Data Mining Using Parallel Multi-objective Evolutionary Algorithms on Graphics Processing Units

Man Leung Wong and Geng Cui

Abstract An important and challenging data mining application in marketing is to learn models for predicting potential customers who contribute large profits to a company under resource constraints. In this chapter, we first formulate this learning problem as a constrained optimization problem and then convert it to an unconstrained multi-objective optimization problem (MOP), which can be handled by some multi-objective evolutionary algorithms (MOEAs). However, MOEAs may execute for a long time for the MOP, because several evaluations must be performed. A promising approach to overcome this limitation is to parallelize these algorithms. Thus we propose a parallel MOEA on consumer-level graphics processing units (GPU) to tackle the MOP. We perform experiments on a real-life direct marketing problem to compare the proposed method with the parallel hybrid genetic algorithm, the DMAX approach, and a sequential MOEA. It is observed that the proposed method is much more effective and efficient than the other approaches.

1 Introduction

How to improve marketing productivity or the return on marketing investment under resource constraints is one of the most challenging issues facing marketing professionals and researchers. The issue seems to be more pressing in hard economic times and given the increasing emphasis on customer relationship management—containing cost and channeling precious marketing resources

M.L. Wong (✉)
Department of Computing and Decision Sciences, Lingnan University, Tuen Mun, Hong Kong
e-mail: mlwong@ln.edu.hk

G. Cui
Department of Marketing and International Business, Lingnan University, Tuen Mun, Hong Kong
e-mail: gcui@ln.edu.hk

to the high-value customers who contribute greater profit to a company. Such situations include (1) upgrading customers—how to provide sizable incentives to the customers who are the most likely to upgrade and contribute greater profit over the long run? (2) modeling customer churn or retention—how to identify and prevent the most valuable customers from switching to a competitor? and (3) predicting loan default—how to predict the small minority who default on their large loans or credit card bills? This problem is particularly acute in direct marketing operations that typically have a fixed budget to target, from the vast list of customers in a company's database, those customers who are the most likely to respond to a marketing campaign and purchase greater amounts.

Most marketing activities, as espoused by marketing scholars and practitioners, are targeted marketing in nature—to reach customers who are the most responsive to marketing activities. Until recently, statistical methods such as regression and discriminant analysis have dominated the modeling of consumer responses to marketing activities. These methods, however, suffer from two shortcomings. First, methods based on OLS regression (i.e., mean regression) survey the entire population and focus on the average customer in estimating parameters. Segmentation methods, such as discriminant analyses, are not informative of their marketing responses. Thus, these methods by design are not entirely compatible with the objectives of targeted marketing. Second, researchers have so far focused on modeling either consumer responses or purchase quantity. Few models jointly consider consumers' responses and the revenue/profit that they generate.

These problems are particularly acute in modeling consumer responses to direct marketing and result in suboptimal performance of marketing campaigns. Conventional methods generate the predicted purchase probabilities for the entire population and do not focus on the top portion of the population, e.g., the top 20 % most attractive customers. This constraint is crucial as most firms have a limited marketing budget and can only target the most attractive customers. Thus, improving the accuracy of predicting purchase and potential sales or profit for these customers is crucial for augmenting the performance of targeted marketing. This is an urgent research problem given the increasing emphasis on customer relationship management and differentiating customers based on their profitability. Predicting loan default, customer churning, and service intervention represent other situations where resources are limited, budget constraints need to be considered, and targeted efforts are required.

To improve the accuracy of customer selection for targeted marketing, we formulate this problem as a constrained optimization problem. Recently, several researchers suggested using multi-objective evolutionary algorithms (MOEAs) to solve constrained optimization problems [24, 26].

However, MOEAs may execute for a long time for some difficult problems, because several objective value evaluations must be performed on huge datasets containing information about customers. Moreover, the non-dominance-checking and the non-dominated-selection procedures are also time-consuming. A promising approach to overcome this limitation is to parallelize these algorithms. In this chapter, we propose implementing a parallel MOEA for constrained optimization within the CUDA (Compute Unified Device Architecture) environment on an nVidia graphics processing unit (GPU). We perform experiments on a real-life direct marketing problem to compare our parallel MOEA with a parallel hybrid genetic algorithm (HGA) [29], the DMAX approach [1], and a sequential MOEA. It is observed that the parallel MOEA is much more effective and efficient than the other approaches. Since consumer-level GPUs are available in omnipresent personal computers and these computers are easy to use and manage, more people will be able to use our parallel MOEA to handle different real-life targeted marketing problems.

In the rest of the chapter, we first give an overview of constrained optimization for direct marketing, MOEAs, and GPU. In Sect. 3, our parallel MOEA for constrained optimization on GPU is discussed. The experiments and the results are reported in Sect. 4. In Sect. 5, conclusions and possible future research are discussed.

2 Overview

2.1 Constrained Optimization for Direct Marketing

Recent emphasis on customer relationship management require marketers to focus on the high-profit customers as in the 20/80 principle: 20 % of the customers account for 80 % profit of a firm. Thus, another purpose of direct marketing models is to predict the amount of purchase or profit from the buyers. However, the distribution of customer sales and profit data is also highly skewed with a very long tail, indicating a concentration of profit among a small group of customers [20]. In empirical studies of profit forecasting, the skewed distribution of profit data also creates problem for researchers. Given the limited and often a fixed marketing budget, the profit maximization approach to customer selection, which include only those customers with an expected marginal profit, is often not realistic [2]. Thus, the ultimate goal of target customer selection is to identify those customers who are the most likely to respond as well as contribute a larger amount of revenue or profit. Overall, unbalanced class distribution and skewed profit data, i.e., the small number of buyers and that of high-profit customers remain significant challenges in direct marketing forecasting [6]. Even a small percentage of improvement in the predictive accuracy in terms of customer purchase probability and profit can mean tremendous cost-savings and augment profit for direct marketers.

To date, only a few researchers have considered treating direct marketing forecasting as a problem of constrained optimization. Bhattacharyya [1] applied a genetic algorithm (GA) to a linear model that maximizes profitability at a given depth of file using the frontier analysis method. The DMAX model was built for a 10 %-of-file mailing. The decile analysis indicates the model has good performance as well as a very good representation of the data as evidenced by the total profit at the top decile. However, a closer look at the decile analysis reveals the model may not be as good as initially believed. The total profit shows unstable performance through the deciles, i.e., profit values do not decrease steadily through the deciles. This unstable performance, which is characterized by major "jumps" in several deciles, indicates the model is inadequately representing the data distribution and may not be reliable for decision support. The probable cause for this "lack-of-fit" is the violation of the assumption of normal distribution in the dependent variable.

