NOVEL EVOLUTIONARY GLOBAL OPTIMIZATION ALGORITHMS AND THEIR APPLICATIONS

Hualin Wang
Purdue University

Okan Ersoy
Purdue University

Follow this and additional works at: http://docs.lib.purdue.edu/ecetr

http://docs.lib.purdue.edu/ecetr/340

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
NOVEL EVOLUTIONARY GLOBAL OPTIMIZATION ALGORITHMS AND THEIR APPLICATIONS

A Technical Report
Submitted to the ECE Department
of
Purdue University
by
Hualin Wang and Okan Ersoy

School of Electrical and Computer Engineering
465 Northwestern Ave.
Purdue University
West Lafayette, IN 47907

December 2006
TABLE OF CONTENTS

LIST OF TABLES ............................................................ vi
LIST OF FIGURES .............................................................. viii
ABSTRACT ................................................................. xi
1 INTRODUCTION ........................................................... 1
   1.1 Motivations ......................................................... 1
   1.2 Research Objectives ............................................... 2
   1.3 Organization of This Report ..................................... 3
2 OVERVIEWS OF OPTIMIZATION AND EVOLUTIONARY ALGORITHMS 5
   2.1 Optimization .......................................................... 5
   2.2 Overview of Some Selected Global Optimization Techniques . . . . 6
      2.2.1 Monte Carlo method ......................................... 7
      2.2.2 Hill Climbing ................................................. 7
      2.2.3 Simulated Annealing ....................................... 8
   2.3 Evolutionary Algorithms .......................................... 10
      2.3.1 Genetic Algorithms ......................................... 12
      2.3.2 Evolutionary Programming ................................. 13
      2.3.3 Evolutionary Strategies ................................... 13
      2.3.4 Disadvantages and Advantages of Evolutionary Algorithms . 14
   2.4 Summary ............................................................. 15
3 GRAY CODE OPTIMIZATION ALGORITHMS ................................. 16
   3.1 Introduction of the Gray Code .................................. 16
   3.2 The Gray Code Optimization Algorithm ........................ 17
      3.2.1 Discussion of the GCO Algorithm ......................... 20
      3.2.2 Experimental Results with the GCO Algorithm .......... 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Hybrid GCO Algorithm</td>
<td>21</td>
</tr>
<tr>
<td>3.3.1</td>
<td>The Nelder-Mead Simplex Method</td>
<td>22</td>
</tr>
<tr>
<td>3.3.2</td>
<td>The HGCO Algorithm</td>
<td>22</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Experimental Results of the HGCO Algorithm</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>Parallel Gray Code Optimization Algorithm</td>
<td>24</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Introduction</td>
<td>24</td>
</tr>
<tr>
<td>3.4.2</td>
<td>The PGCO Algorithm</td>
<td>25</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Experimental Results</td>
<td>26</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Scalability of the PGCO Algorithm</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>MIXTURE GAUSSIAN OPTIMIZATION ALGORITHM</td>
<td>30</td>
</tr>
<tr>
<td>4.1</td>
<td>Using the EM Algorithm to Model GCO Search</td>
<td>30</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Expectation Maximization (EM) Algorithm</td>
<td>33</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Modeling of Histogram Data Sets with the EM Algorithm</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Mixture Gaussian Optimization (MGO) Algorithm</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Experimental Results</td>
<td>36</td>
</tr>
<tr>
<td>4.4</td>
<td>Crossover with the MGO Algorithm</td>
<td>38</td>
</tr>
<tr>
<td>4.5</td>
<td>Convergence of the MGO Algorithm</td>
<td>43</td>
</tr>
<tr>
<td>4.6</td>
<td>Progress Rate on the Sphere Function</td>
<td>46</td>
</tr>
<tr>
<td>4.6.1</td>
<td>The Progress Rate</td>
<td>46</td>
</tr>
<tr>
<td>4.6.2</td>
<td>The Progress Rate for Sphere function</td>
<td>46</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Simulation Results</td>
<td>47</td>
</tr>
<tr>
<td>4.6.4</td>
<td>Discussions</td>
<td>48</td>
</tr>
<tr>
<td>4.6.5</td>
<td>Progress Rate for the Sphere Function - Multiple Children Case</td>
<td>52</td>
</tr>
<tr>
<td>4.6.6</td>
<td>Experimental Results</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>Summary</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>APPLICATION IN MOLECULE CONFORMATION SEARCH</td>
<td>61</td>
</tr>
<tr>
<td>5.1</td>
<td>Molecular Energy Model</td>
<td>61</td>
</tr>
</tbody>
</table>
5.2 Implementation Details ........................................... 62
5.3 Test Problems ....................................................... 63
5.4 Experimental Results .............................................. 65
5.5 Summary ............................................................. 68

6 APPLICATIONS IN NONLINEAR LEAST SQUARE TYPE PROBLEMS . 69
6.1 Curve Fitting with Radial Basis Functions ......................... 69
   6.1.1 Experimental Results ........................................ 70
6.2 Managing the EM Algorithm ....................................... 71
   6.2.1 Introduction .................................................. 74
   6.2.2 Experimental Results ........................................ 75
6.3 Non-uniform Spectral Analysis ..................................... 75
   6.3.1 Introduction .................................................. 75
   6.3.2 Determination of the Significant Frequency Components in the
        Signal ............................................................. 80
   6.3.3 Experimental Results ........................................ 81
6.4 Summary ............................................................. 91

7 AN OPEN STRUCTURE GLOBAL OPTIMIZER ...................... 92
7.1 Strengths and Weaknesses of Evolutionary Algorithms .......... 92
   7.1.1 Introduction to the CMAES Algorithm ..................... 93
   7.1.2 Comparison of the MGO, GA, and CMAES Algorithms ...... 94
7.2 An Open Structure Global Optimizer ................................ 103
   7.2.1 The Algorithm and Its Structure .......................... 103
   7.2.2 The Parallel Environment ................................. 106
   7.2.3 Experimental Results ....................................... 106
7.3 Summary ............................................................. 107

8 SUMMARY AND FUTURE WORK ........................................ 112
8.1 The Main Contributions ............................................ 112
8.2 Gray Code Optimization Algorithm ................................ 113
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3 Mixture Gaussians Optimization Algorithm</td>
<td>114</td>
</tr>
<tr>
<td>8.4 Applications</td>
<td>115</td>
</tr>
<tr>
<td>8.5 An Open Structure Global Optimizer</td>
<td>115</td>
</tr>
<tr>
<td>8.6 Future Research</td>
<td>116</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>117</td>
</tr>
<tr>
<td>A TEST FUNCTIONS</td>
<td>120</td>
</tr>
<tr>
<td>B DATA USED TO APPROXIMATE THE GCO ALGORITHM WITH THE EM ALGORITHM</td>
<td>122</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>68</td>
</tr>
<tr>
<td>6.1</td>
<td>91</td>
</tr>
<tr>
<td>7.1</td>
<td>95</td>
</tr>
<tr>
<td>7.2</td>
<td>96</td>
</tr>
<tr>
<td>7.3</td>
<td>97</td>
</tr>
<tr>
<td>7.4</td>
<td>98</td>
</tr>
<tr>
<td>7.5</td>
<td>99</td>
</tr>
<tr>
<td>7.6</td>
<td>100</td>
</tr>
<tr>
<td>7.7</td>
<td>101</td>
</tr>
<tr>
<td>7.8</td>
<td>102</td>
</tr>
<tr>
<td>7.9</td>
<td>107</td>
</tr>
<tr>
<td>7.10</td>
<td>108</td>
</tr>
<tr>
<td>7.11</td>
<td>108</td>
</tr>
<tr>
<td>7.12</td>
<td>108</td>
</tr>
<tr>
<td>7.13</td>
<td>109</td>
</tr>
<tr>
<td>7.14</td>
<td>109</td>
</tr>
<tr>
<td>7.15</td>
<td>109</td>
</tr>
<tr>
<td>7.16</td>
<td>110</td>
</tr>
<tr>
<td>B.1</td>
<td>122</td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>B.2</td>
<td>123</td>
</tr>
<tr>
<td>Data Set 2, the initial point is 0.5226</td>
<td></td>
</tr>
<tr>
<td>B.3</td>
<td>124</td>
</tr>
<tr>
<td>Data Set 3, the initial point is 0.4449</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>2.1</td>
<td>Schematic of Basic Evolutionary Algorithms.</td>
</tr>
<tr>
<td>2.2</td>
<td>Crossover and Mutation Operations.</td>
</tr>
<tr>
<td>3.1</td>
<td>Generation of a Population from a Single Parent in the GCO Algorithm.</td>
</tr>
<tr>
<td>3.2</td>
<td>Speed up Factor.</td>
</tr>
<tr>
<td>4.1</td>
<td>Histogram Generated with the Initial Point Equals to 0.6946.</td>
</tr>
<tr>
<td>4.2</td>
<td>Histogram Generated with the Initial Point Equals to 0.5226.</td>
</tr>
<tr>
<td>4.3</td>
<td>Histogram Generated with the Initial Point Equals to 0.4449.</td>
</tr>
<tr>
<td>4.4</td>
<td>Modeling of the Histogram Data in Fig. 4.1.</td>
</tr>
<tr>
<td>4.5</td>
<td>Modeling of the Histogram Data in Fig. 4.2.</td>
</tr>
<tr>
<td>4.6</td>
<td>Modeling of the Histogram Data in Fig. 4.3.</td>
</tr>
<tr>
<td>4.7</td>
<td>Convergence of the MGO Algorithm with No Crossover and Small Population Size.</td>
</tr>
<tr>
<td>4.8</td>
<td>Convergence of the MGO Algorithm with No Crossover and Small Population Size.</td>
</tr>
<tr>
<td>4.9</td>
<td>Convergence of the MGO Algorithm with Crossover and Small Population Size.</td>
</tr>
<tr>
<td>4.10</td>
<td>Convergence of the MGO Algorithm with Crossover and Small Population Size.</td>
</tr>
<tr>
<td>4.11</td>
<td>Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of Different Initial Points.</td>
</tr>
<tr>
<td>4.12</td>
<td>Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of Different Coefficient Combinations.</td>
</tr>
<tr>
<td>4.13</td>
<td>Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of the Distance between Gaussian Components.</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>4.14 Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of the Standard Deviations.</td>
<td>51</td>
</tr>
<tr>
<td>4.15 The PDF of the Best Children as the Function of Different $\lambda$.</td>
<td>53</td>
</tr>
<tr>
<td>4.16 Progress Rate as a Function of Different Population Size.</td>
<td>54</td>
</tr>
<tr>
<td>4.17 Convergence Curve for the Mixture Gaussian Model and the Single Gaussian Model, Initial Point = 100.</td>
<td>56</td>
</tr>
<tr>
<td>4.18 Convergence Curve for the Mixture Gaussian Model and the Single Gaussian Model, Initial Point = 1000.</td>
<td>57</td>
</tr>
<tr>
<td>4.19 Convergence Curve for the Mixture Gaussian Model and the Single Gaussian Model, Initial Point = 1500.</td>
<td>58</td>
</tr>
<tr>
<td>4.20 Convergence Curve of the Single Gaussian Model as A Function of the Population Size.</td>
<td>59</td>
</tr>
<tr>
<td>4.21 Convergence Curve of the Mixture Gaussian Model as A Function of the Population Size.</td>
<td>60</td>
</tr>
<tr>
<td>5.1 Global Minimal Energy Configuration of $CH_3(CH_2)_2CH_3$.</td>
<td>64</td>
</tr>
<tr>
<td>5.2 Local Minimal Energy Configuration of $CH_3(CH_2)_2CH_3$.</td>
<td>64</td>
</tr>
<tr>
<td>5.3 Global Minimal Energy Configuration of $CH_3(CH_2)_3CH_3$.</td>
<td>65</td>
</tr>
<tr>
<td>5.4 Local Minimal Energy Configuration of $CH_3(CH_2)_3CH_3$.</td>
<td>65</td>
</tr>
<tr>
<td>5.5 Local Minimal Energy Configuration of $CH_3(CH_2)_3CH_3$.</td>
<td>66</td>
</tr>
<tr>
<td>5.6 Local Minimal Energy Configuration of $CH_3(CH_2)_3CH_3$.</td>
<td>66</td>
</tr>
<tr>
<td>5.7 Global Minimal Energy Configuration of $CH_3(CH_2)_{11}CH_3$.</td>
<td>67</td>
</tr>
<tr>
<td>6.1 Fminsearch, Error 17.74, Elapsed Time 34 Seconds.</td>
<td>71</td>
</tr>
<tr>
<td>6.2 MGO, Error 0.15, Elapsed Time 12 Seconds.</td>
<td>72</td>
</tr>
<tr>
<td>6.3 Fminsearch, Error 61.42, Elapsed Time 212 Seconds.</td>
<td>72</td>
</tr>
<tr>
<td>6.4 MGO, Error 0.97, Elapsed Time 25 Seconds.</td>
<td>73</td>
</tr>
<tr>
<td>6.5 Pure EM, Init = [0.57 0.27 0.23 0.99 0.04 0.61 0.82 0.62 0.77].</td>
<td>76</td>
</tr>
<tr>
<td>6.6 MGOEM, Init = [0.57 0.27 0.23 0.99 0.04 0.61 0.82 0.62 0.77].</td>
<td>77</td>
</tr>
<tr>
<td>6.7 Pure EM, Init = [0.01 1.33 0.91 1.42 1.66 0.98 1.33 0.29 1.49].</td>
<td>78</td>
</tr>
<tr>
<td>6.8 MGOEM, Init = [0.01 1.33 0.91 1.42 1.66 0.98 1.33 0.29 1.49].</td>
<td>79</td>
</tr>
<tr>
<td>6.9 Signal with Two Frequency Components, $f_1 = 20Hz$, $f_2 = 30Hz$.</td>
<td>81</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>6.10 Power Spectrum Using the FFT after Interpolation.</td>
<td>82</td>
</tr>
<tr>
<td>6.11 The Periodogram Algorithm, $f_1 = 20Hz$, $f_2 = 21Hz$, 100 Data Points.</td>
<td>84</td>
</tr>
<tr>
<td>6.12 The Periodogram Algorithm, $f_1 = 20Hz$, $f_2 = 21Hz$, 50 Data Points.</td>
<td>85</td>
</tr>
<tr>
<td>6.13 The Burg Algorithm, $f_1 = 20Hz$, $f_2 = 21Hz$, 50 Data Points.</td>
<td>86</td>
</tr>
<tr>
<td>6.14 The Burg Algorithm, $f_1 = 20Hz$, $f_2 = 21Hz$, 33 Data Points.</td>
<td>87</td>
</tr>
<tr>
<td>6.15 The Peig Algorithm, $f_1 = 20Hz$, $f_2 = 21Hz$, 33 Data Points, Regular Sampling.</td>
<td>88</td>
</tr>
<tr>
<td>6.16 The Peig Algorithm, $f_1 = 20Hz$, $f_2 = 21Hz$, 33 Data Points, Irregular Sampling.</td>
<td>89</td>
</tr>
<tr>
<td>6.17 The MGO Algorithm, $f_1 = 20Hz$, $f_2 = 21Hz$, 33 Data Points, Irregular Sampling.</td>
<td>90</td>
</tr>
<tr>
<td>7.1 The Structure of the Algorithm.</td>
<td>104</td>
</tr>
<tr>
<td>7.2 The Matlab Parallel Computing Toolbox.</td>
<td>106</td>
</tr>
</tbody>
</table>
ABSTRACT

Wang, Hualin, Ersoy, Okan, Purdue University. Novel Evolutionary Global Optimization Algorithms and Their Applications.

The gray code optimization (GCO) algorithm is a deterministic global optimization algorithm based on integer representation. It utilizes the adjacency property of Gray code representation. By controlling the number of bits flipped, it searches through the space efficiently. A further development of the GCO algorithm is conducted in this research to avoid getting stuck in local minima. To further improve the performance, and take the advantage of cheaper but more powerful CPUs, a parallel computation paradigm using MPI is implemented.

Analysis of the mechanism of the GCO algorithm indicated that it can be modeled by mixture gaussian. This led to a new stochastic evolutionary global optimization algorithm based on mixture of gaussians and real numbers. The EM algorithm is used to acquire the parameters of each Gaussian component. With a mathematic model in hand, a lot of theoretical questions, such as convergence property, convergence rate, and the benefits of using the mixture model could be investigated. The relationship between the proposed algorithms and other evolutionary algorithms including genetic algorithms, evolutionary programming and evolutionary strategy will be studied. To combine the merits of different evolutionary algorithms, a uniform global optimizer based on parallel computing was proposed to solve a broad range of problems.

The proposed algorithms are general global optimization methods. They have a broad range of applications in engineering and science. The applications in molecular conformation search, curve fitting, and spectral analysis are reported in this report.
1. INTRODUCTION

1.1 Motivations

Many practical problems in science or engineering can be formulated as optimization problems. For example, structures of materials are decided by minimum energy state principle. Circuit designers try to minimize the lengths of wires in electrical devices. Manufacturers try to find the best schedule to yield the maximum number of products. Mail delvers try to find the best routine to deliver all the mails with the shortest path and minimum time.

Unfortunately, most of the optimization problems turn out to be global optimization problems. A unique global minimum is emerged among many local minima. Until now, there is no general global optimization problem solver which is optimal for all problems. In the last two decades, inspired by the Darwinian principle of the survival of fittest, Evolutionary Algorithms (EAs) have gained more and more ground in the global optimization area. According to Fogel [1], the advantages of evolutionary algorithms over traditional optimization algorithms include 1) conceptual simplicity, 2) broad applicability, 3) outperform classical methods on real problems, 4) potential to use knowledge and to hybridize with other methods, 5) easy to parallelize, 6) robust to dynamic changes and 7) capability of self-optimization.

Although the flexibility of EAs, when applied to specific problems, there are still a lot of magic parameters which need to be tuned. For example, in the genetic algorithm (GA), the crossover rate, the mutation probability, and the population size need to be decided beforehand. Due to a lack of theoretic guide on how to choose those parameters, most of the time, the best set of parameters can only be found after many trials.
Evolutionary algorithms are population based algorithms. During the search process, not a single potential solution is kept, but many potential solutions are kept. For current evolutionary algorithms, the popular population size ranges from 20 to 500. However, for very high dimensional problems, the pre-determined population size either leads to pre-mature sub-optimal points or leads to very slow convergence.

