Evolutionary Computation for Statistical Pattern Recognition

Wang Xiao

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirements for the degree of
Doctor of Philosophy

2006

Acknowledgement

I would like to thank my supervisor, Dr. Wang Han, for his guidance, support and encouragement throughout the duration of this research. I am also indebted to all the colleagues in Intelligent Robotics Laboratory at the School of Electrical and Electronic Engineering (EEE), Nanyang Technological University. Without their help, this research would have not been possible.

Summary

Evolutionary computation as a general problem-solving technique has been extensively applied in statistical pattern recognition. Typically, evolutionary algorithms are developed to solve complex optimization and search problems involved in different folds of designing a recognition system, e.g. feature extraction, supervised classification and clustering. These problems are often characterized by high-dimensional search spaces with convoluted landscape, noisy data, and little information about the objective functions. Traditional optimization methods are not efficient in dealing with them and evolutionary algorithms are therefore introduced.

In the thesis a brief introduction to evolutionary computation is first presented. The synergetic combination of the two fields of evolutionary computation and statistical pattern recognition are then discussed. The state of the art is surveyed by analyzing those representative works. In our own effort of designing evolutionary systems for pattern recognition, two important topics are selected: ensemble learning and Markov random field (MRF) modelling. First, the technique of constructing classifier ensembles by manipulating the training examples is analyzed. It appears that the technique is essentially searching for appropriate weights associated with the examples. Thereby we adopt the idea of genetic search and develop a novel evolutionary learning algorithm. In contrast with conventional designs, here all of the chromosomes throughout evolu-

tion will be exploited. Secondly, energy function minimization in MRF modelling is a long-time challenging problem. While evolutionary optimization seems promising, the problem structure makes an immediate application of genetic algorithms inappropriate. A novel evolutionary algorithm is developed in which the evolution is not carried out by genetic operators, but by building and sampling from probabilistic models. In this way the domain-specific knowledge of contextual constraints is readily exploited. The scheme also leads to a convenient approach to incorporate local search into the evolutionary search. The two algorithms proposed are testified in comparative experiments with those traditional methods in their respective problem domains. The potentials of evolutionary computation in solving statistical pattern recognition problems are vividly demonstrated.

KEY WORDS: statistical pattern recognition, evolutionary computation, genetic algorithms, ensemble learning, evolutionary ensembles, Markov random field, estimation of distribution algorithms.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Pattern recognition is the research of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of the patterns [4, 5]. Among the various frameworks in which pattern recognition has been formulated, the statistical approach is intensively studied and used in practice. In this approach, each pattern is represented in terms of several features, or attributes, and is viewed as a point in a high dimensional feature space. Given a set of training patterns from each class, the objective is to establish decision boundaries in the feature space which separate patterns belonging to different classes.

Interest in *statistical pattern recognition* has been renewed recently due to emerging applications including document classification (efficiently searching text documents), bioinformatics (DNA/Protein sequence analysis), biometrics (person identification based on physical attributes such as face and fingerprints), multimedia database retrieval and

financial forecasting. A common characteristic of a number of these applications is that the available features are typically noisy and in the number of thousands. While similar feature vectors could behave oppositely, vectors far from each other may belong to the same category. To locate decision boundaries in such high dimensional convoluted feature spaces is a sophisticated optimization procedure. It needs careful parameter selection and elaborate algorithm design in order to perform efficient search. The whole process is not only computationally intensive, but also leads to a possibility of losing the optimal solutions.

Evolutionary computation (EC) comprises a class of probabilistic search and optimization algorithms guided by the principles of organic evolution. They have proven efficient, adaptive and robust, producing near-optimal solutions and having a large amount of implicit parallelism. They are not easily cheated by local optima and need little information about the objective function. Therefore, evolutionary computation appears to be a reasonable and appropriate choice for solving problems in statistical pattern recognition. Novel ideas and algorithms on the integration of these two fields have been continuously proposed and developed [6–8].

1.2 Objectives

The work presented in the thesis is to explore new ways in which the integration of evolutionary computation and statistical pattern recognition can be made in order to develop efficient recognition systems. To achieve this objective, first of all, we intend to perform a survey of EC techniques for statistical pattern recognition applications. Over the last two decades researchers in evolutionary computation, computer vision, image processing, machine learning and pattern recognition have investigated a number

of issues related to the topic. However, we are unaware of any updated overviews. We believe that a critical survey will give more insight and hence help clarifying the state and direction of relevant research.

We find ensemble learning as a promising application. There have been successful attempts such as evolutionary artificial neural networks [9, 10], but they are based on injecting randomness into the learning algorithm, e.g. random connection weights and architectures. In our opinion, those methods based on manipulating training examples need also careful exploration. Another problem where we try to adopt evolutionary search is the minimization of a posterior energy in Markov random field modelling. The energy function is usually of very high dimensionality and bearing complicated landscape. For both of our cases we expect that the global and adaptive searching capability of evolutionary algorithms will do much help.

1.3 Main Contributions

The thesis mainly constitutes three aspects of contributions: (1) we give a critical survey about applying evolutionary computation for statistical pattern recognition; (2) we develop an evolutionary algorithm for ensemble learning which works by seeking appropriate weights and assigning them to training examples during learning; and (3) we address the topic of evolutionary optimization with Markov random field prior. More specifically,

- Statistical pattern recognition is mathematically and concisely modelled as multi-stage problems of optimizing composite functions. The reasons why these problems are complex and difficult for traditional optimization algorithms are discussed in

terms of those intrinsic properties. The state of art of applying evolutionary computation techniques is surveyed. Particular attention is paid to ensemble learning and MRF modelling. At each point, the problem is understood from the viewpoint of optimization and the designs of evolutionary algorithm are emphasized.

- An ensemble learning algorithm is developed to build a group of classifiers each of which is trained based on a particular weighting over the training examples (a weighting is a set of weights associated with the examples). The task concerns search in a tremendous weighting space. In this view we propose to use a genetic algorithm (GA). It performs a global search for appropriate weightings. The difference from a traditional GA is that all the weightings throughout evolution will be exploited to form the final ensemble, not just the best weighting. The proposed algorithm compares in a desirable manner in the experiments with its conventional counterparts.

- A novel evolutionary algorithm is presented in which the constituent variables of a solution are modelled by a Markov random field (MRF). A population of potential solutions is maintained at every generation and for each solution a fitness value is calculated. The evolution, however, is not obtained through genetic recombination. Instead, each variable in a solution will be updated by sampling from its estimated distribution. According to the MRF prior, local exploitation is encoded in the conditional probabilities. For evolutionary exploration we estimate the probabilities as fitness-weighted statistics. These two kinds of search are combined smoothly in our algorithm. The algorithm is observed of remarkable performance in the experiments.

1.4 Thesis Outline

Chapter 2 first gives a brief introduction to evolutionary computation. An outline is formulated in which genetic algorithms as an important type are given much attention. Then we include a description of statistical pattern recognition in the viewpoint of function optimization. Some properties of the optimization involved are summarized, which are generally the reasons why evolutionary computation is borrowed. The application of evolutionary computation in statistical pattern recognition is surveyed from two aspects: feature selection/extraction and supervised/unsupervised learning. Those representative works are selected and we discuss in detail how evolutionary algorithms are designed and adapted in solving different problems. After that we focus on two particular problems: ensemble learning and MRF modelling. The strengths and weaknesses of classical algorithms are analyzed and the analysis is used to motivate the work to be presented in chapter 3 and 4. From this chapter the readers will have an overview of the state of the art.

Chapter 3 is about the evolutionary algorithm developed for ensemble learning. We first discuss some similar works in designing evolutionary neural network ensembles. Our algorithm is then presented from several aspects, e.g. the general structure, the genetic operators of selection, crossover and mutation, and the method of training with weighted examples. We suggest to incorporate those chromosomes in early generations into the final ensemble. We also employ fitness sharing to encourage diversity in the ensemble. The algorithm is compared with Bagging and AdaBoost on a number of UCI benchmark data sets and a real application of face detection. Robust and consistently accurate classification is experienced.

Chapter 4 talks about the topic of evolutionary optimization with Markov random field

prior. We first introduce the estimation of distribution algorithms (EDAs) which extend the genetic algorithms by replacing genetic recombination with model estimation and sampling. EDAs directly inspire us to develop a novel evolutionary algorithm in the context of MRF. The algorithm is presented in details including the chromosomes, fitness evaluation, and the population updating scheme. An approach to hybridize the evolutionary search with local search is also described. In the experiments our algorithm is compared with two representative algorithms on noisy and textured image segmentation. Competitive results are reported. Limitations of the algorithm are also analyzed.

Chapter 5 concludes the thesis. Our work is summarized. More importantly, we mention a few directions for future work, including the incorporation of negative correlation learning in our current framework of ensemble learning, multi-objective evolutionary algorithms for ensemble learning, etc. We believe the exploration of these directions will bring improvement to the proposed algorithms as well as development of new algorithms for statistical pattern recognition problems.

Chapter 2

Literature Review

2.1 Evolutionary Computation in Concept

Evolutionary computation is a biologically inspired problem-solving technique. It works on the neo-Darwinian paradigm of natural evolution which is a widely accepted collection of evolution theories. These arguments assert that the history of the vast majority of life can be fully accounted for by physical processes operating on and within populations and species [11]. These processes include reproduction, mutation, competition and selection. Reproduction is an obvious property of extant species. It is accomplished through the transfer of an individual's genetic program (either asexually or sexually) to progeny. Species have such great reproductive potential that their population size would increase at an exponential rate if all individuals of the species were to reproduce successfully. Competition and selection inevitably become the consequences of any expanding population constrained to a finite arena. Mutation, in a positively entropic universe, is guaranteed in that replication errors during information transfer will necessarily occur. Evolution is then the result of these fundamental interacting processes.

Individuals and species can be viewed as a duality of their genetic program, the *genotype*, and their expressed behavioral traits, the *phenotype*. While reproduction and mutation are carried out over the genotypes, selection acts directly on the phenotypes. The phenotype varies as a complex, nonlinear function of the interaction between underlying genetic structures and current environmental conditions. In Wright's metaphor of "adaptive landscape" [12], a population of genotypes maps to respective phenotypes, which are then mapped onto the topography. Each peak corresponds to a fitness-optimized phenotype, and thus one or more optimized genotypes. Evolution probabilistically proceeds up the slopes of the topography toward peaks as selection culls inappropriate phenotypic variants. In an easier word, a single individual of a population is affected by other individuals of the population (e.g., by food competition, predators, and mate), as well as by the environment (e.g., by food supply and climate). The better it performs under these conditions the greater is the chance for the individual to live for a longer while and generate offspring, which in turn inherit the parental genotype. The nondeterministic nature of reproduction results in an unceasing production of novel genetic information and therefore the creation of differing offspring, probably more fitting. Generation by generation, this course leads to the emergence of complex and well-adapted organisms.

The basic idea of evolutionary computation is to simulate natural evolution on a computer. The fundamental works are from Holland [13] on Genetic Algorithms, Rechenberg [14] and Schwefel [15] on Evolution Strategies and Fogel [16] on Evolutionary Programming. The research has resulted in the development of stochastic optimization algorithms which prove robust and efficient in domains as diverse as engineering, natural sciences and economics. In spite of the numerous designs, these algorithms can be formulated in a general structure (Fig. 2.1, we borrow it from [17] and make some modifications).

1. Set $t = 0$; initialize $Popu(t)$.
2. Evaluate $Popu(t)$.
3. **while not** terminate **do**
   (a) Set $Popu'(t) = \text{selection}[Popu(t)]$.
   (b) Set $Popu''(t) = \text{variation}[Popu'(t)]$.
   (c) Evaluate $Popu''(t)$.
   (d) Form the next generation $Popu(t+1)$ by copying from $[Popu''(t) \cup Q]$.
   (e) Set $t = t + 1$.

   end.

Figure 2.1: The general structure of evolutionary algorithms.

In this outline, $Popu(t)$ denotes a population of individuals (potential solutions to the problem) at generation t. All the individuals are evaluated respectively by calculating how well they achieve the goal of the problem. $Popu'(t)$ is a population of solutions selected from $Popu(t)$. Those solutions with above-average evaluation may have multiple copies and those solutions with under-average evaluation may have zero copy. Selection therefore drives the population toward better solutions. An offspring population $Popu''(t)$ is generated by means of variation operators such as recombination and/or mutation from $Popu'(t)$. The newly generated solutions will also be evaluated. Q is a special set of individuals that might be considered for entering into the next generation, e.g., $Q = Popu(t)$ (but $Q = \emptyset$ is also possible). The process of evolution is continued until some predefined termination criterions are satisfied.

2.1.1 Genetic Algorithms

Genetic Algorithms (GAs) [13,18] as an important type of evolutionary algorithms have been receiving wide interest. In the canonical form GAs are implemented as follows:

1. The problem to be addressed is defined and captured in an objective function that indicates the fitness of a potential solution.

2. A population of candidate solutions is randomly initialized subject to certain problem-dependent constraints. Each solution, in Holland's suggestion [13], is coded as a fixed-length string over the alphabet $\{0, 1\}$. This binary string is termed a *chromosome*, with elements being described as *genes* and varying values at specific positions called *alleles*.

3. Each chromosome in the population is decoded into a form appropriate for evaluation and is then assigned a fitness score.

4. Each chromosome is assigned a probability of reproduction, so that its likelihood of being selected is proportional to its fitness relative to the other chromosomes in the population. If the fitness of each chromosome is a strictly positive number to be maximized, selection could be accomplished using *roulette wheel selection* [18].

5. The selected chromosomes generate offspring via the use of genetic operators: crossover and bit mutation. Crossover is applied to two chromosomes (parents) and creates two new chromosomes (offspring) by selecting a random position along the coding and splicing the section that appears before the selected position in the first string with the section that appears after the selected position in the second string, and vice versa. Note that crossover is usually applied by a certain probability, typically $p_c = 0.6$ [19] or $p_c \in [0.75, 0.95]$ [20]. That means not all pairs of parents will reproduce. Mutation is a subsequent operation which is carried out on each of the newly generated chromosomes. It is the occasional random alteration of the value of a string position, changing a 1 to a 0 or vice versa. Typically each bit is assigned a small probability of mutation, $p_m = 0.001$ [19] or $p_m \in [0.005, 0.01]$ [20].

6. The process is terminated if a suitable solution has been found or if the available computing time has expired. Otherwise the process proceeds to step (2), where the new chromosomes are scored and the procedure iterates.

An immediate observation shows that the above-summarized procedure has a rather simple control structure. Nevertheless, it has demonstrated remarkable performance in solving function optimization problems [18]. In order for genetic algorithms to surpass traditional optimization algorithms in the quest for efficiency and robustness, GAs differ in some very fundamental ways:

- GAs work with a coding of the solution set, not the solutions themselves. They exploit similarities among high-performance strings. Because GAs operate at the coding level, they are difficult to fool even when the function is difficult for traditional schemes.
- GAs search from a population of points, not a single point. In this way, GAs find safety in escaping from local optima. By maintaining a population of well-adapted sample points, the probability of reaching a false peak is reduced.
- GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge. Traditional methods rely heavily on such information, and in problems where the necessary information is not available or difficult to obtain, these methods break down. GAs process similarities in the underlying coding together with information ranking the structures according to their survival capability in the current environment. By exploiting such widely available information, GAs may be applied in virtually any search problem.
- GAs use probabilistic transition rules, not deterministic rules. A distinction exists, however, between the randomized operators of genetic algorithms and other

methods that are simple random walks. This may seem unusual, using chance to achieve directed results (the best points), but nature is full of precedent.

In spite of the surprisingly good performance, a canonical GA may not be flexible enough for a practical application, and an engineering insight is always required. Practically, numerous variants on the many issues of GA design have been developed. Here we discuss some of them.

2.1.1.1 The Representation

The strong preference for using binary codings of solutions in a GA is derived from the *schema theory* [13]. A schema is defined by a string of symbols from an alphabet A; it has some fixed components while others are free to vary. For example, consider $A = \{0, 1\}$ and a wild card symbol, #, that matches any symbol from A. The schema $\{01\#\#\}$ may signify $\{0100\}$, $\{0101\}$, $\{0110\}$ and $\{0111\}$. In the opposite way if the string $\{0100\}$ is evaluated to have a fitness, then partial information is also received about the expected fitness of all possible schemata in which that string resides, e.g., $\{01\#\#\}$, $\{0\#\#\#\}$, $\{\#1\#\#\}$, $\{\#\#00\}$. This characteristic is termed *implicit parallelism*, as it is through a single sample that information is gained with respect to many schemata. Holland speculated that it would be beneficial to maximize the number of schemata being sampled, thus providing maximum implicit parallelism, and proved that this is achieved for $|A| = 2$. Binary strings were therefore suggested as a universal representation.

The use of binary representations is not always accepted in the literature, however. The coding function might introduce an additional multimodality, thus making the combined objective function more complex than the original problem. Michalewicz [21] indicates that for real-valued numerical optimization problems, floating-point representations out-

perform binary representations because they are more consistent, more precise, and lead to faster execution. Another problem-oriented chromosome representation is the order-based representation which is particularly useful for those problems where a special sequence of the data is under searching [3]. Many researchers have foregone the bit strings and have achieved reasonable results to difficult problems [22–24]. We also present two-dimensional chromosomes in chapter 4.

2.1.1.2 Selection

Selection in proportion to fitness can be problematic. In a common case the objective function may be intended for minimization that the roulette wheel selection can not be applied directly. Therefore the value of objective function needs be transformed to obtain the fitness. Moreover, scaling of objective function value has become a widely accepted practice to keep appropriate levels of competition throughout simulation. Without scaling, in early generations there is the tendency for a few super individuals to dominate the selection process. In this case the objective function value must be scaled back to prevent takeover of the population by these super strings. Later on, when the population is largely converged, competition among population members is less strong and the simulation tends to wander. In that case the objective function value must be scaled up to accentuate differences between population members to continue rewarding the best performers. In practice, several heuristics have been devised, including the linear scaling, sigma truncation and power law scaling [18].

In addition to the fitness value, the selection mechanism itself needs extra attention. The basic roulette wheel selection method is a stochastic sampling with replacement. It tends to give zero *bias* but potentially inclines to an unlimited *spread* (zero bias is achieved when an individual's probability of selection equals its expected number of

trials; spread is a range of the possible number of trials that an individual may achieve). Stochastic Universal Sampling (SUS) [25] is proposed as a sampling algorithm with minimum spread, zero bias and the time complexity in the order of N (the number of individuals to be selected). Another method that gains popularity is the tournament selection. For an overview of these methods and a characterization of their selective pressure in terms of numerical experiments, the readers may consult [26].

One problem in canonical GA is that the best chromosome in the population may be lost at any generation, and there is no assurance that any gains made up to a given generation will be retained in future generations. This can be overcome by employing the *elitist strategy* [18] which simply always retains a few best chromosomes found so far and let them compete with the newly generated chromosomes to enter the next generation. This technique may increase the speed of domination of a population by a few super chromosomes, but on balance it appears to improve the performance of GAs.

2.1.1.3 Crossover and Mutation

Holland [13] proposed that GAs seek near-optimal performance by at least exponentially increasing the number of well-performing, short (i.e. with small distance between the left-most and right-most defined position), and low-order (i.e., with few specified bits) schemata (so called *building blocks*) throughout generations. This idea has become known as the *building block hypothesis*. The genetic operators, we normally refer to as crossover and mutation, have the ability to generate, promote and juxtapose building blocks. The one-point crossover is suggested because it is expected to maintain building blocks which are associated with above-average performance and not disrupt their linkage. But this has not been clearly demonstrated in the literature. Syswerda [27] conducted function optimization experiments with two-point crossover, uniform crossover

and one-point crossover. Uniform crossover provided generally better solutions with less computational effort.

Crossover provides a main search operator while bit mutation simply serves as a background operator. This philosophical view is reflected by the probabilities commonly assigned to them. But when the strings for crossover are similar, their capacity to generate new building blocks diminishes. Whereas mutation is not a conservative operator, it is capable of generating new building blocks radically. This is especially important during the later generations of evolution. In another point, while crossover performs large-scale exploration, mutation may emphasize on neighborhood exploitation. For example, random mutation has been designed for the real number chromosome [3].

By now we have discussed many issues in designing genetic algorithms. Generally there are no fixed answers for these issues. Most practitioners would prefer specific, problem-related methodologies. In a summary Michalewicz [21] offers:

> It seems that a "natural" representation of a potential solution for a given problem plus a family of applicable "genetic" operators might be quite useful in the approximation of solutions of many problems, and this nature-modeled approach...is a promising direction for problem solving in general.

The opinion is also applicable to other branches of evolutionary computation. Researchers widely agree that the most significant advantage of using evolutionary search lies in the gain of flexibility and adaptability to the task at hand, in combination with robust performance and global search characteristics. Evolutionary computation should be understood as an adaptable concept for problem solving, rather than a collection of related and ready-to-use algorithms. Once a framework of evolutionary system has been developed, it can be incrementally adapted to the problem under consideration, to

changes of the requirements and to modifications of the models.

2.2 Evolutionary Computation for Statistical Pattern Recognition

A general framework to formulate solutions of pattern recognition problems is a statistical one, which recognizes the probabilistic nature both of the information we seek to process, and of the form in which we shall express the results [4]. In statistical pattern recognition, a pattern $\mathbf{o}$ is represented by a set of d features, viewed as a d-dimensional feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$. Let $\mathbb{O}$ be the pattern space, $\mathbb{X}$ be the feature space and $\Omega = \{c_1, c_2, \ldots, c_n\}$ be a set of categories. The pattern recognition problem can be defined as a composite function in the following way:

$$c = C(D(\mathbf{o})) \tag{2.1}$$

where

$$c \in \Omega,\ \mathbf{o} \in \mathbb{O},\ D : \mathbb{O} \rightarrow \mathbb{X},\ C : \mathbb{X} \rightarrow \Omega.$$

In this equation, D represents the feature extraction and selection function, C represents the classification function. The functions D and C both need be learned from some training examples. The overall procedure consists of the transformations, first from the pattern space to the feature space, and then from the feature space to the category space. Finding the optimal transformations (both C and D) therefore constitutes the fundamental problem in statistical pattern recognition.

Usually, a feature vector $\mathbf{x}$ belonging to class c_i is viewed as an observation drawn

randomly from the class-conditional probability function $p(\mathbf{x}|c_i)$. A number of well-known decision rules, including the Bayesian rule and maximum likelihood rule (which can be viewed as a particular case of Bayesian rule), are available to define the decision boundary. The Bayesian rule assigns input feature $\mathbf{x}$ to class c_i for which the conditional risk

$$R(c_i|\mathbf{x}) = \sum_{j=1}^{n} L(c_i, c_j) \cdot P(c_j|\mathbf{x}) \tag{2.2}$$

is minimum, where $L(c_i, c_j)$ is the loss incurred in deciding c_i when the true class is c_j and $P(c_j|\mathbf{x})$ is the posterior probability. In case of $0/1$ loss function, the conditional risk becomes the conditional probability of misclassification [5]. For this choise, Bayesian rule can be simplified as follows (also called the maximum *a posteriori* (MAP) rule): assign input feature $\mathbf{x}$ to class c_i if

$$P(c_i|\mathbf{x}) > P(c_j|\mathbf{x}) \quad \text{for all } j \neq i. \tag{2.3}$$

Depending on the kind of information available about the class-conditional densities, various strategies have been developed to design a classifier. If all of the class-conditional densities are completely specified, we may directly minimize the risk function in (2.2). Unfortunately, the class-conditional densities are usually not known in practice and must be learned from the available training patterns. If the form of the class-conditional densities is known (e.g., multivariate Gaussian) but the parameters of densities (e.g., mean vectors and covariance matrices) are unknown, we have a parameter optimization problem. The optimal parameters are obtained by maximizing or minimizing an objective function of the parameters. If the form of the class-conditional densities is not known, then we operate in a nonparametric mode. In that case, we must either estimate the density function or directly construct the decision boundaries based on the training data.

From the above discussion it is realized that statistical pattern recognition problems, from feature measurement and parameter estimation to classifier training, are multi-staged optimization problems. Search in the spaces of all the possible parameters, density functions or decision boundaries is generally a challenging task. Not surprisingly, we need deal with noise contaminated observations and spaces of tremendous size. For instance, in a feature selection problem of dimension d, the search space is 2^d since every feature can be selected or unselected. The exponential order makes any direct search method impractical. More difficulties come from the objective function. It could be multi-modal having numerous local maxima or minima. In many cases it is non-smooth which is unsuitable for methods utilizing derivatives. The objective function could even be ill-posed. Those solutions that are not optimal may also provide useful information to the problem (this situation is further discussed in chapter 3). All these characteristics appear daunting to traditional optimization methods since most of them work well only with differential, unimodal and low-dimensional objective functions. For instance, gradient strategies need to smooth their world by exploiting the objective function's first-order partial derivatives. The branch-and-bound method is guaranteed to find an optimal solution path if the available heuristic function is admissible. However, with a poor heuristic function, the algorithm deteriorates into blind search. The flaw of the well-known dynamic programming method is that it can only be used for multi-staged discrete problems and the problem should satisfy the principle of optimality: each subsequence must also be optimal in an optimal sequence of decisions or choices. If there is no way to break down the desired problem into stages or the principle of optimality is conflicted, the algorithm is not applicable.

As we see the insufficiency of traditional methods, a sight of evolutionary counterparts naturally comes up. Darwinian evolution is a robust search and optimization mechanism

(referring to section 2.1). The problems that biological species have solved are typified by chaos, chance, temporality and nonlinear interactivity. They are conducively also the characteristics of problems that have been encountered in statistical pattern recognition. In consideration of the benefit evolutionary computation could obtain, practitioners have carried out a lot of attempts. This section is thereby devoted to analyze and survey these attempts of employing evolutionary computation in solving statistical pattern recognition problems. Our survey will come from two aspects: feature selection/extraction and learning.

