Estimation of Distribution Algorithms with Dependency Learning

LI, Gang

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

September 2009

Abstract

For the success of Estimation of Distribution Algorithm (EDA) for optimization, it is important to define an appropriate model to approximate the fitness landscape, and at the same time the model should simplify the problem so as to make the problem easy to solve. To tradeoff the complexity and the learning of distribution models in EDA, this thesis proposes a new framework of Estimation of Dependency and Distribution Algorithm (EDDA) to choose an appropriate learning model automatically. Basically, EDDA partitions an individual representation into separate parts such that they are independent with respect to the fitness function. The independent parts of the individual representation are evolved separately with a different distribution model each. The combination of the optima of the independent parts forms the optimum of the complete individual representation. For the problems which cannot be partitioned into completely independent parts, EDDA also maintains the information of the interdependencies between the separate parts and evolves the interdependencies. The complexity of a model is determined adaptively by the amount of the dependency information maintained in the model.

There are several advantages of EDDA over the standard Evolutionary Computation. First, partitioning the individual representation and evolving the independent parts separately reduces the size of the search space significantly. Consequently, the global optimum becomes easier to be found than in the original space. Second, important dependency information between the separate parts are maintained while the trivial ones are ignored, and so the complexity of the model is selected at an appropriate level. Third, it is easy to control the diversity and convergence of the sub-populations of the separate parts of the individual representation, because the sub-populations are of only a few dimensions. Fourth, compared to other EDAs, EDDA learns the distribution model with all the individuals in the population and with their fitness. EDDA thus estimates a better approximation of a more complete fitness landscape.

Based on the framework of EDDA, four algorithms have been developed for different problems.

A new Genetic Algorithm with Independent Component Analysis (GA/ICA) is proposed for unconstraint function optimization. GA/ICA uses ICA to project the original space into a new space such that the new dimensions are independent from each other with respect to the fitness function. Dividing a solution into independent parts and evolving the parts separately clearly makes the problem easier than evolving in the original space. The experiments show that GA/ICA requires much less function evaluations to produce optimal or close-to-optimal solutions which are better than or comparable to those produced by Orthogonal Genetic Algorithm on the benchmark problems.

A parallel development with GA/ICA is a novel Instruction Matrix based Genetic Programming (IMGP) is designed to evolve programs for problem solving. IMGP evolves instructions separately and at the same time maintains the interdependencies between the instructions in the form of subtrees. It can be shown that IMGP actually evolve some schemata directly, and thus it is efficient and effective in searching the global optimum. The experimental results verify that IMGP outperforms the canonical Genetic Programming and other related algorithms on both the benchmark Genetic Programming problems and classification problems.

EDDA is then applied to an important bioinformatics problem, i.e., computational motif discovery in DNA sequences. Estimation of Distribution Algorithm for Motif Discovery (EDAMD) employs a Gaussian distribution to model the distribution of the motif consensuses in the population. The Gaussian distribution is able to capture the bi-variate linear dependencies between the motif positions. A fast local search method is used to find a set of motif instances from a motif consensus sampled from the Gaussian distribution. EDAMD has achieved a better performance than other Genetic Algorithms on the testing real problems.

A new deterministic algorithm, Cluster Refinement algorithm for Motif Discovery (CRMD), is also designed for this problem. Rather than evolving a population of motif consensuses, CRMD clusters all the subsequences where each cluster has already maximized part of the objective function of motif. With the clusters, CRMD identifies the corresponding sets of motif instances by maximizing the objective function. On a variety of benchmark problems with different levels of difficulties and properties, CRMD has a better performance than the testing state-of-the-art algorithms.

摘要

要更好的將分佈估計算法（EDA）用於優化問題，需要定義一個合適的模型來擬合目標函數地圖，同時該模型也必需儘量簡單，令問題更容易得到解決。要平衡EDA中模型的複雜度和模型的學習難度，本論文提出了一個新的算法框架，相關分佈估計算法（EDDA），來自動選擇一個合適的學習模型。EDDA將一個個體分割成幾個部分，并且這些部分在目標函數的定義下互相獨立。EDDA用不同的分佈模型來分別演化這些個體的獨立部分。這些獨立部分的最優解合起來就成了原問題的最優解。對於那些不能分割成完全獨立部分的問題，EDDA也會維持和演化不同部分之間的相關信息，所以模型的複雜度由模型中保存的相關信息所決定。

相對於標準的演化計算，EDDA有幾個優點。第一，分割演化獨立個體能大大的減小搜索空間，使找到全局最優解也就更容易了。第二，分開的部分之間的相關信息得以保存，但不重要的相關信息卻被移除了，所以模型的複雜度比較合適。第三，控制分開部分種群的多樣性和收斂性比較容易，因為分開的部分的維數和空間較小。第四，比起其他的EDA，EDDA使用種群中的所有個體和他們的目標函數值來學習分佈模型，所以EDDA能更準確的擬合更完整的目標函數地圖。

基於EDDA的框架，本論文開發了四個算法用來解決不同的問題。

本文提出了一個新的基於獨立成分分析的遺傳算法（GA/ICA）來解決無約

束的函數優化問題。GA/ICA用獨立成分分析把原始空間映射到一個新的空間，使得這個新空間的維度相對於目標函數來說彼此獨立。把個體分割成獨立的部分，同時分別演化這些獨立部分，使得問題比在原始空間中更容易得到解決。實驗結果表明，與正交遺傳算法相比，GA/ICA能夠用更少的計算量而找到更好的接近全局最優解的解。

針對遺傳規劃，本文也設計了一個獨特的指令矩陣遺傳規劃（IMGP）用來演化程序以解決問題。IMGP分別演化個體指令，同時也以子樹的形勢保存指令之間的相關性。IMGP實際上是在直接演化個體模板，所以IMGP能有效及迅速的搜索到全局最優解。實驗結果表明，在基準遺傳規劃問題和分類問題上，IMGP的表現要比標準遺傳規劃和類似的遺傳規劃算法好。

EDDA也被應用于一個重要的生物信息問題：計算識別DNA序列中的轉寫因子粘貼點。粘貼點識別的分佈估計算法（EDAMD）用高斯分佈模型來擬合種群中的粘貼片段模式的分佈。高斯分佈考慮到了粘貼片段中兩點之間的線性相關性。高斯分佈隨機產生一個粘貼片段，一個快速的局部搜索方法從這個粘貼片段中找到一組粘貼點。與其他的遺傳算法相比，EDDA在八個實際問題上的結果更好。

一個新的確定性算法，轉寫因子粘貼點識點的聚類精細化算法（CRMD），也可以解決轉寫因子粘貼點的識別問題。CRMD沒有演化一個粘貼片段的種群，它對所有的DNA子序列進行聚類，每個聚類都已經最大化了粘貼點目標函數的一部分。有了這些聚類，CRMD就能通過最大化目標函數來識別對應的粘貼點。對於不同難度和不同特點的實際問題來說，CRMD要比那些技術前沿的算法好。

Acknowledgements

It would never be possible for me to complete this thesis without the support and the help from a lot of people.

I am very honored to have Professors LEUNG Kwong Sak and LEE Kin Hong as my supervisors in my graduate study. They taught me how to do a quality research as a scholar and led me into the research fields of evolutionary computation, machine learning and bioinformatics. They offered me numerous invaluable inspirations, insights and suggestions for my work in the regular meetings in the last five years. Their supervision and guidance have become an indispensable part of this thesis.

I am sincerely obliged to my markers Professors KING Kuo Chin, WONG Tien Tsin and WONG Man Leung. They have attended all my past term presentations and gave me many useful opinions and encouragements. A lot of improvement over my original papers were based on their comments.

I am also grateful to my colleagues for their helps and encouragements in my graduate research. LAU Wai Shing, CHAN Tak Ming and WANG Jin Feng coauthored some of my papers with me, respectively. ZHANG Kun, LI Wen Ye, LIANG Yong, Ni Bing and CHEANG Sin Man had a lot of great discussion with me about my research and my papers. Thank all my friends in CUHK for their help to my research. Our precious friendships make me realize a part of the research is exchanging ideas and learning from each other.

I am deeply indebted to my father, mother and brother. They cared for me when I was a kid, and they sent me away to let me become an adult. Whenever it is and

wherever I am, they have always been supporting me with all they have and loving me with their hearts. In particular, I am dedicating this thesis to my father, who passed away two years ago. I wish he could have read this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Evolutionary Computation (EC) [51][42][61][8][99][108][37][36] is a general powerful optimization framework to solve a large amount of optimization problems which are not amenable to other deterministic *ad hoc* algorithms. By exploiting existing solutions and exploring the solution space in parallel, EC is able to find an optimal or close-to-optimal solution within a reasonable amount of time. Incorporating the statistics of the existing solutions and the properties of the solution space in the searching, EC becomes more effective and efficient to find the optimal solution.

1.1 Optimization

Optimization is indispensable in solving many practical problems. The objective of a problem can be formulated as an explicit or implicit function, and the minimum of the objective function is the best solution of the problem. In Machine Learning, learning a classifier or a regressor is formulated as minimizing the error between the predicted response and the true response. In Artificial Intelligence, training a robot is formulated as minimizing the number of the pellets missed and the times it hits a wall. In bioinformatics, motif discovery is to identify the DNA subsequences with minimal value of a property function.

Ideally, if the objective function of the problem is an explicit mathematical function, it is possible to find its optimum using standard mathematical procedures, such as linear programming [27] and more general convex optimization [19]. For other problems, *ad hoc* algorithms are developed. For example, the objective function in Support Vector Machine [107] is a constrained convex function, whose global minimum is guaranteed, and many algorithms are developed to solve it.

However, it may be difficult to find the global optimum of the objective function in various other cases. In Neural Network [13], the objective function is non-convex, and so only a local optimum can be found. In some high dimensional problems, the search space of the solution is extremely large, and so finding its global optimum is very time consuming. For the notorious NP-hard problems [57], it is impossible to find the global optima with less than exponential time.

To make things worse, the objectives of some problems cannot be formulated as explicit mathematical functions. Therefore, it is very difficult, if not impossible, to find their solutions using only mathematical methods. Other non-mathematical procedures are needed to optimize these type of problems. For example, in robot training [61], a series of controlled actions are to be implemented to collect all the pellets while avoiding hitting the walls in a room. In motif discovery [25], the minimum of the property function can only be attained by trying different combinations of subsequences.

1.2 Evolutionary Computation

Evolutionary Computation (EC) [51][42][61][8][99][108][37][36] is a powerful optimization framework for general purpose. For the aforementioned problems which pose difficulty for deterministic approaches, EC is frequently the last resort to find its optimal or close-to-optimal solution.

EC searches for the solution by evolving a population of individuals which represent candidate solutions to the problem. Evolving, as its analogy in the nature, means fitter solutions survive and produce new solutions. An individual consists of the genes corresponding to the components of the problem solution. A fitness function, i.e., the objective measure, is applied on an individual to evaluate the extent to which the individual solves the problem. The population is initialized with the individuals randomly sampled in the search space. During the evolution, the existing individuals crossover and mutate to exchange the information and generate new offsprings. The individuals with better fitness have more chances to crossover and mutate, and their offsprings are likely to be better than the offsprings generated by the individuals with worse fitness. By generating better and better solutions, EC may find the global optimum eventually.

Employing a population of individuals to search the solution space in parallel, EC covers a large portion of the solution space and therefore EC has a relatively better chance to locate the global optimum. Exploiting the existing good solutions to generate new solutions, EC is actually guided with a heuristic to find better and better solutions during the evolution, and hence it is much more efficient than pure random search. Schema theory also explains that EC evolves numerous schemata, i.e., solution patterns, simultaneously with a limited population, and therefore it is more effective than deterministic approach to uncover the pattern of the global optimum.

EC can be used to solve those problems mentioned in Section 1.1. For the high dimensional problems, finding their global optima using mathematical approaches may induce an inhibitive time and/or space complexity, but EC is a good alternative to find a close-to-optimal solution with a reasonable time. For the non-convex problems with many local optima, such as the learning objective function in Neural Network, EC is able to locate multiple local optima which may include the global optima. For the NP-hard problems with the exponentially growing search space, EC can be applied to find a satisfactory solution given a sufficient amount of time. Even for the problems whose objectives are not mathematical functions, EC can still be used to find an optimal solution as long as there is a fitness function to evaluate the solution.

1.3 Estimation of Dependency and Distribution Algorithm

Estimation of Distribution Algorithm (EDA) [67] calculates the statistics of the population of the individuals, and uses the statistics to guide the application of the genetic operators in the evolution. The original crossover and mutation in the standard EC are totally random, and thus they may not always generate good offsprings in the evolution. Since the population of the individuals covers a part of the fitness landscape, EDA indirectly learns the properties of the fitness landscape by building a distribution model of the population. With the information of the fitness landscape, the genetic operators are less random and more likely to generate good offsprings.

The distribution model used is a key part to the success of EDA. According to the no free lunch theorem [121], there is no universal distribution model that fits all kinds of fitness landscapes. Therefore, a variety of distribution models have been proposed, which involves different properties of the fitness landscape. Complicated models may capture many and subtle properties of the fitness landscape, and thus the genetic operators are better directed in generating good individuals. However, given a limited size of the population, it is difficult to learn a complicated model accurately and thus its advantage is difficult to be realized. On the other hand, a simple distribution model is easy to estimate with a limited number of individuals. Nevertheless, a simple model may be insufficient to approximate a complicated fitness landscape, and thus its benefit to the genetic operators may be trivial.

To tradeoff the complexity and the learning of distribution models in EDA, this thesis proposes a framework of Estimation of Dependency and Distribution Algorithm (EDDA) to choose an appropriate model automatically. Basically, EDDA partitions an individual representation into a few parts such that they are independent with respect to (w.r.t.) the fitness function. The independent parts of the individual representation are evolved separately with a different distribution model each. The combination of the optima of the independent parts forms the optimum of the complete individual representation. For the problems which cannot be partitioned into the completely independent parts, EDDA also maintains the information of the interdependencies between the separate parts and evolves the interdependencies along with the independent parts. The complexity of a model is controlled adaptively by the amount of the independent and interdependent information kept in the model.

There are four major advantages of EDDA over the standard EC,

First, partitioning the individual representation and evolving the independent parts separately may reduce the size of the search space. In the worse case, the search space of all the dimensions in EC is the cartesian product of the individual dimensions, and the size of the complete search space is the product of the sizes of the dimensions. In the best case, the size of the search space in EDDA is the sum of the sizes of the search spaces of the independent parts.

Second, important interdependencies between the separate parts are maintained while the trivial ones are ignored. In EDA, a complicated model may maintain a large amount of information of the interdependencies between the genes. However, some of such interdependency information may be unnecessary, and there are too many parameters to estimate accurately. In EDDA, it is possible to find a proper balance between the estimation accuracy and the model complexity.

Third, it is relatively easy to control the diversity and the convergence of the populations of the separate parts of the individual representation. Diversity and convergence affects how much the solution space is searched directly. In high dimensional space, the population in the standard EC sometimes covers only a small and sparse area in the search space, and it is relatively difficult to manipulate the size and the density of the covered search area. In EDDA, because an independent part of the individual representation consists of only a few dimensions, it may be easier to control the diversity and convergence in such a relatively small search space.

Fourth, compared to some EDAs, EDDA learns the distribution model with all the individuals and their fitness in the population. Therefore, it possible for EDDA to estimate a better approximation of a more complete fitness landscape. On the contrary, some EDAs discard the individuals of bad fitness, and use only the good individuals for model estimation. Consequently, the resulted distribution may be misleading in the area of bad individuals, and thus distorted on the complete fitness landscape.

This thesis proposes and implements four algorithms developed under the framework of EDDA. The thesis structure is as follows,

Chapter 2 introduces the background research related to EDDA in the literature.

EDDA is first employed in Genetic Algorithm (GA) [51][42] to optimize objective functions by converting the problem solution into some independent parts and evolving the independent parts separately. **Chapter 3** describes a new Genetic Algorithm based on Independent Component Analysis (GA/ICA) for unconstrained global optimization of continuous functions. GA/ICA uses Independent Component Analysis [55] to linearly transform the original dimensions of the problem into new components which are independent from each other w.r.t. the fitness function. It projects the population on the independent components and divide the population into sub-populations along the independent components. Genetic operators are applied on the sub-populations to generate new sub-populations, which are then combined into a new population of all the dimensions. In other words, GA/ICA uses GA to find the optima on the independent components, and combines the optima as the global optimum for the problem. The experiment results verify that GA/ICA produces optimal or close-to-optimal solutions better than or comparable to those produced by some of other GAs and it requires much less fitness evaluations of individuals.

EDDA can also be used in Genetic Programming (GP) [61][8] to speed up the GP evolution by evolving the GP instructions and their interactions simultaneously. **Chapter 4** presents a novel Instruction Matrix based Genetic Programming

(IMGP). IMGP maintains an Instruction Matrix (IM) to store the information of tree nodes and sub-trees. The tree nodes are evolved independently and their interdependencies are maintained in the form of subtrees. IMGP extracts program trees from IM, and updates IM with the information of the extracted program trees. As IM actually keeps part of the information of the schemata of GP and evolves the schemata directly, IMGP is effective and efficient to find the optimal schema. The experiments on the benchmark problems have verified that the results of IMGP are not only better than those of Canonical Genetic Programming in terms of the qualities of the solutions and the number of program evaluations, but they are also better than some of the related EDA-based GP algorithms. IMGP is also used to evolve programs for practical classification problems. It has obtained higher classification accuracies than 4 other GP classification algorithms on 4 benchmark classification problems.

Chapter 5 proposes an Estimation of Distribution Algorithm for Motif Discovery (EDAMD) as an application of EDDA to solve a real bioinformatics problem. Motif discovery [25] is to find the binding subsequences of transcriptional factors, i.e., motif instances, on DNA sequences using computational methods. The consensus, i.e., the common pattern of the motif instances, and the instances of a motif are represented in 1-out-of-4 encoding to convert the problem in a continuous domain. A Gaussian distribution models the distribution of the population of the candidate motif consensus. The advantage of using a Gaussian distribution is that it captures the bivariate dependencies between the positions in a motif, and the interdependencies between the independent positions may vanish as estimated from the population. After a new motif consensus is sampled from the Gaussian distribution, EDAMD uses a greedy Gibbs sampling to find the nearest local optimum around it. The experiments show that EDAMD is better than or comparable to other algorithms on the benchmark problems.

Upon the success of EDAMD, it is redesigned as a deterministic algorithm, i.e., Cluster Refinement Algorithm for Motif Discovery (CRMD), which is more efficient and effective. Instead of evolving a population of possible candidate consensuses, all the DNA subsequences are clustered according to their information contents. The consensuses of the clusters are then served as the initial motif candidates for further refinement to locate the corresponding motif instances. The number of the motif instances is adjusted automatically by the controlling thresholds adapted to the motif consensus. CRMD has been tested on a variety of benchmark problems of a wide range of properties. The empirical results show that the clustering provides good initial consensus seeds, and the refinement procedure leads to the local optimal consensus efficiently. The qualities of the discovered solutions are compared favorably with the solutions produced by other state-of-the-art algorithms.

Chapter 6 is the conclusion of the thesis.

Chapter 2

Background

Evolutionary Computation (EC) has four major branches, Genetic Algorithm (GA) [51][42], Genetic Programming (GP) [61][8], Evolutionary Strategy [99][108] and Evolutionary Programming [37][36]. All these four branches follow the basic framework of EC: maintaining a population of individuals, selecting good individuals to crossover and mutation, and putting the offsprings in the population of the next generation. An individual consists of genes to represent a solution to the problem. The usefulness of an individual is evaluated with a fitness function. The genetic operators, i.e., selection, crossover and mutation, are used to generate new individuals in the evolution. The major differences between the four branches of EC are the individual representations, the specific mechanisms of selection, crossover and mutation.

Estimation of Distribution Algorithm (EDA) [67] is an extension of EC with statistical analysis. Generally, in each generation, EDA selects some good individuals from the population, learns the distribution of these good individuals, and then it generates a new population from the distribution. EDA has been applied to both GA and GP.

In this thesis, Estimation of Dependency and Distribution Algorithm (EDDA) is a variant of EDA for GA and GP, which are the most two important kinds of EC.

2.1 Genetic Algorithm and Genetic Programming

2.1.1 Genetic Algorithm

GA [42][51] is a kind of EC for search and optimization problems. The individual is a vector of numbers corresponding to the solution of the problem. In the view point of individual representations, GA can be categorized into two classes: the Binary Coded Genetic Algorithm and the Real Coded Genetic Algorithm.

Binary Coded Genetic Algorithm (BCGA) [51][42] is the traditional GA using binary coding. In BCGA, an individual is encoded as a vector of binary digits, i.e. 0 and 1. Simple crossover is selecting a crossover point randomly in the parents, swapping the segments before and after the crossover point of the parents, and thus producing two offsprings. Uniform crossover [113] determines the values of each gene by randomly selecting the values of the same gene from either of the parents. Other types of crossover of binary coding are reported in [34]. Mutation is selecting a mutation point randomly in the parent, and flipping the binary number of the gene on the mutation point.

Real Coded Genetic Algorithm (RCGA) [50] is suitable for problems in continuous domain. In RCGA, an individual is a vector of real numbers. Simple crossover [123][82] is the same as the one in BCGA. As RCGA uses real numbers for individual representation, complex crossovers have been developed. A typical one is the flat crossover. Suppose the parents are $C_1 = (c_1^1 \cdots c_i^1 \cdots c_n^1)$ and $C_2 = (c_1^2 \cdots c_i^2 \cdots c_n^2)$, the offspring is $H = (h_1 \cdots h_i \cdots h_n)$, where h_i is a real number randomly generated out of the interval of $[c_i^1, c_i^2]$. The simplest mutation in RCGA is the random mutation [82]. Suppose the parent is $C = (c_1 \cdots c_i \cdots c_n)$, and $c_i \in [a_i, b_i]$ is selected as the gene to be mutated, the offspring is $H = (c_1 \cdots c_i' \cdots c_n)$, where c_i' is a random real number from the domain $[a_i, b_i]$. More types of crossover and mutation can be found in [50].

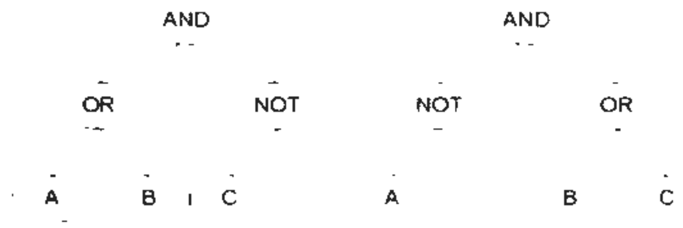


Figure 2.1: Canonical Genetic Programming: (AND OR A B NOT C) & (AND NOT A OR B C)

2.1.2 Genetic Programming

GP [61] automatically constructs computer programs as problem solutions. The individual in GP is a computer program, which receives the inputs from the problem, and gives the output as the answer. GP has successfully produced results competitive with human solutions [10]. There are several types of GP with different individual representations.

Canonical Genetic Programming (CGP) [61] is the original standard GP. It represents a program as a tree, encoded in a LISP-like *s-expression*. Fig. 2.1 shows two examples. The tree is composed of the nodes of functions and terminals. Executing the program is traversing the tree in the post-order recursively. CGP crossover is selecting two crossover points in the two parents respectively, and exchanging their subtrees at the two crossover points. CGP mutation is selecting a mutation point in the parent, and replacing its subtree with a new subtree randomly generated.

Strongly Typed Genetic Programming [84] enforces data type constraints in CGA to manipulate multiple data types. Therefore, it avoids searching in the solution space which involves inappropriate data. It also employs generic functions and generic data types to make it more powerful and practical.

Linear Genetic Programming [8] represents the program as a sequence of machine codes based on a register machine. The program receives the inputs from the registers and puts the output in a specified register. The crossover is swapping the segments of the codes between two crossover points in the two parents. Mutation is replacing a machine code with a new one randomly generated. Evaluating the program is executing the codes sequentially on the register machine.

Stack-based Genetic Programming [89] represents the program as a sequence of functions and terminals. It is executed on a stack-based virtual machine and its instruction set includes the stack operations, e.g., POP & PUSH. It uses simple two points crossover and one point mutation.

Graph Genetic Programming [91] encodes the program in a grid of functions and terminals. Some of the nodes in the grid are connected with the directed links which indicate the order of execution. A sequence of continual links forms an execution path. There can be multiple execution pathes in a grid. Executing the program is evaluating the functions and terminals following the execution pathes in parallel. Crossover and mutation are processed on the level of subgraphs.

Cartesian Genetic Programming [83] is also based on a grid of function nodes. Unlike Graph GP, the program is represented as a sequence of groups of indices. Each group of indices corresponds to a function node, and it consists of three indices for inputs and one index for the function. Crossover and mutation are used to modify the index sequence.

Genetic Parallel Programming [70] evolves a general parallel program on a Multi-Arithmetic-Logic-Unit Processor. A parallel program is composed of a series of parallel instructions, each of which consists of several parallel sub-instructions. Genetic Parallel Programming is observed to evolve parallel programs with less computational effort than their sequential counterparts.

Grammatically-based Genetic Programming [120][122] represents programs indirectly. It uses a set of grammar rules to generate a population of grammar derivation trees. Interpreting the leaves of a tree sequentially translating it into a program. It also employs some advanced mechanisms for grammar evolution, such as type control, grammar modification, merit selection, and encapsulation.

2.2 Estimation of Distribution Algorithm

2.2.1 Estimation of Distribution Algorithm for Genetic Algorithm

Population-Based Incremental Learning (PBIL) [7] maintains a vector of probabilities, each of which is the distribution of the corresponding gene of the solutions in the population. PBIL samples a population of individuals according to the distribution in the vector of probabilities, and updates the vector with the best individual in the population. PBIL is extended to incorporate the crossover operator in [106]. The extended PBIL maintains a set of probability vectors. It not only crossovers between the selected individuals, but also between the probability vectors.

Compact Genetic Algorithm [47] is similar to PBIL. However, it samples only two individuals from the probability vector, and updates the vector with the better one. Compact GA is extended to incorporate the linkage information between the genes in [46].

Univariate Marginal Distribution Algorithm (UMDA) [66] assumes that the genes of the individual are independent w.r.t. the fitness. Therefore, the joint probability density function is a product of Gaussian distributions of the independent genes as Eq. 2.1, where m is the number of the genes, μ_i is the mean of the ith gene of the good individuals, and σ_i is the standard deviation of the ith gene of the good individuals.

$$p_{N}(x;\mu,\sigma) = \prod_{i=1}^{m} p_{N}(x_i;\mu_i,\sigma_i) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2} \tag{2.1}$$

Estimation of Multivariate Normal density Algorithm (EMNA) [65] takes the pairwise dependencies between the variables into account. The joint probability density function is a multivariate normal distribution as Eq. 2.2, where μ is the mean of the good individuals, and Σ is the covariance matrix of the genes of the good individuals. This approach thus considers all the second-order moment statistics of the genes of the good individuals.

$$p_{\mathcal{N}}(x;\mu,\Sigma)=\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu)^{T}\Sigma^{-1}(x-\mu)} \tag{2.2}$$

Estimation of Mixture of Distribution Algorithm (EMDA) [67] employs multiple Gaussian distributions to model the population. A mixture of Gaussian distributions is defined as Eq. 2.3, where c is the number of distributions, $P(i)$ is the prior probability of the ith Gaussian distribution, $p(x|i)$ is the conditional distribution of x w.r.t. the ith Gaussian distribution. The model is updated with the good individuals in the population using the Expectation-Maximization procedure [85].

$$p(x)=\sum_{i=1}^{c}P(i)p(x|i)=\sum_{i=1}^{c}P(i)p_{\mathcal{N}}(x;\mu_i,\Sigma_i) \tag{2.3}$$

In the aforementioned EDA variants using Gaussian distribution, only the pairwise linear correlations are considered. It may be insufficient to model the fitness landscape correctly since other kinds of higher moment dependencies are ignored.

Bayesian Optimization Algorithm [87][88] uses (hierarchical) Bayesian network to model the joint distribution of the variables in the good individuals in order to generate new individuals. It is also able to identify, reproduce and mix the building blocks in the individual representation.

Univariate Marginal Distribution Algorithm with Independent Component Analysis (UMDA/ICA) [126] incorporates ICA [55] into UMDA to resolve the interdependence between the dimensions of the problem. First it uses ICA on the population in each generation to find the independent linear combinations of the original dimensions. Then it transforms the population from the original space into the new space defined by the independent linear combinations. Afterwards, UMDA is used on the transformed population. However, it does not explicitly estimate the distribution functions of the individuals on the independent dimensions. Instead, it crossovers the individuals in the new space and converts the offspring back into the original space as the new population. UMDA/ICA uses only some of the individuals for ICA in the evolution, so it may not find the true independent components of the complete landscape as it loses the information contained in the rest of the population. In addition, the selected individuals are treated equally in both ICA and the evolution disregarding their different fitness. Finally, it uses only crossover in the evolution of each independent component, which may be ineffective for difficult problems.

2.2.2 Estimation of Distribution Algorithm for Genetic Programming

Probabilistic Incremental Program Evolution (PIPE) [103] maintains a probability tree to evolve programs. A tree node is a vector keeping the probabilities of the functions and terminals of the node. In each generation, PIPE creates a population by constructing trees based on the probability tree, and updates the probability tree with the information of the best individual in the population. However, updating the probability tree only with the best individual without the information of the rest of the population may be insufficient to estimate the model accurately. Besides, it ignores the interdependencies between the nodes.

Competent Genetic Programming [105] combines Compact Genetic Algorithm [46] and PIPE as a multivariate probabilistic model of programs. Its significance is that it partitions a tree into subtrees, and builds a probabilistic model of each subtree. Therefore, it is not only able to calculate the probabilities of the nodes, but the probabilities of the subtrees as well. Nevertheless, it involves high computation overhead as it uses a heuristic greedy search, which calculates the complexity of each possible subtree, to identify good subtrees.

Grammar Model-based Program Evolution (GMPE) [110] evolves programs with the probabilistic context-free grammar. The grammars rules have associated production probabilities. GMPE updates the grammar rules with the good individuals in the population, and uses the grammar rules to produce new individuals. A grammar rule generates a single node or a whole subtree, and it thus is able to keep the information of both tree nodes and subtrees. A grammar rule has no position information though, and so the position of its derivative is not fixed.

The Estimation of Distribution Algorithms for Genetic Programming in [18] is similar to Grammar Model-based Program Evolution. It employs a probability distribution over grammar rules to generate new programs. Complex production rules or subfunctions can be introduced by using transformation to expand one production rule into another production rule so as to express high order of dependencies. However, learning advanced production rules with the proposed greedy algorithm may take a lot of time.

Program Evolution with Explicit Learning [109] uses Search Space Description Table (SSDT) to describe the solution space. Ant Colony Optimization [17] is the learning method to update the stochastic components of SSDT. Grammar refinement is employed to focus on the promising solution area by splitting certain rules in SSDT.

Grid Ant Colony Programming [100] applies Ant Colony Metaheuristic [31] to GP. It uses a population of ants to navigate across a grid of functions and terminals. The path traversed by an ant is translated into a program. The ant is guided by the pheromone on the connections between its current location and other nodes. The pheromone is updated as the ant passes along a connection. Furthermore, the pheromone on the tour of the best program is reinforced. In the evolution, the pheromone on a connection becomes stronger as more ants and better ants pass along it. Therefore, the ant may gradually find a series of good connections and complete a good program.

Chapter 3

Genetic Algorithm with Independent Component Analysis

3.1 Overview

EDDA is first employed in Genetic Algorithm (GA) to optimize objective functions efficiently. Genetic Algorithm (GA) [51][42] can solve the unconstrained continuous optimization problem as formulated in Definition 3.1. GA encodes the problem solution in a vector of variables as an individual, i.e. the unknown x in Definition 3.1. The objective function in Definition 3.1 evaluates the fitness of the individual. GA randomly generates a population of individuals to search in the solution space initially, focuses on the promising solution areas via genetic operators gradually, and finally converges to the global optimum.

Definition 3.1 An unconstrained continuous optimization problem is solving the following continuous objective function:

$$maximize\ f(x),\ subject\ to\ l \leq x \leq u$$

where $l \leq x \leq u$ defines the function domain, i.e. the solution space.

GA may fail to find the optima in some high-dimensional problems sometimes, because the size of the solution space grows exponentially with the dimension of the problem. To reduce the size of the solution space, a possible approach is dividing the original problem into several sub-problems by its dimensions. Afterwards GA is applied to the sub-problems to find their sub-optima separately. Finally the sub-optima are combined as the optimum of the original problem. Since a sub-problem has fewer dimensions than the original problem, its solution space is smaller than that of the original problem, so it is easier for GA to solve.

The difficulty of this approach is that the dimensions of the problem are usually interdependent on each other with respect to (w.r.t.) the fitness. In other words, the fitness of a sub-solution for a sub-problem depends on the sub-solutions for the other sub-problems. Suppose we have found the optimum for a sub-problem, if the other sub-solutions change, the original optimum might not be optimal any more. Therefore, even if we find the optima for all the sub-problems, combining them directly may not give us the optimum for the original problem.

A new Genetic Algorithm based on Independent Component Analysis (GA/ICA) is proposed to resolve this difficulty. It uses Independent Component Analysis (ICA) [55] to find a set of components which are linear transformations of the original dimensions. The components are independent from each other w.r.t. the fitness, and so the sub-solutions on the independent components do not affect each other. Afterwards, the original problem is converted into a new problem defined on the independent components. Consequently, GA/ICA can decompose the new problem into sub-problems by the independent components, and use GA to solve the sub-problems separately. There are primarily three issues to be solved to make the algorithm work,

1. ICA is a statistical method while GA is an optimization algorithm. Therefore, we need to transform the original problem into an equivalent new problem so that we can apply ICA on it.

2. When we use GA to solve the sub-problems, we need to know their fitness functions. However, we only have the fitness function for the original problem

with all the dimensions together. Therefore, we need to infer the fitness in the sub-problems from the original fitness.

3. A solution could become a local optimum on a certain dimension when it cannot increase its fitness in either direction along the dimension. Therefore, the sub-problems of a single model problem may be multi-modal, and we need to take extra care when we apply GA on the sub-problems.

The rest of this chapter is organized as follows. Section 3.2 described GA/ICA in detail and how the above three issues are solved in GA/ICA. Section 3.3 presents the experiment results on some benchmark problems. GA/ICA produces optimal or close-to-optimal solutions better than or comparable to those produced by the other GAs tested in this chapter, while GA/ICA requires much less fitness evaluations of individuals. Section 3.4 is the discussion.

3.2 Architecture

GA/ICA consists of two major stages. In the first stage, GA/ICA samples a large population of individuals uniformly in the solution space. Then it uses ICA on the population to transform the original variables into a new set of variables which are independent from each other w.r.t. the fitness. In the second stage, GA/ICA actually evolves the population to find the solution by running GA on the new independent variables separately. Since the new variables are independent, their optima do not affect each other, and the combination of the their optima is the optimum of the complete problem.

3.2.1 Independent Component Analysis

Independent Component Analysis (ICA) [55] is originally used as a data transformation method, especially for Blind Source Separation (BSS). Suppose we observe N m-dimensional data $x^t, t = 1, 2 \ldots N$, ICA tries to find a linear transformation

$y = Wx$, where W is the demixing matrix, so as to make the variables $y_i, i = 1 \cdots m$ as statistically independent from each other as possible. In BSS, ICA tries to find the mixing model $x = As$, where s is the recovered independent source signals and A is the mixing matrix. It is proved that y equals s up to a multiplicative constant and permutation. The difficulty of ICA is that neither A nor s is known beforehand. In statistics, the variables $y_1, y_2, \cdots, y_m$ are mutually independent, if their joint density function can be factorized as the product of their marginal density functions as Eq. 3.1, where $p_i(y_i)$ is the marginal density of y_i.

$$p(y) = p(y_1, y_2, \cdots, y_m) = \prod_{i=1}^{m} p_i(y_i) \tag{3.1}$$

To use ICA in an optimization problem, the optimization problem must be converted into an equivalent problem whose fitness can be regarded as the probability density. Suppose the optimization problem is as defined in Definition 3.1. Intuitively, there should be more individuals of higher fitness than the individuals of lower fitness. If the objective function $f(x)$ has a lower bound $L = inf\{f(x)|l \leq x \leq u\}$, the new fitness function is defined as $f^*(x) = f(x) - L \geq 0$. Further suppose $\int_l^u f^*(x)dx$ is the integral of $f^*(x)$ over the domain $[l, u]$, then $g(x)$ as defined in Eq. 3.2 can be treated as a probability density function as it satisfies the two conditions following its definition. Clearly, the fitness landscapes of the original and the new fitness functions are equivalent up to a translation and scaling.

$$g(x) = \frac{f^*(x)}{\int_l^u f^*(x)dx} = \frac{f(x) - L}{\int_l^u (f(x) - L)dx}, \text{ where } g(x) \geq 0 \ \& \int_l^u g(x)dx = 1 \tag{3.2}$$

It is difficult to calculate $g(x)$ because the analytical form of the integration of $f(x)$ may not exist in practice. However, given an initial population, it is still possible to generate a new population of individuals whose distribution roughly follows the probability density function $g(x)$. Note that $\int_l^u (f(x) - L)dx$ is the same for the $g(x)$ of all the individuals, and so $f(x) - L \propto g(x)$. Therefore, GA/ICA replicates each individual x^j for $\lfloor C \cdot (f(x^j - L)) \rfloor$ times, where C is an appropriate constant to make $C \cdot (f(x^j - L)) \geq 1$. This way, the copies of an individual x is approximately proportionate to its density as $g(x)$. Then ICA is applied on this new population to find the independent components satisfying Eq. 3.3, where $g'(s)$ is the joint density function defined on the independent components and $g'_i(s_i)$ is the univariate marginal density function. During the evolution, $g'_i(s_i)$ can be treated as the implicit fitness function defined on the ith independent component.

$$g(x) = g(As) = g'(s) = \prod_{i=1}^{m} g'_i(s_i) \tag{3.3}$$

GA/ICA uses ICA in a different way than UMDA/ICA does. First, it uses all the individuals in the population for ICA. Second, it uses the fitness of the individuals for the probability densities in ICA. Third, it actually runs GA on the sub-populations of the independent components.

3.2.2 Independent Evolution

After finding the independent components, GA/ICA evolves the population in the new space to find the solution. It projects the original population on the independent components and gets one 1-dimensional sub-population on each independent component, which is evolved separately. The basic steps of the evolution are shown in Algorithm 3.1 with the independent components as the inputs. At first GA/ICA randomly initializes a new population, evaluates the fitness of its individuals, and remembers the best individual in the population. Then GA/ICA evolves till 1000 generations at most or the population converges early. In each generations, it runs the following steps,

1. First GA/ICA needs to decide which genetic operator to use mostly in the current generation. Usually, crossover shrinks the solution area covered by

the population, while mutation makes the population explore a larger solution area. Because a single modal problem could induce a multimodal subproblem on an independent component, GA/ICA uses mutation as the primary genetic operator. When the best-so-far individual has not been improved for a relatively long time, we switches to crossover to focus on the neighborhood of the best-so-far individual.

2. Then GA/ICA projects the population in the original space into the new space defined by the independent components according to the ICA demixing formula $y = Wx$. It divides the population by the independent components into m 1-dimensional sub-populations.

