Applied Soft Computing xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A constraint-guided method with evolutionary algorithms for economic problems

Nanlin Jin [a], Edward Tsang [b,*], Jin Li [c]

[a] University of Leeds, School of Geography, Leeds LS2 9JT, UK
[b] University of Essex, School of Computer Science and Electronic Engineering, Colchester CO4 3SQ, UK
[c] Unilever, UK

ARTICLE INFO

Article history:
Received 30 September 2007
Received in revised form 17 October 2008
Accepted 15 November 2008
Available online xxx

Keywords:
Constraint satisfaction
Genetic programming
Economic problems

ABSTRACT

This paper presents an evolutionary algorithms based constrain-guided method (CGM) that is capable of handling both hard and soft constraints in optimization problems. While searching for constraint-satisfied solutions, the method differentiates candidate solutions by assigning them with different fitness values, enabling favorite solutions to be distinguished more likely and more effectively from unfavored ones.

We illustrate the use of CGM in solving two economic problems with optimization involved: (1) searching equilibriums for bargaining problems; (2) reducing the rate of failure in financial prediction problems. The efficacy of the proposed CGM is analyzed and compared with some other computational techniques, including a repair method and a penalty method for the problem (1), a linear classifier and three neural networks for the problem (2), respectively. Our studies here suggest that the evolutionary algorithms based CGM compares favorably against those computational approaches.

1. Introduction

Many economic problems, such as bargaining, financial investment, supply-chain management, vehicle routing and time-tabling can be treated as optimization problems with certain constraints associated. Optimization entails attempting to find the "best solution" among all possible solutions, where the "best" is always subject to various problem-specific conditions. For example, a delivery company would always like to minimize its overall costs by reducing the total travelling distance and the total delivery time, given a number of customer delivery jobs, a number of vehicles, and a number of driver. To find good solutions to this problem in reality, those constraints mentioned above have to be met. Optimization problems with constraint satisfaction are computationally demanding by nature. To find the best solutions to the problems within an acceptable time is often beyond the capacity of the current computer power. To fill in the gap, along with developing more powerful computers, research attempts to develop and employ faster algorithms to meet this demand.

Many economic problems involve both hard and soft constraints. Hard constraints describe feasibility of solutions [1,2]. Hard constraints must be satisfied. When several hard constraints exist, a feasible solution should satisfy all of them. *Soft constraints* describe levels of preferences, probability, cost, certainty and many other criteria [3]. A soft constraint can be violated. A soft constraint expresses a preference of one group of solutions (for example, having a lower investment risk) over another group of solutions (having a higher investment risk). Solutions can be partially ordered by soft constraints [4]. Some techniques for handling hard constraints are adapted to handle soft constraints [3–5]. The simultaneous existence of both hard and soft constraints complicates problem solving, because a constraint violation is potentially allowed. Thus such a problem becomes a combinatorial constrained optimization problem, rather than a simple constraint satisfaction problem [6].

Evolutionary algorithms are acknowledged as good solvers for economic problems [7–9]. Penalty methods [10–12,15] and repair methods [11,13–15] are among the most popular constraint handling techniques used with evolutionary algorithms. Penalty methods and repair methods were designed to satisfy hard constraints. However, constraints in many economic problems are complicated in that soft constraints can be sacrificed for hard constraints or for other soft constraints. Such features are captured by neither penalty methods nor repair methods. Thus, potential problems arise when it is difficult or inefficient to define a penalty function or to repair solutions to satisfy all hard constraints, while still to take soft constraints into consideration. These two methods cannot decide which constraint can be sacrificed for another. To our best knowledge, little research has

* Corresponding author. Tel.: +44 1206 87 2774; fax: +44 1206 87 2684.
E-mail addresses: n.jin@leeds.ac.uk (N. Jin), edward@essex.ac.uk (E. Tsang), Jin.Li@unilever.com (J. Li).

1568-4946/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2008.11.006

been done into simultaneously handling both soft and hard constraints while employing population-based metaheuristic optimization algorithms such as genetic programming (GP) and genetic algorithms (GAs).

This paper presents a constraint-guided method (CGM) that is capable of effectively handling two types of constraints aforementioned in solving optimization problems. This method is currently applied with population-based evolutionary algorithms. Its central idea is based on the fact that candidate solutions to an optimization problem with both hard and soft constraints can be categorized into three qualitatively different sets: the infeasible, the feasible and the preferable. Solutions in an infeasible set do not satisfy all hard solutions. In contrast, solutions in a feasible set do satisfy all hard constraints, as well as some soft constraints if not all. The solutions in the preferable set are usually considered as best candidate solutions. Within the feasible set, there are some preferable solutions that satisfy some soft constraints with higher priorities. An ideal search approach should be capable of not only identifying the feasible set among all possible solutions, but also distinguishing a preferable set from the feasible sets more easily. This is the motivation under the development of CGM. CGM enables evolutionary algorithms to be a successful optimization approach to find a preferable set within a huge solution space. We achieve this goal by using a carefully designed fitness function, which incorporates problem-specific knowledge about hard and soft constraints directly, without treating them separately.

We demonstrate the efficacy of CGM through testing it on two economic optimization problems: bargaining problems and financial prediction problems. We apply CGM with genetic programming to these two problems. For a classic bargaining problem, we compare the solutions provided by CGM, by a penalty method, by a repair method and by using no constraint handling technique respectively against the benchmark solutions: the game-theoretic equilibriums. For the financial prediction problem, we compare the performance of financial trading rules generated by using CGM against those generated by neural networks and a linear classifier [16]. Both case studies here suggest that the evolutionary algorithms based CGM compares favorably against those computational approaches.

This paper is organized as follows: Section 2 details a generic CGM. Section 3 applies CGM to bargaining problems. We describe our developing process towards the solution representation, GP set-up and a CGM-based fitness function. We compare the experimental results from using CGM against two well known methods of constraint handling techniques. Section 4 details the application of CGM to achieve a low rate of failure in financial prediction problems. We compare the performance of CGM with genetic programming against that of the linear classifier and against that of Time Delay Neural Networks, Recurrent Neural Networks and Probabilistic Neural Networks. Finally, Section 5 concludes.

2. Constraint-guided method

An optimization problem with constraint satisfaction is defined as (X, D, C, f). X is a finite set of variables, $x_i \in X$ and $i = 1, \ldots, n$. n is the number of solutions. D is a finite set of domains where $x_i \in X$ takes its value from the corresponding domain $d_{x_i} \in D$.

C is a finite set of solution sets. Each constraint i maps $c_i, c_i \in C$. A constraint may take the form of a set (of legal or illegal assignment combinations) or a function. Hard constraint(s) defines feasibility of solutions. $h_u (u = 1, \ldots, j)$ is a set of solutions which satisfies the hard constraint u, $h_u = c_u$. $H = h_1 \cap h_2 \cdots \cap h_j$. A feasible solution x must satisfy all hard constraints and is in the set H: $x \in H$. A soft constraint defines preference properties in solutions.

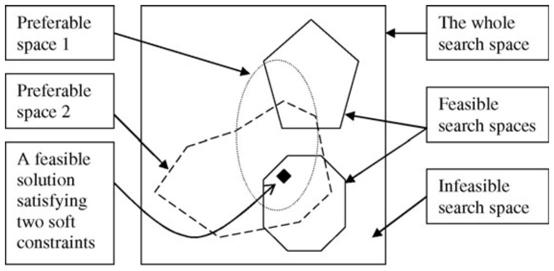

Fig. 1. An example of a search space which has two feasible, infeasible and two preferable spaces. Two soft constraints map the two preferable spaces. The black diamond is a feasible solution inside both preferable search spaces.