2.2.1 Feature Selection and Extraction

Siedlecki and Sklansky [28] introduced the use of genetic algorithms for large-scale feature selection. In their approach, given the d-dimensional feature vector, a candidate subset is represented as a binary-valued vector $\mathbf{a} = [a_1, a_2, \ldots, a_d]^T$, where a_i $(i = 1 \ldots d)$ assumes value 0 if the i-th feature is excluded from the subset and 1 if it is present. They refer to $\mathbf{a}$ as the coding of a feature selection variable. The training error is to be minimized constrained by the size of the feature subset. The genetic operators of selection, crossover and mutation are utilized to evolve a population of potential solutions to an optimal result. The GA approach was compared with sequential search (forward and backward) and a variation of branch and bound algorithm. On a synthetic 24-dimensional data set as well as on a real 30-dimensional data set, GA outperformed these other feature selection methods in terms of both classification performance and computational effort. This technique was later expanded to allow linear feature extraction, by Kelly and Davis [29] and independently by Raymer *et al.* [30]. The single bit associated with each feature is expanded to a real-valued coefficient, allowing independent linear scaling of each feature, while maintaining the ability to remove features from consideration by

assigning a weight of zero.

Principal Component analysis (PCA) [31] is a well-known dimensionality reduction method in which the extracted features are essentially projections onto bases (eigenvectors) associated with the dominant eigenvalues of covariance matrix. However, Liu and Wechsler [1] made the observation that one can expect better classification performance from nonorthogonal bases over orthogonal ones (what PCA achieves). They summarized that search for the optimal projection bases involves various design criteria such as redundancy reduction, minimization of the reconstruction error, maximization of information transmission, sparseness or independence and successful classification. The objective function may lack an analytical form suitable for gradient descent and the procedure usually involves constrained and nonlinear optimization. Therefore they developed the *evolutionary pursuit* algorithm in which a genetic algorithm was used. The bases $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m$ of the m-dimensional PCA space are rotated according to a set of angles $\alpha_1, \alpha_2, \ldots, \alpha_{m(m-1)/2}$ with each angle in the range of $(0, \pi/2)$. Moreover, each axis is associated with a mask bit ($a_i = 0$ or $1, i = 1 \ldots m$) which indicates whether the axis is chosen as a basis. Fig. 2.2 gives an illustration. In their experiments proportionate selection, two-point crossover and bit mutation are used. Remarkable improvement on classification ability is experienced.

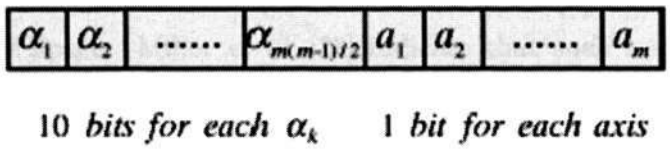


Figure 2.2: Chromosome representation of rotation angles and projection axes in evolutionary pursuit. Ten bits are used for each angle, so that every discretized (angle) interval is less than 0.09 degree, while one bit for each axis is indicating whether this axis is chosen as a basis vector. Copy from [1].

2.2.2 Supervised and Unsupervised Learning

Although feature selection/extraction can also be treated as a learning procedure, here we consider learning only as constructing classifier from labelled training examples (supervised) or unlabelled data (unsupervised). Since the class-conditional densities are hard to model, we often depend on directly identifying decision boundaries by optimizing certain criteria. In this manner the artificial neural networks (ANNs) typically minimize the mean square error between classifier outputs and given labels. Another example is support vector machine which works by maximizing the margin between different classes [32].

Yao [9] provided a comprehensive review on evolutionary artificial neural networks (EANNs). The paper distinguishes among three kinds of evolution in ANNs, i.e., the evolution of connection weights, of architectures and of learning rules. The evolution of connection weights is to find an optimal (or near optimal) set of connection weights for a network. While back-propagation (BP) is the most popular training algorithm for feed-forward ANNs, the drawback exists due to its gradient descent nature. It often gets trapped in a local minimum of the error function. From this view global search procedures like GAs can be used effectively to train an ANN. The evolution of architectures concerns searching the surface defined by the optimality level of ANN's architectures in the architecture space. Miller *et al.* [33] indicate that such a surface has several characteristics including:

- The surface is infinitely large since the number of possible nodes and connections is unbounded.
- The surface is nondifferentiable since change in the number of nodes or connections is discrete and can have a discontinuous effect on ANN's performance.

- The surface is complex and noisy since the mapping from ANN's architecture to ANN's performance after training is indirect, strongly nonlinear, and dependent on initial conditions.
- The surface is deceptive since ANNs with similar architectures can have dramatically different information processing abilities and performances.
- The surface is multimodal since ANNs with quite different architectures can have very similar capabilities.

These characteristics make evolutionary approach a more promising candidate than a heuristic approach like trial-and-error. Most research along this direction concentrates on the evolution of ANN's connectivity, i.e., the number of nodes in an ANN and the connection topology among these nodes. Little work, however, has been done on the evolution of node transfer functions. A key issue, by all means, is to decide a proper representation of the architecture. The third kind of evolution in ANNs is the evolution of learning rules. It is much more difficult to encode dynamic behaviors, like the learning rule, than to encode static properties like the architecture and connection weights in an ANN. Some work has been done on adjusting BP algorithm's parameters, such as the learning momentum, using evolutionary approaches.

Another representative work of applying evolutionary computation in supervised learning is [34]. In the paper, Pal *et al.* described a method for finding decision boundaries, approximated by piecewise linear segments, to classify patterns in a high-dimensional space. He used an elitist model of genetic algorithms. The method involves generation and placement of a set of hyperplanes in the feature space that yields minimum misclassification. Each hyperplane is encoded in terms of several angle variables and a perpendicular distance variable. The fitness of a chromosome is characterized by the

number of points it misclassifies. Since the optimum number of hyperplanes required for proper classification of a given data set is not known *a priori*, a conservative estimation or overestimation is normally done to initiate the algorithm. Consequently some of the hyperplanes may be found to be redundant in the final output as far as their contribution toward the generation of boundary is concerned. The process of elimination of redundant hyperplanes is performed as a postprocessing step. In their experiments the evolutionary algorithm compared favorably with Bayesian classifier, k nearest neighbor classifier and multilayer perceptron.

Unsupervised learning is also known as data clustering which concerns finding natural groupings in multidimensional data [35]. The widely used K-means algorithm is based on an intuitively simple criterion of minimizing the within cluster spread. It is reported to have the limitation of getting stuck at local optima which are not globally optimal [36]. This inspired Maulik and Bandyopadhyay [37] to propose a GA-based clustering technique. The task of GA is to search for the appropriate cluster centers such that the clustering metric is minimized. In the implementation each chromosome is a sequence of real numbers representing the K cluster centers. After the clustering is done, the centers are replaced by the mean points of the respective clusters. Upon these updated centers genetic operators of selection, crossover and mutation are performed. The best chromosome seen up to the last generation provides the solution to the clustering problem. The authors compared the GA-clustering algorithm and K-means over artificial and real-life data sets. The superiority of evolutionary search is demonstrated.

Hall *et al.* [38] described a genetically optimized approach for both fuzzy and hard c-means clustering. Again the cluster centers are elements of the population. The fitness is a reformulated version of HCM/FCM functional (c-means clustering criterion). Binary gray code representation is adopted, together with tournament selection, two-point

crossover, bit mutation and an elitist strategy. While those traditional counterparts are significantly affected by initialization, GA is more robust. In the experiment GA provides good partitions by settling in one of the most desirable extrema (which iterative algorithms also find if given optimal initialization) and never in an extremum representing a degenerate partition.

So far in this section we have seen a number of evolutionary algorithms which outperform classical optimization algorithms when applied to statistical pattern recognition problems. [39–48] are also recommended as successful trials. While the survey is far from complete, a global viewpoint is established about where evolutionary computation may help and how it is usually designed. In the following sections attention will be focused on two specific problems, namely ensemble learning and Markov random field modelling. Through the analysis of strengths and weaknesses of classical methods our work is thereby motivated.

2.3 Ensemble Learning

Learning as a fundamental procedure of pattern recognition usually concerns constructing a classifier from a set of training examples. While a learning machine might outperform others for a specific problem or for a specific subset of the input data, it is uncommon to find a single machine achieving the best result on the whole problem domain. As a consequence an ensemble of learning machines try to exploit the local different behavior of component learners to enhance the accuracy and the reliability of the overall system. There are also hopes that if some learners fail, the overall system can recover the error. Employing multiple learners can also derive from the application context. For example when multiple sensor data are available, they induce a natural

decomposition of the problem and we may use specialized learning machine for each different item.

In the last decade one of the main directions in machine learning research has been represented by methods for constructing ensembles of learning machines [49]. Empirical studies show that ensembles are often much more accurate than the individual base learner that makes them up [50–52]. Around the benefit there are a few explanations:

- Learning algorithms try to find a hypothesis in a given space $\mathcal{H}$ of hypotheses. In many cases we have only limited data sets and the learning algorithm may not find the optimal hypothesis but different ones that appear equally accurate with respect to the available training data. Although we can sometimes select among them the simplest or the one with the lowest capacity, we can avoid the problem by averaging or combining them to get a good approximation of the unknown true hypothesis.

- Another reason arises from the limited representational capability of learning algorithms. When the unknown function to be approximated is not present in $\mathcal{H}$, a combination of hypotheses drawn from $\mathcal{H}$ can expand the space of representable functions, embracing also the true one. Although many learning algorithms present universal approximation properties, with finite data sets these asymptotic features do not hold. From this standpoint ensembles can enlarge the effective hypothesis coverage, expanding the space of representable functions.

- Many learning algorithms apply local optimization techniques that may get stuck in local optima. For instance inductive decision trees employ a greedy local optimization approach, and neural networks apply gradient descent techniques to minimize an error function over the training data. Moreover optimal training with

finite data both for neural networks and decision trees is NP-complete [53, 54]. As a consequence even if the learning algorithm can in principle find the best hypothesis, we actually may not be able to find it. Building an ensemble using, for instance, different starting points may achieve a better approximation.

Numerous algorithms for constructing ensembles have been developed. According to [49], the techniques could be broadly organized into five kinds: subsampling the training examples, manipulating the input features, manipulating the output targets, injecting randomness and algorithm-specific methods. Evolutionary computation has seen successful application in injecting randomness into the learning algorithm (e.g. [55]; see some discussion in chapter 3). In this thesis, however, we focus on subsampling training examples since the technique has been receiving extensive interest. The intuitive idea behind it is that some examples may be hard to classify while others are easy (just imagine some are close to the decision boundary and others are far away from it). One may consider assigning different weights to them when they are input into a learning algorithm (e.g. decision tree or neural network), in other words, training the classifier using weighted examples. A set of weights associated with the examples is known as a *weighting*. Since the optimal weighting is not known *a priori*, it is suggested that a base learning algorithm is called repeatedly, each time fed with the training examples associated with a different weighting. Each time the learning algorithm is called, it generates a new classifier. Thus after many rounds, one shall obtain a number of classifiers and may combine them into an ensemble classifier.

More formally, consider a feature space $\mathbb{X}$, binary label space $\mathbb{Y} = \{-1, +1\}$ and a set of training examples

$$S = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{X}, y_i \in \mathbb{Y}, i = 1, ..., N\}. \tag{2.4}$$

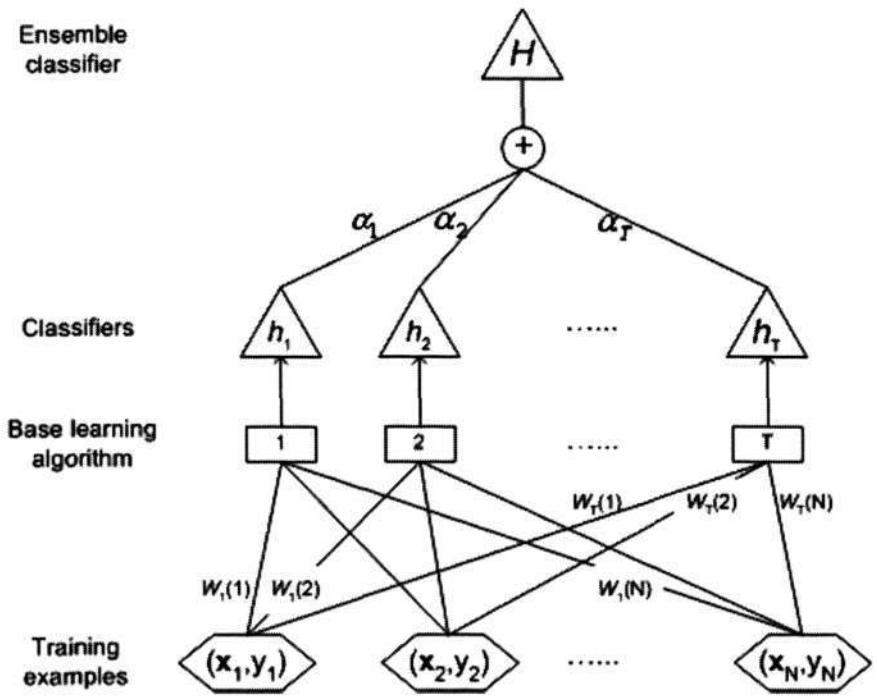


Figure 2.3: Ensemble learning based on manipulating the training examples.

The aim of ensemble learning is to construct a number of classifiers $\{h_t(.)\}$ whose individual options on an input feature $\mathbf{x}$ are voted to derive a consensus decision:

$$H(\mathbf{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})) \tag{2.5}$$

where $\sum_{t=1}^{T} \alpha_t = 1$ are the voting weights. By manipulating training examples each $(\mathbf{x}_i, y_i)$ is assigned a weight $w_t(i)$ at time t to form a non-negative weighting

$$\mathbf{w}_t = [w_t(1), ..., w_t(N)] \tag{2.6}$$

over S (subject to $\sum_{i=1}^{N} w_t(i) = 1$). When trained with a weighted training set $\{S; \mathbf{w}_t\}$, a learning algorithm generates a particular classifier $h_t(.)$. The base learning algorithm will be invoked for T times, each time with different $\mathbf{w}_t$ and therefore producing different

$h_t(.)$. It is expected that the subspaces of features misclassified by these constituent classifiers do not necessarily overlap; therefore the *ensemble classifier* $H(.)$ can improve the classification accuracy. This procedure is illustrated in Fig. 2.3.

The most straightforward way to implement this idea is called *Bagging* [56]. At each time, Bagging feeds the learning algorithm with a training set that consists of N examples drawn randomly (with replacement) from the original set S. The sampling is according to a uniform distribution and on the average each training set contains 63.2% examples of S, with several ones appearing multiple times. Such a set is called a *bootstrap replicate* of S [57]. In another way we can think of Bagging as assigning weight

$$w_t(i) = \text{number of occurance of } (\mathbf{x}_i, y_i)/N \tag{2.7}$$

on each training example. So essentially Bagging is using random weights drawn from a discrete Binomial distribution. At the end the induced classifiers are combined by even voting

$$\alpha_1 = ... = \alpha_T = 1/T. \tag{2.8}$$

There is another representative algorithm, named *AdaBoost* [58, 59] by Freund and Schapire, which sequentially adjusts the weights. AdaBoost starts from a uniform weighting

$$w_1(1) = ... = w_1(N) = 1/N. \tag{2.9}$$

At time t, one shall construct a classifier $h_t(.)$ based on $\mathbf{w}_t$ and calculate the weighted training error ϵ_t:

$$\epsilon_t = \sum_{i=1}^{N} w_t(i) \cdot \mathbf{I}(y_i \neq h_t(\mathbf{x}_i)), \tag{2.10}$$

where $\mathbf{I}(E) = 1$ if the event E occurs or 0 otherwise. Then let

$$\beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}. \tag{2.11}$$

Now one may update the weights by the following rule:

$$w_{t+1}(i) = w_t(i) \cdot \exp\{-\beta_t y_i h_t(\mathbf{x}_i)\}/Z_{t+1}, \tag{2.12}$$

where Z_{t+1} is a normalization constant. A direct explanation here is that weights for those examples which have been misclassified by the previous classifier ($y_i h_t(\mathbf{x}_i) < 0$) will be increased and those that have been correctly classified ($y_i h_t(\mathbf{x}_i) > 0$) will be decreased. The above procedure is iterated for T rounds, and finally one gets the ensemble classifier using (2.5) with

$$\alpha_t = \beta_t / \sum_{t=1}^{T} \beta_t. \tag{2.13}$$

Those constituent classifiers with lower training error will have higher voting weights in the ensemble.

There are several extensions to AdaBoost like BrownBoost [60], Soft Margin AdaBoost [61] and FloatBoost [62]. However, they all start from the same idea of adapting the weights step by step. A careful comparison between Bagging and AdaBoost reveals that basically they employ different ways to search for appropriate weightings $\{\mathbf{w}_t\}$ (considering each different weighting leads to a particular classifier). Bagging performs a "blind" search with randomly generated weightings. The constituent classifiers do not depend on each other and they can be constructed in parallel. Their opinions are given equal weight in the final ensemble. By contrast, AdaBoost is essentially a serial procedure. One shall generate each weighting (except the first one) by making an

adjustment on the previous one, using the evaluation result of the previous classifier to decide the step size and direction of the adjustment. The classifiers are finally combined by weighted voting.

After all, viewing the manipulation of training examples as a search problem opens a new window for us. Along this line we see some characteristics of the problem. The dimensionality of the weighting space is the size of training set which could be a large number. The convoluted shape also complicates the search. Furthermore, we are not looking for a single optimal point in this space, but a set of points. The optimization criterion is ill-posed since what we care about is the generalization ability of the whole ensemble which does not monotonously connect to the quality of a single point (training error of the corresponding classifier).

It is not hard to see that the difficulties which we are facing are exactly what evolutionary computation is good at (referring to section 2.1). This understanding inspires us to design an algorithm of ensemble learning which performs evolutionary search in the *weighting space*. A genetic algorithm can be adopted. Specifically, we may treat a weighting $\mathbf{w}_t$ as a chromosome and evaluate its fitness by the classification accuracy of its corresponding classifier $h_t(.)$ on the training examples. The algorithm shall maintain a population of chromosomes at each generation. Genetic operators will be used to evolve the population. The crossover operator is an attractive one through which large jumps are possible. More areas are thus explored in the weighting space. On the other hand, although GA uses probabilistic transition rules, not deterministic rules, it does much more than a random search. The *survival of fittest* principle continuously drives the algorithm to areas with lower classification error. The component classifiers therefore tend to have high accuracy. After evolution we may choose selecting some optimized classifiers to form the ensemble or even combining all the classifiers from the first to the

last generation.

Above described is a draft of employing genetic search for appropriate weightings over the training set. Notwithstanding its novelties, there are a lot of issues to be determined. Besides the detailed design, one may also wonder how the evolutionary approach will compare to those traditional methods in Bagging and AdaBoost. These issues will be addressed in chapter 3.

2.4 Markov random field modelling

Modelling by the theory of Markov random field (MRF) is an important technique in statistical pattern recognition. It is for specifying the spatial or contextual constraints about the physical world, which are necessary to solve the "inverse" problem of three-dimensional scene interpretation, given the inherently noisy and ambiguous two-dimensional information [63]. For example, image segmentation is to cluster the picture elements (pixels) into different regions and a feature vector consists of the pixel position together with pixel appearance. It is often assumed that in a spatial neighborhood the image pixels are more likely to fall into the same region. Another example is image restoration which aims to recover the genuine pixel values. We are usually interested in surfaces which are either piecewise continuous or piecewise constant [64]. To model *a priori* knowledge of spatial context like in the above examples, MRF provides a mathematically sound means.

Let $S = \{(i,j)|1 \leq i,j \leq n\}$ be the two-dimensional lattice of image pixels. We suppose that each pixel is associated with a random variable $l_{(i,j)}$ which takes its value from a discrete set $\{1,...,M\}$. M could denote the number of regions for image segmentation or simply 255 for image restoration. The random variables are probabilistically dependent

on each other. The two-dimensional field

$$L = \{l_{(i,j)} | \forall (i,j) \in S\} \tag{2.14}$$

is assumed to be Markovian in the sense that the probabilistic dependencies among $\{l_{(i,j)}\}$ are restricted to a spatial neighborhood. That is, we have

$$P(l_{(i,j)}|l_{S-(i,j)}) = P(l_{(i,j)}|l_{N_{(i,j)}}), \tag{2.15}$$

where $S-(i,j)$ includes all the pixels in S except (i,j) and $N_{(i,j)}$ is the set of locally neighboring pixels of (i,j). In other words, the Markovianity says that if the conditions of neighboring pixels are given, the distribution of the status of centering pixel is not dependent on all those pixels which are not neighbors. Based on this assumption, the joint probability of L is found to obey a Gibbs distribution

$$P(L) = Z^{-1} \times e^{-U(L)/T}, \tag{2.16}$$

in which Z is a normalizing constant, T is the temperature which is assumed to be 1 unless otherwise stated, and $U(L)$ is the prior energy. $P(L)$ is also known as the prior probability. The local conditional probability can be accordingly calculated by

$$P(l_{(i,j)}|l_{N_{(i,j)}}) = \frac{e^{-U(l_{(i,j)}|l_{N_{(i,j)}})}}{\sum_{l_{(i,j)}=1}^{M} e^{-U(l_{(i,j)}|l_{N_{(i,j)}})}}. \tag{2.17}$$

In this equation, the contextual constraints among neighboring pixels are expressed more directly.

Besides image segmentation and restoration, the MRF prior finds its application in many other problems including surface reconstruction, optical flow and matching. In many

cases we aim to find an optimal instantiation of the random field L given the image data $D = \{d_{(i,j)} | \forall (i,j) \in S\}$. We consider here a simple likelihood model of linear transformation and zero-mean, i.i.d Gaussian noise, i.e.

$$d_{(i,j)} = \psi(l_{(i,j)}) + e \tag{2.18}$$

where ψ is a mapping from label to the pixel value and $e \sim N(0, \sigma^2)$ is the noise. The likelihood probability is then written as

$$P(D|L) = \frac{1}{\prod_{(i,j)} \sqrt{2\pi\sigma^2}} e^{-U(D|L)}, \tag{2.19}$$

where

$$U(D|L) = \sum_{(i,j)} (d_{(i,j)} - \psi(l_{(i,j)}))^2 / (2\sigma^2) \tag{2.20}$$

is the likelihood energy. In a Bayesian framework the likelihood model will be combined with *a priori* model and the optimal solution L^* is defined using the *maximum a posteriori* (MAP) criterion:

$$L^* = \arg\max P(L|D) = \arg\max\{P(D|L)P(L)\}. \tag{2.21}$$

Combined with (2.16) and (2.19), this MAP estimation is equivalent to minimizing the posterior energy

$$L^* = \arg\min U(L|D) = \arg\min\{U(D|L) + U(L)\}. \tag{2.22}$$

In this way the MAP-MRF paradigm eventually leads to energy function minimization. The optimization induced is a major problem in MRF modelling [2], not only because

it directly affects the solution quality, but also because of its difficulty. A solution L consists of $n \times n$ variables $l_{(i,j)}$. Since each variable may take M possible values, the solution space will have a size of $M^{n \times n}$. Even for two-region segmentation of a 256×256 image, the size $2^{256 \times 256}$ is astronomical and will rule out any enumeration-based search techniques. Furthermore, the function $U(D|L)+U(L)$ bears a complex landscape which is fraught with numerous local optima. A closed-form solution clearly does not exist. How to implement efficient search in such an intractable space presents a fundamental challenge for the pattern recognition community.

Published techniques for minimization of MRF energy function can be broadly classified into two categories: *local* and *global*. In performing local search, the iterated conditional modes (ICM) [65] algorithm uses a *steepest descent* strategy. ICM starts with a random initialization L^0. It sequentially updates each variable $l^t_{(i,j)}$ into $l^{t+1}_{(i,j)}$ by maximizing $P(l_{(i,j)}|D, l_{S-(i,j)})$, the conditional posterior probability. Two assumptions are made in calculating $P(l_{(i,j)}|D, l_{S-(i,j)})$: First, the observation components $d_{(1,1)}, \cdots, d_{(n,n)}$ are conditionally independent given L and each $d_{(i,j)}$ has the same known conditional density function $P(d_{(i,j)}|l_{(i,j)})$ which is dependent only on $l_{(i,j)}$. Thus

$$P(D|L) = \prod_{(i,j)} P(d_{(i,j)}|l_{(i,j)}). \tag{2.23}$$

The second assumption is that $l_{(i,j)}$ depends on the labels in the local neighborhood, which is the Markovianity. From the two assumptions and the Bayes theorem, it follows that

$$P(l_{(i,j)}|D, l_{S-(i,j)}) \propto P(d_{(i,j)}|l_{(i,j)})P(l_{(i,j)}|l_{N_{(i,j)}}). \tag{2.24}$$

Obviously, $P(l_{(i,j)}|d_{(i,j)}, l_{N_{(i,j)}})$ is much easier to maximize than $P(L|D)$, which is the point of ICM. The above iteration defines an updating cycle and is repeated until con-

vergence, which is rapid and guaranteed. Despite its advantages, ICM suffers from a heavy dependence of solution quality on the initialization. Without a good starting point, the algorithm often gets stuck in an undesirable local optimum. Other local methods including relaxation labeling (RL) [66], highest confidence first (HCF) [67] and dynamic programming (DP) [68] face the same drawback with ICM.

Initialize T and L;
Repeat

- Randomly sample L from $N(L)$ under T;
- Decrease T;

Until $(T \rightarrow 0)$;
Return L;

Figure 2.4: The simulated annealing algorithm (copied from [2]). $N(L)$ is the vicinity of L.

