathbf{x}) = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}. \tag{5.24}$$

Given the initial point $\mathbf{x}_0 = [8 \quad -8]^T$ the quadratic parameters are:

$$\begin{aligned} a &= \frac{1}{2}\nabla g_2(\mathbf{x}_0)\nabla^2 g_2(\mathbf{x}_0)\nabla g_2(\mathbf{x}_0)^T && (5.25)\\ &= \frac{1}{2}[28 \quad -26]\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} 28 \\ -26 \end{bmatrix} && (5.26)\\ &= 2188, && (5.27)\\ b &= ||\nabla g_2(\mathbf{x}_0)^T||^2 && (5.28)\\ &= [28 \quad -26]\begin{bmatrix} 28 \\ -26 \end{bmatrix} && (5.29)\\ &= 1460, && (5.30)\\ c &= g_2(\mathbf{x}_0) && (5.31)\\ &= 234. && (5.32) \end{aligned}$$

In this case, $\gamma = 83632 > 0$ so there are two real roots and therefore two possible values for α_q (-0.400 and -0.268). The shortest step size $\alpha_q = -0.268$ is chosen. The quadratic feasibility vector is

$$\begin{aligned} (\mathbf{x}_{1b} - \mathbf{x}_0) &= \alpha_q \nabla g_2(\mathbf{x}_0)^T && (5.33)\\ &= \begin{bmatrix} -7.492 \\ 6.956 \end{bmatrix}. && (5.34) \end{aligned}$$

The linear and quadratic feasibility vectors of violated constraints g_1 and g_2, respectively, are illustrated in Fig. 5.2. At the point $\mathbf{x}_2$ in Fig. 5.2, after two steps of quadratic constraint consensus, the violations of constraints g_1 and g_2 are 1.669 and 32.138, respectively. Note that even though both of the feasibility vectors are exact, the consensus vector $(\mathbf{x}_1 - \mathbf{x}_0)$ is still just an estimate.

5.1.3 SUM Consensus

The FDfar variant for consensus chooses the longest feasibility vector as the consensus vector (see Section 3.2.3) and is one of the most successful CC methods [42]. The predominant weakness of FDfar consensus is that it tends to cycle between constraints as it repeatedly tries to remedy the most violated constraints. SUM consensus is an alternative that may avoid cycling while still exploiting long feasibility vectors. The SUM consensus vector is built by summing all the feasibility vectors at the current iterate. Longer feasibility vec-

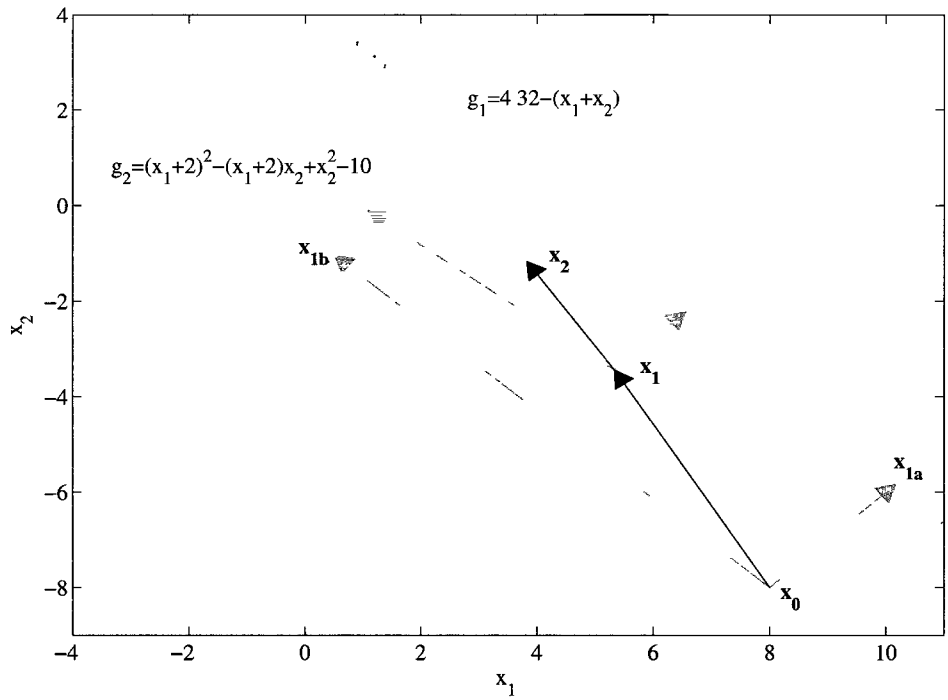


Figure 5.2: Two steps of CC using quadratic feasibility vectors and Basic consensus.

tors maintain more influence than shorter vectors, however unlike FDfar consensus, every violated constraint still provides input to the consensus vector.

An example showing two steps of the SUM consensus method with linear feasibility vectors for the model presented in Section 3.2.3 is shown in Fig. 5.3. Both constraints g_1 and g_2 are violated at the initial point $\mathbf{x}_0 = [8 \quad -8]^T$ and the linear consensus vectors are

$$(\mathbf{x}_{1a} - \mathbf{x}_0) = \begin{bmatrix} 2.160 \\ 2.160 \end{bmatrix} \tag{5.35}$$

and

$$(\mathbf{x}_{1b} - \mathbf{x}_0) = \begin{bmatrix} -4.488 \\ 4.167 \end{bmatrix}. \tag{5.36}$$

SUM consensus calculates the next point $\mathbf{x}_1$ by adding the feasibility vectors of the

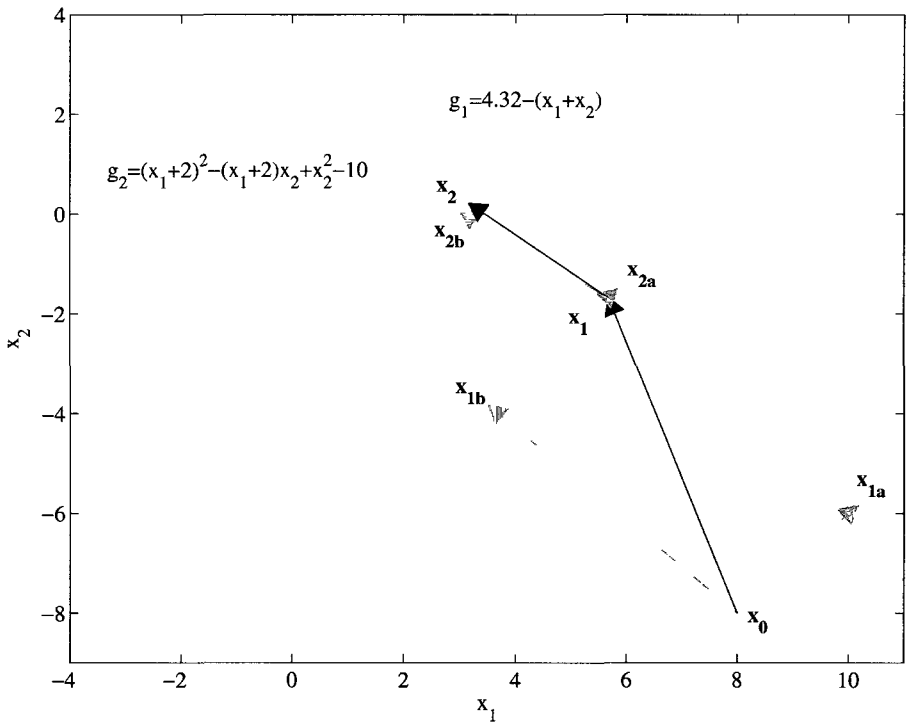


Figure 5.3: Two iterations of CC using linear feasibility vectors and SUM consensus.

violated constraints to the previous point, $\mathbf{x}_0$

$$\mathbf{x}_1 = \begin{bmatrix} 8 \\ -8 \end{bmatrix} + \begin{bmatrix} 2.160 \\ 2.160 \end{bmatrix} + \begin{bmatrix} -4.488 \\ 4.167 \end{bmatrix} \tag{5.37}$$

$$= \begin{bmatrix} 5.672 \\ -1.673 \end{bmatrix} \tag{5.38}$$

Fig. 5.3 depicts two iterations of SUM consensus. Although the algorithm uses linear feasibility vectors in this example, SUM consensus could operate on quadratic feasibility vectors instead. Similarly, the SUM method could be augmented as described in Section 5.1.1.

5.1.4 Pairing Constraint Consensus with Nonlinear Solvers

CC is a computationally cheap algorithm that can provide a high quality starting point for an expensive local solver. The net result is that the combined solution time is usually shorter than simply launching the local solver from the original point [42]. To be effective, CC should output points that have the characteristics that most contribute to the speed and success of the local solver. This section introduces two minor CC variations that

significantly improve the performance of the local solver.

Operating on Nonlinear Constraints

The main step of many NLP solvers is to manipulate a large linear system. In this system the nonlinear constraints are linearly approximated, but the linear constraints are dealt with exactly. As usual, the nonlinear constraints are the cause of most of the difficulty, therefore using the limited amount of initial CC time to reduce the degree of violation of the nonlinear constraints provides the most overall benefit. Limiting CC to operate on only the subset of nonlinear constraints may provide higher quality launch points for the local solver. Experimental results that show the benefit of this heuristic are presented in Section 5.4.1.

Choosing the CC Output Point

Experimental data shows that a positive correlation exists between launch points with low maximum constraint violation (as defined in Section 5.2.5) and solver success [42]. An interesting observation about CC is that it may not monotonically reduce the maximum constraint violation at each iteration. The violation may increase or decrease in successive iterations. CC may perform better if the violation is monitored at each iteration and the local solver is launched from whichever point has the lowest violation.

MacLeod [58] proposed a similar idea. He tracked CC progress via consensus vector length in order to decide when to terminate. In his experiments the CC algorithm was stopped when the consensus vector length failed to decrease by a certain threshold. The point at the final iterate was returned to launch the solver. In contrast, the idea proposed here is to allow CC to proceed until it meets one of the original termination conditions proposed by Chinneck [17], then to use whichever point has the lowest violation (intermediate or final) to launch the local solver.

5.2 Experimental Setup

5.2.1 Hardware and Software

All experiments were conducted on a machine with an Intel®Core™2 Duo Processor E6600 (4M Cache, 2.40 GHz, 1066 MHz FSB) and 3GB of memory running Linux Fedora Core 6. The code is compiled with gcc 4.1.2. The models are formatted in A Modeling Language for Mathematical Programming (AMPL) [31].

The local solver used in the experiments is Ipopt 3.5.5 [97]. Ipopt is an interior-point algorithm for nonlinear programming. Two distinguishing elements of the Ipopt algorithm are its filter line-search method and its feasibility restoration phase [98]. Ipopt is chosen due to its performance relative to other solvers (see [67] for a comparison of NLP solvers) and because it is open source. Ipopt's parameter settings are listed in Section 5.2.3.

5.2.2 The Problem Sets

The experiments in this chapter use the COntinuous CONstraints - Updating the Technology (COCONUT) benchmark [87], a collection of over 1000 problems from a variety of fields, including computational chemistry, robotics, operations research, and economics. Large nonlinear models are of particular interest, so models with 10 or fewer nonlinear constraints were removed from the test set. Furthermore, some models were removed as they caused software errors. The argauss and grouping models were removed because they caused Ipopt to return the runtime exception message *Too few degrees of freedom.* The models argtrig, bratu2d, bratu2dt, bratu3d, camcge, catenary, cbratu2d, ex8_3_13, ex8_3_14, ex8_6_1, oet7, pindyck, and polak3 were removed because the they caused the AMPL interface function *conval()* [33], which is used extensively by the CC software, to return an error.

Model Statistics for Experiment A

Experiment A explores the relative performance of the CC variants over multiple model sizes. For this experiment the test models are separated into three subsets:

- Problem Set IA (PS IA). This set contains 79 models with more than 10 nonlinear constraints and fewer than 101 nonlinear constraints.
- Problem Set IIA (PS IIA). This set includes 62 models with more than 100 nonlinear constraints and fewer than 1001 nonlinear constraints.
- Problem Set IIIA (PS IIIA). This is a set of 72 models, each with more than 1000 nonlinear constraints.

The statistics of the selected test models are presented in Table 5.1. The constraint statistics are divided by constraint type. In this case, the set of nonlinear constraints includes quadratic constraints. The set of all constraints includes both linear and nonlinear constraints.

Table 5.1: Summary Statistics for Test Models A

	PS IA			PS IIA			PS IIIA		
	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.
Variables	78.4	2	720	1079.5	6	8997	6341.1	3	20008
Constraints:									
All	60.3	11	263	975.1	101	7000	5606.2	1002	14000
Quadratic	22.9	0	100	277.8	0	999	2261.0	0	13798
Nonlinear	39.4	11	100	464.7	101	1000	4112.9	1002	13798

Table 5.2: Summary Statistics for Test Models B

	PS IB			PS IIB			PS IIIB		
	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.
Variables	1877.6	2	20008	7041.7	6	13805	1188.4	42	10000
Constraints:									
All	1599.4	11	14000	6738.9	1600	13800	732.2	20	5000
Quadratic	481.8	0	10000	3899.2	0	13798	2.9	0	27
Nonlinear	985.4	11	10000	5431.1	1521	13798	665.1	20	5000

Model Statistics for Experiment B

Experiment B compares the effectiveness of the CC variants when they are paired with a local solver. In this case the results were used to divide the models a posteriori so that the performance statistics presented are meaningful. The three subsets are:

- Problem Set IB (PS IB). This set contains the 175 models that did not cause any timeouts or solver failures.
- Problem Set IIB (PS IIB). This set includes 25 models that caused at least one timeout, but no solver failures.
- Problem Set IIIB (PS IIIB). This is a set of 13 models that caused at least one solver failure.

The statistics of the test models are presented in Table 5.2.

Starting Points

The difficulty of an optimization model is often conditional on the starting point. For instance, a large nonlinear model is easy to solve if the given starting point is the optimum.

To mitigate the effects of starting points, the experiments in this chapter are run in a Monte Carlo style. Each algorithm is run ten times for each model, i.e., 2130 runs in total. All of the algorithms are launched from the same randomly selected start points for each model so that the starting conditions are fair.

Artificial Bounds

For all experiments presented in this thesis, the random start points are chosen within the variable bounds. For any variable that is unbounded (upper, lower, or both), an artificial bound, b_a, is imposed as follows:

- If $\ell_j = -\infty$ and $u_j = \infty$, then the imposed artificial bounds are $-b_a \leq x_j \leq b_a$.
- If $\ell_j = -\infty$ and $u_j \neq \infty$, then the imposed artificial bound is $u_j - b_a \leq x_j \leq u_j$.
- If $\ell_j \neq -\infty$ and $u_j = \infty$, then the imposed artificial bound is $\ell_j \leq x_j \leq \ell_j + b_a$.

The default value used in the experiments presented throughout this thesis is $b_a = 10^4$. This is the same value chosen by Ugray et al. to test the MSNLP solver [93] (Section 3.5.2). These bounds are used for generating points within the variable space, but the bounds are not communicated to the local solvers. Therefore, the local solvers are not restricted to operating within the artificial bounds.

Experimental results justifying the value of $b_a = 10^4$ are reported by MacLeod [58]. Note that MacLeod uses a slightly different set of rules to define the artificial bounds. For instance, if $u_j - \ell_j > 2b_a$, MacLeod introduces the artificial bounds $C - b_a \leq x_j \leq C + b_a$, where $C = \ell_j + \frac{u_j - \ell_j}{2}$ is the midpoint between the original finite bounds.

5.2.3 Description of the Experiments and Parameter Settings

Experiment A: Finding Approximately Feasible Points

This experiment compares the ability of the proposed CC algorithm variants to find points that are close to feasible regions. The algorithms are constrained by a maximum run time and a maximum number of iterations. The feasibility tolerance is $\alpha = 10^{-16}$ and the movement tolerance is $\beta = 10^{-16}$. Small values for these parameters ensure the CC variants do not terminate before the time limit unless they find a point relatively close to feasibility or have stopped making sufficient progress. The maximum number of iterations is $\mu = 100$. A time limit (*max_time*) specific to each problem set is enforced. The time limit is greater for the models with more nonlinear constraints. For PS IA $max\_time = 0.05s$, for PS IIA

$max\_time = 0.5s$, and for PS IIIA $max\_time = 5.0s$. The time limits are based on previous experiments that are not reported here.

The second part of this experiment compares results from using the CC end point and the best CC intermediate point to lower constraint violation, as described in Section 5.1.4.

Experiment B: Pairing Constraint Consensus with a Local Solver

This experiment examines the ability of the CC variants to improve local solver performance, particularly Ipopt [98]. The experiment consists of two stages: pre-solve and solve. In the pre-solve stage a CC variant starts from a randomly selected point and returns a local solver launch point that is closer to feasibility. In the solve stage Ipopt is launched from the point output of the CC variant. The feasibility tolerance is $\alpha = 10^{-3}$ and the movement tolerance is $\beta = 10^{-6}$. These parameter values ensure that CC terminates quickly if it finds a promising launch point or it is failing to make sufficient progress. The maximum number of CC iterations and the CC time limit are the same as those imposed in Experiment A.

The parameter settings for Ipopt are:

- $honor\_original\_bounds = yes$
- $bound\_relax\_factor = 0$
- $max\_iter = 9999999$
- $constr\_viol\_tol = 10^{-6}$
- $max\_cpu\_time = 60.$

All other Ipopt settings are left as default.

5.2.4 Algorithm Variants and their Abbreviations

The main purpose of Experiment A is to determine the most promising CC variants so that future tests can concentrate on fewer algorithms. In all, 18 variants are tested in Experiment A. The best point found by the CC algorithm is the output point (see Section 5.1.4). Also, unless otherwise stated, every algorithm operates on all of each model's constraints. Three variations using linear feasibility vector calculations and without augmentation are tested:

L_Basic - Feasibility vector: Linear, Consensus: Basic, Augmentation: no.

L_FDfar - Feasibility vector: Linear, Consensus: FDfar, Augmentation: no.

L_SUM - Feasibility vector: Linear, Consensus: SUM, Augmentation: no.

L_Basic is the variant initially proposed in [17]. L_FDfar is chosen because it is one of the best performing CC variations [42]. The L_SUM method is the CC variant proposed in Section 5.1.3.

Nine variations of CC with augmentation are tested:

L_Basic_A_3 - Feasibility vector: Linear, Consensus: Basic, Augment: yes, T: 3.

L_Basic_A_11 - Feasibility vector: Linear, Consensus: Basic, Augment: yes, T: 11.

L_Basic_A_17 - Feasibility vector: Linear, Consensus: Basic, Augment: yes, T: 17.

L_FDfar_A_3 - Feasibility vector: Linear, Consensus: FDfar, Augment: yes, T: 3.

L_FDfar_A_11 - Feasibility vector: Linear, Consensus: FDfar, Augment: yes, T: 11.

L_FDfar_A_17 - Feasibility vector: Linear, Consensus: FDfar, Augment: yes, T: 17.

L_SUM_A_3 - Feasibility vector: Linear, Consensus: SUM, Augment: yes, T: 3.

L_SUM_A_11 - Feasibility vector: Linear, Consensus: SUM, Augment: yes, T: 11.

L_SUM_A_17 - Feasibility vector: Linear, Consensus: SUM, Augment: yes, T: 17.

Larger numbers for the recurrence period result in augmentation happening less frequently (see Section 5.1.1). Periods of 3, 11 and 17 are chosen so that the relative performance over a wide range of recurrence periods is gauged.

Finally, six variations using quadratic feasibility vectors are tested, none of them are augmented:

Q_Basic_q - Feasibility vector: quadratic, Consensus: Basic, quadratic constraints only.

Q_Basic_n - Feasibility vector: quadratic, Consensus: Basic, all nonlinear constraints.

Q_FDfar_q - Feasibility vector: quadratic, Consensus: FDfar, quadratic constraints only.

Q_FDfar_n - Feasibility vector: quadratic, Consensus: FDfar, all nonlinear constraints.

Q_SUM_q - Feasibility vector: quadratic, Consensus: SUM, quadratic constraints only.

Q_SUM_n - Feasibility vector: quadratic, Consensus: SUM, all nonlinear constraints.

These methods are divided into two sub-categories: q and n. The variants denoted with q calculate quadratic feasibility vectors for only the quadratic constraints and linear feasibility vectors for all other constraints. The variants denoted with n calculate quadratic feasibility vectors for all the nonlinear constraints, but linear feasibility vectors for the linear constraints.

Results for the combination of quadratic feasibility vectors and augmentation were not collected. Initial results (not reported here) indicated that this combination was not competitive with the other methods. Furthermore, the quadratic variants of CC proved to be relatively inefficient when compared to the linear CC methods (see Section 5.3.1).

5.2.5 Performance Metrics

The CC algorithm is designed to find a point that is close to the feasible region, therefore the main metric used to judge CC performance is the maximum constraint violation

$$\mathcal{V}(\mathbf{x}_k) = \max\left(v_i(\mathbf{x}_k), ..., v_m(\mathbf{x}_k)\right), \tag{5.39}$$

where $\mathbf{x}_k$ is the current iterate. A successful CC run finds a point for which $\mathcal{V}$ is approximately zero. Throughout this thesis a point is considered feasible if and only if $\mathcal{V}(\mathbf{x}_k) \leq 10^{-6}$.

The Monte Carlo experiments collect results for every model by starting each algorithm from 10 randomly selected points. Experiment A results include:

- Avg. Time(s) - The average time in seconds,
- Avg. Its - The average number of iterations,
- Median $\mathcal{V}$ - The median of the maximum constraint violations, and
- Rank - The relative ranking of the algorithms in terms of median violation.

. For instance, Table 5.3 indicates that half of the solutions found by the L_Basic variant have $\mathcal{V} < 29,800$. Meanwhile, half of the solutions found by L_Basic_A_3, the top ranked algorithm, have solutions with $\mathcal{V} < 117$.

The results from the second part of Experiment A are reported in Table 5.6. The end point is the last point that the CC algorithm finds whereas the best point is the point with the lowest violation found throughout the run. This table includes the fraction of runs where the best point has a lower violation than the end point in the column titled *Fraction of Runs with Best < End.*

The results of both the CC variants and Ipopt are reported for Experiment B. All the average times are listed in seconds. The *timeout fraction* is the ratio of runs for which an algorithm failed to terminate before the maximum time limit to the total number of runs. The *feasibility fraction* is the ratio of runs for which an algorithm found a feasible solution to the total number of runs. The *fail fraction* is the ratio of runs for which Ipopt returned an error to the total number of runs. The *infeasible fraction* is the ratio of runs for which an algorithm found an infeasible final solution to the total number of runs. In these cases, Ipopt converged to a locally infeasible solution.

Part of Experiment B is an analysis of the launch points passed from the CC variants to the local solver. For this part of the experiment the following statistics are reported:

Viol - The average number of violated constraints.

OverSat - The average number of inequality constraints satisfied by at least 1 ($g_i(\mathbf{x}_k) \leq -1$).

JustSat - The average number of inequality constraints satisfied by less than 1 ($-1 < g_i(\mathbf{x}_k) \leq 0$).

Wtol - The average number of inequality constraints within the tolerance ($0 < g_i(\mathbf{x}_k) \leq 10^{-6}$). Note that the JustSat and Wtol sets do not overlap.

Close - The average number of equality constraints close to being satisfied ($v_i(\mathbf{x}_k) \leq 1$).

The relationship between constraint function values and violation for equality and inequality constraints is found in Section 2.1.

5.3 Numerical Results

5.3.1 Experiment A: Finding Approximately Feasible Points

This section describes the results of an experiment designed to compare the CC variants' relative abilities at quickly reducing constraint violation. It tests the 18 algorithm variants introduced in Section 5.2.4 with problem sets IA, IIA, and IIIA. The results of Experiment A are reported in Tables 5.3, 5.4, and, 5.5.

The performance of the four algorithms that use Basic consensus and linear feasibility vectors improves as the frequency of augmentation is increased. Of these four algorithms the original version that does not use augmentation at all, L_Basic, performed the worst

Table 5.3: Results for PS IA. The median $\mathcal{V}$ for the start points is $1,710,000$.

Variant	**Avg. Time(s)**	**Avg. Its**	**Median $\mathcal{V}$**	**Rank**
L_Basic_A_3	0.01	87.47	117	1
L_FDfar	0.01	88.40	125	2
L_SUM	0.01	87.69	323	3
L_FDfar_A_17	0.01	87.01	390	4
L_Basic_A_11	0.01	90.23	537	5
L_FDfar_A_11	0.01	86.80	723	6
L_Basic_A_17	0.01	90.76	742	7
L_SUM_A_11	0.01	86.55	857	8
L_SUM_A_17	0.01	86.83	1,525	9
L_SUM_A_3	0.01	90.31	1,740	10
L_FDfar_A_3	0.01	87.20	2,140	11
Q_SUM_q	0.03	68.90	3,197	12
Q_FDfar_q	0.03	69.23	5,114	13
Q_FDfar_n	0.03	65.39	7,914	14
Q_SUM_n	0.03	64.65	8,212	15
L_Basic	0.01	93.07	29,800	16
Q_Basic_q	0.03	71.47	67,987	17
Q_Basic_n	0.04	66.80	148,440	18

Table 5.4: Results for PS IIA. The median $\mathcal{V}$ for the start points is $1,255,000$.

Variant	Avg. Time(s)	Avg. Its	Median $\mathcal{V}$	Rank
L_Basic_A_3	0.24	77.14	140	1
L_FDfar_A_17	0.24	78.82	455	2
L_FDfar_A_11	0.24	78.78	456	3
L_SUM_A_17	0.23	73.18	501	4
L_SUM_A_11	0.23	74.36	502	5
L_FDfar_A_3	0.22	80.18	1,065	6
L_FDfar	0.24	79.62	1,075	7
L_Basic_A_11	0.27	77.08	1,275	8
L_SUM	0.24	76.82	1,315	9
L_SUM_A_3	0.22	81.67	1,540	10
L_Basic_A_17	0.27	76.64	1,725	11
Q_SUM_q	0.42	45.71	4,023	12
Q_FDfar_q	0.43	46.07	5,351	13
Q_SUM_n	0.46	29.67	20,609	14
Q_FDfar_n	0.47	29.26	174,120	15
L_Basic	0.27	76.66	280,500	16
Q_Basic_q	0.44	42.74	412,720	17
Q_Basic_n	0.47	28.61	768,010	18

for each problem set. Conversely, L_Basic_A_3 is the top ranked algorithm out of all 18 variants tested for each problem set.

No obvious trend exists between augmentation frequency and success for the four algorithms that use FDfar consensus and linear feasibility vectors. The relative rank of these variants changes for each of the problem sets reported in Tables 5.3, 5.4, and, 5.5. Of these algorithms, L_FDfar is the highest ranking on average and it never ranks lower than 7th.