$s_v (v = j + 1, \ldots, l)$ is a set of solutions which satisfies the soft constraint v, $s_v = c_v$. Variables u, v, i and j are positive integers. A preferable solution does not need to satisfy all soft constraints. If a solution x satisfies all soft constraints, then $x \in S$ where $S = s_j \cap s_{j+1} \cdots \cap s_l$. Please note that a preferable solution set s_v is not necessary a subset of the feasible set H[1]. Fig. 1 illustrates an example of the relationship among feasible, infeasible and preferable solution sets in the search space.

The function f maps a solution x to an objective value. If T is the set of solution tuples of (X, D, C), then $f : T \rightarrow r$, r is a real value. The task is to find the solution tuple with the optimal f value with regard to the application-dependent optimization max $f(x)$.

CGM is a generic method, applicable to population-based metaheuristic optimization algorithms. We apply an evolutionary algorithm, in particular GP as the search tool to the following economic problems. As known, the genetic operators mutation and crossover are 'blind' to constraints [17], so the selection becomes the major way to guide search into feasible and preferable spaces. CGM utilizes the selection via adjusting fitness values according to a solution's satisfaction to constraints.

The pseudo-code of CGM is

```
Create initial population
Repeat
  Evaluate a candidate solution's problem-specific
      fitness f
  Adjust f on its satisfaction to constraints
  Select pairs of candidate solutions to reproduce
  Breed new candidate solution(s) through crossover
      and mutation
Until <terminating condition>
```

Here f is the function to be optimized. Selection methods (such as fitness proportional selection, rank based selection and tournament selection) and their implementation influence the selection pressures. It is beyond the scope of this paper but we recommend using selection methods which maintain relatively small selective pressures.

[1] An informal example helps understand the idea of hard and soft constraints. The objective is to have a drink. The hard constraint is that "I only take diet drinks"; the soft constraint is "I prefer Coke to Pepsi". Coke is a preferred choice but a non-diet Coke is unacceptable as it violates the hard constraint. In this simple case, (a) "a diet Coke" is the best choice which meet the objective and satisfies all constraints; (b) "a diet Pepsi" is the second order choice which satisfies the hard constraint but is not a soft constraint; (c) "no diet drink is available" or "no drink at all" is the worst as I can not achieve the goal of having a drink.

Definition 1. Fitness function of CGM:

$$R(x) = \begin{cases} f(x) + f' & \text{if } x \in H \wedge x \in S \\ f(x) + k & \text{if } x \in H \wedge x \notin S \wedge x \in s_{j+1} \\ f(x) + k' & \text{if } x \in H \wedge x \notin S \wedge x \in s_{j+2} \\ f(x) + k'' & \text{if } x \in H \wedge x \notin S \\ & \wedge x \in s_{j+1} \wedge x \in s_{j+2} \\ \dots & \\ h(x) & \text{if } x \notin H \end{cases} \qquad (1)$$

The fitness function of CGM is $R(x)$. The optimality of $R(x)$ implies that all constraints are satisfied. f', k, k' and k'' can be a function or a constant, where $f' \geq 0$ (f' is not strictly necessary but is included so that k, k', k'' and h do not have to return negative values to meet the above conditions). For a specific problem, $R(x)$ is accomplished by instantiating f', k, k', k'' and $h(x)$. The effective definitions of f', k, k', k'' and $h(x)$ can channel problem-dependent knowledge about constraints into the search via fitness functions.

The conditions in Eq. (1) separate solutions sets in order. Firstly, a solution x that satisfies all hard and soft constraints is better than any solution x' that violates at least one constraint, even when $f(x)$ equals $f(x')$. So in $R(x)$, we have $f(x) + k < f(x) + f'$, $f(x) + k' < f(x) + f'$, $f(x) + k'' < f(x) + f'$ and $h(x) < f(x) + f'$. Secondly, any feasible solution is better than any infeasible solution: $f(x) + f' > h(x)$, $f(x) + k > h(x)$, $f(x) + k' > h(x)$ and $f(x) + k'' > h(x)$. Thirdly, how to order the importance of soft constraints and combinations of soft constraints is problem-dependent. In general, for feasible solutions, the more soft constraints they satisfy, the higher their fitness: $f(x) + k < f(x) + k''$ and $f(x) + k' < f(x) + k''$. Some times, soft constraints may not be strictly ordered. For example, feasible solutions which satisfy only one soft constraint s_{j+1} or s_{j+2}, can have different fitness values: $f(x_1) + k$ or $f(x_2) + k'$ when $f(x_1) = f(x_2)$. Finally, infeasible solutions are given fitness values so that CGM does not prevent a search from considering infeasible spaces. This is because infeasible solutions may contain valuable genetic materials that are needed for finding globally optimal solutions. However, CGM discourages infeasible solutions to produce offsprings through selection. Thus, under the ordering mechanism described above, evolution is expected to guide search into more promising spaces where the fitness values are higher.

We summarize the major features of CGM:

(1) Firstly CGM can tackle both hard and soft constraints.
(2) Secondly, instead of giving penalty or repairing infeasible solutions, CGM differentially awards fitness values to encourage feasible and preferable solutions.
(3) Thirdly, instead of using information about solutions and search space (for example the distance of an infeasible solutions to a feasible region), CGM exploits problem-specific information about constraints and directly "translates" such information into the fitness function.

In the following sections, we will demonstrate how to apply CGM to two problems: (1) searching equilibriums of bargaining problems and (2) reducing the rate of failure in financial prediction problems.

3. Constraint-guided method for bargaining problems

In this section, we will apply the constraint-guided method, CGM to bargaining problems which contain both hard and soft constraints. We will compare the performance of CGM against two other ways of handling constraints which are also used together with genetic programming.

Bargaining situations are ubiquitous in social and economic activities, such as political parties' coalitions for elections, threats of nuclear war and reaching new international trade agreements. Bargaining is a process of achieving an agreement on how to divide a common interest between (among) players [18]. Clearly all parities involving into a bargaining attempt to maximize their profits via this process. Very often, bargaining parties are constrained by time, resources and initiative [19] and such constraints are always hard to deal with.

3.1. Bargaining problem and its constraints

A classic bargaining problem is modelled by Rubinstein [20]. It describes a bargaining scenario (game) wherein two player P_1 and P_2 make offers and counter-offers alternately until an offer or a counter-offer is accepted by the other. Both P_1 and P_2 try to maximize their payoffs. For convenience, we denote an offer x_i which is proposed by player i for himself and the rest of cake $x_j = 1 - x_i$ for the other. An offer takes the form of a percentage of the cake, which is a number between 0 and 1. A payoff deteriorates over time, which motivates players to make agreements as soon as possible. A share of cake is worth more in round t than in round $t+1$, than in round $t+2$, etc. Player i's *discount factor* $\delta_i \in (0, 1)$ is his bargaining cost per time interval, measuring the cost of time. $\delta_i = e^{-r_i}$ where r_i is the player i's *discount rate*. If an agreement is reached at time t, the payoff p_i gained by player i whose share is x_i from this agreement is, $p_i = x_i \delta_i^t$. In the game-theoretic equilibrium, $P1$ obtains:

$$x_1^* = \frac{1 - \delta_2}{1 - \delta_1 \delta_2} \qquad (2)$$

and $P2$ obtains the rest of cake: $1 - x_1^*$. This solution is called the *Perfect Equilibrium Partition* (PEP). Game-theoretic analysis and proof can be found in [20,18]. This bargaining problem is simple yet is precise enough to allow for rigorous examination of CGM.

As both players try to optimize their payoffs while are constrained by their time cost δ_i, this problem can be classified as an optimization problem with constraints. Obviously this bargaining problem has a hard constraint *C1*:

C1. Any share x_i of cake should not be larger than the size of cake: $x_i \in (0, 1]$. Any share that does not satisfy this constraint is infeasible. This constraint must be satisfied.

From the definition of the discount factor, we derive two soft constraints:

C2. Everything else being equal, the higher discount factor a player i has, the larger share x_i that player i obtains.
C3. Everything else being equal, the higher discount factor the other player j has, the smaller share x_i player i gets.