To deal with this problem the idea of annealing can be borrowed. Annealing denotes the process through which temperature T in the Gibbs distribution (2.16) is gradually decreased. It can be incorporated into a deterministic search like in mean field annealing (MFA) [69] or, more prevalently, combined with a stochastic sampling algorithm like in simulated annealing (SA) [70, 71]. Fig. 2.4 is a typical flow of the SA algorithm. In stead of performing gradient descent, a random sampling method, such as Metropolis algorithm or Gibbs sampler, is used to locate the next configuration. At the initial stages of SA, sufficiently high T results in a nearly random walk whereby state transitions causing higher energy can be accepted; with T slowly decreased, transition to a "bad" state is more and more less happened and eventually when T approaches 0, the system is "frozen" near the global minimum of the energy function. Global convergence is theoretically guaranteed for SA, but it also brings unaffordable computation. Nevertheless, these methods, i.e. SA and MFA, have the capability of jumping over local optima in

the search space; they are, therefore, characterized as *global*.

The traditional algorithms, no matter a local method like ICM or a global method like SA, rely on the step by step adjustment of a single solution. Such an operation could be very inefficient or easily cheated by an undesirable local optimum. It is not surprising to ask how a population-based search like genetic algorithms will behave in the MRF domain. The potential solutions shall be encoded into chromosomes. The chromosomes are allowed to evolve over a number of generations. At every generation, the fitness of each chromosome will be calculated. Based upon their fitness evaluations, the chromosomes are probabilistically chosen to become the parents for mating. A crossover operator is then applied to combine the information contained within pairs of parents to produce the child chromosomes, which are usually better than their parents. To maintain the population diversity, random changes are introduced into the child chromosomes with a small probability, using a mutation operator. The mutated chromosomes then constitute the next generation and the evolution will continue.

Though the above idea seems promising, we argue that a direct application of GAs may not be suitable. It is well-known that the success of GA critically depends on the quality of reproducing and mixing *building blocks* from different chromosomes. When applied to problems with building blocks highly intertwined and spread, the conventional operators of crossover and mutation are not well settled [72]. Specifically, in a problem characterized by MRF, the mechanisms of cut-and-exchange and randomly flipping some genes can not ensure the contextual constraint to be complied with, which is believed a prerequisite to reproduce building blocks. The advantages of GA will then be damaged. That is also to say, we need have genetic operators carefully designed according to the priori knowledge, which is the Markovianity, for a successful adoption of evolutionary search in our problem. Rather than be ignored, the constraint shall play an indispensable

role to confine the search in a reasonable scope. Starting from this thought, chapter 4 will see a novel design of evolutionary algorithm for MRF energy function minimization.

Chapter 3

Classification by Evolutionary Ensembles

3.1 Introduction

Since learning is often formulated as solving difficult optimization problems, the adoption of evolutionary computation makes a natural choice (referring to section 2.2). Among the numerous works in evolutionary learning, probably the most representative one is on evolutionary artificial neural networks (EANNs) [9]. For example, Yao *et al.* [10] developed an automatic system based on evolutionary programming (EP) [73] for designing feedforward ANNs. The initial population of networks is uniformly generated at random. The so called EPNet evolves both the architectures and connection weights of ANNs simultaneously. It does not use crossover operators but relies on novel mutations and a rank-based selection scheme. There are five types of mutation attempted in order: hybrid training, node deletion, connection deletion, connection addition and node addition. Deletions are attempted before additions so that EPNet always encourages small

size ANNs. The experimental results show that EPNet can produce compact networks with good generation ability in comparison with traditional algorithms.

In [55], Yao *et al.* improved their work on EPNet through discussing how to make best use of all the information contained in the whole population. They claimed that:

> Learning is different from optimization because we want the learned system to have best generalization, which is different from minimizing an error function...the maximum fitness is not equivalent to best generalization in evolutionary learning...Other individuals in the population may contain some useful information that will help to improve generalization of learned systems.

With this idea they built an ensemble of EANNs by linearly combining the individuals in the last generation. Different combination methods were investigated. The integrated system by the recursive least-square method was found outperforming the best individual in terms of generalization for all three problems they tested.

The algorithm to be presented in this chapter will show many discrepancies from the above-mentioned work. Although both are characterized by evolutionary learning, EPNet is through injecting randomness into the learning algorithm (random initialization and probabilistic transition of connection weights and architecture) while our algorithm is based on a different thought, i.e. subsampling training examples under diverse distributions. The motivation and a simple draft have been introduced in the previous chapter. On the other hand, our algorithm extends the idea in [55] by making use of all the individuals from the first to the last generation. The individuals in the early stages of evolution are also discovered containing helpful information.

The rest of this chapter is organized as follows. A detailed description of the proposed

algorithm is given in section 3.2. Together included is experimental comparison with two representative algorithms in manipulating training examples: Bagging and AdaBoost. We also discuss incorporating the speciation technique of fitness sharing to improve its performance. In section 3.3 a face detection system is developed based on the proposed algorithm. Its capability is further demonstrated. We summarize in section 3.4.

3.2 A New Algorithm of Evolutionary Learning

3.2.1 Algorithm Description

We have known from 2.3 that a weighting is a group of weights over the training examples. The key job in our algorithm is to find a set of weightings since each of them leads to a particular classifier. It is therefore natural and straightforward to treat a weighting as a chromosome. Formally we assume there are K chromosomes in a population and the population will be evolved for J generations. We shall generate a set of chromosomes $\{\mathbf{w}_{j,k}|j = 1...J, k = 1...K\}$ throughout the evolution, each of which is defined as

$$\mathbf{w}_{j,k} = (w_{j,k}(1), ..., w_{j,k}(N)) \tag{3.1}$$

(subject to $\sum_{i=1}^{N} w_{j,k}(i) = 1$ and $w_{j,k}(i) \geq 0$). Note that we employ real-value coding here, not a conventional bit string coding. So in later description we use terms "weighting" and "chromosome" interchangeably.

Every time a base learning algorithm is invoked and fed with the weighted training set $\{S; \mathbf{w}_{j,k}\}$, we shall obtain a classifier $h_{j,k}(.)$. We compute the weighted training error of

$h_{j,k}(.)$ on the N examples by

$$\epsilon_{j,k} = \sum_{i=1}^{N} w_{j,k}(i) \cdot \mathbf{I}(y_i \neq h_{j,k}(\mathbf{x}_i)) \tag{3.2}$$

Based on (3.2) the fitness of a chromosome $\mathbf{w}_{j,k}$ is defined as

$$f_{j,k} = \frac{1}{2} \log \frac{1 - \epsilon_{j,k}}{\epsilon_{j,k}} \tag{3.3}$$

The two equations above are inspired by (2.10) and (2.11) in AdaBoost. Apparently a lower training error means a higher fitness value. If one views the training set S as an environment where the chromosomes compete to live, $\{f_{j,k}\}$ provide a reasonable indication of how well each chromosome adapts to it. However, the relationship is not proportional but a log function. This is illustrated in Fig. 3.1. When $\epsilon_{j,k}$ varies in $0.01 \sim 0.99$, the fitness only falls in $2.2976 \sim -2.2976$. It is expected that some nice examples (easy to classify) may have high weights while nasty examples (difficult to classify) have low weights. In that case the training error will be low but the fitness will not be too high. The genetic algorithm is thus not easy to be cheated by those undesirable chromosomes and lose the ability to classify nasty examples. Still, the practitioners should bear in mind the danger.

The algorithm is now outlined in Fig. 3.2. In order to make it comparable to Bagging and AdaBoost in computational cost, we always set $J \times K = T$. That is, there are also T classifiers in the final ensemble. The algorithm has a simple and clear control structure. The population of chromosomes in the first generation, $\{\mathbf{w}_{1,k}\}_{k=1}^{K}$, is generated randomly. For example, referring to (3.1), we may generate N random numbers which are uniformly distributed in $(0, 1)$, then normalize them to form a valid weighting. Each population $\{\mathbf{w}_{j,k}\}_{k=1}^{K}$ in the later generations ($j = 2...J$) will be generated from its

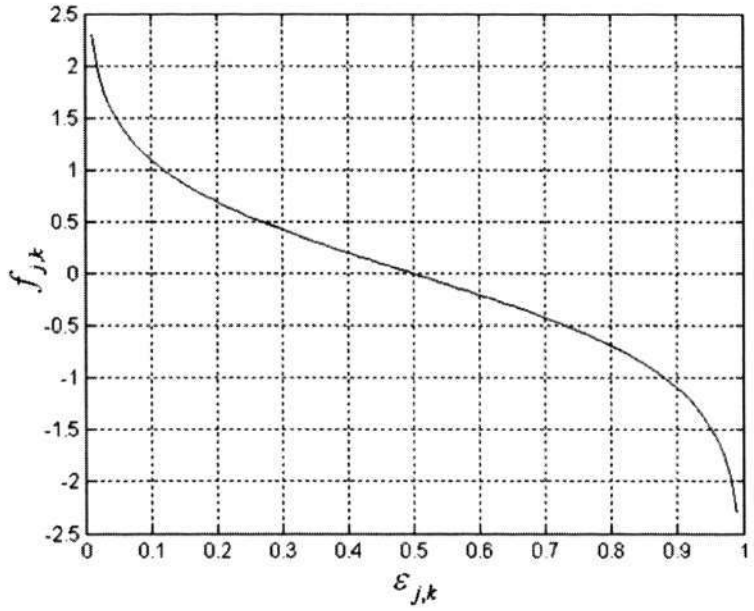


Figure 3.1: The design of fitness function based on classification error. The negative fitness value could be corrected by a linear mapping [3].

preceding population $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ by genetic operators. For each chromosome in the populations, we will train a classifier accordingly and evaluate its fitness. All the induced classifiers are finally combined to derive the ensemble classifier. In the ensemble each component classifier $h_{j,k}(.)$ will be assigned a voting weight $\alpha_{j,k}$. The fitness $f_{j,k}$ of a classifier will determine how its opinion on a pattern is weighted with respect to the opinions of other classifiers, that is, those classifiers with high fitness evaluation will have high voting weights:

$$\alpha_{j,k} = f_{j,k} / \sum_{j=1}^{J} \sum_{k=1}^{K} f_{j,k}. \tag{3.4}$$

There are two important issues to be addressed: one is the mechanism to evolve the chromosomes generation by generation, and the other is how to train a classifier using weighted examples.

Given S, J, K

Initialize for $j = 1$:

- Randomly generate the chromosomes $\{\mathbf{w}_{1,k}\}_{k=1}^{K}$
- Train a classifier $h_{1,k}(.)$ for each $\mathbf{w}_{1,k}$
- Evaluate a fitness value $f_{1,k}$ for each $\mathbf{w}_{1,k}$

For $j = 2...J$:

- Generate the chromosomes $\{\mathbf{w}_{j,k}\}_{k=1}^{K}$ from $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ by reproduction, crossover and mutation
- Train a classifier $h_{j,k}(.)$ for each $\mathbf{w}_{j,k}$
- Evaluate a fitness value $f_{j,k}$ for each $\mathbf{w}_{j,k}$

Output the ensemble classifier:

$$H(.) = \text{sign}(\sum_{j=1}^{J}\sum_{k=1}^{K}\alpha_{j,k}h_{j,k}(.))$$

where $\alpha_{j,k} = f_{j,k}/\sum_{j=1}^{J}\sum_{k=1}^{K} f_{j,k}$.

Figure 3.2: The algorithm of evolutionary learning by manipulating training examples.

3.2.1.1 Reproduction, Crossover and Mutation

Given the chromosomes and their fitness values in generation $j-1$, i.e. $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ and $\{f_{j-1,k}\}_{k=1}^{K}$, we repeat the following procedure:

1. *reproduction*: We use *roulette-wheel* selection [18]. Two chromosomes $\mathbf{w}_{j-1,k1}$ and $\mathbf{w}_{j-1,k2}$ are sampled (with replacement) from $\{\mathbf{w}_{j-1,k}\}_{k=1}^{K}$ according to the distribution

$$[\frac{f_{j-1,1}}{\sum_{k=1}^{K} f_{j-1,k}}, ..., \frac{f_{j-1,K}}{\sum_{k=1}^{K} f_{j-1,k}}] \tag{3.5}$$

The possibility of a chromosome to be selected is proportionate to its fitness. Chromosomes with higher fitness have more chances to copy their genes to the

next generation. Note that the selected two chromosomes may be the same.

2. *crossover*: Only one offspring chromosome is produced. It is a linear combination of the two parent chromosomes $\mathbf{w}_{j-1,k1}$ and $\mathbf{w}_{j-1,k2}$:

$$\mathbf{w}_{offspring} = \frac{f_{j-1,k1} \cdot \mathbf{w}_{j-1,k1} + f_{j-1,k2} \cdot \mathbf{w}_{j-1,k2}}{f_{j-1,k1} + f_{j-1,k2}} \tag{3.6}$$

   Apparently the weight items in $\mathbf{w}_{offspring}$ will also sum to 1, like in $\mathbf{w}_{j-1,k1}$ and $\mathbf{w}_{j-1,k2}$. We emphasize that in this step there is selective pressure involved; parent chromosome which has higher fitness will contribute more into the offspring chromosome. This is different from a traditional GA which normally imposes selective pressure in the *reproduction* step only [18]. Moreover, we always run this crossover operator after reproduction, while in a traditional GA it is normally run with a probability (< 1).

3. *mutation*: Here comes another parameter of the algorithm, probability of mutation p_m. Mutation is a mechanism of random perturbation to create new building blocks not already existing. Considering the outcome $\mathbf{w}_{offspring}$ of crossover which consists of N genes (each is a weight item associated with a training example), we randomly partition the N genes into $N/2$ pairs (if N is odd, then partition into $(N-1)/2$ pairs). For each pair, we exchange the values of the two genes by probability p_m; that is, the weights associated with two training examples are exchanged. The operation protects the population from premature convergence. Usually p_m is given a very small value. We have observed that this kind of disturbance in chromosomes is helpful in improving the final classification accuracy.

The above three-step procedure is repeated for K times, each time the mutated chromosome is saved as a member of the generation j. In the end we get all the chromosomes

$\{\mathbf{w}_{j,k}\}_{k=1}^{K}$. Based on these new weightings, we can train a new generation of classifiers and evaluate their fitness. The iteration in Fig. 3.2 thus continues.

We have mentioned some differences between a traditional GA and our algorithm. They are mostly due to the consideration that the chromosomes throughout all the generations shall make contribution to our final result (through the classifiers); while in a traditional GA one only picks the best chromosome in the final generation as the final result. That also makes the *replacement* issue [3], i.e. how to replace the old chromosomes with the new generated chromosomes, unnecessary in our algorithm. To our best knowledge Yao *et al.* [55] made the first attempt in neural network learning that all the chromosomes in the last generation are combined to create the final result. Our algorithm, in a further way, makes use of all the generations of chromosomes. With this novel idea we will see the benefits from the exploitation of less optimized chromosomes in the experiment.

3.2.1.2 Training with Weighted Examples

If one thinks of evolving the weightings as a high-level job, the low-level job concerns handling weighted training examples in a base learning algorithm. There are two approaches to take a weighting into consideration. First, many learning algorithms are based on minimizing an error function which is a sum of error items caused by each of the training examples. In this case one can design a revised function which assigns a weight to each error item, so that heavily weighted examples are more influential. For example, one may attempt to minimize the weighted error function

$$\sum_{i=1}^{N} w_{j,k}(i) \cdot \mathbf{I}(y_i \neq h_{j,k}(\mathbf{x}_i)), \tag{3.7}$$

not the conventional $\sum_{i=1}^{N} \mathbf{I}(y_i \neq h_{j,k}(\mathbf{x}_i))$. However, if the learning algorithm is not readily adaptable to the inclusion of weights, one shall employ a second approach proposed in [52]. It is based on resampling S with replacement according to the weighting $\mathbf{w}_{j,k}$. This approach is to some extent similar to what is employed in Bagging (which performs uniform sampling). Quinlan [74] reported that both approaches are very effective. But in our algorithm, we prefer the first approach if applicable.

So far we explained the framework in great details. It actually provides a generic approach for ensemble classification. Many modifications are possible in evolving the chromosomes [3]. For example, one may use other selection methods in the reproduction step, or even consider using other coding methods instead of the current real-value coding. The crossover operator can also be redesigned. On the other hand, the algorithm is not fixed to a particular base learner. Different kinds of learning machines, like k nearest neighbor, neural network and decision tree, can all be used. One shall consider all these aspects when applying our algorithm to a particular problem. Nevertheless, we are not covering all these possibilities since the purpose here is not to find the best combination.

Although we initially set a binary label space $\mathbb{Y} = \{-1, +1\}$, our algorithm applies to multi-class problems immediately. The only prerequisite is that the component classifiers can perform multi-class classification, which is fulfilled by most of the kinds of classifiers. Referring to (3.2), for each potential weighting we calculate its weighted training error by only comparing the predicted output $h_{j,k}(\mathbf{x}_i)$ with the actual label y_i. As long as $h_{j,k}()$ works, our algorithm works. There is no need for extra attention on multi-class classification. For those component classifiers which can not perform multi-class classification, the *error correcting output coding* method [75] may be helpful. In principle it reduces the multi-class problem to a number of binary problems.

The computational cost of our algorithm is comparable to that of Bagging and AdaBoost.

They have the same number (T) of ensemble members and hence the same number of calls to the base learning algorithm. The overheads of evolving the weightings are negligible. Furthermore, the strategy of using GA for ensemble classification can support greater computational efficiency by allowing parallelism of learning procedure which is not readily possible with AdaBoost.

3.2.2 Experimental Studies

We have applied our algorithm to a total of 20 data sets selected from the UCI Machine Learning Repository [76]. These data sets have been extensively adopted for benchmark study. A summary of some of their properties is given in Table 3.1. The training set size could be large (e.g. "Letter-Recognition" and "Waveform") or small (e.g. "Labor"), and the dimension of feature space could be high (e.g. "Audiology" and "Sonar") or low (e.g. "Iris"). One may also notice that some of the data sets have more than 2 classes (e.g. "Audiology" and "Letter-Recognition"). In the appendix A, we give a brief explanation for the classification problems involved.

We employ a 5-fold cross-validation in estimating the classification accuracy. All the training examples in a data set are partitioned randomly into 5 parts. 4 of them are selected for training and the constructed ensemble is tested on the remaining part. This process is repeated for five times, each time with a different part for test. The test results are then combined to calculate the classification accuracy over the whole data set. This strategy makes sure that each training example will be used for both training and test. Furthermore, to achieve a reliable estimation, on each data set we repeat the above procedure for 50 times, starting from partitioning the training examples. The obtained fifty values of classification accuracy are averaged to get the final estimation.

Table 3.1: Summary of the 20 UCI benchmark data sets used in the experiments.

Data set	training set size	dimension of feature space	number of classes
Audiology	226	69	24
Automobile	205	24	7
Breast-Cancer-W	699	9	2
Diabetes	768	8	2
German	1000	20	2
Glass	214	9	7
Heart	270	13	2
Hepatitis	155	19	2
Horse-Colic	368	21	2
Iris	150	4	3
Labor	57	16	2
Letter-Recognition	20000	16	26
Segment	2310	19	7
Sonar	208	60	2
Soybean-Large	683	35	19
Splice	3177	60	3
Thyroid	215	5	3
Vehicle	846	18	4
Voting	435	15	2
Waveform	5000	21	3

The proposed algorithm is compared with Bagging and AdaBoost since they are all based on the same thought. For the base learning algorithm RBF network [4] is used. The number of classifiers in an ensemble is set to 100. Clearly, this number is somewhat arbitrary and may not be optimal. However, since it is fixed for all algorithms, the

Table 3.2: Mean classification error rate on 20 data sets. Bagging, AdaBoost, our algorithm and a modified version of our algorithm (denoted by "our algo.*") are compared, using RBF network as the base learning algorithm. Each of the algorithms is implemented for 50 times on every data set. The winner in each comparison is emphasized in bold value.

	error rate			
Data set	Bagging	AdaBoost	our algo.	our algo.*
Audiology	0.2212	**0.1766**	0.1841	0.1955
Automobile	0.2122	0.2114	**0.2113**	0.2118
Breast-Cancer-W	0.0452	**0.0365**	0.0376	0.0398
Diabetes	0.2412	0.2691	**0.2405**	0.2575
German	**0.2679**	0.2763	0.2707	0.2792
Glass	0.3083	**0.2791**	0.2814	0.2898
Heart	**0.1866**	0.2085	0.1871	0.2116
Hepatitis	0.1702	0.1693	0.1693	**0.1691**
Horse-Colic	**0.1558**	0.1962	0.1625	0.1993
Iris	**0.0693**	0.0845	0.0696	0.0830
Labor	0.1847	**0.1836**	0.1852	0.1846
Letter-Recognition	0.0654	**0.0315**	0.0456	0.0435
Segment	0.0352	**0.0242**	0.0247	0.0290
Sonar	0.2991	**0.1839**	0.1907	0.2031
Soybean-Large	0.1233	**0.0993**	0.1001	0.1141
Splice	0.0725	**0.0717**	0.0720	0.0718
Thyroid	0.0429	0.0426	**0.0424**	0.0440
Vehicle	0.2875	**0.2420**	0.2461	0.2468
Voting	**0.1026**	0.1187	0.1098	0.1263
Waveform	0.1524	**0.1287**	0.1295	0.1300

comparison should be fair. Moreover, such a size of ensemble classifier has been widely employed in experimental studies of ensemble learning [77]. For our algorithm we ac-

Table 3.3: Variance of estimation for classification error rate in the experiment. The mean values are recorded in Table 3.2.

	estimation variance			
Data set	Bagging	AdaBoost	our algo.	our algo.*
Audiology	0.0035	0.0047	0.0039	0.0046
Automobile	0.0031	0.0044	0.0040	0.0048
Breast-Cancer-W	0.0006	0.0012	0.0010	0.0015
Diabetes	0.0033	0.0035	0.0029	0.0043
German	0.0037	0.0045	0.0035	0.0047
Glass	0.0024	0.0040	0.0033	0.0042
Heart	0.0021	0.0036	0.0032	0.0035
Hepatitis	0.0028	0.0027	0.0021	0.0033
Horse-Colic	0.0018	0.0030	0.0023	0.0031
Iris	0.0009	0.0005	0.0012	0.0011
Labor	0.0015	0.0036	0.0026	0.0034
Letter-Recognition	0.0006	0.0017	0.0009	0.0021
Segment	0.0005	0.0024	0.0018	0.0022
Sonar	0.0024	0.0035	0.0032	0.0037
Soybean-Large	0.0014	0.0021	0.0028	0.0020
Splice	0.0007	0.0013	0.0011	0.0016
Thyroid	0.0005	0.0009	0.0012	0.0011
Vehicle	0.0026	0.0033	0.0030	0.0041
Voting	0.0018	0.0022	0.0028	0.0029
Waveform	0.0012	0.0025	0.0020	0.0033

cordingly set $J = K = 10$, that means 10 generations of population each containing 10 classifiers. From a viewpoint of traditional genetic algorithms, this may seem a rather limited number of chromosomes and generations for optimization. The algorithm may not converge after such a short-term evolution. However, in the domain of learning, we

aim on the generalization capability which is signified by the classification accuracy on new and unseen examples (we estimate this accuracy using a test set). A high accuracy on the training set does not necessarily mean a same accuracy on the test set. A converged population of classifiers may provide optimized training accuracy, but it tends to suffer from the "over-fitting" effect and provide deteriorated generalization capability [4]. Here we will not take a theoretical study on the topics of model complexity and model selection. However, to clarify our point we modify the proposed algorithm in that only the optimized component classifiers in the last generation will be combined to form the ensemble classifier. This modified version of our algorithm is also tested on the UCI data sets. Note that it is only presented to testify if the inclusion of chromosomes in the early generations is beneficial.

The estimated mean values of classification error rate are listed in Table 3.2. A lower error rate means a higher classification accuracy. Four algorithms: Bagging, AdaBoost, our algorithm and the modified version of our algorithm (denoted by "our algo.*") are compared. The winner in each data set is given in bold style. For further insight the variances of estimation on each data set are also recorded in Table 3.3. We thus make the following observations:

1. The classification error rate tends to be high when the training set size is small while the dimension of feature space is large (e.g. "Sonar"). In the same way, the error rate tends to be low when we have enough training examples (e.g. "Letter-Recognition").

2. In all of 20 data sets, AdaBoost wins in 11 sets (according to the mean classification error rate), Bagging wins in 5 sets, our algorithm wins in 3 sets, and the modified version of our algorithm wins in only 1 set. It seems that AdaBoost does best and

our algorithm is only comparable to Bagging which essentially performs random search. Additionally, in each of the 3 sets where our algorithm wins, its error rate is very close to the second best value. Because of the randomness of the estimation, we may consider that they are statistically equal.

3. Further we check all the sets where our algorithm loses and notice a favorable phenomenon. That is, the error rates of our algorithm are in most cases quite close to those of the best algorithm and much better than the worst algorithm. Our algorithm only ranks the worst in "Labor" while in this case the four error rates are very close $(0.1847; 0.1836; 0.1852; 0.1846)$. Clearly our algorithm is not sensitive to the problem domain and in most cases achieves classification accuracy comparable to the best algorithm. In the mean time, Bagging and AdaBoost are not so stable and may perform poorly in some cases.

4. The modified version of our algorithm uses only the optimized component classifiers in the last generation to construct the ensemble. It wins in the "Hepatitis" data set. However, the four error rates are actually very close $(0.1702; 0.1693; 0.1693; 0.1691)$. Compared with our genuine version, its performance is more dependent on the specific data sets and in total 20 cases it loses 16 cases. We therefore draw the conclusion that the more randomized component classifiers in the early generations should not just be thrown away and they can also contribute to the ensemble as the more optimized ones in the late generations. Since the computational cost of training these classifiers has been paid in early evolution, their inclusion will not bring any additional burden.