The CPUs become cheaper, and more and more powerful. To tackle the slow convergence of EAs on high dimensional problems, a parallel computing paradigm is very attractive.

The Gray code optimization (GCO) [2] is an interesting and efficient global optimization algorithm. It can be roughly classified into evolutionary algorithms sharing similarity with genetic algorithms (GA) and evolutionary programming (EP). However, it could be stuck on local minima sometimes. Its deterministic property sometimes also prevents it from reaching the global optimal point or a very deep minimum.

In the end of the search, due to the stochastic nature of most evolutionary algorithms, the best solution wanders around the true minimum. Usually, it will take a lot of time to get very close to the true minimum. At this stage, a local search method maybe much more efficient and accurate.

1.2 Research Objectives

Since the Gray code optimization algorithm is a simple, efficient algorithm, a lot of research in this report is dedicated to improve it as much as possible. To avoid the local minima, a multiple initial point strategy is used in the early stage of the search. To speed up the convergence in the late search, a hybrid algorithm combining the GCO algorithm with some local search methods were proposed. To take advantage of more and more accessible computers, a parallel GCO computing paradigm was proposed. To test the parallel algorithm, a small cluster with different CPU speeds was built.
Analysis of the mechanism of the GCO algorithm indicated that it can be modeled by a mixture of gaussians. This led to a new stochastic evolutionary global optimization algorithm based on mixture of gaussians and real numbers. The EM algorithm is used to estimate the parameters of each gaussian component. With a mathematic model in hand, a lot of theoretical questions, such as convergence property, progress rate, and the benefits of using the mixture gaussian model could be investigated.

Different evolutionary algorithms have their own strengths and weaknesses. No single algorithm is the clear best choice. To solve a broad range of problems, a uniform global optimizer was proposed. It has an open structure. Different algorithms can work together using a parallel scheme.

Conformation search is an important problem in computational chemistry. Flexible molecules have many rotatable bonds. Under different conditions, they can assume different structures. The problem of finding the global minimal energy conformation is a difficult optimization problem. The improved GCO algorithm was used to locate the best molecule conformation.

A large set of engineering and scientific problems can be represented as non-linear least square problems. The proposed algorithm was used to solve problems in curve fitting, parameter estimation, and spectral analysis.

1.3 Organization of This Report

This report is organized as follows. Chapter 2 gives a brief review of general optimization techniques and a background of evolutionary algorithms. Several global optimization methods are discussed. Chapter 3 discusses all the improvements made on the Gray code optimization algorithm, including multiple starting points in the early stage, hybridization with local optimization algorithm in the end of search and a parallel implementation. Chapter 4 presents a new global evolutionary algorithm based on mixture gaussian model, which is inspired by the GCO algorithm. Some theoretic issues related to the new algorithm are investigated. Molecule conforma-
tional search with the new algorithm is presented in chapter 5. Chapter 6 covers some interesting applications in least square type problems. In chapter 7, we propose an open structure global optimizer which takes the advantage of different evolutionary algorithms. Chapter 8 is the summary and discussion of possible future research.
2. OVERVIEWS OF OPTIMIZATION AND EVOLUTIONARY ALGORITHMS

The first part of this chapter covers an overview of optimization techniques. More attention is paid on global optimization. Several global optimization algorithms are presented. The second part of the chapter provides an overview of the evolutionary algorithms. Three paradigms including genetic algorithms (GAs), evolutionary strategies (ESs), and evolutionary programming (EP) are presented.

2.1 Optimization

Mathematically speaking, optimization is the minimization or maximization of a given function subject to constraints on its variables.

**Definition 2.1.1** Let $S$ be the search space, $f(s): S \rightarrow \mathbb{R}$ be the objective function, and $g_i(s): S \rightarrow \mathbb{R}$ be a set of constraint functions. The optimization problem is then given as

$$
\text{minimize } f(s) \\
g_i(s) \geq 0, \quad \forall i \in (1, 2, 3, \ldots, q), s \in S
$$

Here, the optimization problem is formulated as a minimization task. As well known, a maximization problem could be easily transformed into a minimization problem: maximize $f(s) = - \text{minimize } f(s)$

**Definition 2.1.2** A point $s^*$ is called a global minimum if

$$
f(s^*) \leq f(s), \quad \forall s \in S
$$
**Definition 2.1.3**  A point $s^*$ is called a local minimum if there is a neighborhood $N$ of $s^*$ such that

$$f(s^*) \leq f(s), \quad \forall s \in N$$

Generally, the global minimum point is sought after rather than the local minima. However, it is usually very difficult to identify and locate. Methods of searching the local minima have gained a lot of attention in both theoretical and numerical studies. Many elegant local search algorithms have been developed so far, such as the line search algorithm, the steepest descent algorithm, the simplex algorithm, the conjugate gradient algorithm, the Newton algorithm and so on. On the contrary, there is still no global optimization solver which is efficient enough with all problems.

There is no universal optimization algorithm than can solve all kinds of optimization problems. However, there are numerous algorithms to choose from, and they can be tailored to specific problems at hand. According to [3] [4], some important properties of a good optimization algorithm should have the following merits:

1. **Accuracy.** The algorithm should be able to find a solution close enough to the optimal point within acceptable tolerance.

2. **Efficiency and time complexity.** The algorithm should be able to find an accurate solution within an acceptable time period. An enumeration algorithm or exhaustive algorithm is guaranteed to find the global minimum, but in practice, it maybe useless.

3. **Robustness.** The algorithm should be able to perform well on a variety of problems in their class. At the same time, the algorithm should not be very sensitive to the initial point.

**2.2 Overview of Some Selected Global Optimization Techniques**

In this section, several global optimization algorithms are briefly introduced. The focus lies on those algorithms which do not have too much restriction on the search
space and the object function. For example, no derivative or gradient of the object function is needed, and the constraints on the search space are usually loose.

2.2.1 Monte Carlo method

Monte Carlo algorithm is a random search algorithm. It samples the search space by some fixed distribution. It progresses by memorizing the best point found so far. The main drawback of this method is that it requires the information of a prior distribution. In most cases, there is no any information available. The following is a sketch of the algorithm.

\begin{itemize}
  \item \textbf{Input:} Object function \( f : S \rightarrow R \)
  \item Probability of Distribution \( P \)
  \item \textbf{Output:} \( x \in S \) with the minimal object function found so far.
\end{itemize}

\begin{align*}
i &= 0 \\
x &= \text{sample}(S, P) \\
\text{While } (i < \text{maxiteration}), \text{ do} \\
  \quad x' &= \text{sample}(S, P) \\
  \quad \text{If } f(x') < f(x), \text{ then} \\
  \quad \quad x &= x' \\
  \quad \text{End If} \\
  \quad i &= i + 1 \\
\text{End While} \\
\text{Return } x
\end{align*}

2.2.2 Hill Climbing

Hill climbing method is very similar to the Monte Carlo method. The only difference is that they generate the new sample point in a different way. In the Monte Carlo method, a new point is generated according to a probability distribution. In the hill
climbing method, a new point is generated according to a neighborhood function. So it can only explore a nearby region of the current point. The following is a sketch of the algorithm.

\textit{Input: Object function} \( f : S \rightarrow R \)

\textit{Neighborhood function} \( N \)

\textit{Initial point} \( x_0 \)

\textit{Output:} \( x \in S \) with the minimal object function found so far.

\[ i = 0 \]
\[ x = x_0 \]

\textbf{While} \( (i < \text{maxiteration}) \), \textbf{do}
\[ x' = N(S, x) \]
\[ \text{If} \ f(x') < f(x), \text{then} \]
\[ x = x' \]
\[ \text{End If} \]
\[ i = i + 1 \]
\textbf{End While}

\textbf{Return} \( x \)

\textbf{2.2.3 Simulated Annealing}

Both Monte Carlo and hill climbing algorithms accept the new point only if the objective value is improved. In this way, it is very easy to get stuck in local minima. Simulated annealing is inspired by the physical process of crystallization of materials during annealing process. In an annealing process, the material initially is at high temperature and disordered. Then it is slowly cooled down so that the system is approximately in thermodynamic equilibrium at any time. If the initial temperature of the system is too low or cooling is done insufficiently slowly, the system may become quenched forming defects. The material may not end up in the minimum energy state.
This annealing process is translated into an optimization method as follows: The energy function corresponds to the object function that we are trying to optimize. A neighborhood function is specified to generate a new point from the current point. The algorithm starts from a random initial point \( x_0 \) and a given temperature \( T_0 \). The temperature is decreased according to a cooling scheme. At each temperature setting, a quasi Boltzmann distribution is obtained by randomly generating a point \( x' \), reject or accept the new point with probability

\[
P(\Delta f) = \begin{cases} 
1 & \text{if } f(x') < f(x) \\
\frac{e^{\Delta f/T}}{e^{\Delta f/T}} & \text{else}
\end{cases}
\]

where \( \Delta f = f(x') - f(x) \)

The difference between the simulated annealing (SA) method and the previous two methods is that SA not only accepts a better solution, but also accepts a worse solution with a low probability. When the temperature is high, it is more willing to accept a bad move. When the temperature is low, the probability to accept a bad move is very low. The following is a sketch of the algorithm.

**Input**: Object function \( f : S \to R \)

Neighborhood function \( N \)

Cooling scheme function \( \text{Cool}() \)

Initial point \( x_0 \)

Initial temperature \( T_0 \)

**Output**: \( x \in S \) with the minimal object function found so far.

\( T = T_0 \)

\( x = x_0 \)

**While** (Stop criteria is false), do

**While** (not equilibrium), do

\( x' = N(S, x) \)

If \( f(x') < f(x) \) or \( e^{\Delta f/T} > \text{random}(0,1) \), then
\[ x = x' \]

End If

End While

\[ T = \text{Cool}(T) \]

End While

Return \( x \)

### 2.3 Evolutionary Algorithms

Evolutionary algorithms (EAs) are inspired by the evolving process happening in nature. Only the individuals who can fit the environment will survive, and only the species which can fit the environment will survive. Using this strategy, evolutionary algorithms keep a set of solutions (population). Each solution is evaluated based on the objective function. The good solution has more opportunity to survive in the next generation (population). Bad solutions are discarded or have a very low probability to survive in the next generation. After selecting those who may survive in the next generation, new solutions are generated by recombination and mutation, so the population size is stable. The recombination and mutation operations correspond to the mate and mutation phenomena which happen in nature. A simple flow chart of a basic evolutionary algorithm is depicted in Figure 2.1.

Evolutionary algorithms can be dated back to 1950’s [5] [6]. Based on different origins and time periods, currently there are three main sub areas. Genetic algorithms (GAs) are generally known after Holland [7]. Evolutionary programming (EP) was first proposed by Fogel [8]. Evolutionary strategies (ESs) are due to Rechenberg [9]. After 1980’s, due to more and more communications among this community, the distinctions between these methods have become more and more subtle. Nowadays, evolutionary strategies usually have the recombination operation which was originated in GAs.
Fig. 2.1. Schematic of Basic Evolutionary Algorithms.
2.3.1 Genetic Algorithms

The genetic algorithm was introduced by Holland [7] in 1970’s. It traditionally uses a fixed length binary bit string to represent each individual solution. The two main genetic operators are crossover and mutation. Figure 2.2 gives examples of the two operations. After initializing the population, each individual string is evaluated against the object function to get a fitness value. Then, parents are selected according to a probability function based on their fitness values. Between the two parents, a random crossover point is chosen, and the two parents exchange the bit information around the chosen crossover point. This process is repeated until enough children are generated. Then, a selection scheme is used to discard those individuals with low fitness value. Finally, a binary mutation operation is used on each individual with a very low probability. The process is repeated until the best individual is satisfactory.

A mathematic tool called schema theory was developed by Holland. The schema theory was used to try to explain why the genetic algorithms work. However, recently
several researchers have pointed out some weaknesses of the schema theory [10], [11], [12]. In spite of this, various genetic algorithms have been successfully applied to many practical applications.

2.3.2 Evolutionary Programming

Evolutionary programming (EP) was introduced by L. Fogel in the early 1960’s. It was originally aimed to machine intelligence with limited success. Later in the early 1990’s, D. Fogel [13], [14] applied it to optimization problems.

After initializing the population with $P$ individuals, each individual is mutated to generate a child. Using a probability tournament selection method, $P$ individuals survive. At the same time, the best solution found so far is guaranteed to survive into the next generation, which is called elitism.

In evolutionary programming, the variables are usually represented as real numbers. The mutation is different from the binary mutation used in genetic algorithms. The mutation is done by adding some random gaussian noise. For example $x'_i = x_i + N(0, \sigma_i)$, where $N$ is a standard gaussian distribution.

2.3.3 Evolutionary Strategies

Evolutionary strategies (ESs) were credited to Rechenberg [15] and Schwefel [16] in the middle of 1960’s in Germany. After initializing a population with a certain size, each individual generates a child by adding a gaussian random variable with zero mean and a pre selected standard deviation. At the same time, the standard deviation usually is made self-adaptive. This process is repeated until a satisfactory solution is found.

The original idea focused on a single parent, single child search. This is called (1+1) ES. A single child is generated from a single parent, and then the two individuals compete for surviving into the next step. Later on, two main approaches became more popular. For $(\mu + \lambda)$ ES, $\mu$ parents are used to create $\lambda$ children. Then $\mu$ new
parents are chosen from the $\mu + \lambda$ individuals. For $(\mu, \lambda)$ ES, $\mu$ parents generate $\lambda$ children. When $\mu \leq \lambda$, then $\mu$ new parents are chosen from the $\lambda$ children for the next generation.

Like evolutionary programming, most efforts are focused on the adaptive mutation. However, modern evolutionary strategies usually combine the recombination operation into the algorithms which is usually used in Genetic Algorithms.

2.3.4 Disadvantages and Advantages of Evolutionary Algorithms

Evolutionary algorithms are stochastic search algorithms, and are population based. When compared to some local search algorithms, they are computationally demanding. The magic parameters are difficult to choose, for example, the crossover rate, the mutation rate, the selection rate and the standard deviations used in EPs and ESs. Although evolutionary algorithms are global optimization methods, they are not guaranteed to find the global minimum. The theoretic results of converging to global minimum with probability 1 are useless from a practical point of view. Another drawback of evolutionary algorithms is that it is very difficult to handle the constraints generally.

In spite of the drawbacks mentioned, evolutionary algorithms become more and more popular in all kinds of applications. The simplicity and flexibility of EAs make them applicable to a very broad domain. With difficult problems, such as the traveler sale problem, evolutionary algorithms have gained a lot of success. Another merit of EAs is the parallelism inherited from the population concept. With tremendous progress in computer engineering, the computation burden of EAs may be alleviated to some extent. Robustness is an important property to evaluate an optimization method. EAs are population based algorithms, and they are not very sensitive to the initial points or small computation errors. Usually, this is not the case for most traditional optimization methods, such as the gradient based algorithms.
2.4 Summary

This chapter covered some of the basics of general optimization problems, especially the global optimization problems. Several global optimization techniques were briefly introduced. In the second part of the chapter, three important evolutionary algorithms were introduced. Some of the disadvantages and advantages of evolutionary algorithms were discussed.
3. GRAY CODE OPTIMIZATION ALGORITHMS

In this chapter, a deterministic evolutionary like algorithm called Gray code optimization (GCO) is introduced. The GCO algorithm utilizes the adjacency properties of the Gray code representation. By flipping different bits in the string, it searches through the space. By controlling the number of bits flipped, it keeps a good balance between global search and local search. The original GCO algorithm has the drawback of getting stuck in a local minimum at times. For some correlated object functions, it also tends to converge slowly. A hybrid algorithm combining the GCO algorithm with a local search algorithm is proposed. To further alleviate the computational load for very high dimensional problems, a parallel GCO algorithm is proposed and its performance in a small Beowulf cluster is reported.

3.1 Introduction of the Gray Code

The Gray code was invented by Emile Baudot. It was widely used in digital encoding [17], [18], [19]. Hollstien [20], Haupt [21] explored to use the Gray code representation in genetic algorithms by replacing the ordinary binary code representation. Better performance was reported. Valafar and Ersoy [2] started to use the Gray code more directly in optimization.

The unique property of the Gray code is the adjacency property, where two adjacent integers’ representations differ at only one bit position. There is also one to one mapping between the binary code representation and the Gray code representation. Table 3.1 gives an example of the Gray code from 0 to 7. Transformations between the binary code and the Gray code representation can be written as

\[
\text{Binary}(b_n b_{n-1} \ldots b_2 b_1) \rightarrow \text{Graycode}(g_n g_{n-1} \ldots g_2 g_1)
\]

Where \( b_n = g_n \), \( g_i = b_i \oplus b_{i+1} \) for \( i < n \)
\[ \text{Graycode}(g_n g_{n-1} \ldots g_2 g_1) \rightarrow \text{Binary}(b_n b_{n-1} \ldots b_2 b_1) \]

Where \( g_n = b_n \), \( b_i = b_{i+1} \oplus g_{i+1} \) for \( i < n \)

<table>
<thead>
<tr>
<th>Decimal Number</th>
<th>Binary Code</th>
<th>Gray Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>011</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>010</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>101</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>100</td>
</tr>
</tbody>
</table>

3.2 The Gray Code Optimization Algorithm

The GCO algorithm starts with encoding the variables in a binary string, and then transforms it into a Gray code representation string. By systematically flipping the different portions of the Gray code string, it generates a population. First, it flips all the bits, then flips the first half of the bits, the second half of the bits, the first quarter of the bits ... the first bit, the second bit ... the last bit. The population size depends on the number of the variables, and the bit length for each variable. After generating the whole population, it transforms the Gray code bit strings back to the binary strings and evaluates them. After finding the best child bit string, the algorithm starts over again with the new string. This process is repeated until no improvement can be made, or the maximum epoch number is reached, or the time limit is reached. The pseudo code for the GCO algorithm is sketched as follows:
Stage 1:
1 for = 1: initial points
2 Initialize the start point, each variable is represented by 8 bits
3 While stop criterion is false
   3.1 Transform the parent string into Gray code space, generate the population
   3.2 Transform child strings in the population back to ordinary binary space, and evaluate them
   3.3 Choose the best child string as the parent, go to step 2
4 End
Stage 2
1 Choose the best result from stage 1 as the initial point for the stage 2
2 double the bit length of each variable
3 While stop criterion is false
   3.1 Transform the parent string into Gray code space, generate the population
   3.2 Transform child strings in the population back to ordinary binary space, and evaluate them
   3.3 Choose the best child string as the parent, go to step 3
4 Go to Step 2

The GCO algorithm is divided into multiple stages. In the first stage, a lower resolution is used, which means a short bit string is used for each variable. The algorithm starts from multiple initial points. The best result is used as the initial point for the second stage. During the second stage, a higher resolution with doubled bit length is used. A deeper and better solution is expected to be found in this stage. This is continued until reaching the chosen maximum number of bits for each variable, which is often set as 32. Most optimization algorithms are sensitive to the choice of initial points. With a bad choice, it is very easy to get stuck in a local minimum. Because of the fast speed of the GCO algorithm, especially in the low resolution case, it can explore a huge search space with little effort. With multiple initial points in the low resolution stage, it has a good chance to find a good initial
point for the later stage. The multiple stages method has proved to be the key to avoid most local minima. Figure 3.1 gives a specific example of how the algorithm generates a population from a single parent based on bit flipping. Here, we assume the variables are encoded with bit length 8.