Optimization of the classifier does not necessarily lead to maximization of return on investment (ROI), since maximization of the true-positive rate is often different from the maximization of sales or profit, which determines the ROI under a fixed budget constraint. To solve this problem, Yan and Baldasare [30] proposed an algorithm that uses gradient descent and the sigmoid function to maximize the monetary value under the budget constraint in an attempt to maximize the ROI. By comparison with several classification, regression, and ranking algorithms, they find that their algorithm may result in substantial improvement of the ROI.

2.2 Multi-objective Evolutionary Algorithms

We focus on, without loss of generality, minimization multi-objective problems in this chapter. However, either by using the *duality* principle [7] or by simple modifications to the domination definitions, these definitions and algorithms can be used to handle maximization or combined minimization and maximization problems.

For a multi-objective function Γ from $X(\subseteq \Re^N)$ to a finite set $Y(\subset \Re^m, m \geq 2)$, a decision vector $\mathbf{x}^* = [x^*(1), x^*(2), \cdots, x^*(N)]^T$ is *Pareto optimal* if and only if for any other decision vector $\mathbf{x} \in X$, their objective vectors $\mathbf{y}^* = \Gamma(\mathbf{x}^*) = [y^*(1), y^*(2), \cdots, y^*(m)]^T$ and $\mathbf{y} = \Gamma(\mathbf{x})$ holds either

$$y^*(i) \leq y(i) \text{ for any objective } i \ \ (1 \leq i \leq m)$$

or there exist two different objectives i, j such that

$$\left(y^*(i) < y(i)\right) \wedge \left(y(j)^* > y(j)\right).$$

Thus, for a Pareto optimal decision vector $\mathbf{x}^*$, there exists no decision vector $\mathbf{x}$ which would decrease some objective values without causing a simultaneous increase in at least one other objective. These Pareto optimal decision vectors are good trade-offs for the multi-objective optimization problem (MOP). For finding these vectors, dominance in the objective space plays an important role. An objective vector $\mathbf{y}_1 = \Gamma(\mathbf{x}_1) = [y_1(1), y_1(2), \cdots, y_1(m)]^T$ *dominates* another objective vector $\mathbf{y}_2 = \Gamma(\mathbf{x}_2)$ if and only if the former is partially less than the latter in each objective, i.e.,

$$\begin{cases} y_1(i) \leq y_2(i), & \forall i \in \{1, \cdots, m\} \\ y_1(j) < y_2(j), & \exists j \in \{1, \cdots, m\}. \end{cases} \tag{1}$$

It is denoted as $\mathbf{y}_1 \prec \mathbf{y}_2$. For notational convenience, $\mathbf{y}_1$ is defined to be *incomparable* with $\mathbf{y}_2$ if $\neg(\mathbf{y}_1 \prec \mathbf{y}_2 \vee \mathbf{y}_2 \prec \mathbf{y}_1 \vee \mathbf{y}_1 = \mathbf{y}_2)$. It is denoted as $\mathbf{y}_1 \sim \mathbf{y}_2$. We also denote $\neg(\mathbf{y}_1 \prec \mathbf{y}_2)$ as $\mathbf{y}_1 \not\prec \mathbf{y}_2$. That means $(\mathbf{y}_1 = \mathbf{y}_2 \vee \mathbf{y}_2 \prec \mathbf{y}_1 \vee \mathbf{y}_1 \sim \mathbf{y}_2)$.

Given the set of objective vectors Y, its *Pareto front* Y^* contains all vectors $\mathbf{y}^* \in Y$ that are not dominated by any other vector $\mathbf{y} \in Y$. That is, $Y^* = \{\mathbf{y}^* \in Y | \nexists \mathbf{y} \in Y, \mathbf{y} \prec \mathbf{y}^*\}$. We call its subset a *Pareto optimal set*. Each $\mathbf{y}^* \in Y^*$ is *Pareto optimal* or *non-dominated*. A Pareto optimal solution reaches a good trade-off among these conflicting objectives: one objective cannot be improved without worsening any other objective.

In the general case, it is impossible to find an analytic expression of the Pareto front. The normal procedure to find the Pareto front is to compute the objective values of decision vectors sufficiently enough and then determine the Pareto optimal vectors to form the Pareto front [4]. However, for many MOPs, the Pareto front Y^* is of substantial size, and the determination of Y^* is computationally prohibitive. Thus, the whole Pareto front Y^* is usually difficult to get and maintain. Furthermore, it is questionable to regard the whole Pareto front as an ideal answer [9, 19]. The value of presenting such a large set of solutions to a decision maker is also doubtful in the context of decision support. Usually, a set of representative Pareto optimal solutions are expected. Finally, in a solution set of bounded size, preference information could be used to steer the process to certain parts of the search space. Therefore, all practical implementations of MOEAs have maintained a bounded archive of best (non-dominated) solutions found so far [17].

A number of elitist MOEAs have been developed to address diversity of the archived solutions. The diversity exploitation mechanisms include mating restriction, fitness sharing (NPGA [13]), clustering (SPEA [35], SPEA2 [34]), nearest neighbor distance or crowding distance (NSGA-II [8]), crowding count (PAES [16], PESA-II [5], DMOEA [32]), or some preselection operators [7]. Most of them are quite successful, but they cannot ensure convergence to Pareto optimal sets.

2.3 Graphics Processing Units

The demand from the multimedia and games industries for accelerating 3D rendering has driven several graphics hardware companies devoted to the development of high-performance parallel graphics accelerator. This resulted in the birth of GPU, which handles the rendering requests using 3D graphics application programming interface (API). The whole pipeline consists of the transformation, texturing, illumination, and rasterization to the framebuffer. The need for cinematic rendering from the games industry further raised the need for programmability of the rendering process. Starting from the recent generation of GPUs launched in 2001 (including nVidia GeforceFX series and ATI Radeon 9800 and above), developers can write their own C-like programs, which are called *shaders*, on GPU by using a shading language. Due to the wide availability, programmability, and high-performance of these consumer-level GPUs, they are also cost-effective for many general purpose computations.

The shading languages are high-level programming languages and closely resemble to C. Most mathematical functions available in C are supported by the shading languages. Moreover, high-precision floating-point computation is supported on some GPUs. Hence, GPU can be utilized for speeding up the time-consuming computation in evolutionary algorithms (EAs). Since the shading languages were originally designed for applications in computer graphics, researchers should have knowledge about computer graphics, in order to use the languages effectively to develop different EAs.

Recently, the CUDA technology is developed [21]. It allows researchers to implement their GPU-based applications more easily. In CUDA, multiple threads can execute the same kernel program in parallel. Threads can access data from multiple memory spaces including the local, shared, global, constant, and texture memory. Because of the Single Instruction Multiple Thread (SIMT) architecture of GPU, certain limitations are imposed. Data-dependent for loop is not efficient because each thread may perform different number of iterations. Moreover, if-then-else construct is also inefficient, as the GPU will execute both true and false statements in order to comply with the SIMT design.