5. Among the four algorithms, Bagging generally provides the least variance. This observation is consistent with the conclusion in [50]. AdaBoost and the modified version of our algorithm seem more unstable and have high variance. The situation

is alleviated to some extent in our original algorithm by the inclusion of more randomized component classifiers.

After all, the difference among the four algorithms is generally not so remarkable. Algorithms providing lower error rates generally have higher variance, for example in the "Glass" data set. It says that a specific algorithm may converge to a good local optima temporarily; however, the convergence is sensitive to the starting search point and the results across multiple runs could be much different. Nevertheless, the comparison results show that our algorithm performs consistently well across the data sets. While it may not achieve the best classification accuracy, it always provides a satisfying approximation. It is believed to be a favorable characteristic of evolutionary search which can be exploited in cases with little *a priori* knowledge. This constitutes one of the attractive properties of our algorithm. In the following we continue to study its sensitivity to parameter selection and controlled noise. The results will give further support to our algorithm as an interesting technique for general learning tasks.

3.2.2.1 sensitivity to p_m

Note that there is no parameter for crossover in our algorithm. The parameter p_m, probability of mutation, is always set 0.05. We have experimented a range of values for $p_m \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06\}$ on four benchmark data sets (other settings are the same with the above subsection). The four data sets are selected due to their representability of different classification tasks. The results are recorded in Table 3.4.

In traditional GAs for numerical optimization, the probability of mutation is usually given a very small value (e.g. 0.001) and it has significant influence on the convergence performance. In our formulation of evolutionary ensemble learning, however, p_m is given

Table 3.4: Sensitivity of our algorithm with respect to the probability of mutation p_m. The experiment is on four data sets: Audiology, Iris, Splice and Waveform. The classification error rates are recorded.

p_m	Audiology	Iris	Splice	Waveform
0.01	0.1855	0.0709	0.0741	0.1326
0.02	0.1829	0.0710	0.0731	0.1307
0.03	0.1847	0.0704	0.0735	0.1318
0.04	0.1836	0.0713	0.0719	0.1300
0.05	0.1841	0.0696	0.0720	0.1295
0.06	0.1873	0.0688	0.0729	0.1337

a relatively large value ($0.01 \sim 0.06$) and the influence on classification accuracy is not so significant (referring to Table 3.4). Considering the randomness of experiment, the performance in different settings for p_m is largely equal. This is partially due to the fact that the success of our algorithm does not depend heavily on the convergence of evolution. For a reasonable range of p_m, the performance will not fluctuate too much. Yet, we still observe that a setting of 0.05 is generally better than 0.01 or 0.06. We recognize that this setting is not a universally best choice for all practical problems. In our experiment it is only picked to demonstrate the potential of evolutionary search in ensemble classification. For practitioners, however, the parameter must be fine-tuned to the problem domain, which could be much different from our recommendations. One may consider dynamic parameters and carry out additional search for optimum values [78].

3.2.2.2 sensitivity to T

Another parameter in our algorithm is the size of the ensemble. While it is true that $T = 100$ for all algorithms in comparison is fair, we still need to study its influence by

adjusting this setting. Here we report additional results with $T = 25, 64, 225, 400$ on data set Audiology and Waveform (Table 3.5). For our algorithm that means $J = K = 5, 8, 15, 20$ respectively. The result on $T = 100$ is also listed in the table (copied from Table 3.2).

Table 3.5: Sensitivity of the algorithms (Bagging, AdaBoost and our algorithm) with respect to the size of ensemble. The experiment is on two data sets: Audiology and Waveform. The classification error rates are recorded. The winners are emphasized in bold value.

	Audiology			Waveform		
T	Bagging	AdaBoost	Our algo.	Bagging	AdaBoost	Our algo.
25	0.2991	**0.2409**	0.2517	0.2007	**0.1785**	0.1833
64	0.2507	**0.2015**	0.2090	0.1712	**0.1468**	0.1537
100	0.2212	**0.1766**	0.1841	0.1524	**0.1287**	0.1295
225	0.2046	**0.1635**	0.1673	0.1401	0.1190	**0.1184**
400	0.1893	0.1586	**0.1542**	0.1296	0.1135	**0.1106**

We clearly see that the test error rate continues falling with the increase of the complexity of final ensemble. Within this general trend, however, the three algorithms perform in slightly different ways. Bagging is rather inefficient since by $T = 400$ it could only achieve similar performance with AdaBoost and our algorithm by $T = 100$. It is probably due to the totally random nature of Bagging. In the data set of Audiology, AdaBoost outperforms our algorithm from $T = 25$ to 225. With $T = 225$ their results are quite equal (0.1635 vs 0.1673). But with $T = 400$ our algorithm is the best (0.1542). In Waveform we observe the same behavior. While AdaBoost outperforms our algorithm from $T = 25$ to 100, the situation turns opposite from $T = 225$ to 400 (still, the differences between the two algorithms are not very prominent). We would say that our algorithm is more preferred in the experiments for large ensembles. Nevertheless, the

conclusion is probably dependent on the problem domains and may not be justified for general cases without a theoretical analysis of the optimal values for T. We will discuss a possible way to determine the ensemble size in section 3.4.

3.2.2.3 sensitivity to data set noise

An important criterion of algorithm evaluation is to see how it behaves under different levels of noise. In this study we will compare the behaviors of Bagging, AdaBoost and our algorithm by injecting different percentages of error label into the training examples. That is, as soon as we identify the training set by cross-validation, we substitute the genuine labels of some examples (randomly selected) with different ones. The percentage of error label is controlled. We carry out the experiment on data set Audiology and Waveform. The results are recorded in Table 3.6.

Table 3.6: Sensitivity of the algorithms (Bagging, AdaBoost and our algorithm) with respect to controlled noise in the data sets. The noise level is the percentage of error label in the training set. The experiment is on two data sets: Audiology and Waveform. The classification error rates are recorded (results for noise level= 0% are copied from Table 3.2). The winners are emphasized in bold value.

noise	Audiology			Waveform		
level	Bagging	AdaBoost	Our algo.	Bagging	AdaBoost	Our algo.
0%	0.2212	**0.1766**	0.1841	0.1524	**0.1287**	0.1295
5%	0.2297	0.1913	**0.1902**	0.1581	**0.1339**	0.1342
10%	0.2401	0.2108	**0.2039**	0.1670	0.1426	**0.1418**
15%	0.2535	0.2373	**0.2188**	0.1783	0.1582	**0.1541**
20%	0.2674	0.2612	**0.2315**	0.1927	0.1799	**0.1713**

The performance will doubtlessly deteriorate with the increase of noise. This has been confirmed by the data. But among the three algorithms AdaBoost is the most sensitive

one. Its error rate with 20% noise increases by 47.90% (with respect to no artificial noise) in data set Audiology while the values for Bagging and our algorithm are 20.89% and 25.75% respectively. In data set Waveform, the value is 39.78% for AdaBoost, 26.44% for Bagging and 32.28% for our algorithm. The same phenomenon is also reported in [79]. Ratsch *et al.* [61] gave an explanation that AdaBoost concentrates its resources on a few hard-to-learn patterns; when there are outliers in the training set, the algorithm will be misled and provide suboptimal solution. In our experiment AdaBoost is the best algorithm when no noise is added, but it loses soon after noise is added.

On the other hand the experiment reveals that more randomized approaches like Bagging and our algorithm are preferred with highly noisy patterns. In particular our algorithm leads the comparison in Audiology after noise level is 5% and in Waveform after noise level is 10%. Because of the selective pressure in the evolution and weighted voting, our algorithm appears a bit more sensitive to noise than Bagging. But, since it adopts population-based search and probabilistic transition rules, it is much more robust to outliers than a gradient-descent approach like AdaBoost. In general our algorithm is seen to provide a better balance between accuracy and noise resistance than Bagging and AdaBoost.

3.2.3 Encouraging Diversity by Fitness Sharing

Previously we have introduced a novel formulation of ensemble learning based on manipulating training examples. The algorithm achieves good generalization through the experiments and shows its immunity against noise. However, referring to Table 3.2, it loses to AdaBoost and Bagging in many cases. In total 20 data sets the algorithm only wins in 3 of them. No doubt such performance is far from satisfactory. In a recent

survey Brown *et al.* [80] analyzed the importance of diversity (of those base learners) in ensemble learning and offered that "evolutionary diversity methods do not concern themselves with creating a population that is complementary in any way, but instead with just ensuring the maximum amount of the hypothesis space is being explored in order to find the best single individual." While this thought does not cover our algorithm well which makes use of all the individuals, it does point out the weakness of our original design.

In order to maintain a diverse population while applying the selective pressure, an evolutionary algorithm may adopt the speciation techniques [81, 82], by which individuals in a population will form different species automatically through evolution. Fitness sharing [18] refers to a class of such techniques. It works by reducing the payoff in densely-populated regions. Mathematically, consider an individual i with fitness f_i. Its niche count m_i measures how many other individuals with which i shares fitness. The shared fitness is f_i/m_i. The niche count is calculated with a distance metric d_{ij} that describes the difference between individual i and j. Usually $m_i = \sum_{\text{population size}} sh(d_{ij})$ where the sharing function $sh(d_{ij})$ makes distant individuals share less. Since those individuals in heavily explored regions will be less selected (due to reduced fitness), the evolution is encouraged to search diversified regions.

Fitness sharing has been successfully applied in neural network ensembles with different designs [55, 83, 84]. In our practice, the design in [55] will be adopted because of its simplicity and effectivity. Rather than based on calculating distances between chromosomes, it is based on the idea that those classifiers correctly recognizing the same pattern should share their fitness. The procedure of calculating shared fitness is therefore carried out example by example over the training set. If one training example is learned correctly by p individuals in the population, each of these p individuals receives fitness

$1/p$, and the rest of the individuals in the population receive zero fitness. Otherwise, all the individuals in the population receive zero fitness. The fitness is summed over all training examples. Compared with our original formulation of fitness (3.2) which only cares about classification accuracy, this design takes diversity into attention and encourages the individuals to cover different patterns in the training set.

Table 3.7: Mean and variance of classification error rate using our algorithm with fitness sharing. The test is on 20 data sets and is implemented for 50 times on every set. For comparison with Bagging, AdaBoost and our original design, see Table 3.2 and 3.3.

	Audiology	Automobile	Breast-Cancer-W	Diabetes
mean	0.1792	0.2067	0.0331	0.2378
variance	0.0036	0.0039	0.0012	0.0027
	German	Glass	Heart	Hepatitis
mean	0.2654	0.2782	0.1849	0.1685
variance	0.0032	0.0029	0.0030	0.0023
	Horse-Colic	Iris	Labor	Letter-Recognition
mean	0.1570	0.0681	0.1837	0.0401
variance	0.0021	0.0010	0.0023	0.0009
	Segment	Sonar	Soybean-Large	Splice
mean	0.0226	0.1855	0.0982	0.0704
variance	0.0012	0.0034	0.0020	0.0011
	Thyroid	Vehicle	Voting	Waveform
mean	0.0415	0.2409	0.1033	0.1280
variance	0.0009	0.0032	0.0025	0.0019

We have test our algorithm with this new feature on the 20 UCI data sets. All the other settings remain unchanged. The experimental results are recorded in Table 3.7. A close examination of the data shall confirm a remarkable improvement of our algorithm brought by fitness sharing: the algorithm now wins in 14 cases and the variance is also

reduced (referring to Table 3.2 and 3.3). To have a clearer view we consider using the mean ratio of error rates across domains as a measure of relative performance [50, 74]. However, Webb [77] reported that this measure could have the relative difficulty of error comparison in different domains. He gave the example that two algorithms x and y achieve error rates of 0.10 and 0.20 for domain A and 0.35 and 0.20 for domain B, respectively. The ratios will be 0.50 and 1.75, giving a mean ratio of 1.125, indicating that, averagely algorithm x has error 12.5% greater than algorithm y. But, if we consider the ratio of y to x, we get $0.20/0.10 = 2.0$ and $0.20/0.35 = 0.571$, giving a mean ratio of 1.286, indicating that, algorithm y on average has error 28.6% greater than algorithm x! Obviously it is not desirable to draw the conclusion based on the mean ratio of error rates. Webb therefore adopted the geometric mean ratio rather than the arithmetic mean ratio:

$$e^{\frac{\sum_{i=1}^{n} log(a_i/b_i)}{n}} = \sqrt[n]{\prod_{i=1}^{n} \frac{a_i}{b_i}}. \tag{3.8}$$

The geometric mean ratio of $\{a_i/b_i\}$ enjoys the desirable property that if it is greater than one then the geometric mean ratio of $\{b_i/a_i\}$ will be less than one. In the above example, the geometric mean ratio of x_i/y_i is 0.935 while the geometric mean ratio of y_i/x_i is 1.069, suggesting that x has the true advantage in terms of error reduction over the two domains.

By this reason we choose to calculate the geometric mean ratios of both classification error and variance for the improved version of our algorithm with respect to the original version and other two algorithms. The results are recorded in Table 3.8. In classification error, on average, our algorithm with fitness sharing performs the best, followed by the version without fitness sharing, AdaBoost and then Bagging. AdaBoost falls behind our original algorithm because it gives much worse solutions in a few cases (even though it is slightly better in many cases; see Table 3.2). In variance, Bagging is still the most steady

one, followed by our algorithm with fitness sharing, without fitness sharing and then AdaBoost. Fitness sharing brings variance reduction to our algorithm in 14 data sets (in other 2 sets the variances remain unchanged). After considering above observations in combination we would say that the proposed algorithm with fitness sharing achieves the best performance in the comparison.

Table 3.8: Geometric mean ratios of both classification error and variance for the improved version of our algorithm ("improved") with respect to Bagging, AdaBoost and the original version of our algorithm ("original").

Geometric mean ratio	$\frac{\text{improved}}{\text{Bagging}}$	$\frac{\text{improved}}{\text{AdaBoost}}$	$\frac{\text{improved}}{\text{original}}$
classification error	0.8714	0.9580	0.9655
variance	1.3273	0.8388	0.9207

3.3 Face Detection Using Evolutionary Ensembles

In this section the proposed algorithm (with fitness sharing) is applied to *face detection*, which has been regarded as a challenging problem and attracted much attention in the computer vision community (see [85] for a survey). We design a system to detect upright, frontal views of faces in gray-scale images. Such a system may find wide applications in real-life surveillance and biometric.

Usually face detection is treated as a binary classification problem. For every subregion in the image (say, 19×19 window), one extracts a feature vector and then applies a classifier which outputs a label $+1$ or -1, signifying the presence or absence of a face respectively. The principal component analysis and wavelet transform are often used for feature extraction. To detect faces larger than the window size, the image is repeatedly reduced in size by sub-sampling, and the classifier is applied at each size.

In our system the Haar wavelet feature [86] is used. The Haar transform provides a multiresolution representation of an image with wavelet features at different scales capturing different levels of detail: the coarse scale wavelets encode large regions while the fine scale wavelets describe smaller, local regions. The wavelet coefficients preserve all the information in the original image, but the coding of visual information differs from the pixel-based representation. The main motivation for using wavelets is that they capture visually plausible features of the shape and interior structure of objects that are invariant to certain transformations. The result is a compact representation where dissimilar example images from the same object class map to similar feature vectors. In the application, we do not use all the very fine scales of wavelets as features for learning since these scales capture high frequency details that do not characterize the class well. Similarly, the very coarse scale wavelets are not used as features for learning since they may not encode useful information. So, for this experiment, we only use the medium scales ($2 \times 2, 4 \times 4$) of wavelets as features for learning. These scales depend on the object class and the size of the training images and are chosen a priori. For more information about the wavelet transform, the readers are referred to [86, 87].

The training images including faces and non-faces are copied from the MIT CBCL Face Database [88]. Some of them are shown in Fig. 3.3. The proposed algorithm is used to learn an ensemble of 100 component classifiers, with decision stump as the base learning algorithm. A decision stump is a one-level decision tree which is easily trained [4]. It normally classifies a pattern by setting a threshold for just one item of the feature vector and therefore could only give weak classification (slightly better than random guess). However, we will see that the detection results obtained by the ensemble is quite satisfying. In the same setting (ensemble of 100 decision stumps), we also implement the Bagging and AdaBoost algorithms and compare them with our algorithm.

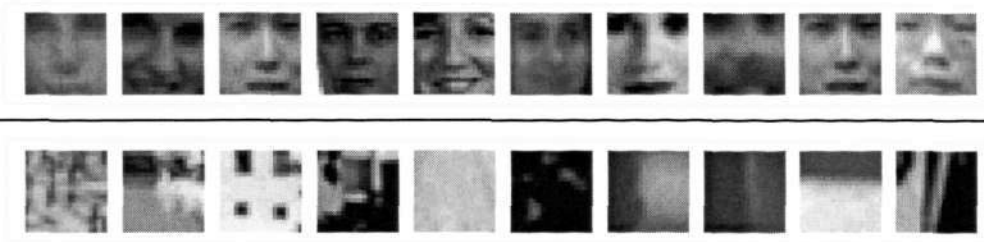


Figure 3.3: Some of the training images. In the upper row are faces; non-faces are in the lower row.

The algorithms are tested on the widely-used face detection database which is created by Rowley *et al.* [89] (available at http://www.cs.cmu.edu/~har/faces.html). It consists of 130 images with a total of 507 frontal faces. Most images contain more than one face on a cluttered background and, so, this is a good test set to assess algorithms which detect upright frontal faces. We summarize the results by the three algorithms in Table 3.9.

Table 3.9: Face detection results on images from the test set which contains 130 images with 507 faces. Bagging, AdaBoost and our algorithm (with fitness sharing) are compared. "faces detected " divided by 507 is the "detection rate". "false detections" is the number of wrong alarms which are actually not faces.

algorithm	**faces detected**	**detection rate**	**false detections**
Bagging	449	88.6%	83
AdaBoost	477	94.1%	32
our algo.	482	95.1%	31

The experimental results show that our algorithm achieves the highest detection rate of 95.1% with 31 false alarms. AdaBoost compares closely with detection rate 94.1% and 32 false detections. The Bagging algorithm, which randomly constructs the component classifiers, achieves clearly worse performance (detection rate 88.6% and 83 false detections). We further depict some of detection results by our algorithm in Fig. 3.4.

The detected faces are framed in white. In each of Fig. 3.4(A,C,D,I), there is only one face; while in the other subfigures there are several faces. The images in Fig. 3.4(D,I) have very low quality. In most of these cases our system detects the faces correctly. Also we obtain missed and false detections. There are one face in Fig. 3.4(B) and two faces in Fig. 3.4(F) missed by our system. This is probably due to the severe rotation involved. Another missed detection in Fig. 3.4(H) is due to the large occlusion on face. An apparent false detection appears in Fig. 3.4(F). The framed window in the upper-right area is rather like a face indeed. Without exploiting the contextual information, a classifier is easy to make an error here. In spite of all the classification errors mentioned above, they are also suffered from by many state-of-the-art face detection systems [85]. On the other hand, the classification accuracy can be improved by using more training images, adopting more salient features and incorporating special techniques to deal with the rotated faces (e.g. rotating in advance the subregions to be classified).

Again, the experiment here gives a vivid demonstration about the potential of the proposed algorithm in ensemble learning.

3.4 Summary and Limitations

In this chapter, we propose to incorporate evolutionary search into ensemble classification which works by manipulating training examples. There are basically two novelties in our algorithm:

a. Ensemble methods by manipulating training examples are directly identified as searching for appropriate weightings. In this view we propose to adopt a genetic algorithm. In the experiments the GA is seen providing ensembles with good

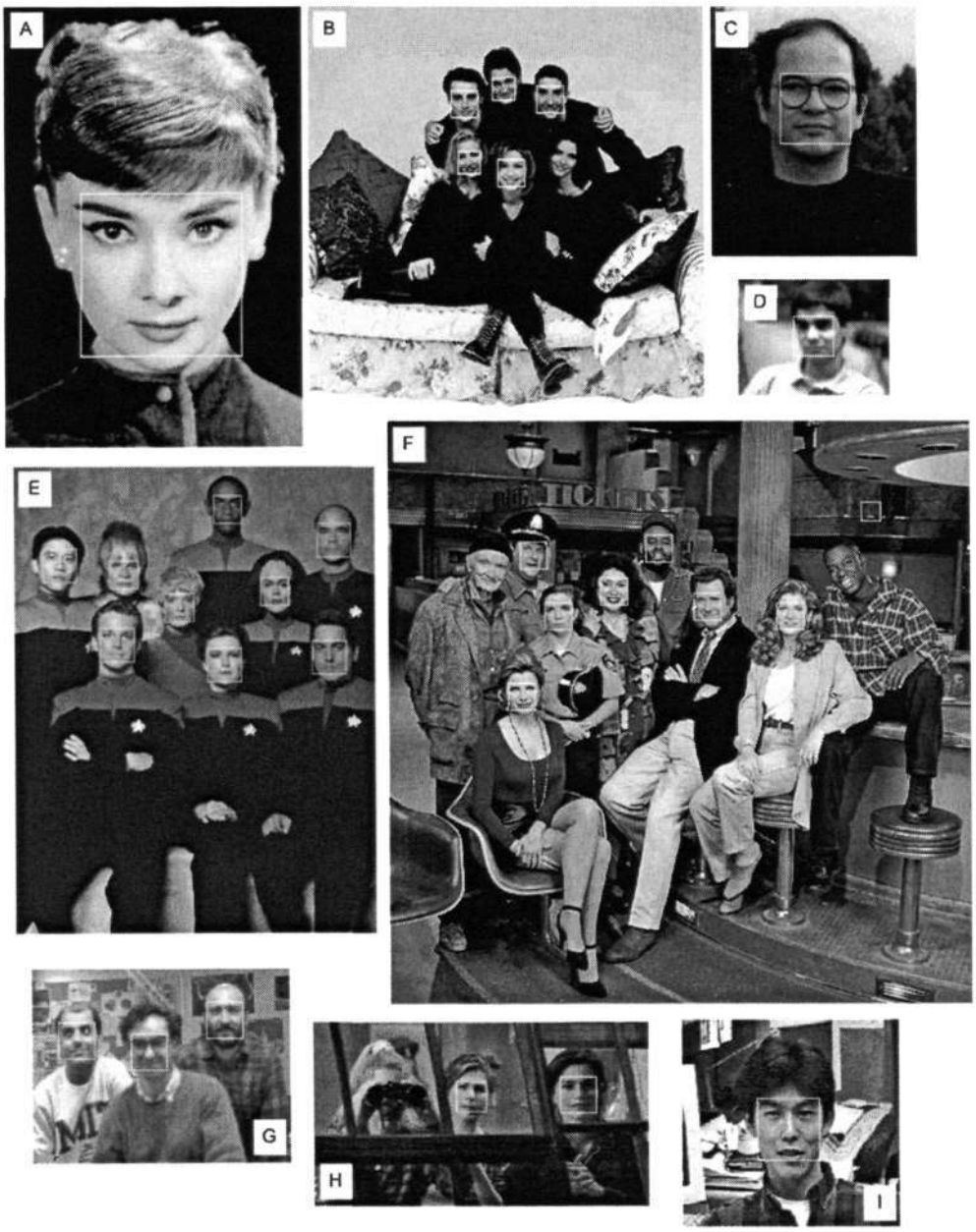


Figure 3.4: Some results on test images. The detected faces are framed in white. There are missed detections in (B,F,H), and a false detection in (F). The missed detections are due to severe rotation or large occlusion. The face detection is on a subregion which is quite like a real face.

generalization ability.

b. Chromosomes in the early generations are not simply discarded. All of them

will be exploited through their corresponding classifiers. The reason is that less optimized chromosomes may have complementary information about the feature space, therefore their inclusion is helpful to the ensemble.

Although the algorithm does not easily lead to a theoretical analysis of generalization ability due to the high nonlinearity and complicated dynamics, the experiments have justified its potential and usefulness. It is therefore a reasonable choice as a general-purpose algorithm for ensemble classification.

A limitation of the algorithm is about choosing appropriate individuals to form the ensemble. Currently we use all the individuals from the first to the last generation and it is based on the observation that the incorporation of less optimized chromosomes is helpful to good generalization. Still, in many applications we prefer small size ensembles for less classification time. It is therefore interesting to investigate if the size of the ensemble could be reduced without worsening the performance too much. In [90], Liu and Yao have experimented with choosing one representative from each species in the last generation. They use the k-means algorithm to cluster the individuals in the population (in terms of neural network output vector) to determine the species. The representative for each species is the fittest individual in the species. The number of species is determined by measuring the accuracy rates of those constructed ensembles (with different k) on the training set. They report that the ensemble does not have to use the whole population to achieve good performance and the size of the ensemble can be substantially smaller than the population size. By this result, the same idea could be implemented in our algorithm and over all the populations throughout evolution. The purpose is to improve the scalability of the algorithm with respect to the population size and number of generations. We identify this as a direction of future work.

Another limitation of the algorithm will concern the diversity of component classifiers in the ensemble. In [80] Brown *et al.* reported that "Bagging is an implicit method, it randomly samples the training patterns to produce different sets for each network; at no point is a measure taken to ensure diversity will emerge. Boosting is an explicit method, it directly manipulates the training data distributions to ensure some form of diversity in the base set of classifiers (although it is obviously not guaranteed to be the right form of diversity)." Along their thought, our algorithm is an implicit method which relies on random initialization, crossover/mutation and fitness sharing to encourage appropriate diversity. The first two mechanisms do not necessarily lead to diversified classifiers. Though fitness sharing is surely helpful, we expect that incorporating some explicit diversity generation and maintenance mechanisms such as negative correlation learning will bring further improvement. About this more discussion is in chapter 5.