In this problem, $C2$ and $C3$ are equally important.

3.2. Genetic programming set-up

We design an evolutionary system to tackle this bargaining problem, trying to find good bargaining strategies. In this system, each player has a set (or population) of candidate solutions. Each player has his own solution set because P_1 may have a first-move advantage [18] and therefore they may apply different strategies.

The evolutionary system works as follows: candidate solutions in a population independently undergo selection based on their performance (fitness). Better performed candidate solutions obtain higher probability to be taken as "raw materials" which will be genetically modified in order to breed new solutions of the forthcoming generation. Newly created candidate solutions will be assessed against the candidate solutions of another population that has gone through a similar evolutionary process. Note that the same genetic operators (selection, crossover and mutation) are employed for both populations.

3.2.1. Solution representation

As the game-theoretic solution PEP is represented by a function and we plan to evolve solutions which include variables δ_1 and δ_2, we use GP for its convenience of coping with function-based representations [9].

In this bargaining scenario, time is an important element. The cost of time is measured by discount factors. When time elapses, players decrease their shares of cake while offering or counter-offering, in order to strike a deal soon. We construct a bidding function in which player i bids b_i at time t:

$$b_i = g_i \times (1 - r_i)^t \tag{3}$$

where g_i is a candidate solution in the form of a genetic program. g_i is constructed with the function set $\{+, -, \times \text{ and } \div \text{ (protected)}\}$ and the terminal set $\{1, -1, \delta_i, \delta_j\}$. Fig. 2 illustrates an example genetic program representing a candidate solution $1/\delta_1 + \delta_2$. In Eq. (3), $(1 - r_i)^t$ is the basic formula of using discount rate r_i to describe the "time value of money". The use of $(1 - r_i)^t$ guarantees that b_i decreases over time.

A *bargaining strategy* determines what action (acceptance or making a counter-offer) a player takes at time t. The following equation describes how player i accepts or rejects a partition $(1 - x_j, x_j)$ at time t:

$$s(g_i) = \begin{cases} \text{accept}: x_i = 1 - x_j & \text{if } (1 - x_j)\delta_i^t \geq b_i(t+1)\delta_i^{t+1} \\ \text{counter-offer at}(t+1): x_i = b_i(t+1) & \text{otherwise.} \end{cases} \tag{4}$$

When player i (he) with strategy $s(g_i)$ receives an offer $(1 - x_j)$ from player j (she) who asks x_j for herself, i compares the payoff $(1 - x_j)\delta_i^t$ from this offer versus the payoff that he will get if his counter-offer $b_i(t+1)$ is accepted at $t+1$. If his payoff from accepting now, $(1 - x_j)\delta_i^t$ is not smaller than his payoff from counter-offering $b_i(t+1)\delta_i^{t+1}$, he accepts this offer now.

3.2.2. Parameters of GP

Table 1 summarizes the operators and parameters, and their values of this GP system. We use "grow" initialization method, three-member tournament selection, sub-tree crossover and sub-tree mutation [9]. The methods and values of genetic operators of any evolutionary algorithm can affect the performance of the algorithm in a significant way [21]. As many researchers do, we choose parameters and operators through experimentation. We try the values of GP operators suggested by [9,22], and test them on the bargaining problem.

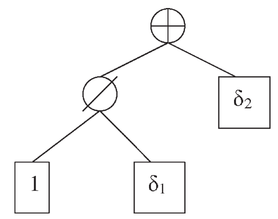

Fig. 2. A genetic program $1/\delta_1 + \delta_2$.

Table 1
Summary of the genetic programming parameters and operators.

GP parameters	Values
Terminal set	$\{\delta_1, \delta_2, 1, -1\}$
Functional set	$\{+, -, \times, \div\}$ ($\div$ is Protected)
Population size	100
Number of generations	300
Initial max depth	5
Maximum nodes of a GP program	50
Initialization method	Grow
Selection method	Three-member tournament
Crossover method	[9]
Crossover rate	(0, 0.1)
Mutation method	Sub-tree mutation
Mutation rate	(0.01, 0.3)

A range of crossover and mutation rates have been tested to reduce their bias on experimental results. Our system is relatively stable with a crossover rate in the range 0–0.1 and a mutation rate in the range 0.01–0.3.

We plan to test 25 pairs of game settings (game parameters) in total. Each pair of game setting is in the format of (δ_1, δ_2), where $\delta_1, \delta_2 \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The combinations of selected δ_1 and δ_2 are evenly distributed over the space $\delta_1 \times \delta_2$. In an experimental run, the fitness of candidate solutions in two populations tends to be stabilized before the 200th generation. To ensure stabilization and not to overuse computational resources, we terminate runs at the 300th generation. Each population has 100 genetic programs. 100 runs were conducted for every pair (δ_1, δ_2), a sufficient number for statistical purpose. We denote x_1 as player P_1's share from the agreement made by the two best-of-generation (highest fitness) genetic programs g_1 and g_2, g_1 from player P_1's population and g_2 from player P_2's population, at the 300th generation. $\bar{x}_1$ is the average of 100 x_1 from 100 runs for a (δ_1, δ_2), each run starting with different random sequences. The results $\bar{x}_2$ are not reported here, because they are merely the complement of $\bar{x}_1$.

3.3. CGM fitness function

Success of evolutionary algorithms relies on appropriate fitness measures. We will show how to encode these three constraints into the CGM fitness function by instantiating f', k, k', k'' and $h(x)$ in Eq. (1).

3.3.1. Sensibility measure and evaluation of attribution

We start with handling the soft constraints. Consider a Sensibility Measure (SM) that measures *whether* a genetic program g_i satisfies the soft constraints $C2$ and/or $C3$. p and q are real numbers, $p, q \in (0, 1)$. Let $g_i(p, q)$ be an instantiation of g_i with δ_i being substituted by p, and δ_j being substituted by $q \cdot \alpha \in (0, 1)$ is an arbitrary real number.

Definition 2. Sensibility measure of a genetic program g_i

$$SM_i(\delta_i, \delta_j, \alpha) = \begin{cases} -\dfrac{g_i(\delta_i, \delta_j) - g_i(\delta_i \times (1+\alpha), \delta_j)}{g_i(\delta_i, \delta_j)} & \text{if } \delta_i \times (1+\alpha) < 1 \\ \dfrac{g_i(\delta_i, \delta_j) - g_i(\delta_i \times (1-\alpha), \delta_j)}{g_i(\delta_i, \delta_j)} & \text{otherwise.} \end{cases} \tag{5}$$

$$SM_j(\delta_i,\delta_j,\alpha)=\begin{cases}\dfrac{g_i(\delta_i,\delta_j)-g_i(\delta_i,\delta_j\times(1+\alpha))}{g_i(\delta_i,\delta_j)} & \text{if } \delta_j\times(1+\alpha)<1\\ -\dfrac{g_i(\delta_i,\delta_j)-g_i(\delta_i,\delta_j\times(1-\alpha))}{g_i(\delta_i,\delta_j)} & \text{otherwise.}\end{cases} \tag{6}$$

SM describes soft constraints *C2* and *C3* in a mathematic manner. According to *C2*, when player *i*'s discount factor increases from δ_i to $(\delta_i\times(1+\alpha))\in(0,1)$, the genetic program g_i should positively correlate. If α is too large to make $(\delta_i\times(1+\alpha))\in(0,1)$, we decrease δ_i to $\delta_i\times(1-\alpha)$ and then $g_i(\delta_i,\delta_j)$ should be larger than $g_i(\delta_i\times(1-\alpha),\delta_j)$ as shown in Eq. (5). The soft constraint *C3* implies that the genetic program g_i should negatively correlate to player *j*'s discount factor δ_j, shown in Eq. (6). That $SM_i(\delta_i,\delta_j,\alpha)$ or $SM_j(\delta_i,\delta_j,\alpha)$ returns a positive value is desirable, because this means that this genetic program g_i satisfies the constraints *C2* or *C3* respectively. Next will show how to treat genetic programs that satisfy a soft constraint.