![Diagram of population generation from a single parent](image)

Fig. 3.1. Generation of a Population from a Single Parent in the GCO Algorithm.
3.2.1 Discussion of the GCO Algorithm

The Gray code optimization algorithm is an evolving algorithm. In each generation, for $n$ bit string, it generates $2n - 1$ child bit strings. It finds the best child string and goes on from there to the next generation. The population is generated from a single parent, which is not the case for most evolutionary algorithms. For evolutionary algorithms, the main operator is mutation, which is implemented by adding some gaussian noise. In the Gray code optimization algorithm, a procedure similar to mutation is implemented by bit flipping. Unlike most evolutionary algorithms, the new algorithm is deterministic. No random mechanism is used during the progress of the algorithm except choosing the initial points. At the same time, there is no need to choose some user defined parameters, such as population size, generation number, mutation parameters and so on.

The only basic operation is the bit operation. This is very fast in current sequential computers. Within a given amount of time, the new algorithm can therefore explore many more points in the search space. This greatly enhances the possibility to find the global optimal point.

The new algorithm balances well between global search and local search. In the Gray code representation, the more bits are flipped, the further the new point is away from the starting point. The fewer bits are flipped, the closer the new point is from the starting point. In this way, it keeps a good balance between global search and local search in each generation.

3.2.2 Experimental Results with the GCO Algorithm

The GCO algorithm was tested on several functions with multiple local minima. Its performance was compared to genetic algorithms. To make the comparisons more meaningful, the population size and max epoch number of the genetic algorithm were chosen so that its function evaluation numbers were close to the function evaluation numbers used by the GCO algorithm. 10 rounds of each algorithm were conducted.
Since some parent points may be repeated in the next generation, we assume genetic algorithms only need evaluate 60% of the population on the average. For example, if the population size is 100, the max number of iteration is 100, the approximate function evaluations are assumed to be $0.6 \times 100 \times 100 = 6000$.

The genetic algorithm toolbox by Haupt [21] was used. The test functions are described in appendix A. The experimental results are summarized in Table 3.2. The time is computed as the number of function evaluations.

<table>
<thead>
<tr>
<th>Test Function</th>
<th>$f_{GCO}$</th>
<th>$f_{GA}$</th>
<th>$Time_{GCO}$</th>
<th>$Time_{GA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>-186.7308</td>
<td>-186.6528</td>
<td>3681</td>
<td>4000</td>
</tr>
<tr>
<td>$f_2$</td>
<td>-16.9487</td>
<td>-16.9159</td>
<td>3754</td>
<td>6000</td>
</tr>
<tr>
<td>$f_3$</td>
<td>-23.8062</td>
<td>-23.7876</td>
<td>3988</td>
<td>24000</td>
</tr>
<tr>
<td>$f_4$</td>
<td>-3.322368</td>
<td>-3.286580</td>
<td>32174</td>
<td>48000</td>
</tr>
<tr>
<td>$f_5$</td>
<td>1.32 $\times 10^{-14}$</td>
<td>1.18 $\times 10^{-2}$</td>
<td>53715</td>
<td>60000</td>
</tr>
<tr>
<td>$f_6$</td>
<td>2.69 $\times 10^{-15}$</td>
<td>1.09 $\times 10^{-3}$</td>
<td>49812</td>
<td>72000</td>
</tr>
</tbody>
</table>

As seen in Table 3.2, the GCO algorithm has no trouble in finding the global optimal points most of the time. However, given similar resources, the genetic algorithms used sometimes found the sub-optimal points, especially evident in the test functions $f_2, f_3, f_4, f_5,$ and $f_6$.

### 3.3 Hybrid GCO Algorithm

The GCO algorithm is a search algorithm purely based on binary mutation, without any knowledge of the object function. It may show slow convergence toward the end of search. It is the same case for most evolutionary algorithms. As well known, if the point is near the true optimum, a local optimizer is much more pow-
erful. To combine the global search power of the GCO algorithm with the speed of a local optimizer, a Hybrid GCO algorithm (HGCO) was proposed. There are many local search algorithms to choose from, such as the line search algorithm, the steepest descent algorithm, the simplex algorithm, the conjugate gradient algorithm, the Newton algorithm and so on. The downhill simplex algorithm does not need any gradient information of the object function, so it was chosen as the local optimizer.

3.3.1 The Nelder-Mead Simplex Method

Since its publication in 1965, the Nelder-Mead simplex algorithm [22] has become one of the most widely used methods for nonlinear unconstrained optimization. The Nelder-Mead algorithm is sometimes confused with the more famous simplex algorithm of Dantzig for linear programming. Both algorithms employ a sequence of simplexes but are otherwise completely different and unrelated. In particular, the Nelder-Mead method is intended for unconstrained optimization.

The Nelder-Mead (NM) algorithm aims to minimize a real variable objective function without any derivative information. It falls in the general class of direct search algorithms. At each step, it maintains a non-degenerate simplex, a geometric figure in \( n \) dimensions of nonzero volume which is a convex hull of \( n + 1 \) vertices.

3.3.2 The HGCO Algorithm

It is straightforward to combine the GCO algorithm with the NM simplex algorithm. Basically, the NM simplex algorithm is appended to the GCO algorithm. The best solution found so far by the GCO algorithm is the initial search point for the NM simplex algorithm. The following is the sketch of the HGCO algorithm. How to decide when to switch from the GCO to the NM algorithm is still a research topic. Currently, we decide to switch to the NM algorithm if there is no significant improvement for a period time.
1 Run the GCO algorithm
2 Set the best solution found in step 1 as the initial point for the NM simplex algorithm
3 Run the NM simplex algorithm

3.3.3 Experimental Results of the HGCO Algorithm

Evolutionary algorithms are stochastic algorithms. In most cases, they can find solutions close to the global optimum point. However, it is very difficult to locate the exact global optimum point. Especially with highly correlated object functions, it will take a lot of random mutations to get close to the global optimum. For the GCO algorithm, it is based on binary representation. The resolution of the solution is decided by how many bits are used to represent the variable and the range of the variable. If the range of the variable is very large, the resolution will be very low. For example, if using 16 bits to represent a variable, when the range is \([-10, 10]\], the resolution is \(3.05 \times 10^{-5}\), and when the range is \([-1000, 1000]\], the resolution is \(3.05 \times 10^{-2}\). The local optimizer uses real number to represent the variables; the resolution is determined by the computer resolution, which is usually high enough.

To illustrate the slow convergence of the GCO algorithm, two new object functions \(f_7\) and \(f_8\) are introduced. The variables of these functions are highly correlated. Table 3.3 shows the experimental results.

<table>
<thead>
<tr>
<th>Test Function</th>
<th>(f_{HGCO})</th>
<th>(f_{GCO})</th>
<th>(\text{Time}_{HGCO})</th>
<th>(\text{Time}_{GCO})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_1)</td>
<td>-186.7308</td>
<td>-186.7308</td>
<td>3202</td>
<td>3681</td>
</tr>
<tr>
<td>(f_4)</td>
<td>-3.322368</td>
<td>-3.322368</td>
<td>25232</td>
<td>32174</td>
</tr>
<tr>
<td>(f_7)</td>
<td>(2.14 \times 10^{-16})</td>
<td>(1.08 \times 10^{-6})</td>
<td>49013</td>
<td>500078</td>
</tr>
<tr>
<td>(f_8)</td>
<td>(1.09 \times 10^{-16})</td>
<td>(4.02 \times 10^{-2})</td>
<td>50170</td>
<td>500210</td>
</tr>
</tbody>
</table>
From Table 3.3, it is clear that for the object functions $f_1$ and $f_4$, both GCO and HGCO algorithms are able to find the global minima, but HGCO performs with less number of function evaluations. For $f_7$ and $f_8$, HGCO algorithm is given only $1/10$ of the resources used by the GCO algorithm, but still manages to find a much deeper global minimum. The experimental results show that combining the GCO algorithm with a local optimizer not only can save the resources, but also can improve the quality of the solutions.

3.4 Parallel Gray Code Optimization Algorithm

A parallel Gray code optimization (PGCO) algorithm is proposed in this section. The Gray code optimization algorithm shares some similarities with genetic algorithms and evolutionary programming. It uses a binary representation, but the only operator is the mutation of a number of bits. The evolving strategy utilizes the adjacency property of the Gray code. By controlling how many bits to flip, it keeps a balance between global search and local search. Another property of the GCO is that the population size is not fixed. It grows linearly with the dimension of the problem, which help to alleviate the curse of the dimensionality. In order to avoid the slow convergence of high dimensional problems, a parallel Gray code algorithm using Message Passing Interface (MPI) was implemented. Its scalability in a Beowulf Windows Cluster was investigated.

3.4.1 Introduction

For high dimensional problems, the fitness functions are very CPU intensive and a lot of function evaluations are needed before convergence. Fortunately, population based evolutionary algorithms (EA) are inherently parallel, because many evolutionary operators are applied independently on different individuals. Especially for the case of evolutionary programming (EP), the only operator is the mutation. There are three different parallel models, which are the farming model, the island model
and the neighborhood model [23]. The farming model is the simplest one. It has a master processor to generate the population, and to perform all the evolutionary operators. The master processor manages a number of slave processors, whose job is to perform the fitness evaluations. The island model divides the population into many subpopulations. Each process has its own subpopulations, and runs its own evolutionary algorithm. From time to time, different subpopulations exchange information. The neighborhood model is an extreme case of the island model. Each processor has only one individual. The individuals exchange information only with its neighborhoods. For the island model, the parameter of subpopulation size, and the strategy of migration have to be chosen. For the neighborhood model, the available CPUs put a limitation on the population size, and how to define the neighborhood is another issue. In this section, the farming model was chosen for its simplicity and better suitability for the proposed algorithm.

3.4.2 The PGCO Algorithm

The parallel Gray code optimization algorithm falls into the farming model category. It has a master processor, whose job is to generate the population, distribute them into available slave processors to evaluate, collect the results back, find the best child string, and repeat the whole operation until the stop criterion is true. The job of a slave processor is to wait for the child string from the master processor. After receiving a child string, it evaluates its fitness value and sends the result back to the master processor. The pseudo code for the PGCO algorithm is sketched as follows:

1. If (processor == master)
   1.1 If stop criteria is false
      1.1.1 Generate the population; distribute it to the slave CPUs
      1.1.2 Collect all the results, find the best child, go to 1.1
      1.2 Else sending stop signal to all the slave processors

2. If (processor == slave)
2.1 If signal is not stop

2.1.1 Wait for child string from the master processor, evaluates it and sends the result back

2.2 Else stop

3.4.3 Experimental Results

The PGCO algorithm was implemented using the message passing interface (MPI) [24]. MPI is a specification of a message passing library for parallel computers. With supports from academia, government and industry, it has now become a de facto standard for parallel computing. The implementation used here is called MPICH [24]. A windows Beowulf cluster was built to test the algorithm. It consists of a domain controller, and 8 computational nodes. The nodes include a variety of different CPUs with different speeds.

Six high dimensional benchmark functions [25], [26] were used in these experimental studies. Appendix A lists all the functions. All the functions are multimodal functions in which the number of local minima increases exponentially with the dimension [25]. In the work of [25], [26], the dimension was set to 30. For comparison, dimension 30 was also used in this study. For functions with so many local minima, the quality of the solution is more significant than the convergence rate. Since the PGCO is a deterministic algorithm (no random number generator is used except in initialization), the PGCO algorithm was configured to run until no improvement could be made. The last three columns of Table 3.4 list the solutions found by PGCO, fast evolutionary program (FEP) [25], and Levy evolutionary program (LEP) [26].

The results show that, among 5 of the 6 test functions, the PGCO algorithm was capable of finding deeper minimum solution than the FEP and LEP. The only exception is the function $f_{10}$. Both FEP and LEP are a little better.

\footnote{If $f$ is less than $1.0 \times 10^{-12}$, it is approximated as 0}
Table 3.4
Experimental Results of the PGCO Algorithm.

<table>
<thead>
<tr>
<th>Test Function</th>
<th>$f_{\text{min}}$</th>
<th>$f_{\text{PGCO}}$</th>
<th>$f_{\text{FEP}}$</th>
<th>$f_{\text{LEP}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_9$</td>
<td>$-1.26 \times 10^4$</td>
<td>$-1.26 \times 10^4$</td>
<td>$-1.25 \times 10^4$</td>
<td>$-1.18 \times 10^4$</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>0</td>
<td>$2.1 \times 10^1$</td>
<td>$4.6 \times 10^{-2}$</td>
<td>$1.25 \times 10^1$</td>
</tr>
<tr>
<td>$f_{11}$</td>
<td>0</td>
<td>$2.98 \times 10^{-8}$</td>
<td>$1.8 \times 10^{-2}$</td>
<td>$1.9 \times 10^{-2}$</td>
</tr>
<tr>
<td>$f_{12}$</td>
<td>0</td>
<td>0</td>
<td>$1.6 \times 10^{-2}$</td>
<td>$2.4 \times 10^{-2}$</td>
</tr>
<tr>
<td>$f_{13}$</td>
<td>0</td>
<td>0</td>
<td>$9.2 \times 10^{-6}$</td>
<td>$6.0 \times 10^{-6}$</td>
</tr>
<tr>
<td>$f_{14}$</td>
<td>0</td>
<td>0</td>
<td>$1.6 \times 10^{-4}$</td>
<td>$9.8 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
3.4.4 Scalability of the PGCO Algorithm

For a parallel algorithm, or a parallel cluster, the scalability is an important performance parameter to estimate how much benefit can be gained by using parallel computing. Under moderate assumptions, the computation time per generation of the PGCO algorithm for an uneven (meaning that the nodes have different CPU speeds) Beowulf cluster can be approximated as

\[ T = T_0 + n \cdot T_c + \frac{2n \cdot T_f \cdot (T_s + T_c)}{m \cdot (T_f + T_s + 2T_c)} \]

where:

- \( n \) = Population size
- \( m \) = Number of computation nodes (assuming half with fast CPUs, half with slow CPUs)
- \( T_c \) = Average communication time between computers
- \( T_f \) = Average computation time used by a fast CPU to evaluate one string
- \( T_s \) = Average computation time used by a slow CPU to evaluate one string
- \( T_0 \) = Average computation time used by the master node for book keeping jobs

Discussions:
1. If \( T_c \approx 0 \) and \( T_f \approx T_s \), then \( T = T_0 + \frac{n \cdot T_f}{m} \), which scales well with increasing \( m \).
2. If \( T_c \approx 0 \) and \( T_s \approx 2T_f \), then \( T = T_0 + \frac{4n \cdot T_f}{3m} \), the performance is degrading by \( 4/3 \).
3. If \( T_f \approx T_s \approx 0 \), then \( T = T_0 + n \cdot T_c \), the number of CPUs used has no influence at all.

From the formula and the experiments, we observed that the PGCO algorithm scales well if (1) the cost function is time consuming, (2) the cluster nodes have similar computation power, and (3) the cluster has a fast network. The speedup of the algorithm in the experimental cluster is shown in Figure 3.2.
According to Cantu-Paz [23], for a parallel evolutionary algorithm, the number of optimal computation nodes belongs to $O(\sqrt{\frac{NT}{T_c}})$. The PGCO algorithm has the feature that the population size is linearly increasing with the dimensions, which make it suitable for parallel computing in high dimensional problems.

### 3.5 Summary

In this chapter, a global optimization algorithm based on the Gray code scheme was further developed. To further improve the speed of convergence, a hybrid Gray code algorithm was proposed. To attack high dimensional problems, and to take advantage of the cheaper and cheaper CPUs, a parallel Gray code optimization algorithm was proposed. To test the new parallel algorithm, a 9 nodes windows cluster was built. It showed sub-linear speed up.
4. MIXTURE GAUSSIAN OPTIMIZATION ALGORITHM

The Gray Code Optimization (GCO) algorithm is a deterministic, binary based algorithm. As shown in the previous chapter, it sometimes suffers from slow convergence and sub-optimal solutions. The Expectation Maximization (EM) algorithm is used to analyze how the GCO explores the search space. This leads to EM algorithm extracting a three component mixture Gaussian model. A novel stochastic optimization algorithm based on the mixture Gaussian model is then proposed. The new Mixture Gaussian Optimization (MGO) algorithm is not only a continuous stochastic algorithm, but also provides a rigorous mathematic model for answering some theoretic questions. In this chapter, using the EM algorithm to approximate the search method by GCO is first introduced. Then the MGO algorithm is described. In the later part, a proof of the convergence of the MGO algorithm using a Markov Model is given. In the end, the convergence rate on a sphere function is studied.