Chapter 4

Evolutionary Optimization with Markov Random Field Prior

4.1 Introduction

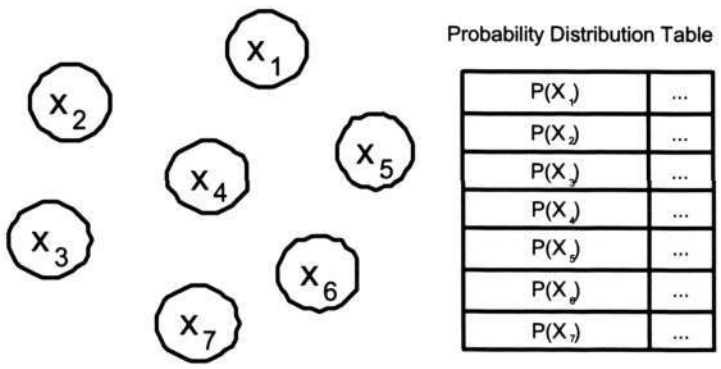
Following the discussion in section 2.4, this chapter focuses on the energy minimization problem in MRF modelling. We have seen the advantages and disadvantages of traditional algorithms like ICM and SA. We also discussed the possibility of employing genetic search in attacking this problem. However, since the genes are highly correlated with each other (through a two-dimensional structure of MRF), conventional genetic operators tend to be inefficient. More suitable and domain-specific designs are hence needed.

4.1.1 Estimation of Distribution Algorithms

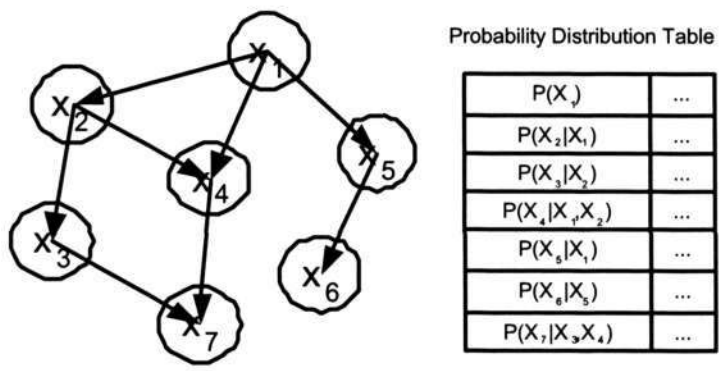
Recently, there has been a growing interest in evolutionary methods that learn the structure of the target problem on the fly and use this information to ensure a proper mixing and growth of building blocks. Algorithms after it are named estimation of distribution algorithms (EDAs) [91, 92]. EDA follows the general procedure of GA, i.e. maintaining a population of potential solutions for evolution. Each solution will also be assigned a fitness value to indicate how well it achieves the objective of the problem. However, the algorithm treats a solution not as a binary-coded chromosome, but as a combination of several component variables. The new population is no longer generated by applying crossover and mutation on some selected solutions. Instead, at every generation EDA selects a number of promising solutions (with above-average fitness) from the population and attempts to learn a probabilistic model from them. The model is about the probabilistic dependencies among the component variables of a solution and their distributions. Based on it a new population of solutions is sampled (usually of the same size as the previous population). Evolution in this manner is continued for several generations until some optimal solutions are found or the computational resource is exhausted.

Probabilistic dependencies among the component variables may have different levels of complexity. In Fig. 4.1 we have an illustration where each variable is depicted as a node. The simplest case is to assume that the variables are independent to each other (Fig. 4.1(a)) and their distributions can be estimated separately. The algorithms of PBIL [93], UMDA [94] and cGA [95] are in this case. More generally, when multivariate interactions exist, the model turns to be a Bayesian network in which variables are connected by directed lines (Fig. 4.1(b)). Estimation of distribution must rely on a belief inference algorithm [96]. The representatives are ECGA [97] and BOA [98]. Note that here we skip the case of pairwise interactions [99, 100]. After all, since the probabilistic

model is learned from some potential solutions which have higher fitness evaluations, it is expected to be a good prototype for promising areas in the search space. Researchers have reported that this procedure can capture the structure of the problem accurately and ensure a very effective mixing and reproduction of building blocks [91, 92].



(a)



(b)

Figure 4.1: Probabilistic models of the component variables of a solution. In this example we assume seven variables: $x_1, \cdots, x_7$. Each variable is represented by a node. (a)variables are probabilistically independent to each other; nodes are separate. (b)variables are mutually dependent; nodes are connected. Note that both the structure of the dependencies and the probability distribution table will be learned from some promising solutions. New solutions can be generated by sampling from the probability distribution table.

The general structure of EDAs is summarized as follows:

1. Set $t = 0$, randomly generate the initial population $Popu(0)$.
2. Select a set of promising solutions $Popu'(t)$ from $Popu(t)$.
3. Estimate the distribution $dist(t)$ from $Popu'(t)$ (a probability distribution table for component variables of a solution).
4. Generate a set of new solutions $Popu''(t)$ by sampling from $dist(t)$.
5. Create a new population $Popu(t+1)$ by replacing some solutions from $Popu(t)$ with $Popu''(t)$.
6. Set $t = t + 1$.
7. Check termination criteria. if satisfied, stop; otherwise go to step 2.

The essential idea behind this procedure is that performing search by some "blind", general-purpose strategies will be inefficient and if we can find a way to learn and exploit the domain-dependent knowledge, it will help the search to a large extent [101]. In the EDAs, the knowledge is learned from some promising solutions at every generation and encoded in a Bayesian network. Can we realize this idea when facing the specific problem which is modelled by Markov random field? A direct application of EDAs is infeasible since a MRF is an undirected graph based model upon which exact inference is infeasible [102]. But on the other hand, *a priori* knowledge in the form of Markovianity (the probabilistic relationships among component variables) exists for our problem. It need not be learned at every generation (the learning step may take a long time [96]). Can the knowledge be incorporated in a reasonable way to help achieving an efficient search? These questions will be discussed shortly.

We present a novel design of evolutionary algorithm in this chapter. Our design is inspired by the estimation of distribution algorithms. It still maintains a population of

potential solutions to the problem and they are evolved for several generations. But different from a genetic algorithm, a solution is not encoded into a chromosome. We notice that a solution is composed of a number of variables and these variables are not independent of each other but probabilistically correlated through a neighborhood structure. If a solution is represented as a chromosome, this prior knowledge may not be easily incorporated and further exploited. We also observe that the genetic operators of crossover and mutation are not suitable. They may not produce reasonable solutions because they may destroy the probabilistic dependencies among the component variables. In view of these difficulties our algorithm has several novelties.

- The variables $\{l_{(i,j)}\}$ of a solution L are partitioned into several groups named *codings*. A *pivot* coding is selected and variables in it are updated at every generation by sampling from some estimated probability distributions. The estimation of probability distribution is biased to those solutions in the previous generation who have higher fitness evaluations, so this step exerts an implicit selective pressure upon the population. Additionally, variables in the other codings are updated according to (2.17), which represents the contextual constraints. Another novelty of our algorithm is a flexible strategy to hybrid with local search. We suggest that the distributions used for variables in the pivot coding should encode not only the pressure from natural selection, but also the pressure from contextual constraints. The algorithm will appear more similar to an evolutionary search at the beginning while the very late stages will be characterized by Gibbs sampler-like local search.

In summary, the proposed algorithm is about building probabilistic models of the component variables in a solution and using these models to drive an evolutionary search.

The rest of this chapter is organized as follows. Section 4.2 introduces the proposed

algorithm in detail. In section 4.3 we discuss how to incorporate local search to help convergence. Experimental results on noisy and textured image segmentation are given in section 4.4. Finally, we conclude in section 4.5 and mention some future work.

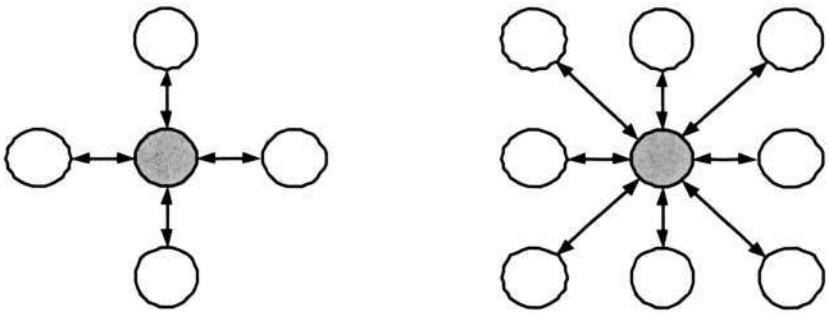
4.2 An Evolutionary Algorithm

In this section we first study the target problem in more detail. Special attention will be concentrated on the structure of a potential solution. After that the evolutionary algorithm is presented.

4.2.1 Study of The Structure of a Potential Solution

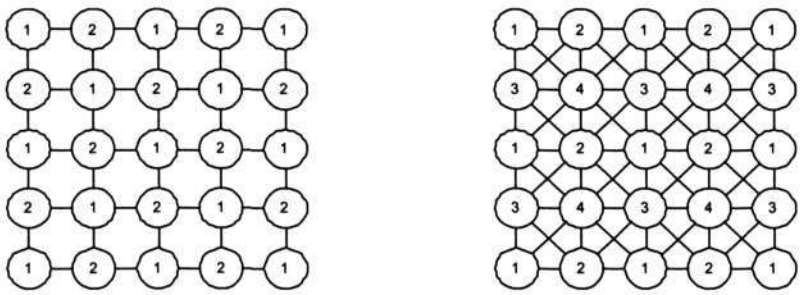
The posterior energy $U(L|D)$ is to be minimized. A potential solution $L = \{l_{(i,j)}|\forall(i,j) \in S\}$ has a very high dimensionality since each pixel (i,j) in the image is associated with an unknown variable $l_{(i,j)}$. With a genetic algorithm we generally do not pay much attention to the relationships among the component variables of a solution. But in our problem a distinctive point is that these variables are not scattered disorderly but correlated with each other through a two-dimensional neighborhood system. The neighborhood system is illustrated in Fig. 4.2. Here we only show the first-order and second-order neighborhood systems; for higher-order neighborhood systems, readers are referred to [2]. Under the assumption of Markovianity, only neighboring variables have direct interactions.

We partition a potential solution L into a few subsets L^k by the *coding* method [103]. These subsets are named codings and no two variables in one coding are neighbors. Fig. 4.3 gives an example. There are two codings for the first-order and four codings for the



(a) (b)

Figure 4.2: Definition of neighborhood systems. A node corresponds to a variable. The surrounding nodes are defined to be the neighbors of the center node which is grey color filled. (a)first-order. Each variable inside the image lattice has 4 neighbors (left, right, up and down); a variable on the border has 3 and a variable on the four corners has only 2 neighbors. (b)second-order. Each variable inside the image lattice has 8 neighbors (left, right, up, down and four diagonal directions); a variable on the border has 5 and a variable on the four corners has only 3 neighbors.



(a) (b)

Figure 4.3: Partition a potential solution L into a few codings according to the (a)first-order neighborhood, (b)second-order neighborhood. A node corresponds to a variable. Variables marked different numbers k belong to different codings L^k.

second-order neighborhood system. Variables neighboring each other are from different codings and are connected by lines to signify that they are probabilistically dependent (here lines are undirected since associations among variables are considered correlational rather than causal in the case of a Bayesian network; compare to Fig. 4.1(b)). Variables from the same coding are not connected because they are mutually independent given

the values of variables from other codings. Mathematically, we have

$$P(L^k) = \prod_{l_{(i,j)} \in L^k} P(l_{(i,j)} | l_{N_{(i,j)}}). \tag{4.1}$$

This is the first characteristic of the structure of a potential solution in our problem.

Secondly we study the conditional probability (2.17) further. From now on we will focus on the second-order neighborhood system and only up to pair-site cliques are taken into account (a clique is a set of neighboring variables [2]). This is explained clearly in Fig. 4.4. We employ the generalized Ising model [71]. In this model, the single-site parameter α will be different for different labels of variable $l_{(i,j)}$ (an indication of different regions). The pair-site parameter β will have four values $(\beta_1, \beta_2, \beta_3, \beta_4)$ for cliques on the four different directions (i.e. horizontal, vertical and two diagonal; see Fig. 4.4). We define the *clique potentials* as

$$V_1(l_{(i,j)}) = \alpha_m, \text{ if } l_{(i,j)} == m, \ m \in \{1, ..., M\} \tag{4.2}$$

for single-site clique and

$$V_2(l_{(i,j)}, l_{(i',j')}) = \begin{cases} \beta, & \text{if } l_{(i,j)} = l_{(i',j')} \\ -\beta, & \text{otherwise} \end{cases} \tag{4.3}$$

for pair-site clique. The local prior energy is then written as

$$U(l_{(i,j)} | l_{N_{(i,j)}}) = V_1(l_{(i,j)}) + \sum_{(i',j') \in N_{(i,j)}} V_2(l_{(i,j)}, l_{(i',j')}). \tag{4.4}$$

It is easy to see that the behavior of this prior model is adjustable through the parameters involved: α and β. For example, in image segmentation we often assume

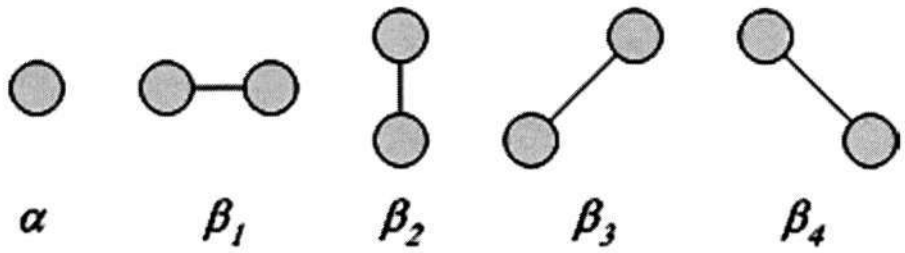


Figure 4.4: Single-site and pair-site cliques and associated parameters in the second-order neighborhood system. One node corresponds to one variable. Nodes are connected because they are probabilistically dependent on each other.

neighboring pixels have a high probability to belong to the same region. To encode this prior knowledge, we can let β take a negative value to cause a negative energy when neighboring variables have the same value and otherwise a positive energy (see (4.3) and (4.4)). Since lower energy is always preferred (it means a higher probability), the model tends to produce homogeneous regions by assigning the same value on neighboring variables. The values for different α_m are usually set same since no knowledge exists on the size ratios among different regions.

Local posterior energy is

$$U(l_{(i,j)}|d_{(i,j)}, l_{N_{(i,j)}}) = U(l_{(i,j)}|l_{N_{(i,j)}}) + U(d_{(i,j)}|l_{(i,j)}), \tag{4.5}$$

in which

$$U(d_{(i,j)}|l_{(i,j)}) = (d_{(i,j)} - \psi(l_{(i,j)}))^2/(2\sigma^2) \tag{4.6}$$

is the local likelihood energy. Extended from (2.17), the conditional posterior probability can be computed by

$$P(l_{(i,j)}|d_{(i,j)}, l_{N_{(i,j)}}) = \frac{e^{-U(l_{(i,j)}|d_{(i,j)}, l_{N_{(i,j)}})}}{\sum_{l_{(i,j)}=1}^{M} e^{-U(l_{(i,j)}|d_{(i,j)}, l_{N_{(i,j)}})}}. \tag{4.7}$$

That is, given the values of the neighboring variables $l_{N_{(i,j)}}$ and the observation data $d_{(i,j)}$, we can deduce the probability distribution of variable $l_{(i,j)}$ over the value set $\{1, ..., M\}$ by (4.7). This equation of conditional probability encodes the contextual constraints in a computable form. It is the second characteristic to be exploited.

The target function of posterior energy (2.22) in our problem can be equivalently written as the sum of local posterior energy

$$U(L|D) = \sum_{(i,j)} U(l_{(i,j)}|d_{(i,j)}, l_{N_{(i,j)}}). \tag{4.8}$$

To summarize the study on the structure of the problem, we find that it is readily represented by a probabilistic model. In this model, variables in one coding are independent to each other given the variables in other codings. Furthermore, the conditional probability among neighboring variables can be computed through (4.7). These two characteristics will be exploited smoothly in the proposed algorithm.

4.2.2 Algorithm Description

We may attempt to use a genetic algorithm to solve the optimization problem. To maintain the neighborhood structure of a solution, we may even adopt a two-dimensional chromosome [45]. But unfortunately, it is inappropriate to produce new potential solutions just by randomly interchanging some parts in the parent solutions. The "blind" crossover operator can not assure that the newly generated solutions comply with the contextual constraints in (4.7). So the new generation of solutions may not be averagely better than the previous generation. That is, the GA may not work well in our case. On the other hand, the EDAs demonstrate that we need not be confined in the schemes of genetic recombination; the evolution can be driven in a manner by learning and building

probabilistic models of a potential solution.

Motivated by these considerations we develop a novel evolutionary algorithm. Suppose we maintain a population $Popu(t)$ of potential solutions at generation t. The algorithm follows:

1. Set $t = 0$, randomly generate the initial population $Popu(0)$.
2. Set $t = t + 1$, $Popu(t) = Popu(t - 1)$
3. Estimate the probability distribution $dist(t)$ for variables in the pivot coding.
4. Update variables in the pivot coding by sampling from $dist(t)$.
5. Update variables in other codings sequentially by sampling from their conditional probabilities.
6. Check termination criteria. if satisfied, stop; otherwise go to step 2.

At every generation, the new population is generated from the current one by applying some statistical rules. We will explain this scheme from three aspects.

4.2.2.1 Representation and Fitness Evaluation

Representation of a potential solution and measuring its fitness are two fundamental issues in designing an evolutionary algorithm. In a traditional GA, a solution is usually encoded into a binary string. This one-dimensional chromosome is not appropriate for our problem. Here, a solution is a two-dimensional array of variables $\{l_{(i,j)}\}$ over the image lattice. This fact and the need to include region information prompt us to use an alternative representation. In [45], Bhandarkar and Zhang use directly the two-dimensional array as a chromosome; see Fig. 4.5. We employ a similar mechanism in

this thesis. That is, each individual in the population will be an instance of the random field $L = \{l_{(i,j)} | \forall (i,j) \in S\}$.

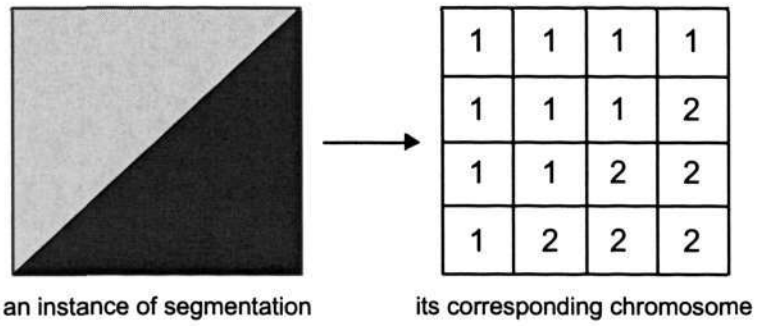


Figure 4.5: Representation. Each chromosome is a two-dimensional array. In this example the image is 4×4 and there are only two regions in it. So the component variables take their values from set $\{1, 2\}$.

For each of these potential solutions, the posterior energy (4.8) (or (2.22)) is a natural indication of its fitness, as a lower energy corresponds to a better solution. Nevertheless the energy value could be negative and a fitness mapping is demanded. Currently we use a simple, linear one

$$F(.) = U_{max} - U(.), \tag{4.9}$$

in which $U(.)$ is the posterior energy of a solution, U_{max} is the maximum energy in the population and $F(.)$ is the fitness value. We have tried more sophisticated mappings, and the performance does not change remarkably.

4.2.2.2 Evolution of The Pivot Coding

In a GA, crossover is the main strategy used to produce offsprings. In that way, two chromosomes are picked from the current population and parts of information are exchanged between them. For two-dimensional chromosomes, a two-dimensional crossover can be employed [45] (see Fig. 4.6). But here, this general-purpose, two-parent operator

is not used.

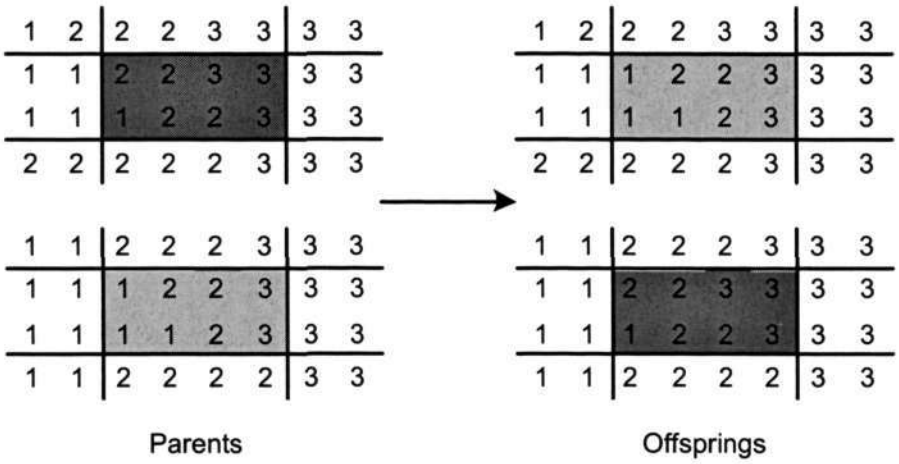


Figure 4.6: Two-dimensional crossover for image segmentation. A rectangular subregion in the image lattice is randomly determined, and the labels within the subregion are exchanged between the two parent chromosomes.

We recall that the variables in a potential solution are partitioned into 4 codings (second-order neighborhood system, see Fig. 4.3(b)). In our algorithm these four codings will be recognized as one pivot coding plus three other codings. The pivot coding can be specified arbitrarily from the four codings; so suppose we pick coding 1. At this step we have the variables in coding 2, 3 and 4 fixed. The variables in coding 1 are thus probabilistically independent of each other. That means we can estimate their distributions separately. To do so, first we compute a fitness-weight

$$FW(.) = \frac{F(.)}{\sum_{u \in Popu(t)} F(u)} \tag{4.10}$$

for each solution in generation t. Since $F(.)$ is the fitness for a potential solution, $FW(.)$ is a comparative indication of the goodness of the solution in the current generation. Then for each variable $l_{(i,j)}$ in the pivot coding, its probability to take value m in the next generation is approximated by summing up the corresponding fitness-weight values

over all solutions that currently have value m at pixel (i, j). That is,

$$P(l_{(i,j)}^{t+1} = m) = \sum_{u \in popu(t)} FW(u | l_{(i,j)}^{t} == m). \tag{4.11}$$

$P(l_{(i,j)}^{t+1} = m)$ denotes the probability that the variable $l_{(i,j)}$ will take value m in generation $t+1$. It is easy to see that $\sum_{m=1}^{M} P(l_{(i,j)}^{t+1} = m) = 1$. Based on (4.11) if those solutions with $l_{(i,j)} == m$ in the current generation have averagely higher fitness, m will be more possible to appear at (i, j) in the next generation. This step is thus seen biased to solutions with higher fitness.

We use a more concise symbol $P_{(i,j)}(m)$ to denote this probability. The vector $\overrightarrow{P}_{(i,j)} = [P_{(i,j)}(1), ..., P_{(i,j)}(M)]$ is an estimated distribution on the discrete set $\{1, ..., M\}$ and is finally obtained for all variables $l_{(i,j)}$ in coding 1 (the pivot coding). Each solution in the population will then have its variables in coding 1 updated by new values sampled from the distribution $\overrightarrow{P}_{(i,j)}$. Fig. 4.7 shows an example. In this step variables in the other codings remain unaltered.

The strategy used here is similar to the bit-based simulated crossover (BSC) operator [104]. In BSC the bits in a chromosome are assumed to be independent of each other (similar to Fig. 4.1(a)) and are evolved by sampling from some estimated population-based statistics. This kind of operation is theoretically justified here because of the first characteristic of solution structure [see (4.1)].

Since each solution's contribution to $\overrightarrow{P}_{(i,j)}$ is weighted by its fitness with respect to the fitness values of all the other solutions in the population, selective pressure is encoded and exerted implicitly. Better solutions contribute more to the distribution vector. Therefore their values for variables in the pivot coding will appear more possibly in the next generation. From another viewpoint, the probability distribution $\overrightarrow{P}_{(i,j)}$ over coding

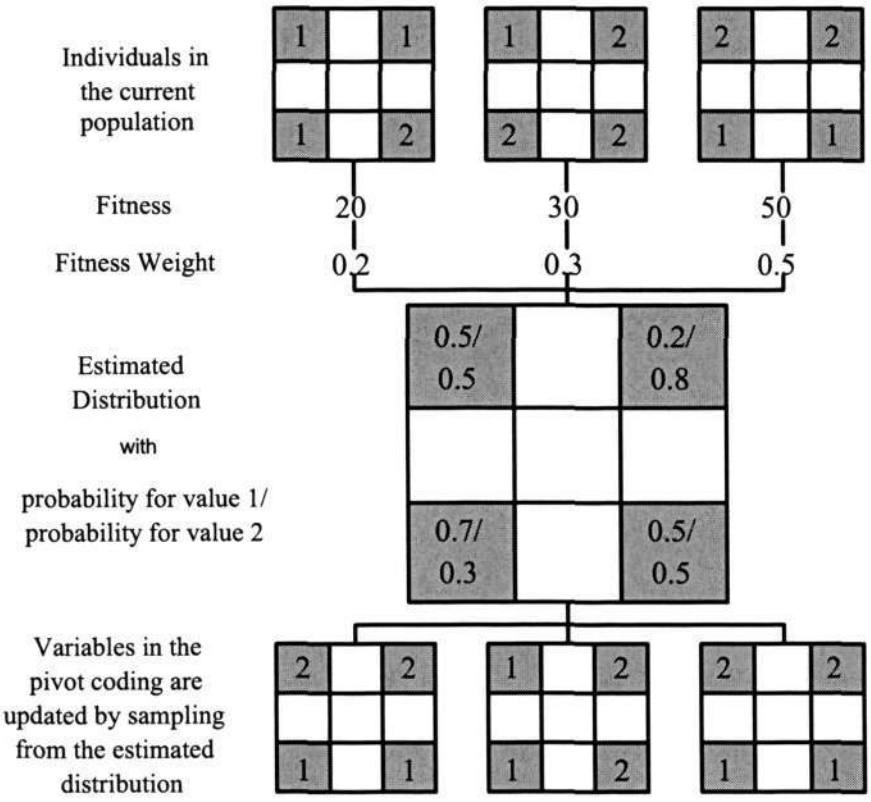


Figure 4.7: Evolution of the pivot coding. This procedure is from top to down. We assume image size of 3×3, only 3 individuals in a population and only two regions in the image. For a clear illustration we show the variables in the pivot coding in grey color filling; variables in the other codings are not displayed. In the procedure we first calculate the fitness and fitness-weight values. Then we estimate the probability distributions for those variables in the pivot coding. The estimation is biased towards those solutions in the current generation which have higher fitness. Next we draw random samples from the probability distributions to update those variables in the pivot coding.