The SUM consensus methods that use linear feasibility vector calculations perform relatively well. Like the FDfar-based methods, no trend exists between augmentation frequency and success for the SUM consensus methods. Overall, the L_SUM variant stands out, ranking third overall for both problem sets IA and IIIA.

All the variants that use quadratic feasibility vectors perform relatively poorly in terms of median violation. The quadratic algorithms outperform their linear counterparts in only a few cases. The algorithms that only apply the quadratic feasibility vector calculation to quadratic constraints always perform better than the algorithms that apply the quadratic feasibility vector to all nonlinear constraints. Q_SUM_q is the best performing quadratic

Table 5.5: Results for PS IIIA. The median $\mathcal{V}$ for the start points is $1,580,000$.

Variant	**Avg. Time(s)**	**Avg. Its**	**Median $\mathcal{V}$**	**Rank**
L_Basic_A_3	3.87	52.24	1,765	1
L_SUM_A_11	3.78	53.78	2,540	2
L_SUM	3.77	51.80	3,760	3
L_SUM_A_17	3.79	53.49	3,760	3
L_SUM_A_3	3.67	58.87	4,110	5
L_Basic_A_11	3.96	48.13	5,595	6
L_FDfar	3.80	52.56	8,965	7
Q_SUM_q	4.50	24.82	19,170	8
L_Basic_A_17	3.99	47.90	91,400	9
L_FDfar_A_3	3.71	59.70	251,000	10
L_FDfar_A_11	3.80	54.28	335,000	11
L_FDfar_A_17	3.81	53.73	481,000	12
L_Basic	4.03	46.87	531,500	13
Q_FDfar_q	4.52	25.36	679,260	14
Q_SUM_n	4.88	8.51	935,445	15
Q_FDfar_n	4.96	6.25	971,490	16
Q_Basic_q	4.61	22.39	1,034,500	17
Q_Basic_n	4.97	6.32	1,163,200	18

variant overall, although it is relatively weak ranking 12th, 12th, and 8th overall for problem sets IA, IIA, and IIIA, respectively.

SUM is a relatively strong version of consensus. The algorithm variations that use SUM consensus rank relatively well compared to their peers. This is especially true for PS IIIA which contains the largest models. Four of the five top variants are SUM consensus for PS IIIA. SUM is also the best performing consensus type when paired with quadratic feasibility vectors. As mentioned previously, Q_SUM_q is the highest ranked quadratic method in each problem set.

The CC Output Point

This experiment tests how often a CC intermediate point has a lower violation than the CC end point. The experiment is run with two sets of CC parameters to show that the advantage gained using the intermediate points is valid for a wide range of tolerances. Three of the best performing algorithms from Experiment A are chosen for this experiment, namely, L_FDfar, L_SUM, and L_Basic_A_3, along with the original CC variant, L_Basic. The results in Table 5.6 indicate that the best intermediate point often has a lower constraint violation than the CC end point. In particular, the data shows that:

- The median violation is considerably less for all algorithms when they use the best point rather than the end point.
- The fraction of runs for which the best point offers an advantage is substantial for all algorithms, especially L_Basic_A_3.
- Performance is enhanced for each set of parameter settings.

Since the best point is defined as the point with the lowest violation it is not surprising that these points return a lower median violation on average. The percentage of runs for which using the intermediate point offers an advantage is significant. Approximately two thirds of the runs are improved for L_Basic_A_3 by simply taking the best intermediate point rather than the end point. Using the best intermediate point offers an advantage when a run contains multiple local minima in constraint violation. Stopping the algorithm when the global minimum in constraint violation for a particular run is found would be advantageous as it would reduce the required runtime. Unfortunately, such termination conditions are still unknown.

Table 5.6: Using Best and End Points. 213 models.

CC Type	α	β	Median Violation Best	Median Violation End	Fraction of Runs with Best < End
L_Fdfar	10^{-16}	10^{-16}	2,155.0	2,875.0	0.38
L_SUM	10^{-16}	10^{-16}	1,985.0	5,075.0	0.49
L_Basic_A_3	10^{-16}	10^{-16}	961.0	1,980.0	0.61
L_Basic	10^{-16}	10^{-16}	198,500.0	225,000.0	0.14
L_Fdfar	10^{-3}	10^{-6}	2,196.3	2,935.9	0.41
L_SUM	10^{-3}	10^{-6}	2,154.9	6,474.8	0.52
L_Basic_A_3	10^{-3}	10^{-6}	1,088.1	3,089.1	0.69
L_Basic	10^{-3}	10^{-6}	195,445.0	225,930.0	0.16

5.3.2 Experiment B: Pairing CC with a Local Solver

Experiment B is designed to identify the CC variant whose combination with a local solver gives the best overall performance in finding a feasible solution. The results compare three of the best performing algorithms from Experiment A, namely, L_FDfar, L_SUM, and L_Basic_A_3, and the original CC variant, L_Basic. The local solver is also run without launch point improvement in order to highlight the advantage gained by using CC a priori. The results for Experiment B are presented in Tables 5.7, and 5.8.

The first section of Table 5.7, labeled *No Timeouts or Failures*, reports the results for PS IB, which includes a majority of the models. The numbers confirm the results from Experiment A, that L_Basic_A_3 is the most effective CC variant. Used to improve local solver launch points, L_Basic_A_3 increases the overall feasibility fraction by more than 5% and reduces the average total run time by more than 20%.

L_FDfar and L_SUM also improve the local solver's performance, both increasing the overall feasibility fraction while reducing the total runtime, however, their effects are not as substantial as the L_Basic_A_3 algorithm. The original variant, L_Basic, returned mixed results. It improved the feasibility fraction slightly, but required a slight increase in overall runtime.

The second section of Table 5.7, labeled *Timeouts*, presents the results for PS IIB. These are the 25 models for which Ipopt timed out at least once, but never returned an error. As expected, the statistics in Table 5.2 show that the 25 models causing a timeout are larger on average than the models in the other sets. The CC results for PS IIB are worse than those for PS IB. The average time and median violation are higher in every case; also the

Table 5.7: Using Constraint Consensus with Ipopt

No Timeouts or Failures (PS IB, 175 models)					
CC	**None**	**L_FDfar**	**L_SUM**	**L_Basic_A_3**	**L_Basic**
Avg.Time per run		0.53	0.53	0.55	0.59
Median Violation		9.36×10^2	1.59×10^3	$\mathbf{4.90 \times 10^2}$	4.60×10^4
Feasibility Fraction		0.012	0.031	**0.038**	0.009
Ipopt					
Avg.Time per run	10.71	9.52	8.97	**7.80**	10.21
Avg. Iterations	401.23	323.39	307.57	**246.07**	372.60
Feasibility Fraction	0.726	0.758	0.761	**0.779**	0.738
Total					
Avg.Time per run	10.71	10.06	9.50	**8.35**	10.80
Timeouts (PS IIB, 25 Models)					
CC					
Avg.Time per run		2.36	2.34	2.34	2.41
Median Violation		2.68×10^5	6.89×10^5	4.66×10^5	$\mathbf{1.29 \times 10^5}$
Feasibility Fraction		0.00	0.00	0.00	0.00
Ipopt					
Avg.Time per run	56.84	52.24	52.64	**46.62**	55.03
Timeout Fraction	0.824	0.820	0.800	**0.732**	0.856
Feasibility Fraction	0.152	0.156	0.164	**0.248**	0.128
Total					
Avg.Time per run	56.84	54.60	54.99	**48.97**	57.45
Failures (PS IIIB, 13 Models)					
Ipopt					
Feasibility Fraction	0.03	0.08	0.09	**0.26**	0.08
Infeasible Fraction	0.05	0.06	0.20	**0.22**	0.20
Fail Fraction	0.92	0.86	0.71	**0.52**	0.72

feasibility fraction for the CC variants is lower in every case. Again, this is attributed to the much larger model size. In terms of relative performance, L_Basic_A_3 performed the best on PS IIB. The L_Basic_A_3/Ipopt pair achieved the lowest timeout fraction and the highest feasibility fraction while using the least amount of time.

Some of the runs timed out because the time limits on CC and Ipopt are not long enough for the larger models. Nonetheless, some of the runs on these models did not time out. This is because the performance of the local solver depends greatly on its starting point. The advantage of using CC to improve solver launch points is clear in the *Timeouts* section of Table 5.7: using L_Basic_A_3 decreases the solver timeout fraction by almost 10% as compared to launching the local solver by itself.

The results for PS IIIB are contained in the third section of Table 5.7, labeled *Failures*. This set includes only a small number of models, yet the performance results are consistent with the previous two sets. In this case L_Basic_A_3 outperformed the other algorithm variants again, achieving both the highest feasibility fraction and the lowest fail fraction. These results in particular emphasize the launch point sensitivity of local solvers.

Overall, the results in Table 5.7 show that the newly proposed CC variants, L_SUM, and L_Basic_A_3, can improve a local solver's performance on a set of large-scale models. Additionally, L_SUM, and L_Basic_A_3 outperform previously proposed CC methods including L_FDfar which is reported as one of the most effective variants [42]. The best performing CC variant is L_Basic_A_3. The pair of L_Basic_A_3 and Ipopt successfully found a feasible solution for 1465 of the 2130 runs, given the strict 60s time limit. Alone, Ipopt requires more time on average and only found a feasible solution for 1332 of the runs.

Applying CC to Nonlinear Constraints Only

Section 5.1.4 suggests that it may be advantageous for CC to concentrate on the subset of nonlinear constraints since they pose the greatest difficulty when finding points that are nearly feasible. The results presented in Table 5.8 are from an experiment designed to investigate this proposition. The first section of Table 5.8 reports the numbers from applying CC to all of the constraints. The second section of the table reports the numbers from only applying the CC algorithms to the nonlinear constraints of the models. The experiment included the 117 models from PS IB, IIB, and IIIB that have at least one linear and nonlinear constraint.

Table 5.8 demonstrates an increase in feasibility fraction for the local solver when the CC variants are only applied to nonlinear constraints. L_SUM is the greatest beneficiary achieving the highest feasibility fraction of 0.73. The feasibility fraction of L_Basic_A_3 is

Table 5.8: Using Constraint Consensus with Ipopt: Concentrating on Nonlinear Constraints. Results from the 117 models with both linear and nonlinear constraints.

Constraint Consensus Applied to all Constraints					
CC	**None**	**L_FDfar**	**L_SUM**	**L_Basic_A_3**	**L_Basic**
Avg.Time per run		0.83	0.83	0.85	0.87
Median Violation		3.83×10^3	5.64×10^3	1.99×10^3	5.15×10^5
Feasibility Fraction		0.01	0.03	0.04	0.00
Ipopt					
Avg.Time per run	16.75	15.37	13.97	11.78	16.33
Avg. Iterations	436.68	318.75	323.69	257.67	416.09
Feasibility Fraction	0.65	0.68	**0.70**	**0.70**	0.65
Total					
Avg.Time per run	16.75	16.21	14.80	**12.62**	17.20
Constraint Consensus Applied Only to Nonlinear Constraints					
CC					
Avg.Time per run		0.73	0.72	0.75	0.81
Median Violation		3.39×10^3	2.60×10^3	2.28×10^3	2.73×10^5
Feasibility Fraction		0.00	0.00	0.01	0.00
Ipopt					
Avg.Time per run	16.75	14.04	12.75	12.42	15.06
Avg. Iterations	436.68	295.33	274.51	295.98	407.76
Feasibility Fraction	0.65	0.71	**0.73**	0.71	0.67
Total					
Avg.Time per run	16.75	14.76	13.47	**13.18**	15.87

increased with only a slight increase in the total time. On the contrary, the overall time taken by L_FDfar, L_SUM, and L_Basic is reduced while their respective feasibility rates are increased. In general, applying CC to only the nonlinear constraints improves overall performance.

5.4 Discussion

5.4.1 Characteristics of Successful Launch Points

This section analyzes the launch points found by the CC algorithms for PS IB. Table 5.9 divides the launch points into four categories by constraint type, namely, linear inequality, nonlinear inequality, linear equality, and nonlinear equality. Equality constraints predominate in the test models; 87% of the constraints are of type equality. Furthermore, nonlinear constraints outnumber linear constraints by a rate of approximately 3:2 and more than 75% of the models contain at least one nonlinear equality constraint.

The characteristics of a start point that are best for a certain type of solver vary. Ipopt is a barrier solver. It computes a series of approximate solutions for a sequence of barrier problems. The barrier problems are constructed by converting inequality constraints into barrier terms that are added to the objective function. The barrier terms are formed using a logarithmic function [98]. Logarithmic barrier terms compute to infinity if a candidate solution is directly on the limiting value of the constraint. Therefore, it is beneficial for barrier solver start points to over-satisfy inequality constraints so that numerical complications are avoided. It is also beneficial if the start point is close to satisfying equality constraints as this will reduce the computational effort required by the solver to find a feasible solution.

Some constraint types appear in models more frequently than others. It is advantageous to use methods that handle these constraint types effectively. For instance, the aforementioned constraint statistics suggest that correctly handling nonlinear equality constraints may prove to be a successful strategy.

The data in Table 5.9 (Section 5.2.5 describes the metrics) provide insight into why L_Basic_A_3 and L_SUM are relatively successful when compared to the other methods. L_Basic_A_3 consistently finds points with the lowest average number of violated inequality constraints in both the linear and nonlinear cases. Furthermore, although none of the methods reduce the average number of violated equality constraints, L_Basic_A_3 and L_SUM find the most points that are relatively close to satisfying nonlinear equality constraints.

Table 5.9: Launch Point Data Divided by Constraint Classification.

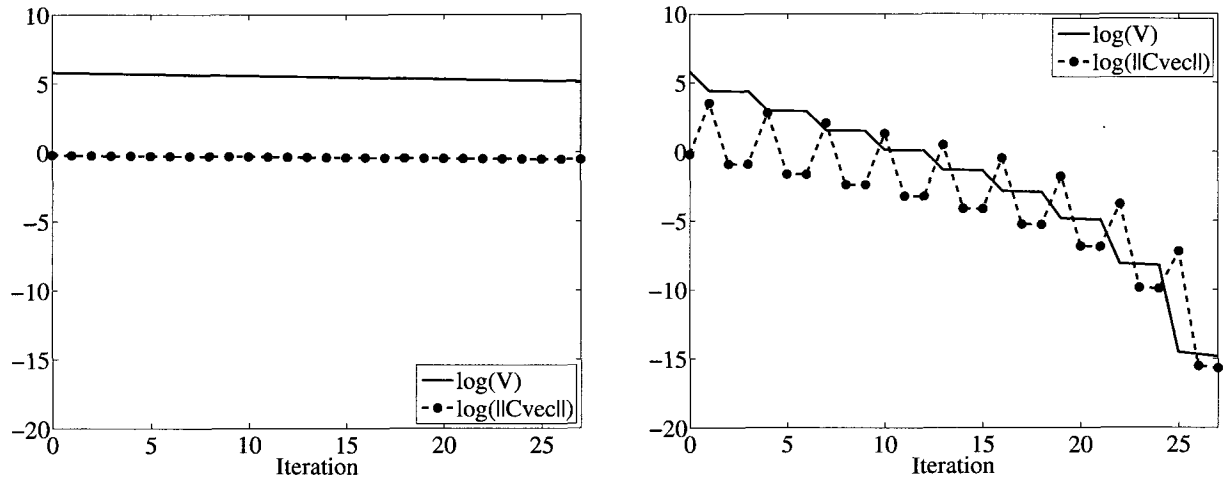
Nonlinear	**Inequality** (Avg. 173.30)				**Equality** (Avg. 812.11)	
CC	**Viol**	**OverSat**	**JustSat**	**Wtol**	**Viol**	**Close**
None	86.50	29.80	57.00	0.00	812.02	5.94
L_FDfar	30.40	69.67	73.01	0.22	810.55	51.73
L_SUM	33.61	**79.11**	51.60	**8.98**	**806.27**	59.33
L_Basic_A_3	**24.95**	71.31	**76.29**	0.74	810.80	**94.18**
L_Basic	52.75	58.35	62.19	0.00	811.66	9.02

Linear	**Inequality** (Avg. 31.71)				**Equality** (Avg. 582.31)	
CC	**Viol**	**OverSat**	**JustSat**	**Wtol**	**Viol**	**Close**
None	20.85	10.25	0.59	0.01	582.31	10.88
L_FDfar	14.24	16.52	0.93	0.03	**582.02**	**31.26**
L_SUM	20.35	9.10	1.20	**1.05**	582.04	8.91
L_Basic_A_3	**13.10**	**16.76**	**1.83**	0.01	582.27	15.22
L_Basic	15.25	15.42	1.03	0.01	582.28	11.04

L_Basic_A_3 performs especially well, finding almost 1.6× as many points that are close to satisfying nonlinear equality constraints as the second most successful variant, L_SUM.

L_Basic_A_3 and L_SUM also tend to find points that over-satisfy more nonlinear inequality constraints than the other methods. This is important because Ipopt is a barrier solver. This partially explains the strength of pairing L_Basic_A_3 or L_SUM with the solver Ipopt. This also highlights that different variations of CC may work better than others depending on the solver they are paired with. This issue is addressed in detail in [42].

Many of the constraints in the models are linear equality constraints. L_FDfar is the best at finding points close to satisfying linear equality constraints, however, none of the methods are particularly successful at reducing the number of violated linear equality constraints. This indifference to linear equality constraints partially explains why applying CC only to nonlinear constraints proved to be successful in Section 5.3.2.



(a) L_Basic. (b) L_Basic_A_3.

Figure 5.4: Successful augmentation. Constraint violation $\mathcal{V}$ and consensus vector length $||Cvec||$ are illustrated at each iteration.

5.4.2 Augmentation and Consensus Vector Length

In general, augmentation improves the performance of CC variants that use Basic consensus. Fig. 5.4a and Fig. 5.4b depict the results for the L_Basic and L_Basic_A_3 algorithms, respectively, operating on the COCONUT model *airport* that is composed of 84 variables and 42 quadratic constraints. The independent axes represent the iteration number and the dependant axes use a logarithmic scale.

Fig. 5.4a shows that the length of the consensus vector ($log(||Cvec||)$) for L_Basic is always short, therefore L_Basic is unable to lower the constraint violation substantially. Conversely, in Fig. 5.4b L_Basic_A_3 takes a relatively long step at every third iteration, and the result is a swift decrease in the constraint violation. In only 27 iterations L_Basic_A_3 decreases $\mathcal{V}$ from 3.35×10^{2} to 4.16×10^{-17} whereas after 27 iterations L_Basic has still made almost no progress.

This example clearly demonstrates the advantage of augmentation; however, this is a particular case and the results from different starting points are not always so sensational. For instance, augmentation offers mixed results for the methods using FDfar and SUM consensus as reported in Section 5.3.1. This may be due to the relative effectiveness of the consensus types. FDfar and SUM consensus approximate the needed consensus vector length better than Basic consensus, which commonly underestimates the update vector length; this is likely the reason augmentation offers a larger advantage to Basic consensus.

5.4.3 Time Complexity of the Quadratic Feasibility Vector Calculation

The linear feasibility vector calculation is exact for linear constraints and the quadratic feasibility vector calculation is exact for quadratic constraints, but both are simply estimates for higher order constraints. Given a single constraint composed of n variables, the asymptotic time complexities of the linear and quadratic feasibility vector calculations are $O(n)$ and $O(n^2)$, respectively. This explains why the algorithms using quadratic feasibility vectors complete fewer iterations on average given the maximum run times. The reason these particular algorithms are less successful at reducing constraint violation than their linear counterparts stems from the fact that the consensus vectors are also approximations, so the additional accuracy for the feasibility vectors does not make a significant difference. The accuracy provided by the quadratic term is not effective given the extra computational cost it requires. This is confirmed by the results in Section 5.3.1. The variants that only use quadratic feasibility vectors for quadratic constraints outperform the variants that use quadratic feasibility vectors for all nonlinear constraints. Also, the methods that use linear feasibility vectors almost always outperform the methods that use quadratic feasibility vectors.

5.4.4 SUM Consensus

SUM consensus is relatively successful compared to the FDfar and Basic consensus methods even though they all require approximately the same amount of computation time. Like Basic consensus, the SUM method builds the consensus vector using components from all of the violated constraints and therefore is more likely to avoid cycling than the FDfar method. The main difference between the Basic and SUM consensus methods is that the Basic method averages together the feasibility vectors whereas SUM consensus adds the feasibility vectors. Averaging can cause the consensus vector to be short as is demonstrated in Section 5.4.2. SUM consensus vectors are in the same direction as Basic consensus vectors, however, they are generally longer. The aggressive steps taken by SUM consensus tend to lead to better performance.

5.5 Summary

This chapter introduces various advances to the CC algorithm. The results from Experiment A present a general ranking of how the algorithm variants compare with one another.

Experiment B provides evidence that the new variants not only outperform existing CC algorithms, but that they also improve local solver success. Overall, augmentation is successful when paired with Basic consensus. L_Basic_A_3 is the best at lowering constraint violation and at boosting local solver performance. SUM consensus is also extremely competitive in these regards. Quadratic feasibility vectors prove to be too expensive given the small increase in accuracy they introduce. Applying CC to the subset of nonlinear constraints is advantageous. This strategy boosts the overall feasibility fraction of the CC and local solver pairs.

On average, successful launch points have a higher percentage of both satisfied and over satisfied inequality constraints and are close to satisfying many nonlinear equality constraints. This positive correlation should not be confused with causation as there are counter examples that show local solvers sometimes perform better when launched from points with higher violations.

Chapter 6

Avoiding Redundant Solver Launches Using Constraint Consensus, Concentration, and Clustering

Naive multistart is a basic algorithm that launches a local solver from many random locations throughout the search space in the hope that at least one of the launches is successful. Local solvers are very expensive in terms of computation time if launched from a poor location, so avoiding ineffective launch points is advantageous. This chapter presents a strategy that uses CC to quickly explore the search space before determining launch points for a local solver. The method uses CC to quickly concentrate points around promising regions in the search space. This chapter also introduces a novel technique for analyzing the inter-point distances of the concentrated points in order to determine data clusters, and a method for incorporating the objective function into a multistart search using CC. Illustrated examples are provided to highlight the advantages and weaknesses of the proposed algorithms.

6.1 Concentration and Clustering via Constraint Consensus

6.1.1 Visualizing Basins of Attraction with Constraint Consensus

The goal of the method introduced in this chapter is to quickly explore the variable space in order to identify promising regions from which expensive local solvers should be launched. The proposed method is a multistart algorithm that finds promising local solver launch

points and reduces the number of redundant local solver launches. This section highlights the visualization of basins of attraction in NLP using CC.

Consider the Branin0 model illustrated in Fig. 6.1:

$$
\begin{aligned}
&find && \mathbf{x} = \{x_1, x_2\} && (6.1)\\
&s.t. && g_1(\mathbf{x}) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + \left(10 - \frac{10}{8\pi}\right)\cos(x_1) + 9 \leq 0 && (6.2)\\
& && g_2(\mathbf{x}) = x_2 + \frac{x_1 - 12}{1.2} \leq 0 && (6.3)\\
& && -5 \leq x_1 \leq 10 && (6.4)\\
& && 0 \leq x_2 \leq 15 && (6.5)
\end{aligned}
$$

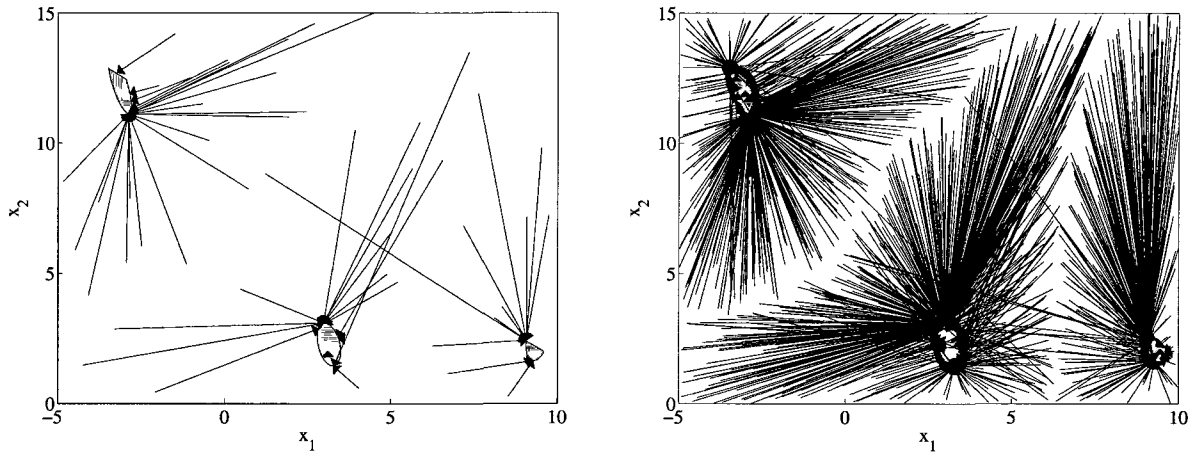
The Branin0 model consists of a linear constraint and a nonlinear constraint. The nonlinear constraint $g_1(\mathbf{x})$ is the Branin function [45] commonly used as a benchmark for optimization methods. Three feasible regions result from the constraints, shown as gray areas in Fig. 6.1.

An advantage of the CC algorithm is that its iterations are relatively inexpensive, therefore it can be used in a multistart framework to quickly search the variable space for promising regions from which more expensive local solvers are launched. The CC algorithm effectively finds points that are closer to feasibility. If multiple CC runs are launched from various points throughout the search space, the CC output points will concentrate into subsets near points of attraction (for more on points of attraction see Section 2.1). A visual approximation of the basins of attraction is rendered when lines are drawn connecting the trails of points from the start of a CC run to its output point. Refer to Fig. 6.1 for an illustrated example with the Branin0 model.

In both Fig. 6.1a and 6.1b the start points for the CC runs were selected by Latin hypercube sampling. Fig. 6.1a illustrates CC start-output pairs for 50 random start points. The CC start point is at the tail of each arrow and the output point is at the head. The basins of attraction start to become obvious in Fig. 6.1b after CC is run 1500 times. The white space indicates the boundaries between the basins of attraction.

A couple of the CC paths in Fig. 6.1 transverse multiple basin approximations. This is because basins can actually be composed of disjoint regions due to the interaction between nonlinear functions and solution algorithms such as CC. For instance, a solution algorithm may be misled by a nonlinear function and take a step that is too large. In these vicinities CC's path may not lead to the closest (in terms of Euclidean distance) feasible region.