4.1 Using the EM Algorithm to Model GCO Search

The GCO algorithm uses bits flipping to search through the space. By flipping a large portion of the bits, it generates points which are far away from the initial point. By flipping a small portion of the bits, it generates points which are closer to the initial point. But it is very difficult to exactly describe how it covers the search space. To further understand the GCO algorithm, a single variable problem is studied here. From an initial point, a population is generated by the GCO algorithm. Then, a histogram is generated to describe how the children points cover the search space. For simplicity, we assume the search range is [0, 1]. Figure 4.1 to Figure 4.3 show 3 different data sets and their histograms generated with the GCO algorithm.

\(^1\)Data are given in Appendix B
Fig. 4.1. Histogram Generated with the Initial Point Equals to 0.6946.

Fig. 4.2. Histogram Generated with the Initial Point Equals to 0.5226.
Fig. 4.3. Histogram Generated with the Initial Point Equals to 0.4449.
4.1.1 Expectation Maximization (EM) Algorithm

The EM algorithm [27] [28] is an efficient iterative procedure to compute the Maximum Likelihood estimation of statistical model parameters in the presence of missing or hidden data. It is often used to approximate a probability density function (p.d.f).

Each iteration of the EM algorithm consists of two processes: The E-step, and the M-step. In the expectation, or E-step, the missing data are estimated given the observed data and the current estimate of the model parameters. This is achieved using the conditional expectation. In the M-step, the likelihood function is maximized under the assumption that the missing data are known.

Let the observed variable be known as $Y$ and the latent variable as $Z$. Together, $Y$ and $Z$ form the complete data. Assume $p$ is a joint model of the complete data with parameters $\theta : p(y, z|\theta)$. The EM algorithm iteratively improves on an initial estimate $\theta_0$ and constructs new estimates $\theta_1, \theta_2, \ldots \theta_n$. Define the expectation $\mathcal{Q}(\theta)$ as

$$
\mathcal{Q}(\theta) = E[\log p(z, y|\theta)|\theta^*, y] = \sum_z p(z, y|\theta^*) \log p(y, z|\theta)
$$

Estimation of $\theta_{n+1}$ from $\theta_n$ can be expressed as

$$
\theta_{n+1} = \arg\max \sum_z p(z, y|\theta^n) \log p(y, z|\theta)
$$

A sketch of the EM algorithm is as follows:

1. $i = 0$, randomly initialize $\theta_0$
2. Compute $\mathcal{Q}(\theta|\theta_i)$
3. Choose $\theta_{i+1}$ to maximize $\mathcal{Q}(\theta|\theta_i)$
4. if $\theta_i$ and $\theta_{i+1}$ are not close enough, $i = i + 1$, go to step 2

The EM algorithm converges to a local maximum of the observed data likelihood function.
4.1.2 Modeling of Histogram Data Sets with the EM Algorithm

A mixture gaussian model can approximate any continuous probability density function. In this research, a three component Gaussian model is used to approximate the histogram data sets generated with the GCO. Figure 4.4 through 4.6 are the results from the EM algorithm corresponding to the data sets 1, 2, and 3 shown in Figure 4.1 through 4.3.

Fig. 4.4. Modeling of the Histogram Data in Fig. 4.1.

Fig. 4.5. Modeling of the Histogram Data in Fig. 4.2.
4.2 Mixture Gaussian Optimization (MGO) Algorithm

Inspired by the mixture Gaussian modeling of the histogram data obtained with the GCO algorithm, a new continuous stochastic optimization algorithm MGO is proposed. Although the GCO and MGO algorithms share some similarities, they are essentially different. The MGO algorithm operates in the continuous space. The representation accuracy is decided by the computer’s machine resolution, not by the bit length as in the GCO algorithm. The MGO is a stochastic algorithm, not a deterministic algorithm as the GCO. Another difference is that for the mutation operation, the GCO uses bit flipping where the MGO uses mixture Gaussian noise.

The MGO algorithm is an evolutionary algorithm, but is quite different from other evolutionary algorithms. The biggest difference between the MGO algorithm and other continuous evolutionary algorithms is that the MGO uses mixture Gaussian; all others use a single Gaussian. The advantage of using mixture gaussian is that it covers a broad range of the search space, and favors global exploration more. Intuitively, it is thus easier to escape local minima and to prevent the pre-mature convergence.

The sketch of the MGO algorithm:

1. Initialize the parent randomly.
2. Generate a population by adding each variable a mixture Gaussian noise. The parameters of the mixture Gaussian model are pre-defined; the size of the population is pre-defined.

3. Evaluate the population, find the best child.

4. If stop criteria is false, go to step 2.

The parameters of the mixture Gaussian distribution are very important. They are chosen experimentally based on the results with the GCO algorithm in the previous section. The first standard deviation $\sigma$ for the major Gaussian is chosen to be small, and the standard deviations for the other two gaussians are chosen to be bigger. Intuitively, small standard deviation favors local search, and big standard deviation favors global search. The three means ($\mu$) are chosen to cover the whole search range as completely as possible. The coefficient of each Gaussian component is chosen empirically based on the results of the previous section.

In practice, we choose 2.0 as the initial value for the standard deviation of the first gaussian component, and 4.0 as the initial value for the standard deviations of the other two gaussian components. The first standard deviation will decrease gradually generation by generation. So the algorithm will not lose the focus. At the same time, the other two standard deviations are kept constant. So the algorithm always keeps some pressure of global exploration.

4.3 Experimental Results

The experiments were conducted on a series of high dimensional problems. Each function has many local minima. For comparison, two popular Evolutionary Algorithm toolboxes were used. GAToolbox 1 is from Matlab [29], GAToolbox 2 is from University of Sheffield, UK [30]. For fairness, each algorithm is given roughly the same resource. Here, it is the function evaluation number of the objective function. The results are the average of 20 independent runs, and they are reported in Table 4.1.
Table 4.1
MGO Experimental Results $n = 30$.

<table>
<thead>
<tr>
<th>Test Function</th>
<th>$f_{\text{min}}$</th>
<th>$f_{\text{MGO}}$</th>
<th>$f_{\text{toolbox1}}$</th>
<th>$f_{\text{toolbox2}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_9$</td>
<td>$-1.26 \times 10^4$</td>
<td>$-1.26 \times 10^4$</td>
<td>$-1.25 \times 10^4$</td>
<td>$-1.26 \times 10^4$</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>0</td>
<td>$3.2 \times 10^{-7}$</td>
<td>$2.7 \times 10^{-2}$</td>
<td>1.25</td>
</tr>
<tr>
<td>$f_{12}$</td>
<td>0</td>
<td>$5.1 \times 10^{-8}$</td>
<td>$2.1 \times 10^{-2}$</td>
<td>$3.4 \times 10^{-7}$</td>
</tr>
<tr>
<td>$f_{13}$</td>
<td>0</td>
<td>$1.3 \times 10^{-8}$</td>
<td>$2.5 \times 10^{-3}$</td>
<td>$6.2 \times 10^{-4}$</td>
</tr>
<tr>
<td>$f_{14}$</td>
<td>0</td>
<td>$4.2 \times 10^{-8}$</td>
<td>$3.4 \times 10^{-3}$</td>
<td>$7.8 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
Compared to the GCO algorithm, the MGO algorithm has no problem to optimize the function $f_{10}$. On the contrary, the GCO algorithm does poorly on this function. Another interesting property is the running time. Although the GCO algorithm is implemented in the C language, the MGO algorithm is implemented in Matlab, the MGO is 10 times faster than the GCO on these high-dimension problems on the average. The main reason is that in Matlab, the function calls can be vectorized. One function call can evaluate the whole population. On the contrary, in C, to evaluate the whole population, a lot of function calls have to be made. When the population size is large, the advantage of parallel function call is apparent.

4.4 Crossover with the MGO Algorithm

Crossover is an important operation in Genetic Algorithms [21] [12]. It tries to simulate the breeding process in nature. In Evolutionary Programming and Evolutionary Strategies, crossover is not that important [1] [31]. In most of these algorithms, crossover is put aside intentionally.

The MGO algorithm uses a greedy strategy. In each generation, it chooses the best dimension over all the dimensions. When the population size is small, sometimes, it will fail to find the global minima or converges very slowly. Since during each generation, the best point per dimension is easy to obtained, a discrete crossover scheme can be designed.

First, we compose an artificial solution $s_1$ by putting the best point in each dimensions together. At the same time, we have the best solution $s_2$ among the whole generation. Then we do the crossover between these two solutions. Assuming the dimension of the problem is $n$, a random variable of length $n$ which includes only 0 and 1 is generated. If the position $i$ is 0, then the $i$th variable comes from $s_1$, otherwise, the $i$th variable comes from $s_2$. A number of such solutions can be built in this way. In the end, we choose the best solution among all the possible solutions as the start point for the next generation.
In the following experiments, the convergence curves with crossover and without crossover on the function \( f_{10} = \sum_{i=1}^{n} [x_i^2 - 10\cos 2\pi (x_i) + 10], \quad x_i \in [-5.12, 5.12] \) were studied. In all the experiments, the dimension of the problem is 50, the population per dimension is 20 and the global minimal is 0. Figure 4.7 shows that if the population size of each dimension is too small, without crossover, it fails to locate the global minimum. Figure 4.8 shows that, even if it locates the global minimum, the convergence is very slow. On the contrary, with crossover, in both Figures 4.9 and 4.10, the convergence is fast and the global minimum is reached without any difficulty.

![Convergence of the MGO Algorithm with No Crossover and Small Population Size.](image)
Fig. 4.8. Convergence of the MGO Algorithm with No Crossover and Small Population Size.
Fig. 4.9. Convergence of the MGO Algorithm with Crossover and Small Population Size.
Fig. 4.10. Convergence of the MGO Algorithm with Crossover and Small Population Size.
4.5 Convergence of the MGO Algorithm

A process that has a random element is called a stochastic process. Such a process can be thought of as a sequence of random events occurring in time:

\[ X_0, X_1, X_2, \ldots \]

Each of the \( X_i \) is a random variable. The possible values that these variables can take are called the state of the system. A simple stochastic process is when the distribution of the state at time \( t \) depends only on what happened at time \( t-1 \). If this is the case, then it is called a first order Markov Process. The sequence \( X_0, X_1, X_2, \ldots \) forms a Markov Chain.

The search of the MGO algorithm can be formulated as a finite state Markov Chain. First, the parent of the next generation can be assumed to depend on the current parent. The process can be viewed as a Markov Process. Secondly, since each real number is represented discretely in a computer, it consists of a string of 0’s and 1’s. The search space can be viewed as a finite space, although it is usually very huge.

This Markov Chain is characterized by a state vector \( \pi \), which is a row vector describing the probability of being in each state in the initial stage of the algorithm, and a transition matrix \( P \), which is the transition probability matrix between states. The probability of being in each state after one transition can be represented by \( \pi P \). The probability of being in each state after \( n \) transitions is \( \pi P^n \). The following gives an example of 3 state case after one and two transitions:

\[
\pi = [0.1 \ 0.5 \ 0.4] \\
P = \begin{pmatrix}
1 & 2 & 3 \\
1 & 0.2 & 0.3 & 0.5 \\
2 & 0.5 & 0.1 & 0.4 \\
3 & 0.7 & 0.1 & 0.2
\end{pmatrix}
\]

After one iteration,

\[ \pi P = [0.55 \ 0.12 \ 0.33] \]
After two iterations,

\[ \pi P^2 = [0.401 \ 0.21 \ 0.389] \]

A state in this chain which has probability 1 to transfer back to the same state is called an absorbing state. Obviously, the global minimal point is an absorbing state. All the other states are called transient state. As time progresses, the behavior of the non-absorbing states can be described by either 1) transition to an absorbing state with nonzero probability in a single step, or 2) transition to some other transient state. Thus, the transition matrix of the MGO algorithm can be described as follows:

\[
P = \begin{bmatrix}
I & 0 \\
R & Q
\end{bmatrix}
\]

Where \(I\) is a \(1 \times 1\) identity matrix, which describes the transition of the absorbing state. \(R\) is a \(t \times 1\) vector, which describes the transitions from non-absorbing states to absorbing state. \(Q\) is a \(t \times t\) matrix, which describes the transitions between non-absorbing states.

According to Goodman [32], the behavior of such a chain after \(n\) iterations have the following format:

\[
P^n = \begin{bmatrix}
I & 0 \\
N_nR & Q^n
\end{bmatrix}
\]

where \(N_n = I_t + Q + Q^2 + \ldots + Q^{n-1}\), and \(I_t\) is a \(t \times t\) identity matrix. As \(n\) goes to infinity,

\[
\lim_{n \to \infty} P^n = \begin{bmatrix}
I & 0 \\
0 & (I_t - Q)^{-1}R
\end{bmatrix}
\]

where the matrix \((I_t - Q)^{-1}\) is guaranteed to exist.

**Theorem 4.5.1** Let \(\pi\) be the probability of being in each state in the initial state. And \(P\) be the transition matrix between states. For simplicity, assume the object function has only one global minimum, which is the absorbing state \(A\). When the iterations of the MGO algorithm tend to infinity, the algorithm reaches the absorbing state \(A\) with probability one.
Proof:
Assume the algorithm reaches a state $s$ after $n$ iteration.

$Probability(s \in A)$

$= \pi \cdot P^n$

$= \pi \cdot \begin{bmatrix} I & 0 \\ N_n R & Q^n \end{bmatrix}$

when $n$ goes to infinity,

$Probability(s \in A)$

$= \lim_{n \to \infty} \pi \cdot P^n$

$= \pi \cdot \begin{bmatrix} I & 0 \\ (I_t - Q)^{-1} R & 0 \end{bmatrix}$

$= \pi \cdot \begin{bmatrix} I \\ (I_t - Q)^{-1} R \end{bmatrix}$

$= \pi \cdot \begin{bmatrix} I \\ I_{t \times 1} \end{bmatrix}$

$= 1$

$\begin{bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$

$= (I_t - Q)^{-1} R = I_{t \times 1}$

Note: Since the sum of each row of $[R \quad Q]$ is 1, so $R = [I_{t \times t} - Q] \begin{bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$, then
4.6 Progress Rate on the Sphere Function

The MGO algorithm is a stochastic global optimization method. Under minor assumptions, the convergence to global optimal location is easily proved. But the global convergence provides little useful information if the algorithm requires an infinite number of generations to converge. A more interesting question is the progress rate, which is how fast the algorithm approaches the optimal location. Studying the progress rate on a general multimodal function is a very difficult problem. To study the dynamics of the MGO algorithm, a more tractable sphere function \( y = x^2 \) problem is studied in this section.

4.6.1 The Progress Rate

According to [33], the progress rate is the rate at which the solution approaches the optimal solution. Assume the optimal solution is \( x^* \), a natural definition of progress rate \( \varphi \) can be defined as the average gain from population to population as follows:

**Definition 4.6.1**

\[
\varphi = E\left[ \frac{1}{\mu} \sum_{m=1}^{\mu} \| x^* - x_m^g \| - \frac{1}{\mu} \sum_{m=1}^{\mu} \| x^* - x_{m+1}^g \| \right]
\]

where \( \mu \) is the population size, and \( g \) is the generation number.

4.6.2 The Progress Rate for Sphere function

For simplicity, assume the mixture gaussian models is given by \( c_1 G_1 + c_2 G_2 + c_3 G_3 \), where \( c_1, c_2, c_3 \) are the coefficients of each gaussian component.

\[
G_1 = \frac{1}{\sqrt{2\pi}\sigma_1} exp\left(-\frac{1}{2}\left(\frac{x-x_0-d}{\sigma_1}\right)^2\right), \quad G_2 = \frac{1}{\sqrt{2\pi}\sigma_2} exp\left(-\frac{1}{2}\left(\frac{x-x_0}{\sigma_2}\right)^2\right),
\]

\[
G_3 = \frac{1}{\sqrt{2\pi}\sigma_3} exp\left(-\frac{1}{2}\left(\frac{x-x_0+d}{\sigma_3}\right)^2\right)
\]

are the Gaussian distributions. Here \( x_0 \) is the current solution, \( d \) is the distance between the main Gaussian component \( G_2 \) and the the other two.
For the sphere function \( y = x^2 \), the optimal point is known to be the origin. For simplicity, assume \( x_0 \) is positive. The progress rate can be simplified as

\[
\varphi = E\left[ \frac{1}{\mu} \sum_{m=1}^{\mu} \| x_m^{g} \| - \frac{1}{\mu} \sum_{m=1}^{\mu} \| x_m^{g+1} \| \right]
\]

For population size equal to 1, it can be further simplified as

\[
\varphi = E \left[ \| x^g \| - \| x^{g+1} \| \right]
\]

Since the selection mechanism only accepts a solution which is better than the current solution. If the initial point is \( x_0 \), any value within \([-x_0, x_0]\) is a better solution. Then the progress rate can be calculated as

\[
\varphi = E \left[ \| x^g \| - \| x^{g+1} \|, \text{ assume } x_g = x_0 \right.
\]

\[
= x_0 - c_1 \int_{-x_0}^{x_0} |x|G_1dx - c_2 \int_{-x_0}^{x_0} |x|G_2dx - c_3 \int_{-x_0}^{x_0} |x|G_3dx
\]

\[
= x_0 - \left\{ \frac{c_1 \sigma_1}{\sqrt{2\pi}} \left[ -e^{-\frac{1}{2} \left( \frac{x^2}{\sigma_1^2} \right)} + 2e^{-\frac{1}{2} \left( \frac{x_0 + d}{\sigma_1} \right)^2} - e^{-\frac{1}{2} \left( \frac{-2x_0 + d}{\sigma_1} \right)^2} \right] + c_1(x_0 - d) \left[ \Phi \left( \frac{d}{\sigma_1} \right) - 2\Phi \left( \frac{-x_0 + d}{\sigma_1} \right) + \Phi \left( \frac{2x_0 + d}{\sigma_1} \right) \right] \right.
\]

\[
+ \frac{c_2 \sigma_2}{\sqrt{2\pi}} \left[ 1 - 2e^{-\frac{1}{2} \left( \frac{x_0}{\sigma_2} \right)^2} - e^{-\frac{1}{2} \left( \frac{-2x_0}{\sigma_2} \right)^2} \right] + c_2 x_0 \left[ \Phi \left( \frac{d}{\sigma_2} \right) - 2\Phi \left( \frac{-x_0}{\sigma_2} \right) + \Phi \left( \frac{-2x_0}{\sigma_2} \right) \right]
\]

\[
+ \frac{c_3 \sigma_3}{\sqrt{2\pi}} \left[ e^{-\frac{1}{2} \left( \frac{-d}{\sigma_3} \right)^2} + 2e^{-\frac{1}{2} \left( \frac{-x_0 - d}{\sigma_3} \right)^2} - e^{-\frac{1}{2} \left( \frac{-2x_0 - d}{\sigma_3} \right)^2} \right] + c_3(x_0 + d) \left[ \Phi \left( \frac{-d}{\sigma_3} \right) - 2\Phi \left( \frac{-x_0 + d}{\sigma_3} \right) + \Phi \left( \frac{2x_0 - d}{\sigma_3} \right) \right] \right\}
\]

where \( \Phi \) is the standard normal cumulative distribution function.