1 can be considered as a prototype for a higher-evaluation area in the search space; the next advance of the population will roughly follow this prototype. That is why we call coding 1 the pivot coding. Note that at this moment the population has already gotten new, probably more promising, values in coding 1; while variables in coding 2, 3 and 4 still keep old values which are inherited from the previous generation. Evolution is just implemented partially.

4.2.2.3 Evolution of Other Codings

Variables in coding 1 have already been updated, now we need the selective pressure be transferred to other codings. Simultaneously updating coding 2, 3 and 4 is infeasible because of the bidirectional interactions. They should be evolved in a sequential manner.

The updating rule here will be different from the evolutionary one used on the pivot coding. First we let the variables in coding 1, 3 and 4 be fixed. For each solution in the population, the conditional probabilities of variables in coding 2 are calculated by (4.7). According to each of these probabilities, a new value is sampled and assigned to its associated variable. In this way we finish updating variables in coding 2 for that solution. The same operation is conducted on the other solutions to update the variables in coding 2. After that, variables in coding 3 and 4 will be updated similarly, e.g. we let variables in coding 1, 2 and 4 be fixed and update variables in coding 3 by their conditional probabilities. After the variables in coding 4 are also updated, the new solution appears as a whole.

The rule used here draws its rationality from the second characteristic of our problem, i.e. (4.7). It confines the search in a reasonably narrow area around the direction which is decided by the pivot coding. Additionally, we can also consider this step as a partial multi-start Gibbs sampler [71]. In a cycle of a classical Gibbs sampler, only one potential solution is manipulated and all the variables in it will be updated by drawing new samples from their conditional probabilities. In our case, there is a population of potential solutions and each solution only have part of its variables (excluding the variables in the pivot codings) updated according to their conditional probabilities.

In a summary our algorithm performs an evolutionary search on the pivot coding and a model-based search on the other codings. Perhaps closest in spirit to our approach

is the work done by Lai and Vemuri [47]. They use a Boltzmann weighted selection operator and a Gibbs sampler based mutation operator. The resulting algorithm is thereby considered "informed" because it incorporates specific domain knowledge into the genetic operators. Our algorithm, however, follows a more innovative approach. This synergetic combination of evolutionary and knowledge-based considerations constitutes one contribution of our algorithm.

The steps of updating pivot coding and other codings are repeated for several generations. The convergence is declared when the posterior energy of the best solution in the population has not changed over the past few generations. A fast convergence is preferred, but some preliminary experiments show that the convergence speed of the algorithm introduced so far is very slow. The same phenomenon is also observed by Bhandarkar and Zhang [45] for a GA. This phenomenon is often explained by the very high dimensionality of the problem in image analysis. We have tried an *elitist* strategy, but it only increases the speed a little. In the next section we propose a simple way to incorporate local search into the current algorithm, and it is found to accelerate the convergence to a satisfactory extent.

4.3 Hybridization with Local Search

The necessity of incorporating local search is advocated by many other practitioners. Bhanu *et al.* [46] improve the best chromosome in every generation using a hill climbing strategy. Yu *et al.* [48] apply morphological operation to selected chromosomes before they are subject to crossover and mutation. Bhandarkar and Zhang [45] resort to the engineered conditioning (EC) operator. The EC operator has two parameters: e_1 which determines how many chromosomes to be influenced and e_2 which denotes how many

variables in the chosen chromosome will be improved. e_1 and e_2 are given small value like 0.1 initially and are increased gradually towards 1 which means that all chromosomes are conditioned over all variables. So the very late stages of the GA are characterized by a large number of local hill climbing moves. They argue that in the later generations of evolution it is highly likely that most of the chromosomes in the population lie in the vicinity of an optimum and in this situation it is imperative for the algorithm to focus on local search.

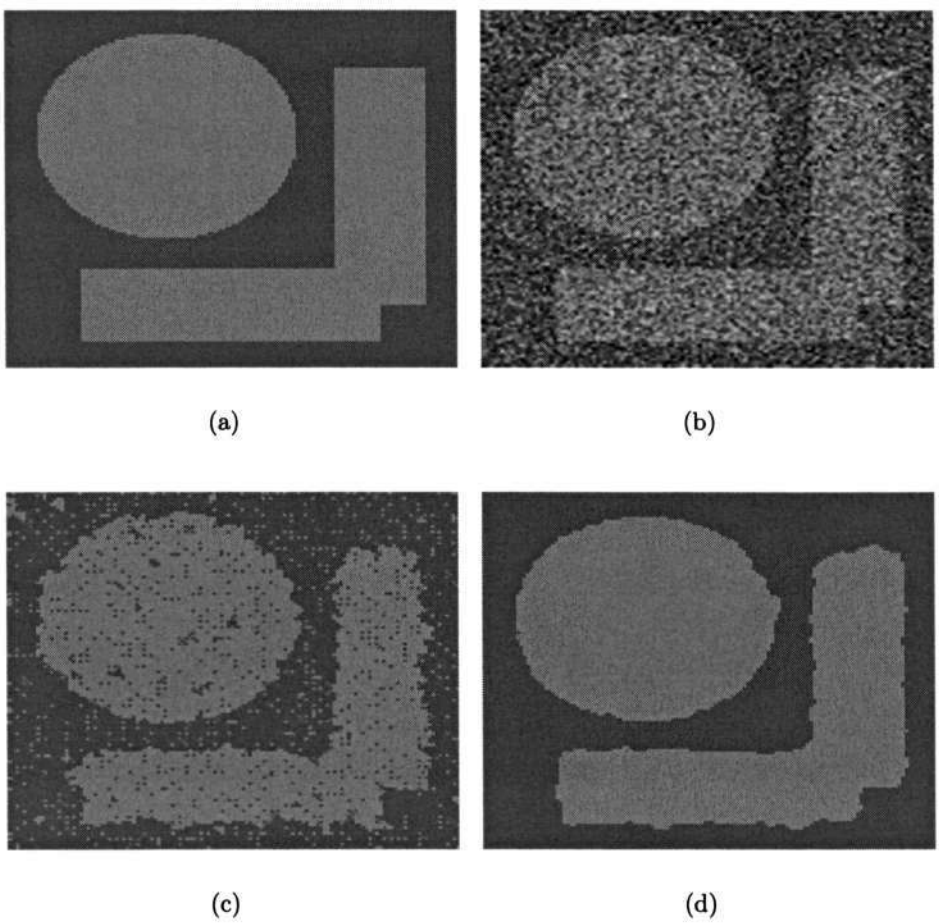


Figure 4.8: Hybridization with local search. (a) original image. (b) noisy image. (c) segmentation by pure evolutionary search. (d) segmentation by hybrid search. The two kinds of search are implemented for the same number of generations.

The method proposed here is inspired by the EC operator, but it takes a more conve-

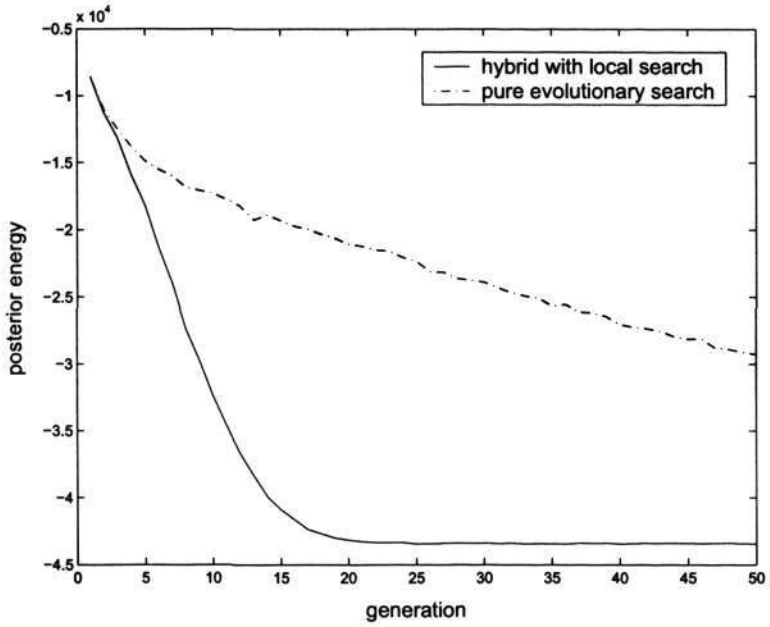


Figure 4.9: Convergence curves with and without local search.

nient way. Recall from the previous section that we estimate a probability distribution for each variable in the pivot coding. The key to our method are these probability distributions. Currently they are estimated from the fitness values and encode only the selective pressure. They represent our prediction of how the variables should behave in the next generation. We use $\overrightarrow{P}^{evo}_{(i,j)}$ to denote one of these distributions. Note that it is computed over a whole population by (4.11).

Now suppose a totally different scheme is adopted whereby each solution has its variables in the pivot coding updated by sampling from their conditional probabilities. A conditional probability $\overrightarrow{P}^{con}_{u;(i,j)}$ for variable $l_{(i,j)}$ of solution u is calculated by (4.7) and only encodes the contextual constraints. Since variables in the other codings are also evolved by this rule, the algorithm then turns to be a multi-start Gibbs sampler [71], i.e. each solution is subject to a series of local improvements based on its own statistics.

A simple yet effective way to incorporate such Gibbs sampler-like local search is to use

a weighted sum of the two distributions:

$$\overrightarrow{P}_{u;(i,j)} = \lambda \times \overrightarrow{P}^{con}_{u;(i,j)} + (1-\lambda) \times \overrightarrow{P}^{evo}_{(i,j)}, \tag{4.12}$$

in which $\lambda \in [0,1]$. At every generation of evolution, we will update variable $l_{(i,j)}$ in the pivot coding of solution u by sampling from this hybrid distribution $\overrightarrow{P}_{u;(i,j)}$. The parameter λ should not be fixed during the run of algorithm. We can adjust it at every generation to embody different judgements - which distribution should be used more, the evolutionary one or the context-conditional one. If λ is given a small value, $\overrightarrow{P}^{evo}_{(i,j)}$ will possess a big weight and the evolutionary global search is paid more attention; otherwise when λ is set high, $\overrightarrow{P}^{con}_{u;(i,j)}$ is more weighted and the context based local search is focused.

The convenience also brings along the problem of how to tune the parameter λ. A lot of heuristics of parameter control exist in the literature (see the survey in [105]). In our implementation a linear model is employed:

$$\lambda(i) = i/\text{NUM\_GENS}, \text{ for } i = 0...\text{NUM\_GENS}, \tag{4.13}$$

where NUM_GENS denotes the number of generations predefined. Initially $i = 0$ and the algorithm is totally evolutionary. At the last generation where $i =$ NUM_GENS, all potential solutions are subject to local search only. Our algorithm is hence considered as a hybridization of those two kinds of search.

We include here an experimental comparison to demonstrate the effect of local search on helping convergence. The task is to segment a degraded image (Fig. 4.8(b)) based on a MRF model. The results by performing a pure evolutionary search and a hybrid search both over 50 generations are shown in Fig. 4.8(c) and 4.8(d) respectively. A visual comparison of segmentation quality shows that a hybrid algorithm does much

better. For a further study, Fig. 4.9 depicts the corresponding curves of convergence (posterior energy/generation). While a pure evolutionary search appears to demand a large number of generations of evolution, a hybrid search is seen to have a desirable convergence rate.

4.4 Experimental Results

Besides the low-level tasks like image segmentation and reconstruction addressed previously, MRF models also find application in high-level tasks such as object matching and pose estimation [106, 107]. In these tasks MRF usually works on irregular relational graphs instead of regular image lattice [2]. Irrespective of particular applications, the proposed algorithm can be employed as long as the solution is modelled by a Markov random field. In our experiments we choose image segmentation as the illustrative application, not only because of its importance, but also because of its representativeness. Since we are focusing on evaluating the quality of the evolutionary algorithm in attacking the energy minimization problem, we compare it with two popular algorithms also in MRF modelling, i.e. ICM [65] and SA with the Gibbs sampler [71]. Specific algorithms which are restricted to particular applications are not used as benchmark.

The comparison is based on visual quality and the achieved lowest energy. For cases with ground truth, the rate of segmentation error is also computed. We will ignore the single-site potentials in (4.2). The pair-site potential parameters β are set by trials. For SA, the cooling schedule is set to $T^{t+1} = 0.99T^t$ with $T^0 = 10^4$. In implementing our algorithm, the population size is set to 100 and the number of generations has an upper-bound 50. Our algorithm has no parameters of crossover probability and mutation probability. The λ in (4.12) is determined adaptively during the evolution. We recognize

that the three algorithms (including ICM) all start search from random initializations. Therefore we run them independently for 30 times on each of the four images, recording the mean and variance, hopefully making a fair comparison.

The first experiment is on a noisy chessboard image (size 128×128, Fig. 4.10(b)). It is obtained by adding i.i.d Gaussian noise with standard deviation 60 (SNR=1) on the original image (Fig. 4.10(a)). The β parameters are set to $(-0.4, -0.4, -0.4, -0.4)$. The second experiment is on a noisy Lenna image (size 256×256, Fig. 4.11(b)). It is obtained by adding i.i.d Gaussian noise with standard deviation 25 on the original image (Fig. 4.11(a)). In this case $\beta = (-1, -1, -1, -1)$ and we perform a three-level segmentation of gray value 40,125 and 200. Note that in both cases we assume Gaussian noise to facilitate the use of a simple likelihood model. Practitioner may extend our algorithm straightforwardly with other likelihood models.

The values of achieved posterior energy over 30 runs are recorded in Table 4.1. The mean values of the energy and variances are also computed, which suggest that SA and our algorithm perform closely while ICM drops behind. For a clearer view we depict their curves of energy in Fig. 4.12. The following observations are thereby summarized:

- ICM converges to different results based on different initializations. The results are generally much worse than those of SA and our algorithm, and they exhibit much stronger fluctuation.
- The results of SA and our algorithm changes more smoothly over the runs. They are significantly better than those of ICM.
- The curves of SA are closely below the curves of our algorithm. So SA performs a little better than our algorithm, but the difference is negligible. The results of our algorithm are comparable to those of SA.

- Our algorithm does not depend heavily on the initializations. It is capable of jumping over local optima in favor of a global one.

Besides the objective comparison, we select those segmentation results of three algorithms which correspond to the lowest posterior energy over the runs (which could be found in Table 4.1), and depict them in Fig. 4.10(c-e) and Fig. 4.11(c-e). In the "Chessboard" image, ICM obtains an error rate of 11.73% while the values for SA and our algorithm are 2.34% and 2.59% respectively. Not surprisingly we find that ICM gets stuck in undesirable segmentations which have several spurious labels inside homogeneous regions; on the other side SA and our algorithm achieve rather good segmentations.

The next two experiments are on textured image segmentation. We employ here the hierarchical MRF model [68] with a high level MRF for the blob-like regions and low level MRFs for the filled textures. Fig. 4.13(a) shows a handwritten numeral image of size 128×128 which is filled with two synthetic textures generated by MRF models with $\beta = (0.5, -0.5, 0.5, -0.5)$ and $(-0.5, 0.5, -0.5, 0.5)$. The high level MRF is with $(-0.4, -0.4, -0.4, -0.4)$. In Fig. 4.14(a), the image size is 256×256 and there are four kind of textures with β parameters $(-1, -1, -1, 1)$, $(-1, 1, 1, 1)$, $(2, -1, 0.5, 0.5)$, $(1, 1, 1, 1)$. The high level MRF is with $(-1.5, -1.5, -1.5, -1.5)$. We show the perfect segmentations in 4.13(b) and 4.14(b) respectively. The textures can be seen to have different patterns.

The values of achieved posterior energy over 30 runs of three algorithms are recorded in Table 4.2. Like in the previous experiments, we depict the curves of energy in Fig. 4.15. Some similar situations appear as in noisy image segmentation, although this time the curves of SA and our algorithm seem to fluctuate more intensely.

Again we select the best segmentation results of each algorithm and depict them in Fig.

4.13(c-e) and Fig. 4.14(c-e). For the "Texture 1" case, ICM obtains an error rate of 4.88% while the values for SA and our algorithm are 1.13% and 1.62%. In Fig. 4.13(c) we see three obvious artefacts: one near the symbol of "8", one on the left border and the other at the bottom of symbol "1". We also see many irregular contours in Fig. 4.14(c) which is the ICM segmentation of "Texture 2". The segmentations of SA and our algorithm are quite near to the perfect one. They are not easily ranked by eye. For this case, ICM obtains an error rate of 5.31% while the values for SA and our algorithm are 1.74% and 2.09% respectively. The subjective evaluation of segmentation quality agrees with the classification error and the achieved posterior energy.

Due to the high nonlinearity and complicated dynamics of evolution, the algorithm does not easily lead to a rigorous theoretical analysis. Still, we notice that the solution quality of the proposed algorithm is rather close to that of SA. The experiments that we have conducted demonstrate its potential as a global method in searching the astronomical and complicated space modelled by Markov random field. This potential can be ascribed to the novel idea of guiding evolution by building probabilistic models of potential solutions.

4.5 Summary and Limitations

This chapter is about designing an evolutionary algorithm for function optimization in which a potential solution consists of a large number of variables forming a two-dimensional neighborhood structure. The structure is readily modelled by a Markov random field. A conventional genetic algorithm may not work well in this situation.

We are partially inspired by the estimation of distribution algorithms. For evolution each variable in a solution will be updated by sampling from its probability distribu-

tion. Although we may calculate the conditional probability by (4.7), it encodes only the contextual constraints from neighboring variables. We notice that the variables can be partitioned into several codings and variables in the same coding are conditionally independent. Therefore to encode the selective pressure that better solutions reproduce more of their *building blocks*, we select a pivot coding and for the variables in it we introduce fitness-weighted calculations into their distributions. The contextual constraints represent local exploitation while the evolutionary rules represent global exploration. Our algorithm combines together the power of both kinds of search. Some experiments on noisy and textured image segmentation have demonstrated its capability.

However, the algorithm suffers from several limitations. First, although the convergence is accelerated through hybridization with local search, in the experiments we find the algorithm needs much more computation time than the local method of ICM. While it is generally much faster than SA, these global methods are not suitable for high speed applications. Second, the algorithm relies on assumptions about the number of categories (recall the M in section 2.4) and the clique potential parameters which may not be satisfied in practice. In view of this limitation there have been works on simultaneous parameter estimation and energy minimization by Markov Chain Monte Carlo (MCMC) like methods [2]. A similar exploration by incorporating model parameters into the evolutionary search would give a further support to the usefulness of the proposed algorithm.

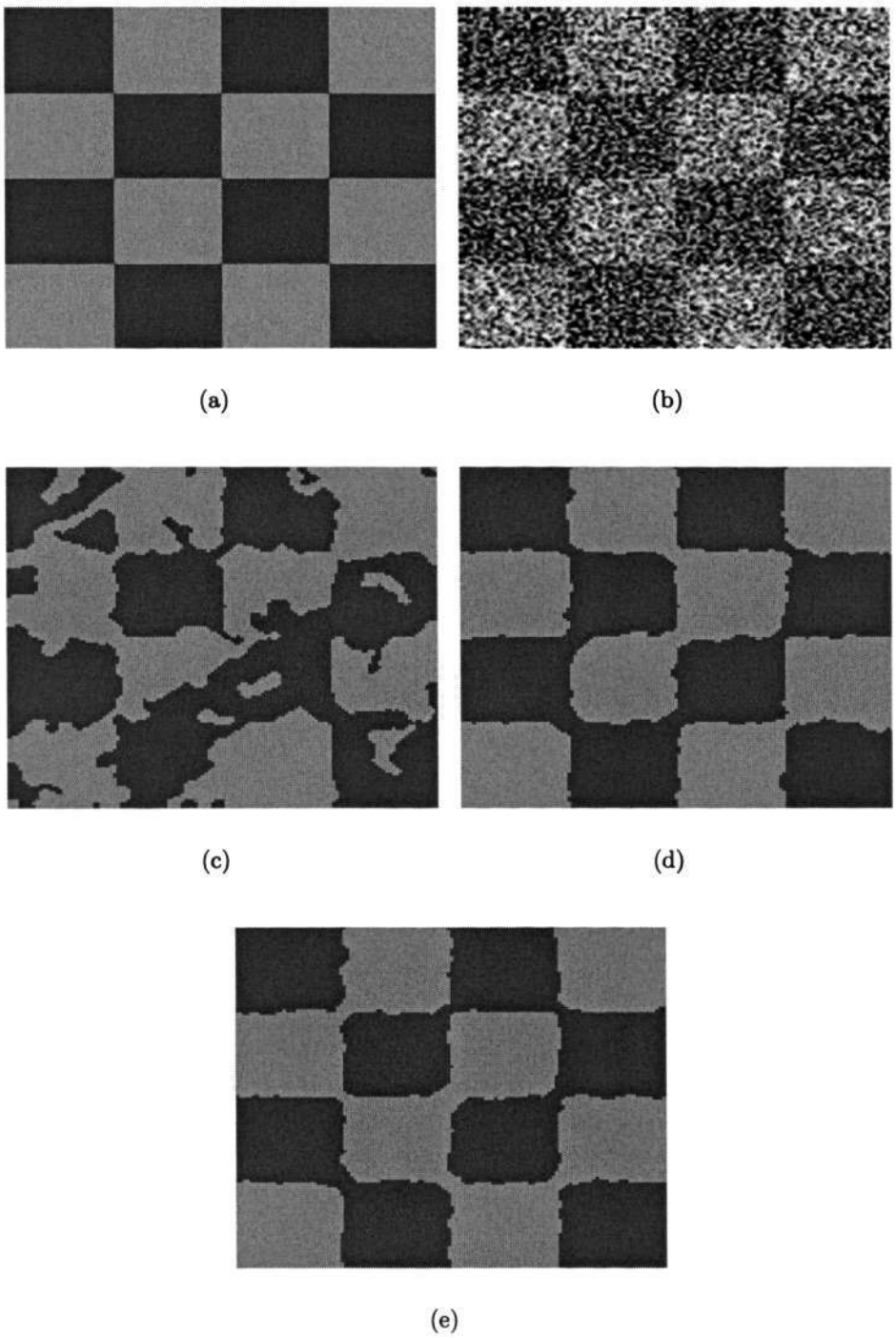
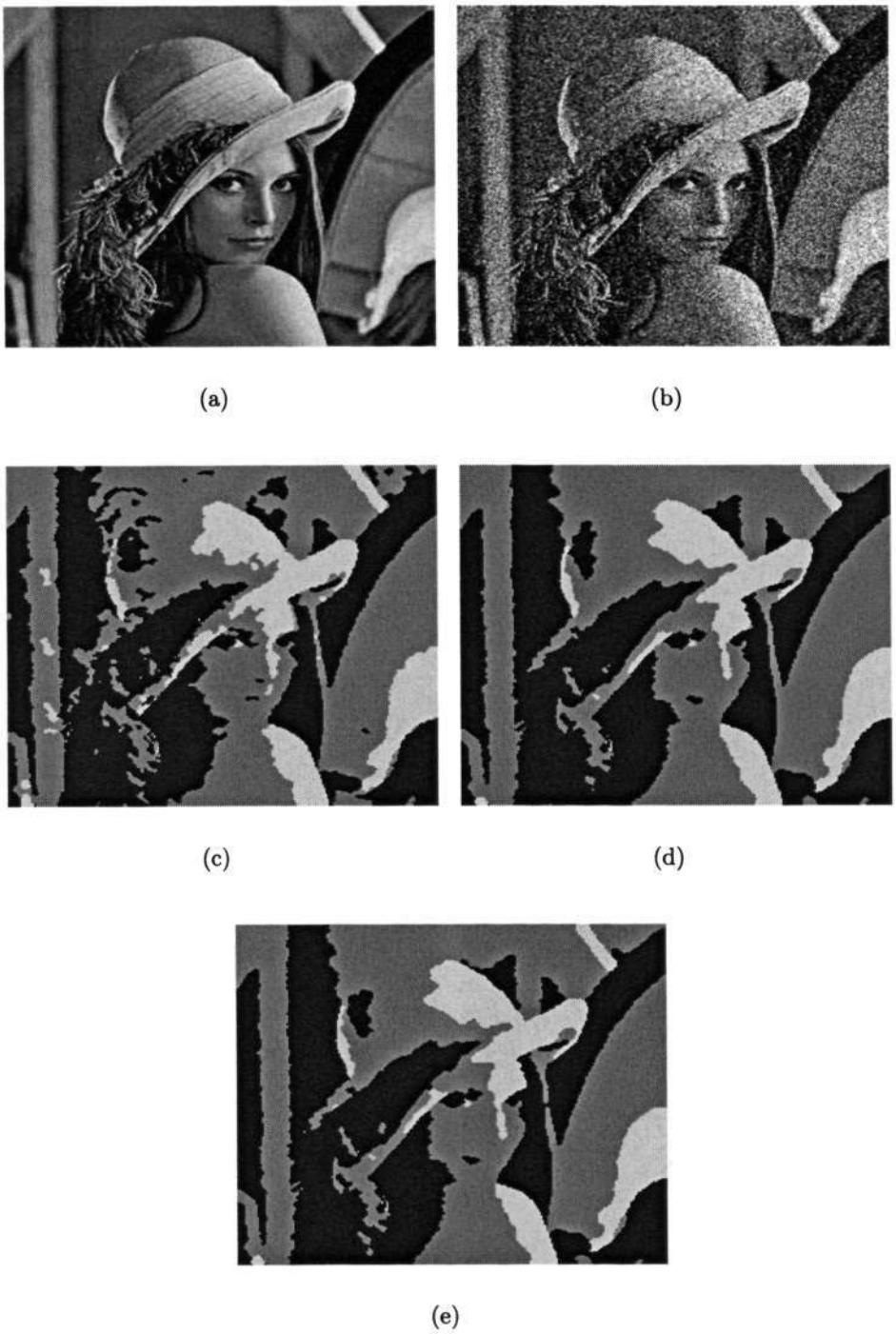


Figure 4.10: Segmentation of noisy chessboard image. (a) original image. (b) noisy image. (c) segmentation by ICM. (d) segmentation by SA. (e) segmentation by our algo.