Definition 3. Evaluation of attribution (*ATT*) defines the value to be rewarded to a genetic program g_i whose sensibility measures are SM_i and SM_j. An *ATT quantifies* g_i's satisfaction to a soft constraint *C2* or *C3*.

$$\text{ATT}(i)=\begin{cases}1 & \text{if } SM_i(\delta_i,\delta_j,\alpha)\ge 0\\ -e^{(1/(SM_i(\delta_i,\delta_j,\alpha)))} & \text{otherwise.}\end{cases} \tag{7}$$

$$\text{ATT}(j)=\begin{cases}1 & \text{if } SM_j(\delta_i,\delta_j,\alpha)\ge 0\\ -e^{(1/(SM_j(\delta_i,\delta_j,\alpha)))} & \text{otherwise.}\end{cases} \tag{8}$$

When SM_i or SM_j returns a positive value, meaning that g_i satisfies a soft constraint, ATT(i) or ATT(j) is given the highest value 1. When SM_i or SM_j returns a negative value, ATT gives a value less than 0. The exact value given depends on how close SM_i or SM_j is to 0. The closer SM_i or SM_j to 0, the higher value is rewarded by ATT(i) or ATT(j). Here we adopt the function $-e^{1/SM}$ to control this rewarding algorithm. For a negative value of *SM*, ATT is always negative in the range $(-1,0)$. For $SM\to 0^-$, ATT quickly approaches 0. For $SM<-1$ and $SM\to-\infty$, ATT quickly approaches -1. The function $-e^{1/SM}$ is certainly not the only way to implement this idea. It is chosen for its simple structure.

3.3.2. CGM-embedded fitness function

A genetic program g_i that satisfies the hard constraint *C1* pairwisely plays bargaining games with those genetic programs g_j that satisfy *C1* in player *j*'s population. A subset of *j*'s population consisting of all genetic programs that satisfy *C*1 is named as P'_j.

Definition 4. Game fitness $GF(s(g_i))$ of $s(g_i)$ is the average payoff that g_i gains from bargaining with every genetic program in P'_j. The size of P'_j is *n*, where the non-negative integer *n* is an experimental parameter.

$$GF(s(g_i))=\frac{\sum_{j\in P'_j} p_{s(g_i)\to s(g_j)}}{n} \tag{9}$$

where $p_{s(g_i)\to s(g_j)}$ is the payoff obtained by $s(g_i)$ from the agreement with $s(g_j)$. $GF(s(g_i))$ indicates the bargaining competence of g_i in its population.

Definition 5. Fitness function using CGM, $F(g_i)$ determines the fitness of g_i whose corresponding strategy is $s(g_i)$, whose sensibility measures are SM_i and SM_j, and whose evaluation of attribution are ATT(i) and ATT(j).

$$F(g_i)=\begin{cases}GF(s(g_i))+3 & \text{if } g_i\in(0,1]\cap SM_i>0\cap SM_j>0\\ GF(s(g_i))+\text{ATT}(i)+\text{ATT}(j) & \text{if } g_i\in(0,1](SM_i<=0\cup SM_j<=0)\\ \text{ATT}(i)+\text{ATT}(j)-e^{(-1/|g_i|)} & \text{otherwise.}\end{cases} \tag{10}$$

$F(g_i)$ is applied to all genetic programs (candidate solutions) in both populations. Having mentioned earlier in $R(x)$, we now set $GF(s(g_i))+3>GF(s(g_i))+\text{ATT}(i)+\text{ATT}(j)>\text{ATT}(i)+\text{ATT}(j)-e(-1/|g_i|)$. As shown in Eq. (10), we add an extra constant 3 into the fitness of genetic programs that satisfy all three constraints and whose game fitness is $GF(s(g_i))$. This ensures that they dominate the rest which fail to meet all constraints and then encourages desired genetic programs to propagate. Genetic programs that satisfy the hard constraint *C*1, but not *C*2 and/or *C*3, are still eligible for playing bargaining games. Their fitness are the game fitness $GF(s(g_i))$ plus $(\text{ATT}(i)+\text{ATT}(j))$. $(\text{ATT}(i)+\text{ATT}(j))$ reflects how close they meet the two soft constraints. For those genetic programs that violate the hard constraint *C*1, it is meaningless to use them to play bargaining games at all. Therefore, they are allocated a fitness solely based on their *SM* and ATT. Their fitness is definitely lower than any genetic programs that satisfies at least, the constraint *C*1.

3.4. Comparative study with other constraint handling techniques

In this section, we compare the empirical data generated by CGM against those by two well-applied constraint handling techniques, penalty methods and repair methods, and against those by the no constraint handling setting, to justify the performance and applicability of CGM.

Penalty methods penalize infeasible solutions in evolutionary systems. They either give a penalty to an infeasible solution or define a cost for repairing such a solution for making it feasible. In general, a penalty method transforms a constrained optimization problem: *max* $f(x)$ subject to $w(x)\le C$, to an unconstrained problem *max* $Y(x)=f(x)-\text{Penalty}(x)$ by defining the penalty function Penalty(x). Although conceptually the penalty method is simple, the freedom in implementing Penalty(x) is vast. A slight change in the implementation could have significant impacts on the effectiveness and efficiency of the algorithm. Given the same value of $f(x)-\text{Penalty}(x)$, $Y(x)$ can not differ x_1 which returns a high value of $f(x_1)$ and a high value of Penalty(x_1) from x_2 which returns a low value of $f(x_2)$ and a low value of Penalty(x_2). Thus x_1 and x_2 are treated equally by the penalty method. It is especially critical if feasible solutions x_1 and x_2 should be differentiated according to soft constraints. Another important constraint handling technique used with evolutionary algorithms is the repair method. Repair methods have successfully been used for solving combinatorial optimization problems [13,14,11,15]. Repair methods use repair procedures to modify infeasible solutions to feasible ones. The repair procedures are not reflected in the fitness functions. Appropriate repair operators may not be derived directly from problem-specific knowledge. It requires extensive knowledge about solutions and about search spaces, however such knowledge are often unavailable directly from the problem.

3.4.1. Measurements

Before presenting the experimental results, we define two variations to measure the difference between the benchmark game-theoretic solution x_1^* and experimentally observed $\bar{x}_1$.

Definition 6. Absolute variation (av) is an unsigned difference between PEP x_1^* and experimentally observed $\bar{x}_1$ for a given (δ_1, δ_2).

$$av = |x_1^* - \bar{x}_1| \tag{11}$$

Definition 7. Relative variation (rv) is an unsigned relative increment of $\bar{x}_1$ over x_1^*.

$$rv = \frac{|x_1^* - \bar{x}_1|}{x_1^*} = \frac{av}{x_1^*} \tag{12}$$

The reasons that we adopt both variation measures are (i) for an absolute variation, $|x_1^* - \bar{x}_1| = |x_2^* - \bar{x}_2|$ is true. For a relative variation, $(|x_1^* - \bar{x}_1|)/(x_1^*) = (|x_2^* - \bar{x}_2|)/(x_2^*)$ does not hold. Therefore, a relative variation very likely produces two different results for one bargaining game, which is not ideal for the measurement; (ii) absolute variation alone is not enough to express the variation on the base of x_1^*. For example, two sets of results $(x_1^* = 0.05, \bar{x}_1 = 0.06)$ and $(x_1^* = 0.95, \bar{x}_1 = 0.96)$, both have the same absolute variation $av = 0.01$, but the former set has a 0.17 increment based on x_1^* and the latter set has only a 0.01 increment based on x_1^*. In this sense, the relative variation is more informative and indicative. As a result, we use both variation measures.