### 4.6.3 Simulation Results

To study how these parameters affect the progress rate, several simulations were conducted. Figure 4.11 shows the progress rate as a function of different initial points. Figure 4.12 shows the progress rate as a function of different coefficients combinations. Figure 4.13 shows the progress rate as a function of the distance between Gaussian
components. Figure 4.14 shows the progress rate as a function of different choices of the standard deviations.

![Graph comparing single Gaussian vs. mixture Gaussian modeling with respect to progress rate as a function of different initial points.](image)

Fig. 4.11. Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of Different Initial Points.

### 4.6.4 Discussions

From the above results, it is clear that if the initial point is far away from the optimal point, the mixture gaussian model is better than the single gaussian model. These figures show that the mixture gaussian model is generally better than the single gaussian model with respect to different combinations of parameters. When the initial point is far away from the optimal point, more random search tends to produce large
Fig. 4.12. Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of Different Coefficient Combinations.
Fig. 4.13. Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of the Distance between Gaussian Components.
Fig. 4.14. Comparison of Mixture Gaussian vs. Single Gaussian Modeling with Respect to the Progress Rate as A Function of the Standard Deviations.
progress rate, and when the solution is very close to the optimal point, a more selective single gaussian is better.

4.6.5 Progress Rate for the Sphere Function - Multiple Children Case

In the previous section, the single child case was studied. For multiple children case, the exact progress rate defined in the previous section is very difficult to compute. In this section, a simpler definition of progress rate is given and its relationship with the population size is studied.

Assume the population has $\lambda$ children now, since each child is independent to each other, based on the result from order statistics [34], the best (minimal) children has the probability density function as $p_\lambda = \lambda p(1 - F)^{\lambda - 1}$ where $p$ is the mixture Gaussian distribution, and $F$ is the cumulative distribution function of $p$. For multiple children case, to carry out the similar calculation as the above single children case is very difficult. Here, a simpler definition called Maximum Likelihood Progress Rate is given. Similar to the maximum likelihood concept, the progress rate is defined as the distance gain between the initial point and the point which has the maximum likelihood under the distribution of $p_\lambda$.

$$\varphi_\lambda = x_0 - \arg \max \{p_\lambda\}$$

$$= x_0 - \arg \max \left\{ \frac{1}{\sqrt{2\pi}} \left[ \frac{c_1}{\sigma_1} e^{-\frac{1}{2} \left( \frac{x - (x_0 - dd)}{\sigma_1} \right)^2} + \frac{c_2}{\sigma_2} e^{-\frac{1}{2} \left( \frac{x - x_0}{\sigma_2} \right)^2} \right. \right.$$

$$+ \frac{c_3}{\sigma_3} e^{-\frac{1}{2} \left( \frac{x - (x_0 + dd)}{\sigma_3} \right)^2} \right\} \left[ 1 - \left( c_1 \Phi \left( \frac{x - (x_0 - d)}{\sigma_1} \right) \right) \right.$$

$$\left. + c_2 \Phi \left( \frac{x - x_0}{\sigma_2} \right) + c_3 \Phi \left( \frac{x - (x_0 + d)}{\sigma_3} \right) \right]^{\lambda - 1} \right\}$$

(4.2)

where $x \leq x_0$.

Two simulations were conducted for the multiple children case. The first simulation studies, under different $\lambda$, the probability density function $P_\lambda$. The second
simulation studies, under the same initial point, how the different population sizes affect the maximum likelihood progress rate.

From Figure 4.15, we can see that bigger population size does push $P_\lambda$ to the optimal point, which is the origin here. From Figure 4.16, we see that the progress rate increases very quickly in the beginning, but saturates when the population size gets too big. For the sphere function, considering the balance between the fast progress rate and the computation time, population size between 30 and 60 is found to be a good choice.
Fig. 4.16. Progress Rate as a Function of Different Population Size.
4.6.6 Experimental Results

In this section, the convergence curves on a simple function \( y = x^2 \) is studied. First, we studied the single children case. Figure 4.17, 4.18, and 4.19 show the results of the two algorithms (Mixture Gaussians, Single Gaussian) started from different initial points. We can clearly see that when the initial point is far from the optimum value, the MGO algorithm converges much quicker than the single gaussian model, which is consistent with the theoretical results from the previous section.

Secondly, we studied how the population size (\( \lambda \)) affects the convergence curve. Figure 4.20, and 4.21 show the results for the MGO algorithm and a single gaussian algorithm. In both cases, we can see that increasing the population size always speeds up the converging process. But in the beginning of the optimization, The MGO algorithm converges much fast. For the two algorithms, the initial point was set to be the same, which is 2500.

4.7 Summary

In this chapter, a new algorithm MGO inspired by the GCO algorithm is proposed. A mixture gaussian model are used to generate the population. With this mathematic model, the convergence of the MGO algorithm is proved by modeling the search process as a Markov chain model, and some explorations on progress rate were conducted.
Fig. 4.17. Convergence Curve for the Mixture Gaussian Model and the Single Gaussian Model, Initial Point = 100.
Fig. 4.18: Convergence Curve for the Mixture Gaussian Model and the Single Gaussian Model, Initial Point \( = 1000 \).
Fig. 4.19. Convergence Curve for the Mixture Gaussian Model and the Single Gaussian Model, Initial Point = 1500.
Fig. 4.20. Convergence Curve of the Single Gaussian Model as A Function of the Population Size.
Fig. 4.21. Convergence Curve of the Mixture Gaussian Model as A Function of the Population Size.
5. APPLICATION IN MOLECULE CONFORMATION SEARCH

Molecule conformations are commonly defined in drug design research as structures that can be generated solely by rotations around molecule bonds. The conformations available to a molecule can have a dramatic effect on its activity. Obtaining global minimum energy conformations of a molecule is a very hard optimization problem. The difficulty arises from the following two factors: the conformational space of a reasonable size molecule is very large, and there are many local minima that are hard to escape from. The energy landscape in the conformational space is very rugged, and there are many large barriers between local minima. Among many optimization methods in conformation search, the traditional gradient based algorithm, the random search algorithm, and the Monte Carlo algorithm are the most popular. In this chapter, the MGO algorithm is used to search the conformation space and to locate the global minimal energy structure. The algorithm is implemented in an embedded language SVL (Scientific Vector Language), which is only available in a commercial software package MOE (Molecular Operating Environment, from Chemical Computing Group, Inc). The results are compared with two popular methods used in MOE.

5.1 Molecular Energy Model

A potential energy model [35], equivalently, a forcefield, assigns a potential energy value to a molecule configuration. Virtually all calculations, from partial charge calculations to mechanics, dynamics, or docking simulations, require evaluation of the potential energy and/or the gradient of the potential energy function. The gradient of the potential gives the forces in the system.
The potential energy model is a function comprising a number of terms each of which models a particular interaction, e.g. bond stretch or electrostatics. Model parameters are obtained by fitting to empirical data. Different models result from tuning the terms and parameters of a model to special classes of data, for example, proteins or carbohydrates. Some empirical models are Kollman’s All-atom [36] model, the MMFF94 [37] medicinal chemistry forcefield, the Engh-Huber [38] united-atom protein forcefield, and the PEF95SAC Carbohydrate forcefield [39].

The potential energy is a sum of interaction energies:

\[
E = E_{\text{str}} + E_{\text{ang}} + E_{\text{stb}} + E_{\text{tor}} + E_{\text{oop}} + E_{\text{ele}} + E_{\text{vdw}} + E_{\text{sol}} + E_{\text{con}}
\]

where

- \(E_{\text{str}}\) - bond stretch energies
- \(E_{\text{ang}}\) - angle bend energies
- \(E_{\text{stb}}\) - stretch-bend cross term energies
- \(E_{\text{tor}}\) - dihedral rotation energies
- \(E_{\text{oop}}\) - out-of-plane energies
- \(E_{\text{ele}}\) - electrostatic interactions
- \(E_{\text{vdw}}\) - van der Waals interactions
- \(E_{\text{sol}}\) - implicit solvent electrostatic correction
- \(E_{\text{con}}\) - constraint and restraint pseudo-energies

5.2 Implementation Details

The MGO algorithm is implemented in SVL, an embedded language of the MOE software package. The MOE has a local energy minimization method MM which is
based on gradient descent. The MM operates on the Cartesian coordinate of each atom in the molecule. Although the speed of the MM is very fast, it has two shortcomings. First, it can only locate local minima. Secondly, the number of variables grows very quickly with the number of atoms in the molecule. For example, for a 20 atom molecule, the number of variables are 60 \((3\times20)\).

In conformation search, we do not consider the Cartesian coordinate. We are interested in the flexible bonds, which can be rotated freely. The bond length is kept fixed. In practice, the local optimization function MM is used to optimize the bond length. Concentrating on the flexible bonds greatly decreases the number of variables needed to be optimized. For example, for molecule \(CH_3(CH_2)_2CH_3\), there are 14 atoms, but only one rotatable bond. If using Cartesian coordinate, there would be 42 variables. In conformation search, there is only 1 variable.

The initial position of the molecule is set randomly by rotating each bond a random angle from the original position. The range of each rotatable bond is from 0 to \(2\pi\).

5.3 Test Problems

To test the algorithm, a series of organic molecules were generated. The size of the molecules ranges from 14 atoms to 41 atoms. In this experiment, the class of molecules \(CH_3(CH_2)_nCH_3\) is used. There are two reasons why they were chosen. First, it has a lot of local minima. Secondly, the global minimal is known to the scientists.

\(CH_3(CH_2)_2CH_3\) has 2 conformations, Figure 5.1 and Figure 5.2 show the two conformations. Figure 5.1 is the global minimal configuration; Figure 5.2 is a local minimal configuration. The energies are \(-5.031\text{ kcal/mol}\) and \(-4.268\text{ kcal/mol}\), respectively.

\(CH_3(CH_2)_3CH_3\) has 4 conformations, Figure 5.3, Figure 5.4, Figure 5.5 and Figure 5.6 show the four conformations. Figure 5.3 is the global minimal configuration; Figure 5.4, Figure 5.5 and Figure 5.6 are the local minimal configurations. The
energies are -5.201 kcal/mol, -1.545 kcal/mol, -3.832 kcal/mol and -4.393 kcal/mol, respectively.

$CH_3(CH_2)_2CH_3$ has more than 500 configurations. Figure 5.7 is the global minimal energy configuration with energy -6.633 kcal/mol.
5.4 Experimental Results

In MOE, there are two built-in conformation search algorithms. The first one is a systematic search algorithm. Systematic conformational search generates molecule
conformations by systematically rotating bonds in a molecule. In any given molecule, all bonds, except bonds to terminal atoms, are candidates for rotation. Such bonds are called rotation bonds. For each rotation bond, a possible relative dihedral increment or step is pre-defined by the user. Once the step is determined for each rotation bond, the algorithm generates all combinations of conformations according to the step list.
For example, if there are two rotation bonds and the step is 60 degree, then there are \((360/60) \times (360/60) = 36\) combinations.

The second algorithm is a Stochastic Conformational Search algorithm. It generates conformations by randomly sampling local minima of the potential energy surface. This method is similar to the RIPS method [40] which generates new molecular conformations by randomly perturbing the position of each coordinate of each atom in the molecule by some small amount, typically less than 2 angstroms, followed by energy minimization. In conformational search, the algorithm is similar in essence except that it is based on random rotations of bonds instead of the Cartesian coordination.

The MGO algorithm is implemented in SVL and the results are compared with the two algorithms mentioned above. Table 5.1 shows the results. It contains the quality of the solution and the time consumed by each algorithm.

The results clearly show that for small molecules, all of the three algorithms can generate satisfactory solutions. With the increasing molecule size, the systematic search algorithm quickly becomes non-applicable. For a moderate size molecule, the stochastic search algorithm still manages to find a good solution but takes consid-

---

1. SS1: Systematic Search
2. SS2: Stochastic Search
3. NOA means that the time to compute the solution is not acceptable
Table 5.1
Conformation Search Results.

<table>
<thead>
<tr>
<th>Molecule</th>
<th>Global minimal</th>
<th>k (kcal/mol)</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH$_3$(CH$_2$)$_2$CH$_3$</td>
<td>SS$_1$</td>
<td>-5.031</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-5.031</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-5.031</td>
<td>0.765</td>
</tr>
<tr>
<td>CH$_3$(CH$_2$)$_3$CH$_3$</td>
<td>SS$_1$</td>
<td>-5.201</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-4.393</td>
<td>4.8</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-5.201</td>
<td>2.625</td>
</tr>
<tr>
<td>CH$_3$(CH$_2$)$_4$CH$_3$</td>
<td>SS$_1$</td>
<td>-5.377</td>
<td>338</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-5.377</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-5.377</td>
<td>8.422</td>
</tr>
<tr>
<td>CH$_3$(CH$_2$)$_5$CH$_3$</td>
<td>SS$_1$</td>
<td>-5.557</td>
<td>33420</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-5.557</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-5.557</td>
<td>12.3</td>
</tr>
<tr>
<td>CH$_3$(CH$_2$)$_6$CH$_3$</td>
<td>SS$_1$</td>
<td>-5.736</td>
<td>3.6e+6</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-5.736</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-5.736</td>
<td>13.2</td>
</tr>
<tr>
<td>CH$_3$(CH$_2$)$_7$CH$_3$</td>
<td>SS$_1$</td>
<td>-5.915</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-5.915</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-5.915</td>
<td>NoA</td>
</tr>
<tr>
<td>CH$_3$(CH$_2$)$_8$CH$_3$</td>
<td>SS$_1$</td>
<td>-6.095</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-6.095</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-6.095</td>
<td>NoA</td>
</tr>
<tr>
<td>CH$_3$(CH$_2$)$_9$CH$_3$</td>
<td>SS$_1$</td>
<td>-6.274</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-6.274</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-6.274</td>
<td>NoA</td>
</tr>
<tr>
<td>CH$_3$(CH$<em>2$)$</em>{10}$CH$_3$</td>
<td>SS$_1$</td>
<td>-6.453</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-6.453</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-6.453</td>
<td>NoA</td>
</tr>
<tr>
<td>CH$_3$(CH$<em>2$)$</em>{11}$CH$_3$</td>
<td>SS$_1$</td>
<td>-6.633</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>SS$_2$</td>
<td>-6.633</td>
<td>NoA</td>
</tr>
<tr>
<td></td>
<td>MGO</td>
<td>-6.633</td>
<td>NoA</td>
</tr>
</tbody>
</table>

erable amount of time. When the molecule size continuously increases, stochastic search algorithm can only find local minima. On the other hand, the MGO algorithm still can locate the global minima within acceptable time period.

5.5 Summary

In this chapter, the MGO algorithm was applied to an important computational chemistry problem called conformation search. By restricting the optimization to take place only on rotatable bonds, the difficult problem was greatly simplified. The MGO algorithm was implemented with the built-in language SVL in MOE software package. The experiments on a series molecules showed that the MGO algorithm outperforms the two commercial algorithms built in with the MOE package.
6. APPLICATIONS IN NONLINEAR LEAST SQUARE TYPE PROBLEMS

A large set of engineering and scientific problems can be converted to optimization problems, such as curve fitting, parameter estimation [27] [28], and spectral analysis [41]. Most existing algorithms use field-specific methods, such as the EM algorithm for estimating mixture model, Fast Fourier Transformation for spectral analysis. Essentially, most of the algorithms are local search algorithms. Here, we try to solve these problems through a direct global optimization approach.

6.1 Curve Fitting with Radial Basis Functions

The combination of Radial basis functions $\exp(-a(x-b)^2)$ has the merit to be able to approximate any function with any accuracy. It has often been used in building radial basis neural networks [42] [43]. For curve fitting purpose, the problem is that given a series of data points, we try to find out the best combination of certain number of radial basis functions, which has the minimum error over all the given data points. The problem can be sketched as follows:

Given a series of data points $(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)$

Find such an object function

$$f(x) = c_1 e^{-a_1(x-b_1)^2} + c_2 e^{-a_2(x-b_2)^2} + \ldots + c_m e^{-a_m(x-b_m)^2}$$

which minimizes

$$error = \sum_{i=1}^{n} \| f(x(i)) - y(i) \|$$

The interesting point of our algorithm is that we do not compute the coefficients $c_i$ explicitly. First, we treat each $c_i$ as constant 1, then we compute a $n \times m$ matrix $A$, for the given $Y = [y_1, y_2 \ldots y_n]$, and we compute $C = [c_1, c_2 \ldots c_m]$ using linear
least squares $C = A^+ Y$, where $A^+$ is the pseudoinverse matrix of $A$. That is why we call it a least square problem. The advantage using this approach is that we can reduce $1/3$ of the variables during nonlinear least square estimation. Since the object function is a linear combination of a series of radial basis functions, this is the main reason we can use the least square method to compute $c_i$.

There is one problem left, which is how to decide how many components ($m$) the objection function should have. We can start from 1 component, and increase it one by one until we are satisfied with the total error. A better way is to use a similar idea as the binary search, start from the middle point of 1 and the maximum possible $m_{\text{max}}$. If the total error is satisfied, half the number of components, otherwise, choose the middle point between current $m$ and $m_{\text{max}}$. Continue this, until we are satisfied with the least number of components.