(a) (b)

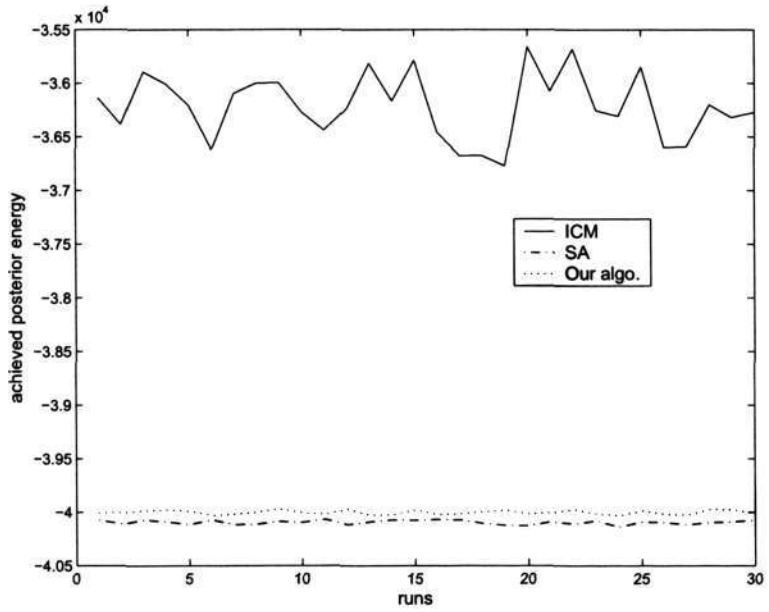
(c) (d)

(e)

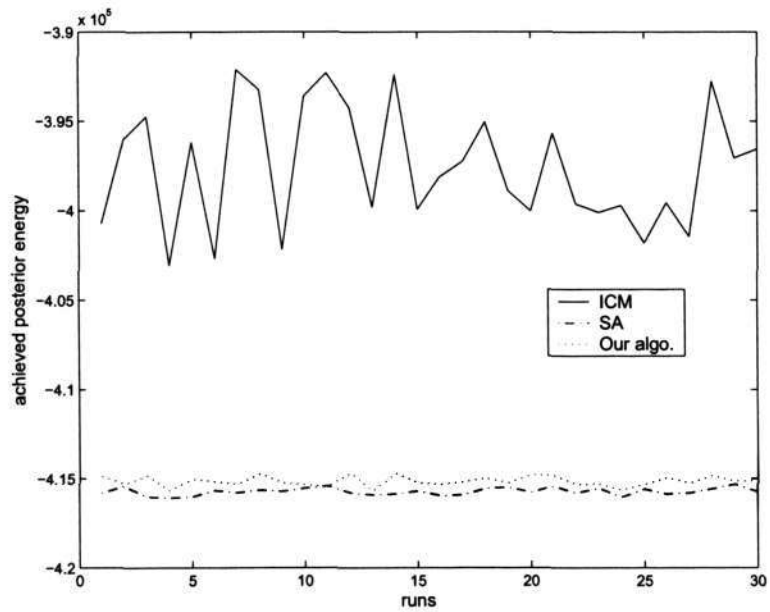
Figure 4.11: Segmentation of noisy Lenna image. (a) original image. (b) noisy image. (c) segmentation by ICM. (d) segmentation by SA. (e) segmentation by our algo.

Table 4.1: Noisy image segmentation: the achieved posterior energy over 30 independent runs of three algorithms. Mean and variance are also computed. Random initializations are used. For each run, the results are recorded in a row. In the "Chessboard" image, the energy values including mean and variance should be multiplied by 10^4. In the "Lenna" image, the energy values including mean should be multiplied by 10^5; the variance should be multiplied by 10^7.

	Chessboard			**Lenna**		
Run	ICM	SA	our algo	ICM	SA	our algo
1	−3.6137	−4.0075	−4.0008	−4.0074	−4.1580	−4.1487
2	−3.6383	−4.0108	−4.0003	−3.9602	−4.1543	−4.1530
3	−3.5895	−4.0071	−3.9985	−3.9476	−4.1602	−4.1483
4	−3.6004	−4.0089	−3.9979	−4.0307	−4.1607	−4.1566
5	−3.6204	−4.0115	−3.9992	−3.9618	−4.1603	−4.1503
6	−3.6621	−4.0074	−4.0031	−4.0269	−4.1567	−4.1518
7	−3.6096	−4.0117	−4.0017	−3.9212	−4.1579	−4.1529
8	−3.6001	−4.0113	−3.9999	−3.9324	−4.1564	−4.1472
9	−3.5996	−4.0088	−3.9969	−4.0218	−4.1572	−4.1524
10	−3.6269	−4.0095	−4.0006	−3.9361	−4.1554	−4.1534
11	−3.6436	−4.0067	−4.0019	−3.9229	−4.1541	−4.1545
12	−3.6238	−4.0119	−3.9972	−3.9424	−4.1579	−4.1468
13	−3.5814	−4.0092	−4.0027	−3.9983	−4.1591	−4.1570
14	−3.6166	−4.0074	−4.0023	−3.9239	−4.1587	−4.1470
15	−3.5786	−4.0075	−3.9980	−3.9993	−4.1570	−4.1522
16	−3.6454	−4.0070	−4.0022	−3.9812	−4.1595	−4.1531
17	−3.6677	−4.0073	−4.0014	−3.9725	−4.1591	−4.1518
18	−3.6675	−4.0107	−3.9998	−3.9504	−4.1555	−4.1497
19	−3.6769	−4.0123	−3.9983	−3.9889	−4.1548	−4.1523
20	−3.5660	−4.0126	−4.0016	−4.0004	−4.1577	−4.1478
21	−3.6075	−4.0090	−4.0008	−3.9568	−4.1543	−4.1483
22	−3.5680	−4.0108	−3.9977	−3.9964	−4.1580	−4.1532
23	−3.6254	−4.0081	−4.0016	−4.0010	−4.1554	−4.1529
24	−3.6306	−4.0138	−4.0033	−3.9972	−4.1604	−4.1565
25	−3.5847	−4.0091	−3.9989	−4.0182	−4.1559	−4.1531
26	−3.6596	−4.0096	−4.0019	−3.9956	−4.1586	−4.1494
27	−3.6595	−4.0116	−4.0026	−4.0146	−4.1580	−4.1525
28	−3.6202	−4.0095	−3.9972	−3.9276	−4.1557	−4.1483
29	−3.6319	−4.0092	−3.9975	−3.9706	−4.1532	−4.1513
30	−3.6270	−4.0076	−4.0013	−3.9656	−4.1574	−4.1512
mean	−3.6214	−4.0095	−4.0002	−3.9757	−4.1572	−4.1514
var.	9.3288	0.0385	0.0400	1.1576	0.0043	0.0080



(a) Chessboard



(b) Lenna

Figure 4.12: Noisy image segmentation: the curves of achieved posterior energy over 30 independent runs of algorithms. Three algorithms are compared: ICM, SA and our algo.

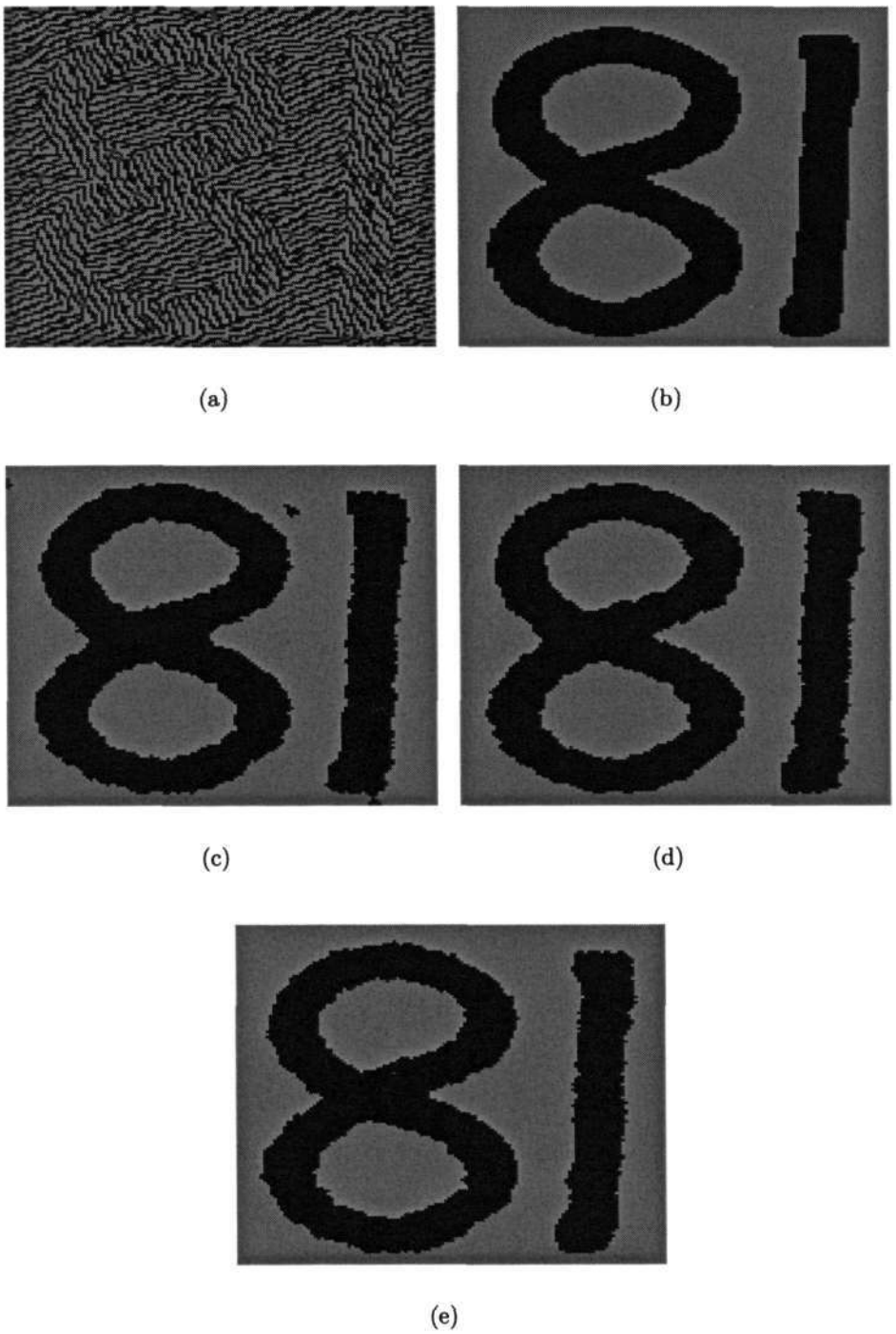
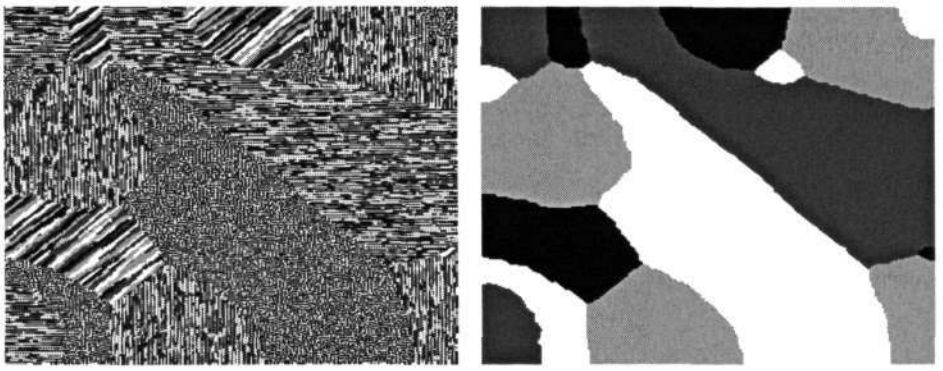
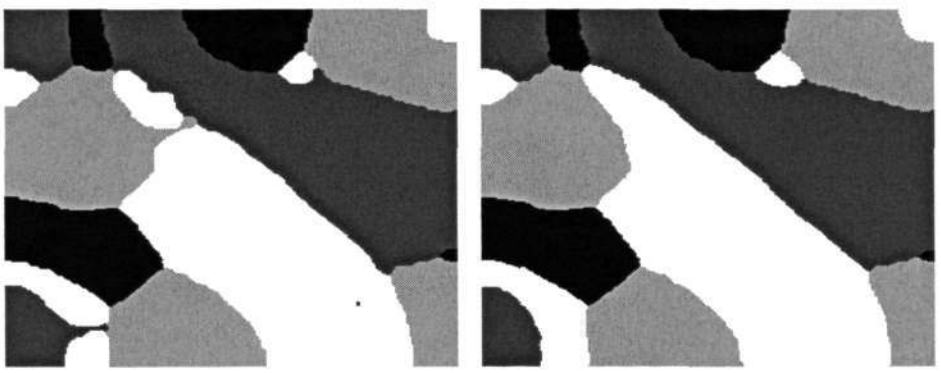


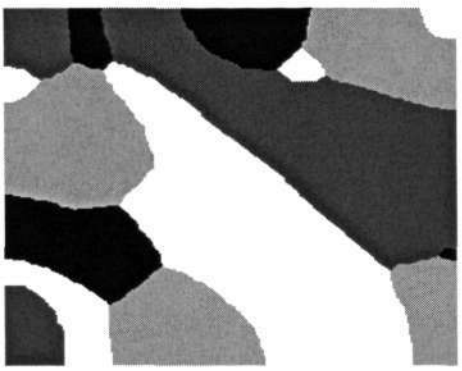
Figure 4.13: Segmentation of textured image 1. (a) original image. (b) perfect segmentation. (c) segmentation by ICM. (d) segmentation by SA. (e) segmentation by our algo. Different textures are represented by different colors in the result.



(a) (b)



(c) (d)

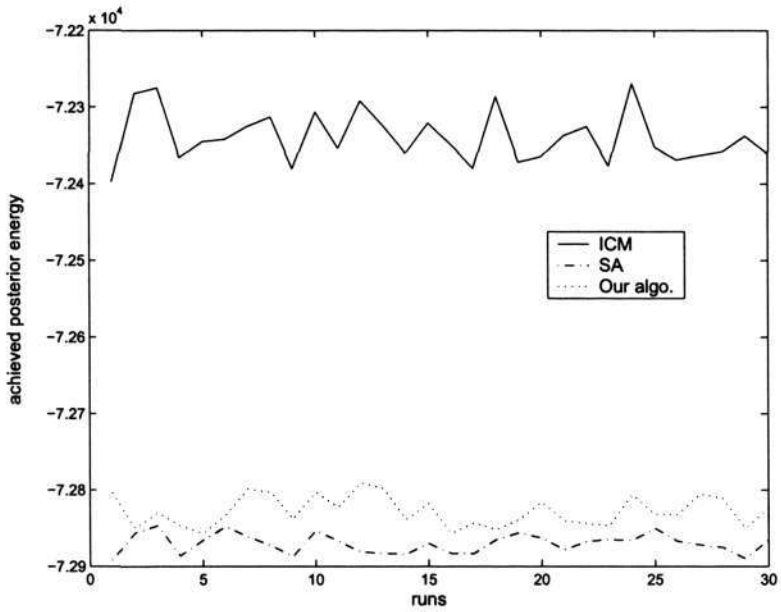


(e)

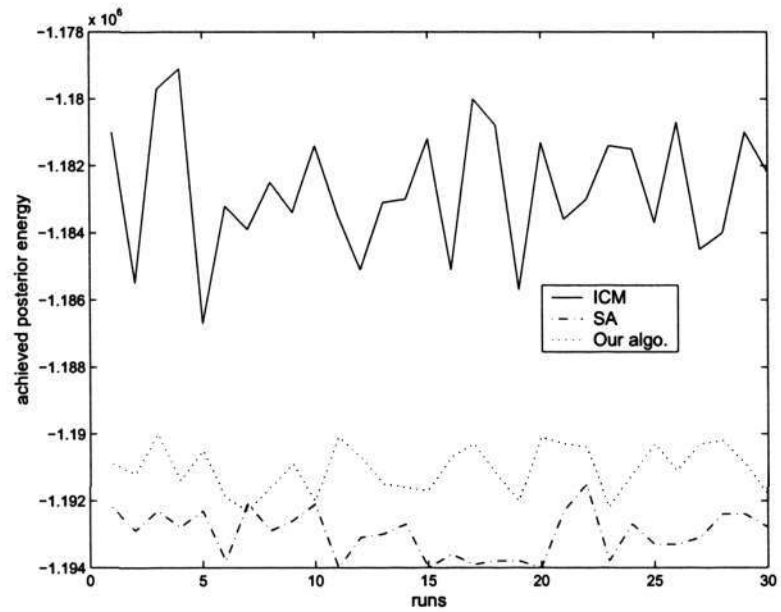
Figure 4.14: Segmentation of textured image 2. (a) original image. (b) perfect segmentation. (c) segmentation by ICM. (d) segmentation by SA. (e) segmentation by our algo. Different textures are represented by different colors in the result.

Table 4.2: Textured image segmentation: the achieved posterior energy over 30 independent runs of three algorithms. Mean and variance are also computed. Random initializations are used. For each run, the results are recorded in a row. In the "Texture 1" image, the energy values including mean and variance should be multiplied by 10^4; the variance should be multiplied by 10^3. In the "Texture 2" image, the energy values including mean and variance should be multiplied by 10^6.

	Texture 1			**Texture 2**		
Run	ICM	SA	our algo	ICM	SA	our algo
1	−7.2398	−7.2892	−7.2804	−1.1810	−1.1922	−1.1909
2	−7.2282	−7.2856	−7.2850	−1.1855	−1.1929	−1.1912
3	−7.2275	−7.2847	−7.2830	−1.1797	−1.1923	−1.1900
4	−7.2366	−7.2886	−7.2847	−1.1791	−1.1928	−1.1914
5	−7.2345	−7.2866	−7.2857	−1.1867	−1.1923	−1.1905
6	−7.2342	−7.2848	−7.2833	−1.1832	−1.1938	−1.1919
7	−7.2325	−7.2861	−7.2799	−1.1839	−1.1921	−1.1923
8	−7.2313	−7.2873	−7.2803	−1.1825	−1.1929	−1.1916
9	−7.2381	−7.2887	−7.2838	−1.1834	−1.1926	−1.1909
10	−7.2306	−7.2853	−7.2803	−1.1814	−1.1921	−1.1920
11	−7.2354	−7.2867	−7.2823	−1.1835	−1.1940	−1.1901
12	−7.2292	−7.2881	−7.2791	−1.1851	−1.1931	−1.1907
13	−7.2324	−7.2883	−7.2798	−1.1831	−1.1930	−1.1915
14	−7.2360	−7.2884	−7.2839	−1.1830	−1.1927	−1.1916
15	−7.2321	−7.2870	−7.2818	−1.1812	−1.1940	−1.1917
16	−7.2349	−7.2883	−7.2857	−1.1851	−1.1936	−1.1907
17	−7.2380	−7.2883	−7.2843	−1.1800	−1.1939	−1.1903
18	−7.2286	−7.2865	−7.2852	−1.1808	−1.1938	−1.1912
19	−7.2372	−7.2856	−7.2839	−1.1857	−1.1938	−1.1920
20	−7.2365	−7.2863	−7.2816	−1.1813	−1.1940	−1.1901
21	−7.2337	−7.2878	−7.2840	−1.1836	−1.1923	−1.1903
22	−7.2325	−7.2867	−7.2844	−1.1830	−1.1915	−1.1904
23	−7.2377	−7.2865	−7.2847	−1.1814	−1.1938	−1.1922
24	−7.2269	−7.2866	−7.2806	−1.1815	−1.1927	−1.1912
25	−7.2352	−7.2850	−7.2832	−1.1837	−1.1933	−1.1903
26	−7.2369	−7.2867	−7.2832	−1.1807	−1.1933	−1.1911
27	−7.2363	−7.2872	−7.2805	−1.1845	−1.1931	−1.1903
28	−7.2358	−7.2875	−7.2811	−1.1840	−1.1924	−1.1902
29	−7.2338	−7.2890	−7.2850	−1.1810	−1.1924	−1.1909
30	−7.2361	−7.2865	−7.2824	−1.1822	−1.1928	−1.1918
mean	−7.2340	−7.2870	−7.2828	−1.1827	−1.1930	−1.1910
variance	1.1769	0.1625	0.3964	3.6627	0.4849	0.4936



(a) Texture 1



(b) Texture 2

Figure 4.15: Textured image segmentation: the curves of achieved posterior energy over 30 independent runs of algorithms. Three algorithms are compared: ICM, SA and our algo.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Evolutionary computation (EC) is a biologically inspired technique working on the neo-Darwinian paradigm of natural evolution. It consists of a class of algorithms such as genetic algorithms which are particularly suitable to solve complex optimization and search problems. Different from traditional optimization techniques, EC searches from a population of points (or solutions), not a single point; it generally uses only the payoff information and does not need other auxiliary knowledge like derivatives; it adopts probabilistic transition rules rather than deterministic rules. EC can perform efficient search in high-dimensional spaces which may bear complicated landscapes. It has the capability of jumping over those local optima in favor of the global one. Moreover, evolutionary algorithms are readily parallelized for improved computational efficiency and robustness. All these attractive characteristics of evolutionary computation make it a favorable choice for those problems involved in statistical pattern recognition.

In the thesis the application of evolutionary computation in statistical pattern recogni-

tion is surveyed according to a common dichotomy, that is, from two aspects: feature selection/extraction and supervised/unsupervised learning. In both aspects those representative works are discussed. For each work, we describe the reasons why evolutionary computation is borrowed and how it is adapted to the problem, i.e. how to design the schemes of representation, fitness evaluation, selection, and generation of new solutions. The observation is that evolutionary computation is not a collection of ready-to-use algorithms. Instead it should be understood as a general idea for problem solving. For statistical pattern recognition which involves various kinds of difficult optimization problems, the most significant advantage of using evolutionary computation lies in the gain of flexibility and adaptability to the task at hand, in combination with robust performance and global search characteristic.

An evolutionary algorithm for ensemble learning is developed. In this topic we focus on methods of manipulating training examples and analyze the two well-known algorithms named Bagging and AdaBoost. We find them essentially two different ways to find a set of weightings associated with the training set. Bagging uses random walk and AdaBoost uses step by step adjustment. This viewpoint inspires us to incorporate a genetic algorithm to perform evolutionary search in the weighting space. It is a novel idea in ensemble learning. The evolutionary search encourages diversified component classifiers with the help of fitness sharing. In the mean time, it does much more than a random walk; the selective pressure always drives the population to areas of high evaluation and therefore the induced classifiers tend to have high classification accuracy. Another novelty of our algorithm lies in that all the chromosomes throughout generations are exploited. We notice that the fitness function is basically ill-posed since a low training error does not necessarily mean good generalization. Hence the inclusion of less-optimized chromosomes into the final ensemble is helpful. In comparative experiments

with Bagging and AdaBoost, the proposed algorithm shows its remarkable performance.

Markov random field is a well-known prior model in statistical pattern recognition. In MRF modelling, the optimal solution to a problem is often defined as the minimum of an energy function. An evolutionary algorithm is developed for this function optimization task. The main novelty in our algorithm is that the new population is not generated by operators of crossover and mutation. At every generation a probabilistic model is learned for the distributions of component variables, which are biased to solutions with higher fitness. A new population is then generated by sampling from this model. The idea is based on the fact that the component variables in a solution do not scatter without a rule; on the contrary, the structure of their relationships are predefined by the MRF model. Therefore the general and "blind" operation of randomly exchanging part information between solutions is not suitable, because it may not maintain the MRF structure and may produce unreasonable solutions. If we update the population by sampling from the probabilistic model, on the one hand, the selective pressure is naturally imposed; on the other hand, this scheme keeps the search in a reasonable area of the solution space. By the proposed scheme, local search strategy is also readily incorporated to improve the convergence speed. The proposed algorithm is thus seen as another example of exploiting *a priori* knowledge to design problem-specific evolution schemes. In the experiments the algorithm is compared with two widely used algorithms, ICM and SA. Again, the adaptability and potential of evolutionary computation is demonstrated.

In a summary, the popularity of evolutionary computation originates from its robustness, global search characteristic, and adaptability to the target problem. Specifically in the domain of statistical pattern recognition, both the literature and the novel algorithms proposed in the thesis respectively for ensemble learning and MRF modelling have proved our observation. The last decade has seen its extensive adoption. With the

steady increase of interest and practice, evolutionary computation as a problem-solving technique will play more and more important roles in statistical pattern recognition.