3. In the function *estimatePop*, GA/ICA estimates the new fitness of the 1-dimensional individuals in the sub-populations. The new fitness instead of the original fitness is used in the evolution of the corresponding sub-population. The details are explained in Section 3.2.2.

4. On each of the independent components, GA/ICA samples a new 1-dimensional sub-population out of its corresponding 1-dimensional sub-population via the genetic operator it has chosen in step 1. The details of the function *icaSample* are described in Section 3.2.2.

5. After completing the evolution on all the independent components, GA/ICA combines all the new 1-dimensional sub-populations into a new m-dimensional population. Then it projects the new population back into the original space using the ICA mixing formula $x = As$, and evaluates the fitness of the individuals.

Fig. 3.1 illustrates how the 1-dimensional sub-populations are combined into a m-dimensional population. On the left, each row of the table is a sub-population. The individuals in the ith 1-dimensional sub-population are denoted as $\{s_i^1, s_i^2, \cdots, s_i^N\}$.

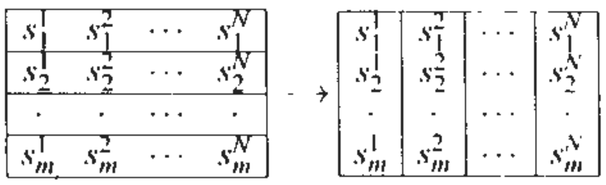


Figure 3.1: Combine the 1-dimensional sub-populations into a true m-dimension population. Row vectors on the left table are the 1-dimensional sub-populations. Column vectors on the right table are the individuals in the population

Algorithm 3.1: Evolution with the Independent Components

```
Input: W, A
Output: bestInd
pop ← initPop();
fitness ← evaluate(pop);
bestInd ← bestFunc(pop, fitness);
for gi ← 1 to 1000 do
    [pop, genOp ] ← checkOp(bestInd, pop, fitness, stagnancy);
    icaOldPop ← W × pop;
    for di ← 1 to dim do
        [$icaPop_{di}$, $icaFit_{di}$] ← estimatePop($icaOldPop_{di}$, fitness, $icaPop_{di}$, $icaFit_{di}$);
        $icaNewPop_{di}$ ← icaSample($icaPop_{di}$, $icaFit_{di}$, genOp);
    pop ← A × icaNewPop;
    fitness ← evaluate(pop);
    [stagnancy bestInd ]← checkState(pop, fitness, bestInd);
    if stagnancy > 50 then
        break;
```

On the right, each column of the table is a m-dimensional individual. The jth individual in the population is denoted as $(s_1^j s_2^j \cdots s_m^j)^T$.

Fitness Estimation

When GA/ICA performs GA on the 1-dimensional sub-population on the independent component s_i, it needs to know the fitness of its 1-dimensional individuals. UMDA/ICA uses the fitness of the original individuals for evolution, i.e. $f(As)$. However, $f(As)$ depends on all the independent components, so it is not the true measure of the fitness of the 1-dimensional individuals on s_i. The ideal measure should be $g_i'(s_i)$ in Eq. 3.3. The difficulty of this measure is that all that we have is $g'(s)$, while $g_i'(s_i)$ is only an implicit term. However, in ICA, it is theoretically possible to calculate the marginal density function $p_i(y_i)$ as in Eq. 3.4, where $-i$ represents the dimensions other that i.

$$p_i(y_i) = \int_{l_{-i}}^{u_{-i}} p(y_{-i}, y_i) dy_{-i} \tag{3.4}$$

Similarly, the theoretical and empirical formulas for calculating $g_i'(s_i)$ are Eq. 3.5 and Eq. 3.6, respectively, where S_i is the set of individuals whose ith variables equal s_i. The problem is that the population may have insufficient individuals which have the same s_i value, especially in high-dimensional space. Therefore, GA/ICA has to take the nearby individuals into account as well by calculating the average of their fitness with bigger weight given to the nearer individuals. This method results in a Parzen window like regression in Eq. 3.7, where σ is the average distance between the individuals and their nearest neighbors, and $\varphi(s_i^j, s_i^k)$ is a distance measure between s_i^k and s_i^j. In this way, the *estimatePop* function in Algorithm 3.1 is able to estimate the fitness of a one-dimensional individual in a sub-population.

$$g_i'(s_i) = \int_{l_{-i}}^{u_{-i}} g'(s_{-i}, s_i) ds_{-i} \tag{3.5}$$

$$g_i'(s_i) = \frac{1}{|S_i|} \sum_{s^j \in S_i} g'(s^j) \tag{3.6}$$

$$g_i'(s_i^j) = \frac{\sum_{k=1}^{N} \varphi(s_i^j, s_i^k) g'(s^k)}{\sum_{j=1}^{N} \varphi(s_i^j, s_i^k)}, \ where\ \varphi(s_i^j, s_i^k) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(s_i^j - s_i^k)^2}{2\sigma^2}} \tag{3.7}$$

Independent Component Sampling

The central part of GA/ICA is generating new 1-dimensional sub-populations on the independent components. EDA samples new individuals from the distribution model of the previous individuals. Since a sub-population has only one dimension, the fitness landscape is relatively easy, and so there is no need for GA/ICA to build such a distribution model. GA/ICA generates new individuals by applying genetic operators to the existing individuals directly. The function *independentSample* in Algorithm 3.1 follows the basic framework of GA except the part of individual evaluation, as it cannot evaluate individuals of only one dimension. Due to the property of a single dimension, it has some advantages over GA, including adaptive genetic operators, fitness prediction and high population diversity.

At the beginning of a generation, *icaSample* calculates the average distance of the individuals to their nearest neighbors, i.e. σ, which is used as the parameter to control the scale of crossover and mutation. In the initial population, the individuals are randomly generated in the whole solution space, so σ is relatively large. As the population converges, σ decreases gradually. With this method, crossover and mutation of GA/ICA adapt to the current sub-populations. Then it runs the following steps iteratively:

1. *icaSample* uses the tertiary-tournament. It randomly selects three individuals, uses the best two individuals for crossover and mutation, and replaces the worst individual with the offspring.

2. *icaSample* then generates two random numbers. One number follows the Laplace distribution in Eq. 3.8, while the other number follows the Gaussian distribution. *icaSample* uses the Gaussian random number for crossover. Laplace distribution has bigger tails than Gaussian distribution. *icaSample* uses the Cauchy random number for mutation to make it more likely for the offspring to jump out of the local optimum [124].

$$p(x) = \frac{1}{2\sigma}exp^{[illegible]\,\frac{\mu}{\sigma}} \qquad (3.8)$$

3. In each generation, GA/ICA chooses crossover or mutation as the primary genetic operator in the current evolution. When GA/ICA chooses crossover,

icaSample does two crossovers of opposite directions and one mutation, so it makes the population converge. When GA/ICA chooses mutation instead, *icaSample* does two mutations of opposite directions and one crossover, so it keeps the individuals search in different solution areas. Here the offspring on the opposite directions of a certain genetic operator are the two offspring generated on both the left and right sides of the original offspring which would be generated by the genetic operator.

4. *icaSample* cannot evaluate the fitness of the offspring candidates directly because they are of only 1 dimension. Instead, it uses the Parzen window like regression, as described in the function *estimatePop*, on the current sub-population to predict the fitness of the candidates. Then it chooses one of them as the offspring probabilistically, with bigger probabilities given to better candidates. This technique enables *icaSample* to search in more promising directions and avoid wasting evaluation time on bad candidates.

5. As discussed in Section 3.1, the number of local optima could increase on an independent component, so GA/ICA needs to make the population diverse to search in a large solution space. Before the offspring is actually put in the new sub-population, it is adjusted to maintain the sub-population diversity. *icaSample* keeps sorted the offspring already generated according to their positions on the one dimension, and finds the location where to insert the new offspring. If the new offspring's distance to either its pre-neighbor or next-neighbor in the list is smaller than the current σ, it is adjusted to make the distance at least σ if possible, otherwise as large as possible. In this way, the offspring are pushed away from each other to maintain the sub-population diversity.

3.3 Experiments

For its optimization performance, GA/ICA is tested to find the global optima of the six testing functions, and the results are compared to the following state-of-the-art algorithms,

1. Orthogonal Genetic Algorithm with Quantization (OGA/Q) [71]: OGA/Q uses orthogonal array to generate the initial population and the offspring in crossover.

2. Canonical Genetic Algorithm (CGA) [71]: This is the conventional Genetic Algorithm, with standard random initialization, crossover and mutation.

3. Fast Evolution Strategy (FES) [124]: FES is ES but with Cauchy mutation.

Eq. 3.9 are the six testing functions. All the testing functions are multimodal with many local optima besides the global optima. The functions' feasible solution spaces, global optimal function values and the population sizes and the maximal generations that the algorithms use are shown in Table 3.1. In GA/ICA, the crossover and mutation rates are 0.66 and 0.33, respectively. In OGA/Q and CGA, the crossover and mutation rates are 0.10 and 0.02, respectively. FES adopts a $(20, 300)$ strategy that generates 300 offspring from Cauchy mutation only in a generation.

Function	Function Space	Optimum	Population Size				Generation			
			GA/ICA	OGA/Q	CGA	FES	GA/ICA	OGA/Q	CGA	FES
f_1	$[-500,500]^{30}$	12569.5	200	200	200	20	200	>1000	>1000	4500
f_2	$[-5.12,5.12]^{30}$	0	400	200	200	20	200	>1000	>1000	2500
f_3	$[-32,32]^{30}$	0	400	200	200	20	200	>1000	>1000	750
f_4	$[-600,600]^{30}$	0	400	200	200	20	200	>1000	>1000	1000
f_5	$[-50,50]^{30}$	0	200	200	200	20	200	>1000	>1000	750
f_6	$[0,\pi]^{100}$	99.2784	600	200	200	NA	200	>1000	>1000	NA

Table 3.1: The Experiment Settings of The Algorithms

$$
\begin{aligned}
f_1 &= \sum_{i=1}^{30} (x_i \sin(\sqrt{|x_i|})) \\
f_2 &= -\sum_{i=1}^{30} (x_i^2 - 10\cos(2\pi x_i) + 10) \\
f_3 &= 20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right) + \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos(2\pi x_i)\right) - 20 - e \\
f_4 &= -\frac{1}{4000}\sum_{i=1}^{30} x_i^2 + \prod_{i=1}^{30}\cos\left(\frac{x_i}{\sqrt{i}}\right) - 1 \\
f_5 &= -\frac{\pi}{30}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2 \cdot [1 + 10\sin^2(\pi y_{i+1})]\right. \\
&\quad \left. +(y_{30} - 1)^2\right\} - \sum_{i=1}^{30} u(x_i, 10, 100, 4) \\
\textit{where } y_i &= 1 + \frac{1}{4}(x_i + 1) \textit{ and } u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases} \\
f_6 &= \sum_{i=1}^{100} \sin(x_i)\sin^{20}\left(\frac{i \times x_i^2}{\pi}\right)
\end{aligned}
\tag{3.9}
$$

The ICA algorithm used by GA/ICA is FastICA [54], which is very fast to find the linear transformation for ICA. Note that ICA does not know the original dimensions of some functions are actually independent, so the functions suffice to verify the capability of ICA to discover the independent components.

Test function	Mean number of function evaluations				Mean optimal function value				p-value
	OGA/Q	CGA	FES	GA/ICA	OGA/Q	CGA	FES	GA/ICA	
f_1	302166	458653	900030	**34420**	12569.45	8444.8	12556.4	**12569.47**	0.0033
f_2	224710	335993	500030	**56760**	**0**	-23.0	-0.2	$-4.2 \cdot 10^{4}$	0.0588
f_3	112421	336481	150030	**44400**	$\mathbf{-4.4 \cdot 10^{16}}$	-2.7	$-1.2 \cdot 10^{2}$	$-5.0 \cdot 10^{6}$	0.0012
f_4	134000	346971	200030	**45160**	**0**	-1.3	$-3.7 \cdot 10^{2}$	$-1.4 \cdot 10^{8}$	0.0162
f_5	134556	346800	150030	**26840**	$-6.0 \cdot 10^{6}$	$-3.7 \cdot 10^{1}$	$-2.8 \cdot 10^{6}$	$\mathbf{-1.4 \cdot 10^{8}}$	$2.2 \cdot 10^{37}$
f_6	302773	338417	NA	**115020**	92.8	83.3	NA	**97.6**	$9.2 \cdot 10^{12}$

Table 3.2: Experiment Results of GA/ICA and other population based GA on seven benchmark functions. The last column shows the p-value of the t-test of the mean values of OGA/Q and GA/ICA, where the hypothesis that the two mean values are the same is rejected at the significant level 0.05

Table 3.2 shows the experiment results. GA/ICA is executed 10 times on each testing functions. For each testing function, the experiment records the mean number of the function evaluations, the mean function value of the best individuals and the p-value of the t-test of the mean values of OGA/Q and GA/ICA, where the hypothesis that the two mean values are the same is rejected at the significant level 0.05. t-test is not performed between GA/ICA and other algorithms, since the results of other algorithms are obviously worse than those of GA/ICA and OGA/Q. The 20,000 individuals used by ICA are not counted in the mean number of the function evaluations, and the good ones among them are not used in the evolution either.

OGA/Q and GA/ICA outperform CGA and FES in terms of the function values of the solutions and the numbers of the function evaluations. Generally, OGA/Q and GA/ICA have comparable performance. For the function f_2, there is no significant difference between the mean values of the solutions returned by OGA/Q and GA/ICA, since the p value is larger than 0.05. For the functions f_1, f_5 and f_6, the solutions of GA/IGA are statistically better that that of OGA/Q since the corresponding p values are smaller than 0.05. For the function f_3 and f_4, the solutions of OGA/Q are statistically better that those of GA/ICA. While for all the test functions, the numbers of the function evaluations that GA/ICA uses are significantly less than those used by OGA/Q. Therefore, the results verify that GA/ICA is able to produce optimal or close-to-optimal solutions better than or comparable to those

Test function	Mean number of function evaluations			Mean function value (standard deviation)		
	UMDA/ICA	BLX-α	GA/ICA	UMDA/ICA	BLX-α	GA/ICA
f_1	54,500	57,020	**34,420**	3,686.7	5,097.4	**12,569.47**
f_2	387,160	377,960	**56,760**	-4.1713	-18.733	$\mathbf{4.24 \times 10^{-4}}$
f_3	47,480	81,640	**44,400**	-3.9328	-4.2158	$\mathbf{5.0 \times 10^{-6}}$
f_4	**44,920**	47,210	45,160	-2.9209	2.8734	$\mathbf{-1.4 \times 10^{-8}}$
f_5	47,100	50,960	**26,840**	-1.9138	-2.3257	$\mathbf{1.40 \times 10^{-8}}$
f_6	529,740	**69,300**	115,020	54.804	28.177	**97.61**

Table 3.3: Experiment Results of GA/ICA and other ICA based GA on seven benchmark functions. The best values are bolded

of OGA/Q while requiring much less function evaluations.

GA/ICA is then compared to UMDA/ICA [126] and BLX-α [114], which also use ICA to transform the problem, under the same experimental settings in Table 3.1. Table 3.3 shows the experimental results. For each testing function, we execute GA/ICA, UMDA/ICA and BLX-α for 10 runs, respectively, and we recorded the mean number of the function evaluations and the mean function value of the best individuals for each algorithm. It is obvious that GA/ICA find better solutions than UMDA/ICA and BLX-α. For the functions f_1 and f_6, the results of UMDA/ICA and BLX-α are far from the global optima. For the functions f_2, f_3, f_4 and f_5, the results of UMDA/ICA and BLX-α are close to the global optima, but the results of GA/ICA are several orders better. By checking the solutions of UMDA/ICA and BLX-α, it is found that UMDA/ICA and BLX-α actually get stuck in the local optima. GA/ICA also showsits advantage over UMDA/ICA and BLX-α in terms of the mean number of function evaluations. For the functions f_4 and f_6, the mean numbers of function evaluations of GA/ICA are not the least, but they are still comparable to the least numbers. While for the other functions, the mean numbers of function evaluations of GA/ICA are significantly less than those of UMDA/ICA and BLX-α.

3.4 Discussion

GA/ICA is a new GA employing ICA to solve unconstrained continuous global optimization problems. It first uses ICA to identify the independent components of the solution space w.r.t. the fitness. Then it divides the population into the sub-populations and evolves the sub-populations on the independent components separately. Finally it combines the optima on the independent components as the global optimum for the original problem. As the high-dimensional problem is divided into many 1-dimensional sub-problems, the solution space is exponentially reduced, and so the problem becomes easier for GA to solve. The experiment results show that GA/ICA requires much less function evaluations to produce optimal or close-to-optimal solutions which are better than or comparable to those produced by other testing GAs on the benchmark problems.

There exist other kinds of decompositions in the field of data analysis, such as factor analysis, non-negative matrix decomposition, principle component analysis, etc. However, none of these is able to discover the latent components which are independent from each other w.r.t. the fitness. Nonlinear ICA which finds the nonlinear transformation to produce the independent components is expected to be more general than ICA. In the case of the function optimization, nonlinear ICA may be general enough to solve the problems that linear ICA cannot find the independent components. Nevertheless, nonlinear ICA has its own challenges for its success, such as the choice of the nonlinear transformation, the indeterminancy of the number of independent components and the demand of a large number of training samples.

Chapter 4

Instruction Matrix Genetic Programming

4.1 Overview

EDDA can also be used in Genetic Programming (GP) to speed up the GP evolution by evolving the GP instructions and their interactions simultaneously. Genetic Programming (GP) [61][8] automatically constructs computer programs by evolutionary process. In GP, an individual represents an executable program. The program receives the inputs from the problem, and gives the output as the answer to the problem. The objective of GP is to evolve an optimal solution for the problem. GP has successfully produced results competitive with human solutions [10]. In Canonical Genetic Programming (CGP) proposed by Koza [61], an individual is a LISP-like program tree. The tree is composed of tree nodes of either functions or terminals.

If tree nodes are viewed as nominal variables, CGP can be treated as a combinatorial optimization problem. CGP has a huge solution space and it is NP-hard. To make things worse, the number of the tree nodes in CGP is not fixed, so the size of the solution space may increase exponentially during the evolution. It is thus quite common that CGP has to evaluate a large number of individuals before it can find the optimal program. In addition, evaluating an individual in CGP is usually time-consuming, because it needs to run the program tree for each training case.

Therefore, the time complexity of CGP is extremely high.

Divide-and-Conquer is a long-standing methodology to solve separable problems. To apply it to GP evolution, a complete program tree is divided into tree nodes, which are evolved separately to reduce the complexity. The difficulty of this approach is that tree nodes are interdependent on each other with respect to (w.r.t.) the fitness. The combination of the good tree nodes is necessarily the optimum of the complete program tree.

The proposed algorithm also takes in account the interdependencies between tree nodes in the form of subtrees. Subtrees are the building blocks in CGP, and they are combined into individuals via crossover [61]. Koza have successfully divided a program tree into subtrees and evolved the subtrees separately [62]. Combining the optima of the subtrees will have a good chance of obtaining the optimal or close-to-optimal complete program tree. In this way, the algorithm has both the advantages of the smaller solution space by dividing the complete program tree into separate tree nodes and maintaining the interdependencies between tree nodes in the form of subtrees.

A new GP framework, Instruction Matrix based Genetic Programming (IMGP)[73], evolves tree nodes and subtrees separately. There is no explicit population to store individual program trees in IMGP. Instead, it uses Instruction Matrix (IM) to maintain the fitness of the tree nodes and the subtrees. A row in IM consists of the cells of all the possible instructions, their fitness and subtrees on a certain tree node. In theory, IMGP can extract all the possible program trees from IM. It extracts a tree node from the corresponding row in IM according to the fitness of the instructions. The tree nodes extracted are combined into a complete program tree. IMGP evaluates the fitness of the program tree and then updates the fitness of the extracted tree nodes in IM accordingly. When the fitness of an instruction is worse than that of its subtree, IMGP extracts the whole subtree instead of extracting the tree nodes separately, and the fitness of the extracted subtree may be updated. Between generations, IMGP replaces bad fitness instructions with good fitness instructions in the same row in IM, and gradually IM is populated with instructions of good fitness.

IMGP is similar to Cooperative Coevolution [92]. It evolves tree nodes and subtrees separately in the sense that tree nodes and subtrees have their own fitness stored in IM. The tree nodes and the subtrees are extracted separately from the corresponding rows in IM. The extracted components cooperate in the form of a complete program tree. The fitness of the complete program tree is also used to update the fitness of its tree nodes and subtrees. A tree node evolves on its own by reproducing instructions of good fitness, and removing instructions of bad fitness.

This chapter is organized as follows. Section 4.2 describes the representation and algorithm of IMGP in detail. Section 4.3 presents the experiments on the benchmark GP problems. Section 4.4 gives an application example of IMGP for classification problems and the experimental results. Section 4.5 is the discussion.

4.2 Architecture

If tree nodes are treated as variables, Canonical Genetic Programming (CGP) is a high dimensional combinatorial optimization problem. To apply the Divide-and-Conquer methodology, a new framework called Instruction Matrix based Genetic Programming (IMGP) is proposed. IMGP evolves tree nodes separately while taking into account of their interdependencies in the form of subtrees. It maintains an Instruction Matrix (IM) to keep the fitness of functions and terminals in tree nodes. It uses a new kind of fixed length expression to represent a program tree. It extracts a program tree from IM by selecting a function or terminal of good fitness for each possible tree node. After the program tree is evaluated, IMGP updates the fitness of corresponding functions or terminals in IM.

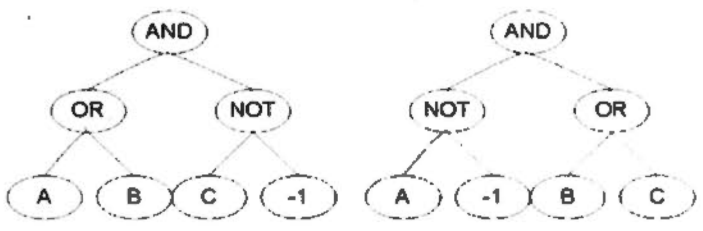
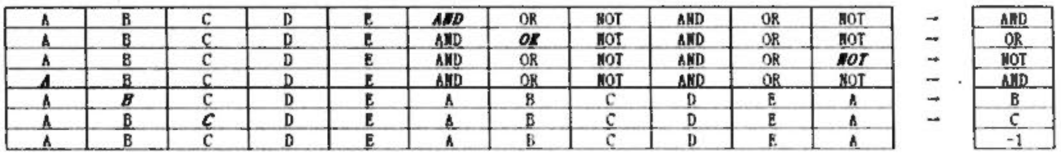


Figure 4.1: The trees represented in hs-expression (AND OR NOT A B C -1) & (AND NOT OR A -1 B C)

4.2.1 Representation

Rather than using *s-expression* as CGP [61], IMGP uses *hs-expression*, which is mapped to program tree. An *hs-expression* is a $2^{D+1} - 1$ long array to store a binary tree of depth D at most. Every possible tree node has a corresponding element in the array, even if the tree node does not exist. The relation between the elements in an *hs-expression* is similar to that used in the array of Heap Sort, but the "larger-than" relation is changed to the "parent-of" relation. The tree root is element 0 in the *hs-expression*. For the kth element in the *hs-expression*, its left and right children are the $2k+1$th and $2k+2$th elements, respectively. If it has no child, the corresponding elements are set to -1 instead. Therefore, the elements in the first half of the array can be either functions, terminals or empty, while the elements in the second half of the array must be either terminals or empty. Fig. 4.1 shows two examples. Unlike the trees represented by *s-expression*, the trees represented by *hs-expression* of the same length have exactly the same shape if -1 is viewed as a virtual node. Another difference is that the elements at the same locus in *hs-expressions* always correspond to the nodes at the same position in the program trees. In comparison to this, Genetic Expression Programming [2] also uses a fixed length string to represent a tree, and some of its elements may not appear in the tree. However, when the degree of a tree node changes, the positions of its subtree and its sibling change drastically.

In addition, *hs-expression* can be easily extended to represent trees of more than 2 branches. To represent a m-branch tree, *hs-expression* is a $m^{D+1} - 1$ long array.



A	B	C	D	E	***AND***	OR	NOT	AND	OR	NOT	→	AND
A	B	C	D	E	AND	***OR***	NOT	AND	OR	NOT	→	OR
A	B	C	D	E	AND	OR	NOT	AND	OR	***NOT***	→	NOT
A	B	C	D	E	AND	OR	NOT	AND	OR	NOT	→	AND
A	***B***	C	D	E	A	B	C	D	E	A	→	B
A	B	***C***	D	E	A	B	C	D	E	A	→	C
A	B	C	D	E	A	B	C	D	E	A		-1

Figure 4.2: Instruction Matrix (IM) and an *hs-expression* extracted from it. IM keeps multiple instructions for each element of the *hs-expression*. An element of the *hs-expression* is extracted from the corresponding row in IM. The cells in bold typeface are the extracted elements

For the kth element in the *hs-expression*, its children are the $m \cdot k + 1$th, $m \cdot k + 2$th, ..., $m \cdot k + m$th elements.

In IMGP, there is no explicit population. Instead, it maintains an Instruction Matrix (IM) to store all the possible instructions used in a program tree. While CGP generates new program trees from existing program trees, IMGP extracts new *hs-expressions* from IM, which are translated to program trees. The cells in IM are data structures consisting of instructions and related information. A row of IM corresponds to an element in *hs-expression*, and hence a tree node in a program tree. The cells in a row stores all the possible instructions which can be used in the corresponding tree node. The mapping between IM and program trees is the same as the mapping between a *hs-expression* and a program tree. Row 0 corresponds to the root of the program tree. Recursively, for the kth row corresponding to a tree node, the $2k+1$th and the $2k+2$th rows correspond to the left and right child of the tree node, respectively. The height of IM is the same as the length of *hs-expression*, i.e., $H = 2^{D+1} - 1$. A row contains multiple instances of any type of instructions, and so the width of IM W is the number of instructions in a row. The lower part of IM, i.e., the part from row $\frac{H}{2}$ to row H, contains only terminals since they correspond to the tree leaves. Fig. 4.2 shows an example of IM and an *hs-expression* extracted from it. Basically, the element at locus k in the *hs-expression* is extracted from row k in IM. The details are described in Section 4.2.2.

Besides an instruction of function or terminal, a cell in IM also keeps some auxiliary data. The pseudocode of its internal data structure and the initial values

opcode	instruction	=	...
double	best_fitness	=	MAX_FITNESS
double	avg_fitness	=	0
int	left_branch	=	-1
int	right_branch	=	-1
int	eval_num	=	0

Figure 4.3: The pseudocode of the internal data structure of the cell in IM which shows the initial values of the fields. The field *instruction* can be either function or terminal. *MAX_FITNESS* and 0 are the maximal and minimal possible fitness, respectively. -1 means there are no links for the left and right child of the tree node at the beginning

are shown in Fig. 4.3. The data structure also stores the information of its best subtree. *instruction* is the operation code of the instruction. *eval_num* is the number of times that the instruction has been evaluated. *best_fitness* and *avg_fitness* are the best and the average fitness of the instruction. *best_fitness* is also the fitness of the best subtree of the instruction. *left_branch* and *right_branch* are the left and right branches of the best subtree. These fields keep some information of the fitness landscape of the tree node, and they are used in the evolution of the tree node. Their specific usage is explained in detail in Section 4.2.2. Note that in IMGP, the smaller the fitness, the better it is.

4.2.2 Algorithm

Algorithm 4.1 is the main program of IMGP, where G is the maximal number of generations, and P is the number of individuals in each generation. It divides a complete tree into separate tree nodes, calculates the fitness of the tree nodes so as to evolve them separately, and combines the optima of the tree nodes into a complete program tree. In each generation, IMGP runs the following steps repeatedly. Firstly IMGP extracts two individuals from IM and calculates their fitness. Then IMGP performs crossover and mutation on them and calculates the fitness of their offspring. After evaluating an individual, IMGP updates the corresponding cells in IM with the fitness of individual. At this point, IMGP deletes all the individuals

Algorithm 4.1: The Main Program of IMGP

```
Output: the best individual
initialize IM;
for gen from 0 to G do
    num ← 0;
    while num < P do
        extract two individuals i and j from IM;
        calculate their fitness respectively;
        update their cells in IM with the fitness;
        if crossover i with j successfully then
            evaluate the offspring and update its cells;
        else if mutate i successfully then
            evaluate the offspring and update its cells;
        if crossover j with i successfully then
            evaluate the offspring and update its cells;
        else if mutate j successfully then
            evaluate the offspring and update its cells;
        num ← num + the number of individuals evaluated;
    shuffle IM;
```

because their information has already been stored in IM. A generation finishes after IMGP evaluates P individuals. Then IMGP uses matrix shuffle to replace cells of bad fitness with those of good fitness in IM. The best individual is reported as the optimal program after G generations.

Individual Extraction

IMGP extracts the tree nodes from IM and combines them into a complete tree. Algorithm 4.2 is the function to extract an individual. Firstly IMGP constructs an empty *hs-expression* filled with -1, and aligns it vertically with IM. It starts to extract the instruction of the tree root from row 0, and puts it at locus 0 in the *hs-expression*. Then IMGP continues to extract the rest of the program tree recursively. The instruction of a tree node is extracted from the corresponding row using binary tournament selection, and then the extracted instruction is placed at the corresponding position in the *hs-expression*. Binary tournament selection is comparing the

Algorithm 4.2: Extract Individual

```
Input: individual, IM, locus, subtree
Output: individual
$best \leftarrow false$;
if $subtree \neq -1$ then
    $individual[locus] \leftarrow subtree$;
    $CELL \leftarrow IM[locus.individual[locus]]$;
    $best \leftarrow true$;
else
    $individual[locus] \leftarrow Tournament(IM.locus)$;
    $CELL \leftarrow IM[locus.individual[locus]]$;
    if $Random(1) < 1 - \frac{CELL.best\_fitness}{CELL.avg\_fitness}$ then
        $best \leftarrow true$;
if $best = true$ and $CELL.instruction$ is $function$ and
$CELL.left\_branch \neq -1$ and $CELL.right\_branch \neq -1$ then
    Extract(individual, IM, locus*2+1, $CELL.left\_branch$);
    Extract(individual, IM, locus*2+2, $CELL.right\_branch$);
else
    Extract(individual, IM, locus*2+1, -1);
    Extract(individual, IM, locus*2+2, -1);
```

fitness of two randomly selected instructions and selecting either of them probabilistically. If the extracted instruction at locus k is a function, IMGP proceeds to extract its left child from the $2k+1$th row, and its right child from the $2k+2$th row. It does so recursively until all the branches are completed. In Fig. 4.2, the words in bold italic typeface are the extracted instructions, and the completed *hs-expression* is on the right. The corresponding tree is depicted on the left in Fig. 4.1. The details of extracting (AND OR NOT A B C -1) from IM is shown in Fig. 4.4

The best subtree of an instruction is its subtree in the best individual that it has ever been extracted into. After a tree node is extracted, IMGP occasionally checks whether the best subtree of the selected instruction should be extracted as a whole so that the tree nodes in the best subtree are extracted directly without further binary tournament selections. How often it does so depends on the best and the average fitness of the instruction. Eq. 4.1 is the probability of extracting the best subtree. The bigger the difference between them is, the more likely its subtree is selected.

```
1. extract AND from row 0 as the root
  2. extract OR from row 1 as the left child of the root
    3. extract A from row 3 as the left child of OR
    4. extract B from row 4 as the right child or OR
  5. extract NOT from row 2 as the right child of the root
    6. extract C from row 5 as the left child of NOT
7. stop as NOT has no right child
```

Figure 4.4: The steps of extracting (AND OR NOT A B C -1) from the IM in Fig. 4.2. For each tree node, IMGP selects two instructions in the corresponding row of IM randomly, compares their average and best fitness, and extracts one of them probabilistically. After extracting a tree node, IMGP recursively extracts its left and right child

The reason is that if the best fitness is much better than the average fitness, the tree constructed with the best subtree is likely to be much better than the tree constructed without it. Since tree nodes are highly interdependent w.r.t. the fitness in GP, best subtrees keep part of the interdependence information between the tree nodes in IM.

$$prob_{best} = 1 - \frac{best\_fitness}{avg\_fitness} \tag{4.1}$$

In the binary tournament selection, IMGP randomly selects two candidate instructions, compares their fitness, and selects the better one probabilistically. An instruction is extracted either separately or together with its best subtree. Therefore, when IMGP compares the fitness of two candidate instructions, it compares not only their average fitness, but also considers their best fitness as well. Eq. 4.2 calculates the expected fitness of an instruction. It considers the probability of selecting the best subtree of the instruction, and IMGP should use the best fitness in that case. The traditional binary tournament selection always selects the better one, so the worst instruction in IM is never selected. To be less greedy, we use Roulette Wheel Selection [42] to select one of the two instructions based on their expected fitness.

$$E(fitness) = prob_{best} * best\_fitness + (1 - prob_{best}) * avg\_fitness \qquad (4.2)$$

Extracting individuals makes IMGP avoid being trapped in a small solution area. In CGP, when an individual is changed by crossover or mutation, it replaces only a subtree with a new one, and so the offspring is still in the neighborhood of the parent(s). Therefore, the solution space that CGP searches is largely determined by the initial population. However, IMGP does not generate an individual from an existing parent. It extracts a completely new individual from IM, and thus the new individual bears little similarity with the previous individuals. Therefore, IMGP searches a relatively large solution space, and the extracted individuals have high diversity. In addition, there are multiple copies of any type of instructions in a row of IM, and each copy has different fitness and subtrees. Even if an instruction has a copy of bad fitness, it might still be selected due to another copy of good fitness. Therefore, IMGP is relatively resistant to local optima.

Instruction Evaluation

In IMGP, an individual is evaluated using the post-order recursive routine. To evaluate a function node, it takes the evaluation of its left and right children as the inputs. To evaluate a terminal node, it evaluates the corresponding program input. Since the individual is discarded right after evaluation and reproduction, it cannot carry along its fitness as in CGP. Instead, the new fitness is fed back to its corresponding cells in IM so that it can be used in extraction later. The feedback comes in two ways:

1. In Eq. 4.3, the new fitness, $fitness'$, is averaged with the old fitness, $fitness$. The evaluation number, $eval\_num$, is incremented by one. With this method, we know how good the instruction is on average.

$$fitness = \frac{fitness * eval\_num + fitness'}{eval\_num + 1} \tag{4.3}$$

2. If the new fitness is better than the best fitness of the instruction, its best fitness is updated, and its left and right branches are changed to those in the current individual accordingly. This actually keeps good subtrees in IM together with their fitness.

The second point is very important. As pointed out in [115], a new building block is unlikely to survive in the next two generations even if the individual constructed with it has an average fitness. In IMGP, whenever a good subtree is identified, it is remembered immediately.

All the individuals, no matter how much their fitness are, contain useful information of the problem. Therefore, IMGP updates IM with not only the good individuals, but all the extracted individuals. For most of the related algorithms discussed in Section 2.2.2, they update their models only with the good individuals and ignore the bad individuals. This would make some of the bad tree nodes spuriously good in the models because they happen to be in the good individuals. On the contrary, updating IM with the bad individuals decreases the fitness of the bad tree nodes, and so they are unlikely to be extracted later.

Genetic Operators

In IMGP, crossover and mutation are similar to those in CGP. However, as IMGP keeps the fitness of the tree nodes in IM, it is able to perform heuristic crossover and mutation on the tree nodes directly. According to the *building block hypothesis* [42], small good building blocks are spread among the individuals and recombined into large good building blocks. Therefore, combining good subtrees is likely to produce good individuals. When IMGP performs crossover on individuals, it replaces a subtree in one parent only with a better counterpart in the other parent so that the offspring is likely to be better. In mutation, IMGP selects a mutation

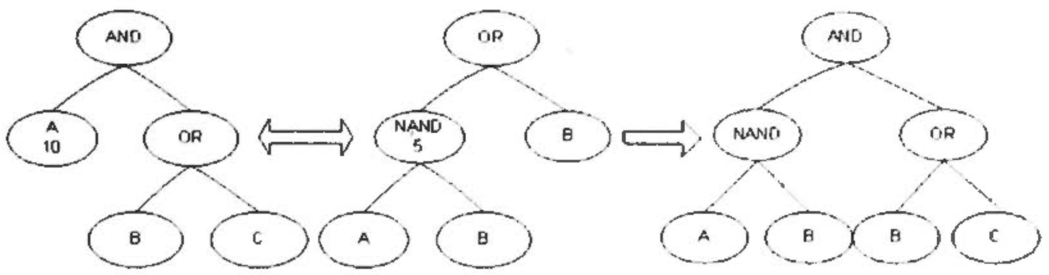


Figure 4.5: Crossover in IMGP. The left subtree in the first individual is replaced with its better counterpart in the second individual

point in the current individual, and replaces the original subtree with a new subtree extracted from IM.

The crossover is similar to context preserving crossover [30] because the two subtrees of the parents must be in the same position to reduce the macro-mutation effect of the standard crossover [8]. However, unlike the crossover used in other GP algorithms, the crossover in IMGP is asymmetric. When IMGP tries crossover between individual *i* and individual *j*, it picks either of the two branches on the roots in both individuals at random. It replaces the subtree of *i* with that of *j* if the latter has a better fitness than the former. Otherwise IMGP recursively tries crossover on the branches of the picked branches. Fig. 4.5 shows an example. Note that crossover would fail if it could not find a better subtree to replace the original one.

Matrix Shuffle

CGP converges by spreading good instructions over the population to reproduce good individuals. IMGP starts with extracting program trees from IM at random, so it samples different solutions in the huge solution space. In the evolution, it is important for IMGP to sample the program trees which are similar to the previously extracted good program trees.

However, there is no explicit population in IMGP since it extracts an individual and discards it later. To ensure that the individuals extracted in the subsequent generations have good instructions, IM should be populated with good instructions. IMGP uses matrix shuffle to propagate good instructions in IM, and consequently

Algorithm 4.3: Matrix Shuffle

```
Input: IM, the current row to be shuffled
Data: $F \cap T$ is the function and terminal set
for $instruction \in F \cap T$ do
    $Count[instruction] \leftarrow$ the number of instruction in $IM[row]$;
while $si < SUCCESS$ and $ti < TRIAL$ do
    $ti \leftarrow ti + 1$;
    $i, j \leftarrow Random(W)$;
    $Ci \leftarrow IM[row, i]$;
    $Cj \leftarrow IM[row, j]$;
    if $Ci.avg\_fitness < Cj.avg\_fitness$ and
    $Ci.best\_fitness < Cj.best\_fitness$ and
    $Count[Ci.instruction] < CONVERGENCY$ and
    $Count[Cj.instruction] > DIVERSITY$ then
        $IM[row, j] \leftarrow IM[row, i]$;
        $si \leftarrow si + 1$;
    if $Cj.avg\_fitness < Ci.avg\_fitness$ and
    $Cj.best\_fitness < Ci.best\_fitness$ and
    $Count[Cj.instruction] < CONVERGENCY$ and
    $Count[Ci.instruction] > DIVERSITY$ then
        $IM[row, i] \leftarrow IM[row, j]$;
        $si \leftarrow si + 1$;
```

to increase the probability of extracting them together in the same program tree. Matrix shuffle processes IM row by row. Algorithm 4.3 shows how it shuffles a row. It selects a certain number of pairs of cells in a row, and for each pair it replaces the worse one with the better one in terms of both the best and the average fitness. While IM evolves with matrix shuffle, good instructions emerge to dominate the rows in IM, and the copies of bad instructions decrease.