6.1.1 Experimental Results

For comparison purpose, the performance of the MGO algorithm was compared to the curve fitting toolbox from Mathworks, Inc [29] which uses a local search method fminsearch [22]. As we can expect, the performance of a local search method greatly depends on the initial points. It is easily to get stuck in a local minimum.

In the first experiment, the object function is $Y = f(X) = 0.5 \exp(-0.37X^2) + 5 \exp(-0.57(X - 1)^2) - 13.2 \exp(-0.71(X - 4)^2) + 6.9 \exp(-1.67(X - 4.5)^2) - 2 \exp(-0.67(X - 3)^2)$. Figure 6.1 shows the result from the initial point 4.0 for each variable. Figure 6.2 shows the result from the same initial point. In the second experiment, the object function is $Y = f(X) = 0.5 \exp(-0.37X^2) + 9 \exp(-0.57(X - 1)^2) - 13.2 \exp(-0.71(X - 4)^2) + 6.9 \exp(-1.67(X - 4.5)^2) - 2 \exp(-0.67(X - 3)^2) + 4.3 \exp(-0.45(X - 7)^2) - 5 \exp(-0.61(X - 8)^2) + 3 \exp(-0.73(X - 8.2)^2) + 10 \exp(-0.66(X - 9.0)^2) - 7 \exp(-0.34(X - 9.5)^2) + 1.3 \exp(-0.35(X - 10)^2) - 5 \exp(-0.39(X - 10.5)^2) + 12 \exp(-0.57(X - 12)^2) + 10 \exp(-1.66(X - 13.0)^2) - 3 \exp(-0.74(X - 14)^2)$. 
Figure 6.3 shows the result from the initial point 2.0 for each variable. Figure 6.4 shows the result from the same initial point.

![Graph showing original data and fitted curve](image)

Fig. 6.1. Fminsearch, Error 17.74, Elapsed Time 34 Seconds.

### 6.2 Managing the EM Algorithm

The Expectation Maximization (EM) [27] [28] [44] algorithm is an interactive optimization method for parameter estimation. As mentioned in Chapter 4, one popular application of the EM algorithm is to estimate mixture gaussian model. Since it is essentially a local search algorithm, the pure EM algorithm has two drawbacks. First, the result greatly depends on the initial values of the parameters. Secondly, the algorithm usually reaches a local minimum of the likelihood function.
Fig. 6.2. MGO, Error 0.15, Elapsed Time 12 Seconds.

Fig. 6.3. Fminsearch, Error 61.42, Elapsed Time 212 Seconds.
Fig. 6.4. MGO, Error 0.97, Elapsed Time 25 Seconds.
6.2.1 Introduction

The mixture density parameter estimation is one of the most widely used applications of the EM algorithm. A mixture model provides a more accurate description for a heterogeneous population than a single model. For a mixture model, we assume the density function has the following form:

\[ p(x|\theta) = \sum_{i=1}^{k} c_i p_i(x|\theta_i) \]

where \( \theta = (\theta_1, \theta_2, ..., \theta_k) \), and \( \sum_{i=1}^{k} c_i = 1 \). Each \( p_i \) is a density function with parameter vector \( \theta_i \).

For a given data set \( X = x_1, x_2, ..., x_n \), the log-likelihood function is computed by

\[ \log(L(\Theta|X)) = \log \prod_{i=1}^{n} p(x_i|\Theta) = \sum_{i=1}^{n} \log \left( \sum_{j=1}^{k} c_j p_j(x_i|\theta_j) \right) \]

The goal is to find the set of parameters \( \Theta \) which maximize the log-likelihood. In most cases, this equation is very difficult to optimize. Fortunately, for the mixture gaussian model, it is tractable. According to [44], the updating rules for the parameters are as follows:

for \( l = 1 \) to \( k \)

\[ c_i^{g+1} = \frac{1}{n} \sum_{i=1}^{n} p(l|x_i, \Theta^g) \]

\[ \mu_i^{g+1} = \frac{\sum_{i=1}^{n} x_i p(l|x_i, \Theta^g)}{\sum_{i=1}^{n} p(l|x_i, \Theta^g)} \]

\[ \Sigma_i^{g+1} = \frac{\sum_{i=1}^{n} p(l|x_i, \Theta^g)(x_i - \mu_i^{g+1})(x_i - \mu_i^{g+1})^T}{\sum_{i=1}^{n} p(l|x_i, \Theta^g)} \]

In this application, we use the MGO algorithm in a different way. We do not use the MGO algorithm to replace the EM algorithm, but on top of the EM algorithm. The EM algorithm operates as the object function for the MGO algorithm. The parameters \( \Theta_0 \) will be initialized in the MGO algorithm, then a generation of \( \Theta_i \) is created by the mixture gaussian model. The EM algorithm works on each \( \Theta_i \) and
returns back to the MGO algorithm. The MGO algorithm then chooses the best $\Theta_i$ based on the average log-likelihood value as the initial value for the next generation.

### 6.2.2 Experimental Results

Here, we generate the testing data from the mixture model $Y = 0.2 \frac{1}{0.5 \sqrt{2\pi}} e^{-\frac{(x-2)^2}{2\times0.5^2}} + 0.3 \frac{1}{0.4 \sqrt{2\pi}} e^{-\frac{(x-10)^2}{2\times0.4^2}} + 0.5 \frac{1}{0.1 \sqrt{2\pi}} e^{-\frac{(x-6)^2}{2\times0.1^2}}$. Figure 6.5 shows that the pure EM algorithm suffers from bad initial points. But if we use the MGO algorithm on top of the EM algorithm, even from a bad initial point, it still can escape from the local minimum as observed in Figure 6.6. If the initial point is good, the EM algorithm is sufficient by itself as seen in Figure 6.7 and 6.8. One serious drawback of this approach is that it consumes a lot of computation time. But it is still interesting to see an optimal solution of the mixture gaussian estimation problem, and normally, there is no short cut to the global optimization problems.

### 6.3 Non-uniform Spectral Analysis

#### 6.3.1 Introduction

Traditional spectral analysis estimates the power distribution over frequency of a finite set of data [45] [41]. It has a broad range of applications in digital signal processing.

Spectrum estimation methods are generally divided into three categories [29]: nonparametric methods, parametric methods and subspace methods. Nonparametric methods work directly with the signal itself, such as the periodogram method and the Welch’s method. Parametric methods assume the signal is generated by a hypothetical linear system driven by white noise, such as the Yule-Walker autoregressive method and the Burg method. The parametric methods work better when only few data points are available. Subspace methods are based on the eigenanalysis of the
Fig. 6.5. Pure EM, Init = [0.57 0.27 0.23 0.99 0.04 0.61 0.82 0.62 0.77].
Fig. 6.6. MGOEM, Init = [0.57 0.27 0.23 0.99 0.04 0.61 0.82 0.62 0.77].
Fig. 6.7. Pure EM, Init = [0.01 1.33 0.91 1.42 1.66 0.98 1.33 0.29 1.49].
Fig. 6.8. MGOEM,Init = [0.01 1.33 0.91 1.42 1.66 0.98 1.33 0.29 1.49].
correlation matrix. These methods are very effective for sinusoidal signals, such as
the Peig algorithm [29].

Some drawbacks of these methods include 1) Require the signal to be uniformly
sampled. If the signals are sampled irregularly, most of them have difficulty. 2) Diffi-
cult to get the exact frequency components of the signals. 3) The phase information
is lost.

Spectral analysis can be viewed as an optimization problem. If we assume the data
$X = x_1, x_2, ..., x_n$ are generated by a linear combination of $k$ frequency components
plus noise, the problem can be described as finding the best parameters $\Theta$, such that
$\sum_{i=1}^{n} (f(\Theta) - x_i)^2$ is minimized. The MGO algorithm is used to optimize this total
error function.

6.3.2 Determination of the Significant Frequency Components in the Signal

Traditional spectral analysis methods do not answer the question explicitly. The
results include the frequencies from 1 to half of the sampling frequency. The user has
to manually decide which components they are interested in. For the optimization
scheme to work, the number of components $k$ has to be decided.

Although the signal $X$ is irregularly samples, the linear interpolation can be used
with the signal to estimate regularly sampled points. Then, an initial spectral analysis
using the Fast Fourier Transformation (FFT) can be used. The frequency range is
divided into many bins, only those bins whose power is above some threshold are
kept. Figure 6.9 shows the signal generated with two sinusoidal components. Figure
6.10 shows the corresponding power spectrum. In this example, if the threshold is
the average of each bin, then there are 3 potential frequency components we may
be interested in. If we decrease the threshold, we may get more potential frequency
components.
After estimating the number of potential components \( k \), we can run the optimization algorithm \( k \) times, and choose the one has the smallest total error. But one thing we have to bear in mind, this approach is working only when there is enough frequency distance between the true models. If the frequencies are very close to each other, one way we can do is to increase \( k \) gradually until we are satisfied with the result.

Fig. 6.9. Signal with Two Frequency Components, \( f_1 = 20 \text{Hz} \), \( f_2 = 30 \text{Hz} \).

6.3.3 Experimental Results

The first testing data are given by by \( X(t) = 2 \cos(2\pi f_1 t + 0.23) + 3 \cos(2\pi f_2 t + 0.51) + 0.01 N(0,1) \), where \( f_1 = 20 \text{Hz} \) and \( f_2 = 21 \text{Hz} \), the sampling rate is \( f_s = 100 \text{Hz} \). For irregular sampling purpose, the time samples were chosen as \( t(i) = \)
Fig. 6.10. Power Spectrum Using the FFT after Interpolation.
0.01 \times i + 0.1 \text{rand}. For the periodogram, the regular sampling was used. Figure 6.11 shows the results of the periodogram method given 100 data points. Figure 6.12 shows the results of the periodogram method given 50 data points. Figure 6.13 shows the results of the burg algorithm given 50 data points with regular sampling. Figure 6.14 shows the results of the burg algorithm given 33 data points with regular sampling. Figure 6.15 shows the results of the peig algorithm given 33 data points with regular sampling. Figure 6.16 shows the results of the peig algorithm given 33 data points with irregular sampling. Figure 6.17 shows the results of the MGO algorithm given 33 data points with irregular sampling.

The results show that for regular sampling, with enough data points, both the periodogram and the burg algorithm can identify the two very close frequency components. When the data points are scarce, the periodogram and the burg algorithm have difficulty to differentiate the two frequency components. For regular sampling, even with very few data points, the peig algorithm is still able to identify the two frequency components. But for irregular sampling with linear interpolation, it fails. The MGO algorithm is able to distinguish the two very close frequency components even when the data points are scarce and irregularly sampled. As in the curve fitting case, we compute the amplitudes of the components by implicit least squares method. With the MGO algorithm, the signal is actually modeled with amplitude, frequency and phase components. Hence, the phase components are also estimated.

Table 6.1 shows results of 5 more signals whose frequency and phase components are estimated with the MGO algorithm.

\text{Signal1} : f = [20, 30],
X(t) = 4 \cos (2\pi f_1 t + 0.27) + \cos (2\pi f_2 t + 0.42) + 0.01 N(0, 1);

\text{Signal2} : f = [15, 42],
X(t) = 4 \cos (2\pi f_1 t + 0.27) + \cos (2\pi f_2 t + 0.62) + 0.01 N(0, 1);

\text{Signal3} : f = [20, 32, 36],
X(t) = 2 \cos (2\pi f_1 t + 0.45) + 3 \cos (2\pi f_2 t + 0.52) + 4 \cos (2\pi f_3 t + 0.26) + 0.01 N(0, 1);

\text{Signal4} : f = [15, 20, 32, 36],
X(t) = 2 \cos (2\pi f_1 t + 0.45) + 3 \cos (2\pi f_2 t + 0.52) + 4 \cos (2\pi f_3 t + 0.26) + 3 \cos (2\pi f_4 t + 0.12) + 0.01 N(0, 1);

\text{Signal5} : f = [15, 17, 20, 25, 40],
X(t) =
2 \cos (2\pi f_1 t + 0.45) + 3 \cos (2\pi f_2 t + 0.52) + 4 \cos (2\pi f_3 t + 0.26) + 3 \cos (2\pi f_4 t + 0.12) + 2 \cos (2\pi f_5 t + 0.33) + 0.01 N(0, 1);
Fig. 6.11. The Periodogram Algorithm, \( f_1 = 20 \text{Hz}, f_2 = 21 \text{Hz}, \) 100 Data Points.
Fig. 6.12. The Periodogram Algorithm, $f_1 = 20\, Hz$, $f_2 = 21\, Hz$, 50 Data Points.
Fig. 6.13. The Burg Algorithm, $f_1 = 20\, \text{Hz}, f_2 = 21\, \text{Hz}$, 50 Data Points.
Fig. 6.14. The Burg Algorithm, $f_1 = 20 \text{Hz}$, $f_2 = 21 \text{Hz}$, 33 Data Points.
Fig. 6.15. The Peig Algorithm, $f_1 = 20\,Hz$, $f_2 = 21\,Hz$, 33 Data Points, Regular Sampling.
Fig. 6.16. The Peig Algorithm, $f_1 = 20\, Hz$, $f_2 = 21\, Hz$, 33 Data Points, Irregular Sampling.
Fig. 6.17. The MGO Algorithm, $f_1 = 20H \text{z}, f_2 = 21H \text{z}$, 33 Data Points, Irregular Sampling.
<table>
<thead>
<tr>
<th>Spectral Model</th>
<th>Frequency (Hz)</th>
<th>Phase (rad)</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal1</td>
<td>[20,30]</td>
<td>[0.27,0.62]</td>
<td>113</td>
</tr>
<tr>
<td>Signal2</td>
<td>[15,42]</td>
<td>[0.27,0.62]</td>
<td>111</td>
</tr>
<tr>
<td>Signal3</td>
<td>[20,32,36]</td>
<td>[0.45,0.52,0.26]</td>
<td>208</td>
</tr>
<tr>
<td>Signal4</td>
<td>[15,20,32,36]</td>
<td>[0.45,0.52,0.26,0.12]</td>
<td>382</td>
</tr>
<tr>
<td>Signal5</td>
<td>[15,17,20,25,40]</td>
<td>[0.45,0.52,0.26,0.12,0.33]</td>
<td>523</td>
</tr>
</tbody>
</table>

### 6.4 Summary

In this chapter, the MGO algorithm is used to solve several least square type applications. In the curve fitting problem, the advantage of the MGO algorithm over the local search algorithm is clearly observed. Then, the MGO algorithm is used for global search with the EM algorithm. Its global search ability to escape from bad initial point is demonstrated. Last, we treat the spectral analysis problem as a global optimization problem. The MGO algorithm has the ability not only to pick up the frequency features, but also the phase information. The MGO algorithm also does not suffer from the non-uniform sampling issues as most of the other algorithms do.
7. AN OPEN STRUCTURE GLOBAL OPTIMIZER

In real life problems, for example, in bioinformatics, most optimization problems are highly non-separable, which means that there are strong correlations between variables. Traditional evolutionary algorithms are not very successful on such problems. The MGO algorithm performs reasonable well on some middle size non-separable problems, such as $f_1$, $f_3$, and $f_{12}$ previously discussed. However, with very high dimensional problems, the performance is still not satisfactory. In the evolutionary computation area, there are many variations. Each algorithm has its own strengths and weaknesses, as shown later in this chapter. At the same time, computer resources are more and more affordable. Multiple CPU computers and computer clusters are not the privileges of super computer centers anymore. To take advantage of this, a global optimizer which can integrate the advantages of different evolutionary algorithms and work together to solve complicated problems is very attractive. In this chapter, an open structure global optimizer based on popular evolutionary algorithms and the MGO algorithm is proposed.

7.1 Strengths and Weaknesses of Evolutionary Algorithms

There is no global optimization algorithm which outperforms all the other algorithms in all cases. In the following experiments, three algorithms were compared with a series of high dimensional problems. The MGO algorithm and the GA algorithm are already discussed in the previous chapters. The third algorithm comes from the evolutionary strategies category. It is called Covariance Matrix Adaptation Evolutionary Strategies (CMAES) algorithm [31]. The following section gives a brief introduction to the CMAES algorithm.
7.1.1 Introduction to the CMAES Algorithm

In [31], an intuitive idea of learning the covariance matrix of the input variables of a function is explored. By learning a full covariance matrix, and generating a new population based on the covariance matrix, the correlations between variables can be preserved and exploited. In evolutionary algorithms, an important step is "SELECTION". By selecting good individuals from the population and using them to update the covariance matrix, the gene information is inherited by the next generation.

With a covariance matrix, the next generation can be generated by

\[ x^{i+1} \sim N(m^i, (\lambda^i)^2 C^i) \sim m^i + \lambda^i N(0, C^i) \]

where the superscript is the generation number and \( N \) is a multivariate normal distribution. Since \( C \) is positive definite, there is a non-singular matrix \( P \) such that \( C = PP' \). Consider the transformation \( Y = P^{-1}(X) \). Then, \( E[Y] = P^{-1}E[X] = 0 \) (assuming \( X \) has zero mean).

\[ \text{Cov}(Y) = E[YY'] = P^{-1}E[XX'](P^{-1})' = P^{-1}PP'(P^{-1})' = I. \]

So \( x^{i+1} \sim N(m^i, (\lambda^i)^2 C^i) \sim m^i + \lambda^i N(0, C^i) \sim m^i + \lambda^i PN(0, I). \)

Choosing the Mean: In each generation, we first rank all the individuals based on their fitness values, and then the best \( \mu \) individuals are chosen to compute the new mean. To favor good individuals, a weighted average can be used.

\[ m^i = \sum_{t=1}^{\mu} w_t x^i_t \]
\[ \sum_{t=1}^{\mu} w_t = 1, \quad w_1 \geq w_2 \geq \ldots \geq w_\mu \geq 0 \]
\[ f(x_1) \leq f(x_2) \leq \ldots \leq f(x_\mu) \]

By choosing the good individuals to update the mean, the search is guided in a good direction.