5.2 Future Work

Although the presented work has been shown creative and testified, several possibilities exist to further enhance the work. Some important directions of further research are identified as follows:

- Despite the explanations in chapter 3 why ensembles can bring improvement of accuracy, there is theoretical analysis based on the bias-variance decomposition [108]. It says that the mean-square error of a classifier can be decomposed as squared bias plus variance (the intrinsic target noise is ignored here). The bias measures how closely the learning algorithm's average guess (over all possible training sets of the given size) matches the target and the variance measures how much the guess "bounces around" for different training sets of the given size. When we extend the analysis to an ensemble of T classifiers, the expected mean-square error can be written as a sum of three parts: the squared bias of the combined system, the variance and covariance of the outputs of the individual classifiers [109]. Among them the variance can be seen to decay at $1/T$ with respect to a single classifier. Obviously it is a major reason why an ensemble is seen having high accuracy. But, the analysis also reveals that performance of the overall system is weakened if the errors of individual classifiers are positively correlated. The problem is further studied by Brown *et al.* [80] in the notion of diversity.

  In order to ensure different components in an ensemble to learn different parts or aspects of the training data, Liu and Yao [110] proposed the Negative Correlation

Learning (NCL) algorithm. NCL introduces a correlation penalty term in the error function of each individual network in the ensemble. That is, the error function E_t for network $h_t(.)$ is defined by

$$E_t = \frac{1}{N}\sum_{i=1}^{N}\mathbf{I}(y_i \neq h_t(\mathbf{x}_i)) + \frac{1}{N}\sum_{i=1}^{N}\lambda p_t(\mathbf{x}_i), \tag{5.1}$$

where $\mathbf{I}$ is the indication function (referring to 2.10). The first item in the right side is the empirical risk function and the second item p_t is the correlation penalty function. The parameter $0 \leq \lambda \leq 1$ is used to adjust the strength of the penalty. The purpose of minimizing p_t is to negatively correlate each network's error with errors for the rest of the ensemble, which could be seen from the definition

$$p_t(\mathbf{x}_i) = (h_t(\mathbf{x}_i) - H(\mathbf{x}_i))\sum_{t' \neq t}(h_{t'}(\mathbf{x}_i) - H(\mathbf{x}_i)). \tag{5.2}$$

Here $H(.)$ is the ensemble classifier. During the training process, all the individual neural networks interact with each other through their penalty terms. The authors reported that unlike other ensemble approaches in which each network learns the same task in independent training and the correlations among them are generally positive, with NCL each network learns different aspects of the training data and their errors tend to be negatively correlated. NCL is also used for extracting diverse and complementary sets of features from high-dimensional data [111]; experiments show that the features extracted from an ensemble of small networks are significantly more useful for classification than the same number of features extracted from a single large network.

Liu and Yao further used the idea of NCL in an evolutionary learning system, i.e. EPNet [10], and developed the EENCL algorithm [90]. There are two levels of

adaptation in EENCL: at the population level the evolution through selection and mutation characterizes the global search for optimized network architectures and connection weights; at the individual level the training of networks in a population is implemented by negative correlation learning. With the help of NCL, the algorithm produces NN ensembles with remarkable generalization ability.

We have seen from EENCL that pure evolutionary strategies may not be sufficient for producing diversified networks and the inclusion of explicit diversity generalization and maintenance mechanisms like NCL is no doubt useful. So it is reasonable to ask if NCL could also be borrowed by our algorithm in chapter 3 to improve the generalization. A straightforward incorporation will be adjusting the error function in (3.7) by adding a correlation penalty term. Our algorithm works by manipulating the training samples which differs from EENCL that works by injecting randomness into the learning algorithm. The correlation penalty term is only calculated within a single population which does not conform with our proposal of exploiting individuals throughout all populations. Although design difficulties exist, we expect that the inclusion of schemes like NCL will help strengthen the diversity among individual learners like in EENCL.

- More recently, multi-objective evolutionary algorithms have been used for ensemble learning [112–114]. In [113] Chandra and Yao analyze the diversity-accuracy trade-off in ensembles and argue that "one very strong motivation for the use of evolutionary multi-criterion optimisation in the creation of an ensemble is that the presence of multiple conflicting objectives engenders a set of near optimal solutions...if one uses multiobjectivity while creating ensembles, one can actually generate an ensemble automatically where the member networks would inadvertently be near optimal." The authors are inspired by NCL and as the two objectives

of accuracy and diversity, they use the empirical risk function and the correlation penalty function respectively (referring to (5.1)). In the evolution, selection is biased to those points in the Pareto front through the non-dominated sorting procedure [115]. The proposed algorithm (DIVACE) is shown producing competitive results in this multiobjective setup.

In [114] Chandra and Yao propose a hierarchical evolutionary framework for the construction of diverse hybrid ensemble, in which there are three levels of evolution present: first is evolving the mixture of the various types of predictors, second is the evolution of the ensemble based on the structure of the training set (e.g. Bagging, AdaBoost and our algorithm) and a process similar to the EENCL or DIVACE algorithm forms the third level. One instance of the framework is the DIVACE-II algorithm in which all three levels are modelled. DIVACE-II is generally seen to outperform DIVACE and EENCL. This establishes the authors' idea of enforcing diversity at multiple levels as being plausible. Following the idea, we are considering to improve our algorithm in chapter 3 with a different level of evolution like DIVACE. Although the framework looks promising, much work still remains to be done to find an appropriate way for the combination.

- A major difficulty in minimizing the MRF energy function is to deal with the high dimensionality (typically $\backsim 10^4$). Conventional evolutionary algorithms may not perform satisfactorily in the large-scale problems. A choice here is to employ the cooperative co-evolutionary algorithms (CCEA) [116]. The principle behind CCEA is *divide-and-conquer*. Specifically, we can decompose the system into many modules, define an individual as a candidate of a module, assign a population to each module, find the best individual in each population and put them together again to form the whole system. In [117] CCEA is exploited and seen to signif-

icantly improve the scalability of fast evolutionary programming (FEP [118]) in numerical optimization: the time to find a near optimal solution appears to increase linearly as the dimensionality increases. However, it is also reported that the resulted algorithm sometimes fell into a local optimum, due to its greedy fitness evaluation.

The idea of CCEA is similar to a certain extent with what is studied in chapter 4. In our design, the variables in a MRF is decomposed into four codings (with second-order neighborhood system). Given the undirected nature of the dependencies among variables, we have to temporarily break the dependencies and rely on iterative evolution of the codings, like the iterative updating of variables in ICM and SA. Unfortunately such operation inevitably leads to a local optimum. While SA uses annealing to treat the dilemma which also brings tremendous computation, our design relies on population-based evolutionary principles. We make use of *a prior* knowledge to confine the search in a reasonable area.

In FEPCC [117], the populations associated with different variables will be evolved sequentially, but the relationships among the variables are not emphasized. For our problem, the situation is special in that the evolution of one coding will influence the evolution of other codings through the contextual constraint. Simply ignoring the relationships and evolving the codings independently is not acceptable. Still, we find FEPCC interesting because "an individual in FEPCC is a component in a vector and other components in the vector remain unchanged, it only needs to calculate the difference caused by the changed component. This fitness evaluation method...takes less computation time" [117]. That means only partial updating of the random field in our context can define a new generation (compared with our current setting in which a new generation demands updating of the whole field).

The algorithm behavior under such setting is not clear and furture study on this topic will be useful.

- There are recently some papers [119] on linear feature extraction which use stochastic gradient algorithms for finding the optimal transformation matrix. It is claimed that the set of all transformation matrixes (known as the Grassmann manifold) is not a vector space and the optimization process needs be modified to account for its curved geometry. Researchers have tried MCMC (Markov chain Monte Carlo) type algorithms. In our opinion, evolutionary algorithms are very effective in dealing with such kind of optimization problem. It is an interesting direction of our future work.

Author's Publications

1. Xiao Wang and Han Wang, "Classification by Evolutionary Ensembles". *Pattern Recognition*, Vol. 39, No. 4, Apr. 2006, p595-607.

2. Xiao Wang and Han Wang, "Evolutionary Optimization with Markov Random Field Prior". *IEEE Trans. On Evolutionary Computation*, Vol. 8, No. 6, Dec. 2004, p567-579.

3. Xiao Wang and Han Wang, "Markov Random Field Modeled Range Image Segmentation". *Pattern Recognition Letters*, Vol. 25, Feb. 2004, p367-375.

4. Xiao Wang and Han Wang, "Evolutionary Gibbs Sampler for Image Segmentation". In Proc. IEEE International Conference on Image Processing (ICIP 2004), Oct. 2004, Singapore.

5. Xiao Wang and Han Wang, "An Extended ICM Algorithm for Range Image Segmentation". In Proc. 8th International Conference on Control, Automation, Robotics and Vision (ICARCV 2004), Dec. 2004, Kunming, China.

6. Xiao Wang and Han Wang, "Evolutionary Optimization in Markov Random Field Modeling". In Proc. 4th International Conference on Information, Communications & Signal (ICICS-PCM 2003), Dec. 2003, Singapore.

7. Xiao Wang and Han Wang, "Markov Random Field Modeled Range Image Segmentation". In Proc. 4th International Conference on Information, Communications & Signal (ICICS-PCM 2003), Dec. 2003, Singapore.

Bibliography

[1] C. Liu and H. Wechsler. Evolutionary pursuit and its application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):570–581, Jun. 2000.

[2] S.Z. Li. *Markov Random Field Modeling in Image Analysis.* Springer-Verlag, 2001.

[3] K. F. Man, K. S. Tang, and S. Kwong. *Genetic Algorithms: Concepts and Designs.* Springer-Verlag, 1999.

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

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* John Wiley & Sons, New York, 2000.

[6] E.S. Gelsema, editor. *Special Issue on Genetic Algorithms, Pattern Recognition Letters*, volume 16. 1996.

[7] S.K. Pal and P.P. Wang, editors. *Genetic Algorithms for Pattern Recognition.* CRC Press, 1996.

[8] Y.-K. Wang and K.-C. Fan. Applying genetic algorithms on pattern recognition: An analysis and survey. In *Proc. International Conference on Pattern Recognition*, pages 740–744, 1996.

[9] X. Yao. A review of evolutionary artificial neural networks. *International journal of intelligent systems*, 8:539–567, 1993.

[10] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, 1997.

[11] A. Hoffman. *Arguments on Evolution: A Paleontologist's Perspective.* Oxford University Press, 1988.

[12] S. Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In *Proc. 6th Int. Gong. Genetics*, volume 1, pages 356–366, Ithaca, NY, 1932.

[13] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, MI, 1975.

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

[15] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung.* PhD thesis, Technische Universität Berlin, Germany, 1975.

[16] L. J. Fogel. *On the organization of intellect.* PhD thesis, University of California, Los Angeles, 1964.

[17] T. Bäck, U. Hammel, and H.-P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, Apr. 1997.

[18] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[19] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, Univ. of Michigan, 1975.

[20] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proc. Third International Conference on Genetic Algorithms*, pages 51–60, 1989.

[21] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, 1992.

[22] D. J. Montana. Automated parameter tuning for interpretation of synthetic images. In L. Davis, editor, *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[23] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[24] A. H. Wright. Genetic algorithms for real parameter optimization. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1991.

[25] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proc. Second International Conference on Genetic Algorithms and their Application*, pages 14–21, 1987.

[26] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1991.

[27] G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[28] W. Siedlecki and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10:335–347, Nov. 1989.

[29] J. D. Kelly and L. Davis. Hybridizing the genetic algorithm and the k nearest neighbors classification algorithm. In *Proc. International Conference on Genetic Algorithms Application*, pages 377–383, 1991.

[30] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn, and A. K. Jain. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(2):164–171, Jul. 2000.

[31] K. Fukunaga. *Introduction to statistical pattern recognition.* Academic Press, 1991.

[32] V. N. Vapnik. *Statistical learning theory.* John Wiley & Sons, New York, 1998.

[33] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In J. D. Schaffer, editor, *Proc. the third international conference on genetic algorithms and their applications*, pages 379–384. Morgan Kaufmann, 1989.

[34] S. K. Pal, S. Bandyopadhyay, and C. A. Murthy. Genetic algorithms for generation of class boundaries. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 28(6):816–828, Dec. 1998.

[35] A. K. Jain and R. C. Dubes. *Algorithms for clustering data.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[36] S. Z. Selim and M. A. Ismail. K-means type algorithms: a generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:81–87, 1984.

[37] U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33:1455–1465, 2000.

[38] L. O. Hall, I. B. Ozyurt, and J. C. Bezdek. Clustering with a genetically optimized approach. *IEEE Transactions on Evolutionary Computation*, 3(2):103–112, 1999.

[39] A. J. Katz and P. R. Thrift. Generating image filters for target recognition by genetic learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):906–910, Sep. 1994.

[40] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal. Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation*, 4(3):242–257, Sep. 2000.

[41] L. I. Kuncheva and L. C. Jain. Designing classifier fusion systems by genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(4):327–336, Sep. 2000.

[42] P. Andrey and P. Tarroux. Unsupervised image segmentation using a distributed genetic algorithm. *Pattern Recognition*, 27(5):659–673, 1994.

[43] P. Andrey and P. Tarroux. Unsupervised segmentation of Markov random field modeled textured images using selectionist relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):252–262, Mar. 1998.

[44] C.J. Veenman, M.J.T. Reinders, and E. Backer. A cellular coevolutionary algorithm for image segmentation. *IEEE Transactions on Image Processing*, 12(3):304–316, Mar. 2003.

[45] S.M. Bhandarkar and H. Zhang. Image segmentation using evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 3(1):1–21, Apr. 1999.

[46] B. Bhanu, S. Lee, and S. Das. Adaptive image segmentation using genetic and hybrid search methods. *IEEE Transactions on Aerospace and Electronic Systems*, 31(4):1268–1291, Oct. 1995.

[47] S.-H. Lai and B.C. Vemuri. Efficient hybrid search for visual reconstruction problems. *Image and Vision Computing*, 17:37–49, 1999.

[48] M. Yu, N. Eua-anant, A. Saudagar, and L. Udpa. Genetic algorithm approach to image segmentation using morphological operations. In *Proc. International Conference on Image Processing*, pages 775–779, Oct. 1998.

[49] T. G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4):97–136, 1997.

[50] E. Bauer and R. Kohavi. An experimental comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, 1999.

[51] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40:139–157, 2000.

[52] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–156, 1996.

[53] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[54] A. Blum and R. L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127, 1992.

[55] X. Yao and Y. Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 28(3):417–425, 1998.

[56] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[57] B. Efron and R. Tibshirani. *An Introduction to the Booststrap.* Chapman and Hall, New York, NY, 1993.

[58] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[59] R.E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37:297–336, 1999.

[60] Y. Freund. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43:293–318, 2001.

[61] G. Ratsch. Soft margins for adaboost. *Machine Learning*, 42:287–320, 2001.

[62] S. Z. Li. Floatboost learning and statistical face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1112–1123, 2004.

[63] T. Poggio, V. Torre, and C. Koch. Computational vision and regularization theory. *Nature*, 317:314–319, Sep. 1985.

[64] S.Z. Li. On discontinuity-adaptive smoothness priors in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):576–586, 1995.

[65] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B*, 48:259–302, 1986.

[66] A. Rosenfeld, R.A. Hummel, and S.W. Zucker. Scene labeling by relaxation operations. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:420–433, Jun. 1976.

[67] P.B. Chou and C.M. Brown. The theory and practice of Bayesian image labeling. *International Journal of Computer Vision*, 4:185–210, 1990.

[68] H. Derin and H. Elliott. Modeling and segmentation of noisy and textured images using Gibbs random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(1):39–55, Jan. 1987.

[69] I.M. Elfadel. *From Random Fields to Networks.* PhD thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1993.

[70] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, New York, NY, 1989.

[71] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, Nov. 1984.

[72] D. Thierens. *Analysis and design of genetic algorithms.* PhD thesis, Leuven, Belgium, 1995.

[73] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, 2000.

[74] J.R. Quinlan. Bagging, boosting, and c4.5. In *Proc. 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 725–730, 1996.

[75] T.G. Dietterich and G. Bakiri. Solving multiclass learning problems vis error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[76] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[77] G. I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40:159–196, 2000.

[78] N. N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21, 1992.

[79] J.R. Quinlan. Boosting first-order learning. In *Proc. 7th International Workshop on Algorithmic Learning Theory*, pages 143–155, 1996.

[80] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Journal of Information Fusion*, 6(1):5–20, 2005.

[81] J. Horn, D. E. Goldberg, and K. Deb. Implicit niching in a learning classifier system: nature's way. *Evol. Comput.*, 2(1):37–66, 1994.

[82] P. Darwen and X. Yao. Speciation as automatic categorical modularization. *IEEE Transactions on Evolutionary Computation*, 1(2):101–108, 1997.

[83] J.-H. Ahn and S.-B. Cho. Speciated neural networks evolved with fitness sharing technique. In *Proc. Congress on Evolutionary Computation*, pages 27–30, May 2001.

[84] V. Khare and X. Yao. Artificial speciation of neural network ensembles. In *Proc. UK Workshop on Computational Intelligence*, pages 96–103, 2002.

[85] M.-H. Yang, D.J. Kriegman, and N. Ahjua. Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58, 2002.

[86] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

[87] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[88] K.-K. Sung and T. Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51, 1998.

[89] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[90] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.

[91] M. Pelikan, D.E. Goldberg, and F.G. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21:5–20, 2002.

[92] P. Larrañaga and J.A. Lozano, editors. *Estimation of Distribution Algorithms.* Kluwer Academic, Norwell, MA, 2002.

[93] S. 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.

[94] H. Muhlenbein and G. Paab. From recombination of genes to the estimation of distributions i. binary parameters. In *Proc. 4th International Conference on Parallel Problem Solving from Nature*, pages 178–187, 1996.

[95] G.R. Harik, F.G. Lobo, and D.E. Goldberg. The compact genetic algorithm. In *Proc. International Conference on Evolutionary Computation*, pages 523–528, 1998.

[96] F.V. Jensen. *Bayesian Networks and Decision Graphs.* Springer, New York, NY, 2001.

[97] G.R. Harik. Linkage learning via probabilistic modeling in the ecga. Technical Report 99010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.

[98] M. Pelikan, D.E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation, and Bayesian networks. Technical Report 98013, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1998.

[99] J.S. De Bonet, C.L. Isbell, and P. Viola. Mimic: Finding optima by estimating probability densities. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. The MIT Press, Cambridge, CA, 1997.

[100] M. Pelikan and H. Muhlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P.K. Chawdhry, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535. Springer-Verlag, London, UK, 1999.

[101] S. Baluja. Using a priori knowledge to create probabilistic models for optimization. *International Journal of Approximate Reasoning*, 31(3):193–220, Nov. 2002.

[102] P. Smyth. Belief networks, hidden Markov models, and Markov random fields: a unifying view. *Pattern Recognition Letters*, 18:1261–1268, Nov. 1997.

[103] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society, Series B*, 36:192–236, 1974.

[104] G. Syswerda. Simulated crossover in genetic algorithms. In L.D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 239–255. Morgan Kaufmann, San Mateo, CA, 1993.

[105] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, Jul. 1999.

[106] S. Umeyama. Least squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:376–380, 1991.

[107] W.M. Wells III. Map model matching. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 486–492, 1991.

[108] R. A. Jacobs. Bias/variance analyses of mixture-of-experts architectures. *Neural Computation*, 9:369–383, 1997.

[109] Y. Liu and X. Yao. Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 29(6):716–725, 1999.

[110] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404, 1999.

[111] G. Brown, X. Yao, J. Wyatt, H. Wersing, and B. Sendhoff. Exploiting ensemble diversity for automatic feature extraction. In *Proc. 9th International Conference on Neural Information Processing*, pages 1786–1790, Singapore, 2002.

[112] H.A. Abbass. Pareto neuro-ensemble. In *Proc. 16th Australian Joint Conference on Artificial Intelligence*, pages 554–566, Perth, 2003.

[113] A. Chandra and X. Yao. Divace: Diverse and accurate ensemble learning algorithm. In *Proc. 5th International Conference on Intelligent Data Engineering and Automated Learning*, pages 619–625, 2004.

[114] A. Chandra and X. Yao. Evolutionary framework for the construction of diverse hybrid ensembles. In *Proc. 13th European Symposium on Artificial Neural Networks*, pages 253–258, Bruges, Belgium, 2005.

[115] N. Srinivas and K. Deb. Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[116] Q.F. Zhao. A general framework for cooperative co-evolutionary algorithms: a society model. In *Proc. 1998 IEEE International Conference on Evolutionary Computation*, pages 57–62, 1998.

[117] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi. Scaling up fast evolutionary programming with cooperative coevolution. In *Proc. the 2001 Congress on Evolutionary Computation*, pages 1101–1108, 2001.

[118] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.

[119] X. Liu, A. Srivastava, and K. Gallivan. Optimal linear representations of images for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):662–666, 2004.

Appendix A

Descriptions of Benchmark Data Sets

Here we briefly describe some properties for each of the data sets that we use in the experiments in chapter 3. For more comprehensive discussions we refer the readers to [76].

Audiology This data set is collected during the study of hearing, especially hearing defects and their treatment. Originally there are 200 training cases and 26 test cases. Each case is represented as a feature vector having 69 attributes. The task is to identify each case into one of the 24 classes, such as "normal ear", "otitis media", "possible brainstem disorder", etc.

Automobile This data set is donated by Jeffrey C. Schlimmer in 1987. The task here is to predict the insurance risk rating of a car using all the numeric and boolean attributes. The rating corresponds to the degree to which the auto is more risky than its price indicates. There are 205 instances in the set. Each instance includes several

attributes of a car such as "length", "width", "height", "wheel-base", etc.

Breast-Cancer-W This breast cancer database is obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The total 699 instances arrive periodically as Dr. Wolberg reports his clinical cases, among which about 65.5% belong to the "benign" class and 34.5% belong to the "malignant" class. The attributes in each instance include "Marginal Adhesion", "Bare Nuclei", "Bland Chromatin", and so on.

Diabetes This is a database gathered among the Pima Indians by the National Institute of Diabetes and Digestive and Kidney Diseases. The database consists of 768 cases, 8 variables and two classes. The variables are medical measurements on the patient plus age and pregnancy information. The classes are: tested positive for diabetes (268) or negative (500).

German This data set classifies people described by a set of attributes as good or bad credit risks. There are 1000 examples among which 700 persons have good credit and 300 have bad credit. The available attributes include qualitative ones like "credit history" and numerical ones like "duration in month" and "credit amount".

Glass This database is created in the Central Research Establishment, Home Office Forensic Science Service Aldermaston, Reading, Berkshire. Each case consists of 9 chemical measurements on one of 7 types of glass. There are 214 cases.

Heart This database is collected at the University of California, San Diego Medical Center. The goal of the medical study is the development of a method to identify high risk patients (those who will not survive at least 30 days) on the basis of the initial 24-hour data. The data includes measurements like "age", "sex", "chest pain type" and "resting blood pressure". In the 270 cases collected, 150 patients survived and 120 died.

Hepatitis There are 155 cases in this database. The attribute information includes "age", "sex", "steroid", "fatigue", etc. Two classes shall be identified: "live" and "die".

Horse-Colic Originally there are 300 instances for training and 69 instances for test in this database. Several attributes about a horse are collected, like "age", "if had surgery", "rectal temperature", "pulse", etc.

Iris This database is created and introduced by R. A. Fisher. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant ("Setosa", "Versicolour" and "Virginica"). One class is linearly separable from the other 2; the latter are not linearly separable from each other. The 4 attributes are "sepal length", "sepal width", "petal length" and "petal width".

Labor This database includes all collective agreements reached in the business and personal services sector for locals with at least 500 members (teachers, nurses, university staff, police, etc) in Canada in 87 and first quarter of 88. The data is used to learn the description of an acceptable and unacceptable contract. The attribute information includes "duration of agreement", "wage increase in first/second/third year of contract", "cost of living allowance", etc.

Letter-Recognition This data set is constructed by David J. Slate, Odesta Corporation. Binary pixel displays of the 26 capital English letters are created and randomly distorted to produce 20,000 images. Sixteen features, consisting of statistical moments and edge counts, are extracted from each image.

Segment The instances were drawn randomly from a database of 7 outdoor images ("brickface", "sky", "foliage", "cement", "window", "pass" and "grass"). The images were hand-segmented to create a classification for every pixel. Each instance is a 3x3 region. The 19 continuous variables include "the column/row of the center pixel of the

region", "measures the contrast of vertically adjacent pixels", etc.

Sonar This data set is first used in a study of the classification of sonar signals using a neural network. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time.

Soybean-Large The soybean data set consists of 683 cases, 35 variables and 19 classes. The classes are various types of soybean diseases. The variables are observations on the plants together with some climatic variables. All are nominal. Some missing values are filled in by their modal values.

Splice This data set is about molecular biology. All examples are taken from Genbank 64.1. There are three categories: "ei" and "ie" including every split-gene for primates in Genbank 64.1, "non-splice" examples taken from sequences known not to include a splicing site. The attributes are the 60 DNA sequence elements given a position in the middle of a window.

Thyroid This thyroid disease database is donated by Stefan Aeberhard. There are 215 cases in the record. In the data five lab tests are used to try to predict whether a patient's thyroid to the class euthyroidism, hypothyroidism or hyperthyroidism. The five attributes include "total Serum thyroxin", "T3-resin uptake test", etc.

Vehicle The purpose is to classify a given silhouette as one of four types of vehicle ("OPEL", "SAAB", "BUS" and "VAN"), using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles. The 18 attributes include "compactness", "Circularity", "radius ratio", etc.

Voting This is the 1984 United States congressional voting records database. The

predicted category is the party affiliation (2 categories). The attribute information includes "handicapped-infants", "adoption-of-the-budget-resolution", "religious-groups-in-schools", etc.

Waveform There are 5000 instances in the database. Each wave is measure in 21 attributes with continuous values between 0 and 6. There are 3 classes of waves.