In CGP, as the population converges, the majority of the individuals have more or less the same instructions, while the other instructions die out. It is hard for CGP to maintain the diversity of the population because measuring the distance between individuals is difficult. However, IMGP evolves on the level of instructions, and so it is possible to maintain the diversity of the instructions. In matrix shuffle, when a good instruction $IM[row, i]$ replaces a bad instruction $IM[row, j]$, where

i and j are the indices for the two instructions, IMGP needs to check two constraints: $Count[IM[row,i]] < CONVERGENCY$ and $Count[IM[row,j]] > DIVERSITY$, where the operator $Count[\cdot]$ is the number of copies of the instruction. The replacement is prohibited if it violates either of the constraints. Clearly, $CONVERGENCY$ and *Diversity* controls the convergence and diversity of the instructions directly. $CONVERGENCY$ should not be too high to discourage convergence, and $DIVERSITY$ should not be too low to enhance diversity. In the current implementation, IMGP uses $CONVERGENCY = W/2$ and $DIVERSITY = 2$.

As pointed out in [21], the edit distance, i.e., the difference between a program tree and the best program tree, generally decreases after the early generations in CGP. It is thus good to keep the edit distance not too small so as to enhance the diversity of the population. Basically, matrix shuffle prohibits good instructions from reproducing themselves too many times, and reserves a minimum number of bad instructions. This thus maintains the diversity of the instructions in IM easily and effectively.

As a summary, Fig. 4.6 is the diagram of the overall program of IMGP. IMGP extracts the program trees from IM by selecting their instructions recursively according to the fitness of the instructions in IM. The fitness of the extracted program trees are used to update the fitness of the instructions in IM. The sub-trees are stored in IM in the form of best children links. Afterwards, IMGP crossover and mutate the extracted program trees based on the fitness of their instructions, and the offsprings are also evaluated and used to update IM. At the end of a generation, IMGP shuffles IM to duplicate better instructions and remove worse instructions.

4.3 Experiment

This section describes the experiments and the results of IMGP. First, IMGP and CGP are tested on the benchmark GP problems and their results are compared. Second, IMGP is compared to the related algorithms in Section 2.2.2 on a few

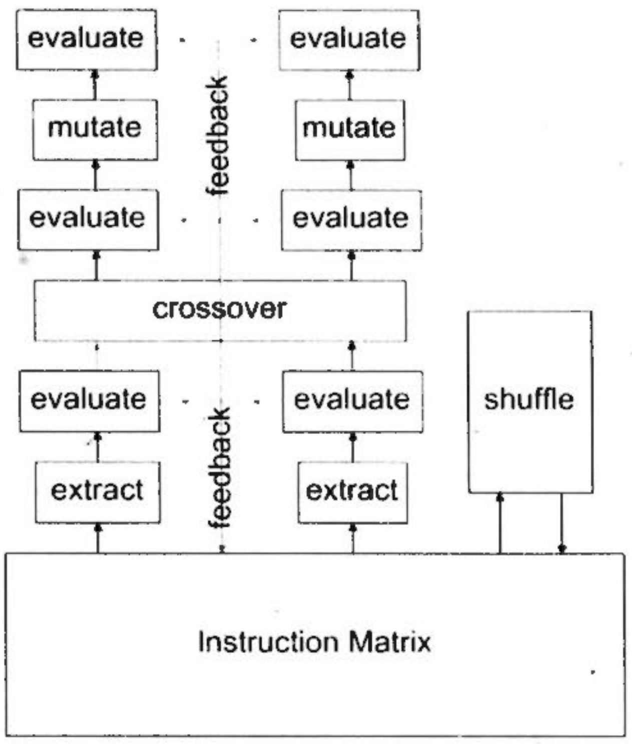


Figure 4.6: The flow chart of IMGP

selected problems.

4.3.1 Comparison with Canonical Genetic Programming

This section compares the performance of IMGP and CGP on 4 benchmark problems [61].

The first problem is the symbol regression problem which searches for a mathematical expression $y = x^4 + x^3 + x^2 + x$, where x is an integer uniformly and randomly generated from the range of $[0, 20)$. The fitness used is the hit count which is incremented by one if the difference between the program output and the correct result is larger than a predefined threshold. The second problem is to discover the even-five-parity expression, $\neg(a \oplus b \oplus c \oplus d \oplus e)$. The training cases are all the 2^5 combinations of the five binary variables. The fitness is calculated as the sum of the wrong results produced by the individual program. The third problem is the artificial ant on Santa Fe Trail. Executing the optimal program repeatedly enables the ant to eat all the 89 food pellets on the trail within 400 steps. The number of

Parameters	Symbol Regression	Even-5-Parity	Artificial Ant	11-multiplexer
Terminals	$\{x\}$	{a,b,c,d,e}	{move,left,right}	{a,b,...,k}
Functions	$\{+,-,\times,/\}$	{and,or,nand,nor}	{if,progn2,progn3}	{if,and,or,not}
Population	500	2000	2000	4000
Matrix Width	40	405	90	150
Matrix Height	63	1023	1093	3280
Generations	100	100	100	100

Table 4.1: The Experiment Settings of IMGP and lilgp on the Benchmark Problems

the food not eaten by the ant is used as the fitness. The fourth problem is boolean 11-multiplexer. Among the 11 variables, three are used as the address to select the output from one of the other 8 variables. However, GP has no idea of which variables are the address. The training cases are all the 2^{11} combinations of the 11 binary variables, and the fitness is the number of incorrect output.

Table 4.1 lists the parameter settings used in the experiments. IMGP has no population, but for convenient comparison with CGP, we refer to the number of the individuals evaluated between the matrix shuffles (generations) as the population size, i.e., P in Algorithm 4.1. IMGP and CGP adopt the same population size and generation number. In Artificial Ant and 11-Multiplexer, some functions require 3 arguments, which means the maximum branches of a node is three instead of two. Therefore, the IM height and the *hs-expression* length is increased to $\frac{3^{D+1}-1}{2}$, where D is the maximal level of a program tree. To determine an acceptable size of IM, IMGP with different sizes is tested on a small number of the training cases for a few generations. By observing the fitness of the best program trees, a suitable tree size can be determined.

lilgp [130] is used as CGP. For fair comparison, the ephemeral random constant is removed from lilgp. The tournament size was two. The population size and the number of generations are set as in Table 4.1. The other parameters are the same as in [61]. Both lilgp and IMGP use the same random seeds, which themselves are randomly generated. For each problem, IMGP and lilgp are executed 20 times with 20 different random seeds.

Fig. 4.7 shows the plots of the average fitness of the best individuals from generation 1 to generation 100 on the four problems. Table 4.2 shows the numerical

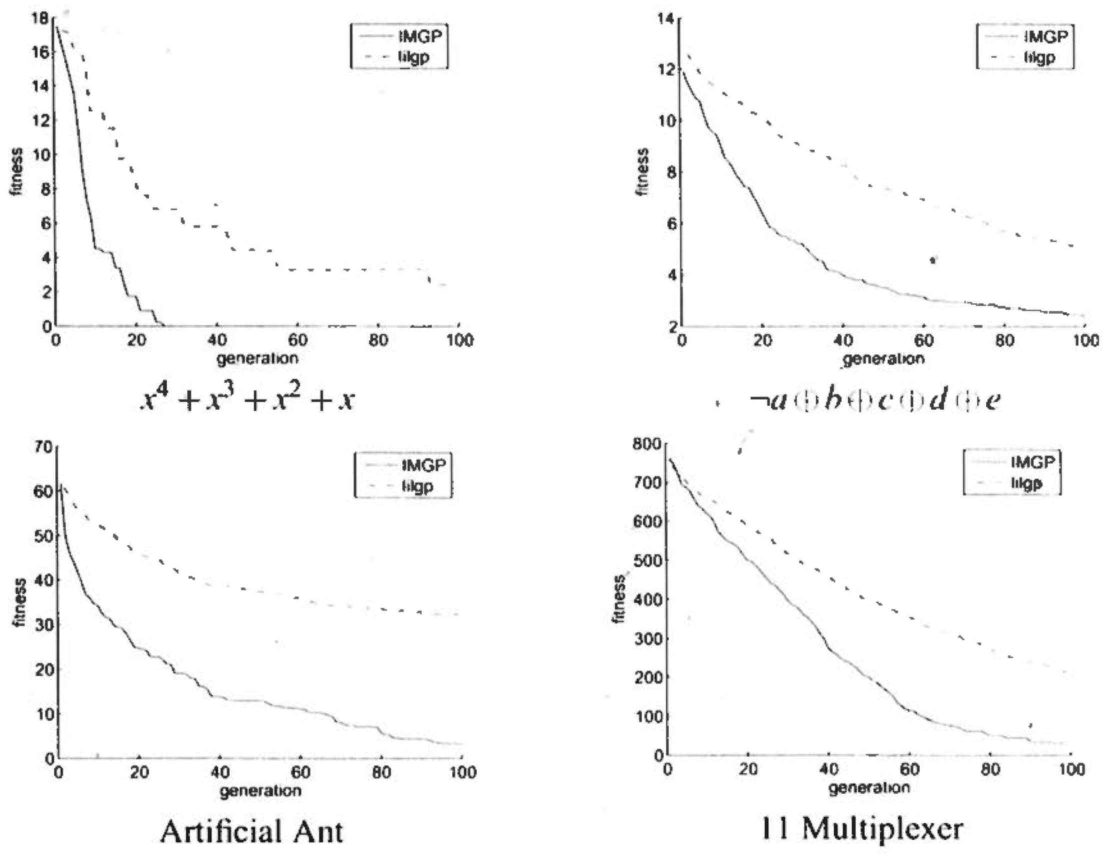


Figure 4.7: The average fitness of the best individuals of IMGP and lilgp through generations on the four testing problems

experimental results of IMGP, including the success rate, the fitness of the best program tree, the size of the best program tree and the running time. The fitness and the size of the best program tree are reported in their minimal, median and maximal values in all the runs.

In symbol regression, IMGP finds the solution in all the 20 runs compared with 17 successful runs in lilgp. In terms of the average fitness of the best individuals, IMGP also converges faster than lilgp. In even-five-parity, IMGP finds the solution in three runs, however, lilgp fails to find the solution in any of the 20 runs. Regarding the convergence speed, IMGP also outperforms lilgp significantly as its average best fitness was 2.4 while lilgp's is nearly double of that. In 20 artificial ants, IMGP find 12 ants eating up all the food pellets. lilgp can not find any successful ant, and the average fitness of its best individuals was 31.8, far from 0. In 11-multiplexer,

Testing	Success		Fitness		Tree Node		Time (sec)	
	IMGP	CGP	IMGP	CGP	IMGP	CGP	IMGP	CGP
Regression	100%	85%	0:0:0	0:6:18	17:41:59	19:141:301	0.30	0.40
Even-5-Parity	15%	0	0:3:5	3:5:6	817:975:1007	231:495:861	264.05	81.15
Artificial Ant	60%	0	0:0:20	11:34:52	111:225:326	23:590:1848	25.00	30.40
11-Multiplexer	65%	0	0:0:128	1768:1832:1952	30:45:62	84:368:909	236.00	2813.40

Table 4.2: The Numerical Experiment Results of IMGP and CGP on the Benchmark Problems. It shows the success rate, the fitness of the best individuals, the number of tree nodes and the running time. The fitness and the size of the best program tree are reported in their minimal:median:maximal values in all the runs

IMGP finds the perfect multiplexer for 13 times out of 20 runs, while lilgp fails all the time. The average best fitness of IMGP is also much better than that of lilgp, although their fitness in the first generation are almost the same.

IMGP and CGP are executed on a Linux workstation of Pentium 2.2GHz. Except for the even-five-parity problem, the running time of IMGP is shorter than that of CGP. This is expected, since IMGP has much larger success rates than CGP. IMGP stops early before the final generation 100 when it finds a perfect solution. For the difficult even-five-parity problem, IM has many rows and so the tree programs have many levels. Therefore, IMGP is slower than lilgp on even-5-parity. However, lilgp cannot find any solution out of the 20 runs.

4.3.2 Comparison with Related Algorithms

IMGP is also tested on those problems which have been tested by the related algorithm to compare their results. However, it is difficult to compare the results precisely, as some of the papers give the results only in the figures without the exact numerical values. Therefore, only the problems whose results are reported in numbers in other papers are used. The experiment settings are the same as in the related algorithms described in Section 2.2.2.

Six-bit parity problem is similar to even-five-parity. It has six boolean arguments, and it returns true if the number of true arguments (1's) is odd and false otherwise. However, other than using the boolean function set, it uses a real-valued function set $\{+, -, \times, \%, sin, cos, exp, rlog\}$, where $rlog$ is the protected log which

Testing Algorithm	Success Rate	Program Evaluation			Tree Node		
		min	median	max	min	median	max
IMGP	100%	2000	2000	18000	10	29	113
PIPE	70%	9432	52476	482545	22	61	100
CGP	60%	64000	120000	396000	24	90	161

Table 4.3: The Experiment Results of IMGP, PIPE and CGP on Six-Bit Parity Problem. It shows the success rate, the number of program evaluations, and the number of tree nodes

Algorithm	Success Rate	Evaluations
IMGP	40%	14050
GMPE	60%	13590
CGP	<60%	100000

Table 4.4: The Experiment Results IMGP, GMPE and CGP on Max Problem. It shows the success rate and the number of program evaluations

returns the log of the absolute value of the argument. The output of the program is mapped to true if it is larger than 0 and false otherwise. 20 independent runs were carried out using IMGP. The result is compared to that of PIPE [103] and CGP in Table 4.3. IMGP is the best and it achieves 100% success rate and it requires much smaller number of program evaluations.

Max problem has a single input with value 0.5 and two functions, $+$ and $\times$. The purpose is to find a tree with maximum fitness under the tree size constraint. Obviously the optimal tree is a full tree, whose nodes on the two levels right above the terminals are + to produce the value of two, and the other nodes on the top are $\times$ to multiply all of the 2's. In this experiment, the maximum tree depth is seven, so the maximum fitness is 65536. The result compared to GMPE [110] and CGP is reported in Table 4.4. Both IMGP and GMPE outperform CGP. With approximately the same number of evaluations, GMPE has a higher success rate than IMGP. However, if IMGP keeps running till 100 generations, its success rate increases to 95%.

Function regression is to search for the function shown in Eq. 4.4. The fitness cases are sampled at 101 equidistant points in the interval [0,10]. The fitness is

Algorithm	mean	std. dev.	min	median	max
IMGP	5.25	2.79	2.34	5.09	13.56
PEEL	6.79	4.91	0.68	5.21	18.90
GGP	7.87	3.54	0.95	7.56	14.00

Table 4.5: The Experiment Results of IMGP, PEEL and GGP on Function Regression. It shows the fitness of the best individual

not the hit count, but the sum of the differences between the outputs and the correct answers. The fitness of the best individuals of IMGP are compared to those of PEEL [109] and GGP [120] in Table 4.5. IMGP gets a smaller error and a smaller standard deviation. Although the minimum errors of PEEL and GGP are smaller than that of IMGP, their median and maximum errors are much larger than those of IMGP. Obviously, IMGP is more stable with a less variance than PEEL and GGP on this problem.

$$f(x) = x^3 \times e^{-x} \times \cos x \times \sin x \times (\sin^2 x \times \cos x - 1) \tag{4.4}$$

4.4 Instruction Matrix based Genetic Programming for Classification

IMGP is a very flexible paradigm, and this section describes an implementation of IMGP for binary class classification problems. The problem of classification has been a major task of machine learning and data mining. Basically, given a set of training data of known classes, a classifier is learned to predict the classes of new data. There are many existing classifiers, such as Decision Tree [95], Neural Network [13], Support Vector Machine [26] and Sparse Kernel Feature Machine in Section 4.4.1, etc. GP can also be used to evolve classifiers in the form of programs.

GP has a few favorable features for classification, such as the variable length representation and the population of different solutions, and the interpretability of the classifier [33][80][118]. First, the structures of many traditional classification models are fixed, and the major task of learning is finding the proper parameters of the models. Because a variable representation of classifier is adopted in GP, it has more freedom to find appropriate model structures than fixed representations. Second, the results of some traditional learning algorithms depend on the initial parameters of the models. In GP, the population contains many individuals, so GP has a better chance to find the optimal structures and the parameters of the models. Third, GP usually uses only simple algebra operators and possibly a subset of all the attributes in a program, and so the resulted classifier is easy to interpret.

In a learning problem, a training dataset (X, t) consists of N samples $\{x^i | i = 1...N\}$ and their targets $\{t^i | i = 1...N\}$. A sample x^i has M attributes $\{x^i_j | j = 1...M\}$. Given a sample x^i, a classifier predicts its class as y^i. The task of the classifier learning algorithm is to find a classifier of minimal error. In this section, the sum of squared error function is only applied on the misclassified data as in Eq. 4.5, where N' is the number of misclassified data.

$$E = \frac{1}{2}\sum_{i=1}^{N'}(t_i - y_i)^2 \tag{4.5}$$

Suppose the instruction set in CGP is $\{x_1, \ldots, x_m, +, -, \times, /\}$, the program tree is actually a mathematical formula and so the class boundary can be represented by a mathematical equation. For a binary class problem, the program tree receives the attributes of a sample as the inputs. If the output of the program tree is positive, the sample is assigned class one, otherwise class two. The classification error on the training data is used as the fitness.

As a variant of CGP, IMGP can also be used for classification problems and is expected to be more efficient and more effective than CGP. The instruction set of IMGP for classification introduces Constant Instruction (CI) in the instruction set. CI is used as the terminal of constant as the parameter of the classifier model defined by the program tree. CGP uses Ephemeral Random Constant (ERC) as a constant. ERC is instantiated to a random number in the initialization of CGP, and the number is fixed in later generations. In IMGP, because a CI will be extracted into different individuals, it should change the constant during evolution. CI has two constants, i.e., the Random Constant (RC) and the Best Constant (BC), corresponding to its average and best fitness, respectively. BC is used when the CI is used in the best program tree for testing on unknown data. RC is assigned a new random number whenever it is extracted in a new individual. The new value of RC replaces the value of BC if the new individual has a better fitness than the best fitness of the CI.

Contrast to CI in IMGP, ERC does not change during evolution even if the change would lead to a better constant given the structure of the program tree. As mentioned in [29], various approaches have been proposed to change constants in the evolution. However, they are only mutation on the constant disregarding the current individual. In Neural Network [13] and Support Vector Machine [26], the structure of the model is given beforehand, and the learning process is adjusting the model parameters to minimize the error function. GP can be viewed as searching for the structure of the model and the parameters of the model simultaneously. It is thus possible to change the parameters without changing the structure to achieve better fitness. If the error in Eq. 4.5 is used as the fitness and the structure of the program tree is fixed, the fitness is actually a continuous function of constants. Therefore, IMGP can modify the constants of a program tree to decrease classification error. As in Neural Network, IMGP uses gradient descent [13] to find the optimal constants. Gradient descent changes a constant is according to the partial derivative of MSE as shown in Eq. 4.6.

$$c' = c - \frac{\eta}{2}\frac{\partial E}{\partial c} = c + \eta \sum_{i=1}^{N'} (t_i - y_i)\frac{\partial y_i}{\partial c} \tag{4.6}$$

η is the learning rate. c is the vector of the constants. y_i is the output of the program tree given the sample x_i. Given a mathematical formula represented by a program tree, IMGP calculates its partial derivative w.r.t. the constant. Suppose the formula is a composite function $y = (f \circ (g, h))(c)$, where function f takes two arguments of functions g and h. The derivative of f with respect to c is $\frac{\partial y}{\partial c} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial c} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial c}$. For example, $(\frac{a}{b})' = \frac{\partial}{\partial a}(\frac{a}{b})a' + \frac{\partial}{\partial b}(\frac{a}{b})b' = \frac{1}{b}a' - (\frac{a}{b^2})b'$. Therefore, IMGP can calculate $\frac{\partial y}{\partial c}$ by traversing the program tree in post-order. However, it needs to calculate both the outputs and the partial derivatives of the tree nodes, so the computation cost is multiplied by the number of constants. To save computation, gradient descent is not used on every program tree. Instead, it is used only when IMGP finds a new best program tree. Gradient descent for GP is also used in [125], which it uses gradient descent for all the individuals.

One of the major concerns of classification problems is generalization or overfitting [49]. Due to the noise in the training and the testing data, a classifier works well on the training data may not be so accurate on the testing data. A reason for the performance degradation is that the classifier is so complex that it classifies noisy data unnecessarily. As described in Appendix A, a common approach to enhance generalization is to trade the classification accuracy for the model complexity, and so the penalty of the model complexity is extensively incorporated in the objective function of supervised learning. IMGP adds a penalty of the tree size to the original classification error as the new fitness shown in Eq. 4.7.

$$fitness = error + w\frac{tree\ size}{M\_ROW} \tag{4.7}$$

w is a small positive constant to control the weighting of the tree size. Because the classification error is always less than 1, the tree size is normalized with the maximum tree size, i.e., M_ROW. In calculating the tree size, IMGP does not count the terminal nodes, since only the functions contribute to the complexity of the program. The linear functions of + and – represent linear models which are simple enough, so they are not counted either. Therefore, in the current instruction set $\{x_1, \ldots, x_m, c, +, -, \times, /\}$ of IMGP for classification, only the nonlinear functions $\times$ and / are counted in the tree size.

IMGP is tested on four benchmark binary classification problems in UCI repository [14], i.e., Breast Cancer Wisconsin (Cancer), Haert Disease (Heart), Pima Indians Diabetes (Pima) and Horse Colic (Horse). The experiment adopts a five-fold method. For each fold, IMGP is run for 20 times of different random seeds, and so there are all together 100 independent runs. Besides the input attributes and the constants, IMGP uses only the basic algebra operators, i.e., $\{+, -, \times, /\}$. Grammar Guided Genetic Programming (G^3P) [118] uses context-free grammars and cellular encoding to evolve four kinds of classifiers, i.e., Decision Tree ($G^3P - DT$), Fuzzy Rule-based System ($G^3P - FRBS$), Artificial Neural Networks ($G^3P - ANN$) and Fuzzy Petri-Net ($G^3P - Petri$). Both G^3P and IMGP use the populations of 2000 individuals, and the maximum generations are 100.

Table 4.6 shows the results of the training and testing errors, reported in their best and average values. Table 4.7 shows the p-values of the t-test of the average testing errors of IMGP with G^3P. The hypothesis that the two average testing errors are the same is rejected at the significant level 0.05. For each problem, the result of G^3P is the average of the results from three different settings in [118]. For the average training errors, the results of IMGP are comparable to the four other algorithms. This verifies the effectiveness of IMGP on the training data. The testing error is a more important measure of performance than the training error. IMGP generalizes very well compared to the four other algorithms. IMGP is significantly better than another algorithm if the corresponding p value is smaller than the cutoff value 0.05. For the cancer dataset, IMGP is statistically better than FRBS, NN and Petri net. For the pima dataset, IMGP is statistically better than DT, NN and Petri net. For the heart dataset, IMGP is statistically better than FRBS, NN and Petri net. For the horse dataset, IMGP is statistically better than all the other algorithms. For the best testing error, IMGP has the 2nd lowest best testing error on the cancer problem, and the lowest best testing errors on the three other problems.

IMGP is also compared to some traditional classifiers other than GP, including, Decision Tree, Neural Networks and Support Vector Machine (SVM). In this

Problem		Training Error (%)					Testing Error (%)				
		DT	FRBS	ANN	Petri	IMGP	DT	FRBS	ANN	Petri	IMGP
Cancer	best	**0.28**	**0.28**	1.43	1.14	1.02	1.72	1.72	**1.14**	2.29	1.43
	avg	2.08	**1.56**	6.21	3.07	1.99	4.29	4.58	6.41	4.52	**3.81**
Pima	best	**15.7**	**15.7**	22.51	20.41	21.30	19.89	21.98	22.51	23.03	**15.58**
	avg	21.85	**18.98**	24.27	23.73	23.14	30.60	25.17	26.24	27.37	**24.90**
Heart	best	**11.79**	14.62	31.83	19.43	15.87	17.9	18.34	19.65	21.83	**17.39**
	avg	**15.29**	16.34	23.88	20.77	19.03	23.10	23.85	24.03	25.62	**22.14**
Horse	best	16.11	15.55	16.66	16.67	**10.38**	28.88	27.77	28.88	25.55	**16.18**
	avg	26.39	21.27	31.09	22.46	12.03	37.25	39.01	36.79	40.12	**20.74**

Table 4.6: Comparison of IMGP and G^3P on the 4 UCI benchmark classification problems. DT, FRBS, ANN and Petri are the 4 GP approaches. We compare the training and testing errors in their best and average results. The errors are reported in unit of percentage

G^3P / IMGP	DT	FRBS	ANN	Petri
Cancer	0.1875	0.0103	6.4927×10^{-8}	0.0191
Pima	2.8083×10^{-6}	0.6780	0.0165	0.0011
Heart	0.0556	2.2265×10^{-4}	9.2068×10^{-6}	1.4575×10^{-6}
Horse	1.3474×10^{-8}	1.1392×10^{-9}	2.4510×10^{-8}	3.1871×10^{-10}

Table 4.7: The p-values of the t-test of the average testing errors of IMGP with G^3P on the four benchmark problems. The hypothesis that the two average testing errors are the same is rejected at the significant level 0.05

experiment, the decision tree used is C5.0 [94], which is the state-of-the-art of decision tree for classification. The SVM used is LIBSVM [24], which uses cross validation on the training data to determine the parameters of the kernel function. A neural network is implemented based on the library provided by MATLAB. The neural network also uses cross validation on the training data to choose the number of hidden neurons, and then uses an automated weight decay learning on the whole training data. Neural network is sensitive to the initial weights, so the program is executed 10 times with different random seeds. SVM is a deterministic algorithm, so it is executed only once. Decision tree is also a deterministic algorithm, but C5.0 uses random 5-fold for training and testing, so C5.0 is executed 10 times as well. Table 4.8 shows the best and average testing errors of all the algorithms. Table 4.9 shows the respective p-values of the t-test of the average testing errors of IMGP and other algorithms on the four benchmark problems. The hypothesis

Problem	best testing error(%)				average testing error(%)			
	IMGP	DT	NN	SVM	IMGP	DT	NN	SVM
Cancer	**1.43**	4.30	*4.14*	*4.14*	**3.81**	5.57	5.63	*4.14*
Heart	**17.39**	20.40	*17.41*	*17.41*	22.14	22.62	*20.00*	**17.41**
Pima	**15.58**	24.60	*22.00*	23.44	24.90	26.41	**22.88**	*23.44*
Horse	*16.18*	**14.70**	23.53	23.44	20.74	**14.70**	28.38	*17.65*

Table 4.8: Comparison of IMGP, Decision tree (DT), Neural Networks (NN) and Support Vector Machine (SVM) on the 4 UCI benchmark problems. The best and average testing errors in unit of percentage are reported. For the deterministic algorithms, i.e., DT and SVM, the best and average results are the same. DT was not run for Horse since the program C5.0 was unable to handle the training and testing datasets separately

IMGP \ Algorithms	DT	NN	SVM
Cancer	5.5013×10^{-33}	1.8441×10^{-4}	9.2725×10^{-4}
Pima	1.7374×10^{-6}	3.7240×10^{-5}	0.0016
Heart	2.5291×10^{-28}	4.0063×10^{-4}	1.4544×10^{-31}
Horse	0.0038	0.0013	0.1083

Table 4.9: The respective p-values of the t-test of the average testing errors of IMGP and other algorithms on the four benchmark problems. The hypothesis that the two average testing errors are the same is rejected at the significant level 0.05

that the two average testing errors are the same is rejected at the significant level 0.05. For the convenience of comparison, the result of SVM is treated as both the best and the average results. IMGP has the smallest best testing errors on all the problems except Horse, where C5.0 is the best. The average testing errors of IMGP are still acceptable and comparable to those of the other algorithms. On the cancer dataset in particular, IMGP is significantly better than the other algorithms, since the corresponding p values are smaller than 0.05.

4.4.1 Sparse Kernel Feature Machine

For the problem whose objective is complicated and costly to evaluate, it is better to use a deterministic algorithm to search for a satisfactory solution. A Sparse Kernel Feature Machine (SKFM) is designed to carry out kernel learning and feature selection simultaneously. First, kernel mutual information is used to filter out irrelevant features in high-dimensional problems. Then, an augmented kernel matrix is composed of the kernel matrices of individual dimensions. Afterwards, a Least Angle Regression without collinearity is performed on the augment kernel matrix to build the solution path w.r.t. the regularization parameter efficiently. The best regularization parameter giving the smallest validation error is selected for further training. In contrast to the standard kernel learning, the selected supporting points contain a single dimension each, and thus SKFM selects important features as well as the deciding values on those features. Empirical results on the real testing datasets show that SKFM not only trains better classifiers than SVM, but it also identifies the relevant features and the corresponding values in the resulted classifiers. For more details on SKFM, please refer to Appendix A.

4.5 Discussion

Instruction Matrix based Genetic Programming (IMGP) maintains Instruction Matrix (IM) to store the fitness of the instructions and their best subtrees. It extracts program trees from IM, updates IM with the fitness of the extracted program trees, performs crossover and mutation on the extracted program trees, and shuffles IM to propagate good instructions. The experimental results have verified its effectiveness and efficiency on the benchmark problems. It is not only superior to CGP in terms of the qualities of the solutions and the number of program evaluations, but it also outperforms the related GP algorithms on the tested problems.

IMGP can also be used for classification problems. To enhance its performance, IMGP uses gradient descent to find the optimal constants in program trees, and incorporates the penalty of program tree complexity in the fitness. In most of the tested problems, IMGP is able to find classifiers of higher classification accuracies than four other GP classifiers. The results of IMGP are also comparable to or better than those of Decision Tree, Neural Networks and Support Vector Machine.

IMGP is an implementation of the EDDA framework for GP. By evolving instructions separately, IMGP actually decomposes a high dimensional problem into small problems of only one dimension. Therefore, both the size of the solution space and the search time is reduced significantly. At the same time, it also maintains the interdependencies between instructions in the form of the links of best subtrees, and thus it is likely that the combination of the optimal instructions is the optimal program tree to the original problem.

Furthermore, IMGP can be viewed as evolving schemata directly [73]. The schema theory originally explained why GA works. It was extended to explain the mechanism of GP later. By maintaining the average and the best fitness of the instructions and the subtrees, IMGP is able to maintain most of the information of the schemata, and make use of the information to evolve schemata directly. The details of the schema theory is explained in Section 4.5.1.

4.5.1 Schema Evolution

Schema theory [51] was original used to explain how and why Genetic Algorithm (GA) works. Schemata are the abstractions of the common patterns in the population. In binary GA, a schema is a vector of characters of $\{0, 1, \#\}$. The wildcard character # is a "don't care" symbol, which maps to either 0 or 1. The number of non-# symbols is called the *order* $\mathcal{O}$ of the schema. The distance between the furthest two non-# symbols is called the *defining length* $\mathcal{L}$ of the schema. An individual contains many schemata, and a schema matches many individuals. The number of individuals matching a good schema increases exponentially through generations as shown in Inequality 4.8. It is postulated that while GA is evolving the population, it is actually looking for the common schemata of the optimal solutions.

$$m(H,t+1) \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)}(1-p_m)^{\mathcal{O}(H)} \times [1-p_c\frac{\mathcal{L}(H)}{N-1}(1-\frac{m(H,t)f(H,t)}{M\bar{f}(t)})] \tag{4.8}$$

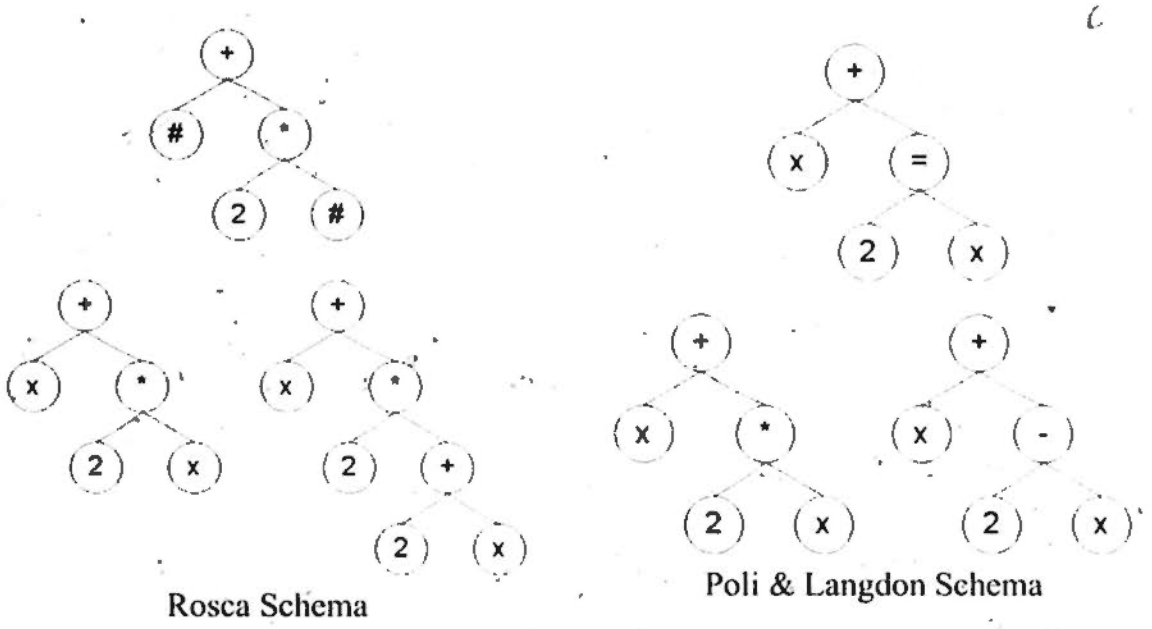
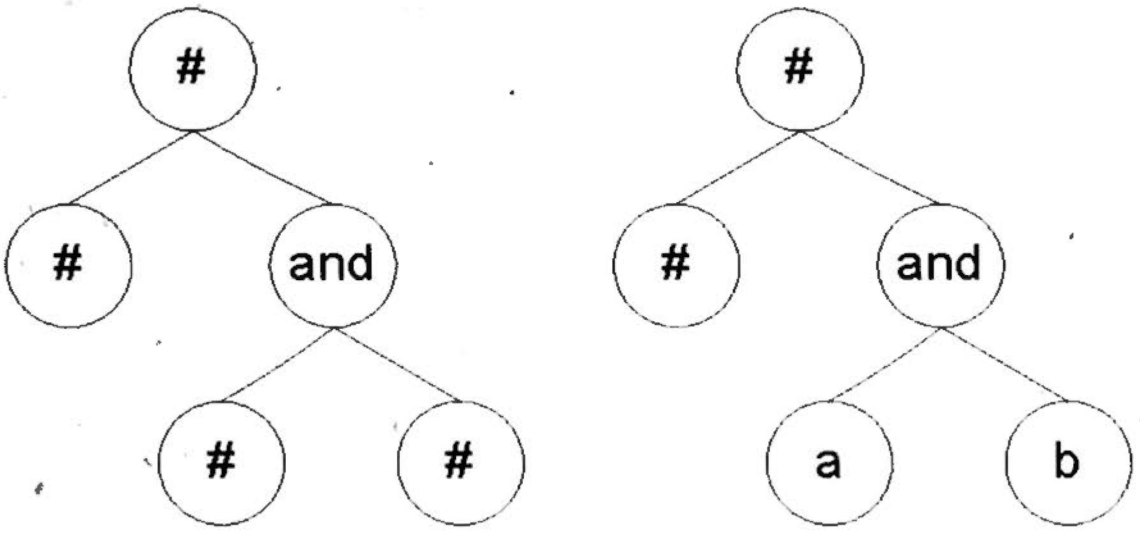


Figure 4.8: The schemata in Canonical Genetic Programming

Rosca [101] and Poli & Langdon [64] introduce two schema theories for GP independently. Their schema is a contiguous tree fragment starting from the tree root. The fragment consists of the nodes of fixed values or the "don't care" symbols. A tree has only one instance of a certain schema and the position of the schema is fixed. However, the "don't care" symbol # in Rosca's schema theory represents a set of subtrees, while the "don't care" symbol $=$ is exactly one tree node in Poli & Langdon's schema theory. Fig. 4.8 illustrates these two position schema theories.

IMGP actually maintains the information of some kinds of schemata. In IM, an instruction's fitness is averaged over the fitness of all the trees containing it at the fixed position. Considering Poli & Langdon's schema theory, we think the fitness of *AND* at the third row of IM in Fig. 4.2 is ~~actually~~ the fitness of the schema $(== AND ====)$, which has *AND* as the root of the right subtree, whose left and right subtrees can be anything except -1. Generally, the fitness of an instruction in IM is the fitness of the order 1 schema with the instruction at the corresponding position in the tree. This way, IMGP maintains the fitness of all the order 1 schemata. Additionally, an instruction has its best fitness together with its best subtree. Suppose the function *AND* at the third row of the matrix in Fig. 4.2 has its best



A Node Schema A Subtree Schema

Figure 4.9: The schemata in Instruction Matrix based Genetic Programming

left child pointing to A, and the best right child pointing to B, then the best fitness of AND is actually the best fitness of the schema $(== AND == AB)$. Its root can be any function, its left subtree can be anything except -1, and its right subtree is $(ANDAB)$. This way, IMGP is able to remember the best fitness of some schemata of order larger than 1. Fig. 4.9 shows these two schemata of the example in Fig. 4.2.

Therefore, IMGP can be regarded as evolving schemata directly. According to the extraction criterion in section 4.2.2, if an instruction's fitness is better than the others', which means its 1-order schema is better than the other 1-order schemata with different instructions at the same position, it will be selected more often than the other instructions, i.e. more programs will sample its schema. Similarly, if an instruction's best fitness is much better than its average fitness, this will not only increase the chance of selecting this instruction, but if it is indeed selected, more trees will sample the schema containing its best subtree. On the other hand, the fitness of a schema is implicitly evaluated by updating the average and the best fitness of the corresponding instructions.

4.5.2 Algorithm Complexity

The complexity of IMGP is no larger than that of CGP. Besides the normal genetic operators and fitness evaluation in CGP, IMGP incurs additional overhead of extracting individuals, updating IM and shuffling IM. First of all, the time complexity of genetic operators is $O(H)$, where H is the height of the IM, i.e., the size of the program tree. Secondly, extracting individuals and updating IM need to go through all the instructions in the program trees, so the complexity is also $O(H)$. Thirdly, the complexity of shuffling IM is of the size of IM, i.e., $O(WH)$, where W is the width of the IM. Forth, the time complexity of evaluating the fitness is $O(NH)$, where N is the number of times to traverse tree programs. For some problems, N is number of training cases. Suppose IMGP runs for G generations, and extracts P program trees in each generation, then the overall time complexity is $O(GPH + GWH + GPNH)$. The time complexity of CGP is $O(GPH + GPNH)$. In the practice, for both IMGP and CGP, most of computation cost is in evaluating the fitness, whose complexity is $O(GPNH)$. For program tress with similar sizes, the time complexity of IMGP is only slightly larger than that of CGP. However, the space complexity of IMGP is smaller than that of CGP. The major part of the space complexity of CGP comes from the population of individuals, i.e., $O(PH)$, while the major part of the space complexity of IMGP comes from IM, i.e., $O(WH)$. Usually W is of order of hundreds, while P is of order of thousands.