Updating the Covariance Matrix: We can estimate the covariance matrix from the sample population:

\[ C_\mu = \frac{1}{\mu-1} \sum_{i=1}^{\mu} (x_i - \frac{1}{\mu} \sum_{j=1}^{\mu} x_j)(x_i - \frac{1}{\mu} \sum_{j=1}^{\mu} x_j)^T \]
\[ = \frac{1}{\mu-1} \sum_{i=1}^{\mu} (x_i - \mu)(x_i - \mu)^T \]
Based on the same reasoning as before, we can use a weighted estimation to give more favor to better individuals:

\[ C_\mu = \sum_{i=1}^{\mu} w_i (x_i - m)(x_i - m)^T \]

To make the estimation more stable, we can combine the estimation with information from previous generation:

\[ C^{i+1} = (1 - \alpha)C^i + \alpha \sum_{j=1}^{\mu} w_j (x_j - m^i)(x_j - m^i)^T \]

In each generation, ideally, a good individual contains some sort of correlation information about the tightly coupled variable. By choosing the best \( \mu \) individual to update the covariance matrix, after a period of time, hopefully, the covariance matrix can represent the correlations between variables.

**Updating the Step Size \( \alpha \):** The general idea is to make the step size adaptive according to the direction of the search. After a series of successful moves, the step size can be increased to speed up the convergence; otherwise, the step size should be decreased.

### 7.1.2 Comparison of the MGO, GA, and CMAES Algorithms

The performances of the MGO, GA [30] and CMAES [46] algorithms on the function \( f_9, f_{10}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16} \), and \( f_{17} \) \(^1\) were reported here. The dimension of each function is set to 100. 5 runs of each algorithm were conducted. For fairness, each algorithm was given the best parameters to our knowledge.

These testing functions are very difficult in high dimensions. They cover a broad range of problems, from non-separable function, separable function to skewed function. Except for the function \( f_9 \), all the global minima are 0. The results in Tables 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, and 7.8 show no clear cut winner for all the problems.

The MGO algorithm works well on the functions \( f_9, f_{10}, f_{13}, f_{14}, f_{15}, \) and \( f_{16} \). The CMAES algorithm works well on the functions \( f_{12}, f_{13}, f_{14}, \) and \( f_{17} \). The GA algorithm works well on the functions \( f_9, f_{12}, f_{13}, f_{14}, \) and \( f_{16} \).

\(^1\)These testing functions are given in Appendix A
Table 7.1  
Performance on the Function $f_9$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>-4.19E+04</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-4.19E+04</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-4.19E+04</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-4.19E+04</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-4.19E+04</td>
<td>60</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>-2.16E+04</td>
<td>919</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-2.11E+04</td>
<td>911</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-2.03E+04</td>
<td>911</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-5.76E+03</td>
<td>912</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-2.18E+04</td>
<td>907</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>-4.19E+04</td>
<td>527</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-4.19E+04</td>
<td>527</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-4.19E+04</td>
<td>528</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-4.19E+04</td>
<td>527</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-4.19E+04</td>
<td>527</td>
</tr>
</tbody>
</table>
Table 7.2
Performance on the Function $f_{10}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>1.32E-08</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.97E-08</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.35E-08</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.86E-08</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.77E-08</td>
<td>110</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>1.99E+00</td>
<td>971</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.98E+00</td>
<td>975</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.00E+00</td>
<td>969</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9.95E-01</td>
<td>971</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.00E+00</td>
<td>970</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>1.95E-07</td>
<td>489</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.28E-07</td>
<td>491</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.99E-07</td>
<td>488</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.92E-07</td>
<td>489</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9.95E-01</td>
<td>491</td>
</tr>
</tbody>
</table>
Table 7.3
Performance on the Function $f_{12}$

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>1.61E-02</td>
<td>978</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8.53E-05</td>
<td>926</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6.16E-03</td>
<td>939</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.73E-03</td>
<td>951</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.02E-09</td>
<td>782</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>0.00E+00</td>
<td>707</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.00E+00</td>
<td>703</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.00E+00</td>
<td>713</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.00E+00</td>
<td>710</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.00E+00</td>
<td>709</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>3.43E-08</td>
<td>497</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.72E-08</td>
<td>496</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.93E-08</td>
<td>495</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.92E-08</td>
<td>497</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.81E-08</td>
<td>496</td>
</tr>
</tbody>
</table>
Table 7.4
Performance on the Function $f_{13}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>4.35E-18</td>
<td>237</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.60E-18</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4.54E-18</td>
<td>228</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.26E-18</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.88E-18</td>
<td>188</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>1.17E-24</td>
<td>2533</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.95E-25</td>
<td>2578</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.31E-24</td>
<td>2623</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.17E-24</td>
<td>2478</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.07E-24</td>
<td>2415</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>2.42E-09</td>
<td>518</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.43E-09</td>
<td>520</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.53E-09</td>
<td>517</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.02E-09</td>
<td>520</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.44E-09</td>
<td>519</td>
</tr>
</tbody>
</table>
Table 7.5
Performance on the Function $f_{14}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>1.30E-17</td>
<td>223</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.14E-17</td>
<td>273</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.37E-17</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.60E-17</td>
<td>234</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9.68E-18</td>
<td>256</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>3.33E-24</td>
<td>2321</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.82E-24</td>
<td>2454</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.73E-24</td>
<td>2473</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.64E-24</td>
<td>2623</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.24E-24</td>
<td>2477</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>1.10E-02</td>
<td>518</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8.32E-05</td>
<td>521</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.34E-09</td>
<td>520</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.42E-08</td>
<td>520</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.49E-08</td>
<td>521</td>
</tr>
<tr>
<td>Test Function ($n = 100$)</td>
<td>No. of Run</td>
<td>Minimum</td>
<td>Time (seconds)</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>6.45E-08</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5.71E-08</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6.45E-08</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7.24E-08</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9.30E-08</td>
<td>150</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>9.95E+01</td>
<td>1040</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9.95E+01</td>
<td>1033</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9.95E+01</td>
<td>1050</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9.95E+01</td>
<td>1048</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9.95E+01</td>
<td>1038</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>9.95E-01</td>
<td>504</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9.96E-01</td>
<td>504</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7.43E-04</td>
<td>505</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.99E+00</td>
<td>503</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.98E+00</td>
<td>504</td>
</tr>
</tbody>
</table>
Table 7.7
Performance on the Function $f_{16}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>1.20E-10</td>
<td>274</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.35E-10</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.27E-10</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.20E-10</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.20E-10</td>
<td>271</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>2.17E+04</td>
<td>955</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.15E+04</td>
<td>960</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.95E+04</td>
<td>965</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.86E+04</td>
<td>965</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.81E+04</td>
<td>958</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>7.56E-07</td>
<td>533</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.06E-06</td>
<td>532</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.53E-07</td>
<td>533</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8.86E-07</td>
<td>534</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.06E-06</td>
<td>532</td>
</tr>
</tbody>
</table>
Table 7.8
Performance on the Function $f_{17}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGO Algorithm</strong></td>
<td>1</td>
<td>4.82E-03</td>
<td>2366</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.49E-03</td>
<td>2361</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5.56E-03</td>
<td>2364</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4.47E-03</td>
<td>2358</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5.23E-03</td>
<td>2363</td>
</tr>
<tr>
<td><strong>CMAES Algorithm</strong></td>
<td>1</td>
<td>9.26E-05</td>
<td>684</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8.80E-05</td>
<td>684</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.93E-05</td>
<td>733</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9.65E-05</td>
<td>695</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9.66E-05</td>
<td>691</td>
</tr>
<tr>
<td><strong>GA Algorithm</strong></td>
<td>1</td>
<td>5.49E+00</td>
<td>667</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.00E+00</td>
<td>668</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.90E+00</td>
<td>669</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4.55E+00</td>
<td>667</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.98E+00</td>
<td>667</td>
</tr>
</tbody>
</table>
Another thing worth mentioning is the lack of robustness for some of the algorithms. For example, on function $f_{10}$, in one run, the CMAES algorithm could locate the global minimum easily, but in another run, it was stuck in a local minimum. The same thing happened with the GA algorithm on test functions $f_{10}$ and $f_{14}$.

7.2 An Open Structure Global Optimizer

From the previous section, we know that every evolutionary algorithm has its own limitation. For a set of problems, there is no clear cut winner. But for any of the tested functions, there is always one algorithm which works well. It is natural to have some way to utilize the strength of each algorithm. Furthermore, all the evolutionary algorithms have the concepts of POPULATION and SELECTION. When one algorithm finds a good solution, if it exchanges the information with other algorithms, they surely can help each other.

Evolutionary algorithms have a huge variation of flavors. Most of them have their own parameters which need to be tuned. It will be very difficult to mix them together. At the same time, once in a while, newer, better algorithms come along. To take full advantage of these trends, an open structure platform is very attractive. A new algorithm can be plugged in and it will contribute with very little change. At the same time, with the rapid progress of the computer hardware, multiple cpu computers, and computer clusters are more and more affordable in an ordinary lab. A parallel global optimizer which takes advantage of different evolutionary algorithms is proposed in this section. Our main goal is to solve a large set of problems with a uniform interface algorithm. Secondly, the structure has to be open, so we can take advantages of the newer, better algorithms.

7.2.1 The Algorithm and Its Structure

We want to keep each specific algorithm as intact as possible. There are two reasons for this. One is that each algorithm is tuned by the authors to work best
on certain type of problems. It is hard to pick up the best set of parameters for
calgorithms which are written by others. Secondly, the structure is desired to be open.
New algorithm can be plugged into this system without change of the other individual
algorithms.

![Diagram of algorithm structure]

Fig. 7.1. The Structure of the Algorithm.

Figure 7.1 shows the essential structure of this algorithm. The algorithm itself is
described as follows:

1. Initialize each individual evolutionary algorithm and the time_{interval}
2. If stop criteria is false, go to step 3, otherwise go to step 4
3. After every time_{interval}, collect the current result from each individual algorithm,
find the best one and send it back to each individual algorithm. Each individual
algorithm uses this new information to update its internal state.
4. Go to step 2
4. Exit

After every time_{interval}, each individual algorithm pauses to send its current result
to a manager processor. After finding the best result so far, the manager processor
sends the best result to each individual algorithm. Then each individual algorithm
tries its best to utilize this information, and update its internal status. By doing
this, we hope each individual can start from a better starting point. The parameter
time\text{interval} \) determines how often each algorithm communicates with the manager
processor. If it is too short, each algorithm may not progress much. If it is too
long, some algorithm may not benefit from this collaboration. The dimension of
the problem also plays a role in choosing the value for \( time\text{interval} \). For very high
dimensional problems, each object function calculation consumes significant time, so
the \( time\text{interval} \) has to take this into account. Our approach is to use a polynomial
function which has the dimension as the variable to decide the appropriate \( time\text{interval} \).
The parameters of the polynomial were determined by some simulations. For example,
for test function \( f_{10} \), the polynomial is \( time\text{interval}(x) = 0.0000068.x^3 - 0.001896.x^2 +
0.409586.x - 9.79183 \), where \( x \) is the dimension of the problem. If dimension is 30,
\( time\text{interval} \) is 1 seconds. If dimension is 100, \( time\text{interval} \) is 19 seconds. If dimension
is 300, \( time\text{interval} \) is 125 seconds.

Most evolutionary algorithms do not work very well on high dimensional problems.
The main reason is that they have fixed population size. For small toy problems, the
population size may be good enough. However, when the dimension increases, small
population size is not enough to sample the search space. This is like finding a needle in
a sea. The MGO algorithm increases the population size when the dimension goes up.
So it does not suffer from the dimension dilemma as most other algorithms. When the
dimension increases, to sample the search space thoroughly, the population size should
increase exponentially. But in reality, we could not afford this due to computational
overload. In this section, based on experimental knowhow, we propose a polynomial
function to decide the population size. For example, for the GA algorithm on test
function \( f_{10} \), the polynomial is \( pop\text{size}(dim) = -0.0000323.dim^3 - 0.02553.dim^2 +
0.24535.dim + 35.28245 \). If dimension is 30, \( pop\text{size} \) is 65. If dimension is 100, \( pop\text{size} \)
is 283. If dimension is 300, \( pop\text{size} \) is 1535.
7.2.2 The Parallel Environment

We initially implemented MPI together with the Matlab engine to create a parallel environment. This further evolved into the use of the distributed computing toolbox and the distributed computing engine toolbox [29] in Matlab. In the Matlab implementation [29], Figure 7.2 describes a typical parallel computing structure. A client sends its parallel job to the jobmanager. The jobmanager sends its sub-task to its workers. Each worker can communicate with each other and sends the results back to the jobmanager. In the end, the jobmanager sends the final results to the client.

![Figure 7.2. The Matlab Parallel Computing Toolbox.](image)

7.2.3 Experimental Results

In the experiments, we used three evolutionary algorithms. One is the MGO algorithm, the other is a genetic algorithm toolbox [30], and the third one is the CMAES algorithm [31]. The MGO algorithm works best with separable object functions. The CMAES algorithm works best with non-separable object function. The GA toolbox works well on problems in between.
For comparisons, we used the same set of testing functions as in the previous section. The dimension is set as 100 as before. The results of 5 runs on each testing function are reported here. Table 7.9, 7.10, 7.11, 7.12, 7.13, 7.14, 7.15, and 7.16 show the results on the functions \( f_9, f_{10}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16}, \) and \( f_{17}. \)

We can clearly see that, when the three algorithms work together, they have no difficulty solving all the testing functions. The system is very robust. Among all the 5 runs, it always locates the global minima. Since there is some communication overhead, the computation time is a little longer than the best time we found when we run each algorithm independently.

Another merit of this system is that it has an open structure. New evolutionary algorithms can be plugged into this system with very little modification.

<table>
<thead>
<tr>
<th>Test Function ((n = 100))</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( MGO + GA + ES ) Algorithm</td>
<td>1</td>
<td>-4.19E+04</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-4.20E+04</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-4.20E+04</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-4.21E+04</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-4.19E+04</td>
<td>105</td>
</tr>
</tbody>
</table>

7.3 Summary

In this chapter, a parallel global optimization system was proposed. It takes the advantage of each individual evolutionary algorithm. By putting different evolutionary algorithms, they collaborate together to solve the problems, rather than claiming one is better than the others. Each evolutionary algorithm has its own strength to solve a certain type of problems. When they work together, they can solve a broad
Table 7.10
Performance of the Global Optimizer on the Function $f_{10}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MGO + GA + ES$ Algorithm</td>
<td>1</td>
<td>1.06E-08</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.56E-09</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.19E-09</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.60E-09</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.91E-08</td>
<td>118</td>
</tr>
</tbody>
</table>

Table 7.11
Performance of the Global Optimizer on the Function $f_{12}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MGO + GA + ES$ Algorithm</td>
<td>1</td>
<td>6.66E-16</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.82E-14</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.04E-14</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.18E-14</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.67E-09</td>
<td>106</td>
</tr>
</tbody>
</table>

Table 7.12
Performance of the Global Optimizer on the Function $f_{13}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MGO + GA + ES$ Algorithm</td>
<td>1</td>
<td>1.67E-11</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.04E-12</td>
<td>292</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9.99E-13</td>
<td>297</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.43E-12</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.14E-12</td>
<td>290</td>
</tr>
</tbody>
</table>
Table 7.13
Performance of the Global Optimizer on the Function $f_{14}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MGO + GA + ES$ Algorithm</td>
<td>1</td>
<td>1.17E-10</td>
<td>230</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.13E-10</td>
<td>236</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.00E-10</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8.24E-11</td>
<td>231</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.23E-10</td>
<td>232</td>
</tr>
</tbody>
</table>

Table 7.14
Performance of the Global Optimizer on the Function $f_{15}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MGO + GA + ES$ Algorithm</td>
<td>1</td>
<td>1.83E-08</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.40E-08</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.45E-08</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.21E-08</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.59E-08</td>
<td>160</td>
</tr>
</tbody>
</table>

Table 7.15
Performance of the Global Optimizer on the Function $f_{16}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MGO + GA + ES$ Algorithm</td>
<td>1</td>
<td>2.35E-08</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5.05E-08</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.42E-07</td>
<td>663</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.21E-08</td>
<td>201</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.07E-08</td>
<td>296</td>
</tr>
</tbody>
</table>
Table 7.16
Performance of the Global Optimizer on the Function $f_{17}$.

<table>
<thead>
<tr>
<th>Test Function ($n = 100$)</th>
<th>No. of Run</th>
<th>Minimum</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGO + GA + ES Algorithm</td>
<td>1</td>
<td>4.63E-06</td>
<td>551</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.22E-06</td>
<td>715</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7.77E-06</td>
<td>697</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6.22E-06</td>
<td>716</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5.53E-06</td>
<td>714</td>
</tr>
</tbody>
</table>
range of difficult problems. In real life applications, without much prior knowledge, a very robust optimization system is much more needed than trying different algorithms one after another. Hence, the hybrid global optimizer is well suited for this purpose.
8. SUMMARY AND FUTURE WORK

The main research topic of this report is global optimization. Our approach is similar to the evolutionary algorithms. However, it is also distinct from other genetic algorithms and evolutionary strategies. This chapter first discusses the main contributions of this report. Then gives the summary of each chapters. In the end, possible future research topics are highlighted.

8.1 The Main Contributions

- Improvement of the basic Gray code Algorithm. The GCO algorithm is a deterministic algorithm. It suffers from slow convergence and getting stuck from some bad initial points. For high dimensional problems, a parallel Gray code algorithm was implemented.

- Invention of the mixture gaussians algorithm. By observing how the GCO algorithm searches through the search space, we found an approximate mixture gaussian model. With this mixture model, a novel evolutionary algorithm based on the mixture model was proposed. To our knowledge, we are the first to introduce the mixture model in evolutionary optimization.

- Theoretical explorations of the advantage of mixture model over traditional single gaussian mutation.

- Implementation of a parallel Matlab system using MPI + Matlab Engine which evolved later into a parallel environment using the Matlab distributed processing toolbox and engine.
• Successfully applied the MGO algorithm on a series of applications. We successfully applied the MGO algorithm on molecule conformation search, curve fitting, spectral analysis.