3.4.2. Experimental results from using CGM

The absolute variations of experimental results by CGM are given in Table 2. The relative variations by CGM are in Table 3.

Our investigation is motivated by three key questions. These questions are: (1) Does CGM outperform a repair method? (2) Does CGM outperform a penalty method? (3) Does CGM produce better results than having no constraint handling at all? To answer these questions, we carried out three extra series of experiments. Each series of experiments is intended to obtain empirical data to be compared against the results of using CGM.

3.4.3. Compare against the results from a repair method

In an attempt to address the first question, we have tried to use a repair method to make an infeasible genetic program feasible. We have to change one or more nodes of a genetic tree and then do testing, until it becomes feasible. This procedure is as time-consuming as to create new feasible solutions. From the experimental results, the repair procedure itself does not contribute to the improvement of quality of feasible solutions. Meanwhile, CGM saves a great amount of computational time without doing this repairing procedure. Obviously CGM beats the repair method on its efficiency.

3.4.4. Compare against the results from a penalty method

The second series of experiments is intended to answer the second question. For this series of control experiments, we design a penalty method. The fitness function of using a penalty method, $F(g_i)'$ is defined below. Infeasible genetic programs are penalized. To be fairly comparable to the CGM fitness function $F(g_i)$, the penalty function for infeasible genetic programs is $\mathrm{ATT}(i) + \mathrm{ATT}(j) - e^{(-1/|g_i|)}$, which is the same function for infeasible genetic programs in $F(g_i)$.

Table 2
Absolute variations by CGM.

δ_2	CGM ($\lvert x_1^* - \bar{x}_1\rvert$)				
	$\delta_1 = 0.1$	0.3	0.5	0.7	0.9
0.1	0.0135	0.0715	0.0520	0.0308	0.0098
0.3	0.2775	0.2308	0.1741	0.1075	0.0371
0.5	0.1606	0.0963	0.0087	0.0841	0.1643
0.7	0.0002	0.0258	0.1016	0.2287	0.1266
0.9	0.0821	0.0280	0.0780	0.0001	0.0122

Table 3
Relative variations by CGM.

δ_2	CGM: ($\lvert x_1^* - \bar{x}_1\rvert / x_1^*$)				
	$\delta_1 = 0.1$	0.3	0.5	0.7	0.9
0.1	0.0149	0.0771	0.0549	0.0318	0.0100
0.3	0.3845	0.3000	0.2114	0.1213	0.0387
0.5	0.3051	0.1638	0.0131	0.1094	0.1807
0.7	0.0007	0.0679	0.2201	0.3889	0.1562
0.9	0.7472	0.2045	0.4292	0.0004	0.0232

Table 4
Absolute variations by the penalty method.

δ_2	Penalty method ($\lvert x_1^* - \bar{x}_1\rvert$)				
	$\delta_1 = 0.1$	0.3	0.5	0.7	0.9
0.1	0.0909	0.0722	0.0526	0.0323	0.0110
0.3	0.2784	0.2308	0.1765	0.1139	0.0411
0.5	0.1625	0.0986	0.0058	0.0813	0.1555
0.7	0.0066	0.0238	0.1025	0.2356	0.0855
0.9	0.0765	0.0184	0.0458	0.0021	0.0252

Table 5
Relative variation by penalty method.

δ_2	Penalty method ($\lvert x_1^* - \bar{x}_1\rvert / x_1^*$)				
	$\delta_1 = 0.1$	0.3	0.5	0.7	0.9
0.1	0.1000	0.0778	0.0556	0.0333	0.0111
0.3	0.3857	0.3000	0.2143	0.1286	0.0429
0.5	0.3088	0.1676	0.0087	0.1057	0.1710
0.7	0.0203	0.0627	0.2221	0.4005	0.1055
0.9	0.6965	0.1340	0.2516	0.0079	0.0479

Definition 8. Fitness function incorporated with a penalty method

$$F(g_i)' = \begin{cases} GF(s(g_i)) + \mathrm{ATT}(i) + \mathrm{ATT}(j) & \text{if } g_i \in (0, 1] \\ \mathrm{ATT}(i) + \mathrm{ATT}(j) - e^{(-1/|g_i|)} & \text{otherwise.} \end{cases} \tag{13}$$

We use the same sequence of random seeds, the same genetic operators, the same sets of game settings and the same number of runs as those used in the series of experiments for CGM (as in Section 3.2 and 3.3). Absolute variations of the experimental results from the penalty method are shown in Table 4 and relative variations are in Table 5.

Table 6 compares the average absolute variations by CGM versus those by the penalty method, categorized into three ranges: $av < 0.15$, $av < 0.10$ and $av < 0.05$. For example, in Table 6, the average av of $\bar{x}_1$ s from using CGM is 0.0509 whilst that of using the penalty method is 0.0519 in the same range $0 \leq av < 0.15$. It shows, in average, for the game settings whose $0 \leq av < 0.15$, CGM's solutions have smaller absolute variations to the game-theoretic solutions, therefore it provides better solutions than the penalty method. m and m' in Table 6 are the numbers of s in a specified range. Table 7 lists the average relative variations by CGM and those by the penalty method, categorized into three ranges: $rv < 0.15$, $rv < 0.10$ and $rv < 0.05$. n and n' are the numbers of relative variations in a specified range.

In Table 6 and 7, a clear pattern is observed: with no exception, CGM has smaller values with respect to both average absolute variations and average relative variations in all three ranges. In other words, $\bar{x}_1$ found by CGM approach noticeably closer to PEP x_1^* than those found by the penalty method. This pattern provides the evidence that CGM yields smaller variations to game-theoretic solutions than the penalty method. Therefore, as the answer to the second question, CGM outperforms the penalty method.

Table 6
Average of absolute variations (av in Eq. (11)) of three ranges.

Method	Average of absolute variations $\frac{\sum av}{m^{(\prime)}}$ when $0 \le av < 0.15$	Average of absolute variations $\frac{\sum av}{m^{(\prime)}}$ when $0 \le av < 0.10$	Average of absolute variations $\frac{\sum av}{m^{(\prime)}}$ when $0 \le av < 0.05$
CGM	0.0509	0.0394	0.0166
m	19	16	10
Penalty method	0.0519	0.0453	0.0212
m'	19	16	10

m and m' are the number of av in the specified range, amongst the 25 game settings.

Table 7
Average of relative variations (rv in Eq. (12)) of three ranges.

Method	Average of relative variations $\frac{\sum rv}{n^{(\prime)}}$ when $0 \le rv < 0.15$	Average of relative variations $\frac{\sum rv}{n^{(\prime)}}$ when $0 \le rv < 0.10$	Average of relative variations $\frac{\sum rv}{n^{(\prime)}}$ when $0 \le rv < 0.05$
CGM	0.0433	0.0302	0.0166
n	13	11	8
Penalty method	0.0628	0.0426	0.0246
n'	15	11	7

n and n' are the numbers of rv in the specified range, amongst the 25 game settings.

Table 8
The number of game settings which CGM outperforms the penalty method, amongst the 25 game settings.

Absolute variations	CGM better	Tie	Penalty method better
Number of game settings	16	1	8
Relative variations	CGM better	Tie	Penalty method better
Number of game settings	16	1	8

Moreover, Table 8 counts the numbers of game settings in three cases: CGM is better; the penalty method is better; or both are equally good. We find that on both absolute and relative variations, CGM works better on 16 out of 25 game settings (δ_1, δ_2) while the penalty method only performs better on 8 game settings. In other words, CGM outperforms on the vast majority of game settings.