Chapter 5

Computational Motif Discovery

5.1 Introduction

In this chapter, EDDA is applied to solve a real bioinformatics problem, i.e., motif discovery. Transcription factor binding sites (TFBS) are small nucleotide fragments (usually ≤ 30 bp) in the cis-regulatory regions of genes in DNA sequences. TFBS are crucial in gene regulation, the understanding of which is a central problem in contemporary biology. Finding the pattern of TFBSs, i.e. motif discovery in DNA sequences, is thus important for uncovering the underlying regulatory relationship and the evolutionary mechanism of living organisms. Computational methods provide promising results for further biological validations which alone are expensive and laborious. However, computational motif discovery is a well-known challenging problem because of the low signal-to-noise ratio due to both weak conservation and short motif widths. Although additional evidence, such as expression data and phylogenetic information, can be incorporated to help recognizing some noisy sequences without motifs, the fundamental problem of finding TFBSs on the sequence level is still very difficult for computational methods. One major challenge is the difficulty of searching for the global optimum in a high dimensional space. Numerous algorithms, typically consensus-based search algorithms and statistical optimization methods, have been proposed. Consensus search algorithms suffer from the insufficient descriptive power of string patterns and are limited by the motif width and the maximal error they can handle. Statistical methods are significantly affected by their starting points and often trapped in local optima. Population-based evolutionary algorithms perform a rather time consuming search and evaluate a large number of useless candidate solutions.

In this chapter, two new algorithms for motif discovery is proposed. The first is a new Estimation of Distribution Algorithm for Motif Discovery (EDAMD) which handles more general assumptions for TFBS identification/motif discovery. EDAMD relaxes the simplified assumption of one instance per sequence in the collected sequences. The objective of EDAMD is to search for the optimal Position Frequency Matrix (PFM) and the corresponding motif instances in DNA sequences. EDAMD models the PFMs of the sampled motif instances as a weighted Gaussian distribution, which is able to capture the possible pairwise dependencies between the probabilities of the positions in the corresponding PFMs. New PFMs are sampled from the Gaussian distribution. Moreover, a local searching technique inspired by Gibbs sampling [76][68] and local filtering techniques [23] refines the sampled PFMs efficiently. With the EDA output, a post processing procedure improves the results to be even more accurate and complete. Experimental results show that the results of EDAMD are comparable to or better than two other GA-based algorithms, namely GAME and GALF.

It is observed that EDAMD always gets the same results even with different random seeds. Therefore, a new deterministic approach called Cluster Refinement Algorithm for Motif Discovery (CRMD) is designed. CRMD manages to locate the local optimal solutions efficiently and effectively and identify the global optimum from a small number of local optima. CRMD employs a flexible statistical model of motif which allows a variable number of motifs and motif instances. First CRMD uses a novel entropy-based clustering method to find a set of complete and good starting candidate motifs from the input sequences. Then it employs a fast refinement method to search for optimal motifs from the candidate motifs. The clustering chooses informative motif candidates of various types, where the probable initial solutions are maintained and those non-informative ones are discarded to reduce the search space significantly. The refinement method incorporates a greedy sampler to obtain the optimal motif instances from the initial candidate motifs, and it returns a variable number of motif instances by removing or adding motif instances adaptively according to the auto-adjusted thresholds. CRMD can be easily extended if prior knowledge, such as One Occurrence Per Sequence (OOPS), is available. Endowed with an appropriate similarity test of motifs, CRMD is also capable of discovering multiple distinct motifs.

In the experiments, CRMD has the best results on most of the 800 synthetic datasets of a comprehensive range of difficulties. The results on the extensive real datasets, including a set of eight selected real datasets, ABS database [15], SCPD database [128], *Escherichia coli* datasets [53] and Tompa's datasets [117], also show that CRMD seldom falls into local optima as MEME [6] and Motif Sampler [77] do, and its performance is even better than or competitive with those of GAME [119] and GALF-P [22]. GAME and GALF-P are time consuming Genetic Algorithm based motif discovery approaches and are supposed to locate the close-to-optimal binding sites. If the OOPS assumption is assumed, the qualities of the results of CRMD can be further improved. For a real multiple motif problem, CRMD locates a significantly larger number of binding sites than MEME and Motif Sampler. In addition, CRMD has shorter running time than most of the other algorithms tested in this chapter even if it is implemented in MATLAB and executed on Windows.

The rest of this chapter is organized as follows. In Section 5.2, the background of the motif discovery problem and the existing methods are briefly introduced. In Section 5.3, the problem details are given and formulated mathematically. Section 5.4 and Section 5.5 describe the algorithms and the experimental results of EDAMA and CRMD in detail, respectively. The last section is the discussion.

5.2 Existing Algorithms

TFBSs interact with transcription factors (TFs) and affect the transcriptional activity (or gene expression). The cis-regulatory regions are usually upstream to the transcription start sites (TSS) of the genes. TFBSs typically have a width of 5-10 bp, but there are also real cases such as the CRP binding sites with widths up to around 20 bp. In general, the range of widths can be restricted to around 5 bp to 25 bp. Some well-known characterized TFBSs such as the TATA box are proximal to the TSS, but generally there is no prior spatial knowledge of where the TFBSs occur in the regulatory regions.

Computational methods for identifying TFBSs, namely *de novo* motif discovery, have been proposed as an attractive pre-screening procedure and alternative to the expensive and laborious biological experiments such as DNA footprinting [41] and ChIP-chip [52]. The basis is that certain conserved pattern, called the "motif", exists among the TFBSs in the cis-regulatory regions for a set of similarly expressed genes (co-expressed genes), because those genes are probably regulated by the same or similar TFs. Benefitting from the availability of the large amount of sequencing and microarray data, now we can identify co-expressed genes by clustering and then extract their cis-regulatory regions. *de novo* motif discovery methods try to identify the motif, or equivalently the set of TFBS instances of co-expressed genes without prior knowledge about their consensus appearance.

There have been a few excellent surveys of motif discovery algorithms [117][53][104]. Current motif discovery methods can be categorized into enumerative (consensus based) approaches and statistical (matrix based) ones. They either discover the string pattern (the consensus) using combinatorial approaches or identify the profile of the TFBSs, typically the Position Frequency Matrix (PFM), or Position Weight Matrix (PWM), using statistical modelling.

In enumerative approaches for motif discovery, exact string matching methods fail and exhaustive enumeration is also infeasible due to the NP-hardness [74]. Existing consensus based approaches [90], set the constraint that the maximal hamming distance between the consensus and the motif instances, d, is assumed to be known. They try to enumerate all the strings satisfying the constraints in polynomial time. Typical works include optimizing the data structure using suffix trees [102][12] and projections [20][97]. However, such approaches cannot meet the requirements of real world problems well because they can only handle short motif widths (in general up to 14) and small d within reasonable computational time. In the real cases, however, the width can be up to 22 (in the CRP dataset tested in this chapter). d is also difficult to determine beforehand and it varies case by case. With too small a d, most of the true TFBSs are missed due to the stringent criteria. With too large a d, the computation time becomes intolerable and a large number of false positives will be output. Another major drawback of consensus based approaches is that the discrete consensus of the motif is not accurate enough to represent the weak conservation between different nucleotides.

A more accurate choice is to use the PFM and PWM to represent a motif with continuous frequency or likelihood of each nucleotide appearing at each position within the motif. Some statistical methods such as Expectation Maximization [6][16] and Motif Sampler [76][68] have been proposed and shown some successes in TFBS identification. However, since statistical methods sample TFBSs probabilistically, they may take a long time for their solutions to converge and stabilize. Another disadvantage is that they are sensitive to initial settings, and are often trapped in local optima since many of these methods perform local search only, and their results might not even be local optimal if the searching is ineffective. In TFBS identification, the problems of being trapped in local optima become more critical because the weakly conserved TFBSs are typically weak signals surrounded by a large amount of noise.

Genetic Algorithms (GA) [119][22][35][79][72] have been applied to TFBS identification as well. The advantage of such GA based methods is that they are likely to locate the global optimum in a typically difficult search space. Other advantages of GA compared with the conventional motif discovery methods include the flexibility of representations and scoring functions in which advanced models can be easily incorporated, and good scaling property which is promising for the large amount of data in DNA sequences. On the other hand, they are stochastic and so they may fail to report consistent results in different runs. They require a large population of solutions and the computation time is typically long. Nevertheless, the results of the state-of-the-art GA-based method provide a close-to-exhaustive-search-based benchmark to evaluate the performance of motif discovery algorithms.

Recently, approaches incorporating multiple evidence besides the DNA sequences have been proposed to improve the prediction accuracy for real motif discovery problems. Recent reviews usually include these integrated approaches [28][45]. The evidence generally comprises of microarray data for the input sequences, phylogenetic footprinting, ChIP-chip and negative sequences previously known to contain no motifs, just to name a few. Multiple evidence also means additional data sources are needed specifically. While these methods gain success in specific cases, the general motif discovery problem remains challenging because it is usually difficult to have these additional information and the search on sequences known to have certain motifs is still difficult. This work focuses on the motif discovery involving only DNA sequences, and the improvement on DNA sequences alone will certainly further enhance the methods integrated with additional evidence.

5.3 Objective

Biologically, the TFBS identification problem is to locate the subsequences in the cis-regulatory regions which are bound by a common protein. Up to now, the process of factor binding is still obscure to biologists, let alone the properties of the binding sites. To cope with this problem with computational methods, the problem is formulated as an optimization problem of a certain mathematic objective function in the following subsections. The algorithms to maximize the objective function are presented in Sections 5.4 and 5.5.

5.3.1 Problem Formulation

Given a set of DNA sequences, it is required to find the binding sites corresponding to the motif instances and the common string pattern of the motif. To be consistent with the biological observation, there is no assumption of the maximal distance between the motif instances and the number of motif instances in the sequences.

Data Input: a set of sequences $S = \{S_i | i = 1, 2, ..., D\}$ of nucleotides defined on the alphabet $B = \{A, T, G, C\}$. $S_i = (S_i^j | j = 1, 2, ..., l_i)$ is a sequence of nucleotides, where l_i is the length of the sequence.

The motif width is w nucleotides long, which is assumed known throughout the chapter. The set of all the w long subsequences contained in S is $\{s_i^{j_i} | i = 1, 2, ..., D, j_i = 1, 2, ..., l_i - w + 1\}$, where j_i is the binding site of a possible motif instance $s_i^{j_i}$ on sequence S_i.

Position Output: the Position Indicator Matrix (PIM) $A = \{A_i | i = 1, 2, ..., D\}$ of the motif, where $A_i = \{A_i^j | j = 1, 2, ..., l_i\}$ is the indicator row vector with respect to (w.r.t.) a sequence S_i. A_i^j is 1 if position j in S_i is a binding site, and 0 otherwise. The number of motif instances is referred to as $|A| = \sum_{i=1}^{D} \sum_{j=1}^{l_i} A_i^j$.

Induced by A is a set of $|A|$ motif instances denoted as $S(A) = \{S(A)_1, S(A)_2, ..., S(A)_{|A|}\}$, where $S(A)_i = S(A)_i^1 S(A)_i^2 ... S(A)_i^w$ is the ith motif instance in $|A|$. $S(A)$ can also be expanded as $(S(A)^1, S(A)^2, ..., S(A)^w)$, where $S(A)^j = S(A)_1^j S(A)_2^j ... S(A)_{|A|}^j$ is the list of the nucleotides on the jth position in the motif instances.

Consensus Output: the string abstraction of the motif instances or, in the absence of a string consensus, the Position Count Matrix (PCM) $N(A)$ of the numbers of different nucleotide bases on the individual positions of the motif instances of A. $N(A) = (N(A)^1, N(A)^2, ..., N(A)^w)$, and $N(A)^j = \{N(A)_b^j | b \in B\}\}$, where $N(A)_b^j = |\{S(A)_i^j | S(A)_i^j = b\}|$.

(a) sequences S	(b) PIM A	(c) instances $S(A)$	(d) PCM $N(A)$	(e) PFM $\hat{N}(A)$
acgtCGATTGCctaag	0000100000000000	CGATTGC		
taTGATCGAtgacgca	0010000000000000	TGATCGA	A:0261107	A: 0.0 0.2 0.6 0.1 0.1 0.0 0.7
cgaCAATTGAgcttac	0001000000000000	CAATTGA	C:8023323	C: 0.8 0.0 0.2 0.3 0.3 0.2 0.3
gCGCTCGAcaagctgt	0100000000000000	CGCTCGA	G:0800080	G: 0.0 0.8 0.0 0.0 0.0 0.8 0.0
cgttTGTCACAgtcta	0000100000000000	TGTCACA	T:2026600	T: 0.2 0.0 0.2 0.6 0.6 0.0 0.0
tcagcCACACCCagct	0000010000000000	CACACCC		
ccagagCGTCTGAttg	0000001000000000	CGTCTGA		
gacttcaCGACTGAgc	0000000100000000	CGACTGA	$M(S)_A$:38 $M(S)_C$:47	θ_{0A} 0.2375 θ_{0C} 0.2938
gctgcccatCGATTGA	0000000001000000	CGATTGA	$M(S)_G$:38 $M(S)_T$:37	θ_{0G} 0.2375 θ_{0T} 0.2313
ccaggtacCGATTGCa	0000000010000000	CGATTGC		

Figure 5.1: An artificial problem of motif discovery. It shows (a) the sequences S, (b) the Position Indicator Matrix A, (c) the motif instances $S(A)$, (d) the Position Count Matrix $N(A)$ and the count of the background nucleotides $\{M(S)_b | b \in B\}$, (e) the Position Frequency Matrix $\hat{N}(A)$ and the background relative frequencies $\{\theta_{0b} | b \in B\}$. In the sequences S, the letters in lower case are the background bases, and the letters in upper case are the motif instances

$N(A)$ can be further normalized by $|A|$ as the Position Frequency Matrix (PFM) $\hat{N}(A) = \frac{N(A)}{|A|}$, which can be regarded as a virtual consensus, i.e., the relative frequencies of the nucleotide types on the individual positions in the motif instances. Given an A, it is trivial to calculate $N(A)$. On the contrary, it is not straightforward to find the corresponding A from $N(A)$.

Fig. 5.1 illustrates an artificial motif discovery problem. $M(C) = \{M(C)_b | b \in B\}$ denotes the numbers of different nucleotides in the dataset C, where $M(C)$ applies to all the positions in C. Similarly to PFM, $M(S)$ can be normalized as the relative frequencies of the nucleotides in the sequences S, which is denoted as $\theta_0 = \{\theta_{0b} = \frac{M(S)_b}{\sum_{b \in B} M(S)_b} | b \in B\}$.

5.3.2 Maximum A Posteriori

In a motif discovery problem, it is required to find the optimal PIM A or PCM $N(A)$ in terms of a certain optimization measure. There are various methods to evaluate a set of candidate motif instances. The Bayesian analysis is adopted to derive the posterior probability of the motif instances, and thus the motif discovery is to find the motif instances of the maximal probability. To make it easy to understand the proposed algorithms, the major steps of the derivation in [56] are repeated herein.

For the likelihood of the motif instances, it is usually assumed that the nucleotides in a motif instance are generated independently across positions. Therefore, the motif instances A follow the multinomial distribution $\prod_{j=1}^{w} p(N(A)^j)$, where $p(N(A)^j)$ is the independent probability of generating the nucleotides on the jth position of the motif instances. It is further assumed that the probabilities of generating the nucleotides on a position of the different motif instances are independent. The joint probability $p(N(A)^j)$ is thus the product of the probabilities of the nucleotides on position j in the sequences respectively, i.e., $p(N(A)^j) = \prod_{b\in B} \theta_{jb}^{N(A)^j_b}$, where θ_{jb} is the latent probability of generating base b in position j, $N(A)^j_b$ is the number of nucleotide b on position j. In a more succinct form, $\prod_{b\in B} \theta_{jb}^{N(A)^j_b}$ can be written as $\theta_j^{N(A)^j}$, where θ_j is the vector of the latent probabilities $\{\theta_{jb}|b \in B\}$ on position j in the motif instances. In summary, the motif instances A follow the multinomial distribution $\prod_{j=1}^{w} \theta_j^{N(A)^j}$.

For the likelihood of the background sequences, it is assumed that the nucleotides on the sequences excluding the motif instances follow a multinomial distribution $\theta_0^{N(S(A^C))} = \prod_{b\in B} \theta_{0b}^{N(S(A^C))_b}$, where θ_0 is the vector of the probabilities generating the background nucleotides and A^C is the complement of A w.r.t. S. In this chapter, it is assumed θ_0 is fixed as the relative frequencies of the bases in S, which is indifferent to the positions of the bases. Similarly, it is also assumed an independent binomial distribution of the number of motif instances $|A|$, i.e, $p(A|p_0) = p_0^{|A|} \times (1 - p_0)^{L-|A|}$, where $L = \sum_{i=1}^{N} (l_i - w + 1)$ is the total number of the subsequences and p_0 is an abundance ratio to indicate the probability of a position being a binding site.

The PIM A can be viewed as the missing label of the data S, θ is the latent parameters of the distribution model of A, and p_0 is also unknown beforehand. The likelihood of S is the product of the probabilities of the background sequences and the motif instances as follows,

$$p(S|\theta,\theta_0,A,p_0) = p_0^{|A|}(1-p_0)^{L-|A|}\theta_0^{M(S(A^C))}\prod_{j=1}^{w}\theta_j^{N(A)^j}$$

For Bayesian analysis, a multinomial Dirichlet distribution is employed as the conjugate prior for θ, i.e., $p(\theta|\alpha) \propto \prod_{j=1}^{w}\prod_{b\in B}\theta_{jb}^{\alpha_b-1}$, where α is a small common prior for all the θ_js. A Dirichlet distribution is also prescribed as the conjugate prior for p_0, i.e. $p(p_0|p_a,p_b) \propto p_0^{p_a-1}(1-p_0)^{p_b-1}$. Therefore, the posterior distribution of A, θ and p_0 is as follows, where we have used $\theta_0^{M(A^C)} = \frac{\theta_0^{M(S)}}{\theta_0^{M(A)}} \propto \frac{1}{\theta_0^{M(A)}}$.

$$\begin{aligned}
&p(\theta,A,p_0|S,\theta_0,\alpha,p_a,p_b)\\
&= p(S|\theta,\theta_0,A,p_0)p(A|p_0)p(\theta|\alpha)p(p_0|p_a,p_b)\\
&= \frac{p_0^{|A|+p_a-1}(1-p_0)^{L-|A|+p_b-1}}{\theta_0^{M(S(A))}}\prod_{j=1}^{w}\theta_j^{N(A)^j+\alpha-1}
\end{aligned}$$

θ and p_0 can be integrated out using the conversion between the beta function and the gamma function[1]. The resulted posterior conditional distribution of A alone is shown in Eq. 5.1, which has used $|\alpha| = \sum_{b\in B}\alpha_b$ and $\sum_{b\in B}N(A)_b^j = |A|$.

$$\begin{aligned}
p(A|S,\theta_0,\alpha,p_a,p_b) &\propto \int p(\theta,A,p_0|S,\theta_0,\alpha)d\theta dp_0\\
&= \frac{\Gamma(|A|+p_a)\Gamma(L-|A|+p_b)}{\theta_0^{M(S(A))}}\prod_{j=1}^{w}\frac{\prod_{b\in B}\Gamma(N(A)_b^j+\alpha_b)}{\Gamma(|A|+|\alpha|)} \qquad (5.1)
\end{aligned}$$

The objective of motif discovery can thus be formulated as to maximize the posterior probability of A in Eq. 5.1. Prior knowledge, such as the abundance of motif instances in the dataset, the background frequencies of the nucleotide types and the probabilities of nucleotides in the motif instances, can be easily incorporated in the model.

[1]The beta function $B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt$, the gamma function $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$, and $B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$

In the implementation of Cluster Refinement Algorithm for Motif Discovery in Section 5.5, the gamma function Γ frequently causes overflow with a large argument, so $p(A)$ in Eq. 5.1 is not used as the measure of A directly. Instead, the log of the posterior probability of A in Eq. 5.2 is used, where the log gamma function $L\Gamma$ avoids overflow in typical cases and K is invariant with A.

$$\begin{aligned}
log(p(A|S,\theta_0,\alpha,p_a,p_b)) &= K + \\
& L\Gamma(|A| + p_a) + L\Gamma(L - |A| + p_b) - M(S(A))log\theta_0 + \\
& \sum_{i=1}^{w}\sum_{b\in B} L\Gamma(N(A)^i_b + \alpha_b) - wL\Gamma(|A| + |\alpha|)
\end{aligned} \tag{5.2}$$

If the log gamma function is not used, Eq. 5.1 can still be simplified using the Burnside approximation[2]. After some tedious derivation, the log of the approximated posterior conditional probability of A is shown as Eq. 5.3. Instead of Eq. 5.1, Eq. 5.3 is used as the objective function in Estimation of Distribution Algorithm for Motif Discovery in Section 5.4.

$$\begin{aligned}
\psi(A) &= log(p(A|S,\theta_0,p_0,\alpha)) \\
&\approx K + |A|(log(p_0) - log(1 - p_0)) - M(A)log(\theta_0) \\
&\quad + \sum_{i=1}^{w}(N(A)^i + \alpha - 0.5)log(N(A)^i + \alpha - 0.5) \\
&\quad - w(|A| + |\alpha| - 0.5)log(|A| + |\alpha| - 0.5)
\end{aligned} \tag{5.3}$$

where K is invariant w.r.t all the variables. For vectors v and v', $vlog(v')$ is used as a shorthand for $\sum v_i log(v'_i)$. After further simplification, Eq. 5.3 contains a term of Information Content (IC), namely $\prod_{i=1}^{w}\theta_i\log\frac{\theta_i}{\theta_0}$, as used in other algorithms, such as GAME [119] and GALF [23]. IC models the discriminatory motif since it

[2]Burnside approximation $\Gamma(x+1) = x! \approx (x+0.5)^{x+0.5}e^{-x-0.5}\sqrt{2\pi}$

is an approximation of the difference between the log probability of motif w.r.t. the latent probabilities θ and the one w.r.t. the background probabilities θ_0. However, IC neglects the variable number of motif instances, and the simplification from Eq. 5.3 might be too coarse.

5.4 Estimation of Distribution Algorithm for Motif Discovery

Estimation of Distribution Algorithm for Motif Discovery (EDAMD) identifies the motif instances which maximize the objective function Eq. 5.3. In a nutshell, EDAMD consists of two levels. On the outer level, EDAMD employs a Gaussian distribution to model the Position Frequency Matriics (PFM) of the individuals in the population. The Gaussian distribution not only maintains the probabilities of the nucleotides on the positions of the motif consensus, but also capture the possible pair-wise dependency between the positions of the motif consensus. In the evolution, EDAMD updates the Gaussian distribution with the motif instances in the population, and it generates new PFM based on the Gaussian distribution to find the corresponding potential motif instances. On the inner level, each PFM generated from the Gaussian distribution is further refined by a local search heuristic to find the local optimum around the initial PFM. While the outer level ensures EDAMD to search the solution space thoroughly w.r.t. the PFM distribution, the second level makes EDAMD considers the local optimal PFMs only which may contain the global optimal solution.

Algorithm 5.1 is the overall program of EDAMD. For a generation, half of the PFMs are sampled from a Gaussian distribution $\mathscr{G}$. To enhance the diversity of PFMs, the other half are sampled from a uniform distribution $\mathscr{U}$. After finding a set of motif instances $A_{(i)}$ based on a sampled PFM $\hat{N}(\tilde{A})_{(i)}$ via the Greedy Refinement function *Greedy*, the best set of motif instances BA is updated if $\psi(A_{(i)})$ is better than the best fitness FIT. At the end of the generation, the function $Update$ updates the Gaussian distribution model with the sets of motif instances $\{S(A)_{(i)}\}$, their fitness $\{\psi(A)_{(i)}\}$ and the conditional probabilities of the instances $\{\rho(A)_{(i)}\}$. Finally, the Post Processing function $Post$ is applied on BA to adjust the discovered motif instances.

Algorithm 5.1: The Main Program of EDAMD

```
Input: The Sequences S
Output: The Best Motif Instance BA
$FIT \leftarrow 0$;
randomly initialize $\mathcal{G}$;
for $g$ from 0 to $G$ do
    for $i$ from 0 to $T$ do
        if $i < \frac{T}{2}$ then
            $\hat{N}(\tilde{A})_{(i)} \sim \mathcal{G}(x|\mu,\Sigma)$;
        else
            $\hat{N}(\tilde{A})_{(i)} \sim \mathcal{U}(x)$;
        $N(\tilde{A})_{(i)} \leftarrow D \times \hat{N}(\tilde{A})_{(i)}$;
        $[A_{(i)}, S(A)_{(i)}, \psi(A)_{(i)}, \rho(A)_{(i)}] \leftarrow Greedy(N(\tilde{A})_{(i)}, S)$;
        if $\psi(A)_{(i)} > FIT$ then
            $FIT \leftarrow \psi(A)_{(i)}$;
            $BA \leftarrow A_{(i)}$;
    $[\mu,\Sigma] \leftarrow Update(\{S(A)_{(i)}\}, \{\psi(A)_{(i)}\}, \{\rho(A)_{(i)}\})$;
$BA \leftarrow Post(S, BA)$;
```

5.4.1 Searching Method

The inner level is a heuristic local search procedure to find the nearby local optimum around an initial PFM. In ordinary GA, crossover and mutation are the primary searching operator. However, to take advantage of the properties of the motif discovery problem, EDAMD employs more efficient searching operators than the generic GA operators. The first operator Greedy Refinement is a local search mutation. It finds a new set of binding sites based on the initial set of binding sites. The new set of motif instances has a higher posterior probability than the old one.

The second operator Post Processing is applied on the best set of motif instances after the evolution. It retains most of the motif instances, and adds some new motif instances to increase the posterior probability.

Greedy Refinement

Given a motif PIM A, it is easy to calculate its fitness according to $\psi(A)$ in Eq. 5.3. In the other way around, finding the optimal A is maximizing $\psi(A)$. However, solving for the optimum of $\psi(A)$ analytically is intractable. Instead, EDAMD iteratively searches for the optimal A_{ij} (0 or 1), while fixing the rest of A. Moreover, as indicated by the Eq. 5.3, searching for the optimal A is equivalent to searching for the optimal $N(A)$ as long as there actually exists a set of motif instances A whose PCM is $N(A)$.

For the time being, it is assumed that each sequence S_i contains a single motif instance, meaning only a single element in A_i is 1. In that case A collapses to a vector P where P_i is the index of the binding site on S_i.

Given a set of motif instances, which may not be the optima necessarily, EDAMD tries to find better instances. It takes an iterative procedure to refine the motif instances one by one, and the maximal iteration is D. Suppose it has located the binding sites on all the sequences except a single sequence S_i it is working on, it looks for the subsequence on S_i which matches the other instances best. A measure of similarity of a site A_i^j to the other binding sites is the probability of A_i^j being a binding site conditional on the other binding sites. The dissimilarity of A_i^j can be measured as the conditional probability of A_i^j not being a binding site. Therefore, the Bayes factor $\varphi(A_i^j)$ in Eq. 5.12, which is derived using Bayes inference, can be used to determine whether A_i^j is a binding site. The derivation is very similar to the one used in Eq. 5.1, where the equation $\Gamma(x+1) = x\Gamma(x)$ is used. $N(P^*)_b^k$ is the number of nucleotide $b = S_i^{j+k-1}$ in $S(P^*)^k$, meaning the same nucleotide at position k in $S(P^*)$ as the one b in $S(A_i^j)$. $|P^*|$ is the number of motif instances already identified.

$$\begin{aligned}\varphi(A_i^j) &= \frac{p(A_i^j = 1|P', S)}{p(A_i^j = 0|P', S)} \\ &= \frac{\int p(A_i^j = 1|\theta, P', S)p(\theta|P', S)\,d\theta}{\int p(A_i^j = 0|\theta, P', S)p(\theta|P', S)\,d\theta} \\ &\propto \frac{1}{\theta_0^{M(A_i^j)}}\prod_{k=1}^{w}\frac{N(P')_h^k + \alpha_h}{|P'| + |\alpha|} \end{aligned} \qquad (5.4)$$

After calculating the Bayes factors of all the A_i^j on S_i, EDAMD needs to determine which one is the binding site. Gibbs sampling [77] selects a site randomly in proportion to the $\varphi(A_i^j)$ in Eq. 5.12 in S_i. It iteratively samples the binding sites in all the sequences one after another (possibly rewinds to S_1 after sampling on S_N), and updates P' accordingly after each sampling. Gibbs sampling is a Markov Chain Monte Carlo method, and so it may takes a long time before generating samples following the target distribution.

EDAMD adopts a different method to select the binding sites. Instead of sampling the binding sites probabilistically, it selects the site of the maximal $\varphi(A_i^j)$ directly, i.e., $P_i = \text{argmax}_{j=1,\ldots,l_i}\varphi(A_i^j)$. Similar to Gibbs sampling, EDAMD selects the binding sites on all the sequences iteratively. After selecting the binding site on S_i, it continues to select the binding site on S_{i+1}. It processes all the sequences in a round, and then it returns to S_1 and begins a new round. It stops when P remains the same in two consecutive rounds.

Post Processing

Post Processing addresses the issue of variable number of motif instances in a sequence. Greedy Refinement finds a binding site on each sequence S_i. However, a sequence may have zero, one, or more than one binding site(s). It is therefore important to allow the program to remove some spurious binding sites, and add more potential binding sites. The position indicator vector P can be easily converted to position indicator matrix A. The conversion is $A_i^j = \delta_{j,p_i}$, where δ_{j,p_i} returns 1 if the two arguments are equal, and 0 otherwise. Adding or removing a binding site depends on whether it contributes to the score of the whole of the binding sites. To check if a binding site A_i^j contributes to the score or not, EDAMD calculates the ratio $\xi(A_i^j)$ between the posterior probabilities Eq. 5.1 of the motif instances with it and without it as in Eq. 5.5.

$$\xi(A_i^j) = \frac{p(A'|S)}{p(A|S)} = \frac{p_0}{1-p_0}\frac{1}{\theta_0^{M(A_i^j)}}\prod_{j=1}^{w}\frac{N(A)_h^j + \alpha_h}{|A| + |\alpha|} \tag{5.5}$$

where A' is A added with the motif instance A_i^j. If $\xi(A_i^j) > 1$, the binding site A_i^j contributes to the overall scoring fitness, otherwise it affects the fitness negatively. Note Eq. 5.5 and Eq. 5.4 are actually the same except for the additional term $\frac{p_0}{1-p_0}$. This is expected, since the Bayes factor also compares the posterior conditional probabilities with or without a certain binding site.

EDAMD adopts a two-phase procedure to post-process the binding sites identified with Greedy Refinement. In the first phase, EDAMD calculates all the $\xi(A_i^{p_i})$, $i = 1, 2, \ldots, N$, and removes the binding sites of ξ values less than a threshold t_1. Although the assumption of every sequence contains at least a motif instance may not be true, Greedy Refinement tries to find a motif instance on each sequence. Therefore, the first phase is important to eliminate the spurious binding sites introduced by Greedy Refinement. In the second phase, for each sequence S_i, EDAMD calculates $\xi(A_i^j)$ for all the possible positions on S_i, and adds the binding sites of ξ values bigger than another threshold t_2. This thus locates more binding sites on the sequences. The order of the two phases is not reversible. Due to the possible spurious binding sites, some noise may be embedded in the latent probabilities θ. Therefore, the noise must be removed before searching for more motif instances.

It is difficult to choose appropriate thresholds t_1 and t_2. If t_1 is fixed, a large t_1 may cause true motif instances to be removed, while a small t_1 may not filter out the possible noise. EDAMD calculates t_1 automatically based on the current motif instances. A multinomial distribution parameterized by θ can be induced by $N(A)$ via the Maximum Likelihood approach, i.e., $\theta = \hat{N}(A)$. Suppose a set of artificial motif instances were generated according to the induced distribution, their contribution to the current motif instances can be calculated as Eq. 5.5, and thus the expected contribution $E(\xi)$ of the motif instances under the distribution θ is obtained. For a motif instance identified by Greedy Refinement, if its ξ is less than $E(\xi)$, it is rejected. The threshold $t_1 = E(\xi)$ is calculated as Eq. 5.6, where a_i is one of 4^w possible motif instances (w is the length of the motif, and each position of the motif has four possible nucleotides: A,C,G and T), and b_i^j is the nucleotide on position j in the motif instance a_i. Note Eq. 5.14 is used to calculate the sum of all the $\xi(a_i)p(a_i), i = 1, 2, \cdots, 4^w$ so that Eq. 5.6 can be solved analytically.

$$
\begin{aligned}
t_1 = E(\xi(a)) &= \sum_{i=1}^{4^w} \xi(a_i) p(a_i|\theta) \\
&= \frac{p_0}{1-p_0} \frac{1}{(|A|+|\alpha|)^w} \sum_{i=1}^{4^w} \prod_{j=1}^{w} \frac{N(A^j)_{b_i^j} + \alpha_{b_i^j}}{\theta_0^{b_i^j}} \frac{N(A^j)_{b_i^j}}{|A|} \\
&= \frac{p_0}{1-p_0} \frac{1}{(|A|+|\alpha|)^w} \prod_{j=1}^{w} \sum_{b \in B} \frac{N(A^j)_b + \alpha_b}{\theta_0^b} \frac{N(A^j)_b}{|A|}
\end{aligned}
\tag{5.6}
$$

$$
\underbrace{\sum_{i_1}^{4} \sum_{i_2}^{4} \cdots \sum_{i_w}^{4}}_{w} \prod_{j=1}^{w} x_{i_j}^j = \prod_{j=1}^{4} \sum_{i=1}^{4} x_i^j
\tag{5.7}
$$

From the preliminary experiments, it is found that the value of t_2 should not be fixed beforehand either, since the appropriate t_2 varies case by case. EDAMD uses a heuristic rule to adjust t_2 adaptively. After removing some motif instances in the first phase, it uses the minimum of all the ξ of the remaining instances as the initial t_2 in the second phase. The second phase is then carried out in rounds iteratively. In a round, a set of candidate motif instances $\{a_i'|\xi(a_i') > t_2\}$ are selected, and then EDAMD calculates the new t_2 as $min(\{\xi(a_i')|\xi(a_i') > 1\})$, which is used in the next round. It use a small initial t_2 at the beginning of the second phase to select a sufficient number of candidate motif instances, and afterwards it increases t_2 adaptively so as to select the motif instances of positive contributions only.

5.4.2 Estimation of Distribution Algorithm

The outer level of EDAMD is basically an iterative algorithm. It samples new PFMs from a Gaussian distribution, searches for the corresponding motif instances based on the PFMs (in the inner level), and updates the distribution model with the motif instances discovered. As pointed out in Section 5.3, the ultimate goal of EDAMA is finding the binding site of the motif, and the solution can also be represented as the motif consensus, i.e., $N(A)$. There may be interdependencies among the positions in the motif instances, which means the nucleotides on one position affect the nucleotides on another position [86][127][9]. In other words, the relative frequencies of the nucleotides on different positions have some correlation with each other. The fitness 5.3 does not incorporate the complicated inter-relation of positions, however, EDAMD uses a Gaussian distribution, i.e., $\mathscr{G}(x|\mu,\Sigma) \propto e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$, to model the motif instances and captures the possible pairwise correlations across the positions. Since the argument x in the Gaussian model $\mathscr{G}(x|\mu,\Sigma)$ is a column vector, EDAMD concatenates all the columns of PFM together, and refers to the resulted vector as Position Frequency Vector (PFV). PFV actually represents the same nucleotide frequencies of a set of motif instances as PFM.

After sampling a PFM Q, Greedy Refinement is used to find a set of the corresponding motif instances. However, Greedy Refinement starts with a set of motif instances $S(A)$, while Q may not correspond to a real set of motif instances, so it cannot be used in Greedy Refinement directly. To provide $S(A)$ for Greedy Refinement, EDAMD makes up a set of artificial motif instances $\tilde{A}$, each of which is exactly the same as Q. Consequently, $D \times Q$ is equal to the $N(\tilde{A})$ of the artificial motif instances, and Q is equal to $\hat{N}(\tilde{A})$. $D \times Q$ is then provided as the initial $N(A)$ for

Greedy Refinement to find the matching motif instances. Note the artificial motif instance $\tilde{A}$ may not exist in S, and they usually do not contain the valid nucleotides since the numbers in $N(\tilde{A})$ are fractional.

Afterwards, EDAMD uses the sets of motif instances found by Greedy Refinement to update the Gaussian distribution. Traditionally, the mean and the covariance are updated as $\mu = \frac{1}{T}\sum_{i=1}^{T} x_i$ and $\Sigma = \frac{1}{T}\sum_{i=1}^{T}(x_i-\mu)(x_i-\mu)'$ respectively, where T is the number of data samples. However, not all the motif instances are genuine motif instances, and they have different similarities to the common consensus. In addition, the fitness of the individuals (the sets of the motif instances) are not the same. Therefore, the motif instances should not be treated equally. Instead, EDAMD uses the weighted updating formula in Eq. 5.8, where $\{z_i | i = 1, 2, \ldots, T\}$ are the weights to measure the importance of the motif instances. The term $\sigma \times I(4w)$ in Σ is an identity matrix multiplied with a small positive constant. In the evolution, the motif instances may converge to the common motif consensus. To enhance the diversity of the sampled PFMs, EDAMD keeps the diagonal elements of Σ larger than 0.

$$\begin{aligned} \mu &= \frac{\sum_{i=1}^{T} z_i \times x_i}{\sum_{i=1}^{T} z_i} \\ \Sigma &= \frac{\sum_{i=1}^{T} z_i \times x_i x_i'}{\sum_{i=1}^{T} z_i} + \sigma \times I(4w) \end{aligned} \tag{5.8}$$

Calculating the weight of a motif instance is straightforward. Since the motif instances are usually weakly conserved, they are usually different from the common consensus and each other. Therefore, even for the motif instances in the same set A found from a common PFM Q, the instances may have different conditional probability on A. Intuitively, the weights associated with the instances should be the product of the posterior probability of the motif and the conditional probability of the moitf instances, i.e., $z_i = \psi(A) \times p(a_i|A), i = 1, 2, \ldots, |A|$, where a_i is a motif instance. The conditional probability of motif instance a_i w.r.t. A is calculated in

Eq. 5.9, where S_{a_i} is the sequence containing a_i, $S+S_{a_i}$ is the sequences S plus the sequence S_{a_i}, $A+a_i$ is the set of motif instances A plus a_i. The term $\Gamma(|A|+|\alpha|+1)$ is ignored since in the evolution, the number of motif instances $|A|$ is fixed to the number of sequences D.

$$\begin{aligned}
\rho(a_i) = p(a_i|A,S) &\propto \int p(a_i|A,S,\theta)p(A,S|\theta)p(\theta)d\theta \\
&= \int \frac{\theta_0^{M(S_{a_i})}}{\theta_0^{M(a_i)}} \prod_{j=1}^{w} \theta_j^{N(a_i^j)} \frac{\theta_0^{M(S)}}{\theta_0^{M(A)}} \prod_{j=1}^{w} \theta_j^{N(A^j)+\alpha-1} d\theta \\
&= \frac{\theta_0^{M(S+S_{a_i})}}{\theta_0^{M(A+a_i)}} \prod_{j=1}^{w} \frac{\prod_{b\in B}\Gamma(N(A)_b+N(a_i)_b+\alpha_b)}{\Gamma(|A|+|\alpha|+1)} \\
&\propto \frac{1}{\theta_0^{M(A+a_i)}} \prod_{j=1}^{w} \prod_{b\in B} N(A^j)_b+N(a_i^j)_b+\alpha_b-0.5^{N(A^j)_b+N(a_i^j)_b+\alpha_b-0.5} \qquad (5.9)
\end{aligned}$$

Alternatively, EDAMD can also use $\varphi(a_i)$ in Eq. 5.4 to calculate the weight. After all, given a set of potential binding sites, the order of their Bayesian factors and that of their posterior conditional probabilities are the same. If $\rho(a_1) > \rho(a_2)$, then $\varphi(a_1) > \varphi(a_2)$.