• Implemented a parallel global optimizer, which takes the advantages of different evolutionary algorithms.

8.2 Gray Code Optimization Algorithm

Gray code optimization algorithm uses the adjacent property of the gray code representation, by flipping the bits in a special way, marches through the search space toward the global optimum point. The flipping strategy makes sure the algorithm keep a balance between global exploration and local search. The algorithm has several stages. In the beginning, it uses 8 bits to represent the variables. In the later stages, it uses 16/32 bits to represent the variables. So, it roughly searches the space in the early stage, then does a fine tuning in the later stages. Even in the later stages, it still keeps some global exploration.

There are several drawbacks of the GCO algorithm. First, it is a deterministic algorithm, the final solution is greatly affected by the initial point. Secondly, in the later stages, the algorithm is very slow since much more bits are used to represent the variable. When the dimension of the problem is high, the slow convergence is more severe.

For escaping the local minima, the GCO algorithm could be started from multiple initial points in the early stage. The reason is that, in the early stage each variable only occupies few bits, the algorithm is very fast. Even start from multiple initial points, it has little performance overhead. In the end of the search, the algorithm is very slow to reach the global optimal point. A hybrid algorithm which adds a local search in the later stage is proposed. With this local search, the GCO algorithm can be cut off early. The experiments show that the hybrid algorithm not only saves the computation time, but also reaches deep optimal point.
In each generation, the GCO algorithm has only one parent. This makes the GCO algorithm very easy to be parallelized. For high dimensional problems, it takes a lot of time to compute the object function which makes the optimization a formidable task. Fortunately, the price of computer hardware keeps dropping. Computer clusters are more and more often seen in regular research labs. A parallel GCO algorithm using the MPI package has been implemented. The results show its effectiveness in some high dimensional problems.

8.3 Mixture Gaussians Optimization Algorithm

By studying the GCO algorithm, the way it searches through the space was modeled as a mixture of gaussians. Based on this observation, a novel mixture gaussians optimization algorithm was proposed. It is different from the GCO algorithm in several ways. First, it is a random algorithm, not a deterministic algorithm anymore. Secondly, the resolution is not controlled by how many bits are used to represent the variables, but is decided by the resolution of the computer. Thirdly, we have more control of the balance between global and local search.

To speed up the convergence, a novel crossover operation was introduced. During each generation, we have the information of the best value for each dimension. By composing an artificial solution from the best value per dimension, we can do the crossover between this artificial solution and the best solution so far. Experiments show that the crossover not only improves the convergence speed, but also makes the convergence much more smooth.

Intuitively, mixture gaussians favor more global search than a single gaussian. In the early stage of the search, we prefer global exploration than local search. The theoretical analysis shows that the mixture model has the advantage over a single gaussian when the initial point is far away from the optimal point. The theoretical analysis and experiments also show how the population size affects the search progress. The results justify the population concept often used in evolutionary algorithms.
8.4 Applications

Molecule conformation search is an important and difficult problem. It has a lot of applications in bioinformatics and chemistry. One molecule can assume different conformations under different conditions. The MGO algorithm was implemented in a host language SVL in a MOE environment. When the size of the molecule increases, the performance of the systematic search algorithm degrades exponentially. The random algorithm is also often stuck in local minima. On the tested molecules, the MGO algorithm successfully finds the minimal energy configuration of each molecule.

Curve fitting has a broad range of applications. The performance of normal local search algorithms greatly depends on the initial points. When the underlying model is complicated, it is easy for a local search algorithm to get stuck in a local minimum. The MGO algorithm has been successfully applied with a series of mixture radial basis function model estimations.

In the EM application, the MGO algorithm is not used directly to optimize the problem, but is used to manage another local search algorithm to avoid local minima.

There are a lot of classical algorithms in spectral analysis. We formulate spectral analysis as an optimization problem. When the sampling is irregular, most of the traditional algorithms which depend on the FFT fail to correctly estimate the frequency information buried in the signal. The MGO algorithm is able to identify the frequency information as well as the phase information.

8.5 An Open Structure Global Optimizer

There are a lot of evolutionary algorithms available. Every algorithm has its strengths and weaknesses. If different algorithms can work together, we could hope to solve a broad range of problems. By running different algorithms in parallel and jointly, we can solve a large set of difficult problems without sacrificing performance.
In the design of the parallel global optimizer, another important issue is extendability. When new, better algorithms come along, they can be plugged into the system with only little modification.

8.6 Future Research

The MGO algorithm works best when the input variables have little correlations. The crossover operator alleviates this drawback to some extend, but is still not perfect. On the contrary, the CMAES algorithm is very effective when the variables are tightly coupled together. The two algorithms have very similar structures. Conceptually, it should be possible to extract the covariance learning procedure from the CMAES algorithm, and incorporate it into the MGO algorithm.

In general, global optimization is a very formidable task. Without any domain knowledge, it will be extremely difficult to solve real optimization problems. The domain knowledge could help the initialization, guide the search direction, and fine tuning the algorithm parameters. How to represent the domain knowledge and how to incorporate it into the algorithm are very interesting future research questions.

The proposed parallel global optimizer looks very promising in practice. When incorporating new evolutionary algorithms into this system, the quality of the solution and the performance need to be tested on more practical problems to draw any solid conclusion.

Compared with traditional mathematical optimization techniques, the lack of theoretical support always casts some shadows and doubts on the evolutionary algorithms. More efforts are needed to explain the dynamics of the evolutionary algorithms including the algorithms proposed.

In addition to the molecule conformation application, there are plenty potential applications in the bioinformatics areas, such as drug docking, protein structure detection and so on which can be searched effectively with the proposed algorithms by formulating them in terms of global optimization.
LIST OF REFERENCES
LIST OF REFERENCES


APPENDICES
A. TEST FUNCTIONS

\[ f_1 = (\sum_{i=1}^{5} (i \cos ((i + 1)x_1 + i))) (\sum_{i=1}^{5} (i \cos ((i + 1)x_2 + i))), \quad x_i \in [-10, 10] \]

\[ f_2 = -e^{-0.2\sqrt{(x-1)^2+(y-1)^2}+3(\cos 2x+\sin 2y)}, \quad x, y \in [-5, 5] \]

\[ f_3 = -x_1 \sin \sqrt{|x_1 - (x_2 + 9)|} - (x_2 + 9 \sin \sqrt{|x_2 + 0.5x_1 + 9|}, \quad x_i \in [-20, 20] \]

\[ f_4 = -\sum_{i=1}^{4} c_i \exp (-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2)), \quad x_j \in [0, 1] \]

\[ c = \begin{bmatrix} 1 & 1.2 & 3 & 3.2 \end{bmatrix}, \quad a = \begin{bmatrix} 10 & 3.0 & 17 & 3.5 & 17 & 8.0 \\ 0.05 & 10 & 17 & 0.1 & 8.0 & 14 \\ 3.0 & 3.5 & 1.7 & 10 & 17 & 8.0 \\ 17 & 8.0 & 0.05 & 10 & 0.1 & 14 \end{bmatrix} \]

\[ p = \begin{bmatrix} 0.1312 & 0.1686 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5742 & 0.1091 & 0.0381 \end{bmatrix} \]

\[ f_5 = 1 + \sum_{i=1}^{5} \frac{x_i^2}{400} - \prod_{i=1}^{5} \cos (x_i), \quad x_i \in [-600, 600] \]

\[ f_6 = 50 + \sum_{i=1}^{5} [x_i^2 - 10 \cos 2\pi (x_i)], \quad x_i \in [-5.12, 5.12] \]

\[ f_7 = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4, \quad x_i \in [-10, 10] \]

\[ f_8 = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i)^2 + x_i - 1]^2, \quad x_i \in [-30, 30] \]

\[ f_9 = -\sum_{i=1}^{n} x_i \sin (\sqrt{|x_i|}), \quad x_i \in [-500, 500] \]
\[ f_{10} = \sum_{i=1}^{n} [x_i^2 - 10 \cos 2\pi (x_i) + 10], \quad x_i \in [-5.12, 5.12] \]

\[ f_{11} = -20\exp[-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}] - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e \quad x_i \in [-32, 32] \]

\[ f_{12} = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos \left(\frac{x_i}{\sqrt{100}}\right), \quad x_i \in [-600, 600] \]

\[ f_{13} = \frac{\pi}{n} 10\sin^2(\pi y_i) + \sum_{i=1}^{n-1} (y_i - 1)^2[1 + 10\sin^2(\pi x_{i+1})] + (y_n - 1)^2 + \sum_{i=1}^{n} \mu(x_i, 10, 100, 4), \quad x_i \in [-50, 50] \]

\[ y_i = 1 + \frac{1}{4}(x_i + 1), \mu(x, a, k, m) = \begin{cases} 
    k(x - a)^m & x > a \\
    k(-x - a)^m & x < a \\
    0 & \text{else} 
\end{cases} \]

\[ f_{14} = 0.1\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2[1 + 3\sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)] + \sum_{i=1}^{n} \mu(x_i, 5, 100, 4), \quad x_i \in [-50, 50] \]

\[ f_{15} = \sum_{i=1}^{n} [y_i^2 - 10 \cos 2\pi (y_i) + 10], y_i = 10x_i, \text{if} \quad x_i \geq 0, \text{otherwise,} \quad y_i = x_i, \quad x_i \in [-5.12, 5.12] \]

\[ f_{16} = 418.9828872724393n - \sum_{i=1}^{n} x_i \sin (\sqrt{|x_i|}), \quad x_i \in [-500, 500] \]

\[ f_{17} = \sum_{i=1}^{n-1} (x_i^2 + x_{i+1}^2)^{0.25}, \sin^2(50(x_i^2 + x_{i+1}^2)^{0.1}) + 1.0, \quad x_i \in [-100, 100] \]
## B. DATA USED TO APPROXIMATE THE GCO ALGORITHM WITH THE EM ALGORITHM

Table B.1
Data Set 1, the initial point is 0.6946

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.69457075714473</td>
<td>0.10702188036102</td>
<td>0.69457681213845</td>
<td>0.10701582536730</td>
<td></td>
</tr>
<tr>
<td>0.69457073246468</td>
<td>0.69457683681850</td>
<td>0.69295332550373</td>
<td>0.10863325700831</td>
<td></td>
</tr>
<tr>
<td>0.69457075481642</td>
<td>0.69457073479299</td>
<td>0.69457111477260</td>
<td>0.69457647919063</td>
<td></td>
</tr>
<tr>
<td>0.69441816925407</td>
<td>0.69310591339439</td>
<td>0.73363325715382</td>
<td>0.06957075699921</td>
<td></td>
</tr>
<tr>
<td>0.69457075667907</td>
<td>0.69457075528208</td>
<td>0.69457076495351</td>
<td>0.69457072734241</td>
<td></td>
</tr>
<tr>
<td>0.69457063793544</td>
<td>0.69457123398189</td>
<td>0.69456884979610</td>
<td>0.69457838653926</td>
<td></td>
</tr>
<tr>
<td>0.69454023956660</td>
<td>0.69444868683220</td>
<td>0.69505903839484</td>
<td>0.69261763214427</td>
<td></td>
</tr>
<tr>
<td>0.70238325714655</td>
<td>0.72582075715201</td>
<td>0.56957075711563</td>
<td>0.19457075702831</td>
<td></td>
</tr>
<tr>
<td>0.69457075737756</td>
<td>0.69457075691190</td>
<td>0.69457075598058</td>
<td>0.69457075411793</td>
<td></td>
</tr>
<tr>
<td>0.69457075039264</td>
<td>0.69457075784322</td>
<td>0.69457077244338</td>
<td>0.69457074294206</td>
<td></td>
</tr>
<tr>
<td>0.69457068333742</td>
<td>0.69457056412813</td>
<td>0.69457080254670</td>
<td>0.69457127938386</td>
<td></td>
</tr>
<tr>
<td>0.69457223305818</td>
<td>0.69457032570955</td>
<td>0.69456651101228</td>
<td>0.69457414040681</td>
<td></td>
</tr>
<tr>
<td>0.6945888161775</td>
<td>0.69452836403961</td>
<td>0.69446732888335</td>
<td>0.69434525857082</td>
<td></td>
</tr>
<tr>
<td>0.69458939919588</td>
<td>0.6950768044599</td>
<td>0.69410111794577</td>
<td>0.69214799294531</td>
<td></td>
</tr>
<tr>
<td>0.68824174294440</td>
<td>0.69605424294622</td>
<td>0.71167924294986</td>
<td>0.74292924295713</td>
<td></td>
</tr>
<tr>
<td>0.68042924294258</td>
<td>0.55542924291348</td>
<td>0.80542924297169</td>
<td>0.30542924285527</td>
<td></td>
</tr>
</tbody>
</table>
Table B.2
Data Set 2, the initial point is 0.5226

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.52259895636761</td>
<td>0.18511397954661</td>
<td>0.52260787610957</td>
<td>0.18510505980465</td>
</tr>
<tr>
<td>0.52259893541285</td>
<td>0.52260789706432</td>
<td>0.52104255988287</td>
<td>0.18666145628939</td>
</tr>
<tr>
<td>0.52259895776459</td>
<td>0.52259893401586</td>
<td>0.52259836032116</td>
<td>0.52260849311077</td>
</tr>
<tr>
<td>0.52250740363321</td>
<td>0.52113411261726</td>
<td>0.56166145637670</td>
<td>0.14759895628029</td>
</tr>
<tr>
<td>0.52259895590194</td>
<td>0.52259895823025</td>
<td>0.52259896381819</td>
<td>0.52259892656528</td>
</tr>
<tr>
<td>0.52259883715832</td>
<td>0.52259847953045</td>
<td>0.52260086371624</td>
<td>0.5226058576214</td>
</tr>
<tr>
<td>0.52262947394574</td>
<td>0.52247688605508</td>
<td>0.52308723761772</td>
<td>0.52064583136715</td>
</tr>
<tr>
<td>0.53041145636942</td>
<td>0.55384895637488</td>
<td>0.64759895639671</td>
<td>0.02259895625119</td>
</tr>
<tr>
<td>0.52259895660044</td>
<td>0.52259895613478</td>
<td>0.52259895706610</td>
<td>0.52259895892874</td>
</tr>
<tr>
<td>0.52259895203445</td>
<td>0.52259896265403</td>
<td>0.52259897755519</td>
<td>0.52259894775287</td>
</tr>
<tr>
<td>0.52259888814823</td>
<td>0.52259876893894</td>
<td>0.52259900735752</td>
<td>0.52259853052036</td>
</tr>
<tr>
<td>0.52259948419468</td>
<td>0.52260139154331</td>
<td>0.52260526024058</td>
<td>0.52261283563511</td>
</tr>
<tr>
<td>0.52259756846040</td>
<td>0.52262809442417</td>
<td>0.52268912958044</td>
<td>0.52256705926791</td>
</tr>
<tr>
<td>0.52281119989297</td>
<td>0.52329948114308</td>
<td>0.52232291864285</td>
<td>0.5203697364240</td>
</tr>
<tr>
<td>0.51646354364149</td>
<td>0.52427604364331</td>
<td>0.50865104363967</td>
<td>0.53900104364695</td>
</tr>
<tr>
<td>0.60240104366150</td>
<td>0.72740104369060</td>
<td>0.97740104374881</td>
<td>0.47740104363239</td>
</tr>
</tbody>
</table>
Table B.3
Data Set 3, the initial point is 0.4449

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4448683125370</td>
<td>0.85657175999521</td>
<td>0.44485911341497</td>
<td>0.85658095913394</td>
<td></td>
</tr>
<tr>
<td>0.44486829253027</td>
<td>0.4485913343841</td>
<td>0.44251845903753</td>
<td>0.85893081265011</td>
<td></td>
</tr>
<tr>
<td>0.44486831488201</td>
<td>0.44486829020196</td>
<td>0.44486867018157</td>
<td>0.44485877581054</td>
<td></td>
</tr>
<tr>
<td>0.44495986528810</td>
<td>0.44242690630314</td>
<td>0.48393081256280</td>
<td>0.81986831264102</td>
<td></td>
</tr>
<tr>
<td>0.44486831301937</td>
<td>0.44486831441635</td>
<td>0.44486832000429</td>
<td>0.44486828275138</td>
<td></td>
</tr>
<tr>
<td>0.44486819334442</td>
<td>0.44486878939086</td>
<td>0.44486640520507</td>
<td>0.44486068315917</td>
<td></td>
</tr>
<tr>
<td>0.44483779497557</td>
<td>0.4499038286623</td>
<td>0.44438003130359</td>
<td>0.44291518755325</td>
<td></td>
</tr>
<tr>
<td>0.45268081255552</td>
<td>0.47611831256098</td>
<td>0.31986831252460</td>
<td>0.94486831267012</td>
<td></td>
</tr>
<tr>
<td>0.44486831232087</td>
<td>0.44486831278654</td>
<td>0.44486831185521</td>
<td>0.44486831371786</td>
<td></td>
</tr>
<tr>
<td>0.44486831744315</td>
<td>0.44486832489373</td>
<td>0.44486830999257</td>
<td>0.44486828019025</td>
<td></td>
</tr>
<tr>
<td>0.44486822058560</td>
<td>0.44486810137631</td>
<td>0.44486833979489</td>
<td>0.4448681663205</td>
<td></td>
</tr>
<tr>
<td>0.44486977030637</td>
<td>0.44486786295773</td>
<td>0.44486404826047</td>
<td>0.4448564186593</td>
<td></td>
</tr>
<tr>
<td>0.44487167765500</td>
<td>0.44484116007687</td>
<td>0.44490219523313</td>
<td>0.44502426554566</td>
<td></td>
</tr>
<tr>
<td>0.44526840617072</td>
<td>0.44478012492060</td>
<td>0.44380356242037</td>
<td>0.44185043741992</td>
<td></td>
</tr>
<tr>
<td>0.4379418741901</td>
<td>0.44575668742083</td>
<td>0.46138168742447</td>
<td>0.49263168743174</td>
<td></td>
</tr>
<tr>
<td>0.43013168741719</td>
<td>0.30513168738809</td>
<td>0.05513168732988</td>
<td>0.55513168744630</td>
<td></td>
</tr>
</tbody>
</table>