A number of GPU-based evolutionary programming (EP) [10,28], GAs [29], and genetic programming (GP) [3, 12, 18, 27] have been proposed by researchers.

3 Parallel MOEA for Constrained Optimization on Graphics Processing Units

We propose a learning algorithm to handle the constrained optimization and cost-sensitive problems. Let $E = \{e_1, e_2, \cdots, e_K\}$ be the set of K potential customers and $c(e_i)$, $1 \leq i \leq K$, be the amount of money spent by the customer e_i. Assume that r % of the customers will be solicited. If we can learn a regression function to predict their expected profits or induce a ranking function to arrange the cases in descending order according to their expected profits, we can solicit the first $\lceil K * r\,\% \rceil$ cases to achieve the goal of maximizing the total expected profits of the solicited cases. However, Yan and Baldasare [30] pointed out that this approach tackles an unnecessarily difficult problem and often results in poor performance.

On the other hand, we can learn a scoring function f that divides the K cases into two classes: U and L. The sizes of U and L should be $|U|$ and $|L|$, respectively. Consider a case e_i in U; its $f(e_i)$ must be greater than the scoring values of all cases in L. Moreover, the total of the expected profits of all cases e_i in U is maximized. In other words, we can formulate the learning problem as the following constrained optimization problem:

Find a scoring function f that

$$\max\left\{\sum_{e_i \in U} c(e_i)\right\}, U = \left\{e_i \in E \mid \nexists e_j \in L[f(e_i) \leq f(e_j)]\right\} \tag{2}$$

subject to

$$\begin{cases} |U| = \lceil K * r\,\% \rceil \\ |L| = K - \lceil K * r\,\% \rceil. \end{cases} \tag{3}$$

Since the orderings of all cases in U and all cases in L are insignificant to our objective, it would be easier to learn the scoring function that achieves an optimal partial ranking (ordering) of cases. Although the problem of learning the scoring function is easier, the procedure of finding U and L is still time-consuming because it is necessary to find the $(100 - r)$ percentile of E. Thus, we simplify the above constrained optimization problem to the following constrained optimization problem:

Find a scoring function f and a threshold τ that

$$\max \left\{ \sum_{e_i \in U} c(e_i) \right\}, U = \left\{ e_i \in E | f(e_i) > \tau \right\} \tag{4}$$

subject to

$$|U| = \lceil K * r\,\% \rceil. \tag{5}$$

We can convert the constrained optimization problem to an unconstrained MOP with two objectives [26],

$$\max\{\sum_{e_i \in U} c(e_i)\}, U = \{e_i \in E | f(e_i) > \tau\}, \tag{6}$$

$$\min\{\text{maximum}(0, |U| - \lceil K * r\,\% \rceil)\}. \tag{7}$$

By limiting f to be a linear function, a MOEA can be used to find the parameters of the scoring function f and the value of τ. We apply a parallel MOEA on GPU that finds a set of non-dominated solutions for a multi-objective function Γ that takes a vector $\mathbf{x}$ containing the parameters of the scoring function f as well as the value of τ and returns an objective vector $\mathbf{y}$, where $\mathbf{x} = [x(1), x(2), \cdots, x(N)]^T$, $\mathbf{y} = [y(1), y(2), \cdots, y(m)]^T$, N is 1 plus the number of the parameters of f, and m is the number of objectives. The algorithm is given in Fig. 1.

In the algorithm given in Fig. 1, $\mathbf{x_i}$ is a vector of variables evolving and $\boldsymbol{\eta}_\mathbf{i}$ controls the vigorousness of mutation of $\mathbf{x_i}$. Firstly, an initial population is generated and the objective values of the individuals in the initial population are calculated by using the multi-objective function Γ.

Next, the rankings and the crowding distances of the individuals are found. All non-dominated individuals will be assigned a ranking of 0. The crowding distance of a non-dominated individual is the size of the largest cuboid enclosing it without including any other non-dominated individuals. In order to find the rankings

```
1. Set $t$, the generation count, to 0.
2. Generate the initial population $P(t)$ of $\mu$ individuals, each of which can
   be represented as a set of real vectors, $(\mathbf{x_i}, \boldsymbol{\eta_i}), i = 1, \ldots, \mu$. Both $\mathbf{x_i}$ and
   $\boldsymbol{\eta_i}$ contain $N$ independent variables,
     $\mathbf{x_i} = \{x_i(1), \cdots, x_i(N)\}$,
     $\boldsymbol{\eta_i} = \{\eta_i(1), \cdots, \eta_i(N)\}$.
3. Evaluate the objective vectors of all individuals in $P(t)$ by using the multi-
   objective function.
4. Calculate the rankings and crowding distances of all individuals
   a. Execute **Dominance-checking**($P(t)$, $C$, $S$).
   b. Execute **Non-dominated-selection**($P(t)$, $C$, $S$, $V$, $\mu$).
5. While the termination condition is not satisfied
   a. For $i$ from 1 to $\mu/2$, select two parents $P^1_{parent_i 1}$ and $P^1_{parent_i 2}$ from $P(t)$
      using the tournament selection method.
   b. For $i$ from 1 to $\mu/2$, recombine $P^1_{parent_i 1}$ and $P^1_{parent_i 2}$ using single point
      crossover to produce two offspring that are stored in the temporary
      population $P^2$. The population $P^2$ contains $\mu$ individuals.
   c. Mutate individuals in $P^2$ to generate modified individuals that are
      stored in the temporary population $P^3$. For an individual $P^2_i = (\mathbf{x_i}, \boldsymbol{\eta_i})$,
      where $i = 1, \ldots, \mu$, create a new individual $P^3_i = (\mathbf{x_i}', \boldsymbol{\eta_i}')$ as follows:
      for $j = 1, \ldots, N$
        $x'_i(j) = x_i(j) + \eta_i(j) R(0,1)$,
        $\eta'_i(j) = \eta_i(j) \exp(\frac{1}{\sqrt{2N}} R(0,1) + \frac{1}{\sqrt{2\sqrt{N}}} R_j(0,1))$
      where $x_i(j), \eta_i(j), x'_i(j)$, and $\eta'_i(j)$ denote the $j^{th}$ component of
      $\mathbf{x_i}, \boldsymbol{\eta_i}, \mathbf{x_i}'$, and $\boldsymbol{\eta_i}'$ respectively. $R(0,1)$ denotes a normally distributed
      1D random number with zero mean and standard deviation of one.
      $R_j$(0,1) indicates a new random value for each value of $j$.
   d. Evaluate the objective vectors of all individuals in $P^3$.
   e. Combine the parent population $P(t)$ with $P^3$ to generate a population
      $P^4$ containing $2\mu$ individuals.
   f. Check the dominance of all individuals in $P^4$ by executing **Dominance-
      checking**($P^4$, $C$, $S$).
   g. Select $\mu$ individuals from $P^4$ and store them in the next population
      $P(t+1)$. The individuals are selected by executing **Non-dominated-
      selection**($P^4$, $C$, $S$, $V$, $\mu$).
   h. $t = t + 1$.
6. Return the non-dominated individual with the smallest value for the sec-
   ond objective in the last population.
```