3.4.5. Compare against the results of imposing no constraint

To answer the third question, we further do a series of experiments whose fitness functions control none of the three mentioned constraints *C1*, *C2* and *C3*. In this series of experiments, a genetic program's fitness function is simply its game fitness $GF(g_i)$. All genetic programs in both populations play the bargaining game. Under this non-constrained fitness function $GF(g_i)$, experimental results show that for a given game setting (δ_1, δ_2), the majority of the 100 runs end up with x_1 that are within the cake size, i.e. satisfying the hard constraint. However, their average value $\bar{x}_1$ can be out of the range 0 to 1. When a few runs finish with x_1 which are exceptionally large or negative, $\bar{x}_1$ deviates far away from the PEP x_1^*. For example, in one 100-run of experiments on game setting $(0.9, 0.1)$, 83 out of 100 x_1 satisfy $0 < x_1 \le 1$ but the rest 17 x_1 get $x_1 > 1$. The largest x_1 even equals 8×10^{14}. This is probably because the search had no chance to enter the space (0, 1]. As a result, the average of this 100-run, $\bar{x}_1$ thus does not even meet the hard constraint $x_i \in (0, 1]$. From the above experimental results, we conclude that the performance of imposing no constraint into the fitness function is hardly comparable to that of CGM.

3.5. CGM to other bargaining problems

We have applied CGM to two other types of bargaining problems, namely incomplete information bargaining problems [23] and an outside-option bargaining problem [24]. These two problems are also constrained by hard and soft constraints, which make such problems game-theoretically challenging. CGM used with genetic programming also exhibits its superiority with respect to the variations to the game-theoretic solutions.

4. Constraint-guided method for financial prediction

Financial prediction is also considered as an optimization problem involved with constraints. Investors generally would like to make maximal profit with minimal risk under some financial conditions. In this section, we demonstrate the effectiveness of the CGM approach for predicting financial market trends. We show how the CGM approach could be applied to find some preferable predictive rules which tend to have the low rate of failure (i.e., high positive predictivity). The rules with low rate of failure are more likely linked with lower risks.

4.1. Genetic programming for financial forecasting

Tsang et al. have developed *Evolutionary Dynamic Data Investment Evaluator* [25,26]. It is an interactive financial forecasting tool. This tool is implemented by genetic programming. A set of trading rules is represented as a *genetic decision tree* (GDT).

The basic elements of GDTs are *rules* and *forecast values*. A single *rule* is consisted of one useful indicator for prediction, one relational operator such as "greater than", or "less than", etc, and a threshold (real value). Such a single rule interacts with other rules in one GDT through logic operators such as "Or", "And", "Not", and "If-Then-Else". Forecast values are either a *positive position* (i.e. $r\%$ return within n days can be achievable) or *negative position* (i.e. $r\%$ return within n days can not be achievable).

We follow our earlier work by adopting the indictors that were derived from finance literature [27–30]. They include three types of technical analysis rules (i.e. moving average rules, filter rules, trade range break rules) as follows: (1) MV_{12} = Today's price - the average price of the previous 12 trading days; (2) MV_{50} = Today's price – the average price of the previous 50 trading days; (3) $Filter_5$ = Today's price - the minimum price of the previous 5 trading days; (4) $Filter_{63}$ = Today's price – the minimum price of the previous 63 trading days; (5) TRB_5 = Today's price – the maximum price of the previous 5 trading days (based on the Trading Range Breakout rule [29]); (6) TRB_{50} = Today's price – the maximum price of the previous 50 trading days. Below shows an example of a simple GDT built by using the above grammar. It is a simplistic GDT concerning the prediction of 2.2% return within one month. A useful GDT in the real world could be a lot more sophisticated than this.

```
(IF (PMV_50 < -18.45)
  THEN Positive
 ELSE (IF ((TRB_5 > -19.48) AND (Filter_63 < 36.24))
        THEN Negative
        ELSE Positive))
```

This example GDT suggests that if today's price is more than 18.45 below the average price of last 50 days, then today is very likely a positive position (i.e., one could achieve a return of 2.2% or more within the next one month); otherwise one should make decisions depending on the values of TRB_5 and $Filter_{63}$. If today's price is no more than 19.48 above the maximum price of the previous 5 trading days or today's price is more than 36.24 above the minimum price in the last 63 days, then it is also an alternative good opportunity to make a buy decision. The generated GDTs are used to predict whether or not the price will rise a required $r\%$ (for example, $r = 2.2$) or more within a user-specified period n (for example, 21 days).

4.1.1. Parameters for running GP

A GP system for evolving GDTs uses standard GP operators. In this section, all the experiments were carried out on a Pentium PC (200 MHz) using a population size of 1200. The termination condition was 50 generations or maximum of 2 h running, whichever reached. For each independent run, when it terminates we chose a best-so-far GDT in terms of fitness value over training data. Then, we apply it to test data for prediction. All results reported in this section are performances over test data. Major parameters are displayed in Table 9.

4.1.2. A linear measure of investment performance

Our earlier work [26] and [25] has shown that forecasting accuracy could be improved by using genetic programming technique. However, in financial prediction domain, prediction accuracy is not a sole concerned issue. The performance of any decision is usually justified in terms of user's preferences. Such preferences are often constrained by key issues such as prediction accuracy, time, budget, cost and risk. For example, apart from prediction accuracy (or the *Rate of Correctness* (*RC*)), one may be more concerned with grasping every possible opportunity by reducing the *Rate of Missing Chances* (*RMC*), or with making each forecasting more reliable by reducing the *Rate of Failure* (*RF*). In practice, one of the most concerned factors is to reduce risk of forecasting through achieving a low rate of failure. A low rate of failure means any positive position generated by the system turns out to be reliable with a higher possibility. Mistaking an actual negative position for a positive position (*a false positive position*) is much more costly than the opposite of mistaking an actual positive position for a negative position (*a false negative position*), because the latter error only means missing a chance, no loss at all in respect of investment. Another reason for us to focus on achieving a low rate of failure is that higher prediction accuracy is not available or even impossible in some cases of financial forecasting. Therefore, it would be of great value to reduce *RF* without affecting an overall prediction accuracy.

In financial investment, the key performance measures are *RC*, *RMC* and *RF*. Thus we proposed a linear measure as follows:

$$f_0 = w_{RC} \times RC - w_{RMC} \times RMC - w_{RF} \times RF \qquad (14)$$

where $0 \leq w_{RC}, w_{RMC}$, and $w_{RF} \leq 1$. The task is to optimize f_0.

It involves three performance values, i.e. *RC*, *RMC* and *RF*, each of which is assigned a different weight: w_{RC}, w_{RMC} or w_{RF}. These three weights quantify the relative importance among *RC*, *RMC* and *RF* to a particular user. Obviously, the goodness of a GDT is assessed by a synthetical value, which is the weighted sum of its three performance values. By appropriately adjusting sizes of three weights, we are able to place more emphasis on one performance than on the others. f_0 returns the raw fitness of a GDT without adding any constraint in.

In order to achieve a low *RF*, for example, one possible way is:

(1) to assign w_{RC} a higher value (e.g. $w_{RC} = 1$) to highly award the GDT that has a good *RC* performance;
(2) to set w_{RF} a higher value to heavily penalize the GDT that has a poor *RF* performance;
(3) to assign w_{RMC} a smaller value or even zero, to slightly penalize GDT that has a poor performance of *RMC* or even not to penalize it at all.

Thus, we conjectured that f_0 with appropriate weights should work and lead the GP system to search for GDTs with lower RFs. However, our substantial trials showed that it did not work as we expected. To a certain extent, f_0 does allow us to reduce *RF*. But, it has two drawbacks: (a) the sizes of three weights are too sensitive to choose; (b) results are unstable.

From experiments, it is clear that the measure f_0 is not able to guide the GP system effectively to search for good solutions in terms of the performance of *RF*. We need to provide a mechanism in the fitness function that is able to lead GP to effectively and efficiently seek for better solutions. In the following subsection, we shall demonstrate that a CGM constraint handling is capable of taking this role.