The algorithms presented in this chapter do not identify basins of attraction; instead the following sections outline a method for identifying the regions of the search space in



(a) 50 points. (b) 1500 points.

Figure 6.1: Basin visualization via CC for the Branin0 model.

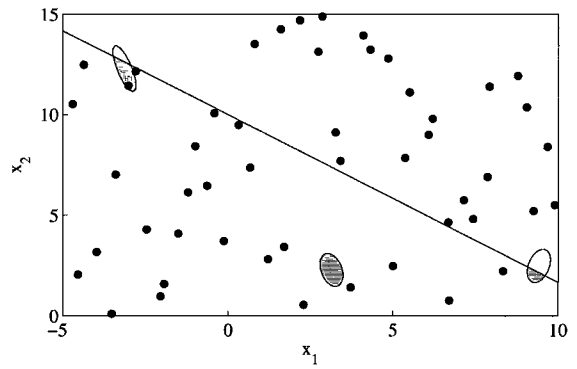
which multiple CC runs concentrate.

6.1.2 Concentration and Inter-point Distance Distributions

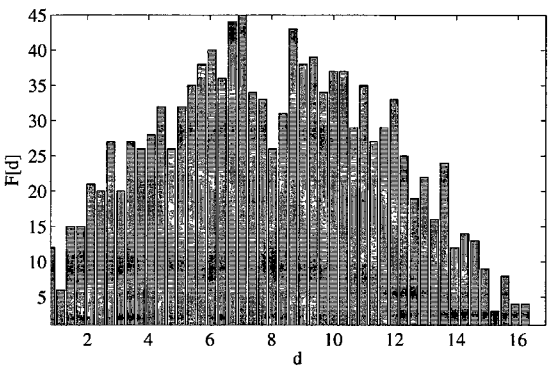
The last section briefly introduces the idea of concentration. CC is run from multiple start points and the output points concentrate near the various points of attraction where constraint violations are minimized. The Branin0 model has three such regions as shown in Fig. 6.1. Ultimately, this chapter explores the idea that at least some of the CC runs will concentrate near feasible regions. This section examines the relationship between sets of points concentrated by CC and their inter-point distance distribution.

High-dimensional data can be reduced to a distribution of distances between data points [10]. If clusters exist in the data, the inter-point distance distribution will be multi-modal with peaks that correspond to the distances within and between sets of concentrated points. A set of randomly sampled points in equally weighted dimensions has an inter-point distance distribution that is most frequently unimodal and resembles a Rayleigh distribution. Conversely, the frequency distribution of inter-point distances for the concentrated set will be multi-modal if there is more than one region of attraction because points concentrated near the same region of attraction will be closer to each other than to points concentrated in other regions.

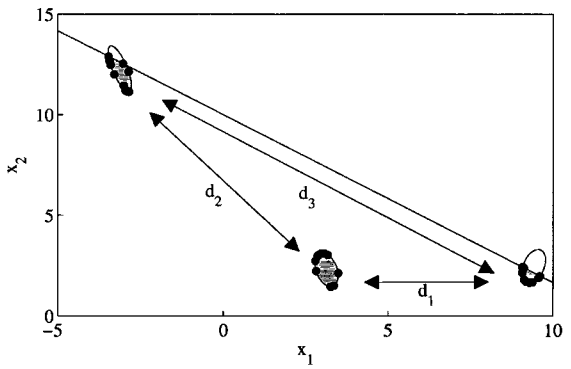
Fig. 6.2 illustrates inter-point distance distributions for the Branin0 model. Fig. 6.2a shows the fifty initial points selected by Latin hypercube sampling and Fig. 6.2c depicts the concentrated output points generated by launching CC from each of the initial points.



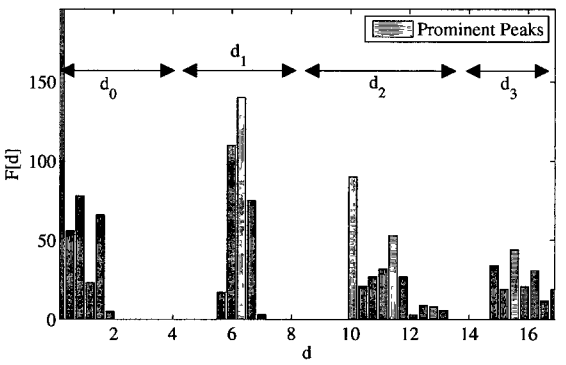
(a) Latin hypercube sample.



(b) Initial distribution.



(c) Concentrated points.



(d) Distribution of concentrated points.

Figure 6.2: Inter-point Distance Distributions.

The inter-point distance distributions corresponding to the initial and CC output points are illustrated in Fig. 6.2b and 6.2d, respectively.

Fig. 6.2c and 6.2d illustrate the relationship between clusters and the multi-modal inter-point distance distribution. Consider the four distinct groups of inter-point distances clearly visible in Fig. 6.2d: d_0, d_1, d_2, and d_3. The set of inter-point distances labeled d_0 are the distances between points near the same region of attraction whereas the inter-point distances labeled d_1, d_2, and d_3 delineate the distances between the points in different concentrated sets as depicted in Fig. 6.2c.

The inter-point frequency distribution, $F[d]$, of the concentrated points has peaks because some subsets of points are close to each other and, in general, further away from other sets of points. Distances, d, that correspond to minima in $F[d]$ are effective critical distances for the single linkage clustering algorithm (see Section 3.3.1) because they are the distances that separate clusters. For instance, $d \simeq 4$ separates the three clusters depicted in Fig. 6.2c. Alternatively, $d \simeq 9$ separates the clusters centered around $x_2 = 3$ from the cluster centered near $x_2 = 12$.

The inter-point distance distribution is created by sorting all the inter-point distances into their respective distance ranges, or *bins*. Collectively, the bins include all the inter-point distances from the shortest, d_{min}, to the longest, d_{max}. The range of an individual bin, d_{width}, is determined by dividing the difference between the longest and shortest distance by the number of sample points, p. The nominal distance associated with a bin is its center. The number of inter-point distances that fall into each bin determines the frequency distribution, $F[d_i]$, where

$$d_i = d_{min} + (i + \frac{1}{2})d_{width}, i \in \{0, p-1\} \tag{6.6}$$

are the bin centers. A *prominent peak* in the frequency distribution is defined as a bin with a population that is higher than the ω preceding bins and the subsequent ω bins.

When paired with CC and concentration, hierarchical clustering can estimate how many regions of attraction exist in a given model. The next section outlines a method to calculate the critical distance for the single-linkage clustering routine using the inter-point frequency distribution of points concentrated by CC.

6.1.3 Extracting a Critical Distance for Clustering from Concentrated Points

The critical distance is crucial for the single linkage clustering algorithm and has a major impact on the number of clusters that are identified. Critical distances that approximate the separation of regions of attraction in the search space can be extracted from the multi-modal inter-point distance distribution of the concentrated CC output points. As mentioned in the last section, minima in the inter-point distance distribution are effective critical distances because they are the distances that separate sets of concentrated CC output points.

The differences in frequencies between successive distances tend to be low in the regions near the minima. For instance, consider the region $2 < d < 6$ illustrated in Fig. 6.2d; the frequencies are 0 for most bins in this range. This section proposes a method of finding the minima in the inter-point frequency distribution whereby maxima in the inter-point distance distribution are identified and the minima are approximated using the midpoints between the maxima.

Fig. 6.3 helps illustrate the practicalities of the algorithm using a fictitious example with 7 points and therefore 21 inter-point distances. In this example $p = 7$, $d_{min} = 2$, $d_{max} = 16$, and $d_{width} = \frac{16-2}{7} = 2$. As mentioned previously, the nominal distance associated with a bin is calculated by taking the midpoint. In this case for instance, the bin $[2, 4)$ has nominal

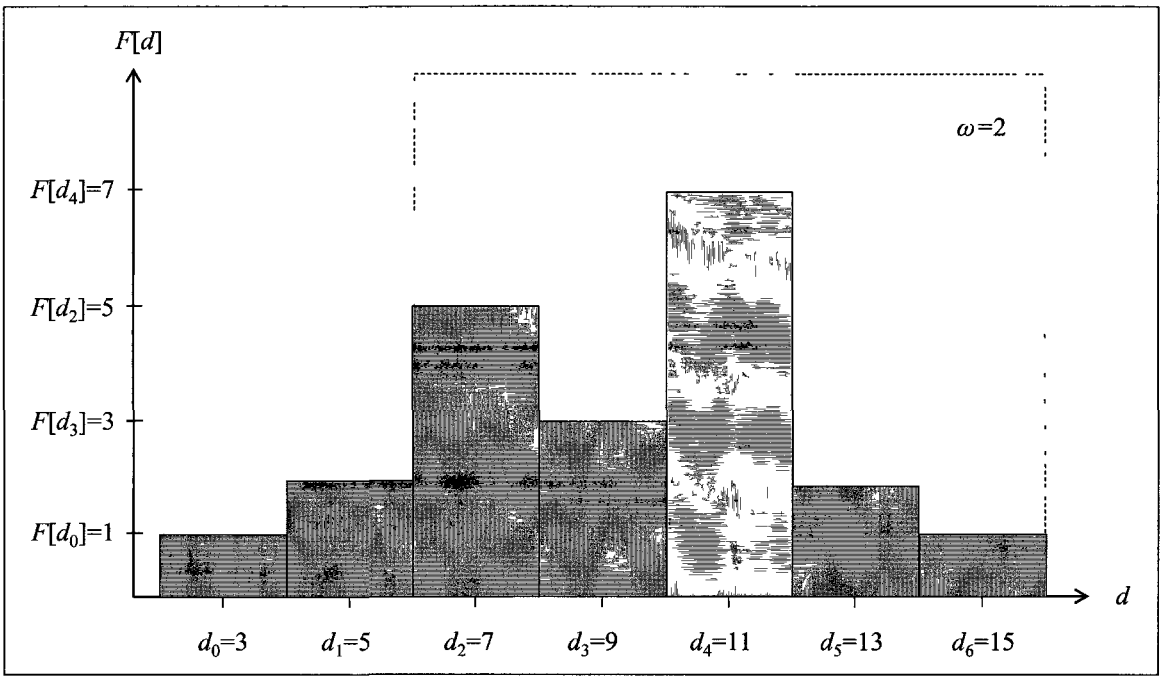


Figure 6.3: Calculating prominent peaks. Only one prominent peak exists at $d = 11$ when $\omega = 2$. The gray region highlights the bins involved in the prominent peak calculation for the bin centered at $d = 11$.

distance $d_0 = 3$.

The illustration in Fig. 6.3 highlights the prominent peak centered at $d_4 = 11$ when $\omega = 2$. In this case $F[d_4] = 7$ is greater than the frequencies of the two preceding bins, $F[d_2] = 5$ and $F[d_3] = 3$, and the two subsequent bins, $F[d_5] = 2$ and $F[d_6] = 1$. There are two prominent peaks if $\omega = 1$: the bin centered at d_4 whose frequency is greater than the bins centered at d_3 and d_5, and the bin centered at d_2 whose frequency is greater than its preceding and subsequent bins.

The critical distance d_c is an approximation of the inter-point frequency minima in $F[d]$. It is calculated as the midpoint between prominent peaks in the distribution. As an example consider the inter-point distribution for the Branin0 model in Fig. 6.2d. The first prominent peak distance is approximately 6 ($q_1 \simeq 6$) and the first minimum occurs somewhere in the range $2 < d < 5$. The initial critical distance is found by averaging d_{min} and the distance associated with the first peak as described in Step 2 of Alg. 1: $d_c \simeq \frac{6+0}{2} = 3$. The initial critical distance is accepted if the resulting number of clusters falls below the threshold, τ_c. On the other hand, if the resulting number of clusters exceeds the threshold, then a distance between the first peak and the second peak is calculated and tried as the clustering distance. The second prominent peak of the Branin0 example

Algorithm 1: Critical Distance Extraction

Step 0: Set τ_c and initialize ω.

Step 1: Determine the set of prominent peak distances $\{q_j | \ j \in \{1, ..., Q\}\}$ using ω, where q_j is the bin center of the j^{th} prominent peak. Order the q_j from smallest to largest. If the number of peaks is zero (i.e., $Q = 0$) and $\omega \neq 0$, then reduce ω by one and repeat step, else exit unsuccessfully.

Step 2: Cluster the concentrated points using the critical distance $d_c = (q_1 + d_{min})/2$. If the number of clusters is less than or equal to τ_c then exit successfully, else go to step 3.

Step 3: Try clustering the concentrated points with $d_c = (q_j + q_{j+1})/2$ for each element $j \in \{1, ..., Q-1\}$. If the number of clusters found is ever less than or equal to τ_c then exit successfully, else reduce ω by one and go to step 1.

---

is located at $q_2 \simeq 10$, so the second critical distance candidate is $\frac{6+10}{2} = 8$.

If all the prominent peaks associated with a specific ω fail, then ω is reduced by 1 and the process is repeated. For instance, in Fig. 6.3 the initial critical distance extracted is $\frac{11+2}{2} = 6.5$ if $\omega = 2$. If this candidate distance fails then ω is decremented to 1. When $\omega = 1$ the first critical distance is 4.5 and the second is 9.0.

The aforementioned steps are repeated until the number of clusters falls below the threshold or the pool of candidate critical distances is exhausted. The threshold τ_c is the maximum number of clusters deemed acceptable by the user. Its relation to the number of solver launches is explained in more detail in the next chapter. The pool of candidate critical distances becomes exhausted when ω is reduced to 0. The set of concentrated points is considered ill-conditioned in this case. An example of an ill-conditioned set is one where all the points have the same coordinates.

The full method is described in Alg. 1.

6.1.4 Ranking Points

At various places in the algorithms proposed in Chapters 6 and 7 the most *promising* point is chosen. The following hierarchical system is used to rank points:

1. All feasible points are considered more promising than infeasible points,
2. If two points are feasible, the one with the lower objective function value is considered more promising, and
3. If two points are infeasible the one with the smallest maximum constraint violation is considered more promising.

Note that throughout this thesis, a solution is considered feasible if and only if $\mathcal{V}(\mathbf{x}_k) \leq 10^{-6}$, as explained in Section 5.2.5.

Selecting Launch Points from Clusters

The CC output points are clustered so that a diverse set of high-quality launch points can be determined. The most promising CC output point from each cluster is chosen so that the launch point is of relatively high-quality. A launch point is chosen from each cluster so that there is diversity, in terms of location, amongst the set of launch points. The results reported in Section 7.3.2 highlight that both these factors are important for multistart algorithms to be successful.

6.1.5 Adding Aspiration Constraints to Constraint Consensus

MacLeod [58] describes a technique for adding aspiration values to CC in order to seek optimal solutions rather than feasible solutions. The optimization problem is reconfigured into the following

$$\begin{aligned} \text{find} \quad & \mathbf{x} && (6.7)\\ \text{s.t.} \quad & f(\mathbf{x}) \leq a_k && (6.8)\\ & g_i(\mathbf{x})\{\leq, =\}0, \quad \forall i \in I \doteq \{1, 2, ..., m\} && (6.9)\\ & \ell_j \leq x_j \leq u_j, \quad \forall j \in J \doteq \{1, 2, ..., n\}. && (6.10) \end{aligned}$$

Eqn. 6.8 is the aspiration constraint and a_k is an aspiration scalar that decreases as the method progresses. The aspiration scalar is initially set to infinity until a feasible solution is found, then it is reset to the objective function value of the first feasible solution minus

some small threshold. Every time the aspiration constraint is over-satisfied at a feasible point the aspiration value is decreased. Therefore, CC seeks an optimal solution while attempting to satisfy all of the original constraints.

This section proposes a novel multistart algorithm in which aspiration values are inherited from previous CC runs. The result is a method that explores the search space for optimal solutions rather than feasible solutions. The method is outlined below.

Due to the way the optimization model is setup (Eqn. 6.7-6.10), a feasible solution not only satisfies the usual constraints, it must also satisfy the aspiration constraint. Initially, the aspiration scalar is set to infinity so that the aspiration constraint is satisfied at all points. In this circumstance, CC runs effectively attempt to find a solution that satisfies only the original constraints (Eqn. 6.9-6.10) since the aspiration constraint is satisfied everywhere. The aspiration scalar will continue to have no effect on any of the CC runs until a feasible solution is found and the aspiration scalar is reduced from infinity.

The method proposed here is to run CC on the reconfigured model (Eqn. 6.7-6.10) from multiple start points. Initially, the aspiration scalar is set to infinity. When a CC run finds a feasible solution the aspiration scalar is reduced to the objective function value at the feasible point, then another CC run is started. The key is that the aspiration scalar is passed on to subsequent CC runs. For example, if the first CC run finds a feasible point with objective function value 5, then the second CC run will start with the aspiration scalar set to 5 rather than infinity.

To be feasible, a solution has to satisfy the original constraints and the aspiration constraint with the latest aspiration scalar. Therefore, once the aspiration scalar is reduced from infinity the subsequent CC runs must consider the aspiration constraint when searching for feasible regions. The aspiration scalar only gets reduced when feasible points with objective function values lower than the aspiration scalar are found. The end result is a multistart CC method that seeks optimal, rather than feasible, solutions.

The algorithm is presented in Alg. 2 and is examined further in Section 6.3.

6.2 Experimental Setup

6.2.1 Hardware and software

The setup is the same as described in Section 5.2.1.

Algorithm 2: Multistart CC with Aspiration

Step 0: Set $a_0 = \infty$, $k = 0$, $i = 0$, and μ to the desired number of CC starts. Set the CC parameters: α, β, maximum iterations, and maximum runtime.

Step 1: Select μ start points using Latin hypercube sampling.

Step 2: Start CC from the i^{th} start point using the aspiration constraint with the k^{th} aspiration value. If a feasible solution, $\mathbf{x_f}$, is found increment k and adjust the right hand side of the aspiration constraint: $a_k \leftarrow f(\mathbf{x_f})$.

Step 3: Set $i \leftarrow i + 1$. If $i < \mu$ go to step 2, else stop.

6.2.2 Problem Descriptions

Experiments are run on the Branin0 model introduced in Section 6.1.1 and the Rastrigin0 and Schwefel0 models described next.

The Rastrigin0 model is

$$
\begin{aligned}
find \quad & \mathbf{x} = \{x_1, x_2\} && (6.11)\\
s.t. \quad & g_1(\mathbf{x}) = x_1^2 + x_2^2 + 20 - 20(\cos 2\pi x_1 + \cos 2\pi x_2) \leq 0 && (6.12)\\
& g_2(\mathbf{x}) = x_2 - x_1^3 \leq 0 && (6.13)\\
& -5 \leq x_1 \leq 5 && (6.14)\\
& -5 \leq x_2 \leq 5. && (6.15)
\end{aligned}
$$

Rastrigin0 has significantly more feasible regions than the Branin0 model. Furthermore, the feasible regions are denser. This makes the task of finding all the feasible regions more difficult because they are not as widely separated and there are fewer sample points per feasible region. The Rastrigin0 model also has many infeasible regions of attraction due to the sinusoidal functions.

The Schwefel0 model is

$$
\begin{aligned}
find \quad & \mathbf{x} = \{x_1, x_2\} && (6.16)\\
s.t. \quad & g_1(\mathbf{x}) = x_1 \sin(\sqrt{|x_1|}) + x_2 \sin(\sqrt{|x_2|}) + 125 \leq 0 && (6.17)\\
& g_2(\mathbf{x}) = x_2 - \frac{1}{16}x_1^2 + 150 \leq 0 && (6.18)\\
& -150 \leq x_1 \leq 150 && (6.19)\\
& -150 \leq x_2 \leq 150. && (6.20)
\end{aligned}
$$

This model has 6 feasible regions that vary greatly in relative size, as well as two regions of attraction near the boundaries of the search space at $(120, -150)$ and $(-150, 120)$. The distances between the feasible regions also vary. Like Rastrigin0, this model has infeasible regions of attraction due to the sinusoidal functions.

Three minimization objective functions are used to test CC with aspiration constraints: (1) $x_1 + x_2$, (2) x_2, and (3) $x_1^2 + x_2^2$. The number at the end of a model's name represents the objective function included in the model. A 0 represents that no objective function is included in the model. For instance, Branin3 refers to a model with the Branin constraints and the objective function $x_1^2 + x_2^2$.

6.2.3 Algorithm Variants and Parameters

CC Parameters

L_Basic_A_3 is the CC algorithm used for the experiments reported in this chapter. The CC output points are the best points found by the CC run (Section 5.1.4). The parameter values for CC are: $\alpha = 10^{-6}$ and $\beta = 10^{-3}$. No time limit is imposed upon the CC runs. Section 6.3.1 contains results for concentration via CC without aspiration constraints. Section 6.3.2 reports the results for concentration with CC using aspiration constraints.

Clustering Parameters

The number of CC launches used throughout this chapter is $p = 50$. For many models it may be advantageous to use a larger value for p. p is kept constant for all the experiments so they can be compared fairly. τ_c is set to 25 for all the experiments. These parameter settings are discussed in more detail in the next chapter. The selection of ω is described next.

Table 6.1: Numbers of Clusters Returned Over 100 Tests.

ω	Branin0		Rastrigin0		Schwefel0	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
1	8.25	2.75	10.84	9.75	18.23	3.55
2	5.25	1.98	**11.27**	10.4	16.59	4.75
3	4.15	1.52	10.73	9.28	12.64	3.90
4	3.79	1.40	8.55	8.88	10.82	2.06
5	3.54	1.11	4.91	5.25	**10.47**	2.07
6	**3.02**	0.20	4.05	3.42	10.52	2.59

Tuning ω

Table 6.1 lists the mean and standard deviation for clusters calculated when the clustering algorithm is run 100 times for each model. Each row in Table 6.1 corresponds to a different ω value. Ideally, the clustering algorithm would find 3, 20, and 6 feasible regions, respectively, for the Branin0, Rastrigin0, and Schwefel0 models. Boldface is used to indicate the mean result closest to the actual number of feasible regions for that particular model, i.e., the number of clusters found on average that is closest to the actual number of feasible regions. Note that lower ω values usually result in shorter critical distances.

The clustering algorithm finds more than 3 clusters on average for the Branin0 model because of the shape of the feasible regions. CC runs tend to concentrate at the top and bottom of the feasible regions. When the critical distance is short, two or more clusters redundantly identify the same feasible region. As ω is increased, the mean result for the clusters calculated converges to 3.

The Rastrigin0 model has many small feasible regions close together. If the critical distance is short enough, the CC output points get separated properly into distinct clusters. If the critical distance is too long most of the CC output points get clustered into a large cluster because the feasible regions are uniformly spaced, so the CC output points also tend to be uniformly spaced. The relatively large standard deviations for the Rastrigin0 model in Table 6.1 reflect this reality.

The Schwefel0 model is also very difficult. It contains both large and small feasible regions that are separated by various distances. The large feasible regions tend to attract multiple clusters. On the other hand, the small feasible regions that are close together tend to attract too few clusters. In this case there is a tradeoff between longer critical distances that reduce the number of feasible regions attracting multiple clusters and shorter critical distances that increase the number of clusters, and hence the number of local solver

launches.

The results listed in Table 6.1 show that there is no value for ω that will work best for all models. For the remaining experiments in this thesis $\omega = 3$ is chosen. It is difficult to tune this parameter for larger models because there is no way to determine in advance how many feasible regions they contain.

Selecting Launch Points

The most promising point in each cluster is chosen as the launch point. This process is described in Section 6.1.4.

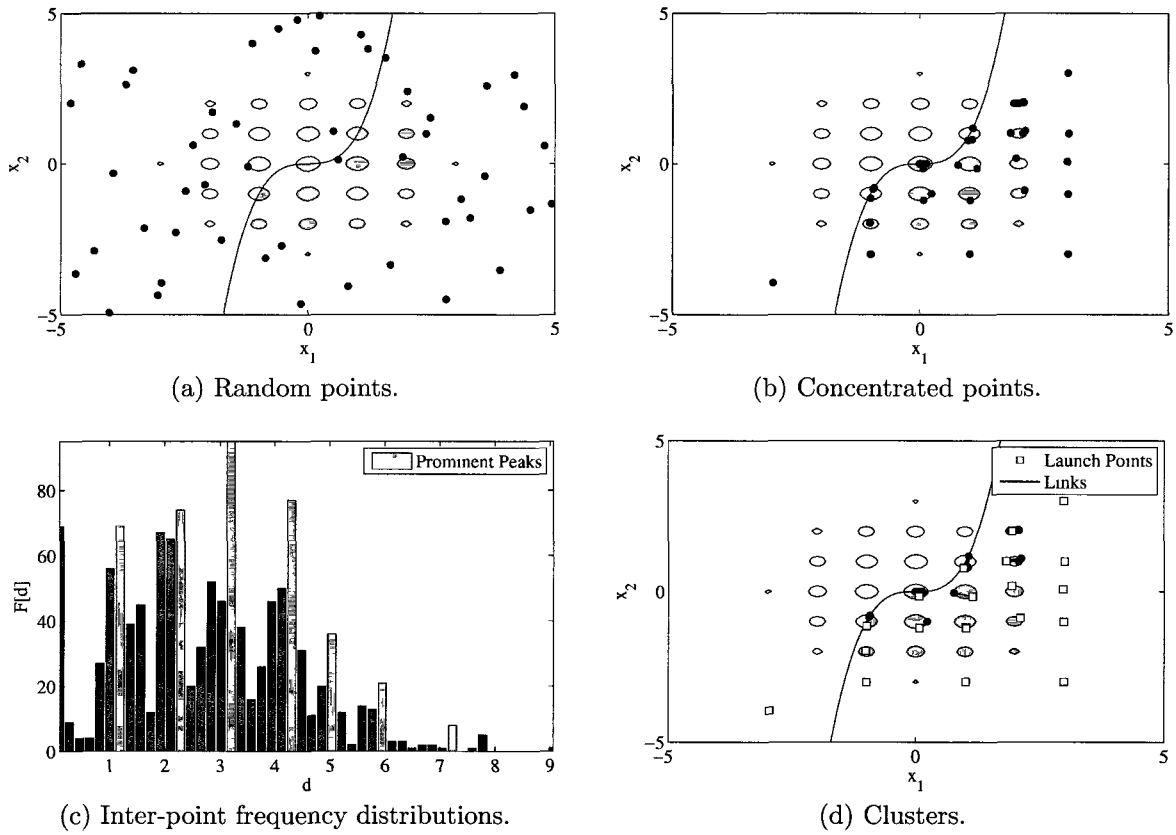
6.2.4 Performance Metrics

The experiments in this section are run on several variations of three particular models. The results are presented as illustrations when possible and a table is used to list the following performance metrics

- **Clusters calculated** is the number of clusters the single linkage algorithm found in the concentrated data sets.
- **Feasible regions attracting exactly one cluster** is the number of feasible regions that are identified correctly by a single cluster.
- **Feasible regions attracting multiple clusters** is the number of feasible regions that are identified by two or more clusters.
- **Feasible regions attracting no clusters** is the number of feasible regions that are not identified by any clusters.
- **Infeasible regions attracting at least one cluster** represents the number of clusters that do not identify a feasible region.