Fig. 1 The MOEA algorithm

```
Procedure Dominance-checking($Pop$, $C$, $S$)

• $Pop$ be the input population.
• $C$ be an array of integers.
• $S$ be an array of sets of integers.

1. Set $k$ be the number of individuals in the population $Pop$.
2. For $i$ from 1 to $k$
   • Set $P_i$ be the $i^{th}$ individual of $Pop$.
   • Set $C_i$, the $i^{th}$ element of $C$, be 0.
   • Set $S_i$, the $i^{th}$ element of $S$, be an empty set.
   • For $j$ from 1 to $k$
     – If $i$ is not equal to $j$, then
           If $P_i$ is dominated by $P_j$, then
               increase $C_i$ by 1
           else if $P_j$ is dominated by $P_j$, then
               $S_i = S_i \cup \{j\}$.
```

Fig. 2 The **dominance-checking** algorithm

and the crowding distances of other individuals, the non-dominated individuals are assumed to be removed from the population and thus another set of non-dominated individuals can be obtained. The rankings of these individuals should be larger than those of the previous non-dominated individuals. The crowding distances of the individuals can also be found. Similarly, the same approach can be applied to find the rankings and the crowding distances of all other individuals. The procedures **dominance checking** and **non-dominated selection** are used to find these values. Their algorithms are given in Figs. 2 and 3, respectively.

Then, $\mu/2$ pairs of parents will be selected from the population. Two offspring will be generated for each pair of parents by using crossover and mutation. In other words, there will be μ offspring. The objective vectors of all offspring will be obtained, and the parent population will be combined with the μ offspring to generate a selection pool. Thus there are 2μ individuals in the selection pool. The rankings and the crowding distances of all individuals in the selection pool can be obtained by using the **dominance-checking** and **non-dominated-selection** procedures. μ Individuals will be selected from the selection pool, and they will form the next population of individuals. This evolution process will be repeated until the termination condition is satisfied.

Finally, the non-dominated individual with the smallest value for the second objective in the last population will be returned. In general, the computation of the parallel MOEA can be roughly divided into five types: (1) Fitness value evaluation (steps 3 and 5(d)) (2) Parent selection (step 5(a)) (3) Crossover and mutation (steps 5(b) and 5(c), respectively) (4) The **dominance-checking** procedure designed for parallel algorithms (steps 4(a) and 5(f)) (5) The **non-dominated-selection**

```
**Procedure Non-dominated-selection**($Pop$, $C$, $S$, $V$, $L$)
• $Pop$ be the input population.
• $C$ be an input array of integers.
• $S$ be an input array of sets of integers.
• $V$ be an array of numbers. If the $i^{th}$ element is 1, the $i^{th}$ individual is
  selected, otherwise it is not selected.
• $L$ be the number of individuals to be selected.

1. Set $Selected$ to 0. It represents the number of individuals that are selected.
2. Set all elements of $V$ to 0.
3. Set $rank$, the current ranking of individuals, to 0.
4. Set $F$, the current non-dominated set, to be an empty set.
5. For each $P_i \in Pop$
       If the corresponding $C_i$ is 0, then
           $F = F \cup \{P_i\}$.
6. Calculate the crowding distances of all individuals in $F$.
7. While $Selected + |F| \leq L$
   a. Set $NF$, the next non-dominated set, to be an empty set.
   b. For each $P_i \in F$
        i. Set the ranking of $P_i$ to $rank$.
       ii. Set the corresponding $V_i$ to 1.
      iii. For each $k \in S_i$
             • Decrease $C_k$ by 1.
             • If $C_k$ is 0, then set $NF = NF \cup \{P_k\}$.
       iv. Increase $Selected$ by 1.
   c. Calculate the crowding distances of all individuals in $NF$.
   d. Increase $rank$ by 1.
   e. Set $F$ to $NF$.
8. If $Selected < L$, then
   • Based on the crowding distances of individuals in $F$, select ($L -$
     $Selected$) individuals.
   • Set their rankings to $rank$.
   • Set their $V_i$ to 1.
```

Fig. 3 The **non-dominated-selection** algorithm

procedure which selects individuals from the selection pool (steps 4(b) and 5(g)) These operations will be discussed in the following subsections.

3.1 Data Organization

Suppose we have μ individuals and each contains N variables. The most natural representation for an individual is an array. Figure 4 shows how we represent μ

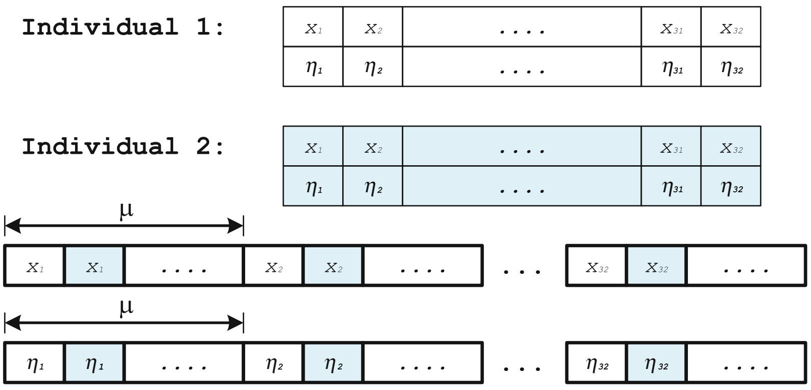
Individual 1:
Individual 2:
x_1 x_2 .... x_{31} x_{32}
η_1 η_2 .... η_{31} η_{32}
μ
x_1 x_1 .... x_2 x_2 .... x_{32} x_{32} ....
η_1 η_1 .... η_2 η_2 .... η_{32} η_{32} ....

Fig. 4 Representing individuals of 32 variables on global memory

individuals in the global memory. Without loss of generality, we take $N = 32$ as an example of illustration throughout this chapter.

Since the global memory space is not cached in some GPUs,[1] it is important to use the right access pattern to get maximum memory bandwidth. When the concurrent memory accesses by 16 CUDA threads in a half wrap (for GPUs of compute capability 1.x) or 32 threads in a warp (for GPUs of compute capability 2.x) [2] can be coalesced into a single memory transaction, the global memory bandwidth can be improved [21]. In order to fulfill the requirements for coalesced memory accesses, the same variables from all individuals are grouped and form a tile of μ values in the global memory as shown in Fig. 4. On the other hand, the efficiency of accessing the same variables of all individuals in parallel will be reduced, if an individual is mapped to 32 consecutive locations, because the simultaneous memory accesses cannot be coalesced and multiple memory transactions are required.