5.4.3 Experiments

EDAMD has been tested on eight real DNA datasets. A testing dataset consists of DNA sequences with motif instances already tagged. It is assumed the widths of the motifs are known beforehand. A motif instance is correctly recovered if the predicted binding site is within three bp away from the true binding site. The three bp tolerance is reasonable since in a real dataset, the widths of the tagged motif instances vary around the known width, and they are sometimes larger than the indicated width. It is contemplated that the true motif instance should lie somewhere around the two ends of the tagged instances. This criterion of successful prediction is also used in the GAs for motif discovery, i.e., GAME [119] and GALF [23]. To measure the performance of EDAMD and other algorithms, the experiments adopt the standard metrics of *Precision*, *Recall* and $F-score$ as defined in Eq. 5.10, where the operator $|\cdot|$ is the cardinality of the set. After EDAMD finds the candidate instances computationally, the results need to be verified in biological experiments. A higher *Precision* avoids wasting more effort on the false motif instances, while a higher *Recall* misses few true motif instances. $F-score$ mixes *Precision* and *Recall* since there is a tradeoff between *Precision* and *Recall*. Sometimes a high *Recall* means a large number of candidate instances, which may consist of many false positives. On the contrary, some true weakly conserved motif instances are deleted by mistake in order to achieve a high *Precision*.

$$\begin{aligned} Precision &= \frac{|correct\ motif|}{|motif\ found|} \\ Recall &= \frac{|correct\ motif|}{|true\ motif|} \\ F-score &= 2 \times \frac{Precision * Recall}{Precision + Recall} \end{aligned} \tag{5.10}$$

The eight real datasets are CREB, CRP, ERE, E2F, MEF2, MYOD, SRF and TBP [15][119][23]. The cyclic Amp receptor protein (CRP) binds in *Escherichia coli*. The estrogen receptor binds in the sequences of estrogen response elements (ERE). The E2F family also contains known binding sites. The datasets of CREB, MEF2, MYOD, SRF and TBP are published in ABS database of annotated regulatory binding sites. The benchmark datasets have a variety of the numbers of sequences, the lengths of sequences, the widths of motifs and the numbers of motif instances as shown in Table 5.1. EDAMD is tested on each dataset for 20 times with different random seeds. The population size T is 100, and the maximal generation G is 10. Moreover, the motif widths are the same as used in GAME and GALF. The best and the average results in the 20 runs are recorded.

The performance of EDAMD is compared to those of GAME and GALF in

dataset	#sequence	length	width	#instance
CREB	17	350	8	19
CRP	18	105	22	23
ERE	25	200	13	25
E2F	25	200	11	27
MEF2	17	199	7	17
MYOD	17	200	6	21
SRF	20	345	10	36
TBP	95	200	6	95

Table 5.1: The setting of the benchmark datasets: the number of sequences, the length of sequences, the width of motifs and the number of motif instances

Dataset	GAME			EDAMD			GALF		
	Precision	Recall	*F*-score	Precision	Recall	*F*-score	Precision	Recall	*F*-score
CREB	**0.78**	0.74	0.76	0.73	**0.84**	**0.78**	0.76	0.68	0.72
CRP	0.86	**0.78**	0.82	**0.94**	0.74	**0.83**	**0.94**	0.74	**0.83**
ERE	0.53	**0.80**	0.63	**0.76**	0.76	**0.76**	**0.76**	0.76	**0.76**
E2F	**0.80**	**0.89**	**0.84**	0.71	0.80	0.75	**0.80**	0.74	0.77
MEF2	0.89	**1.00**	0.94	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
MYOD	0.48	0.48	0.48	0.86	**0.90**	**0.88**	**0.88**	0.71	0.79
SRF	0.73	**0.92**	0.81	0.77	**0.92**	**0.84**	**0.95**	0.53	0.68
TBP	0.80	0.85	0.83	0.85	**0.94**	0.89	**0.93**	0.93	**0.93**

Table 5.2: Comparisons of EDAMD, GALF and GAME on the eight datasets: Best results (precisions, recalls and *F*-scores)

Tables 5.2 and 5.3. Table 5.2 shows the best results of the three algorithms in 20 runs. As regard to the F-score, EDAMD is the best in 6 problems. In the remaining two problems, it is worse than GAME on E2F, and worse than GALF on TBP. Table 5.3 shows the average results of the three algorithms in 20 runs. As regard to the F-score, EDAMD is the best on 7 problems, and it is worse than GAME on E2F. A remarkable observation is that the average results are the same as the best results in EDAMD. Actually, given a sufficiently large population, EDAMD always get the same results no matter what the random seed is. An explanation is that Greedy Refinement always finds the same best motif instances even from the set of different initial PFMs. Due to the lack of the results of the individual runs or the standard deviations in the original papers of GAME and GALF [119][23], there is no way to have the significant test of the performance comparison between the algorithms.

In addition, both GALF and GAME employ a population of 500 individuals.

Dataset	GAME			EDAMD			GALF		
	Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score
CREB	0.43	0.42	0.42	0.73	**0.84**	**0.78**	**0.76**	0.68	0.72
CRP	0.79	**0.78**	0.78	**0.94**	0.74	**0.83**	0.93	0.73	0.82
ERE	0.52	**0.78**	0.62	**0.76**	0.76	**0.76**	**0.76**	0.76	**0.76**
E2F	**0.79**	**0.87**	**0.83**	0.71	0.80	0.75	0.76	0.70	0.73
MEF2	0.52	0.55	0.53	**1.00**	**1.00**	**1.00**	0.97	0.97	0.97
MYOD	0.14	0.14	0.14	0.86	**0.90**	**0.88**	**0.88**	0.71	0.79
SRF	0.71	0.86	0.78	0.77	**0.92**	**0.84**	**0.88**	0.49	0.63
TBP	0.81	0.74	0.77	0.85	**0.94**	**0.89**	**0.88**	0.88	0.88

Table 5.3: Comparisons of EDAMD, GALF and GAME on the eight datasets: Average results (precisions, recalls and F-scores)

GALF runs up to 300 generations, and GAME runs up to 3000 generations. In addition, GALF and GAME use multi-start GA. In each run of GAME and GALF, GA is executed 20 times. Consequently, the total numbers of fitness evaluations are 3,000,000, and 30,000,000 in GALF and GAME, respectively. On the contrary, the number of fitness evaluations of EDAMD is only 1000, which is significantly small compared to GALF and GAME. On the other hand, the Greedy Refinement on an individual is computationally intensive, However, it is difficult to compare the running time fairly, because GALF was implemented in C, GAME was implemented in Java, and EDAMD was implemented in MATLAB.

5.5 Cluster Refinement Algorithm for Motif Discovery

As mentioned in Section 5.4.3, EDAMD always finds the same set of motif instances given a sufficiently large population, even though the GA evolution is affected by the initial random seed. In other words, a sufficient number of local optima might include the global optimum in motif discovery. Therefore, Cluster Refinement Algorithm for Motif Discovery (CRMD) is proposed as a deterministic algorithm, which is more efficient and effective than EDAMD.

5.5.1 Algorithm

Given a set of sequences S and the motif width w, solving for the optimal PIM A in terms of $p(A)$ in Eq. 5.1 directly is computationally intractable. Under the assumption of exactly one occurrence (of motif instance) per sequence (OOPS), the (w, d) motif discovery problem is already NP-hard [74]. If the OOPS assumption is relinquished, the search space becomes much bigger, and thus the problem is even more difficult. However, as shown in EDAMD in Section 5.4 and Motif Sampler [77] and MEME [6], given an initial PCM, it is possible to search for the PIM A whose $N(A)$ is the local optimum of the original PCM via an iterative procedure. Therefore, it is likely to obtain the global optimal A among the local optimal A's from a sufficient number of different initial PCMs.

Algorithm 5.2 is the main program of CRMD. Firstly, all the w long subsequences are extracted from the sequences ($sub(S)$ in Step 1). In each sequence, the subsequences starting positions range from the first possible binding site 1 until the last possible binding site $|S_i| - w + 1$. Secondly, the *Cluster* procedure partitions the set of all the candidate subsequences $sub(S)$ so as to group the similar subsequences in the same $cluster_i$, whose PIM is $A(cluster_i)$. Each $cluster_i$ is then used to construct a set of D artificial motif instances $\tilde{A}_{(i)}$ whose PFM $\hat{N}(\tilde{A}_{(i)})$ is equal to the PFM $\hat{N}(A(cluster_i))$ of $cluster_i$. Thirdly, the *Refine* procedure uses the PCM $N(\tilde{A}_{(i)})$ of the artificial motif instances $\tilde{A}_{(i)}$ to search for the local optimal PIM $A_{(i)}$. The best set of motif instances $A_{(i)}$ in terms of $p(A_{(i)})$ is returned as the result. If we know the motif is consistent with OOPS, a post *Adapt* procedure can be applied to further enhance the best set of motif instances.

Fig. 5.2 illustrates the execution path of CRMD with the example in Fig. 5.1. First, the set of all the subsequences s are extracted from the sequences S. The subsequences are then grouped into separate clusters using *Cluster*. In this example, there are altogether 17 clusters. The PFMs of the clusters are multiplied by $D = 10$

Algorithm 5.2: Main: the main program of CRMD

```
Input: The sequences $S$
Output: The best set of motif instances $BA$
$P \leftarrow -\infty$;
$clusters \leftarrow Cluster(sub(S))$;
foreach $cluster_i \in clusters$ do
    $N(\tilde{A}_{(i)}) \leftarrow D \times \hat{N}(A(cluster_i))$;
    $[A_{(i)}, p(A_{(i)})] \leftarrow Refine(N(\tilde{A}_{(i)}), S)$;
    if $p(A_{(i)}) > P$ then
        $P \leftarrow p(A_{(i)})$;
        $BA \leftarrow A_{(i)}$;
if $OOPS$ then
    $[BA, P] \leftarrow Adapt(BA)$;
```

as the PCMs of the artificial sets of motif instances. The artificial PCMs are subsequently used as the initial PCMs to find the local optimal sets of motif candidates in *Refine*. Finally, the best set of the motif instances is returned, in which the correct motif instances are highlighted in upper cases.

The following Sections *A*, *B* and *C* describe the *Cluster*, *Refine* and *Adapt* procedures in the main program of CRMD in Algorithm 5.2, respectively. Section *D* shows how CRMD is extended to handle multiple motif discovery problems.

Entropy-based Clustering

The *Cluster* procedure in Algorithm 5.2 chooses the initial PCMs for the *Refine* procedure. A good initial PCM is important for *Refine* as the resulted local optimum is more likely to be the global optimum than a bad initial PCM. A random PCM usually contains too much noise, and its PFM bears little similarity with the existing subsequences, and so searching from a random PCM rarely leads to true motif instances. Using an existing subsequence in S as the initial PFM is better than a random one since it is better conserved and it has at least one similar subsequence. However, for a typical motif discovery problem, there are thousands of subsequences, and so using all of them would be expensive. MEME selects some

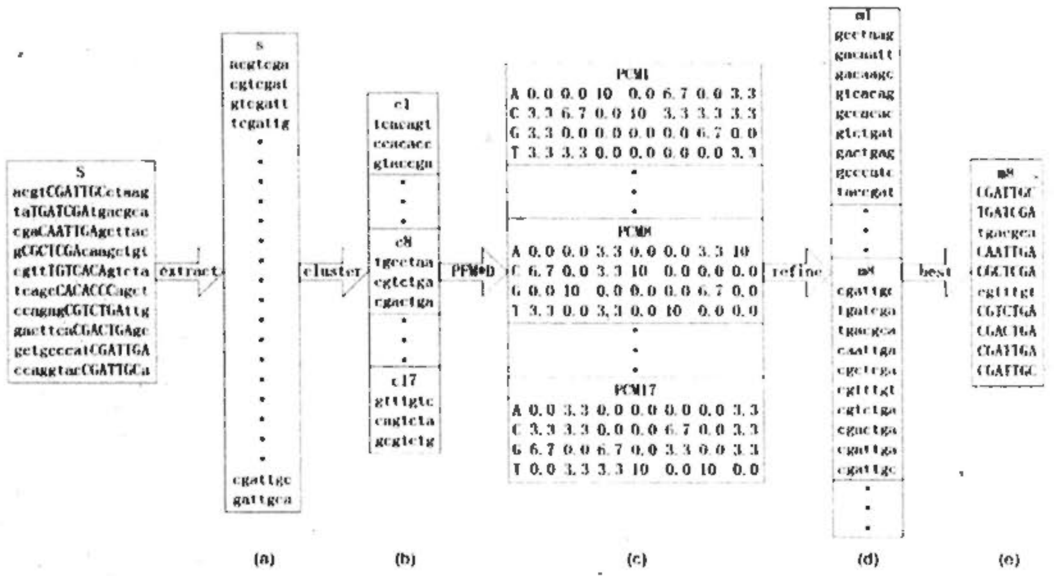


Figure 5.2: The execution path of CRMD with the example in Fig. 5.1. (a) all the subsequences of seven bps are extracted from the sequences S. (b) the subsequences are then grouped into separate clusters. (c) the initial PCMs are calculated as the PFMs of the clusters multiplied with D (10 in this example). (d) the initial PCMs are subsequently refined to find the local optimal set of motif candidates. (e) the best set is returned as the discovered motif instances where the correct instances are in upper cases

subsequences randomly and perturbs their PFMs somehow as the starting points in its EM algorithm. Nonetheless, there is still no guarantee that the randomly selected subsequences definitely occur in the motif instances.

CRMD creates and selects the initial PCMs by clustering all the subsequences into modest-sized groups. There are four advantages of clustering. First, clustering all the subsequences guarantees that every subsequence has a large chance to occur in a certain cluster and thus is likely to be considered in the subsequent process. Second, grouping the similar subsequences together exempts CRMD from the costly computation of processing every subsequence later, and in the extreme case the huge number (4^w) of all the potential consensus. Third, clustering similar subsequences into the same group has already accomplished part of the job of maximizing the posterior probability in Eq. 5.1. Forth, clustering has an explicit control of the number of the subsequences in a cluster, as it discards small insignificant

Algorithm 5.3: Cluster: partition the subsequences into separate clusters

```
Input: The subsequences $s$
Output: the clusters
$clusters \leftarrow \emptyset$;
if $|s| \leq D$ then
    if $|s| \geq \frac{D}{4}$ then
        $clusters \leftarrow s$
else
    $[pos, base] \leftarrow Pos(s)$;
    $clusters \leftarrow clusters \cup Cluster(s^{pos}_{base})$;
    $clusters \leftarrow clusters \cup Cluster(s^{pos}_{b/base})$;
```

clusters and partitions large clusters to remove the noise. Compared to other clustering algorithms for motif discovery [16][93], the third and the fourth advantages are very important in finding the motif efficiently and effectively.

Algorithm 5.3 is the pseudocode of the procedure *Cluster* in Algorithm 5.2. Provided with a set of subsequences s, *Cluster* checks the size of s, i.e., $|s|$, at first. If $|s|$ is smaller than $\frac{D}{4}$, s is discarded. If $|s|$ is larger than $\frac{D}{4}$ and smaller than D, it is returned as a cluster. If $|s|$ is larger than D, *Cluster* continues to partition s. Step $Pos(s)$ selects the optimal position pos and the optimal nucleotide $base$ to partition s into two sets of the subsequences. The subsequences in the first set s^{pos}_{base} have the nucleotide $b = base$ on position pos, while the subsequences in the other set $s^{pos}_{b/base}$ have nucleotides other than $base$ on position pos. Both sets are then recursively clustered in $Clustering(s^{pos}_{base})$ and $Clustering(s^{pos}_{b/base})$, respectively. In this way, the set of subsequences s are separated into smaller and smaller clusters by applying *Cluster* recursively.

Cluster keeps a set of subsequences s intact and returns it as a cluster if and only if its size is in the range $[\frac{D}{4}, D]$. If the cluster size is too large, the subsequences in the cluster may have too much diversity which introduces unnecessary noise into the resulted PCM. If the cluster size is too small, the subsequences may constitute no significant motif and thus the cluster is discarded, because motif discovery is looking for the binding sites of a common transcription factor bound to sufficient sequences. Even if a motif instance happens to be included in a discarded cluster, it is still possible to recover it from another cluster consisting of other motif instances. In the extreme case, a set of exact $\frac{D}{4}$ subsequences is split and returned as a cluster in each recursion of Algorithm 5.3. Therefore, all the clusters consist of exact $\frac{D}{4}$ subsequences, and the maximal number of clusters is $\frac{4L}{D}$, where L is the total number of all the subsequences. The actual number of clusters is much smaller than the maximal number because the number of the subsequences in a cluster is usually larger than $\frac{D}{4}$. On the other hand, if the number of the subsequences in a cluster is smaller than $\frac{D}{4}$, the cluster is discarded directly. It is empirically observed that for a typical dataset of thousands of subsequences, the number of clusters is up to only several hundreds.

To choose the optimal position and nucleotide base in step $Pos(s)$ to partition the current set of subsequences s, CRMD adopts the clustering criterion in Eq. 5.11. For each potential partitioning position pos and nucleotide type $base$, $Pos(s)$ calculates the relative entropy of the subset resulted from partitioning the subsequences s on position pos according to nucleotide type $base$, i.e., $En(s_{base}^{pos})$, and then scales the entropy with the size of the subset, i.e., $|s_{base}^{pos}|$. The position and nucleotide type giving the largest scaled entropy are chosen for partitioning. $|s_{base}^{pos}|$ is considered in finding $Pos(s)$ so that a large cluster is preferred. $En(s_{base}^{pos})$ is the sum of the relative entropies of the subsequences s_{base}^{pos} on all the positions. The relative information entropy is used because CRMD aims to find the set of subsequences which are similar to each other and yet different from the background sequences.

$$\begin{aligned} Pos(s) &= \underset{pos \in \{1,2,\ldots,w\}, base \in B}{\text{argmax}} En(s_{base}^{pos})|s_{base}^{pos}| \\ En(s_{base}^{pos}) &= \sum_{j=1}^{w} \sum_{b \in B} \frac{N(s_{base}^{pos})_b^j}{|s_{base}^{pos}|} log(\frac{N(s_{base}^{pos})_b^j}{|s_{base}^{pos}|} \frac{1}{\theta_{0b}}) \end{aligned} \tag{5.11}$$

The choice of the clustering criterion in Eq. 5.11 is deliberate, as it enables

Cluster to find the clusters of approximately large posterior probability as defined in Eq. 5.1. Actually, if we simplify Eq. 5.1 using the Burnside formula[3] to approximate the gamma function, we may get the log of $p(A)$ as follows,

$$\begin{aligned}\tilde{L}(A) &= log(p(A|S, \theta_0, p_a, p_b, \alpha)) \approx K + f(|A|, p_a, p_b, \alpha) \\ &+ \sum_{i=1}^{w} \sum_{1 \le b \le B} (N(A)^i_b + \alpha_b - 0.5) log \frac{N(A)^i_b + \alpha_b - 0.5}{|A| + |\alpha| - 0.5} \frac{1}{\theta_{0b}}\end{aligned}$$

where K is an invariant constant w.r.t all the variables, and $f(|A|, p_a, p_b, \alpha)$ is a function of $|A|$ only. If A is substituted with s^{pos}_{base}, the last term is approximate to $En(s^{pos}_{base})|s^{pos}_{base}|$ in Eq. 5.11, where $\frac{N(s^{pos}_{base})^i_b}{|s^{pos}_{base}|} \approx \frac{N(A)^i_b + \alpha_b - 0.5}{|A| + |\alpha| - 0.5}$ and $|s^{pos}_{base}| \approx |A| + |\alpha| - 0.5$. Therefore, the set of the subsequences s^{pos}_{base} of large $En(s^{pos}_{base})|s^{pos}_{base}|$ is likely to have a large $p(s^{pos}_{base})$.

Greedy Refinement

In Algorithm 5.2 and Fig. 5.2, the *Refine* procedure finds a local optimal set of motif instances from an initial cluster. Rather than using the actual subsequences in the cluster, *Refine* uses the PCM $N(\tilde{A}) = D \times \hat{N}(A(cluster))$ of the initial subsequences $\tilde{A}$ as the seed for further refinement. $\tilde{A}$ is simply a symbol consisting of no actual subsequences since only its PCM is needed in the refinement, and its PFM is equal to $\hat{N}(A(cluster))$. Greedy Refinement subsequently finds a new set of subsequences A, whose $N(A)$ is similar to and yet better conserved than $N(\tilde{A})$.

There are two advantages of the *Refine* procedure. In Section 5.5.1 (Selecting Motif Instances), it uses a fast greedy local search method to find the local optimal motif instances. A greedy heuristic makes the search deterministic, while the Gibbs sampling in Motif Sampler is a random process, and thus *Refine* converges much faster. In Section 5.5.1 (Changing Instance Number), it uses auto-adjusted thresholds to change the number of motif instances adaptively. The search is flexible as it

[3]Burnside formula $\Gamma(x+1) = x! \approx (x+0.5)^{x+0.5} e^{-x-0.5} \sqrt{2\pi}$

allows a variable number of instances. At the same time, the number is changed by at most one instance each iteration, and thus *Refine* still converges very fast.

Algorithm 5.4 shows the overall pseudocode of the *Refine* procedure. Iteratively, it replaces the old motif candidate instances $A^{(t-1)}$, which is $\tilde{A}$ initially, with the new candidate motif instances $A^{(t)}$. The new candidate motif instances are selected among all the subsequences to maximize the posterior probability $p(A^{(t)})$ based on the old candidate motif instances. There are at most D iterations. At the beginning, NUM, the number of motif instances, is set to the number of sequences, i.e., D. In each iteration, after finding NUM candidate motif instances, *Refine* tries to remove the least likely candidate motif instance s_1 to increase $p(A^{(t)})$. If it is removed successfully, NUM is decreased. Otherwise *Refine* tries to add the next most likely subsequence s_2 to increase $p(A^{(t)})$. If it is added successfully, NUM is increased. *Refine* stops iterating when A remains the same in two consecutive iterations, and finally it returns the last A.

The two main steps in *Refine*, the selection of motif instances and the adaptive changing the number of motif instances are given below:

Selecting Motif Instances

Iteratively, *Refine* finds a new set of more conserved motif instances $A^{(t)}$ which are similar to the old set of candidate motif instances $A^{(t-1)}$. The similarity of a subsequence A_i^j to the existing motif instances A^* is measured by how much the posterior probability $p(A^*)$ increases if A_i^j is added in A^*. Instead of calculating the two probabilities with or without A_i^j and then comparing them, CRMD calculates the Bayes ratio between them directly, which is derived using the Bayesian inference. $Ratio(N(A^*), A_i^j)$ in Eq. 5.12 is the strength of a position A_i^j being a binding site based on the current $N(A^*)$. The derivation is similar to Eq. 5.1, where the equation $\Gamma(n+1) = n\Gamma(n)$ is used to cancel out the Gamma functions in both numerator and denominator. $N(A^*)^k_{b^k}$ is the number of nucleotide $b^k = S_i^{j+k-1}$ in $S(A^*)^k$, i.e., the same nucleotide type on position k in $S(A^*)$ as the one b^k in the subsequence $S(A_i^j)$.

Algorithm 5.4: Refine: identify the motif instances based on a cluster

```
   Input: the initial $N(\tilde{A})$ and $S$
   Output: The Local Optimal $A$ and $p(A)$
   $NUM \leftarrow D$;
   $A^{(0)} \leftarrow \emptyset$;
   $N(A^{(0)}) \leftarrow N(\tilde{A})$;
   for $i \leftarrow 1$ to $D$ do
       $ratios \leftarrow Ratio(N(A^{(i-1)}), S)$;
1      $A^{(i)} \leftarrow \text{argmax}_{A_i^j}(ratios_i^j, NUM)$;
2      if not $OOPS$ then
           $s_1 \leftarrow \text{argmin}_{s_j \in A^{(i)}} ratios(s_j)$;
           $T_1 \leftarrow Expect(N(A^{(i)} - \{s_1\}), \theta_0)$;
           if $Ratio(N(A^{(i)} - s_1), s_1) < T_1$ then
               $NUM \leftarrow NUM - 1$;
               $A^{(i)} \leftarrow A^{(i)} - \{s_1\}$
           else
               $s_2 \leftarrow \text{argmax}_{s_j \notin A^{(i)}} ratios(s_j)$;
               $T_2 \leftarrow Expect(N(A^{(i)}), \hat{N}(A^{(i)}))$;
               if $Ratio(N(A^{(i)}), s_2) > T_2$ then
                   $NUM \leftarrow NUM + 1$;
                   $A^{(i)} \leftarrow A^{(i)} + s_2$
       if $A^{(i)} = A^{(i-1)}$ then
           $A \leftarrow A^{(i)}$;
           $p(A) \leftarrow p(A|S, \theta_0, p_a, p_b, \alpha)$;
           return;
```

$$
\begin{aligned}
Ratio(N(A^*), A_i^j) &= \frac{p(A_i^j = 1|A^*, S)}{p(A_i^j = 0|A^*, S)} \\
&= \frac{\int p(A_i^j = 1|\theta, A^*, S, p_0) p(\theta|A^*, S) p(p_0|p_a, p_b)\, d\theta}{\int p(A_i^j = 0|\theta, A^*, S, p_0) p(\theta|A^*, S) p(p_0|p_a, p_b)\, d\theta} \\
&= \frac{1}{\theta_0^{M(S(A_i^j))}} \frac{|A^*| + p_a}{L - |A^*| + p_b - 1} \prod_{k=1}^{w} \frac{N(A^*)_{b^k}^k + \alpha_{b^k}}{|A^*| + |\alpha|}
\end{aligned}
\tag{5.12}
$$

After calculating the ratios of all the A_i^j based on the old instances $A^{(i-1)}$, *Refine* selects NUM subsequences of the maximal ratios directly and replaces $A^{(i-1)}$ with

$A^{(i)}$ in Step 1. *Refine* is greedy because it always select the subsequences of the best matches, and so it may get stuck in local optima. This is exactly why CRMD adopts a multi-start approach with *Cluster* to locate the global optimum out of many local optima. However, being greedy, *Refine* converges fast as $A^{(i)}$ usually stabilizes in less than $\frac{D}{2}$ iterations. On the contrary, Motif Sampler uses Gibbs sampling to iteratively select subsequences with probabilities in proportion to their Bayes factors. As a Markov Chain Monte Carlo method, Gibbs sampling may take an undetermined time before generating samples following the target distribution.

Changing Instance Number

An important issue in discovering motif instances is choosing an appropriate number of predicted motif instances. Predicting too many motif instances may lead to many false instances, while predicting too few motif instances may miss many true instances.

To address the issue of the unknown number of motif instances, *Refine* changes the number of motif instances *NUM* adaptively to increase the posterior probability as defined in Eq. 5.1. More specifically, Algorithm 5.4 adds or removes a marginal motif instance by comparing its ratio to the thresholds T_1 and T_2, which are calculated adaptively based on the existing motif instances. The number of the predicted motif instances is changed by at most one in an iteration, and so it is fast and easy for the motif instances to converge in the Greedy Refinement.

In detail, after sampling *NUM* candidate instances, *Refine* selects the one with the smallest ratio, i.e., $s_1 \leftarrow \text{argmin}_{s_j \in A^{(i)}} ratios(s_j)$, and checks if removing it would increase the posterior probability of the rest of the candidate instances $A^{(i)} - \{s_1\}$. *Refine* calculates the ratio $Ratio(N(A^{(i)} - \{s_1\}), s_1)$. A small ratio means s_1 affects $A^{(i)} - \{s_1\}$ negatively and thus should be removed. Otherwise, *Refine* checks if the subsequence of the largest ratio in the remaining subsequences, i.e., $s_2 \leftarrow \text{argmax}_{s_j \notin A^{(i)}} ratios(s_j)$, would benefit the probability of the current set of motif instances $A^{(i)}$. Similarly, *Refine* calculates the ratio $Ratio(N(A^{(i)}), s_2)$ and adds s_2 if the ratio indicates that it will increase the probability. *NUM* is decreased or increased depending on whether a subsequence is removed or added. The order of removing and adding motif instances is irreversible. Due to the possible spurious binding sites, some noise may be included in the current set of motif instances.. Therefore, the noise must be removed first before searching for more motif instances.

In Algorithm 5.4, *Refine* compares the ratios with the thresholds T_1 and T_2 in the two "if" conditions, and it removes or adds the subsequence if the condition is satisfied. It is important to choose appropriate values for the two thresholds T_1 and T_2 since they control the value of NUM directly. Intuitively, both thresholds should be 1 since the ratio is 1 when the posterior probabilities of a set of motif instances with or without the subsequence are equal. However, since the motif is usually weakly conserved, it is possible that a true binding site is mutated somehow and looks quite different from the others, and so removing it (or not adding it) may actually increase the posterior probability of the set of the motif instances. *Refine* also has to be prudent to add new motif candidates since a false subsequence may readily increase the posterior probability of a very weakly conserved motif. Therefore, 1 may be inappropriate for the thresholds.

Refine adjusts T_1 and T_2 automatically to account for two concerns. First, because each iteration in Algorithm 5.4 may have a different set of motif instances $A^{(i)}$, the thresholds are always calculated in accordance with the current $A^{(i)}$. Second, since there is no prior knowledge of the subsequence to be included or excluded, the thresholds should take into account all the possible subsequences of a certain distribution. Therefore, the threshold that *Refine* uses is the expected ratio of a random subsequence generated from a certain distribution Θ w.r.t. the current set of the motif instances A, i.e. $E(Ratio(N(A), s)|\Theta)$. A naive yet computationally intensive method to calculate the expectation is to collect the ratios of all the possible subsequences over the current motif instances A and taking their average in proportion to their probabilities of the specified distribution. Fortunately, Eq. 5.13 shows an analytical formula to calculate the expected ratio efficiently. In Eq. 5.13,

s_i is one of the 4^w possible subsequences generated from the distribution parameterized by Θ with the probability $p(s_i|\Theta)$. b_i^j is the nucleotide base on position j in the subsequence s_i.

$$
\begin{aligned}
E(Ratio(N(A),s)|\Theta) &= \sum_{i=1}^{4^w} Ratio(N(A),s_i)p(s_i|\Theta) \\
&= \frac{|A|+p_a}{L-|A|+p_b-1}\sum_{i=1}^{4^w}\prod_{j=1}^{w}\frac{N(A)^j_{b_i^j}+\alpha_{b_i^j}}{(|A|+|\alpha|)\theta_0^{b_i^j}}\Theta^j_{b_i^j} \\
&= \frac{|A|+p_a}{L-|A|+p_b-1}\prod_{j=1}^{w}\sum_{b\in B}\frac{N(A)^j_b+\alpha_b}{(|A|+|\alpha|)\theta_0^b}\Theta^j_b. \qquad (5.13)
\end{aligned}
$$

The computation of the part $\sum_{i=1}^{4^w}\prod_{j=1}^{w}$ in Eq. 5.13 is greatly simplified by using Eq. 5.14, which reduces $4^w \times w$ variable references (of only $4w$ distinct $x^j_{k^j}$) on the left to $4w$ variable references (without repetition) on the right. The reason is that $\sum_{i=1}^{4^w}\prod_{j=1}^{w}$ in Eq. 5.13 actually involves only the complete enumeration over the cartesian product of the sets $\{\frac{N(A)^j_b+\alpha_b}{(|A|+|\alpha|)\theta_0^b}\Theta^j_b|b\in B\}$, where $j=1\cdots w$. Therefore, $\sum_{i=1}^{4^w}\prod_{j=1}^{w}$ can be rewritten as the left of Eq. 5.14, where $x^j_{k^j}=\frac{N(A)^j_{k^j}+\alpha_{k^j}}{(|A|+|\alpha|)\theta_0^{k^j}}\Theta^j_{k^j}$, and simplified as the right of Eq. 5.14.

$$
\overbrace{\sum_{k^1\in B}\sum_{k^2\in B}\cdots\sum_{k^w\in B}}^{w}\prod_{j=1}^{w}x^j_{k^j} = \prod_{j=1}^{w}\sum_{k^j\in B}x^j_{k^j} \qquad (5.14)
$$

Consequently, *Refine* uses Eq. 5.13 to calculate T_1 and T_2 under different distributions. For removing a motif instance, *Refine* uses the possible subsequences generated from the background distribution ($\Theta=\theta_0$) to calculate the threshold $T_1 = E(Ratio(N(A),s)|\theta_0)$. If the ratio of the instance in question is smaller than T_1, it is no better than a background subsequence, and so it is definitely discarded. For adding a subsequence, $T_2 = E(Ratio(N(A),s)|\hat{N}(A))$ is used, which is the average ratio of the subsequences generated from the current PFM. Derived from the Maximum Likelihood principle, $\hat{N}(A)$ is actually the latent probabilities generating the current motif instances. Therefore, a new subsequence should be definitely added if its ratio is bigger than the average ratio of the motif instances, namely T_2.

Post Adaptation

Greedy Refinement allows a variable number of motif instances, and it adjusts the number of motif instances automatically in the searching. If the problem has One Occurrence of motif instance Per Sequence (OOPS), the performance of CRMD can be further enhanced since the search space is greatly reduced.

There are two modifications to Algorithm 5.4 of *Refine* to take the advantage of the OOPS assumption. First, in Step 1, the binding sites are selected on the sequences separately. For each sequence, *Refine* compares the ratios of its subsequences, and then selects the one and the only one of the maximal ratio. Second, the number of motif instances NUM is constantly D, and so the part of changing NUM in the $if - then$ loop is not executed (Step 2). Since NUM is not changed anymore, it becomes easier and faster for $A^{(t)}$ to stabilize.

However, in the case that OOPS is only an approximation, we still need to fine tune the number of motif instances. Considering that the motif is close to OOPS, it is better not to change the number of motif instances NUM inside *Refine* since the iterative searching may amplify the noise introduced by any additional candidate instance due to the changing NUM.

In Algorithm 5.2 of the main program, the procedure *Adapt* further processes the best set of motif instances BA. The original BA is consistent with OOPS, but the true motif may be slightly different from OOPS. *Adapt* first removes the existing instances in BA whose ratios are smaller than T_1 even though it might remove all the candidate instances on a certain sequence. *Adapt* then adds new instances not in BA if their ratios are larger than T_2 even though it might find more than one candidate instance on a certain sequence. Here T_1 and T_2 are calculated in the same way as in the procedure *Refine*. It is unnecessary to apply *Adapt* to every $A_{(i)}$ returned by *Refine* in Algorithm 5.2, because *Adapt* usually does not change the ranking of the posterior probabilities of the sets of motif instances if the problem is inherently OOPS.

Multiple Motifs Discovery

CRMD can be extended to solve the multiple motif discovery problem. It is possible for a set of real DNA sequences to contain multiple motifs. The multiple motifs may have various kinds of consensuses, numbers of instances and degrees of conservations. Due to the diversity of the multiple motifs, the signal-to-noise ratios are even lower than that in the single motif discovery problem. Therefore, it is usually more difficult to find multiple motifs than a single motif.

Some traditional motif discovery algorithms run their single motif searching procedures multiple times to locate different motifs. After finding a motif, the corresponding subsequences and their neighbors are masked off the sequences so that the overlapping subsequences will not be identified as new motif instances later on. The shortcoming of this masking scheme is that the discovery of subsequent motifs depends on the previously predicted motifs. If some spurious instances are included in a motif, the neighboring subsequences which might be true motif instances are masked off. Even if a true binding site is predicted but included in a wrong motif, masking it off too early may corrupt the consensus of the corresponding motif and thus affect the discovery of the motif later.

To avoid the aforementioned drawbacks, CRMD finds multiple motifs without masking simultaneously. As *Refine* samples the sets of the motif instances based on the cluster, a straightforward approach for CRMD is to select a few candidate motifs among all the sets of the motif instances returned by *Refine*, i.e., $\{A_{(i)}\}$. The selection is performed according to two criteria, namely the posterior probabilities of the possible motifs and the similarities between the selected motifs. Since the motifs are weakly conserved and the clusters from *Cluster* may have similar consensuses, it is possible that the resulted motifs after *Refine* are similar and predict many common binding sites. Therefore, among a group of similar motifs, only the motif of the highest posterior probability is selected.

In the current implementation, CRMD specifies the number of motifs M beforehand. When a new motif is returned by *Refine*, it is firstly checked if it is similar to any of the motifs already selected. If so, the new motif replaces the similar motif if the former has a higher probability. If there is no similar motifs already selected, the new motif replaces the selected motif of the lowest probability if the former has a higher probability. The approach ensures that the M output motifs are different from each other, and at the same time they are of as high probability as possible.

To measure the similarity between the *PCMs* of two motifs, CRMD adopts the homogeneity test using the χ^2 distance in [59]. Basically, it shifts and aligns the two motifs. If their PCMs on all the overlapping positions are statistically generated from the same distribution, the two motifs are deemed similar.

5.5.2 Experiments

CRMD has been tested on both synthetic and real DNA datasets. A testing dataset consists of DNA sequences with motif instances already tagged, and hence it can be used for the algorithm performance evaluation. For some datasets, the widths of the motifs are assumed known beforehand and are tested directly with CRMD. For the other datasets with unknown widths, CRMD either uses a common fixed width or tries a range of different widths and selectes the width giving the best result.

Some researchers use two levels of performance indices to evaluate the algorithm [117][53]. On the nucleotide level, it is calculated that how many nucleotides that the predicted instances and the true instances overlap for. On the site level, a predicted instance is correct if it overlaps with the true instance for at least one nucleotide. To combine the performance indices on both levels, CRMD adopts the criterion that a motif instance A_i^j is correctly recovered if either of its ends is within three bp away from the corresponding end of the true motif instance [119][22]. More formally,

$$A_i^j = 1\ is \begin{cases} correct & \text{if } |j - j_s| < 3 \text{ or } |j + w - j_e| < 3 \\ incorrect & \text{otherwise} \end{cases} \tag{5.15}$$

where j_s and j_e are the indices of the starting and ending positions of the closest true motif instance. The three bp tolerance is reasonable since the widths of the tagged motif instances vary around the known width in a real dataset. It is conjectured that the true motif instance should lie somewhere between the two ends of the tagged instances [117]. This criterion of successful prediction is strict and practical since it does tell a biologist where to look for the true binding sites. In contrast to comparing binding sites, comparing the PFM or PWM of the discovered motif and the true motif may be insufficient, because a small difference in PFM or PWM may lead to very different binding sites.