6.3 Experimental Results

The experiments reported in this section illustrate the strengths and weaknesses of the proposed clustering algorithm and applying aspiration constraints to CC.



(a) Random points.

(b) Concentrated points.

(c) Inter-point frequency distributions.

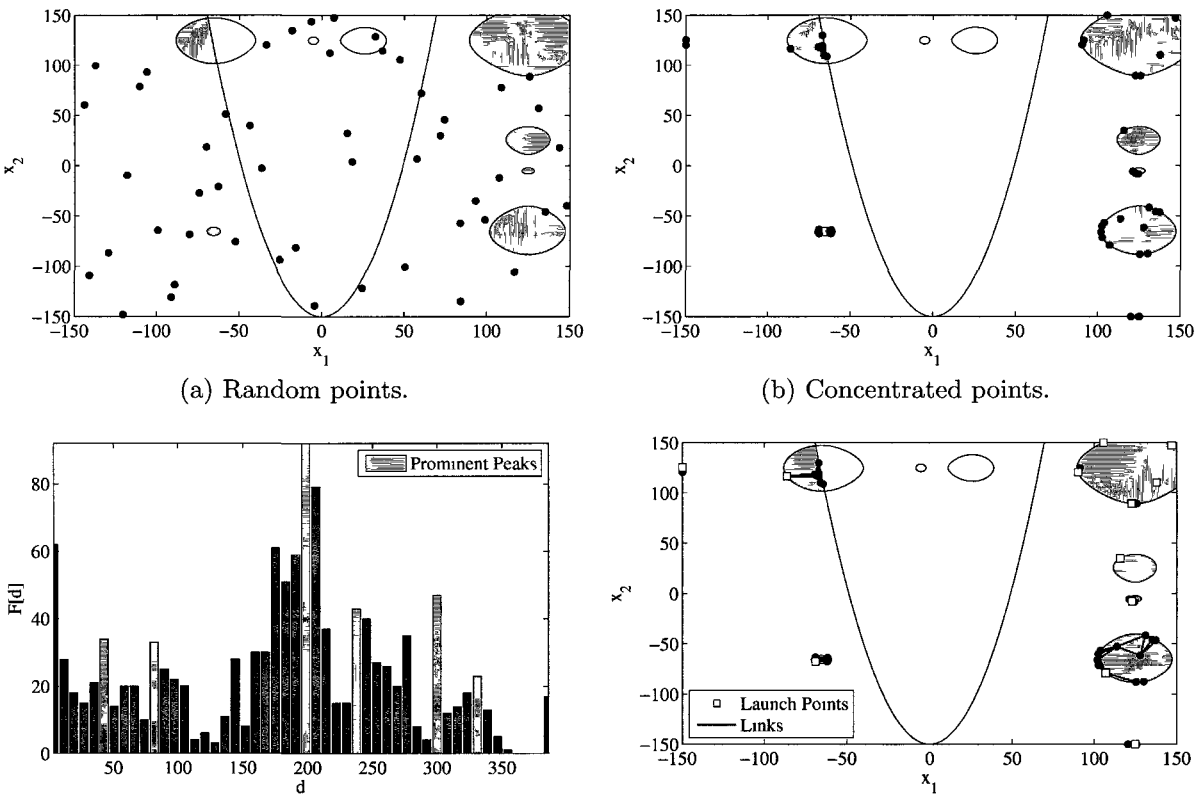
(d) Clusters.

Figure 6.4: Rastrigin0 Case Study: (a) The initial locations of the 50 sample points. (b) The new locations of the 50 points after concentration with CC. (c) Frequency distribution of the inter-point distances after concentration. (d) Links connect points that are closer than the critical distance of 0.5945. 16 of the 20 feasible regions are identified correctly.

6.3.1 Seeking Feasibility

Fig. 6.4 shows the results from concentrating and clustering 50 CC output points for the Rastrigin0 model. The initial sample and the concentrated data points are shown in Fig. 6.4a and Fig. 6.4b, respectively. Three of the concentrated sets are near infeasible points of attraction (approximately: (-3,-4),(3,-3), and (3,3)). Fig 6.4c illustrates the inter-point distance frequency distribution and the prominent peaks. The separation between the prominent peaks is smaller in this distribution compared to the distribution for the Branin0 model because the clusters are closer together. In this case there are 7 prominent peaks when $\omega = 3$ and the first critical distance, $d_c = 0.5945$, is accepted. The launch points are depicted in Fig 6.4d.

The results from concentrating and clustering 50 CC runs on the Schwefel0 model



(a) Random points.

(b) Concentrated points.

(c) Inter-point frequency distributions.

(d) Clusters.

Figure 6.5: Schwefel0 Case Study: (a) The initial locations of the 50 sample points. (b) The new locations of the 50 points after concentration with CC. (c) Frequency distribution of the inter-point distances after concentration. (d) Links connect points that are closer than the critical distance of 21.42. All 6 of the feasible regions are identified.

are shown in Fig 6.5. The initial sample and the concentrated data points are shown in Fig. 6.5a and Fig. 6.5b, respectively. The large feasible region centered near $(125, 125)$ attracts points to five distinct locations, resulting in five different clusters. The inter-point frequency distribution illustrated in Fig 6.5c shows the prominent peak with the greatest frequency is around $d = 200$. This is the approximate length of the sides of a square with vertices located at the centers of the three largest feasible regions and the small one near $(-60, -60)$. The large frequency around $d = 200$ is due to the many clustered points located near the imagined square's four vertices. With $\omega = 3$, the first critical distance of 21.42 is accepted. The launch points are highlighted in Fig 6.5d. There are two infeasible regions of attraction and possibly four redundant launches in the large feasible region centered near $(125, 125)$.

Table 6.2 summarizes the illustrated results for the Branin0, Rastrigin0, and Schwefel0

Table 6.2: Clustering Results

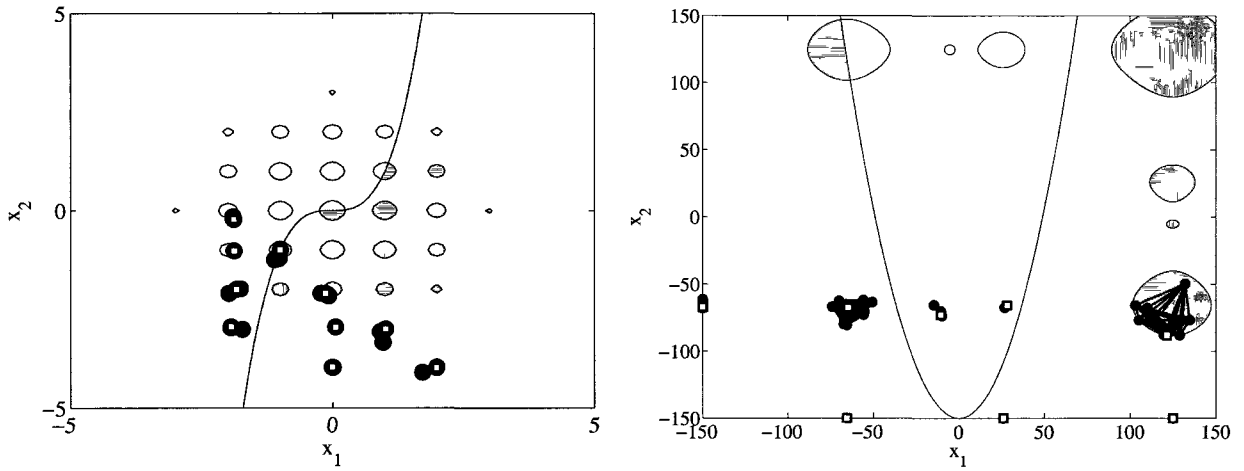
Model:	Branin0	Rastrigin0	Schwefel0
d_{min}:	0	0	0
d_{max}:	1.71	9.15	389
d_{width}:	0.341	0.183	7.79
Peaks found:	4	7	6
Critical distances tried:	1	1	1
Critical distance:	3.16	0.595	21.4
Clusters calculated:	3	19	12
Feasible regions attracting exactly one cluster:	3	16	5
Feasible regions attracting multiple clusters:	0	0	1
Feasible regions attracting no clusters:	0	4	0
Infeasible regions attracting at least one cluster:	0	3	2
Actual number of feasible regions:	3	20	6

test cases. The clustering method correctly identified all three of the feasible regions in the Branin0 model, each by a single cluster. It identified 19 clusters for the Rastrigin0 model and 16 of the 20 feasible regions are correctly identified by a single cluster. The method also identified 3 extraneous clusters, each near an infeasible region of attraction. All 6 of the feasible regions in the Schwefel0 model are identified, along with 2 infeasible regions of attraction. 5 of the 6 feasible regions attracted a single cluster, but the largest feasible region attracted 5 different clusters.

6.3.2 Seeking Optimality

This section explores the use of aspiration constraints with CC in a multistart framework. By considering the objective function the CC output points should not only concentrate near feasible regions, but in the feasible regions with the lowest objective function values. The experimental results highlight the advantages and weaknesses of this strategy. Section 6.4 provides insight to some of the strange results exhibited when CC is coupled with aspiration constraints.

A couple of examples of clustering with aspiration are illustrated in Fig 6.6. The Rastrigin1 model is shown in Fig 6.6a. There are 12 regions that attract CC output points when the aspiration constraint is used: 6 of these regions are feasible and 6 are not. In this particular case, the global optimum (the small region centered near $(-1, -3)$) is not actually identified by the method. However, many promising locations near the global



(a) Minimize $x_1 + x_2$. (b) Minimize x_2.

Figure 6.6: Examples of CC with aspiration: (a) Rastrigin1 (b) Schwefel2.

optimum are located.

The Schwefel2 model shown in Fig 6.6b has two feasible regions close to the lowest objective function value: a smaller one centered around $(-60, -60)$ and a larger one centered around $(125, -60)$. Only the larger feasible region actually contains the lowest objective function value. At least 6 infeasible regions of attraction exist when CC is run with the aspiration constraint. In this case all of the launch points are in locations where both the objective function and the constraint violations are minimized, however, only two launch points are in feasible regions.

Table 6.3 lists the results from running the clustering algorithm 100 times on each of the models under consideration. For each experiment $\omega = 3$. The numbers in columns *feasible* and *infeasible* under the *Attractive Regions* heading indicate how many feasible and infeasible regions of attraction exist in each model, respectively. The numbers under the heading *Clusters Calculated* are the mean and standard deviations of 100 test runs using multistart CC with aspiration constraints (Alg. 2).

Table 6.3 shows that the number of clusters calculated is commonly greater than the number of attractive feasible regions. There are several reasons for this that are discussed in Section 6.4. The obvious reason is the negative influence of the attractive infeasible regions. In all cases the number of solver launches would be substantially less than 25, the number of launches the standard multistart method would be forced to use if $\tau_c = 25$.

Table 6.3: Statistics from 100 tests using aspiration constraints

	Attractive Regions		Clusters Calculated	
	Feasible	Infeasible	Mean	Std. Dev.
Branin1	3	0	4.23	1.22
Branin2	2	0	3.85	1.45
Branin3	1	5	4.22	1.49
Rastrigin1	6	6	16.8	4.58
Rastrigin2	3	7	20.3	5.09
Rastrigin3	1	0	4.81	2.64
Schwefel1	1	6	12.0	3.41
Schwefel2	2	6	8.50	1.97
Schwefel3	3	6	11.7	4.47

6.4 Discussion

Various circumstances lead to curious results for the concentration and clustering methods. Some explanations are provided in this section.

6.4.1 Infeasible Regions of Attraction

The CC algorithm finds points that minimize the constraint violations. Models sometimes have regions with low violations that are not feasible. These regions become apparent when CC is used for concentration. The example shown in Fig. 6.6a, for instance, illustrates many infeasible regions of attraction. This is a common problem for local NLP solvers.

Using an aspiration constraint with CC during the concentration phase can exacerbate this problem. The reason is that the direction for minimizing the objective function may be in the opposite direction from the feasible region. This situation may cause CC to calculate short consensus vectors and make very little progress. An example where introducing the aspiration constraint causes points to concentrate near infeasible regions of attraction is depicted in Fig. 6.7.

Fig. 6.7 illustrates the results of using multistart CC with aspiration on the Branin3 model. In this model, the original constraint violations are minimal in the three shaded areas. The aspiration constraint is minimal at $(0, 0)$. The objective of CC with aspiration is to minimize the whole set of constraint violations, including the aspiration constraint. In this case, some CC runs never find a feasible region because the aspiration constraint is effectively pulling them in the opposite direction. For instance, there is a group of CC

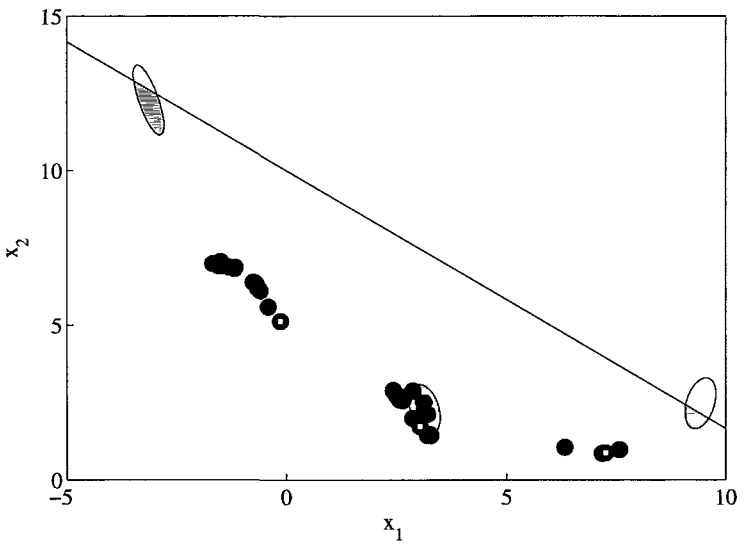


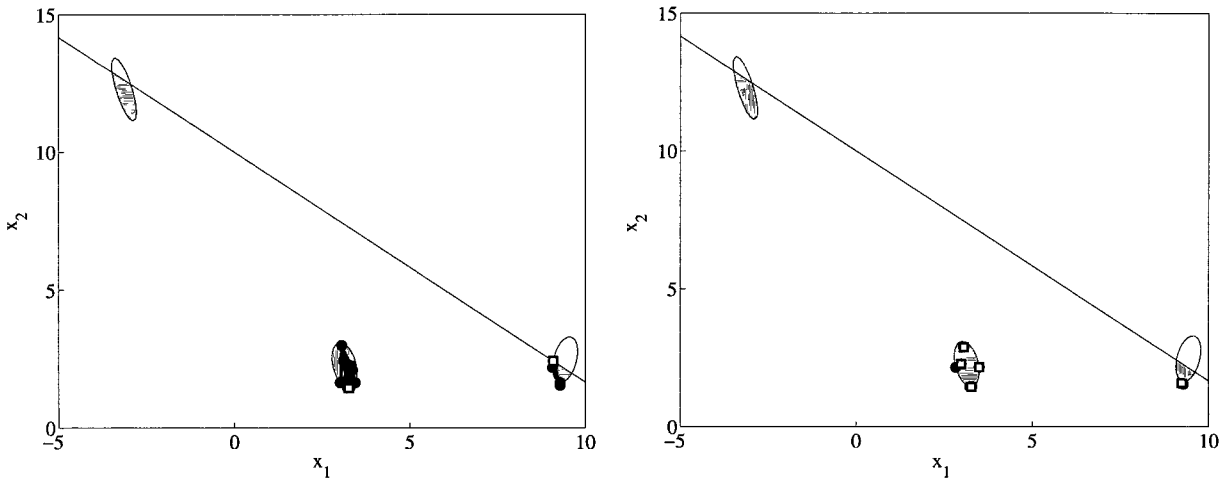
Figure 6.7: Examples of infeasible regions of attraction for the Branin3 model caused by the introduction of an aspiration constraint for the objective: minimize $x_1^2 + x_2^2$.

output points near $(-1, 6)$. These points are on a line that intersects $(0, 0)$ and the feasible region centered near $(-3, 12)$. Without the aspiration constraint CC would usually corral these points into the feasible region centered near $(-3, 12)$. However, Fig. 6.7 shows that when the aspiration constraint is applied, these points get stuck in an infeasible region of attraction.

6.4.2 Relative Versus Absolute Cluster Separation

An issue with the clustering method, regardless of aspiration constraints, is the division of the CC output points into clusters such that there is exactly one cluster per discontiguous feasible region. The selection of the critical distance is made automatically using the inter-point frequency distribution. An interesting result occurs when points are concentrated into several distinct groups that are close, yet equally spaced. The critical distance may be selected such that these groups get separated into individual clusters, even though they are relatively close together.

Fig. 6.8a illustrates when two clusters are found correctly for the Branin2 model, one for each feasible region with relatively low objective value functions. On the other hand, Fig. 6.8b depicts a situation where the points happen to concentrate into close, yet distinct groups in one of the feasible regions. The result is that the automatically selected critical distance causes 5 clusters to be returned. Although this behaviour is not intended, it is not necessarily negative; there may be multiple optima within a feasible region. Also, feasible



(a) 2 Launch points identified.

(b) 5 Launch points identified.

Figure 6.8: The effect of close, yet distinct groups of points in the Branin2 model. In (a) there are no distinct groups of points in the feasible region centered near $(3, 3)$. As a result the clustering routine finds only one cluster to represent the points near this feasible region. In (b), the points concentrated near the feasible region centered at $(3, 3)$ are in four distinct groups. As a result, Alg. 1 calculates a shorter critical distance and the points near the same feasible region are assigned to different clusters.

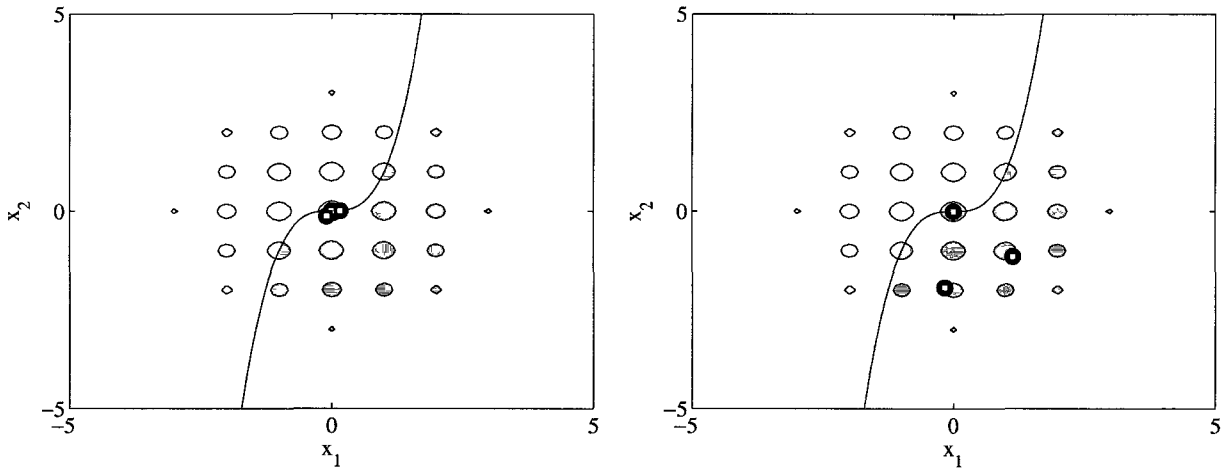
regions that are further away are still identified.

6.4.3 Aspiration Artifacts

When CC is run with an aspiration constraint the initial aspiration value is ∞. This setting effectively means that the aspiration constraint is satisfied, so the CC algorithm will not incorporate it into the consensus calculation. This only changes once a feasible solution vector is found and the aspiration value is reduced. The consequence is that initial CC runs may not take advantage of the aspiration constraint at all. The resulting points are called artifacts because they are calculated by CC without using the aspiration constraint.

Fig. 6.9 shows an example of aspiration artifacts. The objective function in this case is minimize $x_1^2 + x_2^2$, whose unconstrained minimum is optimal at the origin. In Fig. 6.9a the initial CC run finds a feasible solution and the aspiration constraint is activated with a good initial aspiration value. In Fig. 6.9b CC runs a couple times before finding an aspiration value that causes the remaining runs to concentrate near the origin.

Aspiration constraints have less effect on early CC runs because the aspiration value is set to a relatively easy target. Furthermore, if no feasible regions are found the aspiration



(a) Most desirable feasible region found. (b) Two other feasible regions also found.

Figure 6.9: An example of aspiration artifacts in the Rastrigin3 model.

constraint will not change the algorithm's behaviour at all. The data in Table 5.7 show that the best CC variant L_Basic_A_3 finds feasible solutions for only a small fraction of the set of large nonlinear models. For this set of models, using aspiration as described in Section 6.1.5 would have little if any effect.

6.4.4 Meaningful Distance Metrics

An issue with the clustering algorithm is how it interprets distances. The clustering algorithm does not scale the variable space, therefore, dimensions in which there are relatively large distances between points will influence the clusters more than dimensions in which there are only relatively small distances between points. Possible solutions include scaling each dimension so that the bounds on each variable are the same, or scaling each dimension so that the maximum distance between the furthest points is the same in every dimension. However, these solutions introduce other scaling related issues. The problem stems from the fact that the clustering routine does not know what the units of the variables are, and therefore, it is difficult to make meaningful spatial comparisons of the points. That being said, the results reported in Chapter 7 indicate that for many models clustering CC output points, without scaling the variable space, is an effective method for choosing solver launch points.

6.5 Summary

This chapter establishes an automatic method for extracting a critical distance from the inter-point distance distribution of a set of points concentrated by CC. The experimental results show that the method can quickly identify where feasible regions are located throughout the search space and suggest launch points for a local solver. A technique to incorporate the objective function into the multistart space exploration is also introduced. Updating the aspiration value throughout the process can further concentrate CC end points to promising regions, however, as the results show, there can also be negative consequences. The proposed aspiration algorithm is only advantageous if CC can find a feasible solution. This is not the case for the models examined in the next chapter, therefore aspiration constraints are not used.

Chapter 7

Applying Clustering to Global Optimization

The previous chapter introduced a clustering routine that is used to identify the various promising regions in the search space and ultimately determine launch points for an NLP solver. Although this method initially requires additional computation, time savings are eventually realized as redundant solver launches are eliminated. This chapter presents numerical evidence that over a large set of models the proposed method is successful in reducing overall time while maintaining solution integrity. The proposed method is shown to be a promising alternative to naive multistart using the open source solver Ipopt, and the commercial multistart solvers Knitro 6.0 and MSNLP from Ziena Optimization Inc. and Optimal Methods Inc., respectively.

7.1 Efficient Algorithms for Selecting Local Solver Launch Points

Note that the terms *point* and *vector* are used interchangeably throughout the text. Furthermore, a *candidate solution* or a *solution vector* is simply a point located in the search space. Initially, every point in the search space is a candidate solution and the goal of the global optimization methods is to find the overall best point: the feasible point with the lowest objective function value.

7.1.1 MS+C: Reducing Redundant Solver Launches

Multistart plus concentration (MS+C) combines the ideas developed in the previous chapters into an algorithm for constrained global optimization. The algorithm identifies launch points that are redundant before they are passed to an expensive local solver. Also, since a launch point is selected from each cluster, spatial variety is preserved in the set of launch points. The result is that the local solver is launched fewer times while solution integrity is maintained.

MS+C selects initial points using a Latin hypercube sample of the search space. L_Basic_A_3, applied only to nonlinear constraints (see Section 5.1.4), is run from each of the initial points. This CC variant is chosen because its performance is superior to the other methods, as shown in Section 5.3. The CC output points (see Section 5.1.4) concentrate in promising regions of the search space where the greatest constraint violation is minimized. A critical distance is selected using Alg. 1 from Section 6.1.3 and the concentrated points are clustered with a single linkage routine (see Section 3.3.1). The points in each cluster are ranked using the system presented in Section 6.1.4 and the point in each cluster deemed the most promising is singled out as a launch point.

The launch points are ordered from most to least promising and then the local solver is initiated from each launch point, in order. The reason order is taken into account when the local solver is launched is to make sure that the more promising launch points are used before less promising points. This technique is advantageous when either the run time or the number or local solver launches is restricted. The solution vector returned by each local solver launch is saved and the most promising solution vector, as determined by the system in Section 6.1.4, is returned as the final solution. The full algorithm is presented in Alg. 3.

MS+C has two search modes: (0) launch limited and (1) time limited. The objective of MS+C when running in mode 0 is to explore the search space and effectively determine a reduced set of promising points from which to launch the local solver. The focus is on using τ_c or fewer local solver launches, though the total time consumed might vary considerably. When running in mode 0, MS+C terminates after finishing step 5 of Alg. 3. Alternatively, in mode 1 MS+C runs until a maximum time limit is reached or the total number of solver launches reaches τ. In this case, if MS+C finishes step 5 and there is still time remaining and the total number of solver launches is less than τ, then MS+C will start over at step 1 with a new set of random start points. When running in mode 1, MS+C continues to explore the search space until the total time available is exhausted. Note that in mode 0 $\tau = \tau_c$.

Algorithm 3: MS+C

Step 0: *Initialization.* Set the number of sample points p, clustering parameters τ_c and ω, the CC parameters including α and β, and the search mode. If $mode = 0$ set the maximum run time per local solver launch. Else if $mode = 1$ set the maximum total run time (in mode 1 the maximum solver run time per launch equals the maximum total run time) and the total number of solver launches available, τ. Create an empty list of candidate solutions.

Step 1: *Initial Sample.* Randomly select an initial sample of p points by Latin hypercube sampling, which ensures that the initial points are distributed throughout the search space.

Step 2: *Concentration.* Start CC from each of the sample points and output the best point resulting from each start.

Step 3: *Clustering.* Calculate the Euclidean inter-point distances between all of the CC output points and assemble the frequency distribution. Identify the prominent peaks in the distribution and use these to calculate a critical distance using Alg. 1. If Alg. 1 fails go to step 1, otherwise form clusters among the CC output points using single linkage clustering until every cluster is separated by a distance greater than the critical distance found in the previous step.

Step 4: *Choose local solver launch points.* Choose the most promising launch point in each cluster using the system presented in Section 6.1.4. Order these launch points from most to least promising. If the most promising launch point is feasible add it to the list of candidate solutions.