3.2 Fitness Value Evaluation

In steps 3 and 5(d) of Fig. 1, the objective vectors of all individuals in the initial population and all offspring in the temporary population P^3 are calculated. Each CUDA thread returns an objective vector by feeding the multi-objective function Γ with the decision variables of the individual. This evaluation process usually consumes significant part of the computational time.

[1]Cache for global memory is only available in GPUs of compute capability 2.x.

[2]In CUDA, the GPU creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. A half-warp is either the first or second half of a warp.

Since no interaction among threads is required during evaluation, the evaluation is fully parallelizable. Recall that the individuals are broken down and stored in the tiles within the global memory. The evaluation kernel looks up the corresponding variable in each tile during the evaluation. The objective vectors are saved in an output array of size $m \times \mu$, because each thread generates a vector of m values.

3.3 Parent Selection

The selection process determines which individuals will be selected as parents to reproduce offspring. Different selection methods including the *roulette wheel selection, truncation selection*, and *stochastic tournament* have been applied in the field [11]. The stochastic tournament is employed in our parallel MOEA, because it is not practical to implement a parallel method on GPU to collect statistical information on the whole population. Since this information is not required in the stochastic tournament while it is needed for the other two methods, the stochastic tournament is more suitable for GPU.

In the tournament selection method, two groups of q individuals are randomly chosen from the population for each CUDA thread. The number q is the tournament size. The two individuals with the smallest rankings within the two groups will be selected as the parents to produce offspring by using crossover and mutation. If more than one individual in a group has the smallest ranking, the one with the largest crowding distance will be chosen. A GPU-based random number generator [14, 22] is used to generate a large number of random numbers stored in the global memory. These random numbers can then be used for selecting individuals randomly. Since there are $\mu/2$ CUDA threads (see step 5(a) of Fig. 1), $\mu \times q$ random numbers are used.

We implement our parent selection method in a kernel program. The input of the kernel is the arrays containing the rankings and the crowding distances of the individuals, as well as the array containing the random numbers. While the output of the kernel is the addresses of the breeding parents selected. The addresses of all selected parents are stored in an output array of size μ.

3.4 Crossover and Mutation

The selection operator focuses on searching promising regions of the solution space. However, it is not able to introduce new solutions that are not in the current population. In order to escape from local optima and introduce larger population diversity, the crossover and mutation operators are applied.

We apply single-point crossover in our parallel MOEA. The kernel program takes input arrays containing the addresses of the selected parents, the individuals, and the random numbers. It generates μ offspring individuals that are stored in the global memory.

To accomplish the mutation process on GPU, we designed two kernel programs, one for computing $\mathbf{x}'$ and the other for $\boldsymbol{\eta}'$. They implement the Cauchy mutation method proposed by Yao and Liu [31]. The individuals $\mathbf{x_i}$ and $\boldsymbol{\eta}_\mathbf{i}$ are stored in two input arrays while the mutated offspring are generated and written to two output arrays $\mathbf{x_i}'$ and $\boldsymbol{\eta}'_\mathbf{i}$. Besides, random numbers stored in the global memory are used by the two kernels.

3.5 Dominance Checking

We implement the fast dominance-checking procedure applied in NSGA-II [8]. For each individual in the population, two entities are evaluated: C_i is the number of the other individuals that dominates the ith individual and S_i is the set of the other individuals that are dominated by the ith individual. Thus the ith individual is non-dominated if the corresponding C_i is 0. This procedure is efficient because only $O(\mu^2)$ dominance comparisons should be performed.

Suppose that there are 2μ individuals (step 5(f) of Fig. 1), the C_i is stored in an array of short integer. On the other hand, the S_i are represented in a 2D array of bit, S. If the jth bit of the ith row of S is 1, the jth individual is dominated by the ith individual. The size of C is 2μ short integers, while that of S is $4\mu^2$ bits. For example, their sizes are 16 KB and 8 MB, respectively, if μ is 4,096. Their sizes are, respectively, 64 KB and 128 MB if μ is 16,384.

The kernel program implementing the procedure of Fig. 2 takes an input array of objective vectors of all individuals and produces C and S in the global memory. Since there is no interaction among the CUDA threads, they can efficiently execute this kernel in parallel.

Because of the reasons described in Sect. 3.6, the objective vectors, C and S, are transferred from the global memory to the CPU memory after computing C and S.

3.6 Non-dominated Selection

Instead of executing non-dominated selection on GPU, this procedure is performed on CPU because of a number of reasons. First, the number of individuals in F (step 7(b) of Fig. 3) varies from one iteration to other iterations. Some CUDA threads may be idle in some iterations. Second, it is necessary to sort the individuals in NF (step 7(c) of Fig. 3) according to their objective vectors, in order to calculate their crowding distances. Although it is possible to execute a sorting algorithm on GPU [15], its efficiency is doubtful when it is used to sort a relatively small number of values. Third, many synchronizations may be performed as the variables C_k and NF may be accessed and modified by different threads concurrently (step 7(b)iii of Fig. 3).

The CPU implementation of the procedure accesses the objective vectors of individuals, C and S, that are already stored in the CPU memory. It finds the rankings and the crowding distances and summarizes the selected individuals in the output array V. Then, the rankings, the crowding distances, and V are transferred to the GPU memory.

Based on the information stored in V, the actual replacements of individuals, the rankings, and the crowding distances are performed on GPU. Thus, the data movement between the GPU memory and the CPU memory can be reduced, because it is not necessary to transfer the individuals between the two memory spaces.

In summary, the whole MOEA program, except the non-dominated-selection procedure, is executed on GPU. Thus, our parallel MOEA gains the most benefit from the SIMT architecture of GPU.

4 Experiments

In this section, the parallel MOEA is applied to a data mining problem in direct marketing. The objective of the problem is to predict potential prospects from the buying records of previous customers. Advertising campaign, which includes mailing of catalogs or brochures, is then targeted on the group of potential prospects. Hence, if the prediction is accurate, it can help to enhance the response rate of the advertising campaign and increase the ROI. The direct marketing problem requires ranking the customer database according to the customers' scores obtained by the prediction models [23].