4.2. Embedding CGM into fitness function

To resolve the above undesirable predicaments of the linear fitness function f_0, we introduce a constraint into f_0. This c onstraint is the expected range of ratios of the number of positive positions predicted to the total number of training data cases. We denote the constraint with *R*, which consists of two elements

Table 9
Summary of the genetic programming parameters and operators.

GP parameters	Values
Input terminals	MV_{12}, MV_{50}, $Filter_5$, $Filter_{63}$, TRB_5 and TRB_{50}; real values as thresholds
Prediction terminals	$\{0, 1\}$: 1 represents "Positive"; 0 represents "Negative".
Non-terminals	If-then-else, And, Or, Not, $>$, $\geq$, $<$, $\leq$, =.
Crossover rate	0.9
Mutation rate	0.01
Population size	1200
Maximum no. of generations	50
Termination criterion	2-h running time.
Selection method	Tournament selection, size = 4.
Max depth of individual programs	17
Max depth of initial individual programs	4
Run times (h)	Max 2-h
Hardware and operating system	Pentium PC 200 MHz Windows 95 with 64M RAM
Software	Borland C ++ (version 4.5).

represented by percentage. The range of the constraint R is determined by $C_{\min}$, the minimum and $C_{\max}$, the maximum.

$R = [C_{\min}, C_{\max}]$

where $C_{\min} = (P_{\min}/N_{tr}) \times 100\%$, $C_{\max} = (P_{\max}/N_{tr}) \times 100\%$, and $0 \le C_{\min} \le C_{\max} \le 100\%$;

N_{tr} is the total number of training data cases; $P_{\min}$ is the minimum of expected number of positive positions predicted; and $P_{\max}$ is the maximum of expected number of positive positions predicted.

Since ranges of R s chosen are mutually exclusive for all our experiments, we introduce a comparison notion for constraints, R s. A constraint R is said to be smaller than another $R' = [C'_{\min}, C'_{\max}]$, $R < R'$ if and only if $0 < C_{\min} < C_{\max} \le C'_{\min} < C'_{\max}$.

We now integrate the constraint into the linear performance measure f_0. We call the constraint-guided fitness function f_{CGM}:

$$f_{CGM} = w'_{RC} \times RC - w_{RMC} \times RMC - w_{RF} \times RF \qquad (15)$$

where $0 \le w_{RMC}$, and $w_{RF} \le 1$. The task is to optimize f_{CGM}.

The fitness value from f_{CGM} is still a composite value that takes all three performances into account with corresponding weights. However, w'_{RC}, the weight for RC, does not only take a constant like w_{RC} in f_0, more importantly, it could also possibly take the value of zero on conditions of the size of C_+ and the constraint R. w'_{RC} is defined by

$$w'_{RC} = \begin{cases} w_{RC} & \text{if } C_+ \in R_{[C_{\min}, C_{\max}]} \\ 0 & \text{otherwise} \end{cases} \qquad (16)$$

where $0 \le w_{RC} \le 1$. Note that C_+ is the percentage of the number of positive positions predicted by a GDT based on training data to the total number of training data cases, given by $C_+ = N_+/N_{tr} \times 100\%$, where N_+ is the number of positive positions predicted by the GDT. For simplicity, a positive position predicted is also called a *signal* in the following context. The size of R is specified with setting up two elements: $C_{\min}$ and $C_{\max}$ by the user beforehand. The hard constraint is that C_+, the percentage of the number of positive positions predicted by a GDT must be within a specified range: $C_+ \in R_{[C_{\min}, C_{\max}]}$. Users can further breakdown this range into subranges to which they could prefer differently. Such differences can be again integrated into the CGM fitness function.

With the constraint R embedded into the fitness function, only GDTs which can satisfy the constraint, are awarded to a great extent. In contrast, those GDTs which cannot satisfy the constraint, are heavily disfavored by being assigned negative fitness values, and consequently would be extinct during evolving. Notably, f_{CGM} has a more generalized form compared to f_0. f_0 can be treated as a specific case of f_{CGM}, where R is taken as $[0\%, 100\%]$, which makes w'_{RC} equal to w_{RC} as any GDT satisfies the constraint. We denote the GP system using f_{CGM} as CGM-GP.

4.3. Comparative study with NNs and a linear classifier

In order to evaluate CGM-GP against existing other approaches, we have reviewed literature and fortunately found a similar research study that aims to achieve low false alarm while addressing an identical prediction problem. Saad and his colleagues deliberately developed three Neural Networks (NNs), i.e., Time Delay Neural Networks (TDNN), Recurrent Neural Networks (RNN) and Probabilistic Neural Networks (PNN) and a linear classifier to tackle the financial prediction problem with respect to 10 American individual stock daily closing prices [16]. In their work, TDNN used are designed as "feed-forward multi-layer perceptrons, in which the internal weights are replaced by finite impulse response filters" to predict trends but not prices; PNN approximate the Bayesian decision rules; RNN is "a type of discrete-time recurrent multi-layer perceptrons". The linear classifier used is the Fisher linear classifier [31]. We evaluate CGM-GP against the three NNs and the linear classifier based on a specific prediction problem: to predict whether or not the price will rise 2% or more within 22 days.

4.3.1. Training and test data

We requested 10 stocks closing data from the author, Mr. Saad. The 10 stocks cover a larger variety of categories:

- Apple (AAPL), IBM (IBM), Motorola (MOT) and Microsoft (MSFT) represent the technology group which generally has high volatility.
- American Express (AXP) and Well Fargo (WFC) represent the banks.
- Walt Disney Co. (DIS) and McDonald (MCD) represent the consumer stocks.
- Public Svc New Mexico (PNM) and Energras (V.EEG) are cyclical stocks.

Following what was done in [16], for each stock, the last 100 days were chosen as the test data. All data series were ended at 06/03/1997, but with different starting dates.

4.3.2. Experiments

In experiments, for each data set, we run CGM-GP 10 times. For each run, we took 500 trading data just before 100 test data as training data, and took a constraint $R = [20\%, 30\%]$ for most data set except for AAPL, PNM and V.EEG, for which we took a constraint $R' = [10\%, 20\%]$. The three weights in the CGM fitness function f_{CGM} are $w_{RC} = w_{RF} = 1$ and $w_{RMC} = 0$.

Here, for each stock data set, we report both the mean and the standard deviation (STD) of GDTs' results over 10 runs on the test data. Results reported here focus on RF and the number of signals (N_+) as they were selected as the performance indicators in [16]. Note that an ideal method should be capable of achieving a lower RF and meanwhile setting off a larger number of signals (N_+). In order to compare fairly, we selected the best GDT from 10 runs in terms of the performance of RF and reported its results. This strategy follows the one in [16] in which only the best result of three NNs were reported for each stock data.

4.3.3. Comparative results

Table 10 lists all performance results of three different NNs, a linear classifier and CGM-GP on 10 stocks with respect to RF and the number of signals (N_+). The "Total" column summarises the total number of signals on all 10 stocks for each method. The last column, "Ave." column reports the average rate of failure over 10 stocks.

Like NNs., CGM-GP out-performs the linear classifier for most stocks. 10 best GDTs set off 373 signals totally, which is slightly more than 372, set off by the linear classifier. However, the average RF of the 10 best GDTs (5.08%) is much better than 18.62%, of the linear classifier.

The best results of GDTs are either as good as or better than those of NNs in terms of the number of "zero" prediction failure over the total 10 stocks. The best of the above 10 GDTs, achieved 8 zero-RF, in contrast, PNN only got 4 zero-RF; RNN, 5 zero-RF, and TDNN, 8 zero-RF. Although both the best GDT and TDNN achieve equally 8 zero-RF, the total number of signals set off by the best GDT over all 10 stocks is more than twice as large as that by TDNN (i.e. 385 versus 186). In terms of the average RF, the best GDT, which achieves a mean RF of 1.29% over 10 data sets, out-performs each of three types of NNs, which achieve average RF s of 3.05%, 3.61% and 7.56%, respectively.