Step 5: *Launch Solver.* If $mode = 0$, launch the local solver from each of the selected points, in order, and append the solutions to the list of candidate solutions. Rank the candidate solutions and return the most promising as the final solution. Else if $mode = 1$, monitor run time while running the local solver from each of the ordered launch points. Append local solver solutions to the list of candidate solutions. If the number of local solver launches reaches τ or the run time is exhausted, return the most promising solution from the list of candidate solutions. If time and solver launches remain after running the local solver from each of the selected launch points, go to step 1.

7.1.2 Other Multistart Techniques

MS+CC: Multistart with CC Only

Multistart plus Constraint Consensus (MS+CC) is a simple extension of multistart whereby CC is used to find the launch points for a local solver. From each start point the following two-part process is executed: CC is run from the start point, and a local solver is launched from the CC output point. MS+CC is listed in Alg. 4.

Experiment D tests a second version of this algorithm denoted MS+CC*. MS+CC* initially selects a set of random start points and runs CC from each. Then MS+CC* ranks all the CC output points in terms of promise (Section 6.1.4). The local solver is launched from each point, in order, until stopping conditions are satisfied. In both algorithms, the solution vector from each local solver run is added to a candidate solution list and in the end the most promising solution is returned.

The difference between the two variations of MS+CC is how the local solver launch points are ordered. The advantage of ranking the launch points is that the local solver may return better candidate solutions earlier in the process. This idea is analyzed in Section 7.3.1.

The MS+CC algorithm is included in the following experiments to provide basic performance results from pairing CC with a local solver in a multistart framework. The purpose is to show that pairing CC with a local solver can improve the results of a multistart algorithm; however, the advanced search undertaken by MS+C improves performance even further.

MS+CCR: Multistart with CC and Reduction

It can be beneficial to quickly explore the search space with CC before deciding on a location from which to launch an NLP solver. Like MS+C, Multistart plus CC and Reduction (MS+CCR) uses numerous CC runs to decide the location of each local solver launch. The major difference is that MS+CCR does not use clustering. MS+CCR consists of three stages: exploration, reduction, and solution. During the exploration stage MS+CCR runs CC from various start points chosen by LHS. In the reduction stage the CC output points are ranked in terms of promise and the most promising point is selected and passed to the solution stage. In the solution stage the local solver is launched and the result is added to a candidate solution list. The three stages are repeated until the time limit is reached, at which time the most promising candidate solution is returned. MS+CCR is listed in Alg. 5.

Algorithm 4: MS+CC

Step 0: *Initialization.* Set the maximum number of solver launches τ and the CC parameters including α and β. Also set $i = 0$.

Step 1: *Initial Sample.* Randomly select an initial sample of τ points by Latin hypercube sampling.

Step 2: *Run CC.* Run CC from the i^{th} sample point and output the best point as a solver launch point.

Step 3: *Launch Solver.* Launch the local solver from the i^{th} CC output point. Save the result and increment i. If $i = \tau$ or the time limit is reached, return the best known solution. Otherwise, go to Step 2.

MS+CCR is an extension to MS+CC that uses multiple CC runs per solver launch to explore the variable space more thoroughly. Rather than using one CC run to determine a launch point, multiple CC runs are executed and only the most promising result is used as a launch point. MS+CCR requires more samples of the search space and spends more time running CC than MS+CC. MS+CCR is included in the following experiments to illustrate the performance of an algorithm that uses a relatively large fraction of the available run time to search for promising local solver launch points with CC.

MS: Multistart

Multistart (MS) is a basic algorithm used in the following experiments. MS simply launches the local solver from a series of randomly chosen start points until stopping conditions are satisfied. MS is included in the following experiments to illustrate the performance of a basic multistart algorithm. MS is listed in Alg. 6. It is expected to be the worst performing algorithm in every experiment.

Algorithm 5: MS+CCR

Step 0: *Initialization.* Set the maximum number of solver launches τ, the number of samples per iteration p, and the CC parameters including α and β. Also set $i = 0$.

Step 1: *Initial Sample.* Randomly select an initial sample of p points by Latin hypercube sampling.

Step 2: *Run CC.* Run CC from each of the sample points and save the highest ranked output point as the i^{th} solver launch point.

Step 3: *Launch Solver.* Launch the local solver from the i^{th} point. Save the result and increment i. If $i = \tau$ or the time limit is reached, return the best known solution. Otherwise, go to Step 1.

Algorithm 6: MS

Step 0: *Initialization.* Set the maximum number of solver launches τ and set $i = 0$.

Step 1: *Initial Sample.* Randomly select an initial sample of τ points by Latin hypercube sampling.

Step 2: *Launch Solver.* Launch the local solver from the i^{th} point. Save the result and increment i. If $i = \tau$ or the time limit is reached, return the best known solution. Otherwise, repeat Step 2.

7.2 Experimental Setup

7.2.1 Description of the Experiments

Four main experiments are reported in this chapter. Experiment C tests the algorithm performances given a restriction on the maximum number of local solver launches. The purpose of this experiment is to show that the MS+C algorithm can successfully select a subset of high-quality launch points. Experiment D contrasts the MS+CC and MS+CC* algorithms in order to highlight the advantages and disadvantages of ranking launch points by relative promise before using a local solver. Experiment E tests the algorithm performances when they are run for a set amount of time. This is the way optimization algorithms are often used in an industrial setting, and hence provides arguably the most relevant set of results. Finally, Experiment F tests the algorithms in terms of finding *best known solutions*: the best solutions reported by a variety of other solvers. The results show that the proposed algorithms find solutions of comparable quality to those found by current state of the art global optimization solvers.

7.2.2 Hardware and Software

These experiments use the same basic setup as described in Section 5.2.1.

The local solver used throughout this chapter is Ipopt, as described in Section 5.2.1. Ipopt parameter settings that are constant throughout all of the experiments are:

- $honor\_original\_bounds = yes$,
- $bound\_relax\_factor = 0$,
- $max\_iter = 9999999$, and
- $constr\_viol\_tol = 10^{-6}$.

The max_cpu_time parameter is set differently in each experiment. For Experiment C, $max\_cpu\_time = 60$ for Problem Set IC and $max\_cpu\_time = 600$ for Problem Set IIC. Experiment D has two parts. In part one, max_cpu_time is set to 600. In part two, max_cpu_time is set to 7200. In Experiment E, max_cpu_time is tested at values of 7200 and 15000, respectively. In Experiment F, $max\_cpu\_time = 600$. All the aforementioned time numbers represent seconds. Unmentioned parameters are left at their default settings and the problem sets mentioned above are described in Section 7.2.3.

Two state-of-the-art commercial solvers are used for comparison purposes:

- Knitro is a local solver that uses interior point and active-set methods for solving continuous, nonlinear optimization problems [100]. A recent release, Knitro 6.0, offers a multistart procedure that essentially restarts the Knitro solver from different initial points and returns the most promising solution found. For the experimental results presented herein, Knitro is run with the following parameters:
  - $ms\_enable = 1$,
  - $feastol = 0.0$,
  - $feastol\_abs = 10^{-6}$,
  - $opttol = 0.0$,
  - $opttol\_abs = 10^{-6}$,
  - $maxit = 9999999$,
  - $honorbnds = 1$,
  - $outmode = 0$, and
  - $outlev = 1$.

  For Experiment C, $maxtime\_cpu = 600$ and $ms\_maxtime\_cpu = 15000$. For Experiment E, both $maxtime\_cpu$ and $ms\_maxtime\_cpu$ are set to 7200 for the first part of the experiment, then 15000 for the seond part. All times are in seconds. $ms\_maxsolves = 25$ for Experiment C and $ms\_maxsolves = 200$ for Experiment E. All other parameters are left at their default settings. The parameter settings are chosen so that the stopping conditions of the Knitro and Ipopt local solvers are as similar as possible. This is important to ensure a fair comparison between the competing multistart algorithms.

- MSNLP is a global solver composed of four main components: a randomized driver, a distance filter, a merit filter, and a local solver. The distance and merit filters are discussed in detail in Section 3.5.2. The local solver used by MSNLP is CONOPT, a generalized reduced gradient solver designed for large-scale optimization [27]. A randomized driver is an algorithm that selects the initial points. The MSNLP results reported in this chapter are for three different randomized drivers, namely, pure random (PR), Optquest (OQ), and smart random normal (SR), all of which are described in [93]. The MSNLP results presented in this chapter are from Section 6 of [93]. The settings for MSNLP are listed in Table 1, Section 4 of [93]. Some of the key parameter settings are:

Table 7.1: Model Statistics for Experiments C, D, and E.

Problem Set:	ICa (22 models)			ICb (126 models)		
	Avg.	Min.	Max.	Avg.	Min.	Max.
Variables	259.0	3	2505	536.5	2	8997
Nonlinear Constraints	120.3	10	800	232.9	10	1000
All Constraints	216.1	10	2495	481.1	10	7000
Nonzeros in Jacobian	1980.8	30	11425	3067.4	39	36185
Problem Set:	IICa (16 models)			IICb (59 models)		
	Avg.	Min.	Max.	Avg.	Min.	Max.
Variables	6055.7	200	11215	6282.5	3	20008
Nonlinear Constraints	4323.5	1024	10000	3897.5	1000	13798
All Constraints	5083.3	1024	11192	5615.5	1000	14000
Nonzeros in Jacobian	33001.3	8000	128004	31567.8	2998	128004

- time limit = 6000s,
- solver call limit = 1000,
- total sample points = 1000, and
- artificial bound = 10^4 (see Section 5.2.2).

7.2.3 Test Models

The models used in Experiments C, D, and E are the ones presented in Section 5.2.2. The problems are divided into two sets based on the number of nonlinear constraints. Problem Set IC (PS IC) contains the 148 models with 10 to 1000 nonlinear constraints and Problem Set IIC (PS IIC) contains the 75 models with more than 1000 nonlinear constraints.

To eliminate bias, the two problem sets are randomly subdivided into tuning and testing sets. The tuning subsets (denoted with an *a*) are used to determine the maximum time parameter for the CC runs in the algorithm and consist of approximately 20% of the models. The testing subsets (denoted with a *b*) are used to test the algorithm. The statistics of the selected models are shown in Table 7.1.

There are two problem sets for Experiment F as shown in Table 7.2. Problem set ID (PS ID) consists of the 366 models from the COCONUT benchmark that have at least one

Table 7.2: Model Statistics for Experiment F

Problem Set:	ID (366 models)			IID (131 models)		
	Avg.	Min.	Max.	Avg.	Min.	Max.
Variables	57.25	2	802	65.36	2	802
Nonlinear Constraints	55.95	1	4950	78.64	1	4950
All Constraints	69.26	1	5049	96.05	1	5049
Nonzeros in Jacobian	526.6	2	19998	419.7	2	19998

nonlinear constraint and a published best known objective function value. Problem set IID (PS IID) is the subset of models from PS ID that are also used to test MSNLP by Ugray et al., as reported in [93]. Therefore, the models in PS IID also have at least one nonlinear constraint and a published best known objective function value.

The local solver Ipopt is unsuccessful at solving the model *launch* to a feasible solution from all the start points used in the experiments and the default point provided by the modeler, which is frequently a high-quality start point. Since the purpose of the experiments presented in this chapter is to compare multistart methods and not local solver performance, the model *launch* is removed from PS IID.

The algorithms using the Knitro and Ipopt solvers ignored the initial points provided with some of the models. This made the problem set more difficult for the optimization algorithms because the provided points are often of high-quality and are likely to lead local solvers to a feasible solution. This gave MSNLP an advantage in Experiment F since the first step of the MSNLP algorithm launches a local solver from the provided modeler point [93].

Appendix A contains a table of all the models used at any point during this thesis along with the corresponding best known objective function values, if they are known, and the specific problem sets that contain the models. These values are a collection of the best values found by the MS, MS+CC, MS+CCR, and MS+C algorithms described in this thesis, the MSNLP and Knitro solvers, and any other best known solution provided with the COCONUT benchmark [87].

7.2.4 Algorithms and Parameter Settings

The basic parameters examined below are: (i) p, the number of CC start points per iteration, (ii) τ, the maximum number of local solver launches, (iii) ω, the number of preced-

ing and following bins in the inter-point frequency distribution that is used to identify a prominent peak, (iv) the maximum time allowed for a single CC run, (v) the maximum time allowed for a single run of the local solver, and (vi) the maximum total time.

A common issue for global optimization algorithms is the question of how many sample points and/or local solver launches to use. For instance, no irrefutable rule exists for determining the number of particles used for particle swarm optimization or the population size for a genetic algorithm. For Experiment C, p is 25 for PS IC and 50 for PS IIC, and $\tau = 25$. For Experiment D and E, $\tau = 200$, but p varies depending on the algorithm. When possible, the same random start points are used for each algorithm. Knitro uses its own routine to select start points so it uses a different set of start points in each case.

CC is used by various algorithms throughout this chapter. Unless stated otherwise, L_Basic_A_3 is used as the CC variant with $\alpha = 10^{-6}$ and $\beta = 10^{-3}$. The most promising point found during a CC run is used as the output point and CC is only applied to the nonlinear constraints as described in Section 5.1.4 and evaluated in Section 5.3.2.

Experiment C: Number of Solver Launches Restricted

In this experiment each algorithm can make up to a maximum of 25 solver launches. MS+C identifies redundant solver launches and may decide to use fewer than the available 25 solver launches after searching the variable space. Knitro, MS and MS+CC do not attempt to eliminate redundant solver launches and always launch the local solver 25 times. The maximum solver run times are 60s and 600s for PS IC and PS IIC, respectively. A maximum total run time is not enforced; instead the maximum number of solver launches is restricted. The four competing algorithms are:

- **Knitro**: Knitro in multistart mode with the parameter settings listed in Section 7.2.2.
- **MS**: Basic Multistart using the solver Ipopt with the parameter settings listed in Section 7.2.2.
- **MS+CC**: Multistart with Constraint Consensus using the local solver Ipopt with the parameter settings listed in Section 7.2.2.
- **MS+C**: Multistart with clustering using Constraint Consensus, operating in mode 0. $p = 25$ for PS IC and $p = 50$ for PS IIC. ω is set to 3 as is explained in Section 6.2.3. $\tau_c = 25$. A single most promising point is identified for each cluster using the ranking scheme listed in Section 6.1.4. The local solver is Ipopt with the parameter settings listed in Section 7.2.2.

The maximum number of solver launches, τ, is set to 25 for all the algorithms. Different values for this parameter may change the relative performance of the algorithms compared in the experiments. For the purpose of comparison, consider the default number of launch points for Knitro's multistart procedure, $min\{200, 10n\}$, where n is the number of variables [100]. Given the average number of variables in the problem sets (listed in Table 7.1), the default setting for Knitro would run the solver 200 times for many models in PS IC and all the problems in PS IIC. Setting $\tau = 25$ is relatively low compared to Knitro's default setting, however, the results presented in Section 7.3.1 indicate that $\tau = 25$ is a reasonable choice for many of the models.

The local solver is allotted 60s per launch for the smaller problems in PS ICb, and 600s for the larger problems in PS IICb. The 600s time limit is likely overly restrictive for the larger models, but is necessary for practical reasons in order to compare a number of alternative algorithms over a large number of models. Since the variations in results are mostly due to the solver launch points chosen, this time restriction should not materially affect the conclusions.

Experiment D: Ranking Candidate Launch Points

Experiment D is composed of two parts. The first part compares the effect of ranking candidate launch points on the time to find the *initial incumbent solutions*: the first feasible solution that a multistart algorithm finds. The competing algorithms are given up to 600s to find an incumbent solution. The results from this experiment show ranking the launch points can reduce the amount of time required to get an initial incumbent solution.

The second part of Experiment D tests if ranking the candidate launch points translates into overall success in finding a global optimum. Both the overall maximum time limit and the run time per solver launch are set to 7200s. The maximum number of solver launches is set to 200. The algorithms are compared based on the best overall solution that they find.

The problem set used for both experiments presented in this section is PS IICb. This problem set is selected because it contains relatively large and difficult models.

The two algorithms used in this experiment are:

- **MS+CC**: Multistart plus Constraint Consensus using the solver Ipopt with the parameter settings described in Section 7.2.2.
- **MS+CC***: Multistart plus Constraint Consensus with Launch Point Ranking, using the solver Ipopt with the parameter settings described in Section 7.2.2.

A similar method for ranking candidate launch points could be applied to the MS algorithm presented in Section 7.1.2. Since the results achieved by the MS+CC method are superior to those of the MS method, only the combination of MS+CC and ranking launch points is explored here.

Experiment E: Total Run Time Restricted

In this experiment the number of solver launches is not restricted to 25 launches; instead the total number of launches is set much higher at 200. The important distinction between this experiment and Experiment C is that the maximum total run time is restricted, but the run time per solver launch is not restricted. The solver is allowed to proceed without being terminated prematurely, however, if a poor start point is chosen, a single launch of the local solver could possibly consume all of the search time. The experiment is run twice on PS IIC, first with the total time restriction of 7200s per model, then with the time restriction of 15000s per model.

For some models the local solver finds a solution very quickly, hence, the secondary stopping condition terminates the algorithms after the 200^{th} launch of the local solver. In this regard, Experiment E does have an upper limit on the number of solver launches. In most cases far fewer than 200 local solver launches are actually made within the time limit due to the required time per solver launch for the models tested. The limit is selected to be 200 because this is the default maximum value for Knitro. This hard upper limit is necessary because the local solver can complete a launch in as little as a tenth of a second for some of the models tested. Therefore, it could possibly complete 150,000 launches if given the full 15000s. The amount of time required to read and write to files in this case is impractical.

The four competing algorithms and their parameter settings are:

- **MS**: Basic Multistart using Ipopt as the local solver with the parameter settings listed in Section 7.2.2. The maximum number of solver launches is $\tau = 200$.
- **MS+CC**: Multistart with CC. The local solver is Ipopt with the parameter settings listed in Section 7.2.2. The maximum number of solver launches is $\tau = 200$.
- **MS+CCR**: Multistart with CC and Reduction. $p = 50$ and $\tau = 200$. The local solver is Ipopt with the parameter settings listed in Section 7.2.2.
- **MS+C**: Multistart with clustering using CC, operating in mode 1. $p = 50$, $\tau_c = 25$, $\tau = 200$, and $\omega = 3$. The local solver is Ipopt with the parameter settings listed in

Section 7.2.2.

Experiment F: Comparing to Best Known Solutions

Experiment F compares the results obtained with the global solvers described in this thesis to published best known solutions of the models summarized in Table 7.2. Furthermore, Experiment F compares the published results from MSNLP variants (Section 7.2.2) to the solvers proposed in this thesis directly.

This experiment is set up in a similar fashion to that of Experiment E. The main difference is that an algorithm is stopped if it finds a high-quality solution (a solution with $Gap < 1\%$ as described in Section 7.2.5). The MS, MS+CC, MS+CCR, and MS+C algorithms are run until they find a solution with $Gap < 1\%$ or until they exceed a maximum time limit of 600s. For PS ID the algorithms are also stopped if they reach the limit of 1000 local solver launches per model (these results are listed in Table 7.8 and Table 7.9). Alternatively, for PS IID the algorithms are stopped if they reach the limit of 1000 sample points (these results are listed in Table 7.10).

Only MS, MS+CC, MS+CCR, and MS+C are considered for PS ID. In this case, the 1000 local solver launch limit is selected as it is fair for all the algorithms. The alternative experimental setup chosen for PS IID is so that the results are compared as fairly as possible to those published in [93]. Given this setup, the MS+CCR and MS+C algorithms will not be able to launch the local solver as many times as the other algorithms. This is because MS+CCR and MS+C search the variable space with multiple sample points and CC before deciding where to launch the local solver.

The competing algorithms are:

- **PR**: MSNLP with pure random driver and using the local solver CONOPT.
- **OQ**: MSNLP with the Optquest driver and using the local solver CONOPT.
- **SR**: MSNLP with smart random driver and using the local solver CONOPT.
- **MS**: Basic multistart with the Ipopt local solver. $\tau = 1000$.
- **MS+CC**: Multistart with CC and the Ipopt local solver. $\tau = 1000$.
- **MS+CCR**: Multistart with CC, reduction, and the Ipopt local solver. $p = 50$ and $\tau = 1000$. The second part of this experiment limits the number of samples to 1000, and MS+CCR requires 50 samples per solver launch, therefore, the maximum number

of solver launches MS+CCR will actually make in the second part of this experiment is 20.

- **MS+C**: Multistart with clustering using CC, operating in mode 1. The local solver used is Ipopt. $p = 50$, $\tau_c = 25$, and $\tau = 1000$, but like the aforementioned MS+CCR method, the most solver launches MS+C will make in the second part of this experiment is less than 1000 due to the limit on the number of sample points. The most launches MS+C can possibly make for the second part of this experiment is 500.

The parameters for the PR, OQ, and SR variants of MSNLP are listed in full in Table 1., Section 4 of [93]. These algorithms are run with a maximum time limit of 6000s and a limit of 1000 sample points. For all 7 algorithms, any unbounded variables are constrained with an artificial bound of 10^4, as described in Section 5.2.2. In a secondary experiment, MS, MS+CC, MS+CCR, and MS+C are run using an artificial bound of 10^2. These results are reported in Table 7.8.

Note that the MSNLP algorithms are run with a maximum time limit of 6000s versus 600s for MS, MS+CC, MS+CCR, and MS+C. The MSNLP algorithms are tested on a different computer than MS, MS+CC, MS+CCR, and MS+C so time cannot be used to compare the performance of the algorithms. Furthermore, the MSNLP algorithms use a different local solver which also makes a direct comparison difficult. There is more discussion on these topics in Section 7.3.4.

Tuning CC

Steps 0-4 of MS+C (Alg. 3) are run on the tuning sets to choose the maximum time limit for a single CC run. The results are listed in Table 7.3. The independent variable in the experiments is the maximum time limit for a single CC run, ranging from 0.05s to 0.8s for PS ICa, and from 0.25s to 4s for PS IICa. The table shows:

- **Median violation** - The median violation (see Section 5.2.5) of the most promising launch points found by the clustering routine.
- **Average clusters** - The average number of launch points calculated.
- **Average run time** - The average run time of the clustering method, including the time required to initially sample the space.
- **Feasible solutions** - The percentage of models for which feasible solutions are found by the clustering routine without running a local solver.

Table 7.3: Clustering Times

Tuning Problem Set ICa (22 models)					
Max CC Time (s)	0.05	0.1	0.2	0.4	0.8
Median Violation	15.15	1.07	0.14	0.15	0.11
Average Clusters	16.0	16.0	14.6	15.3	14.5
Average Run Time (s)	1.30	2.36	4.42	8.52	16.59
Feasible Solutions (%)	40.9	40.9	45.5	45.5	45.5

Tuning Problem Set IICa(16 models)					
Max CC Time (s)	0.25	0.5	1	2	4
Median Violation	4260.0	1310.0	912.8	455.9	345.5
Average Clusters	10.3	14.1	12.5	11.1	11.4
Average Run Time (s)	26.7	31.8	52.9	100.8	188.0
Feasible Solutions (%)	0.0	6.3	12.5	12.5	12.5

Based on the tuning statistics, the maximum time for a single CC run is set to 0.05s for PS IC and to 1s for PS IIC. The data in Table 7.3 indicate that longer times yield only relatively small and costly decreases in the median violation, and only a slight if any increase in the percentage of feasible solutions found by the CC method.

Note that there is no expectation that individual CC runs will reach feasibility. CC is only expected to find a point that is close to feasibility: the local solver is expected to proceed to feasibility and optimality. Nonetheless, CC still finds a feasible point for a good fraction of the small models in PS ICa. It finds far fewer feasible solutions in PS IICa because the models are on average much larger and more difficult than those in PS ICa.

7.2.5 Performance Metrics

Throughout this section, the solvers are compared using the *relative percentage gap* (*Gap*) as proposed by Ugray et al.[94]. *Gap* is defined as:

$$Gap = \frac{100(f_s - f_b)}{1 + |f_b|} \tag{7.1}$$

where f_s is the best feasible objective function value found by a solver and f_b is the best known feasible objective function value. *Gap* is a measure of the deviation between a solver's output and the best known solution. The larger *Gap* is, the worse the solver's solution is relative to the best known solution. If *Gap* is equal to 0% then the solutions are the same. For this section, a solver's result is considered a *failure* if its result has $Gap \geq 1\%$.

Some of the results in this section are presented as performance profiles (see [26]). The success rate for the performance profiles in Fig. 7.1 and Fig. 7.2 is the fraction of models for which a particular algorithm found a feasible solution with $Gap < 1\%$. The ratio to best time compares the relative times the algorithms took to find a solution with $Gap < 1\%$. This type of performance profile measures how fast the algorithms find high-quality solutions. The algorithm that performs the best will have a line in the performance profile that is closest to the top left corner. This means that the algorithm solves the most models successfully in the least amount of time.

Experiment C

Experiment C compares a variety of algorithms to determine which one most frequently finds a solution with $Gap < 1\%$ in the shortest amount of time given the same maximum number of local solver launches. Note that this is not the same as imposing an upper time limit. For example, MS+C may determine that a certain model has 3 distinct promising regions and hence will launch the local solver 3 times and stop. In contrast, a traditional multistart algorithm (e.g., MS) will use all 25 local solver launches and hence around 8 times as much computation time. For this reason MS+C is expected to use less time on average, especially for the larger models. MS+C is also expected to reach better solutions more frequently than the other multistart algorithms due to its relatively aggressive exploration of the variable space prior to launching the local solver.

Table 7.4 lists the following data:

- **Feasible Solutions** - The number of models for which the corresponding algorithm found at least one feasible solution.
- **Solutions with Gap < 1%** - The number of models for which the corresponding algorithm found a feasible solution within 1% of the best known solution.
- **Average Time(s)** - The average time per model in seconds taken by the corresponding algorithm.

Experiment D

The second stage of a multistart algorithm in which a local solver is launched is usually assigned much more run time than the initial stage in which launch points are chosen. For instance, the first stage of naive multistart simply selects random launch points from the search space. The computation time involved in selecting random points is negligible when compared to the computation time that a local solver frequently requires for large nonlinear models.

Given that time is often restricted, it may be beneficial to quickly rank the launch points prior to running the local solver. In cases where there is not enough time to run the local solver from each of the selected launch points, the solver will get launched from the most promising points before the time limit is reached. Experiment D tests this hypothesis using the MS+CC and MS+CC* algorithms. Four basic statistics are collected:

- **Number of Initial Incumbents** - The number of models for which an algorithm found a feasible solution within the given time limit.
- **Average Time to Initial Incumbent** - The average time in seconds that an algorithm took to identify a feasible solution.
- **Feasible Solutions** - The number of models for which an algorithm found at least one feasible solution.
- **Solutions with Gap $< 1\%$** - The number of models for which the corresponding algorithm found a feasible solution within 1% of the best known solution.