We compare the parallel MOEA, the parallel HGA [29], and the DMAX approach for learning prediction models. Since the parallel HGA is a single-objective optimization algorithm, it is used to optimize the objective defined in (6). The experiment test bed was an Intel Pentium Dual E2220 CPU with an nVidia GTX 460 display card, with 2,048 MB main memory and 768 MB GPU memory. The CPU speed is 2.40 GHz and the GPU contains 336 unified shaders. Microsoft Windows XP Professional, Microsoft Visual C++ 2008, and nVidia CUDA version 3.1 are used to develop the parallel MOEA and the parallel HGA. On the other hand, the DMAX approach is developed in Java. The following parameters have been used in the experiments:

- Population size: $\mu = 256$
- Tournament size: $q = 2$
- Maximum number of generation: $G = 500$
- The percentage of customers to be solicited: $r\,\% = 20\,\%$

4.1 Methodology

The prediction models are evaluated on a large real-life direct marketing dataset from a US-based catalog company. It sells multiple product lines of merchandise including gifts, apparel, and consumer electronics. This dataset contains the records of 106,284 consumers in a recent promotion as well as their purchase history over a 12-year period. Furthermore, demographic information from the 1995 US Census and credit information from a commercial vendor were appended to the main dataset. Altogether, each record contains 361 variables. The most recent promotion sent a catalog to every customer in this dataset and achieved a 5.4 % response rate, representing 5,740 buyers.

Typical in any data mining process, it is necessary to reduce the dimension of the dataset by selecting the attributes that are considered relevant and necessary. Towards this feature selection process, there are many possible options. For instance, we could use either a *wrapper* selection process or a *filter* selection process [25]. In a wrapper selection process, different combinations are iteratively tried and evaluated by building an actual model out of the selected attributes. In a filter selection process, certain evaluation function, which is based on information theory or statistics, is defined to score a particular combination of attributes. Then, the final combination is obtained in a search process. In this experiment, we have applied the forward selection method to select 17 variables, that are relevant to prediction, out of the 361 variables.

Since direct marketers usually have a fixed budget and can only contact a small portion of the potential customers in their dataset (e.g., top 20 %), simple error rates or overall classification accuracy of models are not meaningful. To support direct marketing and other targeted marketing decisions, maximizing the number of true positives at the top deciles is usually the most important criterion for assessing the performance of prediction models [1, 33].

To compare the performance of different prediction models, we use decile analysis which estimates the enhancement of the response rate and the profit for marketing at different depth-of-file. Essentially, the ranked list is equally divided into ten deciles. Customers in the first decile are the top-ranked customers that are most likely to give response and generate high profit. On the other hand, customers in the tenth decile are ranked lowest. To measure the performance of a model at different depths of file, direct marketing researchers have relied on the "lift," which is the ratio of true positives to the total number of records identified by the model in comparison with that of a random model at a specific decile of the file. Thus, comparing the performance of models across depths of file using cumulative lifts or the "response rate" are necessary to inform decisions in direct marketing. Profit lift is the amount of extra profit generated with the new method over that generated by a random method. In this sense, the goal to achieve higher lifts in the upper deciles becomes a ranking problem based on the scores returned by the model and help to evaluate the effectiveness of targeted marketing and to forecast sales and profitability of promotion campaigns.

4.2 Cross-Validation Results

In order to compare the robustness of the prediction models, we adopt a ten-fold cross-validation approach for performance estimation. A dataset is randomly

Table 1 Cumulative lifts of the models learned by different methods

Decile	Parallel MOEA	Parallel HGA	DMAX
0	**358.47 (25.11)**	147.53 (16.84)$^+$	310.60 (34.10)$^+$
1	**270.46 (9.54)**	132.51 (6.98)$^+$	234.20 (20.50)$^+$
2	**219.11 (6.58)**	125.68 (5.82)$^+$	195.50 (12.30)$^+$
3	**182.48 (3.92)**	120.65 (5.88)$^+$	170.60 (6.30)$^+$
4	**156.11 (2.54)**	115.84 (4.70)$^+$	150.90 (3.90)
5	**138.17 (2.51)**	111.59 (3.42)$^+$	136.60 (2.90)
6	124.95 (2.99)	109.12 (2.69)$^+$	**125.20 (2.10)**
7	114.51 (2.63)	106.17 (1.88)$^+$	**115.40 (1.60)**
8	106.77 (1.18)	103.66 (0.84)$^+$	**106.90 (1.20)**
9	100.00 (0.00)	100.00 (0.00)	100.00(0.00)

partitioned into 10 mutually exclusive and exhaustive folds. Each time, a different fold is chosen as the test set, and other nine folds are combined together as the training set. Prediction models are learned from the training set and evaluated on the corresponding test set.

In Table 1, the average of the cumulative lifts of the models learned by different methods are summarized. Numbers in the parentheses are the standard deviations. The highest cumulative lift in each decile is highlighted in bold. The superscript $^+$ represents that the cumulative lift of the model obtained by the parallel MOEA is significant higher at 0.05 level than that of the models obtained by the corresponding methods. The superscript $^-$ represents that the cumulative lift of the model obtained by the parallel MOEA is significantly lower at 0.05 level than that of the corresponding models.

From Table 1, the models generated by the parallel MOEA have the average cumulative lifts of 358.47 and 270.46 in the first two deciles, respectively, suggesting that by mailing to the top two deciles alone, the models generate over twice as many respondents as a random mailing without a model. Moreover, the average cumulative lifts of the models learnt by the parallel MOEA are significantly higher than those of the models obtained by the other methods for the first four deciles.

The average of the cumulative profit lifts of the models learned by different methods are summarized in Table 2. It is observed that the average cumulative profit lifts of the models learnt by the parallel MOEA are significantly higher than those of the models obtained by the other methods for the first three deciles. The average profits for different models are listed in Table 3. Direct marketers can get \$11,461.63 if they use the parallel MOEA to generate models for selecting 20 % of the customers from the dataset. On the other hand, they can get only \$10,514.24 if they apply the DMAX approach for selecting customers. The parallel HGA cannot learn good models because the objective (i.e., (7)) representing the constraint is not considered in this approach.