To measure the performance of CRMD and other algorithms, the metrics of *Precision*, *Recall* and $F-score$ [119][22] are defined as follows, where the operator $|\cdot|$ is the cardinality of the set.

$$\begin{aligned} Precision &= \frac{|correct\ motif|}{|motif\ found|} \\ Recall &= \frac{|correct\ motif|}{|true\ motif|} \\ F-score &= 2 \times \frac{Precision * Recall}{Precision + Recall}. \end{aligned}$$

After an algorithm finds the candidate instances computationally, the results need to be verified in biological experiments. The algorithm hopes for a high *Precision* to avoid wasting too much effort on the false motif instances. In the meanwhile, it should miss as few true motif instances as possible, so a high *Recall* is preferred. However, there is often a tradeoff between *Precision* and *Recall* in real problems. Sometimes a high *Recall* means a large number of candidate instances, which may consist of many false positives. On the contrary, a high *Precision* can be achieved by retaining only the highly conserved motif instances at the risk of deleting some true weakly conserved motif instances by mistake. Therefore, *F score* is introduced to mix *Precision* and *Recall*.

CRMD are compared to Motif Sampler [77], MEME [6], GAME [119] and GALF-P [22]. Since Motif Sampler and MEME are sensitive to the initial settings, they are executed in the manner of multi-start with different starting points. GAME and GALF-P are GA-based, and their results may be inconsistent and affected by the random seeds, so only the average results of GAME and GALF-P in 20 runs are reported. In each run, the total numbers of the sets of motif instances searched by GALF-P and GAME are 3,000,000 and 30,000,000, respectively. With such a large number of sampling, the searching of GALF-P and GAME are relatively exhaustive, and their results are expected to be close-to-optimal.

The following subsections *A*, *B* and *C* give the details of the results for the synthetic single motif, real single motif and real multiple motif discovery problems tested in our experimental evaluations. In the real single motif discovery experiment, the following datasets are tested: the eight selected datasets in GAME [119] and GALF-P [22], the ABS database [15], the SCPD database [128], the *Escherichia coli* dataset [53] and the Tompa dataset [117].

Synthetic Datasets

A total of 800 synthetic datasets with length 300 bp for each sequence are generated with the following eight combinations of scenarios: (1) motif width: short (8 bp) or long (16 bp); (2) number of sequences: small (20) or large (60); (3) motif conservation: high or low. For each combination, 100 datasets are generated randomly and embedded with the instances of a random motif. In the high conservation scenario, on every position of the motif instances, the dominant nucleotide is generated with 0.91 probability (while all other three nucleotides with 0.03 each). In the low conservation scenario, only 60% of the positions in the motif instances are as highly conserved as in the previous high conservation scenario, while the rest 40%

of the positions are lowly conserved, where the dominant nucleotide is generated only with probability 0.55 (while all other three nucleotides with 0.15 each) in every instance. To simulate the noisy situation in real data, in each synthetic dataset, the probability of containing no motif instances is 0.1 for each sequence. In the rest of the sequences which contain motif instances, the probability for a sequence to have more than one instance is 0.1. The number of additional instance(s) in such a sequence follows the geometric distribution with $p = 0.5$, i.e., $p(k) = (1-p)^{k-1}p$, and so there are expectedly $\frac{1}{p} = 2$ additional motif instances embedded in the sequence.

Table 5.4 shows the results of the five algorithms. For each scenario, the results are averaged over the 100 datasets. CRMD has the highest average $F-scores$ on six out of eight scenarios. In the remaining two scenarios CRMD has the second highest average $F-scores$. CRMD also has the highest average $F-score$ over all the 800 problems of eight different scenarios, which proves that CRMD is relatively robust in a variety of problems. For the easy datasets in the last two scenarios (with long motif width and high conservation), all the algorithms have very good results ($F-scores$ around 0.98), and so there is no big room for the improvement for CRMD. For the other more difficult problems, the results of the algorithms vary in a wider range and the advantage of CRMD is more apparent. It is also interesting to notice that MEME has the highest average *Precision* in most of the scenarios, while GALF-P has the highest average *Recall* in most of the scenarios. However, CRMD has both good *Precisions* and *Recalls* on most of the datasets, and thus it yields the highest average $F-scores$ due to the good balance between *Precision* and *Recall*.

Real Datasets

To investigate the performance of CRMD on real datasets, and how it is compared to other algorithms, CRMD and other algorithms are also tested on a wide range of real datasets. Section 5.5.2 (Eight Selected Datasets) describes a detailed analysis of the results on the eight datasets tested by GAME [119] and GALF-P [22]. Section

Scenario			GALF-P			GAME			MEME			Sampler			CRMD		
Width	Num	Con	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Short	Small	Low	0.38	0.56	0.44	0.29	0.32	0.30	0.49	0.34	0.39	0.44	0.37	0.40	0.46	0.45	**0.46**
Short	Large	Low	0.52	0.59	**0.55**	0.42	0.32	0.36	0.63	0.33	0.42	0.55	0.41	0.46	0.53	0.53	0.53
Long	Small	Low	0.87	0.91	0.89	0.78	0.87	0.82	0.91	0.86	0.88	0.87	0.89	0.88	0.91	0.88	**0.91**
Long	Large	Low	0.91	0.90	0.91	0.92	0.90	0.90	0.96	0.85	0.90	0.89	0.92	0.91	0.92	0.92	**0.92**
Short	Small	High	0.73	0.90	0.80	0.71	0.80	0.75	0.87	0.84	0.85	0.85	0.85	0.85	0.86	0.83	**0.86**
Short	Large	High	0.81	0.86	0.83	0.83	0.83	0.83	0.91	0.76	0.83	0.87	0.83	**0.85**	0.84	0.83	0.84
Long	Small	High	0.97	0.99	0.98	0.94	0.99	0.97	0.98	0.99	0.98	0.96	1.00	0.98	0.99	0.99	**0.99**
Long	Large	High	0.97	0.97	0.97	0.98	0.99	0.98	0.99	0.98	0.98	0.96	1.00	0.98	0.99	0.99	**0.99**
Average			0.77	0.84	0.80	0.73	0.75	0.74	0.84	0.74	0.78	0.80	0.78	0.79	0.81	0.80	**0.81**

Table 5.4: Average results for the synthetic datasets experiment: Width is for the motif width, Num is for the number of sequences, Con is for conservation degree, P is for *Precision*, R is for *Recall* and F is for *F*-score. Sampler refers to Motif Sampler

5.5.2 (ABS and SCPD databases) reports the results on the ABS database [15] and the SCPD database [128]. Section 5.5.2 (E. coli and Tompa datasets) reports the results on the *Escherichia coli* dataset [53] and the Tompa dataset [117].

Eight Selected Datasets

Following GAME [119] and GALF-P [22], the experiment has tested eight real datasets, i.e., CREB, CRP, ERE, E2F, MEF2, MYOD, SRF and TBP, and compared the performance with the other four algorithms. These eight datasets consist of the sequences from many different species. The CRP dataset contains TFBSs bound by the cyclic amp receptor protein in *Escherichia Coli* [112][69][78]. The ERE dataset contains the estrogen receptor elements that ER binds, from the sequences of various species [60]. The E2F dataset contains TFBSs of the E2F family from different mammalian species [58][11][38]. The datasets of CREB, MEF2, MYOD, SRF and TBP are chosen by GAME from the ABS database of annotated regulatory binding sites [15]. As shown in Table 5.5, the real datasets have a variety of the numbers of sequences, the lengths of sequences, the widths of motifs and the numbers of motif instances. The same motif widths are adopted as used in [119] and [22]. For a fair comparison, all the algorithms are run with as few prior knowledge as possible, and most of their running options are set to their default values.

Table 5.6 compares the results of the four algorithms (GAME, MEME, Motif

dataset	#sequence	length	width	#instance
CREB	17	350	8	19
CRP	18	105	22	23
ERE	25	200	13	25
E2F	25	200	11	27
MEF2	17	199	7	17
MYOD	17	200	6	21
SRF	20	345	10	36
TBP	95	200	6	95

Table 5.5: the real datasets: the numbers and the lengths of sequences, the width and the numbers of motif instances

Problem	GAME			MEME			Sampler			CRMD		
	P	R	F	P	R	F	P	R	F	P	R	F
CREB	0.43	0.42	0.42	0.71	0.63	**0.67**	0.71	0.63	**0.67**	0.67	0.63	0.65
CRP	0.79	0.78	0.78	0.89	0.67	0.76	0.94	0.70	0.80	1.00	0.74	**0.85**
ERE	0.52	0.78	0.62	1.00	0.68	**0.81**	0.75	0.72	0.73	0.71	0.80	0.75
E2F	0.79	0.87	0.83	0.82	0.85	0.84	0.88	0.85	0.87	0.83	0.93	**0.88**
MEF2	0.52	0.55	0.53	0.93	0.82	0.88	0.72	0.76	0.74	0.85	1.00	**0.92**
MYOD	0.14	0.14	0.14	0.29	0.19	0.23	0.46	0.29	0.35	0.86	0.90	**0.88**
SRF	0.71	0.86	0.78	0.74	0.89	0.81	0.76	0.86	0.81	0.79	0.86	**0.83**
TBP	0.81	0.74	0.77	0.83	0.69	0.76	0.74	0.67	0.70	0.83	0.89	**0.86**
Average	0.59	0.64	0.61	0.78	0.68	0.72	0.74	0.69	0.71	0.82	0.84	**0.83**

Table 5.6: The results for the real datasets assuming no OOPS: P is for *Precision*, R is for *Recall* and F is for *F*-score. Sampler refers to Motif Sampler

Sampler and CRMD) on the eight real datasets. Due to the adaptive thresholds adopted in the Greedy Refinement, CRMD is able to choose an appropriate number of motif instances, and thus finds a good balance between *Precision* and *Recall*, which consequently leads to the highest $F-scores$ on six problems. Compared to Motif Sampler, CRMD is better on seven out of eight datasets in terms of the $F-scores$. Compared to MEME, CRMD wins on all but two datasets. For the MYOD problem in particular, because its motif width is short and the number of the sequences is small, the signal-to-noise ratio is low. Other algorithms are unable to identify the true motif in MYOD probably because of the marginal win of the fitness of the true motif. On average, CRMD has the highest *Precision*, *Recall* and $F-score$, which proves that the performance of CRMD is quite stable on the eight problems.

The algorithms have also been tested with the prior knowledge of OOPS. A close

Problem	GALF-P			MEME			CRMD		
	P	R	F	P	R	F	P	R	F
CREB	0.70	0.84	0.76	0.71	0.63	0.67	0.73	0.84	**0.78**
CRP	0.99	0.73	**0.84**	0.67	0.52	0.59	0.94	0.74	0.83
ERE	0.82	0.76	**0.79**	0.76	0.76	0.76	0.67	0.80	0.73
E2F	0.77	0.85	**0.81**	0.76	0.70	0.73	0.68	0.85	0.75
MEF2	0.91	0.98	0.95	0.94	0.94	0.94	1.00	1.00	**1.00**
MYOD	0.57	1.00	0.72	0.06	0.05	0.05	0.86	0.90	**0.88**
SRF	0.75	0.89	0.82	0.95	0.53	0.68	0.77	0.92	**0.84**
TBP	0.87	0.87	0.87	0.94	0.94	**0.94**	0.85	0.94	0.89
Average	0.80	0.87	0.82	0.72	0.63	0.67	0.81	0.87	**0.84**

Table 5.7: The results for the real datasets assuming OOPS: P is for *Precision*, R is for *Recall* and F is for *F*-score

investigation of the eight problems reveals that they are more or less consistent with the OOPS assumption. As shown in Table 5.5, except for the problem SRF, the numbers of motif instances are close to the numbers of sequences. CRMD activates the *Adapt* procedure and searches for OOPS solution only in the *Refine* procedure. The experiment has also tested GALF-P and MEME on the eight problems, which are capable of searching for OOPS consistent motifs. GALF-P searches for OOPS solutions only in its GA procedure, and then it shrinks or expands the solutions with a heuristic post processing procedure. MEME has an OOPS running option which enables MEME to search for exactly one instance in each sequence. Table 5.7 shows the results of CRMD, GALF-P and MEME. Mostly, CRMD obtains better results than those obtained without OOPS in Table 5.6. Compared to GALF-P and MEME, CRMD has the highest $F-scores$ on four problems, and the second highest $F-scores$ on the other four problems. CRMD also has the highest average $F-score$. In particular, on the problem of MYOD, CRMD has a remarkable advantage over the other two algorithms. Even though GALF-P is the best on three problems, its results have some variance since it is GA-based and sensitive to the initial population.

ABS and SCPD databases

Besides the eight selected datasets, the experiment has tested CRMD, MEME and Motif Sampler on the ABS [15] and the SCPD [128] databases as well. The ABS database has 650 experimental binding sites from 69 transcription factors in

database	MEME			Sampler			CRMD		
	P	R	F	P	R	F	P	R	F
ABS	0.10	0.21	0.13	0.15	0.10	0.11	0.18	0.15	**0.16**
SCPD	0.10	0.05	0.05	0.29	0.19	0.23	0.31	0.26	**0.28**

Table 5.8: The average results of MEME, Motif Sampler and CRMD on the ABS and the SCPD databases. P is for *Precision*, R is for *Recall*, F is for $F-score$

human, mouse, rat and chicken genome sequences. The sequences and the binding sites are downloaded from the website of ABS database, and re-grouped the sequences of the same transcription factors together, and thus a total of 69 datasets is obtained, each of which consists of the sequences bound by a common transcription factor. SCPD is a promoter database of the yeast *Saccharomyces cerevisiae*. It contains 580 experimentally mapped transcription factor binding sites. Because the website provides no FASTA files, the sequences and the binding sites have to be collected and organized manually, and the transcription factors of less than four binding sites are deleted, and thus a total of 28 datasets is obtained.

No prior knowledge of the exact widths for the motifs is assumed in the ABS and the SCPD datasets. CRMD and Motif Sampler use the commonly adopted fixed widths of 10 bps and 13 bps in ABS and SCPD, respectively, which are the medians of the widths of the true motif instances in ABS and SCPD, respectively. For MEME, the widths vary between $[6, 26]$ and $[7, 26]$ in ABS and SCPD, respectively, which are actually the ranges of the widths of the true motif instances in ABS and SCPD, respectively. Because the actual motif widths are not used, the tolerance in Eq. 5.15 is relaxed to six bps. Table 5.8 shows the average results of CRMD, MEME and Motif Sampler on the ABS and the SCPD database. For the ABS dataset, CRMD has higher *Precision* while lower *Recall* rate than MEME, and still it has the highest $F-score$. For the SCPD dataset, CRMD has the highest *Precision*, *Recall* and $F-score$.

E. coli and Tompa datasets

The experiment has also tested CRMD on the *Escherichia coli (E. coli)* [53] and the Tompa [117] datasets, which are collected and setup as the benchmark problems for testing motif discovery algorithms. The *E. coli* dataset is of prokaryotic data. It consists of 62 motifs of a variety characteristics, such as the motif width, the number of sites per sequence and the sequence length, etc. The Tompa dataset consists of 56 eukaryotic datasets, covering fly, human, mouse and yeast. The motifs are very weakly conserved in the Tompa dataset, which is by far the most difficult dataset tested with CRMD.

The *E. coli* and the Tompa datasets are already tested with other algorithms, including MEME and Motif Sampler, in [53] and [117], respectively. They use different performance evaluation indices other than the *Precision*, *Recall* and $F-score$. Their performance indices can be categorized on two levels. On the nucleotide level, the performance indices (with the prefix n) are calculated w.r.t. to the number of the nucleotides that the true and the predicted instances overlap. On the site level, the performance indices (with the prefix s) are calculated w.r.t. the number of the motif instances that the predicted instances overlaps with the true instances for at least one nucleotide. Suppose on the nucleotide level, nTP (true positive) is the number of true motif nucleotides correctly predicted, nTN (true negative) is the number of true background nucleotides not predicted, nFP (false positive) is the number of falsely predicted motif nucleotides and nFN (false negative) is the number of true motif nucleotides not predicted. Eq 5.16 defines the nucleotide level performance indices. The site level ones are similarly defined by replacing all the "n" with "s" in Eq. 5.16. The original papers [117][53] have more details on the definitions of their performance indices.

algorithm	nucleotide level				site level				best
	nPC	nSn	nSP	nF	sPC	sSn	sSp	sF	nPC
MEME	0.158	0.259	0.199	0.225	0.295	0.461	0.436	0.448	0.116
Sampler	0.153	0.179	0.237	0.204	0.302	0.331	0.476	0.390	0.069
CRMD	**0.286**	0.321	0.412	**0.346**	**0.459**	0.531	0.625	**0.558**	**0.221**

Table 5.9: The average performance of MEME, Motif Sampler and CRMD on the E. coli datasets. Each algorithm outputs five motifs, and the one of the best *nPC* among the five outputs is recorded as the result. The last column reports the *nPC* of the top-scored motif in terms of the score function used in the individual algorithm

$$\begin{aligned} nSn &= nTP/(nTP+nFN) \\ nSp &= nTP/(nTP+nFP) \\ nPC &= nTP/(nTP+nFP+nFN) \\ nF &= (2\times nSn\times nSp)/(nSn+nSp) \end{aligned} \tag{5.16}$$

For the *E.coli* dataset, each algorithm is required to output five motifs, and the one with the best *nPC*, is recorded as the result. The widths of the motifs vary from problem to problem, and thus CRMD use 15 as the fixed common width for all the problems, as used by other algorithms in the original paper [53]. Table 5.9 shows the average results of CRMD, MEME and Motif Sampler on all the datasets, where the results of MEME and Motif Sampler are quoted from the paper [53]. On all the nucleotide level and site level performance indices, CRMD has around 10 percentage better results. More surprisingly as indicated in the last column (best *nPC*), if CRMD outputs a single motif, its performance is already better than the best of the five outputs of MEME and Motif Sampler.

For the Tompa dataset, the algorithms are permitted to fine tune the parameters and report the best result. Since the exact widths of the motifs in the datasets are unknown beforehand, CRMD is executed with a series of widths from 10 to 15. For each width, 10 motifs (without the similarity test) are output, and so a total of 60 motifs is obtained. Among the 60 motifs, CRMD calculates the similarity between each pair of motifs in terms of the χ^2 distance [59], and it selects the motif

algorithm	nSn	nPPV	nPC	sSn	sPPV	sASP
MEME	0.067	0.107	0.043	0.111	0.139	**0.125**
Sampler	0.060	0.107	0.040	0.098	0.101	0.100
CRMD	0.091	0.088	**0.047**	0.141	0.108	**0.125**

Table 5.10: The average results of MEME, Motif Sampler and CRMD on the Tompa dataset. $xPPV$ is $xTP/(xTP+xFP)$ for both nucleotide and the site levels, and $sASP$ is $(sSn+sPPV)/2$

which has the largest number of similar motifs as the result. Table 5.10 shows the average results of MEME, Motif Sampler and CRMD on the 56 Tompa datasets, where the results of MEME and Motif Sampler are quoted from the paper [117]. Generally, the sensitivity of CRMD is slightly better, and the specificity of CRMD is slightly worse, which result in marginally better performance coefficient. As the motifs in most of the Tompa datasets are very weakly conserved, the algorithms usually predict no correct results on both nucleotide and site levels on those datasets. Therefore, the average performance indices of the three algorithms are pretty low, which shows that the *de novo* motif discovery on real datasets of complex organisms is still difficult for the current algorithms with often more than questionable results.

Multiple Motif Dataset

The liver-specific dataset [63] contains multiple motifs. Biological experiments verified that the liver-specific gene expression is controlled by the combined action of a small set of TFs, primarily HNF-1, HNF-3, HNF-4 and C/EBP. The dataset contains 19 sequences and annotates 60 binding sites belonging to ten motifs. However, three motifs have only one instance each, and three other motifs have only two instances each. These six motifs are supposed to be very difficult to find due to the extreme low signal-to-noise ratio. The rest four motifs have 19, 13, 13 and 11 instances, respectively, among which three motifs have less instances than the sequences. The widths of the motifs vary from 6 to 31, and even the motif instances of the same motif may have different lengths.

GAME, MEME, Motif Sampler and CRMD are tested on the liver-specific

#output	GAME			MEME			Sampler			CRMD		
	P	R	F	P	R	F	P	R	F	P	R	F
5	0.27	0.33	0.3	0.31	0.18	0.23	0.34	0.17	0.23	0.46	0.5	**0.48**
10	0.32	0.6	0.42	0.30	0.23	0.36	0.40	0.18	0.25	0.44	0.78	**0.56**

Table 5.11: The results of MEME, Motif Sampler and CRMD on the liver-specific dataset of multiple motifs. Each program is executed twice with five and ten outputs, respectively. P is for *Precision*, R is for *Recall*, F is for $F-score$

dataset to evaluate their performance of multiple motif discovery, while GALF-P is incapable of handling multiple motif discovery problem. The average width 15 is used in all the experiments. There are two sets of experiments. One is with five outputs to account for the four motifs with most TFBSs, and the other is with ten outputs to account for the motifs with fewer TFBSs as well. The prediction tolerance is relaxed to six bps because the motifs are very weakly conserved.

Table 5.11 shows the results of the four algorithms in the two sets of experiments of five outputs and ten outputs, respectively. The *Precisions* in all the experiments are lower than 0.50, and CRMD is the only one whose *Precisions* are higher than or equal to 0.44. The low *Precisions* indicate that there are many false positives in the results. This is expected since the algorithms have to output more predicted motifs than the true ones due to the low signal-to-noise ratio. MEME and Motif Sampler have very low *Recalls* in both 5-output and 10-output experiments. When the number of the outputs is increased from 5 to 10, the *Recalls* of GAME and CRMD are increased significantly as more binding sites are correctly predicted. CRMD has the highest *Precisions* and *Recalls* in both experiments, and so its $F-scores$ are also the highest. The advantage of the $F-scores$ of CRMD over the $F-scores$ of GAME, which are the second highest, is more than 10 percentages.

5.6 Discussion

Estimation of Distribution Algorithm for Motif Discovery (EDAMD) uses the fitness function derived by Bayesian analysis to measure the posterior conditional probability of a set of motif instances. Therefore, it is able to handle variable number of motif instances in the DNA sequences. It adopts a Gaussian distribution to model the distribution of the sets of motif instances. The Gaussian distribution is capable of capturing the bivariate correlation among the positions of motif instances. When a new Position Frequency Matrix (PFM) is sampled from the Gaussian distribution, the local optimal set of the motif instances nearby is identified from the PFM via the Greedy Refinement operation. At the end of a generation, the Gaussian distribution is updated with the sets of the motif instances considering the fitness and the probabilities of the motif instances. Since Greedy Refinement finds a single motif instance on a sequence, a Post Processing procedure is used to find more motif instances after the evolution. The experiments have verified that EDAMD outperforms GAME and GALF on most of the eight selected testing real problems, and its results stay constant with different initial populations.

A deterministic algorithm is also proposed for motif discovery, i.e., Cluster Refinement Algorithm for Motif Discovery (CRMD). CRMD uses the Entropy-based Clustering to find good initial motif candidates first, and then it uses Greedy Refinement to find the local optima of the initial candidates. CRMD searches for motifs by maximizing the posterior probabilities of the motif instances. The posterior probability allows a variable number of motif instances and it requires little prior knowledge of motifs. Entropy-based Clustering partitions all the subsequences of DNA sequences into clusters of maximal relatively information entropies, and thus clustering alone has already maximized part of the posterior probability. The number of the clusters is much smaller than the number of all the subsequences, and so the computation cost is significantly reduced. Greedy Refinement finds the local optimal binding sites given the initial clusters. It selects the motif instances of maximal probabilities deterministically without taking extra time in sampling subsequences probabilistically. It also automatically removes or adds motif instances according to the thresholds which change adaptively following the distribution of the current motif instances. If the prior knowledge of OOPS is available, CRMD can further enhance its prediction performance by searching for OOPS consistent solutions only and adjusting the number of motif instances later on. For multiple motif problem, CRMD measures the similarities among the candidate motifs using the χ^2 homogeneity test, and thus it is able to keep only distinct motifs of high probabilities.

Compared with other state-of-the-art algorithms, CRMD has been tested extensively on both synthetic and comprehensive real datasets of single and multiple motifs. As observed from the empirical results, CRMD is very competitive, and often the best among the testing algorithms. The synthetic data are generated with a variety of properties and difficulties. CRMD has achieved a good balance between *Precision* and *Recall*, and thus obtained the highest $F-scores$ on most of the synthetic problems. The real datasets tested are comprehensive. On the eight real datasets selected by GAME [119] and GALF-P [22], CRMD still has the highest $F-scores$ on most of the problems, and its average *Precision*, *Recall* and $F-score$ are the highest as well. With the OOPS assumption, the performance of CRMD is further enhanced, and its results are better than or comparable to those of the other two OOPS algorithms.. On other four databases, i.e., the ABS database [15], the SCPD database [128], the *Escherichia coli* dataset [53] and the Tompa dataset [117], CRMD has also achieved the best performance in terms of either of the default metrics or of the nucleotide and site level metrics used in [117] and [53]. For the liver-specific dataset of multiple motifs, CRMD identifies significantly more binding sites than the other multiple motif discovery approaches. Compared to EDAMD, CRMD has a similar performance on the eight selected real datasets. Due to the relaxed assumption, CRMD is more general to solve more difficult problems.

5.6.1 Time Complexity

Table 5.12 shows the time complexities of the motif discovery algorithms. Suppose the sequence length is L and the number of the sequences is D. For EDAMD,

	EDAMD	CRMD	GALF(P)	GAME	MEME	Sampler
complexity	$O(PGD^2)$	$O(LD)$	$O(PGD)$	$O(PG)$	$O(CI)$	$O(CI)$

Table 5.12: The complexities of the motif discovery algorithms. L is the sequence length. D is the number of the sequences. P is the population size. G is the number of generations. C is the number of initial consensuses. I is the number of iterations

the population size is P, the maximal generation is G, the maximal iteration in the Greedy Refinement is D, each iteration samples D instances, and so the complexity of EDAMD is $O(PGD^2)$. For CRMD, the number of the clusters is $\frac{4L}{D}$, the maximum iteration in the Greedy Refinement is D, each iteration samples D instances, and so the time complexity of CRMD is $O(LD)$. For GALF(P), the population size is P, the maximal generation is G, the instances uncovered in the local filtering is D, and so the complexity of GALF(P) is $O(PGD)$. For GAME, the population size is P, the maximal generation is G, and so the complexity of GAME is $O(PG)$. For MEME, the number of the initial consensuses is C, the number of the iterations in expectation maximization is I, and so the complexity of MEME is $O(CI)$. For Motif Sampler, the number of the initial consensuses is C, the number of the iterations in sampling is I, and so the complexity of Motif Sampler is $O(CI)$.

The time complexities of the algorithms should not be compared directly. Different GAs may require different population sizes and generation numbers. GALF(P) and GAME evaluate 3,000,000 and 30,000,000 individuals, respectively, while EDAMD evaluates only 1,000 individuals. For MEME and Gibbs Sampler, the number of iterations I cannot determined beforehand and thus is quite indefinite, while the maximal iteration in EDAMD and CRMD is fixed to D and the actual number is usually smaller than D.

To evaluate the effectiveness of *Cluster* in CRMD, Table 5.13 compares the total numbers of all the subsequences, the theoretical maximal numbers of the clusters and the actual numbers of the clusters. The theoretical maximal number of clusters is $\frac{4L}{D}$ which is already significantly smaller than the total number of all the subsequences L. In the eight testing real datasets, the actual numbers of clusters are even

Problem	#subsequence	$\max(\frac{4L}{D})$	#cluster	reduction
CREB	3544	834	292	92%
CRP	1512	336	118	92%
ERE	4700	752	276	94%
E2F	4750	760	268	94%
MEF2	3293	775	267	92%
MYOD	3315	780	280	92%
SRF	4127	825	292	93%
TBP	18525	780	296	98%

Table 5.13: The numbers of the subsequences, the theoretical maximal numbers of the clusters $\frac{4L}{D}$, the numbers of the clusters and the reductions of the seeds for *Refine*

much smaller than the theoretical maximal numbers of the clusters. The reductions over the total numbers of the subsequences are over 90%. This shows that *Cluster* not only provides good initial *PCM*s for *Refine*, but also saves a lot of computation time.

It is also interesting to inspect the performance of CRMD without either of *Cluster* and *Refine*. As regard to *Cluster*, it is empirically observed that with the same number of initial candidates, the performance of using random initialization instead for *Refine* is worse than that of using *Cluster*. The results of random initialization also vary with the different pseudo-random number seeds. Especially for the problem MYOD, which has short width and small number of sequences, the precision and the recall of the random initialization are zero in the worst case. As regard to *Refine* (with *Cluster* still in CRMD), if the thresholds T_1 and T_2 in Algorithm 5.4 are fixed at 1, the performance of CRMD deteriorates a lot. On the problems MEF2 and MYOD in particular, CRMD cannot find any correct motif instance at all.

Dataset	EDAMD			CMRD (OOPS)			CRMD		
	Precision	Recall	*F*-score	Precision	Recall	*F*-score	Precision	Recall	*F*-score
CREB	0.73	0.84	**0.78**	0.73	0.84	**0.78**	0.67	0.63	0.65
CRP	0.94	0.74	0.83	0.94	0.74	0.83	1.00	0.74	**0.85**
ERE	0.76	0.76	**0.76**	0.67	0.80	0.73	0.71	0.80	0.75
E2F	0.71	0.80	0.75	0.68	0.85	0.75	0.83	0.93	**0.88**
MEF2	1.00	1.00	**1.00**	1.00	1.00	**1.00**	0.85	1.00	0.92
MYOD	0.86	0.90	**0.88**	0.86	0.90	**0.88**	0.86	0.90	**0.88**
SRF	0.77	0.92	**0.84**	0.77	0.92	**0.84**	0.79	0.86	0.83
TBP	0.85	0.94	**0.89**	0.85	0.94	**0.89**	0.83	0.89	0.86

Table 5.14: The comparison between EDAMD, CRMD with OOPS and CRMD without OOPS on the eight selected datasets

5.6.2 Comparison between Estimation of Distribution Algorithm and Cluster Refinement Algorithm

There are a few differences between EDAMD and CRMD. First, EDAMD is a GA-based algorithm, while CRMD is deterministic. Clearly, it is difficult to determine the suitable population size and max generations for EDAMD for a problem of big motif width and large number of sequences. CRMD clusters all the subsequences, and thus the number of potential motifs is automatically determined. Second, EDAMD assumes One Occurrence Per Sequence (OOPS) and thus it is able to solve OOPS problems only. CRMD can disable the OOPS assumption, and has the freedom to change the number of motif instances in the local search. Third, EDAMD adopts the Gaussian distribution to model the population. There is only one peak in the Gaussian distribution, and so EDAMD searches for only one motif. CRMD adopts a similarity measure to differentiate motifs, and thus CRMD is able to discover multiple motifs in a single run. .

As shown in Table 5.14, the performances of EDAMD and CRMD on the eight selected real datasets are quite comparable. Surprisingly, the performance of EDAMD and CRMD with OOPS are very close except on the dataset ERE. CRMD without OOPS is not as good as EDAMD and CRMD with OOPS on five datasets, but it wins on the other two datasets. This is expected since most of the datasets are more or less OOPS, and the prior knowledge of OOPS really helps the local search.

Chapter 6

Conclusion

To tradeoff the complexity and the learning of the distribution model in Estimation of Distribution Algorithm (EDA), this thesis proposes a new framework of Estimation of Dependency and Distribution Algorithm (EDDA) to choose an appropriate learning model automatically. Basically, EDDA partitions an individual representation into separate parts such that they are independent with respect to (w.r.t.) the fitness function. The independent parts of the individual representation are evolved separately with a different distribution model each. The combination of the optima of the independent parts forms the optimum of the complete individual representation. For the problems which cannot be partitioned into completely independent parts, EDDA also maintains the information of the interdependencies between the separate parts and evolves the interdependencies as well. The complexity of a model is controlled adaptively by the amount of the dependency information maintained in the model.

There are a few major advantages of EDDA over the standard Evolutionary Computation (EC),

First, partitioning the individual representation and evolving the independent parts separately reduces the size of the search space significantly. In EC, the search space of all the dimensions is the cartesian product of the domains of the individual dimensions, and the size of the complete search space is the product of the sizes of the dimensions. In EDDA, the size of the search space is the sum of the sizes of the search spaces of the independent parts, and the size of the search space in the standard EC is exponentially larger than the size of the search space of an independent part.

Second, important dependency information between the separate parts are maintained while the trivial ones are ignored. According to schema theory, a population of individuals evolves numerous schemata, and a schema is actually a pattern of the genes involved. In EDA, a complicated model, which contains a lot of model parameters to estimate, maintains a large amount of information of the dependency between the genes. However, much of such interdependency information is unnecessary, and the parameters are difficult to estimate accurately and may actually mislead the evolution.

Third, it is easy to control the diversity and the convergence of the sub-populations of the separate parts of the individual representation. Diversity and convergence affect how much the solution space is searched directly. In high dimensional space, the population in the standard EC covers only a small and sparse area in the search space, and it is difficult to manipulate the size and the individual density of the covered search area. In EDDA, because an independent part of the individual representation consists of only a few dimensions, it is much easier to control the diversity and convergence in such a relatively small search space.

Fourth, compared to other EDAs, EDDA learns the distribution model with all the individuals in the population and with their fitness. EDDA thus estimates a better approximation of a more complete fitness landscape. On the contrary, other EDAs usually discard the individuals of bad fitness, and use only the good individuals for model estimation. Nevertheless, the resulted distribution may be misleading in the area of bad individuals, and may distort the complete fitness landscape.

This thesis includes four implementation of EDDA for different applications.

EDDA is first employed in Genetic Algorithm (GA) to optimize objective functions by converting the problem solution into some independent parts and evolving the independent parts separately. **Chapter 3** has described a Genetic Algorithm with Independent Component Analysis (GA/ICA) to solve unconstrained continuous function optimization problems. It first uses ICA to identify the independent components of the solution space w.r.t. the fitness, and then it divides the population into sub-populations and evolves the sub-populations on the independent components separately. Finally, it combines the optima on the independent components into the global optimum for the original problem. As the high-dimensional problem is divided into many 1-dimensional sub-problems, the solution space is significantly reduced, and so the problem becomes easier for GA to solve. The experiment results show that GA/ICA requires much less function evaluations to produce optimal or close-to-optimal solutions which are better than or comparable to those produced by OGA/Q on the benchmark problems.

EDDA can also be used in Genetic Programming (GP) to speed up the GP evolution by evolving the GP instructions and their interactions simultaneously. **Chapter 4** has described a novel Genetic Programming algorithm, Instruction Matrix based Genetic Programming (IMGP). IMGP maintains an Instruction Matrix (IM) to store the information of the instructions and their best subtrees. It extracts program trees from IM, updates IM with the fitness of the extracted program trees, performs crossover and mutation on the extracted program trees, and shuffles IM to propagate good instructions. It is contemplated that IMGP actually evolves some schemata directly. The experimental results have verified the effectiveness and the efficiency of IMGP on the benchmark problems. IMGP is not only superior to CGP in terms of the qualities of the solutions and the number of program evaluations, but it also outperforms other related GP algorithms on the testing problems. To enhance its performance for classification problems, IMGP uses gradient descent to find the optimal parameters in a program tree, and incorporates the complexity of the program tree in the fitness. In most of the testing problems, IMGP is able to find classifiers of higher classification accuracies than 4 other GP classifiers. The results of IMGP are also comparable to or better than those of Decision Tree, Neural Networks and Support Vector Machine.

Chapter 5 has proposed an Estimation of Distribution Algorithm for Motif Discovery (EDAMD) as an application of EDDA to solve a real bioinformatics problem. EDAMD uses the fitness function derived by Bayesian analysis to measure the posterior conditional probability of a set of motif instances, and the number of motif instances is allowed to vary. EDAMD adopts a Gaussian distribution to model the distribution of the sets of motif instances. The Gaussian distribution is capable of capturing the bivariate correlation among the positions of motif instances. When a new Position Frequency Matrix (PFM) is sampled randomly from the Gaussian distribution, the local optimal set of the motif instances nearby is identified from the PFM via the Greedy Refinement operation. At the end of a generation, the Gaussian distribution is updated with the sets of the motif instances, their fitness and the conditional probabilities of the motif instances. Since Greedy Refinement finds only a single motif instance on a sequence, a Post Processing procedure is used to find more motif instances after the evolution. The experiments have verified that EDAMD outperforms GAME and GALF on most of the eight selected testing real problems, and its results stay constant with different initial populations.

A deterministic algorithm has also been proposed for motif discovery, Cluster Refinement Algorithm for Motif Discovery (CRMD). CRMD uses the Entropy-based Clustering to find good initial motif candidates first, and then it uses Greedy Refinement to find the local optima from the initial candidates. Entropy-based Clustering partitions all the subsequences of DNA sequences into clusters of maximal relatively information entropies, and thus clustering alone has already maximized part of the posterior probability. The number of the clusters is much smaller than the number of all the subsequences, and so the computation cost is significantly reduced. Greedy Refinement finds the local optimal binding sites given the initial clusters. It selects the motif instances of maximal probabilities deterministically without taking extra time in sampling subsequences probabilistically. It also automatically removes or adds motif instances according to the thresholds which change adaptively following the distribution of the current motif instances. If the prior knowledge of One Occurrence Per Sequence (OOPS) is available, CRMD can further enhance its prediction performance by searching for OOPS consistent solutions only and adjusting the number of motif instances later on. For multiple motif problems, CRMD measures the similarities among the candidate motifs using the χ^2 homogeneity test, and thus it is able to keep only distinct motifs of high probabilities. The empirical results show that the clustering provides good initial consensus seeds, and the refinement procedure leads to the local optimal consensus efficiently. The qualities of the discovered solutions are compared favorably with the solutions produced by other state-of-the-art algorithms. The performances of EDAMD and CRMD on some problems are similar, however, CRMD is more general and is able to solve more difficult problems than EDAMD.

Bibliography

[1] S.I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[2] C ândida Ferreira. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

[3] F.R. Bach and M.I. Jordan. Kernel independent component analysis. *The Journal of Machine Learning Research*, 3:1–48, 2003.

[4] F.R. Bach, G.R.G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the twenty-first international conference on Machine learning*. ACM New York, NY, USA, 2004.

[5] F.R. Bach, R. Thibaux, and M.I. Jordan. Computing regularization paths for learning multiple kernels. In *Advances in Neural Information Processing Systems 17: Proceedings Of The 2004 Conference*, page 73. MIT Press, 2005.

[6] T. L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pages 28–36, 1994.

[7] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning,. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.

[8] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming - An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, January 1998.

[9] Yoseph Barash, Gal Elidan, Nir Friedman, and Tommy Kaplan. Modeling dependencies in protein-DNA binding sites. In *RECOMB*, pages 28–37, 2003.

[10] Forrest H. Bennett III, John R. Koza, James Shipman, and Oscar Stiffelman. Building a parallel computer system for $18,000 that performs a half petaflop per day. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honâvar, Mark Jakiela, and Robert E. Smith, editors, *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, pages 1484–1490, San Francisco, CA, 1999. Morgan Kaufmann.

[11] B.P. Berman, Y. Nibu, B.D. Pfeiffer, P. Tomancak, S.E. Celniker, M. Levine, G.M. Rubin, and M.B. Eisen. Exploiting transcription factor binding site clustering to identify cis-regulatory modules involved in pattern formation in the Drosophila genome. *Proceedings of the National Academy of Sciences*, 99(2):757–762, 2002.