Experiment E

Experiment E compares the algorithms when they are run for a fixed time on each model. Performance profiles are not used to report these results since all the algorithms are run for the same fixed amount of time. Instead, the results are reported in Table 7.7 with the following metrics:

- **Feasible Solutions** - The number of models for which the corresponding algorithm found at least one feasible solution.
- **Solutions with Gap $< 1\%$** - The number of models for which the corresponding algorithm found a feasible solution with $Gap < 1\%$.

- **Solutions with Gap $< 0.01\%$** - The number of models for which the corresponding algorithm found a feasible solution with $Gap < 0.01\%$. A smaller Gap percentage indicates that a solution is closer to the best known.
- **Average Launches** - The average number of local solver launches per model.
- **Average Time(s)** - The average time per model in seconds taken by the corresponding algorithm.

Experiment F

Experiment F compares the algorithms in terms of finding the best known solutions. All the models in this experiment have published best known solutions. When a global search algorithm finds a feasible solution with $Gap < 1\%$ it is stopped. Note that the local solver is run until it meets its internal stopping conditions, i.e., it does not necessarily stop if it finds a solution with $Gap < 1\%$. The following statistics are reported:

- **Solutions with Gap $< 1\%$** - The number of models for which the corresponding algorithm found a feasible solution with $Gap < 1\%$.
- **Solutions with Gap $< 0.01\%$** - The number of models for which the corresponding algorithm found a feasible solution with $Gap < 0.01\%$. Note that this is not a stopping condition; it is a measure of how many of the successful points (i.e., points with $Gap < 1\%$) also had $Gap < 0.01\%$.
- **Ipopt Time (hrs)** - The total time used by the local solver Ipopt, in hours.
- **Total Time (hrs)** - The total time used by all components of the algorithm, in hours.
- **Timeouts** - The number of models for which an algorithm exceeded the total time limit without finding a feasible solution with $Gap < 1\%$.
- **Ipopt Calls** - The total number of times Ipopt is launched.
- **Infeasible Ipopt Calls** - The total number of times Ipopt returned an infeasible solution.

As described in Section 7.2.4, the results reported for the MSNLP variants PR, OQ, and SR are from [93].

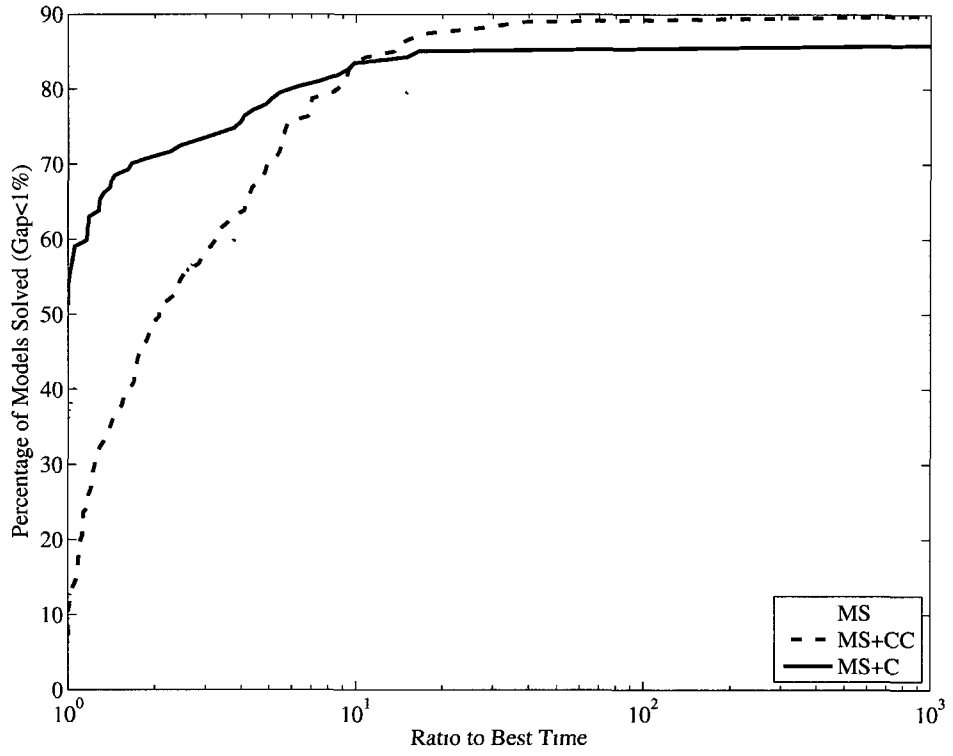


Figure 7.1: Performance Profile for PS ICb

7.3 Numerical Results

The goal of this research is improved methods for multistart global optimization of large scale nonlinear programs. For this reason, the results of main interest are for PS IIC, which contains the largest models. At the same time, the proposed algorithms should not perform poorly on smaller models; hence the results for the other problem sets are also relevant. Experiment C is run on the small and large models found in PS IC and PS IIC, respectively. Experiments D and E are only run on PS IIC which contains the larger models. Experiment F is run on PS ID and PS IID, which contain smaller models in general. These smaller models are selected as they have published best known solutions.

The best results in each table are displayed in boldface font.

7.3.1 Experiment C: Restricting Solver Launches

Results for the Small Models

The performance profile in Fig. 7.1 shows that MS+C is more successful than the other methods for time ratios less than 10. MS+C finds the most solutions with $Gap < 1\%$ the

fastest for 51% of the models. The MS and MS+CC methods only find solutions with $Gap < 1\%$ the fastest for 37% and 7% of the models, respectively. The results listed for PS ICb in Table 7.4 show that MS, MS+CC, and MS+C find solutions with $Gap < 1\%$ for 84%, 90%, and 86% of the models, respectively. The results also show that MS+CC and MS+C find more feasible solutions and more solutions with $Gap < 1\%$ than MS, which demonstrates the value of using CC prior to launching the local solver. For PS ICb, MS+C is the fastest on average while MS+CC is the most robust, successfully solving 90% of the models.

Results for the Large Models

The large models in PS IICb are more difficult to solve: they contain on average many more variables and nonlinear constraints than the models in PS ICb. In general, a much smaller fraction of the large models are solved to optimality than for the small models, likely because of the time restriction on the local solvers. The performance profile shown in Fig 7.2 shows that the MS+C algorithm dominates the other methods over all time ratios. It finds the most solutions with $Gap < 1\%$ the fastest for 35% of the models, and is the most robust, finding solutions with $Gap < 1\%$ for 60% of the models.

Table 7.4 shows that MS+CC and MS+C find slightly more feasible solutions than MS for PS IICb. It also shows that MS+C finds the most solutions with $Gap < 1\%$ and uses the least amount of time on average. MS+C uses 41% less time than MS+CC, its nearest competitor in terms of time, while returning 2 additional solutions with $Gap < 1\%$. MS+C is the fastest and most successful for the large models.

Comparison of MS+C and Knitro 6.0 in Global Mode

Knitro is run on PS IICb so that MS+C could be compared directly to a commercial multistart algorithm. The parameters for Knitro are set so that the stopping conditions of the local solver are as similar as possible to those used by Ipopt and hence a fair comparison is made. The intention is not to compare the local solvers, but the multistart methods they use for global optimization. Comparisons of Knitro and Ipopt as local solvers are found in [67] and in Section 5.1.3 of [97]. Mittelmann's NLP benchmark [67] shows that Knitro is often faster than Ipopt.

The performance profile in Fig. 7.3 indicates that Knitro outperforms MS over most of PS IICb. As reported in Table 7.4, both algorithms found feasible solutions to 35 (59%) of the models. Knitro found the solutions with $Gap < 1\%$ for 31 of the models (vs. 30 for MS). Knitro is faster at finding many of the solutions; however, the total times show

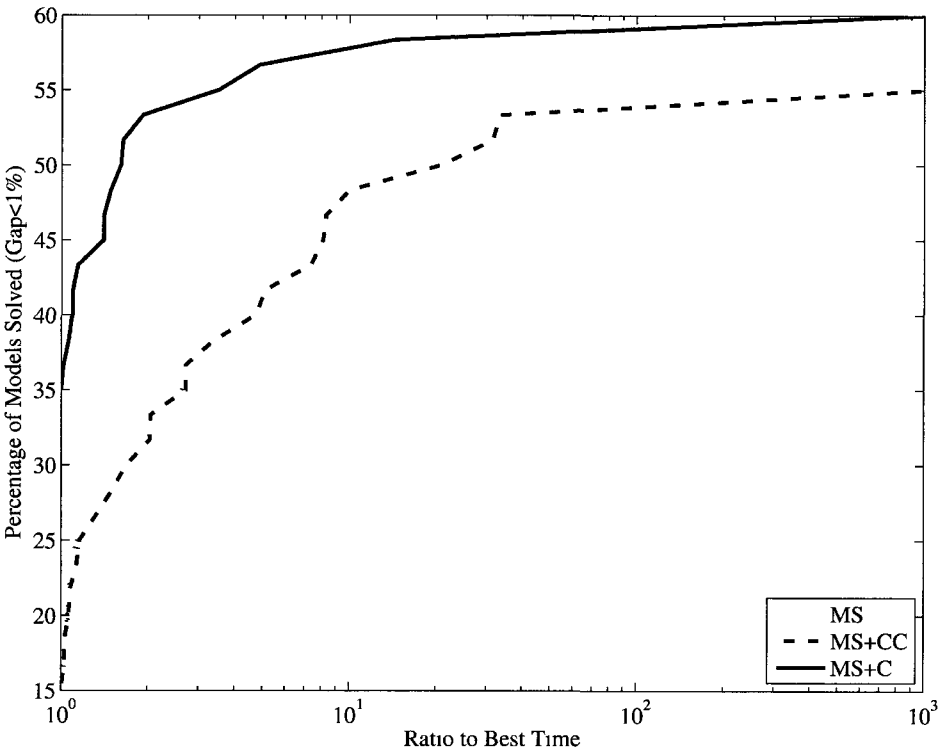


Figure 7 2 Performance Profile for PS IICb

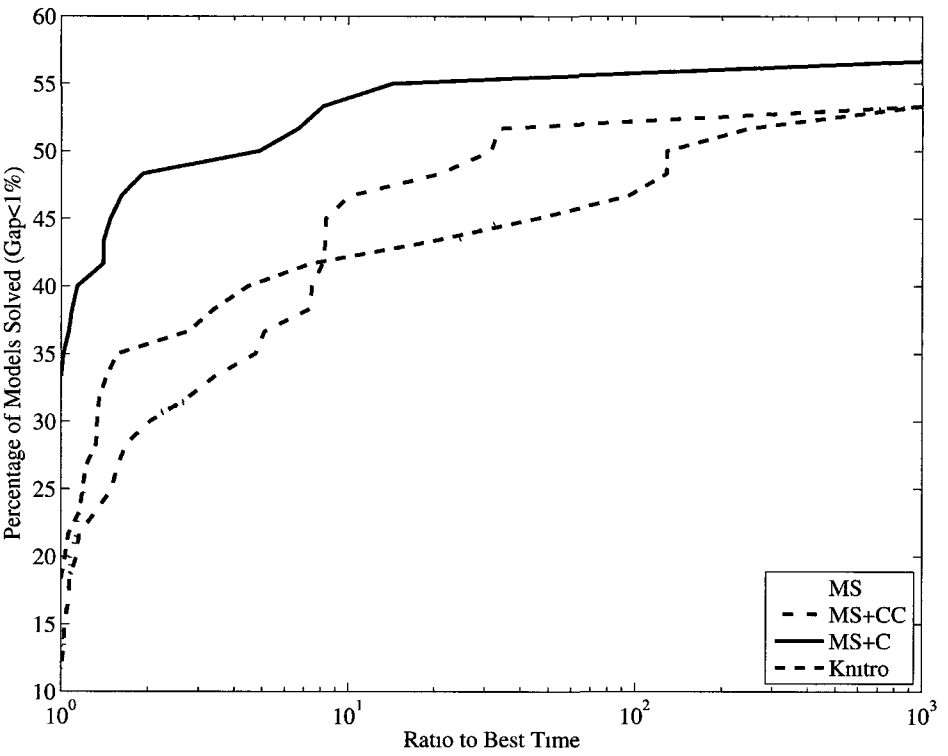


Figure 7.3: Performance Profile, including Knitro, for PS IICb

that Knitro is also slower when it does not find a solution. The average time per model for Knitro is 8859.9s compared to 6867.5s for MS. Overall, Knitro and MS produced similar results for PS IICb.

Fig. 7.3 illustrates that MS+C outperforms Knitro for all ratios to best time. MS+C found feasible solutions for 38 models (vs. 35 for Knitro), and solutions with $Gap < 1\%$ for 33 models (vs. 31 for Knitro). When the number of solutions found with $Gap < 0.01\%$ are compared Knitro performs well, finding 30, 7 more than the next best, MS+C. This means that for 7 models the successful solutions that Knitro found are actually slightly closer to the best known solution than the successful solutions that MS+C found. The biggest difference between the algorithms is the average time used per model: MS+C used only 3737.95s, 58% less time than Knitro.

These results show that the MS+C algorithm compares well against a state of the art commercial multistart method. Solution times are much smaller, and feasible solutions are found for more models. MS+C also outperforms Knitro slightly in terms of number of successful solutions found.

Discussion

An inherent problem with multistart methods for global optimization is that it is unknown when to stop launching the local solver: it is always possible that a better solution will be returned by the next local solver launch. A major advantage of the MS+C algorithm is that it will stop when it thinks that the local solver has been launched once near each feasible region. This can also be a disadvantage since it is possible that a local solver launch from some other part of the same cluster will produce a better solution, especially if the cluster is quite large, in terms of the amount of variable space it covers. The performance profiles in Fig. 7.1 and Fig. 7.2 show that MS+C is generally both faster and more robust than the other algorithms examined. The variable space exploration conducted by the MS+C method is clearly effective in producing recommendations for a small and efficient set of local solver launch points.

Table 7.3 shows that the concentration and clustering method identifies an average of 14-16 clusters for the small models in tuning set ICa and 10-14 clusters for the large models in tuning set IICa. This shows that MS+C is likely to make far fewer than 25 solver launches on average, which accounts for much of its speed advantage. In fact, the MS+C algorithm averaged 14.1 and 11.9 solver launches for problem sets IC and IIC, respectively, while the MS and MS+CC algorithms had no option but to use the full 25 solver launches for each problem. The relatively large numbers of clusters, on average, for both large and small models highlights the opportunity for efficiency improvement by avoiding redundant solver launches near the same regions of attraction. This is exactly the opportunity that is exploited by MS+C.

Table 7.4 shows that MS+C does not do quite as well as MS+CC in terms of the number of feasible solutions and solutions with $Gap < 1\%$ for the small models in PS ICb. Given the experimental setup for PS ICb which specifies $p = \tau = 25$, this is not unexpected. MS+CC launches the local solver from all 25 points, while MS+C launches from only a subset of the same 25 points (i.e. only the most promising point for each cluster found). In a few cases, the most promising launch point for a cluster is not actually the best launch point for that cluster, however, the results also show that it frequently is the best choice. For better exploration of the variable space prior to a local solver launch, p is set to a larger value than τ. The improved effect when $p = 50$ and $\tau = 25$ is seen in the results for the large models in PS IIC.

The importance of using CC (in particular L_Basic_Aug_3) to improve the initial point prior to the local solver launch is demonstrated in Table 7.4: the MS+CC and MS+C algorithms both find solutions with $Gap < 1\%$ more often and faster than the simple MS

Table 7.4: Experiment C Results

PS ICb (126 small models)			
Algorithm	MS	MS+CC	MS+C
Feasible Solutions	113	**122**	120
Solutions with $Gap < 1\%$	106	**113**	108
Average Time (s)	115.6	106.6	**94.4**

PS IICb (59 large models)				
Algorithm	MS	MS+CC	MS+C	Knitro
Feasible Solutions	35	**38**	**38**	35
Solutions with $Gap < 1\%$	30	31	**33**	31
Solutions with $Gap < 0.01\%$	21	21	23	**30**
Average Time (s)	6867.5	6385.4	**3737.9**	8859.9

variant. CC accounts for only 1.4% of the total solution time of MS+CC in PS ICb and 0.4% of the total solution time of MS+CC in PS IICb. Similarly, CC and clustering account for only a small fraction of the total solution time for the MS+C algorithm, specifically 1.6% for PS ICb and 1.5% for PS IICb. These relatively small investments in exploration of the variable space prior to launching the local solver generate a large return in terms of reduced total solution time and improved solution success.

Table 7.4 shows that for the large models in PS IICb, MS+C is not only the fastest by a significant margin, using 41% less time than the next fastest method on average, but it also solves the most models to $Gap < 1\%$. The performance profile in Fig 7.3 shows that MS+C is a promising alternative to the Knitro multistart algorithm, included to represent the state of current commercial solvers available. MS+C provides valuable improvements in both the effectiveness and efficiency of multistart methods for global optimization.

7.3.2 Experiment D: Ranking Candidate Launch Points

Initial Incumbent Results

The results from the first part of Experiment D are summarized in Table 7.5. Given the 600s time limit, MS+CC and MS+CC* found initial incumbent solutions for 21 and 25 of

Table 7.5: Initial Incumbent Results for Launch Point Ranking. The average time is calculated over the 19 models for which both algorithms found an initial incumbent solution.

Algorithm:	MS+CC	MS+CC*
Number of initial incumbents:	21	**25**
Average time to initial incumbent:	68.4s	**53.5s**

Table 7.6: Overall Results for Launch Point Ranking

Algorithm:	MS+CC	MS+CC*
Feasible Solutions	**38**	31
Solutions with $Gap < 1\%$	**29**	26
Solutions with $Gap < 0.01\%$	**26**	22

the 59 models, respectively. Further, the average time to initial incumbent is slightly less for MS+CC*. These results show that ranking the candidate launch points prior to running the local solver has a positive effect if time is overly restricted. The term *overly restricted* is used because there are 32 models for which neither algorithm found an incumbent solution in the 600s time limit.

It should also be noted that a failure by MS+CC* does not imply a failure by MS+CC, that is, ranking did not always produce a positive outcome. There are two models for which MS+CC found incumbent solutions in the 600s time limit and MS+CC* did not. Therefore, ranking launch points may have a net positive effect; however, it is not always advantageous.

Overall Results

Table 7.6 summarizes the results for the second part of Experiment D. The results in Table 7.6 show that ranking does not have a net positive effect on the overall solution quality when the algorithms are run with a greater maximum time limit. MS+CC is the better algorithm in terms of finding feasible solutions, returning 38. MS+CC* found far fewer feasible solutions: 31. In terms of success (finding solutions with $Gap < 1\%$), MS+CC only slightly outperformed MS+CC*, finding 27 best solutions versus 26. Similarly, a greater number of the MS+CC solutions compared to the MS+CC* solutions satisfied $Gap < 0.01\%$. These results indicate that given longer run times, ranking candidate launch points does not have a positive effect on the MS+CC algorithm.

Discussion

There are two keys to a good set of launch points. The first is quality. As shown in Section 5.4.1, good launch points tend to be close to the feasible region. For a barrier solver, good launch points typically over-satisfy many of the inequality constraints and are close to satisfying many of the equality constraints. The second key to a good set of launch points is variety of location. A local solver will perform differently depending on the start point from which it is launched, but if the local solver is launched from two points that are relatively close together it will likely return the same solution.

Ranking the launch points ensures that the local solver is launched from the best known points, however, it does not guarantee that the points are diverse. In fact, ranking may actually increase the odds that the local solver makes redundant launches if many of the most promising points are from the same basin of attraction. This is the reason that MS+CC* is outperformed by MS+CC.

The results show that ranking candidate launch points positively influences the number of initial incumbents found when maximum run time is restricted. Overall, ranking candidate launch points is not necessarily positive as it may decrease launch point variety, and ultimately cause a multistart algorithm to perform worse. For this reason, only the MS+CC algorithm is used in the experiments presented in the following sections.

7.3.3 Experiment E: Restricting Run Time

Results

Experiment E tests which algorithms perform the best given a limited run time. Table 7.7 highlights key results from this experiment. The test is run twice, first with a time limit of 7200s and then with a time limit of 15000s. 7200s is close to the average time taken by the algorithms in Experiment C and 15000s is the maximum accumulated time the algorithms were allotted in Experiment C.

MS+CC found the most feasible solutions, 38, given the 7200s time limit. MS+CCR and MS+C were close finding 36 and 37 feasible solutions, respectively. MS and Knitro found the fewest feasible solutions, 28 and 31, respectively. This is likely because the MS and Knitro algorithms do not use CC to improve solver launch points. A similar trend exists in the solutions data. The CC-based algorithms MS+C, MS+CCR, and MS+CC identified 29, 30, and 31 solutions with $Gap < 1\%$, respectively, while MS and Knitro only found 23 and 26, respectively. The numbers of solutions found with $Gap < 0.01\%$ are similar. These results show that using CC to search the variable space for improved solver

Table 7.7: Experiment E Results

Time=7200s					
	MS	MS+CC	MS+CCR	MS+C	Knitro
Feasible Solutions	28	**38**	36	37	31
Solutions with $Gap < 1\%$	23	29	30	**31**	26
Solutions with $Gap < 0.01\%$	22	**26**	**26**	**26**	24
Average Launches	71.3	81.6	63.1	**58.1**	-
Average Time (s)	5,653.0	**5,316.0**	5,898.6	6,008.5	7,411.0

Time=15000s					
	MS	MS+CC	MS+CCR	MS+C	Knitro
Feasible Solutions	30	**39**	**39**	37	31
Solutions with $Gap < 1\%$	24	29	31	**33**	26
Solutions with $Gap < 0.01\%$	22	24	22	**26**	24
Average Launches	92.5	92.4	72.9	**59.0**	-
Average Time (s)	**10,514.2**	10,579.6	11,407.8	11,958.8	14,965.2

launch points definitely has a positive result.

The results are similar for the 15000s experiment; the CC-based algorithms outperformed MS and Knitro. The extra time allowed MS, MS+CC, and MS+CCR to find more feasible solutions, but MS+C is still the top performer in terms of best solutions. Overall, MS+C is the most robust, finding the most high-quality solutions in each of the two tests.

Discussion

Terminating the search algorithms after 200 local solver launches has a pronounced effect on the average statistics displayed in Table 7.7, the most obvious being that the average times are less than the given time limits (except in Knitro's case, which is discussed below). This is directly due to the fact that for some of the models (15 when $Time = 7200s$, and 19 when $Time = 15000s$) the solver runs are quick enough that the search algorithms completed 200 local solver launches before the time limit is reached. Given the increase in maximum time to 15000s, the average numbers of solver launches did not increase too much due to the 200 solver launch limit. As a result of the solver launch limit, the number of launches only increased for the models that require a large amount of local solver search time, i.e.,

the models for which fewer than 200 solver launches are possible given the maximum time limits. Since these models require a large amount of time per solver launch, only a few extra launches are possible with the extended time limit.

For an unknown reason Knitro exceeded the time limit for some models. This is obvious in the average time result of the 7200s experiment that reports that Knitro used 7,411.0s on average. It is also true for the 15000s experiment. There are 21 models that Knitro reported using more time than the limit. For a couple models Knitro reported using almost twice the maximum allowed time.

The search routines that require a greater ratio of CC search time to solver search time (MS+CCR and MS+C) used fewer solver launches on average. This is expected because using CC to search the variable space takes away from the time available to run the local solver. The exception is the MS+CC algorithm. In the 7200s test it actually uses more solver launches on average than MS and almost the same number of launches in the 15000s case. This is because CC reduces the amount of time the solver requires per run, as shown in Chapter 5.

Table 7.7 does not report the average numbers of solver launches for Knitro as the data is not available.

7.3.4 Experiment F: Finding Best Objective Function Values

Results

Experiment F tests the algorithms' abilities to find the best known solutions. Each of the models used in this experiment has a previously discovered best known solution. The algorithms stopped immediately if they found a solution with $Gap < 1\%$, compared to the best known solution. Otherwise, the algorithms proceeded until they met maximum time, sample, or launch restraints.

Two sets of results are listed in Table 7.8. The first set reports the data using the artificial bound 10^4, while the second set reports the data with the artificial bound 10^2, i.e., a smaller search space. The results are expected to be better when the search space is smaller as most of the models should be easier to solve with the tighter bounds. The artificial bound is applied by the method described in Section 5.2.2.

Table 7.8 reports results for the 366 models in PS ID. For 250 of these models, MS found the best known solution on the first local solver launch. This is likely because either the bounds on these models are relatively tight or because the model contains only one feasible region and optimum. There were an additional 3 models for which the solver timed

out on its first and only launch. Therefore, there were 253 models for which only one solver launch was performed. This thesis targets difficult models with multiple disconnected feasible regions, thus the results of the other 113 models for which the local solver has difficulty finding a solution are of greater interest. The results for the 113 difficult models are found in Table 7.9. Note that the average number of variables and average number of nonlinear constraints for the 253 *easy* models are 54.5 and 53.5, respectively. The averages for the 113 *difficult* models are higher; specifically, the average number of variables and average number of nonlinear constraints are 63.5 and 61.4, respectively.

MS+C is the best performing algorithm when the artificial bound is set to 10^4. MS+C found the most solutions with $Gap < 1\%$ and with $Gap < 0.01\%$, used the least amount of time (by a small margin), timed out the least often, and used the fewest local solver calls. Furthermore, the local solver returned infeasible solutions least often when MS+C selects launch points.

The results when the bounds are decreased are mixed. Although MS+C found the most solutions with $Gap < 1\%$, MS+CCR found the most solutions with $Gap < 0.01\%$. MS+CC used the least amount of time, however, MS+C did not use much more. Both MS+CCR and MS+C timed out the least often: 9 times each. MS+C uses the fewest local solver calls. The other methods all use considerably fewer local solver calls than when the artificial bound is larger. This is an indication that the models are easier to solve when the artificial bounds are decreased. MS+CCR caused the fewest local solver launches to return with infeasible solutions, far fewer than when the artificial bounds are larger. In general, all the algorithms performed better with the tighter bounds.

Table 7.9 lists results from the subset of 113 models for which MS required more than one solver launch to find the best known solution. These models are more likely to contain multiple feasible regions and optima. They are also more difficult. As shown by Table 7.9, these models are responsible for most of the runtime and local solver calls used by the proposed algorithms. The results also show that MS+C outperforms the other algorithms in most of the categories. It finds the most high-quality solutions while using the fewest solver launches and least amount of time.