In order to study the effect of the value of r on the performance of the models learnt by the parallel MOEA, we apply different values of r and compare

Table 2 Cumulative profit lifts of the models learned by different methods

Decile	Parallel MOEA	Parallel HGA	DMAX
0	**621.00 (49.16)**	242.35 (38.54)$^+$	550.80 (61.00)$^+$
1	**382.03 (14.14)**	188.26 (16.5)$^+$	350.80 (24.10)$^+$
2	**278.72 (10.26)**	166.24 (9.71)$^+$	261.70 (15.70)$^+$
3	**221.96 (8.05)**	151.66 (12.04)$^+$	213.30 (7.40)
4	**181.81 (4.90)**	139.39 (9.96)$^+$	179.10 (5.30)
5	155.32 (5.27)	129.93 (5.91)$^+$	**156.30 (4.20)**
6	135.55 (4.51)	121.79 (4.74)$^+$	**137.60 (3.90)**
7	121.20 (3.71)	114.63 (2.95)$^+$	**122.10 (3.00)**
8	**110.20 (1.57)**	108.2 (1.16)$^+$	109.70 (2.00)
9	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)

Table 3 Average profits for the models learned by different methods

Decile	Parallel MOEA	Parallel HGA	DMAX
0	**$9,339.04**	$3,646.49	$8,254.34
1	**$11,461.63**	$5,650.60	$10,514.24
2	**$12,545.96**	$7,472.12	$11,765.58
3	**$13,317.80**	$9,077.67	$12,786.13
4	**$13,631.38**	$10,429.06	$13,420.04
5	$13,974.12	$11,677.79	**$14,053.96**
6	$14,234.71	$12,768.95	**$14,434.60**
7	$14,542.35	$13,744.59	**$14,638.41**
8	**$14,867.79**	$14,588.65	$14,795.77
9	$14,986.09	$14,986.09	$14,986.09

Table 4 Cumulative lifts of the models learned by the parallel MOEA

Decile	$r = 10\,\%$	$r = 30\,\%$	$r = 40\,\%$	$r = 50\,\%$
0	363.28 (24.59)	354.39 (27.66)	363.28 (24.59)	363.28 (24.59)
1	267.32 (8.40)	267.02 (12.43)	267.32 (8.40)	267.32 (8.40)
2	214.89 (5.59)	220.53 (7.28)	214.89 (5.59)	214.89 (5.59)
3	180.27 (4.61)	185.51 (4.78)	180.27 (4.61)	180.27 (4.61)
4	155.63 (3.76)	161.84 (3.74)	155.63 (3.76)	155.63 (3.76)
5	137.96 (3.03)	143.03 (1.98)	137.96 (3.03)	137.96 (3.03)
6	124.69 (2.68)	128.90 (1.85)	124.69 (2.68)	124.69 (2.68)
7	114.10 (1.95)	117.33 (1.48)	114.10 (1.95)	114.10 (1.95)
8	106.16 (1.50)	108.72 (0.85)	106.16 (1.50)	106.16 (1.50)
9	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)

the cumulative lifts and the cumulative profit lifts of the induced models. From Tables 4 and 5, it is found that our approach is quite stable because it can learn good models for different values of r.

Table 5 Cumulative profit lifts of the models learned by the parallel MOEA

Decile	$r = 10\,\%$	$r = 30\,\%$	$r = 40\,\%$	$r = 50\,\%$
0	621.08 (46.31)	616.99 (51.25)	621.08 (46.31)	621.08 (46.31)
1	377.88 (14.76)	377.51 (18.49)	377.88 (14.76)	377.88 (14.76)
2	276.31 (9.96)	281.15 (11.13)	276.31 (9.96)	276.31 (9.96)
3	217.91 (6.91)	222.98 (10.05)	217.91 (6.91)	217.91 (6.91)
4	181.69 (5.75)	185.63 (6.98)	181.69 (5.75)	181.69 (5.75)
5	154.24 (4.39)	158.46 (3.67)	154.24 (4.39)	154.24 (4.39)
6	134.84 (4.25)	139.18 (3.18)	134.84 (4.25)	134.84 (4.25)
7	119.91 (2.99)	123.25 (2.36)	119.91 (2.99)	119.91 (2.99)
8	108.40 (2.52)	111.31 (0.82)	108.40 (2.52)	108.40 (2.52)
9	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)

Table 6 The average execution time (in seconds) of the CPU implementation

	Generation								
	100	150	200	250	300	350	400	450	500
OT	129.29	194.15	259.10	324.09	389.11	454.15	519.20	584.29	648.11
DC	0.50	0.79	1.06	1.34	1.60	1.88	2.15	2.41	2.67
NS	0.01	0.02	0.02	0.03	0.03	0.05	0.06	0.06	0.07
FE	128.78	193.35	258.02	322.73	387.48	452.22	516.99	581.82	645.37
ratio	0.9960	0.9959	0.9958	0.9958	0.9958	0.9958	0.9958	1.00	1.00

The OT, DC, NS, and FE rows show the average time in performing, respectively, all steps, the dominance-checking step, the non-dominated-selection step, and fitness evaluations

Table 7 The average execution time (in seconds) of the GPU implementation

	Generation								
	100	150	200	250	300	350	400	450	500
OT	5.75	8.55	11.35	14.14	16.93	19.72	22.51	25.31	28.04
DC	0.03	0.04	0.06	0.07	0.09	0.11	0.14	0.16	0.17
NS	0.03	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.11
FE	5.68	8.46	11.23	14.00	16.77	19.52	22.28	25.05	27.75
ratio	0.989	0.989	0.989	0.990	0.990	0.990	0.989	0.990	0.990

4.3 Comparison Between GPU and CPU Approaches

We compare the CPU and the GPU implementations of the MOEA. The average execution time of different steps of the CPU implementation is summarized in Table 6. The ratios of the time used in fitness evaluations to the overall execution time are also reported in this table. It can be observed that the fitness evaluation time is significantly higher than that of the other steps because the training sets are very large. The average execution time of the GPU implementation is summarized in Table 7. The parallel MOEA takes about 28 s to learn a model. On the other hand, it takes about 648 and 7,315 s, respectively, for the CPU implementation of the MOEA and the DMAX approach to learn a model.

Table 8 The speedups of the GPU implementation with the CPU implementation

	Generation								
	100	150	200	250	300	350	400	450	500
OT	22.505	22.713	22.835	22.919	22.978	23.024	23.058	23.085	23.115
DC	15.317	18.740	17.542	19.153	18.368	16.782	15.007	15.457	15.440
NS	0.354	0.305	0.346	0.350	0.388	0.527	0.572	0.542	0.602
FE	22.669	22.869	22.982	23.054	23.111	23.167	23.209	23.231	23.255

Table 8 displays the speedups of the overall programs and different steps of the programs. The speedups of the GPU implementation of the dominance-checking procedure range from 15.00 to 18.74. On the other hand, the Pnon-dominated-selection procedure of the GPU implementation is slower than that of the CPU approach. The overall speedup is about 23.1.

Since a marketing campaign often involves huge dataset and large investment, prediction models which can categorize more prospects into the target list are valuable as they will enhance the response rate as well as the ROI. From the experimental results, the prediction models generated by the parallel MOEA are more effective than the other models and the parallel MOEA is significantly faster than the DMAX approach.

5 Conclusions

An important issue in targeted marketing is how to find potential customers who contribute large profits to a firm under constrained resources. In this chapter, we have proposed a data mining method to learn models for identifying valuable customers. We have formulated this learning problem as a constrained optimization problem of finding a scoring function f and a threshold value τ. We have then converted it to an unconstrained MOP.