[12] P. Bieganski, J. Riedl, J. V. Carlis, and E.M. Retzel. Generalized suffix trees for biological sequence data: applications and implementations. In *Proc. of the 27th Hawaii Int. Conf. on Systems Sci.*, pages 35–44, 1994.

[13] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[14] E. Keogh C. Blake and C. J. Merz. UCI repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

[15] Enrique Blanco, Domènec Farré, M. Mar Albà, Xavier Messeguer, and Roderic Guigó. ABS: a database of annotated regulatory binding sites from orthologous promoters. *Nucleic Acids Research*, 34(Database-Issue):63–67, 2006.

[16] K. Blekas, D.I. Fotiadis, and A. Likas. Greedy mixture learning for multiple motif discovery in biological sequences. *Bioinformatics*, 19(5):607 617, 2003.

[17] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

[18] Peter A. N. Bosman and Edwin D. de Jong. Grammar transformations in an EDA for genetic programming. In *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, June 2004.

[19] S.P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[20] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In *RECOMB*, pages 69–76, 2001.

[21] E. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness, 2004.

[22] Tak-Ming Chan, Kwong-Sak Leung, and Kin-Hong Lee. Tfbs identification based on genetic algorithm with combined representations and adaptive post-processing. *Journal of Bioinformatics*, 24:341, 2007.

[23] Tak-Ming Chan, Kwong-Sak Leung, and Kin-Hong Lee. TFBS identification by position- and consensus-led genetic algorithm with local filtering. In *GECCO '07: Proceedings of the 2007 conference on Genetic and evolutionary computation*, pages 377–384, 2007.

[24] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[25] E.M. Conlon, X.S. Liu, J.D. Lieb, and J.S. Liu. Integrating regulatory motif discovery and genome-wide expression analysis. *Proceedings of the National Academy of Sciences*, 100(6):3339–3344, 2003.

[26] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273, 1995.

[27] G.B. Dantzig. *Linear programming and extensions*. Princeton Univ Pr, 1998.

[28] Modan K Das and Ho-Kwok Dai. A survey of DNA motif finding algorithms. *BMC Bioinformatics*, 8(S21), 2007.

[29] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. Constant creation in grammatical evolution. *Int. J. of Innovative Computing and Applications*, 1:23–38, April 2007.

[30] Patrik D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[31] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[32] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of statistics*, pages 407–451, 2004.

[33] J. Eggermont, A. E. Eiben, and J. I. van Hemert. A comparison of genetic programming variants for data classification. In Eric Postma and Marc Gyssens, editors, *Proceedings of the Eleventh Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'99)*, pages 253–254, Kasteel Vaeshartelt, Maastricht, Holland, November 1999.

[34] Larry Eshelman, Richard A. Caruana, and J. David Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufman.

[35] G. B. Fogel, D. G. Weekes, G. Varga, E. R. Dow, H. B. Harlow, J. E. Onyia, and C. Su. Discovery of sequence motifs related to coexpression of genes using evolutionary computation. *Nucleic Acids Res.*, 32(13):3826–3835, 2004.

[36] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

[37] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Callahan, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems: Proc. of the 2nd Cybernetic Sciences Symposium*, pages 131–155, Washington, D.C., 1965. Spartan Books.

[38] M.C. Frith, U. Hansen, J.L. Spouge, and Z. Weng. Finding functional sequence elements by multiple local alignment. *Nucleic Acids Research*, 32(1):189, 2004.

[39] K. Fukumizu, A. Gretton, X. Sun, and B. Scholkopf. Kernel measures of conditional dependence. *Adv. NIPS*, 2008.

[40] T.S. Furey, N. Cristianini, N. Duffy, D.W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data, 2000.

[41] D. J. Galas and A. Schmitz. DNAse footprinting: a simple method for the detection of protein-DNA binding specificity. *Nucleic Acids Res.*, 5(9):3157–3170, September 1987.

[42] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[43] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[44] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1):389-422, 2002.

[45] Sridhar Hannenhalli. Eukaryotic transcription factor binding sites modeling and integrative search methods. *Bioinformatics*, 24(11):1325–1331, 2008.

[46] G. Harik. Linkage learning via probabilistic modeling in the ecga. Technical Report IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, 1999.

[47] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE-EC*, 3(4):287, November 1999.

[48] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *The Journal of Machine Learning Research*, 5:1391–1415, 2004.

[49] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, July 2001.

[50] Francisco Herrera, Manuel Lozano, and José L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[51] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, Mass., 2nd edition, 1992.

[52] C.E. HORAK and M. SNYDER. ChIP-chip: A genomic approach for identifying transcription factor binding sites. *Methods in enzymology*, 350:469 - 483, 2002.

[53] J. Hu, B. Li, and D. Kihara. Limitations and potentials of current motif discovery algorithms. *Nucleic Acids Research*, 33(15):4899–4913, 2005.

[54] A. Hyvaerinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE-NN*, 10(3):626, May 1999.

[55] Aapo Hyvärinen and Erkki Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4–5):411–430, 2000.

[56] S.T. Jensen, X.S. Liu, Q. Zhou, and J.S. Liu. Computational discovery of gene regulatory binding motifs: a Bayesian perspective. *Statistical Science*, 19(1):188–204, 2004.

[57] R.M. Karp. Reducibility among combinatorial problems. *Complexity of computer computations*, 43:85-103, 1972.

[58] A.E. Kel, O.V. Kel-Margoulis, P.J. Farnham, S.M. Bartley, E. Wingender, and M.Q. Zhang. Computer-assisted identification of cell cycle-related genes: new targets for E2F transcription factors. *Journal of Molecular Biology*, 309(1):99–120, 2001.

[59] S.M. Kielbasa, D. Gonze, and H. Herzel. Measuring similarities between transcription factor binding sites. *BMC Bioinformatics*, 6(1):237, 2005.

[60] C.M. Klinge. Estrogen receptor interaction with estrogen response elements. *Nucleic Acids Research*, 29(14):2905, 2001.

[61] John R. Koza. *Genetic programming: On the programming of computers by natural selection.* MIT Press, Cambridge, Mass., 1992.

[62] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, May 1994.

[63] W. Krivan and W.W. Wasserman. A Predictive Model for Regulatory Sequences Directing Liver-Specific Transcription. *Genome Research*, 11(9):1559, 2001.

[64] W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming.* Springer-Verlag, 2002.

[65] Lozano J. A. Larranaga, P. and E. Bengoetxea. Estimation of distribution algorithms based on multivariate normal and gaussian networks. Technical Report KZZA-IK-1-01, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 2001.

[66] Pedro Larranaga, Ramon Etxeberria, Jose A. Lozano, and Jose M. Pena. Optimization in continuous domains by learning and simulation of gaussian networks. In *Optimization By Building and Using Probabilistic*, pages 201–204, Las Vegas, Nevada, USA, 8 July 2000.

[67] Pedro Larrañaga and Jose A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[68] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wooton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262(8):208–214, October 1993.

[69] CE Lawrence and AA Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7(1):41 51, 1990.

[70] Kwong Sak Leung, Kin Hong Lee, and Sin Man Cheang. Parallel programs are more evolvable than sequential programs. In E. Costa C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, editor, *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*, volume 2610 of *LNCS*, pages 107–118, Essex, UK, 2003. Springer Verlag.

[71] Yiu-Wing Leung and Yuping Wang. An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE-EC*, 5:41-53, February 2001.

[72] G. Li, T.M. Chan, K.S. Leung, and K.H. Lee. An Estimation of Distribution Algorithm for Motif Discovery. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2411–2418, 2008.

[73] Gang Li, Kin-Hong Lee, and Kwong-Sak Leung. Evolve schema directly using instruction matrix based genetic programming. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 271–280, Lausanne, Switzerland, March 2005. Springer-Verlag.

[74] Ming Li, Bin Ma, and Lusheng Wang. Finding similar regions in many sequences. *Journal of Computer and System Sciences*, 65:73–96, 2002.

[75] Y. Li, C. Campbell, and M. Tipping. Bayesian automatic relevance determination algorithms for classifying gene expression data, 2002.

[76] J. S. Liu, A. F. Neuwald, and C. E. Lawrence. Bayesian models for multiple local sequence alignment and Gibbs sampling strategies. *J. Am. Stat. Assoc.*, 90(432):1156–1170, November 1995.

[77] J. S. Liu, A. F. Neuwald, and C. E. Lawrence. Bayesian models for multiple local sequence alignment and Gibbs sampling strategies. *J. American Statistical Association*, 90(432):1156–??, 1995.

[78] J.S. Liu. The Collapsed Gibbs Sampler in Bayesian Computations With Applications to a Gene Regulation Problem. *Journal of the American Statistical Association*, 89(427):958–966, 1994.

[79] M.A. Lones and A.M. Tyrrell. Regulatory Motif Discovery Using a Population Clustering Evolutionary Algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 403–414, 2007.

[80] Thomas Loveard and Victor Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, May 2001. IEEE Press.

[81] L. Meier, S. van de Geer, and P. Buhlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[82] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, third edition, 1996.

[83] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty, editors, *Proceedings of the Third European Conference on Genetic Programming (EuroGP-2000)*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, Scotland, 2000. Springer Verlag.

[84] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[85] TK Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[86] Ruadhan A. O'Flanagan, Guillaume Paillard, Richard Lavery, and Anirvan M. Sengupta. Non-additivity in protein-DNA binding. *Bioinformatics*, 21(10):2254–2263, 2005.

[87] M. Pelikan, D.E. Goldberg, and E. Cantu-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume 1, pages 525–532. Citeseer, 1999.

[88] M. Pelikan, K. Sastry, M.V. Butz, and D.E. Goldberg. Hierarchical BOA on random decomposable problems. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 431–432. ACM New York, NY, USA, 2006.

[89] Tim Perkis. Stack-based genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[90] P.A. Pevzner and S.H. Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278, 2000.

[91] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.

[92] M.A. Potter and K.A. De Jong. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[93] Z.S. Qin, L.A. McCue, W. Thompson, L. Mayerhofer, C.E. Lawrence, and J.S. Liu. Identification of co-regulated genes through Bayesian clustering of predicted regulatory binding sites. *Nature Biotechnology*, 21:435–439, 2003.

[94] JR Quinlan. *Data Mining Tools See5 and C5.0*. http://www.rulequest.com/see5-info.html, 2007.

[95] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[96] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 775–782. ACM New York, NY, USA, 2007.

[97] B. Raphael, Liu Lung-Tien, and G. Varghese. A uniform projection method for motif discovery in dna sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(2):91–94, 2004.

[98] G. Rätsch, S. Sonnenburg, and C. Schäfer. Learning interpretable SVMs for biological sequence classification. *BMC bioinformatics*, 7(Suppl 1):S9, 2006.

[99] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[100] Sergio A. Rojas and Peter J. Bentley. A grid-based ant colony system for automatic program synthesis. In Maarten Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, 26 July 2004.

[101] Justinian P. Rosca. Analysis of complexity drift in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 286–294, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[102] M. F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *LATIN '98: Theoretical Informatics, Lecture Notes in Computer Science*, pages 111–127. Springer-Verlag, 1998.

[103] Rafal Salustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

[104] G.K. Sandve and F. Drablos. A survey of motif discovery methods in an integrated framework. *Biology Direct*, 1(11), 2006.

[105] Kumara Sastry and David E. Goldberg. Probabilistic model building and competent genetic programming. In Rick L. Riolo and Bill Worzel, editors, *Genetic Programming Theory and Practice*, chapter 13, pages 205–220. Kluwer, 2003.

[106] Martin Schmidt, Kim Kristensen, and Thomas Randers Jensen. Adding genetics to the standard PBIL algorithm. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1527–1534, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[107] B. Scholkopf and A.J. Smola. *Learning with kernels*. MIT press Cambridge, MA, 2002.

[108] Hans Paul Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series. John Wiley & Sons, Inc., New York, 1995.

[109] Y. Shan, R. I. McKay, H. A. Abbass, and D. Essam. Program evolution with explicit learning: a new framework for program automatic synthesis. In Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1639–1646, Canberra, 8-12 December 2003. IEEE Press.

[110] Yin Shan, Robert I. McKay, Rohan Baxter, Hussein Abbass, Daryl Essam, and Hoai Nguyen. Grammar model-based program evolution. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 478–485, Portland, Oregon, 20-23 June 2004. IEEE Press.

[111] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531 -1565, 2006.

[112] G.D. Stormo, G.W. Hartzell, et al. Identifying Protein-Binding Sites from Unaligned DNA Fragments. *Proceedings of the National Academy of Sciences of the United States of America*, 86(4):1183–1187, 1989.

[113] Gilbert Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, June 1989.

[114] Masato Takahashi and Hajime Kita. A crossover operator using independent component analysis for real-coded genetic algorithms. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 643–649, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[115] Dirk Thierens and David E. Goldberg. Mixing in genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference*.

on Genetic Algorithms, pages 38–45, San Mateo, CA, 1993. Morgan Kaufman.

[116] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[117] M. Tompa, N. Li, and T.L Bailey. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23:137–144, 2005.

[118] Athanasios Tsakonas. A comparison of classification accuracy of four genetic programming-evolved intelligent structures. *Information Sciences*, 176(6):691–724, March 2006.

[119] Zhi Wei and Shane T. Jensen. GAME: detecting cis-regulatory elements using a genetic algorithm. *Bioinformatics*, 22(13):1577–1584, 2006.

[120] Peter A. Whigham. Grammatically-based genetic programming. In J. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, San Mateo, CA, July 1995. Morgan Kaufmann.

[121] DH Wolpert, WG Macready, I.B.M.A.R. Center, and CA San Jose. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[122] Man Leung Wong and Kwong Sak Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*, volume 3 of *Genetic Programming*. Kluwer Academic Publishers, January 2000.

[123] A. Wright. Genetic algorithms for real parameter optimization. In G. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, 1991.

[124] Xin Yao and Yong Liu. Fast evolution strategies. In Peter J. Angeline, Robert G. Reynolds, John R. McDonnell, and Russ Eberhart, editors, *Evolutionary Programming VI*, pages 151–161, Berlin, 1997. Springer. Lecture Notes in Computer Science 1213.

[125] Mengjie Zhang and Will Smart. Genetic programming with gradient descent search for multiclass object classification. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 399–408, Coimbra, Portugal, April 2004. Springer-Verlag.

[126] Qingfu Zhang, Nigel M. Allinson, and Hujun Yin. Population optimization algorithm based on ICA. In *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pages 33–36, May 02 2000.

[127] Qing Zhou and Jun S. Liu. Modeling within-motif dependence for transcription factor binding site predictions. *Bioinformatics*, 20(6):909–916, 2004.

[128] J. Zhu. SCPD: a promoter database of the yeast Saccharomyces cerevisiae. *Bioinformatics*, 15(7):607–611, 1999.

[129] J. Zhu, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*, page 49. Bradford Book, 2004.

[130] Douglas Zongker and Bill Punch. lilgp 1.01 user's manual. Technical report, Michigan State University, USA, 26 March 1996.

[131] H. Zou. The Adaptive Lasso and Its Oracle Properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006.

Appendix A

Sparse Kernel Feature Machine

A.1 Overview

In Machine Learning and Pattern Recognition, one of the fundamental problems is regression. Given N pairs of training samples $\{(x_i, y_i)\}_{i=1}^{N}$, where x_i consists of m attributes $(x_i^j)_{j=1}^{m}$ and y_i is the corresponding target, regression is to estimate the function from $\{x_i\}_{i=1}^{N}$ to $\{y_i\}_{i=1}^{N}$. The samples $\{x_i\}_{i=1}^{N}$ are independently and identically distributed (i.i.d.) and the targets are usually corrupted with noise, i.e., $\{y_i = \tilde{y}_i + \xi_i\}_{i=1}^{N}$, where $\tilde{y}_i$ is the true response and ξ_i is a noisy variable following an unknown distribution. In regression, the target is a real number to be predicted by the regression function. If the target is nominal, and binary in particular, the regression function is also a classifier to predict the class of a sample.

Kernel methods have shown some successes in solving the regression and classification problems [107]. Instead of performing linear methods in the original space, kernel methods implicitly map the data into a higher dimensional feature space and apply linear methods in the feature space. The kernel methods are significant for two reasons. First, by utilizing linear methods in the feature space, kernel methods actually perform nonlinear learning (regression or classification) in the original space. Second, the implicit mapping from the original space into the high-dimensional feature space is accomplished through the so-called kernel tricks, and thus kernel methods avoid the time-consuming space conversion and high-dimensional linear learning.

Formally, kernel methods models the regression or the classification function using the expansion in Eq. A.1, where $\{\alpha\}_{i=0}^{N}$ are the function coefficients to be learned. $k(\cdot,\cdot)$ in Eq. A.2 is a kernel function which is equivalent to the dot product of the images of the arguments in the feature space. $\phi(\cdot)$ is the mapping from the original space to the high-dimensional feature space. If the mapping is simply the identity function, the kernel function $k(\cdot,\cdot)$ is the dot product in the original space and the function in Eq. A.1 is actually linear. With complex kernel functions and thus non-linear mappings, Eq. A.1 becomes a non-linear function.

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i) + \alpha_0 \tag{A.1}$$

$$k(x_1, x_2) = < \phi(x_1) \cdot \phi(x_2) > \tag{A.2}$$

With the form of the learning function in Eq. A.1, the objective of kernel methods is usually formulated as Eq. A.3, where $L(\cdot)$ is a loss function as a measure of the difference between the true target y and the predicted target $f(x)$. In regression, the loss function is often the sum of squared errors, since the noise is assumed to be Gaussian distributed. In classification, the loss function can be the hinge loss function, which returns zero as long as the predicted target and the true target are on the same side of a threshold. $\Omega f(\cdot)$ measures the complexity of the function $f(\cdot)$, which are related to the structure and the parameters of the function. λ is the regularization parameter which tradeoffs the learning objective between the loss function and the function complexity.

$$argmin_{\alpha} \sum_{i=1}^{N} L(f(x_i|\alpha), y_i) + \lambda \Omega f(\cdot|\alpha) \tag{A.3}$$

Regularizing the function complexity is related to the model selection, which is one of the central problems in regression and classification. Due to the noise in the training samples, a complicated learning function may overfit the training dataset unnecessarily and thus generalizes poorly with respect to (w.r.t.) the testing dataset. To counteract the effect of overfitting, the regularization on the model complexity is imposed so that a simple model is preferred since it happens to be unable to fit the noise in the training dataset. In Eq. A.3, the two terms of the training error and the model complexity are combined to form a single objective function, where the relative importance of the two terms are controlled with the regularization parameter λ.

However, it is difficult to choose an appropriate λ without prior knowledge of the problem. A too large λ may over-penalize the function complexity and make the function underfit the data. A too small λ may be unable to prevent the overfitting and the resulted learning function is too complex to fit the testing data well. In practice, people may try out a series of different λs and use cross-validation to select the most appropriate λ. This approach is quite time consuming since the learning process has to be repeated for each different λ.

Another problem with the usual kernels in Eq. A.2 is that it neglects feature selection, i.e., using only a subset of the features from $\{x^j\}_{j=1}^m$ in the learning function. Feature selection poses two advantages for regression and classification. First, by selecting only the relevant features, the learning method discards the irrelevant features and thus controls the model complexity effectively. Second, with the useful features highlighted, it is easy for human to interpret the relevant features in the resulted regression function. The kernel in Eq. A.2 involves all the features and thus it is unable to discriminate between relevant and irrelevant features.

This appendix proposes a new kernel learning method, i.e., Sparse Kernel Feature Machine (SKFM), to address the two aforementioned problems with the standard kernel learning. Instead of using the kernel functions of all the attributes, SKFM equips a kernel function and forms a kernel feature for each original attribute. An augmented kernel matrix is constructed by concatenating all the kernel matrices of the attributes. Least Angle Regression is then applied on the augmented kernel matrix to perform step-wise linear regression in the feature space. Collinear kernel features are detected and removed from the set of the kernel features automatically. SKFM forms the solution path of the function coefficients which is piece-wise linear in the regularization parameter, and then SKFM interpolates the solutions of a series of different regularization parameters and chooses the best parameter in cross-validation. Compared to using all the kernel features, SKFM selects the kernel features one by one in the solution path, and uses only the most important kernel features in the final solution. In the experiments, SKFM has been tested on four real medical classification problems, i.e., Diabetes, Hepatitis B Virus, Colon Cancer and *C. elegans*. The results verify that SKFM not only outperforms Support Vector Machine (SVM), but it also points out the most important features in the classification.

The rest of the appendix is organized as follows. Section A.2 briefly introduces the existing methods related to SKFM. Section A.3 describes the architecture and the algorithms of SKFM in detail. Section A.4 gives the experimental results of SKFM on some real problems. Section A.5 is the discussion about SKFM.

A.2 Related Work

A.2.1 Least Angle Regression

Least Angle Regression (LARS) [32] is a special algorithm for linear regression in Eq. A.4, where x is a row vector of a data sample and a dataset is denoted as a matrix $X = (x^j)_{j=1}^{m}$ consisting of the columns of attributes. Starting from an empty set, LARS adds attributes in the linear model one by one and solves the corresponding coefficients $(\beta^j)_{j=1}^{m}$ along the way. LARS thus find a series of solutions which consist of overlapping attribute sets $\{\beta_q\}_{q=1}^{Q}$ of increasing sizes. For high-dimensional problems, solving for the solution of all the dimensions is computationally inhibitive. In addition, the resulted ordinary linear square (OLS) estimate is easily overfitting since there may be insufficient data given the large number of dimensions, let alone that many of the dimensions may be irrelevant. Therefore, as a feature selection algorithm, LARS is fast and robust.

$$f(x) = x\beta + \beta^0 \tag{A.4}$$

There are three particular advantages of LARS compared to other linear model selection algorithms. First, LARS is less greedy than forward selection algorithm in adding features. In forward selection algorithm, a new feature is added if it has the maximum correlation with the current residual (the part of the response not explained), when the rest of the response has been fully explained by the features already selected. However, LARS selects a new feature when it has the same correlation with the current residual as the features already selected, and so a useful feature is unlikely to be overlooked as in the aggressive forward selection algorithm. Second, although LARS finds a series of different solutions, each solution is not computed from scratch on its own. Instead, a solution is computed on the basis of the one before, and so LARS is computationally efficient.

A significant advantage of LARS is that it is capable of generating the solution path w.r.t. the regularization parameter t in "least absolute shrinkage and selection operator" (Lasso) [116] in Eq. A.5. Lasso is a linear implementation of the general objective function in Eq. A.3 with L1-norm regularization on the learning coefficients. The solution path of Lasso is piece-wise linear in the regularization parameter t, meaning that the solution β follows a linear function of t given a fixed set of selected features. When t changes to the point where the selected features change (adding or removing a feature), the solution β follows a different linear function. The reason for the linearity is that given a fixed set of features, the optimal solution of Eq. A.5 can be obtained by equating its partial derivative of its Lagrangian form to zero. Clearly, the resulted root (the coefficient solution) of the zero derivative is a linear function of t.

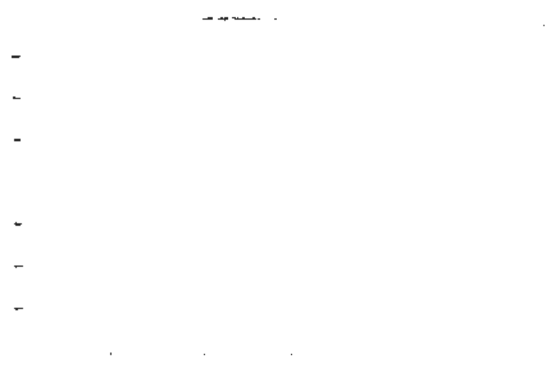


Figure A.1: The solution path of the linear coefficients and the evolution of the testing error by LARS

$$\beta = argmin_{\beta} \sum_{i=1}^{N} (y_i - x_i\beta)^2 \ s.t. \sum_{j=1}^{m} |\beta^j| < t \tag{A.5}$$

An example of a solution path and the evolution of the testing error is illustrated in Fig. A.1 (generated from the LARS coded by Karl Skoglund). The horizontal axes are the normalized numbers of the iterations, which can also be viewed as $\sum_{j=1}^{m} |\beta^j|$ in Eq. A.5, i.e., the regularization parameter t. On the right, the figure shows the testing errors of the models under different regularization parameter, and the minimal testing error is obtained with a modest t. On the left, the vertical axis is the value of a coefficient, and each curve is the path of a coefficient changing along with the iteration. Whenever a new curve appears or an existing curve reaches zero (the set of selected features changes), an existing straight line turns its direction into another straight line. Clearly, the coefficient between any two adjacent joining knots can be linearly interpolated from the values of the coefficients on the surrounding knots.

This property of piece-wise linearity is very useful in practice. In each fold of cross-validation, people can run LARS on the training dataset once and obtain a series of solutions. The computation cost of the calculating solution path is significantly less than that of the OLS estimate in a high-dimensional problem. With the solution path, the solutions under various regularizing parameter t can be computed directly using simple linear interpolation. The most appropriate t with the least validation error is selected to be used in training on the whole dataset.

A.2.2 Support Vector Machine

Support Vector Machine (SVM) [107] is a popular and powerful kernel learning method for regression and classification. In regression, it learns a function $f(x)$ which fits the targets within a certain bound and whose curve is pushed as flat as possible. The learning function in SVM is the same as the usual kernel learning function in Eq. A.1. The objective function for SVM optimization is adapted from Eqs. A.3 to A.6, which is a weighted combination of the hinge loss function $[\cdot,\cdot]_+$ and the functional norm in the Hilbert space induced from the kernel function, i.e., $\alpha^T K\alpha$. The hinge loss function used here returns zero as long as the $f(x_i|k)$ and y_i are of the same sign and $|f(x_i|k)|$ is larger than one. According to the Karush-Kuhn-Tucker (KKT) conditions, many function coefficients of α in the minimal solution of Eq. A.6 are zero, and so a sparse solution is obtained in SVM.

$$\sum_{i=1}^{N}[1-y_i f(x_i|k)]_+ + C\alpha^T K\alpha \tag{A.6}$$

A typical kernel function used in SVM and other kernel learning methods is the Gaussian kernel in Eq. A.7, where σ is a kernel parameter specified beforehand. The kernel trick is used to implicitly map the data into a high-dimensional feature space without actually computing the costly mapping. Another advantage is that while a problem is linearly inseparable in the original space, it may become linearly separable in the feature space and easy to be solved.

$$k(x_1,x_2) = exp^{-\frac{(x_1-x_2)^2}{2\sigma^2}} \tag{A.7}$$

The solution path in SVM has been investigated in [48] and [129]. [48] solves SVM in its original form of ℓ_2-norm as in Eq. A.6. The optimal solution of the objective function in Eq. A.6 is clearly a linear function of the regularization parameter C, as long as the supporting vectors with non-zero α remain the same. In [48], the algorithm starts with a zero C, and gradually increase it to infinity. During this process, the algorithm keeps track of all the events when certain vectors reverse their roles as supporting vectors or non-supporting vectors. In this way, the algorithm is able to build the entire solution path w.r.t. C.

[129] solves for the solution path in L1-norm SVM, whose regression and objective functions are Eqs. A.1 and A.3, respectively. Instead of using the regularization on the functional complexity, a constraint on the ℓ_1-norm of the function coefficients is imposed to encourage the coefficients to be sparse. It is proved that the derivatives of $\{\alpha^j\}_{j=1}^N$ w.r.t. s is piece-wise constant, and thus $\{\alpha^j\}_{j=1}^N$ is piece-wise linear w.r.t. s.

$$\begin{aligned} min \quad & \sum_{i=1}^{N}[1-y_i(\alpha^0+\sum_{j=1}^{N}\alpha^j k(x_j,x_i))]_+ \\ s.t. \quad & \|\alpha\|_1=\sum_{j=1}^{N}|\alpha^j|<s \end{aligned} \tag{A.8}$$

A.2.3 Multiple Kernel Learning

Multiple Kernel Learning (MKL) [4][111][96] uses multiple kernel functions in the kernel regression or classification. There exists various kernel functions of various properties. Even for the same kernel function, different kernel parameters, such as the σ in a Gaussian kernel, may induce different feature spaces, and thus different learning functions are obtained. Without the prior knowledge of the nature of the problem, it is desirable to infer the appropriate kernel automatically during the learning process. In MKL, the kernel is usually a weighted sum of a set of candidate kernel functions as in Eq. A.9, where the constraint $\sum_{i=1}^K \beta_i = 1$ is imposed to force some weights to zero.

$$\begin{aligned} k(x_1,x_2) &= \sum_{i=1}^{K} \beta_i k_i(x_1,x_2) \\ s.t. \quad & \beta_i \geq 0 \; and \; \sum_{i=1}^{K} \beta_i = 1 \end{aligned} \tag{A.9}$$

There are various formulations of MKL, depending on how the functional complexity and the constraints are defined. One of the formulations for generic loss function is as Eq. A.10. $\mathbf{w}_i$ is the multivariate coefficient vector associated with kernel function $k_i(\cdot)$. The kernel weight β_i in Eq. A.9 is omitted since it is absorbed into the corresponding $\mathbf{w}_i$. In the objective function, the block ℓ_1-norm $(\sum_{k=1}^{K} ||\mathbf{w}_k||_2)^2$ is used to encourage the $\{\mathbf{w}_i\}_{i=1}^{K}$ to be zero vectors, which is also used in group Lasso [81].

$$\begin{aligned} min \quad & \frac{C}{2}(\sum_{k=1}^{K} ||\mathbf{w}_k||_2)^2 + \sum_{i=1}^{N} L(f(x_i),y_i) \\ s.t. \quad & f(x_i) = \sum_{k=1}^{K} < \phi_k(x_i) \cdot \mathbf{w}_k > +b, \forall i = 1,\ldots,N \end{aligned} \tag{A.10}$$

[5] considers the solution path w.r.t. the regularization parameter C in MKL. However, the solution path is no longer piece-wise linear, and instead it is estimated by the logarithmic barrier and numerical continuation techniques. MKL can also be used for feature selection if a kernel function is associated with each original dimension [98]. As a result, the dimensions of non-zero kernel weights $\{\beta_i\}_{i=1}^{K}$ in Eq. A.9 are the selected features.

A.2.4 Mutual Information

Mutual Information (MI) is a measure of the dependency between two (multivariate) variables. Its theoretical formula is given by Eq. A.11, where $p(\cdot)$ is the corresponding probability density function of the argument variable. MI can be used to measure the dependency between the features and the target, and thus to select features in classification and regression. Unfortunately, it is difficult, if not impossible, to estimate the probability densities, especially in high-dimensional space. Therefore, many algorithms use only pair-wise MI to filter out irrelevant features before learning [43], which can be estimate using discretization and non-parametric methods. However, the relevant features clearly have a joint effect on the target, and thus an estimate of the joint multivariate MI is more meaningful.

$$I(x_1;x_2) = \int_{-\infty}^{\infty} p(x_1,x_2) log \frac{p(x_1,x_2)}{p(x_1)p(x_2)} \qquad \text{(A.11)}$$

Some kernel methods have been proposed to measure MI approximately [3][39]. [39] uses the Hilbert-Schmidt norm of the normalized cross-covariance operator as the measure of dependency. For the centered kernel matrices G_X and G_Y of the features and the target, let $R_X = G_X(G_X + n\varepsilon_n I_n)^{-1}$ and $R_Y = G_Y(G_Y + n\varepsilon_n I_n)^{-1}$, where ε_n is a regularization parameter, and the empirically dependence measure I^{NOCCO} (*normalized cross-covariance operator*) is defined in Eq. A.12, where $Tr[\cdot]$ is the matrix trace. This measure is originally proposed to find the transformation which results in independent variables in Independent Component Analysis, but it can be potentially used to select the features which have high I^{NOCCO} with the target.

$$I^{NOCCO}(X;Y) = Tr[R_Y R_X] \qquad \text{(A.12)}$$

A.3 Architecture

Sparse Kernel Feature Machine (SKFM) is a kernel learning framework for classification and regression. It performs kernel learning and feature selection simultaneously, generates the solution path w.r.t. the regularization parameter, and selects the most appropriate regularization parameter automatically. This section consists of three parts. Section A.3.1 proposes the objective function in SKFM and depicts the overall picture of SKFM. Section A.3.2 presents the data structure used in SKFM and how it is obtained. Section A.3.3 describes the algorithm to find the solution path of the objective function in SKFM.

A.3.1 Overall Program

Eq. A.13 is the learning function in SKFM. $f(\cdot|k)$ is a nonlinear function realized with dimension-wise kernels $\{k^l\}_{l=1}^{m}$, where $(\beta_{i,l})_{i=1\cdots N}^{l=1\cdots m}$ is the matrix of the learning coefficients associating with the kernels. SKFM aims to minimize the objective function in Eq. A.14. The traditional sum of squared errors is used as the loss function to simplify the optimization, and ℓ_1-norm on the learning coefficients is used to force the learning coefficients to be sparse.

$$f(x_i) = \sum_{j=1}^{N} \sum_{l=1}^{m} \beta^{j,l} k^l(x_i^l, x_j^l) + \beta^0 \tag{A.13}$$

$$\underset{\beta}{\text{argmin}} \sum_{i=1}^{N} (y_i - \beta^0 - \sum_{j=1}^{N} \sum_{l=1}^{m} \beta^{j,l} k^l(x_i^l, x_j^l))^2 \ s.t. \sum_{j=1}^{N} \sum_{l=1}^{m} |\beta^{j,l}| < t \tag{A.14}$$

Basically, SKFM applies a Kernelized Least Angle Regression (KLARS) on the augmented kernel matrix of the data samples to learn the coefficients in Eq. ??. It addresses the issue of curse-of-dimensionality in two perspectives. If the dimension of the problem is larger than the number of samples, the irrelevant features are detected and removed from the dataset using Kernelized Mutual Information

(KMI). KLARS selects the relevant features one by one in the order of their usefulness to predicting the targets. SKFM accomplishes nonlinear learning by running LARS in the feature space. LARS is traditionally used for linear regression in the original space. Since the data are nonlinearly mapped into the feature space, linear regression in the feature space is actually nonlinear regression in the original space. SKFM selects the best regularization parameter without learning using different regularization parameters repeatedly. Instead, the solutions of a series of regularization parameters are easily interpolated from the solution path returned by KLARS, and then the most appropriate regularization parameter is selected with the least validation error.

Algorithm A.1 is the pseudo-code of the overall program of SKFM. If the problem is high-dimensional, SKFM first uses KMI to select only the relevant features to reduce the dimensions. Then SKFM employs cross-validation to select the most appropriate regularization parameter which results in the solution of the least average validation error. Afterwards, the best regularization parameter is used to train the whole dataset. To make use of LARS in SKFM, the learning function must be a regression function predicting a real-valued response. Therefore, to use the real-valued response for classification, a threshold on the response is estimated to maximize the training accuracy.

In learning on each fold of cross-validation, SKFM carries out the following steps in sequence. In the first step, SKFM constructs the augmented kernel matrix of the training part by concatenating all the kernel matrices of the features. Instead of using a kernel function of all the dimensions, SKFM equips a kernel function and forms a kernel matrix for each original feature. Each column in the augmented kernel matrix is treated as a kernel feature. In the second step, SKFM performs KLARS on the augmented kernel matrix to perform step-wise linear regression in the feature space. Collinear kernel features are detected and removed from the set of the kernel features automatically. KLARS returns the solution path of the function coefficients which is piece-wise linear in the regularization parameter. In the third

Algorithm A.1: Sparse Kernel Feature Machine

```
Input: the training dataset $(X, Y)$
Output: the solution of coefficients $\beta$
if high dimensional problem then
    use Kernelized MI for feature filtering
partition $(X, Y)$ into 5 fold of training parts and validation parts $\{(X_i, Y_i)\}_{i=1}^{5}$;
foreach fold $(X_i, Y_i)$ do
    construct the augmented kernel matrix $A_i^{tr}$ of the training part $X_i^{tr}$ alone;
    perform Kernelized LARS on $A_i^{tr}$ and $Y_i^{tr}$;
    interpolate the solutions $S_i = \{s_i^j\}_{j=1}^{|T|}$ on the regularization parameters $T = \{t^j\}_{j=1}^{|T|}$;
    construct the augmented kernel matrix $G_i^{tv}$ of the training and validation parts $(X_i^{tr}, X_i^{va})$;
    measure the validation errors $e_i = \{e_i^j\}_{j=1}^{|T|}$ of $s_i$ with $A_i^{tv}$
select the regularization parameter $t = argmin_t(mean(\{e_i\}_{i=1}^{5}))$ of the least average validation error;
construct the augmented kernel matrix $A$ of $X$;
perform Kernelized LARS on $A$ and $Y$;
interpolate the solution $\beta$ of the best regularization parameter $t$;
find the threshold for $\beta$ to classify the training data
```

step, SKFM linearly interpolate the solutions under a set of different regularization parameters, which are tested on the validation part to measure their validation errors.

The details of the procedures in Algorithm A.1 will be described in the following sections. KMI and the augmented kernel matrix are described in Section A.3.2. KLARS and interpolating solutions are described in Section A.3.3.

A.3.2 Augmented Kernel Matrix

In the standard kernel learning, the learning function in Eq. A.1 can be written in the matrix form in Eq. A.15, where the function values of all the kernel functions on all the data samples constitute a kernel matrix $K = (k(x_i, x_j))_{i,j=1}^{N}$. Kernel learning minimizes the objective function in Eq. A.16 by calculating the optimal weights α for all the kernel vectors in the kernel matrix.

$$f(x_i) = K_i\alpha + \alpha_0 \tag{A.15}$$

$$\begin{pmatrix} k^1(x_1,x_1) & \cdots & k^1(x_1,x_N) & \cdots & k^j(x_1,x_1) & \cdots & k^j(x_1,x_N) & k^m(x_1,x_1) & k^m(x_1,x_N) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & & & \\ k^1(x_N,x_1) & \cdots & k^1(x_N,x_N) & \cdots & k^j(x_1,x_1) & & k^j(x_1,x_N) & \cdots \; k^m(x_N,x_1) & k^m(x_N,x_N) \end{pmatrix}$$

Figure A.2: An example of augmented kernel matrix

$$\sum_{i=1}^{N} L(y_i, f(x_i|\alpha)) + C\Omega f(\cdot|\alpha) \tag{A.16}$$

An augmented kernel matrix is proposed to incorporate feature selection in kernel learning. Since the kernel function in Eq. A.15 involves all the dimensions of the data, it is not easy to perform feature selection during learning. SKFM resolves this disadvantage by using one-dimensional kernel functions for all the dimensions separately, and thus the kernel matrix in SKFM is the concatenation of all the kernel matrices of the individual dimensions. Suppose the kernel matrix of a single dimension i is M^i, the augmented kernel matrix of all the dimensions is then $(M^j \cdots M^m)$. An example kernel matrix is shown in Fig. A.2, which has N rows and mN columns. In analogy to the data matrix X, whose columns are the data features, the columns of the augmented kernel matrix are called the kernel features in SKFM.