Fig. 7.4 contains two performance profiles: one for the complete set of 366 models, and another for the subset of 113 models. Fig. 7.4a illustrates that MS+CC is the fastest algorithm for many of the models. 253 of the 366 models required one local solver launch. For these models the extra time MS+CCR and MS+C used to search the space before launching the local solver negatively affected their time results. MS+CC outperformed MS because the pair of CC and local solver runs faster on average than the local solver by itself.

Table 7.8: Experiment F Results: Comparing Multistart Algorithms on 366 COCONUT Benchmark Models with Known Solutions

Artificial Bound = 10^4				
Algorithm	MS	MS+CC	MS+CCR	MS+C
Solutions with $Gap < 1\%$	327	343	342	**346**
Solutions with $Gap < 0.01\%$	307	325	326	**329**
Ipopt Time (hrs)	3.68	2.50	**1.31**	2.24
Total Time (hrs)	3.68	2.55	2.56	**2.49**
Timeouts	19	13	12	**11**
Ipopt Calls	22,929	13,025	14,959	**11,787**
Infeasible Ipopt Calls	3,310	1,502	1,912	**1,095**

Artificial Bound = 10^2				
Algorithm	MS	MS+CC	MS+CCR	MS+C
Solutions with $Gap < 1\%$	332	346	347	**348**
Solutions with $Gap < 0.01\%$	313	327	**331**	329
Ipopt Time (hrs)	3.67	1.94	**1.35**	1.93
Total Time (hrs)	3.67	**1.96**	2.38	2.14
Timeouts	20	10	**9**	**9**
Ipopt Calls	18,365	12,553	12,032	**11,560**
Infeasible Ipopt Calls	2,211	1,095	**178**	543

Table 7.9: Comparing Multistart Algorithms on 113 Models for which MS took more than one launch to find the best known solution (Artificial Bound = 10^4).

Algorithm	MS	MS+CC	MS+CCR	MS+C
Solutions with $Gap < 1\%$	77	92	91	**95**
Solutions with $Gap < 0.01\%$	71	85	87	**89**
Ipopt Time (hrs)	3.15	2.15	**0.96**	1.89
Total Time (hrs)	3.15	2.19	2.17	**2.10**
Timeouts	16	11	10	**9**
Ipopt Calls	22,676	12,692	14,444	**11,452**
Infeasible Ipopt Calls	3,307	1,500	1,910	**1,093**

Table 7.10: Comparing Against MSNLP on 131 COCONUT Benchmark Models with Artificial Bound = 10^4

Algorithm	PR	OQ	SR	MS	MS+CC	MS+CCR	MS+C
Number $Gap < 1\%$	128	129	**130**	126	127	125	129
Number $Gap < 0.01\%$	120	**124**	122	118	120	118	123

The performance profile also shows that MS+C found the most solutions with $Gap < 1\%$.

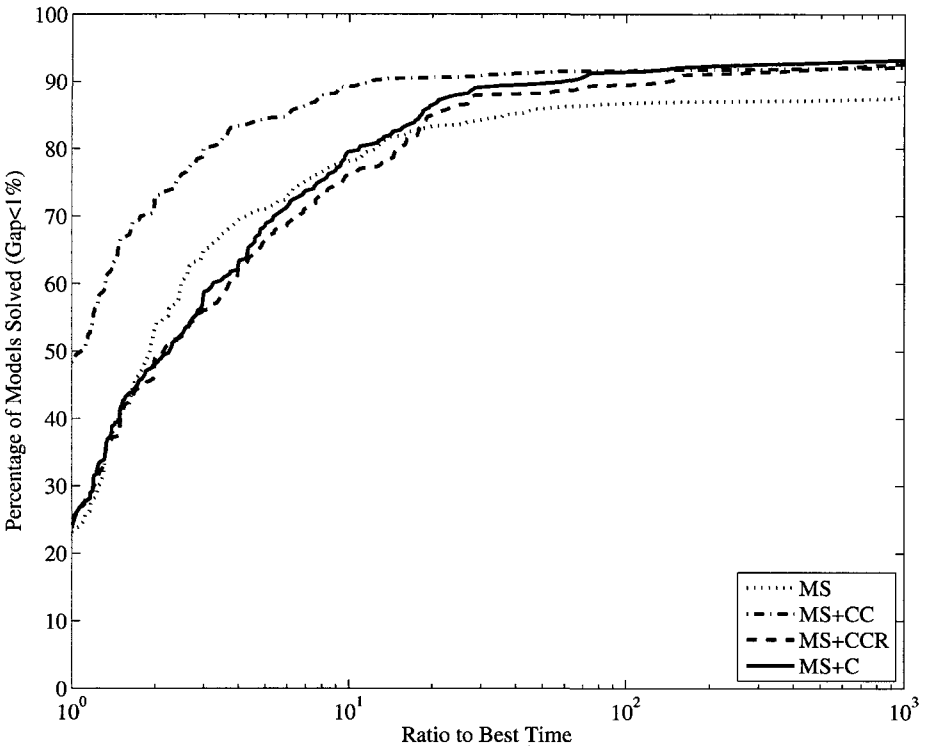
The performance profile in Fig. 7.4b illustrates how the performance gap decreases when the *easier* models are removed from the test set. Although MS+CC is still the fastest for many of the models, the performance gap between it and the MS+CCR and MS+C algorithms closes. This is because the extensive searches that the latter algorithms perform before launching the local solver are more valuable for the difficult models. The performance profile shows that MS+C found the most solutions with $Gap < 1\%$ for this subset of models.

Table 7.10 compares the proposed algorithms with the commercial solver MSNLP using three different *random drivers*, namely, PR, OQ, and SR. The results indicate that MS+C is the best of the proposed algorithms as it finds the most solutions in both *Gap* categories. The best MSNLP variant is SR for $Gap < 1\%$ and OQ when $Gap < 0.01\%$. In both cases, MS+C obtains competitive results with the MSNLP variants.

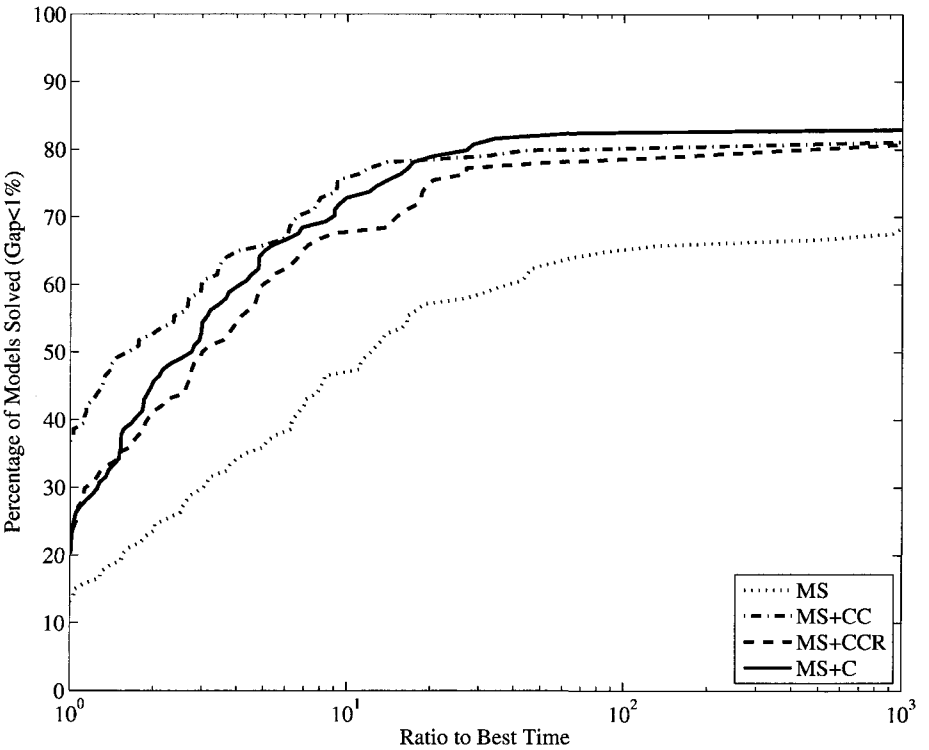
Discussion

Of the 131 models in PS IID, the proposed algorithms MS+CC, MS+CCR, and MS+C collectively failed to find solutions with $Gap < 1\%$ on a total of 6 different models. This is because the combination of CC and Ipopt frequently failed on the same models. Although the MSNLP variants failed fewer times collectively, they still failed on a total of 6 different models. There are no models for which more than a single MSNLP variant failed. Furthermore, the set of models that the MSNLP variants failed on is almost entirely different than the set of models that MS+CC, MS+CCR, and MS+C failed on. Only one model is in both sets. MS+C is successful for every one of the 6 models that at least one of the MSNLP variants failed on. This is likely due to the fact that different local solvers are used by the two sets of algorithms.

The MSNLP variants and MS+C produce similar results for PS IID. The algorithms are not compared on a time basis because they were run on different computers with different hardware specifications. The comparison between the global search algorithms



(a) All 366 models.



(b) Subset of 113 models.

Figure 7.4: Performance Profiles for PS ID: (a) All models are included, (b) Only the subset of 113 models for which MS required more than one local solver launch to find the best known solution are included.

is also complicated by the different local solvers used, each with their own strengths and weaknesses. To isolate the performance of the global search routines the same local solver would have to be used by each algorithm. In terms of practical use, the results suggest that using some combination of both of the local solvers would probably be advantageous.

All the algorithms solve a large fraction of the models because on average, the models in PS IID are not particularly difficult.

7.4 Summary

This chapter uses empirical results to show the promise of the proposed algorithms, especially MS+C. In Experiment C, MS+C typically finds the highest-quality solution in significantly less time than competing algorithms, including Knitro. This highlights the ability of MS+C to reduce the number of solver launches while maintaining solution integrity. In Experiment E, MS+C did not find feasible solutions as often as MS+CC and MS+CCR, however, it was still most robust at finding solutions with $Gap < 1\%$. The results from Experiment F show that MS+C typically finds the most high-quality solutions as compared to the other proposed methods. This experiment also shows that MS+C is competitive with three variants of the solver MSNLP.

Experiment D shows that ranking candidate launch points is not necessarily advantageous. Ranking the launch points effectively prioritizes points on their relative promise, however, the variety of the launch points is sometimes reduced, especially if the algorithm only has time to launch the local solver a few times. The result is that the search space is not explored adequately and a lower quality solution is returned.

Using CC to search the variable space is shown to be an effective method in this chapter. The algorithms using CC outperformed MS in most cases and in some cases the performance gap was very wide. The relative performance of MS+C to the other CC-based algorithms shows that the clustering routine adds additional value. MS+C successfully provides the local solver with a set of high-quality and diverse launch points. The comparisons of MS+C to Knitro and MSNLP show that the proposed method is competitive with currently available commercial global optimization software. A list of all the best known solutions is found in Appendix A.

Chapter 8

Conclusions

1. The primary goal declared in Chapter 4 has been achieved: an improved multistart heuristic framework for global optimization was developed. It is both effective and efficient at solving a wide range of large-scale constrained continuous nonlinear programs, as shown in Chapter 7. The secondary goal has also been achieved: improved methods for quickly seeking feasibility in large-scale nonlinear programs were devised. Results in Chapter 5 show that the new methods can both increase feasibility rates and decrease total run times when they are paired with a local solver.

2. A critical distance for clustering can be automatically estimated from the distribution of inter-point distances of concentrated points. This is illustrated in Section 6.3. Furthermore, the resulting clusters can be used to determine high quality local solver launch points. This is shown in Section 7.3.

3. Augmentation improves CC algorithms that use Basic consensus and linear feasibility vectors but does not consistently improve the performance of CC variants that use FDfar or SUM consensus and linear feasibility vectors. Of the CC variants tested, L_Basic_A_3 is best at reducing median violation given time and iteration constraints, and improved Ipopt's overall performance the most, both increasing Ipopt's feasibility rate and decreasing the total time required. This is shown in Sections 5.3.1 and 5.3.2.

4. SUM-based consensus methods produce good results overall when combined with linear feasibility vectors and relatively good results when paired with quadratic feasibility vectors. This is shown in Section 5.3.1. L_SUM and L_SUM_A_11 are two of the best performing CC algorithms overall. They perform exceptionally well at quickly reducing constraint violation. L_SUM proved to be successful at improving

Ipopt's feasibility rate and decreasing the amount of overall run time, although not by the same margins as L_Basic_A_3. This is shown in Section 5.3.2.

5. CC may not monotonically reduce the violation at each iteration, hence the CC end point may not be the point with the lowest violation. CC performs better when all of its intermediate points are tracked and the best point, in terms of violation, is returned. This is shown in Section 5.3.1.

6. Successful launch points for barrier solvers satisfy many nonlinear inequality constraints and are close to satisfying many nonlinear equality constraints. L_Basic_Aug_3 and L_SUM, applied only to the nonlinear subsets of the violated constraints, were the best CC variants at quickly locating points with these characteristics (Section 5.4.1).

7. Applying the CC algorithms exclusively to nonlinear constraints improves the feasibility fraction results of the CC algorithms. This is shown in Sections 5.1.4 and 5.4.1.

8. Ranking candidate launch points prior to calling the local solver does not necessarily improve performance. Section 7.3.2 shows that a multistart method may find a feasible incumbent solution quicker when using a set of ranked launch points, but overall, the variety of unranked launch points proves to be more effective.

9. MS+C is competitive with the commercial software Knitro 6.0 running in multistart mode (Section 7.3.3).

10. MS+C is competitive with the commercial software MSNLP (Section 7.3.4).

11. The objective function can be converted into an aspiration constraint for CC during the concentration phase of the global search routine. Updating the aspiration value throughout the process can help to further concentrate CC output points to points near optimality rather than just points near feasibility, however, there can also be negative consequences. This is illustrated in Sections 6.3.2 and 6.4.

12. Quadratic feasibility vector calculations take much more time than linear calculations. As a result, the quadratic-based CC variants complete fewer iterations and generally achieve worse overall median violations than the linear-based variants. The quadratic variants that only apply quadratic feasibility vector calculations to quadratic constraints (and use the linear feasibility vector calculations for all other nonlinear constraints) outperformed the variants that used the quadratic feasibility

vector calculations for all nonlinear constraints. The best performing quadratic variant is Q_SUM_q, however it performs relatively poorly versus the linear variants. This is shown in Section 5.3.1.

8.1 Summary of Contributions

1. Developed a method for identifying local solver launch points that are distributed among different promising regions of the search space so that redundant local solver launches are reduced, and which compares well to the state of the art.

2. Developed a novel method for estimating the critical clustering distance from the inter-point distribution of concentrated points and demonstrated how the resulting clusters can be used to select a set of high quality launch points for a local solver.

3. Developed a predictor-corrector approach for CC whereby the previous feasibility vectors are quickly augmented to improve results.

4. Introduced the concept of using the best intermediate CC point in terms of constraint violation as the local solver launch point rather than the CC end point.

5. Introduced the concept of applying CC to the subset of violated nonlinear constraints only. This method concentrates computational effort towards the nonlinear constraints which are typically the most challenging aspect of many optimization models.

6. Performed an analysis of solver launch points over a wide variety of models to verify the general characteristics of successful launch points for barrier solvers.

7. Developed a consensus variant for CC that effectively sums the feasibility vectors.

8. Developed an alternative feasibility vector calculation for CC that uses an additional quadratic term.

8.2 Future Research

1. MS+C may be improved by using the start, intermediate, and end points of the CC runs to determine the size, shape, and location of the basins of attraction for identified clusters so that new random CC start points can be placed in unexplored regions of the variable space. Alternatively, if a basin of attraction is found to be large

it may be beneficial to identify multiple launch points from within the same basin. With these improvements MS+C would conduct a more thorough investigation of the search space.

2. The routine for extracting a critical distance from the inter-point distance distribution may be improved by using shape analysis to identify specific conditions. Preconceived responses could be initiated when certain conditions are identified. For example, if the inter-point distance distribution is a unimodal bell-type curve the model may only have one feasible region. In this case it might be efficient to launch the local solver once from the center of the cluster of points.

3. The introduction of adaptive methods for setting parameter values could improve the performance of MS+C as well as the CC algorithms. For example, the number of initial sample points for MS+C could be based on the size of the model or it could change dynamically based on the performance of the CC runs.

4. L_Basic_A_3 may be improved by augmenting a superposition (possibly weighted) of the previous consensus vectors rather than just the previous consensus vector.

5. Current versions of CC operate on the subset of violated constraints in order to find points that are close to feasibility. To incorporate optimality an aspiration constraint can be introduced as shown in Section 6.1.5. Alternatively, CC with an aspiration constraint could be applied to the barrier form of the problem, the same problem Knitro and Ipopt solve. This method is more complicated as it requires the introduction of many new variables, however it is unknown what the benefits, if any, would be.

6. There are certain circumstances that local solvers have trouble dealing with. For instance, there are a variety of situations that cause barrier solvers like Knitro and Ipopt to sometimes get stuck in regions of the variable space where they continuously make short steps and very little progress. In these cases, it may be beneficial to switch to CC in an attempt to find a new launch point for the solver. CC would be used throughout the optimization process, quickly finding a new point for a barrier method whenever the solver's progress is insufficient, rather than just at the beginning. Currently, conjugate gradient methods are used in this role by the Knitro and Ipopt solvers. Using CC first before trying the other methods may enhance performance because an iteration of CC is relatively fast compared to the other methods, espe-

cially for large-scale models. The result would be a new local solver that is possibly faster than the current versions of Knitro and Ipopt.

References

[1] C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, and J.S. Park. Fast algorithms for projected clustering. *SIGMOD Rec.*, 28(2):61–72, 1999.

[2] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:382–392, 1954.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, 1998.

[4] R. Aharoni and Y. Censor. Block-iterative projection methods for parallel computation of solutions to convex feasibility problems. *Linear Algebra Appl.*, 120:165–175, 1989.

[5] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, Philadelphia, PA, USA, 2007.

[6] R.W. Becker and G.V. Lago. A global optimization algorithm. *In Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, pages 3–12, 1970.

[7] R.E. Bellman. *Adaptive Control Processes.* Princeton University Press, Princeton, NJ, 1961.

[8] I.O. Bohachevsky, M.E. Johnson, and M.L. Stein. Generalized simulated annealing for function optimization. *Technometrics*, 28(3):209–217, 1986.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems.* Oxford University Press, Inc., New York, NY, USA, 1999.

[10] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB95)*, pages 574–584, 1995.

[11] S.H. Brooks. A discussion of random methods for seeking maxima. *OPERATIONS RESEARCH*, 6(2):244–251, 1958.

[12] T.E. Bruns. Topology optimization by penalty (top) method. *Computer Methods in Applied Mechanics and Engineering*, 196(45-48):4430–4443, 2007.

[13] R. H. Byrd, J. Nocedal, and R. A. Waltz. Knitro: An integrated package for nonlinear optimization. *in Large-Scale Nonlinear Optimization, G. di Pillo and M. Roma, eds*, pages 35–59, 2006.

[14] Y. Censor, D. Gordon, and R. Gordon. Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems. *Parallel Computing*, 27:777–808, 2004.

[15] Y. Censor and A. Lent. Cyclic subgradient projections. *Mathematical Programming*, 24:233–235, 1982.

[16] Y. Censor and S. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications.* Oxford Univ. Press, New York, NY, 1997.

[17] J.W. Chinneck. The constraint consensus method for finding approximately the feasible points in nonlinear programs. *INFORMS Journal on Computing*, 16(3):255–265, 2004.

[18] J.W. Chinneck. *Feasibility and Infeasibility in Optimization*, volume 118 of *International Series in Operations Research & Management Science.* Springer, New York, NY, first edition, 2008.

[19] P. Chootinan and A. Chen. Constraint handling in genetic algorithms using a gradient-based repair method. *Comput. Oper. Res.*, 33(8):2263–2281, 2006.

[20] M. Clerc. *Particle Swarm Optimization.* ISTE, 1999.

[21] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, Feb 2002.

[22] G. Coath and S.K. Halgamuge. A comparison of constraint-handling methods for the application of particle swarm optimization to constrained nonlinear optimization problems. In *Proc. of Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, pages 2419–2425 Vol.4, 2003.

[23] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, 2000.

[24] J. Denzinger and J. Kidney. Evaluating different genetic operators in the testing for unwanted emergent behavior using evolutionary learning of behavior. In *Proc. of the IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, pages 23–29, Washington, DC, USA, 2006.

[25] A.B. Djurisic, J.M. Elazar, and A.D. Rakic. Genetic algorithms for continuous optimization problems - a concept of parameter-space size adjustment. *J. of Physics A: Mathematical and General*, 30(22):7849–7861, 1997.

[26] E. D. Dolan and J. J. More. Benchmarking optimization software with performance profiles. *Math. Program.*, 91:201–213, 2002.

[27] A. Drud. Conopt: A grg code for large sparse dynamic nonlinear optimization problems. *Math. Program.*, 31:153–191, 1985.

[28] O. A. Elwakeil and J. S. Arora. Two algorithms for global optimization of general nlp problems. *Int. J. for numerical methods in engineering*, 39:3305–3325, 1996.

[29] X. Z. Fern and C. E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *Proc. of the Twentieth International Conference on Machine Learning (ICML-2003)*, Washington DC, 2003.

[30] C.A. Floudas. *Deterministic Global Optimization: Theory, Methods and Applications*, volume 37 of *Nonconvex Optimization and Its Applications*. Springer-Verlag New York, Secaucus, NJ, USA, 2005.

[31] R. Fourer, D.M.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2002.

[32] S. Garnier, J. Gautrais, and G. Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31, 2007.

[33] D.M. Gay. Hooking your solver to ampl. Technical report, Computing Sciences Research Center, Bell Laboratories, 1997.

[34] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[35] F. Glover. A template for scatter search and path relinking. *Lecture Notes In Computer Science, Springer-Verlag*, 1363:3–54, 1997.

[36] L.G. Gubin, B.T. Polyak, and E.V. Raik. The method of projections for finding the common point of convex sets. *USSR Computational Mathematics and Mathematical Physics*, 7(6):1 – 24, 1967.

[37] Q. He and L. Wang. A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. *Applied Mathematics and Computation*, 186:1407–1422, March 2007.

[38] A. Hinneburg, C. Aggarwal, and D.A. Keim. What is the nearest neighbor in high dimensional spaces? In *VLDB '00: Proc. of the 26th Int. Conf. on Very Large Data Bases*, pages 506–515, San Francisco, CA, USA, 2000.

[39] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *J. ACM*, 8(2):212–229, 1961.

[40] R. Horst and P. M. Pardalos, editors. *Handbook of Global Optimization.* Kluwer Academic Publishers, Dordrecht, Netherlands, 1995.

[41] X. Hu and R. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *Proc. of the Sixth World Multiconference on Systemics, Cybernetics and Informatics.*, 2002.

[42] W. Ibrahim and J.W. Chinneck. Improving solver success in reaching feasibility for sets of nonlinear constraints. *Comput. Oper. Res.*, 35(5):1394–1411, 2008.

[43] L. Ingber. Adaptive simulated annealing (asa): lessons learned. *J. Control and Cybernetics*, 25(1):33–54, 1996.

[44] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *Systems, Man, and Cybernetics, Part B, IEEE Trans. on*, 35(6):1272–1282, Dec. 2005.

[45] C. Jansson and O. Knueppel. A global minimization method: the multi-dimensional case. *Technische Informatik III, TU Hamburg-Hamburg*, page 35 (problem "BR"), Jan. 1992.

[46] S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[47] D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, 1998.

[48] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods part i: Clustering methods. *Math. Program.*, 37:27–56, 1987.

[49] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods part ii: Multi level methods. *Math. Program.*, 37:57–78, 1987.

[50] R.B. Kearfott. An overview of the globsol package for verified global optimization, 2002. University of Louisiana at Lafayette. Accessed online 14-May-2008, from `http://www.mat.univie.ac.at/ neum/glopt/mss/Kea02.pdf`.

[51] J. Kennedy. Bare bones particle swarms. *Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE*, pages 80–87, 24-26 April 2003.

[52] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceeding of the IEEE International Conference on Neural Networks*, pages 1942–1947, 1995.

[53] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1671–1676, Washington, DC, USA, 2002.

[54] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, News Series*, 220(4598):671–680, 1983.

[55] M. Laguna and R. Marti. *Scatter Search: Methodology and Implementations in C.* Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[56] L. Lasdon and J. Plummer. Multistart algorithms for seeking feasibility. *Computers and Operations Research.*, 35(5):1379–1393, May 2008.

[57] Y. Liu, Z. Qin, Z. Shi, and J. Chen. *Content Computing: Rule Discovery with Particle Swarm Optimization.* Springer Berlin / Heidelberg, 2004.

[58] M. MacLeod. Multistart constraint consensus for seeking feasibility in nonlinear programs. MASc Thesis, Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2006.

[59] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I - Convex underestimating problems. *Math. Program.*, 10:147175, 1976.

[60] G.P. McCormick. *Nonlinear programming: Theory, algorithms, and applications.* John Wiley and Sons, New York, NY, first edition edition, 1983.

[61] M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[62] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *Evolutionary Computation, IEEE Trans. on*, 8(3):204–210, June 2004.

[63] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. *Proc. of the 4th Annual Conf. on Evolutionary Programming*, pages 135–155, 1995.

[64] Z. Michalewicz and C.Z. Janikow. Genocop: a genetic algorithm for numerical optimization problems with linear constraints. *Commun. ACM*, 39(12es):175, 1996.

[65] Z. Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proc. of the Second IEEE Int. Conf. on Evolutionary Computation*, pages 647–651, 1995.

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

[67] H.D. Mittelmann. Benchmarks for optimization software: Ampl-nlp benchmark, 2010. Accessed Online: October 29, 2010. `http://plato.asu.edu/ftp/ampl-nlp.html`.

[68] T.S. Motzkin and I.J. Schoenberg. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:393–404, 1954.

[69] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 13:271–369, 2004.

[70] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinko. A comparison of complete global optimization solvers. *Math. Program.*, 103(2):335–356, 2005.

[71] E. Ozcan and C.K. Mohan. Particle swarm optimization: Surfing the waves. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1939–1944, Mayflower Hotel, Washington D.C., USA, 6-9 1999.

[72] K.E. Parsopoulos and M.N. Vrahatis. Particle swarm optimization method for constrained optimization problems. In *In Proceedings of the Euro-International Symposium on Computational Intelligence 2002*, pages 214–220, 2002.

[73] K.E. Parsopoulos and M.N. Vrahatis. Unified particle swarm optimization for solving constrained engineering optimization problems. *Lecture Notes in Computer Science*, (3612):582–591, 2005.

[74] R. Penrose. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, 51:406–413, 1955.

[75] R.E. Perez and K. Behdinan. Particle swarm approach for structural design optimization. *Comput. Struct.*, 85(19-20):1579–1588, 2007.