By limiting f to be a linear function, a parallel MOEA on GPU has been used to handle the MOP and find the parameters of f as well as the value of τ. We have used tenfold cross-validation and decile analysis to compare the performance of the parallel MOEA, the parallel HGA, and the DMAX approach for a real-life direct marketing problem. Based on the cumulative lifts, cumulative profit lifts, and average profits, it can be concluded that the models generated by the parallel MOEA significantly outperform the models learnt by other methods in many deciles. Thus, the parallel MOEA is more effective. Moreover, it is significantly faster than the DMAX approach.

We have performed experiments to compare our parallel MOEA and a CPU implementation of MOEA. It is found that the overall speedup is about 23.1. Thus, our approach will be very useful for solving difficult direct marketing problems that involve large datasets and require huge population sizes.

For future work, we will extend our method to learn nonlinear scoring functions and apply it to other targeted marketing problems under resource constraints.

Acknowledgements This work is supported by Hong Kong RGC General Research Fund LU310111.

References

1. Bhattacharyya, S.: Direct marketing performance modeling using genetic algorithms. INFORMS J. Comput. **11**(3), 248–257 (1999)
2. Bult, J.R., Wansbeek, T.: Optimal selection for direct mail. Manag. Sci. **14**(4), 1362–1381 (1995)
3. Chitty, D.M.: A data parallel approach to genetic programming using programmable graphics hardware. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO ’07), vol. 2, pp. 1566–1573 (2007)
4. Coello Coello, C.A., Toscano Pulido, G., Salazar Lechuga, M.: Handling multiple objectives with particle swarm optimization. IEEE Trans. Evol. Comput. **8**(3), 256–279 (2004)
5. Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J.: PESA-II: region-based selection in evolutionary multiobjective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’2001), pp. 283–290 (2001)
6. Cui, G., Wong, M.L.: Implementing neural networks for decision support in direct marketing. Int. J. Market Res. **46**(2), 1–20 (2004)
7. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, New York (2001)
8. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
9. Fieldsend, J.E., Everson, R.M., Singh, S.: Using unconstrained elite archives for multiobjective optimization. IEEE Trans. Evol. Comput. **7**(3), 305–323 (2003)
10. Fok, K.L., Wong, T.T., Wong, M.L.: Evolutionary computing on consumer-level graphics hardware. IEEE Intell. Syst. **22**(2), 69–78 (2007)
11. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
12. Harding, S., Banzhaf, W.: Fast genetic programming on GPUs. In: Proceedings of the 10th European Conference on Genetic Programming (EuroGP’2007), pp. 90–101 (2007)
13. Horn, J., Nafpliotis, N., Goldberg, D.E.: A niched Pareto genetic algorithm for multiobjective optimization. In: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, vol. 1, pp. 82–87 (1994)
14. Howes, L., Thomas, D.: Efficient random number generation and application using CUDA. In: Nguyen, H. (ed.) GPU Gems 3, pp. 805–830. Addison-Wesley, Reading (2007)
15. Kipfer, P., Westermann, R.: Improved GPU Sorting. In: Pharr, M. (ed.) GPU Gems 2, pp. 733–746. Addison-Wesley, Reading (2005)
16. Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the Pareto Archived Evolution Strategy. Evol. Comput. **8**(2), 149–172 (2000)
17. Knowles, J.D., Corne, D.W.: Properties of an adaptive archiving algorithm for storing nondominated vectors. IEEE Trans. Evol. Comput. **7**(2), 100–116 (2003)
18. Langdon, W.B., Banzhaf, W.: A SIMD interpreter for genetic programming on GPU graphics cards. In: Proceedings of the 11th European Conference on Genetic Programming (EuroGP’2008), pp. 73–85 (2008)
19. Laumanns, M., Thiele, L., Deb, K., Zitzler, E.: Combining convergence and diversity in evolutionary multi-objective optimization. Evol. Comput. **10**(3), 263–282 (2002)
20. Mulhern, F.J.: Customer profitability analysis: measurement, concentration, and research directions. J. Interact. Market. **13**(1), 25–40 (1999)
21. nVidia: NVIDIA CUDA C Programming Guide Version 3.1. Technical Report, nVidia Corporate (2010). http://developer.nvidia.com/object/cuda.html

22. Pang, W.M., Wong, T.T., Heng, P.A.: Generating massive high-quality random numbers using GPU. In: Proceedings of the 2008 Congress on Evolutionary Computation (CEC 2008), pp. 841–847 (2008)
23. Rud, O.P.: Data Mining Cookbook: Modeling Data for Marketing, Risk and Customer Relationship Management. Wiley, New York (2001)
24. Runarsson, T.P., Yao, X.: Search biases in constrained evolutionary optimization. IEEE Trans. Syst. Man Cybern. C **35**(2), 233–243 (2005)
25. Singh, M.: Learning Bayesian networks for solving real-world problems. Ph.D. thesis, University of Pennsylvania (1998)
26. Wang, Y., Cai, Z., Guo, G., Zhou, Y.: Multiobjective optimization and hybrid evolutionary algorithm to solve constrained optimization problems. IEEE Trans. Syst. Man Cybern. B **37**(3), 560–575 (2007)
27. Wilson, G., Banzhaf, W.: Linear genetic programming GPGPU on Microsoft's Xbox 360. In: Proceedings of the 2008 Congress on Evolutionary Computation (CEC'2008), pp. 378–385 (2008)
28. Wong, M.L., Wong, T.T., Fok, K.L.: Parallel evolutionary algorithms on graphics processing unit. In: Proceedings of the 2005 Congress on Evolutionary Computation (CEC'2005), pp. 2286–2293 (2005)
29. Wong, M.L., Wong, T.T., Fok, K.L.: Parallel hybrid genetic algorithms on consumer-level graphics hardware. In: Proceedings of the 2006 Congress on Evolutionary Computation (CEC'2006), pp. 10330–10337 (2006)
30. Yan, L., Baldasare, P.: Beyond classification and ranking: constrained optimization of the ROI. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 948–953 (2006)
31. Yao, X., Liu, Y.: Fast evolutionary programming. In: Evolutionary Programming V: Proceedings of the 5th Annual Conference on Evolutionary Programming. MIT Press, Cambridge (1996)
32. Yen, G.G., Lu, H.: Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation. IEEE Trans. Evol. Comput. **7**(3), 253–274 (2003)
33. Zahavi, J., Levin, N.: Applying neural computing to target marketing. J. Direct Market. **11**(4), 76–93 (1997)
34. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm. In: Giannakoglou, K., Tsahalis, D., Periaux, J., Papailou, P., Fogarty, T. (eds.) EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, pp. 95–100. International Center for Numerical Methods in Engineering, Barcelona (2002)
35. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999)