In the form of the augmented kernel matrix M, the learning function of SKFM in Eq. A.13 can be rewritten as Eq. A.17 and the objective function in Eq. A.14 can be rewritten as Eq. A.18. M_i is a row vector in A representing all the kernel features of data sample x_i, and α is the vector of the concatenation of all the column vectors in β. β^0 and α^0 actually denote the same variable in both forms of the objective function. Interestingly, the optimal solution to Eq. A.18 not only tells which original features are important, but also tells which values on the selected features are important since a kernel feature M^j is parameterized with a specific value on a specific feature.

$$f(x_i) = M_i\alpha + \alpha_0 \tag{A.17}$$

$$\underset{\alpha}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \alpha^0 - M_i \alpha)^2 \; s.t. \sum_{j=1}^{mN} |\alpha^j| < t \tag{A.18}$$

However, in high-dimensional problems, especially when the number of the dimensions is larger than the number of the data samples, there is a huge number mN of kernel features. Under such circumstances, the optimal solution is not unique and very likely to be overfitted. Therefore, it is necessary to detect and remove irrelevant features from the original data first.

SKFM uses Sparse Kernelized Mutual Information (SKMI) similar to I^{NOCCO} in Eq. A.12 to measure the joint dependency between the features X and the response Y. Optimizing the overall SKMI w.r.t. the data features tells the degrees of the dependencies of the response on the individual features. There are some existing methods estimating the pair-wise MI between the individual dimensions and the response. However, such methods usually ignores the joint dependency between the response and multiple features, which may have redundancy among themselves. While there are also some methods measuring the joint MI between a subset of features and the response, they have to select the useful features one by one using local search methods. The resulted feature set may be overly greedy and order-dependent, and thus it may miss some relevant features.

The original I^{NOCCO} measures the dependency between two variables only. To apply it on multivariate cases, the centered kernel matrix of a set of features is constructed as a weighted conic combination of the kernel matrices on the individual dimensions as $G_X = \sum_{j=1}^{m} w_j^2 G_{X^j}$. The square of a weight is used to make sure it is positive and the resulted kernel matrix is still positive-definite. With this new kernel matrix, the objective function to maximize the $NOCCO$ between the predictors and the response is Eq. A.19. The constraint on the sum of the squares of the weights makes all the w^2 less than 1, otherwise w^2 would grow to infinity to maximize the $NOCCO$. With such a constraint, more relevant features will have relatively large weights, while less relevant features will have relatively small weights. To further push the small weights to zero, a ℓ_1-norm regularization on the squares of the weights are included in the objective function of SKMI as Eq. A.20, where λ_j is the adaptive individual regularization parameter of weight w_j.

$$\begin{aligned} I^{NOCCO}(X;Y|w) &= Tr[G_Y(G_Y+n\varepsilon_n I)^{-1}\sum w_j^2 G_{X_j}(\sum w_j^2 G_{X_j}+n\varepsilon_n I)^{-1}] \\ &= Tr[(\sum w_j^2 G_{W_j}+n\varepsilon_n I)^{-1} G_Y(G_Y+n\varepsilon_n I)^{-1}\sum w_j^2 G_{X_j}] \\ s.t. \quad & \sum_{j=1}^{m} w_j^2 = 1 \end{aligned} \tag{A.19}$$

$$\begin{aligned} SKMI(X;Y|w) &= Tr[(\sum w_j^2 G_{X_j}+n\varepsilon_n I)^{-1} G_Y(G_Y+n\varepsilon_n I)^{-1}\sum w_j^2 G_{X_j}] + \sum_{j=1}^{m}\lambda_j w_j^2 \\ s.t. \quad & \sum_{j=1}^{m} w_j^2 = 1 \end{aligned} \tag{A.20}$$

The negatives of the objective functions in Eqs. A.19 and A.20 are non-convex with multiple local optimal solutions. In the current implementation, SKFM uses natural gradient descent [1] to find the local optimum from the initial weights. Since the objective functions are multi-modal and the gradient descent is a local greedy search method, good initial weights are vital to finding the global maximum. SKFM uses the pair-wise I^{NOCCO} between the individual dimensions and the target as the initial weights.

Subsequently, SKFM maximizes the objective functions Eqs. A.19 and A.20 in two phases. In the first phase, SKFM maximizes the objective function $I^{NOCCO}(X;Y|w)$ in Eq. A.19 to find an optimal set of weights, i.e, w^{NOCCO}. In the second phase, w^{NOCCO} are used as the initial weights to maximize the objective function $SKMI(X;Y|w)$ in Eq. A.20. Eq. A.20 is inspired by Adaptive Lasso [131] to further push w^{NOCCO} towards zeros. The pushing degrees are controlled by the regularization parameters $\{\lambda_j = 1/w_j^{NOCCO}\}_{j=1}^{m}$. Intuitively, a small weight has a large regularization param ·ter, which pushes it further to zero.

A new set of weights updated with gradient descent may violate the constraint $\sum_{j=1}^{m} w_j^2 = 1$. To constantly satisfy the constraint, the natural gradient descent in Eq. A.21 is used to update the weights iteratively, where η is a small learning rate and $\frac{\partial f}{\partial w}$ is the derivative of the objective function f w.r.t. w. The derivatives of Eq. A.19 and Eq. A.20 w.r.t. the weight w_i are shown in Eq. A.22.

$$w' = w + \eta w(\frac{\partial f}{\partial w})^T w \qquad (A.21)$$

$$\begin{aligned}
\frac{\partial I^{NOCCO}(X;Y)}{\partial w_i} &= 2Tr[(\sum w_j^2 G_{X^j} + n\varepsilon_n I)^{-1} G_Y (G_Y + n\varepsilon_n I)^{-1} \\
&\quad \times (I - \sum w_j^2 G_{X_j} (\sum w_j^2 G_{X^j} + n\varepsilon_n I)^{-1}) w_i G_{X^i}] \\
\frac{\partial SKMI(X;Y)}{\partial w_i} &= 2Tr[(\sum w_j^2 G_{X^j} + n\varepsilon_n I)^{-1} G_Y (G_Y + n\varepsilon_n I)^{-1} \\
&\quad \times (I - \sum w_j^2 G_{X^j} (\sum w_j^2 G_{X^j} + n\varepsilon_n I)^{-1}) w_i G_{X^i}] + 2\lambda_i w_i
\end{aligned}$$
(A.22)

After SKFM maximizes Eqs. A.19 and A.20, the dimensions of non-zero weights are identified as relevant features and retained in constructing the augmented kernel matrix for further processing, while the dimensions of zero weights are deemed irrelevant and removed from the dataset.

A.3.3 Kernelized Least Angle Regression

The objective function Eq. A.18 can be written in a simple form as Eq. A.23. It is assumed that both A and Y have been standardized, and so the implicit zero intercept α_0 is omitted. If Eq. A.23 is compared with the objective function in Lasso Eq. A.24, an analogy can be immediately drawn between M and X. Therefore, LARS, which solves Lasso in Eq. A.24, is also able to solve Eq. A.23 efficiently.

$$\underset{\alpha}{\operatorname{argmin}}\,(Y - M\alpha)^T(Y - M\alpha)\ s.t. \|\alpha\|_1 < t \tag{A.23}$$

$$\underset{\alpha}{\operatorname{argmin}}\,(Y - X\alpha)^T(Y - X\alpha)\ s.t. \|\alpha\|_1 < t \tag{A.24}$$

The original LARS assumes that the features are linearly independent. However, the kernel features are easily collinear, especially in discrete problems, because the values on an individual dimension are very likely to be identical and thus the corresponding kernel features are equivalent. Therefore, special care must be taken to remove collinear kernel features from the augmented kernel matrix.

Algorithm A.2 outlines the major steps in the implementation of Kernelized LARS (KLARS) for SKFM. KLARS selects the kernel features iteratively and builds up the kernel regression and the coefficients of the kernel features stepwise. Initially, the regression is $\mu_0 = \mathbf{0}$, the coefficients are $\alpha_0 = \mathbf{0}$ and the residual between the regression and the response is $\mathbf{r}_0 = y - \mu_0$. In each iteration, firstly the unselected feature of the maximal correlation with the current residual is identified and selected into the active feature set. Then the equiangular vector of the active feature set is determined to be used to regress the residual in the current iteration. The scale of the equiangular vector is calculated to determine the current regression, and the coefficients are updated accordingly. Meanwhile, the kernel features which are collinear with the active features are removed. When the norm of the residual converges below a predefined bound, the iteration terminates and returns a series of solutions of the coefficients. The details of the steps in Algorithm A.2 are described as follows.

With the initial regression $\mu_0 = \mathbf{0}$ and the coefficients $\alpha_0 = \mathbf{0}$, KLARS builds up $\{\mu_i\}$ and $\{\alpha_i\}$ iteratively by adding active kernel features. In an iteration, the residual is $y - \mu_i$ and the correlation of the kernel features with the residual is $c = M'(y - \mu_i)$. For the set of indices $\mathcal{A}$ which correspond to the active features, define a matrix of the active kernel features only as $M_{\mathcal{A}} = (\cdots s^j M^j \cdots)_{j \in \mathcal{A}}$, where s^j is the

Algorithm A.2: Kernelized Least Angle Regression

```
Input:- K,Y
Output: $\alpha$ .
the initial coefficients $\alpha_0 = 0$;
the initial regression $\mu_0 = 0$;
the initial residual $r_0 = y - \mu_0$;
the initial active feature set $\mathcal{A} = \varnothing$;
the initial candidate feature set $\mathcal{C} = \{i : \exists M_i\}$;
$i \leftarrow 1$;
while $\|s\| < \varepsilon$ do
    calculate the correlations with the current residual $\mathbf{c} = M'(y - \mu_i)$;
    select the feature with maximal correlation $\mathcal{A} = \mathcal{A} \cup \text{argmax}_j\, \mathbf{c}^{j \in \mathcal{C}}$;
    calculate the equiangular u of the current active features;
    remove the collinear kernel features $\mathcal{R}$ and the active features $\mathcal{A}$ from the candidate
    feature set $\mathcal{C} = \mathcal{C} - \mathcal{R} - \mathcal{A}$;
    $i \leftarrow i + 1$;
    calculate the current regression $\mu_i$ and update the coefficients $\alpha_i$;
```

sign of the correlation of the kernel feature M^j with the residual. The correlations of the active features with the residual are the same as the maximum of c, i.e., $\mathcal{A} = \{j : |\mathbf{c}^j| = C\}$, where $C = max(|\mathbf{c}|)$.

Since the active features have the same maximal correlation with the current residual, the new regression is performed on the equiangular vector of the active features. Let $\mathcal{G}_{\mathcal{A}} = K'_{\mathcal{A}} K_{\mathcal{A}}$ and $A_{\mathcal{A}} = (\mathbf{1}'_{\mathcal{A}}\mathcal{G}^{-1}_{\mathcal{A}}\mathbf{1}_{\mathcal{A}})^{-\frac{1}{2}}$, where $\mathbf{1}_{\mathcal{A}}$ is a $|\mathcal{A}|$ long vector of 1's. Therefore, the equiangular vector of the current active features is $\mathbf{u}_{\mathcal{A}}$ in Eq. A.25, where $\mathbf{w}_{\mathcal{A}} = A_{\mathcal{A}}\mathcal{G}^{-1}_{\mathcal{A}}\mathbf{1}_{\mathcal{A}}$ is the coefficients of the active features in the equiangular vector. The vector is actually equiangular as can be verified that all the inner products of the active features with the equiangular vector are equal, i.e., $K'_{\mathcal{A}}\mathbf{u}_{\mathcal{A}} = A_{\mathcal{A}}\mathbf{1}_{\mathcal{A}}$. In addition, the inner products of all the kernel features with the equiangular vector is $\mathbf{a} = X'\mathbf{u}_{\mathcal{A}}$.

$$\mathbf{u}_{\mathcal{A}} = M_{\mathcal{A}}\mathbf{w}_{\mathcal{A}} = M_{\mathcal{A}}A_{\mathcal{A}}\mathcal{G}^{-1}_{\mathcal{A}}\mathbf{1}_{\mathcal{A}} \tag{A.25}$$

With the current regression μ and the equiangular vector $\mathbf{u}$, the next regression $\mu_{\mathcal{A}+}$ is built up as $\mu_{\mathcal{A}+} = \mu_{\mathcal{A}} + \gamma_{\mathcal{A}}\mathbf{u}_{\mathcal{A}}$, where γ is the scale of the equiangular vector used in the regression. The scale should be computed such that the maximal correlation of the candidate kernel features with the residual in the next iteration is equal to the correlation between the residual and the active kernel features, i.e., $max(M'_{\mathcal{A}}(y-\mu_{\mathcal{A}^+})) = max(M'_{\mathcal{C}}(y-\mu_{\mathcal{A}^+}))$. Since a correlation is considered with its absolute value, the maximal correlation occurs in either of its positive or negative direction. Equating the correlations of the active features and the correlations of the candidate features, KLARS obtains the γ on the current equiangular vector $\mathbf{u}$ as Eq. A.26, where $min\{\cdots\}_+$ means the minimum is taken over only the positive elements.

$$\gamma_{\mathcal{A}} = min_{j\in\mathcal{C}}\left\{\frac{C-c^j}{A_{\mathcal{A}}-a^j}, \frac{C+c^j}{A_{\mathcal{A}}+a^j}\right\}_+ \tag{A.26}$$

To ensure the active kernel features are linearly independent, the redundant kernel features must be removed from the candidate kernel features $M_{\mathcal{C}}$ in advance. Eq. A.26 not only determines the scale of the equiangular vector, but the kernel feature to be selected in the next iteration, i.e., the feature of the minimum of Eq. A.26. In the augmented kernel matrix, it is possible that a certain candidate kernel feature is identical to an existing active feature or the current equiangular vector. Such identical features would be selected in the next iteration if they also induce the γ in Eq. A.26 or they have the same maximal correlation as the active kernel features. Eq. A.27 determines the set of the indices I of the kernel features which may be selected in the active kernel features in the next iteration, where $\{j : |c^j| = C\}$ corresponds to the candidate kernel features which are identical to the existing active kernel features or the equiangular vector. To check if the kernel features in I to be added are linearly dependent with the existing active kernel features, they are regressed with the existing active kernel features. If the norm of the residual of such a kernel feature in I is smaller than a predefined threshold, it is marked as a redundant kernel feature in $X_{\mathcal{R}}$. All the linearly dependent kernel features $X_{\mathcal{R}}$ are then removed from the candidate kernel feature set, namely $X_{\mathcal{C}} = X_{\mathcal{C}} - X_{\mathcal{R}}$.

$$I = \operatorname*{argmin}_{j\in \mathcal{I}^c} \left\{ \frac{C-c^j}{A_{\mathcal{A}} - a^j}, \frac{C+c^j}{A_{\mathcal{A}} + a^j} \right\}_{>0} \cup \{ j : |c^j| == C \} \tag{A.27}$$

The solution obtained at iteration j is the accumulated sum of all the coefficients in the previous iterations up to j as shown in Eq. A.28, where $\alpha_j = (\alpha_j^i)_{i=1}^{mN}$ is the jth solution consisting of the coefficients of all the kernel features. Through the iterations in Algorithm A.2, SKFM builds up a series of the solutions of the learning coefficients and thus forms a solution path.

$$\alpha_j = \sum_{k=1}^{j} \gamma_{\mathcal{A}_k} w_{\mathcal{A}_k}, j = 1 \cdots q \tag{A.28}$$

As mentioned in Section A.2, there are two significant advantages of the solution path. First, the solution path can be computed by KLARS very efficiently. It is noted in [32] that when the dimensions are far more than the sample size, such as the case of the $N \times mN$ augmented kernel matrix, the computation cost is only $O(N^3)$, which is significantly smaller than the cost of the OLS solution w.r.t. the augmented kernel matrix, i.e., $O(m^3N^3)$. Therefore, with much less computation than that required for a single complete solution, KLARS obtains a series of solutions.

Second, the solution path is piecewise linear w.r.t. the regularization parameter t in Eq.A.23 and the solution corresponding to other values of t can be linearly interpolated from the existing solution path directly. Eq. A.29 shows the Lagrangian form of the objective function in Eq. A.23, where λ is the Lagrangian multiplier. Clearly, the optimal solution of Eq. A.29 (the root of its derivative) is a linear function of t as long as the active kernel feature set remains intact. Therefore, after the joints connecting the pieces of the solution path where the active kernel feature set changes are determined, the complete solution path can be calculated from the solutions on the joints.

$$\operatorname*{argmin}_{\alpha} (Y - M\alpha)^T (Y - M\alpha) + \lambda(||\alpha||_1 - t) \tag{A.29}$$

SKFM takes the advantage of the solution path to search for an appropriate regularization parameter t. In Algorithm A.1, SKFM uses cross-validation to evaluate the validation error of the solutions of a series of different regularization parameters. After KLARS returns a solution path on the training part in a fold, SKFM interpolates the solutions $S_i = \{s_i^j(t^j)\}_{j=1}^{|T|}$ of a common set of regularization parameters $T = \{t^j\}_{j=1}^{|T|}$. The interpolated solutions are tested on the corresponding validation part and their validation errors are recorded as $E_i = \{e_i^j(t^j)\}_{j=1}^{|T|}$. After all the five folds, the regularization parameter which leads to the least total validation error, i.e., $t = \operatorname{argmin}_{t^j} \sum_{i=1}^{5} e_i^j(t^j)$, is selected as the best regularization parameter, which is used in learning on the whole training dataset.

A.4 Experiment

The experiment tests SKFM on a few real classification problems. The experimental results verify that SKFM not only produces comparable results to SVM on the testing datasets, but it also identifies the important features relevant to the target class label. The better and more easily interpretable results of SKFM may help to enhance the understanding the mechanisms of biological organisms.

A.4.1 Diabetes Classification

The classification problem on diabetes is to predict whether the patient have Diabetic Nephropathy (DN), a kidney disease developed from diabetes. The diabetes dataset consists of 1386 records of the diabetes patients. A patient record is composed of the clinical measurements, the Single-Nucleotide Polymorphism (SNP) information and the labels of whether the patient has DN. There are 99 attributes, among which 23 attributes are the clinical measurements and 76 attributes are the SNPs. There are two class labels: the first label is an earlier DN diagnosis which can be either positive or negative, and the second label is a combination of an earlier diagnosis and a later diagnosis. The second label has four possible values as follows:

1. The two diagnosis are non-DN.
2. The earlier diagnosis is DN (and so is the later diagnosis).
3. The earlier diagnosis is non-DN, but the later diagnosis is DN.
4. The later diagnosis is unavailable.

The experiment removes the second class label and uses the first class label as the target, since almost all the patient records have the available value in the first class label. However, the first class label is an inaccurate measure of the medical condition of the patient, since a non-DN patient in the first diagnosis may become DN in the second diagnosis even though the SNP information of the patient remain the same. Therefore, the first class label in the original dataset is cleansed and three new datasets are obtained for the experiment as follows:

1. The patient records whose first class labels are unavailable are removed from the original dataset, and the rest of the patient records then become the first dataset.
2. The patient records which are non-DN in the first diagnosis and DN in the second diagnosis are removed from the first dataset, and the rest of the patient records then become the second dataset.
3. In the first dataset, the first class labels of the patient records who are non-DN in the first diagnosis but change to DN in the second diagnosis are also considered to be DN, and all the patient records then become the third dataset.

SKFM and LibSVM are tested on the three datasets. LibSVM [24] is a popular implementation of SVM, which uses cross-validation to choose the learning parameters. Each dataset is separated into training and testing sets using 5-fold partition,

	Diabetes 1				Diabetes 2				Diabetes 3			
	Accuracy %		F-score %		Accuracy %		F-score %		Accuracy %		F-score %	
fold	SKFM	SVM	SKFM	SVM	SKFM	SVM	SKFM	SVM	SKFM	SVM	SKFM	SVM
1	86.6	86.6	49.3	49.3	86.9	84.0	61.0	55.2	79.0	76.8	61.3	57.3
2	77.9	80.8	79.6	43.0	77.4	78.7	51.3	43.5	71.7	68.5	55.7	45.3
3	80.0	79.7	44.4	36.4	79.9	81.6	52.4	56.3	72.8	72.8	58.6	52.2
4	89.9	89.1	46.2	40.0	88.1	87.7	49.1	51.6	83.3	80.8	55.8	50.5
5	92.4	92.4	43.2	36.4	94.4	92.7	53.3	43.8	87.8	86.3	50.0	44.1
AVG	85.4	85.7	45.0	41.0	85.3	84.9	53.4	50.1	78.9	77.0	56.3	49.9
S.D.	6.2	5.4	2.9	5.4	6.8	5.5	4.5	6.1	6.8	6.9	4.2	5.4

Table A.1: The comparison of the results of SKFM and SVM on the three diabetes datasets. Each dataset is partitioned using 5-fold cross-validation. The individual performance on each partition and the average performance on all the partitions are included

and thus the algorithms are executed five times on each dataset. Both SKFM and SVM use the gaussian kernel in Eq. A.7, whose parameter σ is fixed in SKFM but needs to be tuned in LibSVM using cross-validation. Table A.1 compares the results of SKFM and LibSVM on the three datasets, where the total accuracies and the positive class $F-score$ are included. In medical classification problems, the accuracy on the positive class is obviously more important than the accuracy on the control class. In addition, the diabetes datasets are unbalanced with only less than 20% of the patient records being DN, and thus the accuracy on the positive cases is likely to be traded off with the accuracy on the negative cases, Eq. A.30 defines the positive class $F-score$, where the operator $|\cdot|$ is the cardinality of the set.

$$\begin{aligned} Precision &= \frac{|true\ positives|}{|true\ positives \cup false\ positives|} \\ Recall &= \frac{|true\ positives|}{|true\ positives \cup false\ negatives|} \\ F-score &= 2 \times \frac{Precision * Recall}{Precision + Recall} \end{aligned} \tag{A.30}$$

The average accuracies of SKFM are quite competitive with those of SVM. Except for the first dataset, the average accuracies of SKFM are better than those of SVM. The advantages of the $F-scores$ of SKFM over those of SVM are even more obvious, where the average $F-scores$ of SKFM are a few percentages better.

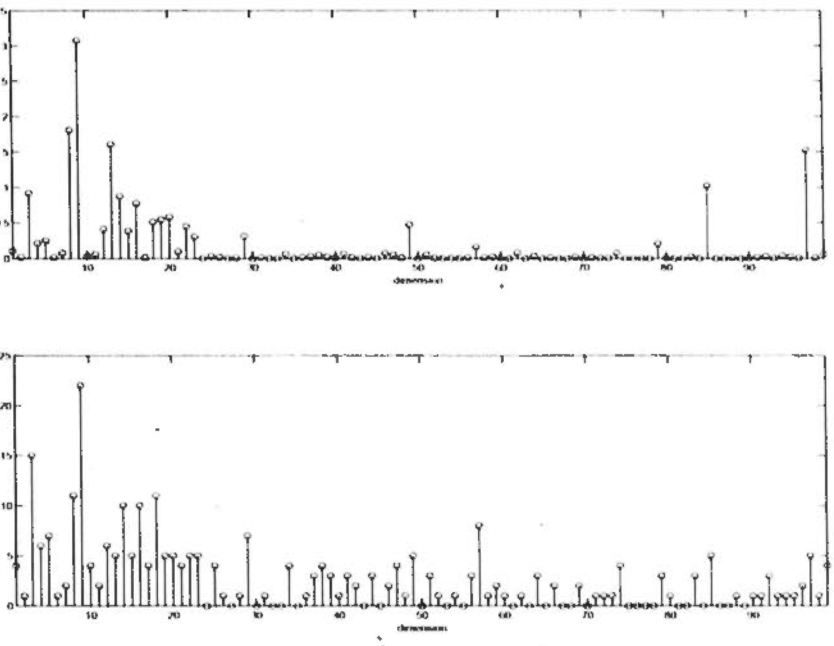


Figure A.3: The weights of the attributes and the counts of the attributes selected in the classification of the first diabetes dataset

This can be explained by the reason that SKFM removes the irrelevant feature w.r.t. the target, and thus the resulted model may be more consistent with the unknown pattern of diabetes. For both algorithms, the average accuracies and $F-scores$ on the first and second datasets are more or less the same. However, the $F-scores$ of SKFM and SVM are enhanced after the ambiguous cases, i.e., those of non-DN in the first label while DN in the second label, are removed from the first dataset. This means that the ambiguous patient cases are indeed misleading in the classification. However, if their first class label are manually set to DN as in the third dataset, the average accuracies of both algorithms drop significantly. This is possibly because that their clinical attributes might change a lot in the second diagnosis, and thus those in the first diagnosis are outdated.

The resulted models of SKFM is also able to show which attributes are used and how important they are. Fig. A.3 shows the selected features of SKFM on the first dataset of diabetes, and the results on the other two datasets are similar.

Note according to the structure of the augmented kernel, a resulted model may contain multiple kernel features corresponding to the same original attributes. On the other hand, it may also contain multiple kernel features parameterized by the same data point. Two plots are shown in Fig. A.3. The first plot shows the sums of the kernel feature coefficients corresponding to the original attributes. The second plot show the counts of the selected kernel features corresponding to the original attributes. Most of the 23 clinical attributes are included in the model. This is expected, since the DN patient should show some clinical pattern. Most of the SNPs are unimportant in the DN classification except for a few ones, such as VGBi, VDR2i, SELP1i, LTAf and ICAM1i, which may be genetically related to DN.

A.4.2 Hepatitis B Virus Classification

Hepatitis Virus B (HBV) classification is to predict whether a patient has HBV based on a segment of his/her DNA sequence. The DNA sequence is a string of four possible nucleotide bases, i.e., $\{A, C, G, T\}$. Due to the measurement noise, the nucleotides on some positions are ambiguous, as they can be one of two or even three nucleotide bases. The dataset collected contains 88 DNA sequences, and each sequence is 3214 bp long. The problem is difficult as it is very high-dimensional compared to the number of the samples.

SKFM uses the string kernel for this problem. It puts a 100 bp window on a DNA sequence, and slides the window from position 1 till position $3115 = 3214 - 100 + 1$. The subsequence inside a window is treated as a dimension of the data sample, and so a sequence has 3115 dimensions. The kernel between two subsequences is the product of the Gaussian kernels on the 100 nucleotides. A Gaussian kernel on a nucleotide is calculated with the 1-out-of-4 encoding of the nucleotide as the kernel argument.

As the diabetes classification, SKFM and SVM are tested on the HBV dataset using 5-fold partition. The accuracy and the positive class $F-score$ on the testing

	Accuracy %		F-score	
	SKFM	SVM	SKFM	SVM
1	70.6	76.5	61.5	66.7
2	58.8	52.9	53.3	50.0
3	52.9	41.2	42.9	16.7
4	70.6	70.6	61.5	61.5
5	65.0	60.0	53.3	50.0
AVG	63.6	60.2	54.5	49.0
S.D.	7.7	14.0	7.7	19.5

Table A.2: The comparison of the results of SKFM and SVM on the HBV dataset. The dataset is separated into training and testing set using 5-fold partition. The individual performance on each partition and the average performance on all the partitions are included

set are reported in Table A.2. Both the average accuracy and the $F-score$ of SKFM are better than those of SVM, and their standard deviations are smaller. SKFM is also able to identify the genetic information relevant to HBV. Fig. A.4 shows the weights and counts of the subsequences selected in SKFM. Among the 3205 subsequences, only a small portion of the subsequences are selected, and most of the selected subsequences have small weights and counts in the regression function. Therefore, it can be conjectured that the subsequences of large weights and counts may contain the HBV-related genetic information.

A.4.3 Colon Cancer Classification

Colon cancer classification is to distinguish cancer from normal tissue using microarray data. The data contains 22 normal and 40 cancer tissues, and each tissue contains 2000 features. The dataset is preprocessed with the following steps: taking the log of all the values, standardizing the sample vectors and then the feature vectors, passing the values through $tanh$ function to diminish the effect of outliers. The datasets are prepared with 100 random partitions, and each partition contains 50 training samples and 12 testing samples. Since it is a high-dimensional problem with real-valued features, it is better to use linear regression to mitigate the curse of

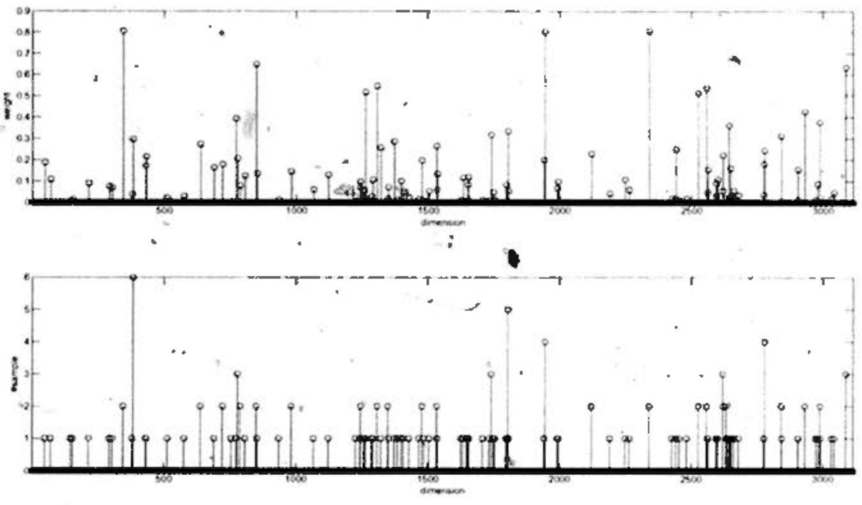


Figure A.4: The weights of the subsequences and the counts of the subsequences selected in the classification of the HBV dataset

	ARD	RFE	Fisher	SKFM
#error	2.90	2.84	2.68	2.25
#feature	8.55	4.25	14.41	10.02

Table A.3: The results of ARD, RFE with SVM, Fisher score with SVM and SKFM on the Colon cancer classification. The number of wrong predictions and the number of selected features are included

dimensionality, and so the linear dot product is used as the kernel function.

The results of three comparing algorithms are quoted from [75]. Automatic Relevance Determination (ARD) [75] uses the Bayesian analysis to estimate the coefficients of a linear regression function, and the coefficient of an irrelevance features vanishes when its prior variance approaches zero. Recursive Feature Elimination (RFE) [44] trains a series of SVMs while features are successively eliminated during training. In addition, [40] uses Fisher score to rank and select features prior to training with SVMs.

Table A.3 shows the average results of ARD, RFE with SVM, Fisher score with SVM and SKFM on the colon dataset. The number of wrong predictions among the 12 testing samples and the number of selected features are included. Clearly, SKFM compares favorably to other algorithms on the number of wrong predictions, and it

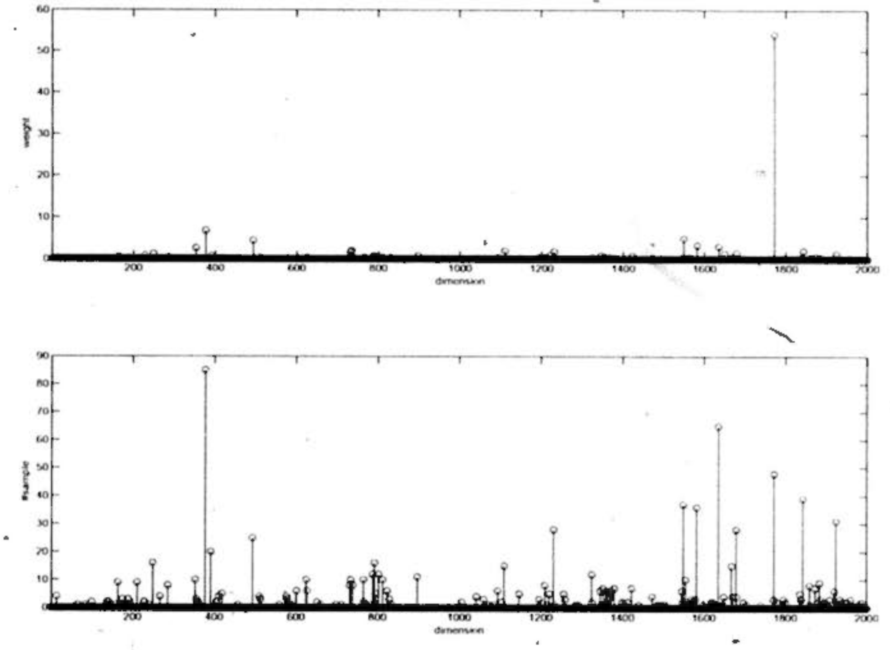


Figure A.5: The weights of the subsequences and the counts of the subsequences selected in the classification of the colon dataset

uses a modest number of features in the learning. Fig. A.5 shows the coefficients and the counts of the 2000 features in the learning function among the 100 random partitions. Surprisingly, there is a single feature which has a significantly larger coefficient than the others in the first plot. There is also a single feature which is used in almost all the partitions. It seems that these two features may have some medical meaning to the colon cancer.

A.4.4 Splice Site Classification

The splice site classification problem is to classify the DNA sequences containing acceptor splice sites from those of no splice sites. After a DNA sequence is transcribed, splicing removes the introns from the RNA and joins the remaining exons into the messenger RNA. Splicing takes place on the sites which separate the introns and the exons. The task of SKFM for this problem is not only to classify the DNA sequences successfully, but also tells the sites involved in the splicing process.

The *C. elegans* dataset [98] consists of 262421 sequences, and only 15507 of them contain a true splice site each. Following the website of the multiple kernel learning (MKL) [98], the bootstrapping test is adopted in the experiment. 100 bootstrap datasets are sampled randomly from the original complete dataset. A bootstrap dataset contains 1500 sequences for training and 10000 sequences for testing.

The average Area Under Curve (AUC) of MKL is 97.5%. If SKFM uses a single nucleotide as a feature, the average AUC of the resulted classifier is 97.3%. If SKFM uses the sub-sequence in a 10 bp sliding window as a feature, the average AUC of the resulted classifier increases to 97.9%. The difference between the two kinds of features and the resulted performances of SKFM shows that there may be a joint effect of the nucleotides in the splicing process.

Fig. A.6 shows the weights and the counts of the positions selected by SKFM. SKFM clearly identifies a few positions which seem to be vital to identify the splicing sequences. Compared to Fig. A.7, such positions also have high relative entropies in the splicing sequences over the non-splicing sequences. This is consistent with the biological knowledge of the splicing process. The nucleotides around the splicing site are relatively conserved so that the splicing site can be recognized by the spliceosome.

A.5 Discussion

This appendix proposes and implements a new kernel learning method, i.e., Sparse Kernel Feature Machine (SKFM). SKFM performs (kernel) feature selection and kernel learning simultaneously. SKFM generates a solution path w.r.t. the regularization parameter, which enables the automatic selection of the appropriate regularization parameter. Instead of using the kernel functions of all the dimensions, SKFM equips a kernel function and forms a kernel feature for each original feature. An augmented kernel matrix is constructed by concatenating all the kernel matrices

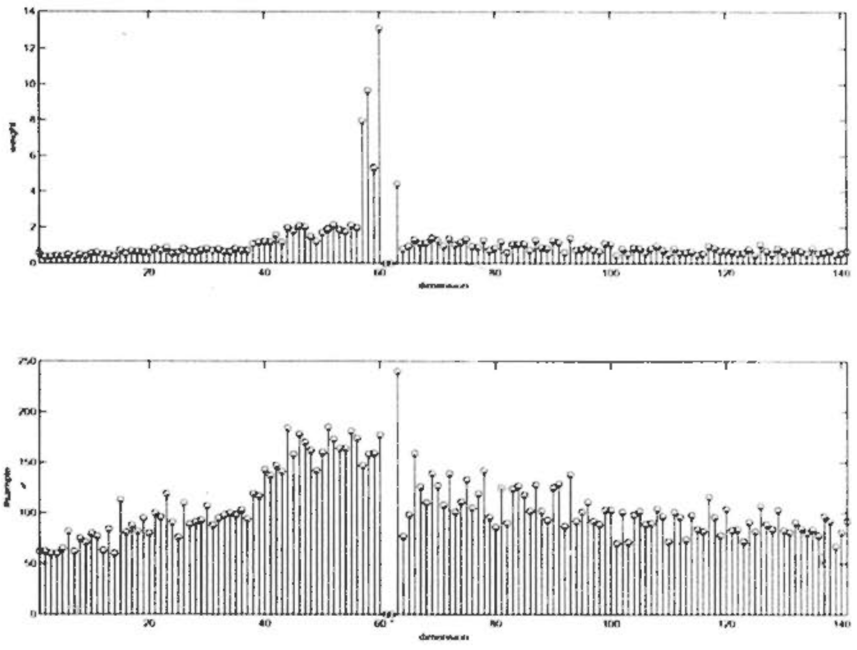


Figure A.6: The weights of the positions and the counts of the positions selected in the classification of the *C. elegans* dataset

of the individual features. From the selected kernel features in the augmented kernel matrix, SKFM can infer the relevant features and the relevant values on those features to the class target.

Kernelized Least Angle Regression (KLARS) is applied on the augmented kernel matrix to perform step-wise linear regression in the feature space by adding features iteratively. Collinear kernel features are detected and removed from the set of the kernel features in the iterations. KLARS forms the solution path of the regression coefficients of the kernel features which is piece-wise linear in the regularization parameter. From the solution path, SKFM is able to interpolate the solutions under different regularization parameters and chooses the best regularization parameter in cross-validation. Since the kernel features are added in the solution path in the order of their importance, a good regularization parameter may keep the important kernel features and remove the irrelevant kernel features.

In the experiments, SKFM has been tested on four real medical classification

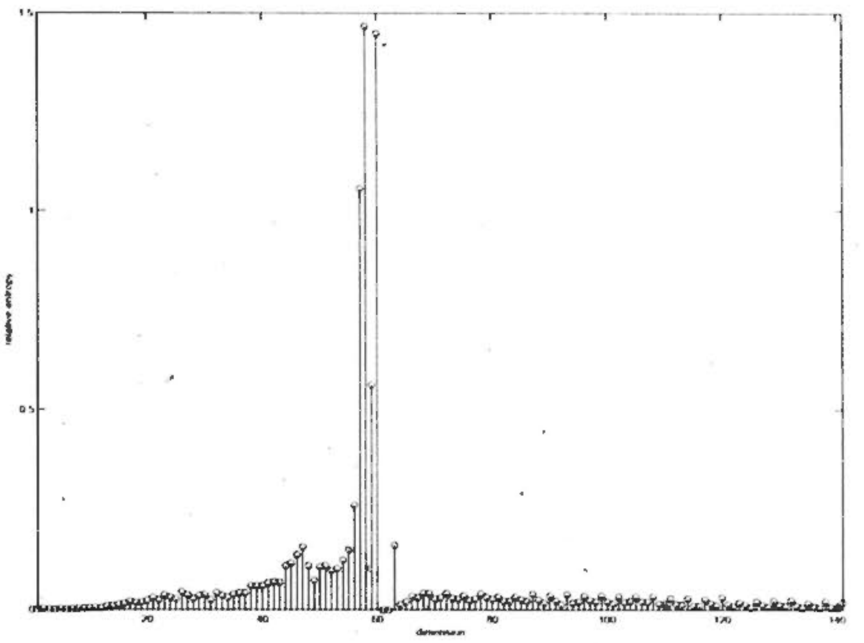


Figure A.7: The relative entropies on the positions of the splicing sequences over the non-splicing sequences in the classification of the *C. elegans* dataset

problems, i.e., Diabetes, Hepatitis B Virus, Colon Cancer and *C. elegans*. The results verify that SKFM not only outperform Support Vector Machine (SVM), but it also point out the most important features, including the clinical and genetic information, leading to the diseases.