[76] D. Powell and M.M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 424–431, San Francisco, CA, USA, 1993.

[77] G.T. Pulido and C.A.C. Coello. A constraint-handling mechanism for particle swarm optimization. *Evolutionary Computation, 2004. CEC2004. Congress on*, 2:1396–1403, 19-23 June 2004.

[78] W.T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, 1983.

[79] M.G.C. Resende and C.C. Ribeiro. *In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics.* Kluwer Academic Publishers, 2003.

[80] R.Y. Rubinstein. *Simulation and Monte Carlo Method.* John Wiley and Sons, New York, 1981.

[81] Kaczmarz S. Angenherte auflsung von systemen linearer gleichungen. *Bulletin International de l'Academie Polonaise des Sciences et des Lettres*, 35(series A):335–357, 1937.

[82] N.V. Sahinidis and M. Tawarmalani. *BARON 7.2.5: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2005. Accessed online 28-May-2009, from `http://www.gams.com/dd/docs/solvers/baron.pdf`.

[83] J.F. Schutte and A.A. Groenwold. A study of global optimization using particle swarms. *Journal of Global Optimization*, 31(1):93–108, 2005.

[84] J.F. Schutte, J.A. Reinbolt, B.J. Fregly, R.T. Haftka, and A.D. George. Parallel global optimization with the particle swarm algorithm. *J. of Numerical Methods in Engineering*, 61:2296–2315, 2003.

[85] J.H. Sendin, J.R. Banga, and T. Csendes. Extensions of a multistart clustering algorithm for constrained global optimization problems. *Industrial & Engineering Chemistry Research*, 48(6):3014–3023, 2009.

[86] A.E. Sepulveda and L. Epstein. The repulsion algorithm, a new multistart method for global optimization. *Structural Optimization*, 11:145–152, 1996.

[87] O. Shcherbina, A. Neumaier, D. Sam-Haroud, X. Vu, and T. Nguyen. Benchmarking global optimization and constraint satisfaction codes. In *Proc. of Global Optimization and Constraint Satisfaction, First Int. Workshop on Global Constraint Optimization and Constraint Satisfaction, COCOS 2002, LNCS2861*, pages 211–222, 2003.

[88] M. Steinbach, L. Ertoz, and V. Kumar. The challenges of clustering high-dimensional data. In *Proc. of New Vistas in Statistical Physics: Applications in Econophysics, Bioinformatics, and Pattern Recognition.* Springer-Verlag, 2003.

[89] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Math. Program.*, 99(3):563–591, 2004.

[90] Y.J. Tjalling. Historical development of the Newton-Raphson method. *SIAM Rev.*, 37(4):531–551, 1995.

[91] A.A. Torn. Cluster analysis using seed points and density-determined hyperspheres as an aid to global optimization. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(8):610–616, Aug. 1977.

[92] T. Tsuchiya and T. Mori. Optimal Conceptual Design of Two-Stage Reusable Rocket Vehicles Including Trajectory Optimization. *Journal of Spacecraft and Rockets*, 41:770–778, September 2004.

[93] Z. Ugray, L. Lasdon, J.C. Plummer, and M. Bussieck. Dynamic filters and randomized drivers for the multi-start global optimization algorithm msnlp. *Optimization Methods Software*, 24:635–656, August 2009.

[94] Z. Ugray, L. Lasdon, J.C. Plummer, F. Glover, J. Kelly, and R. Marti. Scatter search and local nlp solvers: A multistart framework for global optimization. *INFORMS J. Comput.*, 19:328–340, 2007.

[95] D. Vanderbilt and S.G. Louie. A monte carlo simulated annealing approach to optimization over continuous variables. *J. of Computational Physics*, 56:259–271, Nov 1984.

[96] A.I. Vaz and L.N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *J. of Global Optimization*, 39(2):197–219, 2007.

[97] A. Wachter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2002.

[98] A. Wachter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.

[99] R.A. Waltz and T.D. Plantenga. Knitro users manual, 2008. Accessed online 09-Dec-2008, from `http://www.ziena.com`.

[100] R.A. Waltz and T.D. Plantenga. Knitro user's manual: Version 6.0, 2009.

[101] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[102] K.P. Wong and Y.W. Wong. Genetic and genetic/simulated-annealing approaches to economic dispatch. *Generation, Transmission and Distribution, IEE Proc.*, 141(5):507–513, Sep 1994.

[103] B. Yang, Y. Chen, Z. Zhao, and Q. Han. A master-slave particle swarm optimization algorithm for solving constrained optimization problems. *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, 1:3208–3212, 2006.

[104] A.E.M. Zavala, A.H. Aguirre, and E.R.V. Diharce. Constrained optimization via particle evolutionary swarm optimization algorithm (peso). In *GECCO '05: Proc. of the 2005 conference on Genetic and evolutionary computation*, pages 209–216, New York, NY, USA, 2005.

[105] K. Zielinski and R. Laur. Constrained single-objective optimization using particle swarm optimization. In *Proc. of 2006 IEEE Congress on Evolutionary Computation*, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21, 2006.

Appendix A

Models, Problems Sets, and Best Known Objective Function Values

Table A.1: Best Known Objective Function Values for Feasible Solution Vectors to Test Models from the COCONUT Benchmark

Model	Best Known $f(\mathbf{x})$	Problem Sets
aircrfta	0	ID
airport	47952.6996	IA,IB,ICb,ID
aljazzaf	75.0049	ID
alkyl	-1.765	ID,IID
allinitc	30.4748	ID
alsotame	0.082085001	ID
artif	-	IIIA,IB,IICa
batch	259177.9452	ID
bdvalue	0	IIIA,IB,IICa
brainpc0	0.384737744	IIIA,IIB,IICb
brainpc1	0.000164107	IIIA,IIB,IICb
brainpc2	-	IIIA,IIB,IICb
brainpc3	0.000335755	IIIA,IIB,IICb
brainpc4	0.000382985	IIIA,IIB,IICb
brainpc5	0.000356108	IIIA,IIB,IICb
brainpc6	0.000337008	IIIA,IIB,IICb
brainpc7	0.000355281	IIIA,IIB,IICb

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(x)$	Problem Sets
brainpc8	0.000356325	IIIA,IIB,IICb
brainpc9	0.000364115	IIIA,IIB,IICb
bratu2d	-	IICb
bratu2dt	-	IICb
bratu3d	-	IICb
britgas	0	IIA,IB,ICb,ID
broydn3d	-	IIIA,IB,IICb
broydnbd	-	IIIA,IB,IICb
bt1	-1.0001	ID
bt10	-1	ID
bt11	0.824891778	ID
bt12	6.1881	ID
bt13	0	ID
bt2	0.0325682	ID
bt4	-45.5106	ID
bt5	-1047.858	ID
bt6	0.277	ID
bt7	306.4975	ID
bt8	0.999999993	ID
bt9	-1	ID
byrdsphr	-4.683300133	ID
camshape100	-4.284146872	IIA,IB,ICb,ID,IID
camshape200	-4.2785	IIA,IB,ICb,ID,IID
camshape400	-4.275687479	IIA,IB,ICb
camshape800	-4.274273415	IIA,IB,ICb
cantilvr	1.339956363	ID
catena	-23077.7753	IA,IB,ICa,ID
catmix100	-0.0481	IIA,IB,ICb,ID,IID
catmix200	-0.0481	IIA,IB,ICb,ID,IID
catmix400	-	IIA,IB,ICa
catmix800	-0.048055605	IIIA,IB,IICb
cb2	1.9522	ID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
cb3	2	ID
cbratu3d	0	IIIA,IIIB,IICb
chaconn1	1.9522	ID
chaconn2	2	ID
chain100	5.069784611	ID,IID
chain200	5.0689	ID,IID
chain400	5.0686	ID,IID
chain50	5.072261494	ID,IID
chakra	-179.1336	IA,IB,ICb,ID,IID
chance	29.89437817	ID,IID
chandheq	0	IA,IB,ICa,ID
chemrcta	0	IIIA,IB,IICb
chemrctb	0	IIA,IB,ICb
chenery	-1058.9199	IA,IB,ICb,ID,IID
circle	4.5742	ICa,ID,IID
clnlbeam	344.8762185	IIA,IB,ICb
cluster	0	ID
concon	-6373.11	ID
congigmz	28	ID
coolhans	0	ID
core1	91.0562	IA,IB,ICa,ID
core2	72.9	IA,IIIB,ICb,ID
corkscrw	-0.048055605	IIA,IB,ICb,IICb
coshfun	-0.798572812	IA,IIIB,ICb
cresc100	-58.80385274	IIA,IB,ICb
cresc132	-31.38206131	IIIA,IIB,IICb
cresc50	-58.80385274	IA,IB,ICb
csfi2	55.0176	ID
deconvc	0	ID
demo7	-1589042.386	ID,IID
demymalo	-3	ID
dipigri	680.6300749	ID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
disc2	0.766968957	IA,IB,ICa,ID
discs	12.00007501	IA,IB,ICb,ID
dispatch	3155.2879	ID,IID
dittert	-1.9976	IIA,IB,ICa,ID
dixchlng	0	ID
dixchlnv	0	IA,IB,ICb,ID
dnieper	0	IA,IB,ICb,ID
drcav1lq	0	IIIA,IIB,IICb
drcav2lq	90.68782488	IIIA,IB,IICa
drcav3lq	-31.38206131	IIIA,IIB,IICb
dtoc1na	12.70202991	IIA,IB,ICb
dtoc1nb	12.70202991	IIA,IB,ICb
dtoc1nc	15.93777765	IIA,IB,ICb
dtoc1nd	11.95171977	IIA,IB,ICb,ID
dtoc2	19.61999634	IIIA,IB,IICb
dtoc4	-	IIIA,IB,IICb
dtoc5	1.535111532	IIIA,IB,IICb
dtoc6	19.61999634	IIIA,IIIB,IICa
eg3	0	IIA,IB,ICb,ID
eigena2	0	IA,IB,ICb,ID
eigenaco	0	IA,IB,ICb,ID
eigenb2	0	IA,IB,ICb,ID
eigenbco	0	IA,IB,ICa,ID
eigenc2	0	IIA,IB,ICb,ID
eigencco	0	IA,IB,ICb,ID
eigmaxa	-100	IIA,IB,ICb,ID
eigmaxb	-98	IIA,IB,ICb,ID
eigmaxc	-10.7462	IA,IB,ICb,ID
eigmina	-10.74619418	IIA,IB,ICb,ID
eigminb	0.000967435	IIA,IB,ICb,ID
eigminc	-1	IA,IB,ICb,ID
elec100	-1	IA,IB,ICa,ID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(x)$	Problem Sets
elec200	0	IIA,IB,ICb,ID
elec25	243.8127	IA,IB,ICb,ID,IID
elec50	243.8127603	IA,IB,ICb,ID,IID
etamac	-15.2947	ID,IID
ex14_1_1	0	ID,IID
ex14_1_2	0	ID,IID
ex14_1_3	0	ID,IID
ex14_1_4	0	ID,IID
ex14_1_5	0	ID,IID
ex14_1_6	0	IA,IB,ICb,ID,IID
ex14_1_7	0	IA,IB,ICb,ID,IID
ex14_1_8	0	ID,IID
ex14_1_9	0	ID,IID
ex14_2_1	0	ID,IID
ex14_2_2	0	ID,IID
ex14_2_3	0	ID,IID
ex14_2_4	0	ID,IID
ex14_2_5	0	ID,IID
ex14_2_6	0	ID,IID
ex14_2_7	0	ID,IID
ex14_2_8	0	ID,IID
ex14_2_9	0	ID,IID
ex3_1_1	7049.2083	ID,IID
ex3_1_2	-30665.54	ID,IID
ex3_1_3	-310	ID,IID
ex3_1_4	-4	ID,IID
ex4_1_8	-16.7389	ID,IID
ex4_1_9	-5.508013267	ID,IID
ex5_2_2_case1	-400	ID,IID
ex5_2_2_case2	-600	ID,IID
ex5_2_2_case3	-750	ID,IID
ex5_2_4	-450	ID,IID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
ex5_2_5	-3500.0001	IA,IB,ICb,ID,IID
ex5_3_2	1.864159474	ID,IID
ex5_3_3	-3499.999999	IA,IB,ICb,ID,IID
ex5_4_2	7512.2259	ID,IID
ex5_4_3	4845.462	ID,IID
ex5_4_4	10077.7754	ID,IID
ex6_1_1	-0.0202	ID,IID
ex6_1_2	-0.0325	ID,IID
ex6_1_3	-0.3525	ID,IID
ex6_1_4	-0.294541288	ID,IID
ex7_2_1	3.234018235	IA,IB,ICb,ID,IID
ex7_2_10	0.1	ID
ex7_2_2	-0.388811432	ID,IID
ex7_2_3	7049.2181	ID,IID
ex7_2_4	3.918	ID,IID
ex7_2_5	10122.4828	ID
ex7_2_6	-83.2499	ID
ex7_2_7	-5.7399	ID
ex7_2_8	-6.082	ID
ex7_2_9	1.1436	ID
ex7_3_1	0.3417	ID,IID
ex7_3_2	1.089863926	ID,IID
ex7_3_3	0.8175	ID,IID
ex7_3_4	6.2746	ID,IID
ex7_3_5	1.2036	IA,IB,ICb,ID,IID
ex7_3_6	0	ICb,ID,IID
ex8_1_7	0.0293	ID,IID
ex8_1_8	-0.388811432	ID,IID
ex8_2_1	-979.1783	IA,IIIB,ICa,ID,IID
ex8_2_2	-552.6662398	IIIA,IB,IICb
ex8_2_5	-830.3380631	IIIA,IB,IICa
ex8_3_1	-0.819590288	IA,IB,ICb,ID,IID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
ex8_3_10	-1.551113654	IA,IB,ICb,ID,IID
ex8_3_11	-0.799571901	IA,IB,ICb,ID,IID
ex8_3_12	-1.551113654	IA,IB,ICa,ID,IID
ex8_3_2	-0.412330163	IA,IB,ICb,ID,IID
ex8_3_3	-0.819590288	IA,IB,ICb,ID,IID
ex8_3_4	-3.58	IA,IB,ICb,ID,IID
ex8_3_5	-0.416603062	IA,IB,ICb,ID,IID
ex8_3_6	-3.579982361	IA,IB,ICb,ID,IID
ex8_3_7	-1.232619169	IA,IB,ICb,ID,IID
ex8_3_8	-3.25611907	IA,IB,ICb,ID,IID
ex8_3_9	-1.232619169	IA,IB,ICa,ID,IID
ex8_4_1	0.618572759	ICb,ID,IID
ex8_4_2	0.485152487	ICa,ID,IID
ex8_4_3	-3.25611907	IA,IB,ICa,ID,IID
ex8_4_4	0.212459839	IA,IB,ICb,ID,IID
ex8_4_5	0.0003	IA,IB,ICb,ID,IID
ex8_4_6	0	ID,IID
ex8_4_7	28.898	IA,IB,ICb,ID,IID
ex8_4_8	3.3218	IA,IIIB,ICb,ID,IID
ex8_5_1	-4.07E-07	ID,IID
ex8_5_2	-0.0002	ID,IID
ex8_5_3	-0.0042	ID,IID
ex8_5_4	-0.000425147	ID,IID
ex8_5_5	-0.0108	ID,IID
ex8_5_6	-0.0012	ID,IID
ex9_1_1	-13	ID,IID
ex9_1_10	-3.25	ID,IID
ex9_1_2	-16	ID,IID
ex9_1_4	-37	ID,IID
ex9_1_5	-1	ID,IID
ex9_1_8	-3.25	ID,IID
ex9_2_1	17	ID,IID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
ex9_2_2	99.9995	ID,IID
ex9_2_3	0	ID,IID
ex9_2_4	0.5	ID,IID
ex9_2_5	5	ID,IID
ex9_2_6	-1	ID,IID
ex9_2_7	17	ID,IID
fletcher	11.6568	ID
flowchan100	0.000307486	IIA,IB,ICb
flowchan100_ed	0	IIA,IB,ICb
flowchan200	29.04730672	IIA,IB,ICb
flowchan400	0	IIIA,IB,IICb
flowchan50	-	IIA,IB,ICb
gasoil100	0	IIA,IB,ICb
gasoil200	-552.6662398	IIIA,IB,IICb
gasoil400	-830.3380631	IIIA,IIB,IICb
gasoil50	0.00523664	IIA,IB,ICb
gausselm	-17.89542494	IIIA,IB,IICb
gigomez1	-3	ID
glider100	-1255.058601	IIIA,IB,IICb
glider200	-1247.075526	IIIA,IB,IICb
glider400	-1247.725493	IIIA,IB,IICb
glider50	-1286.977373	IIA,IB,ICb,ID
gottfr	0	ID
gpp	14400.9271	IIA,IIIB,ICb,ID
hadamard	0.005236594	IIA,IB,ICb,ID
haifam	-45.0004	IIA,IB,ICb,ID
haifas	-0.45	ID
haldmads	-1256.460375	IA,IB,ICb,ID
hanging	-620.1761	IIA,IB,ICa,ID
hatfldf	0	ID
hatfldg	0	IA,IB,ICb,ID
haverly	-400	ID,IID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
heart6	0	ID
heart8	0	ID
hhfair	-87.15903761	ID,IID
himmel11	-30665.54	ID,IID
himmel16	-45.00036038	IA,IB,ICb,ID,IID
himmelbc	0	ID
himmelbd	0	ID
himmelbk	0.000122386	IA,IB,ICb,ID
himmelp2	-62.0539	ID
himmelp3	-59.0131235	ID
himmelp4	-59.0131235	ID
himmelp5	-59.0131235	ID
himmelp6	-59.0131235	ID
house	-4500	ID,IID
hs006	0	ID
hs007	-1.7321	ID
hs008	-1	ID
hs010	-1	ID
hs011	-8.4985	ID
hs012	-30	ID
hs013	0.9801	ID
hs014	1.393464983	ID
hs015	306.5	ID
hs016	0.25	ID
hs017	1	ID
hs018	5	ID
hs019	-6961.8176	ID
hs020	38.1987	ID
hs022	1	ID
hs023	2	ID
hs026	0	ID
hs027	0.04	ID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
hs029	-22.627417	ID
hs030	1	ID
hs031	6	ID
hs032	1	ID
hs033	-4.5858	ID
hs034	-0.834032438	ID
hs039	-1	ID
hs040	-0.25	ID
hs042	13.85786438	ID
hs043	-44	ID
hs046	0	ID
hs047	-0.026714183	ID
hs056	-3.456	ID
hs057	0.030647619	ID
hs059	-7.8028	ID
hs060	0.0325682	ID
hs061	-143.6461422	ID
hs063	-1038.285	ID
hs064	6299.8375	ID
hs065	0.9535	ID
hs066	0.518163278	ID
hs070	0.0089	ID
hs071	17.014	ID
hs072	727.5418	ID
hs073	29.89437817	ID
hs074	5126.498	ID
hs075	5174.41	ID
hs077	0.2415	ID
hs078	-2.919700409	ID
hs079	0.078776821	ID
hs080	0.0539	ID
hs081	0.0539	ID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
hs083	-30665.54	ID
hs084	-5280335.133	ID
hs085	-1.905155235	IA,IB,ICb,ID
hs088	1.3618	ID
hs089	1.3617	ID
hs090	1.3625	ID
hs091	1.3626	ID
hs092	1.3625	ID
hs093	135.0759638	ID
hs095	0.0156	ID
hs096	0.0156	ID
hs097	3.1358	ID
hs098	3.1358	ID
hs099	-831079900	IA,IB,ICb,ID
hs100	680.6300749	ID
hs100lnp	680.6300572	ID
hs100mod	678.7547	ID
hs101	1809.7583	ID
hs102	911.8772	ID
hs103	543.6669	ID
hs104	3.9511	ID
hs106	7049.2083	ID
hs107	5055.0082	ID
hs108	-0.866025403	IA,IB,ICb,ID
hs109	5326.8513	ID
hs111	-47.7611	ID
hs111lnp	-47.7611	ID
hs113	24.3062	ID
hs114	-1768.8074	ID
hs116	97.5875	ICb,ID
hs117	32.3487	ID
hs99exp	-1008062500	IA,IB,ICb,ID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(x)$	Problem Sets
hvycrash	-1.905155235	IIA,IB,ICb,ID
hydro	4366944	ID,IID
hypcir	0	ID
integreq	-831079891.5	IA,IB,ICa,ID
kissing	-0.866025403	IIA,IB,ICb,ID
kiwcresc	0	ID
korcge	-339.2130282	IA,IIIB,ICb,ID,IID
lakes	-1008062500	IA,IB,ICb,ID
launch	-0.2185	IA,IB,ICa,ID,IID
lch	-4.3183	ID
lewispol	0	ID
lnts100	0.554595401	IIA,IB,ICb,ID,IID
lnts200	0.554577016	IIA,IB,ICb
lnts400	-1255.058601	IIIA,IIB,IICa
lnts50	-339.213028	IIA,IB,ICb,ID,IID
lootsma	1.4142	ID
madsen	0.6164	ID
madsschj	-797.2837027	IIA,IB,ICb,ID
makela1	-1.414213557	ID
makela2	7.2	ID
makela3	0	IA,IB,ICb,ID
manne	-0.974168304	IIA,IB,ICb
maratos	-1	ID
matrix2	0	ID
mconcon	-6373.11	ID
methanol100	0.009022309	IIA,IB,ICb
methanol200	-1247.075526	IIIA,IIB,IICb
methanol400	-1247.725493	IIIA,IIB,IICb
methanol50	0.009022298	IIA,IB,ICb
mhw4d	0.0293	ID,IID
mifflin1	-1	ID
mifflin2	-1	ID

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(x)$	Problem Sets
minc44	-797.2837027	IIA,IB,ICb,ID
minlphi	582.2361	ID,IID
minmaxbd	4.87E-08	IA,IB,ICa,ID
minmaxrb	0	ID
minperm	0.00036288	IIIA,IB,IICb
mistake	-1	IA,IB,ICb,ID
msqrta	0	IIIA,IB,IICa
msqrtb	0	IIIA,IB,IICb
mwright	1.288382901	ID
ngone	-0.642085182	IIIA,IB,IICb,ID
nuffield	-0.642085182	IIIA,IIB,IICb
nuffield2	0	IIIA,IIB,IICb
nuffield2_trap	0	IIIA,IB,IICb
oet2	0.087159641	IIIA,IB,IICb,ID
optcdeg2	0.009022309	IIA,IB,ICb
optcdeg3	0.009022298	IIA,IB,ICb
optcntrl	0.002573029	IA,IB,ICb,ID
optctrl3	2048.0165	IA,IB,ICb,ID
optctrl6	-0.999999997	IA,IB,ICb,ID
optmass	-0.189542471	IA,IB,ICb,ID
optprloc	-16.4198	IA,IB,ICa,ID
orthrdm2	5.195043992	IIIA,IIB,IICb
orthrds2	37.41153983	IA,IB,ICb,ID
orthrega	1414.055887	IIA,IB,ICb,ID
orthregb	0	ID
orthregc	-	IIIA,IIB,IICa
orthregd	0.087159641	IIIA,IIB,IICb
orthrege	0.4509	IA,IB,ICb,ID
orthrgdm	19057.98593	IIIA,IIB,IICb
orthrgds	-	IIIA,IIB,IICa
otpop	-0.189542471	IA,IB,ICb,ID,IID
pinene100	19.87217015	IIIA,IB,IICb

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
pinene200	27045.7008	IIIA,IB,IICb
pinene25	-	IIA,IB,ICa
pinene50	19.87216552	IIA,IB,ICb
polak4	0	ID
polak5	50	ID
polak6	-44	ID
polygon100	-0.726868387	IIIA,IB,IICa,ID,IID
polygon25	-0.7722	IIA,IB,ICb,ID,IID
polygon50	-0.7757	IIIA,IB,IICb,ID,IID
polygon75	-0.72686841	IIIA,IB,IICb
popdynm100	-0.726868387	IIIA,IB,IICa
popdynm200	-0.726868345	IIIA,IB,IICa
popdynm25	2.750189447	IIA,IB,ICb
popdynm50	2.97E-19	IIA,IB,ICb
porous1	-0.72686841	IIIA,IB,IICb
porous2	19746528602	IIIA,IB,IICa
powellsq	0	ID
process	-1161.336602	ID,IID
prodpl0	60.9192	ID
prodpl1	53.037	ID
prolog	0	ID,IID
ramsey	-2.4875	IA,IB,ICb,ID,IID
reading1	-	IIIA,IB,IICa
reading3	-1.72E-15	IIA,IB,ICb,ID
rk23	0.0833	ID
robot	5.4628	ID
robot100	-0.752581181	IIA,IB,ICb
robot200	9.141396586	IIIA,IB,IICb
robot400	9.141030382	IIIA,IB,IICa
robot50	9.146878638	IIA,IB,ICb,ID,IID
rocket100	-1.012831898	IIA,IB,ICb,ID,IID
rocket200	-1.012835436	IIIA,IB,IICb

Continued on Next Page...

Table A.1 – Continued

Model	Best Known $f(\mathbf{x})$	Problem Sets
rocket400	-1.012836105	IIIA,IIB,IICb
rosenmmx	-44	ID
s332	-2.487468633	IA,IB,ICb
s365mod	52.1399	ID
sample	726.6367	ID,IID
sawpath	-181.573	IIA,IB,ICb,ID
semicon1	0	IIA,IB,ICb,IICb
semicon2	-1.012835436	IIA,IB,ICb,IICb
sinrosnb	-99901	IIA,IB,ICb
smmpsf	29.92435035	IA,IB,ICb,ID
snake	-0.0085	ID
spiral	0	ID
sreadin3	-1.012836105	IIIA,IB,IICb
ssebnln	181.5729437	IA,IB,ICa,ID
ssnlbeam	337.7724	ICb,ID
svanberg	0	IIIA,IB,IICa
swopf	0	IA,IB,ICb,ID
trainf	0	IIIA,IB,IICb
trainh	-	IIIA,IB,IICb
try-b	0	ID
twirism1	-99901	IIA,IB,ICb,ID
twobars	1.50865242	ID
ubh5	8361.422768	IIIA,IB,IICb
vanderm1	0	IA,IB,ICb,ID
vanderm2	-1.009357402	IA,IIIB,ICb,ID
vanderm3	0	IA,IIIB,ICb,ID
vanderm4	0	ID
wall	-1.000004662	ID,IID
water	0	IA,IB,ICa,ID,IID
womflet	0	ID
zecevic3	97.3094	ID
zecevic4	7.5575	ID

Continued on Next Page...

Table A.1 – Continued

Model	**Best Known $f(\mathbf{x})$**	**Problem Sets**
zigzag	1.8	ICb,ID
zy2	2	ID