Table 10
Performance comparisons among NNs, a linear classifier and CGM-GP in terms of *RF* and N_+ (the total number of positive positions predicted).

	Stocks										Total	Ave.
	AAPL	IBM	MOT	MSFT	AXP	WFC	DIS	MCD	PNM	V.EEG		
Profit opp. ($r = 2\%$; $n = 22$)	62	72	81	87	92	85	74	73	50	70	746	74.6
PNN												
Total N_+	51	25	48	49	20	45	19	4	63	14	338	
RF (%)	7.84	4.00	18.75	4.08	0.00	4.44	0.00	0.00	36.50	0.00		7.56
TDNN												
Total N_+	10	9	27	61	17	19	7	6	22	8	186	
RF (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	18.00	12.50		3.05
RNN												
Total N_+	16	22	33	46	49	29	48	53	35	37	368	
RF (%)	0.00	0.00	3.03	2.17	0.00	0.00	0.00	5.66	17.14	8.11		3.61
The linear classifier												
Total N_+	82	24	87	17	10	22	2	32	20	76	372	
RF (%)	31.71	20.83	18.39	0.00	0.00	13.64	0.00	21.88	60.00	19.74		18.62
CGM-GP (10 GDTs)												
Total N_+ mean	18.5	68.7	20.7	26.8	38.3	66.6	20.1	40.2	23.4	49.4	373	
Total N_+ STD	9.9	3.9	5.1	6.2	9.9	11.1	3.1	1.8	5.9	9.6		
RF (%) mean	9.16	10.15	1.33	3.10	3.72	8.20	0.40	0.00	13.07	4.83		5.08
RF (%) STD	5.66	1.13	2.82	2.47	3.10	2.33	1.30	0.00	12.30	3.90		
The best GDT from the above 10												
Total N_+	4	70	28	33	39	69	22	43	28	49	385	
RF (%)	0.00	8.57	0.00	0.00	0.00	4.35	0.00	0.00	0.00	0.00		1.29

The prediction is for $r = 2\%$ return within $n = 22$ days.

Our conclusion, based on Table 10, is that CGM-GP performs significantly better than the linear classifier and favorably compares against each of three NNs.

Results presented here are merely based on one set of better solutions that are chosen by us. The solutions we think, have a good trade-off between the performance of *RF* and the quantity of signals. Numeric potential solutions are still available if different constraints were applied to the task. By adjusting the size of constraint in the CGM fitness function, either a further lower *RF* is still available at the price of reducing the number of signals or a further higher *RF* is still achievable with the consequence of increasing the number of signals. This provides more options to the users with different preferences.

4.4. Related work

The specific task addressed here is closely related to cost-sensitive learning, a subject in machine learning [32]. More specifically, the target that the novel constraint-guided fitness function is intended to attack is similar to misclassification-cost classification.

Machine learning approaches to misclassification-cost classification generally fall into three main categories based on data processing stages: (1) pre-processing: re-sampling training data [33]; (2) during processing: varieties of biases applied in the process of building decision trees [34]; (3) post-processing: adjustment of threshold or ordering rules generated [35]. In general, those approaches show superiority to their corresponding systems without considering misclassification cost. However, none of them could reduce misclassification errors under certain controls. In contrast, the constraint embedded in the fitness function f_{CGM}, enable the misclassification errors to be reduced (i.e., the rate of failure) progressively. The smaller the constraint is chosen, the lower rate of failure is achievable.

There are also several papers that address classification problems using genetic programming (e.g., [36,37]. However, none of the approaches could address cost-sensitive classification problems.

To our best knowledge, ICET [32] is the only evolutionary algorithms based system that could handle misclassification costs. But ICET uses genetic algorithm as a supplementary means of finding a set of better parameters for a decision tree induction algorithm. The fittest tree is constructed directly through decision tree induction algorithms, rather than genetic algorithms. In contrast, our CGM approach allows genetic programming to act directly as a main technique to attack cost-sensitive classification problems. Besides, like other cost-sensitive methods in learning system, ICET cannot provide no means of finding varied potential solutions either.

We argue that novelty of CGM in handling misclassification cost is to provide users with a means to effectively identify various solutions. Some solutions are more favorable to others depending on user's preferences. This study shows that varying the constraints, embedded in fitness function, could easily lead to different GDTs with varied *RF* s accordingly. Investors tend to choose the GDTs that likely reflect their risk preferences.

5. Conclusion

This research is motivated by the need to cope with the difficulties caused by various constraints in many economic problems. We introduce a constraint-guided method, CGM as a constraint handling technique used with evolutionary algorithms. A CGM is especially suitable for optimization problems that have both hard and soft constraints. CGM categorizes solutions into different groups by the nature of constraints. It defines the relevance of each type of constraint into the quality of a solution. Thus all candidate solutions can be categorized into partially ordered sets. The set of solutions that violates hard constraints is definitely less favorable to the set of solutions that only violates some soft constraints, which is in turn less favorable to solutions that violate no constraint. The importance of constraints is thus translated into the fitness function of evolutionary algorithms. This technique helps guide the evolutionary search by allocating more effort into more promising spaces in which solutions are given higher fitness, without totally denying access to other areas.

Applications of CGM to two economic optimization problems having both hard and soft constraints are reported to evaluate the usefulness of CGM. For the bargaining problems, we have developed GP systems that incorporate CGM to explore bargaining strategies. The novelty of CGM in this application lies in a crucial constraint-handling mechanism embedded into the fitness function. We have demonstrated that CGM successfully produces solutions with less variations to the game-theoretic solutions than a comparable penalty function, a repair method and a no-constraint-handling setting. These findings also apply to two other types of bargaining problems, namely incomplete information bargaining problems and outside option bargaining problems.

For the financial prediction problem, CGM has been embedded into GP system to handle constraints. Illustration is given by a comparative study, together with analysis in detail. CGM-GP is compared against three NNs and a linear classifier system [16] with respect to the same prediction task over several individual American share prices available to us. Results show that CGM-GP beats a linear classifier and compares favorably against NNs. We have reviewed most closely related work in machine learning, particularly in cost-sensitive learning as well as work using classification-oriented evolutionary algorithms. We conclude that the CGM principle is effective for achieving the goal of reducing the rate of failure to make each forecasting more reliable in financial prediction problems. This enables users to make cautious investments, which is particularly relevant in finance. By turning size of a constraint, CGM-GP is capable of achieving different levels of rate of failure. This makes CGM-GP more attractive. It provides the users multiple options, which may reflect the users' different investment preferences.

Although this study presents and demonstrates the constraint-guided method in bargaining problems and financial forecasting, the ideas we introduce are general and the phenomena we have observed here do not seem to depend on any special properties of bargaining problems or financial forecasting. CGM can be applied to other optimization problems with constraints as well. More importantly, we argue that the concepts of guiding and embedding constraints into the processes of decision tree generation could potentially be applicable to other problem domains in machine learning, in particular, when misclassification costs need to be taken into account.

So far, our consistent and encouraging results have demonstrated that CGM is a promising constraint handling technique used with genetic programming that is worth further investigation and development. We have been exploring the potential of incorporating constraint satisfaction techniques [6] into GP to further improve its capability. We believe that such a combination is of value for several reasons: first, in practice, constraints ubiquitously exist in real economic problems and financial markets, for example, time window, transaction costs, risk of investment and capital adequacy, etc. Second, different users may have different preferences. Such preferences are preferably to be reflected in decision-making. Third, suitable constraints define reasonable search spaces that could facilitate genetic programming technique to work efficiently and effectively. Fourth, any little improvement by using such techniques would mean a lot.

Acknowledgments

Nanlin Jin and Jin Li were partly supported by University of Essex Research Studentships.

Authors want to thank Dr Alberto Moraglio (University of Coimbra, Portugal), Aimin Zhou (University of Essex, UK), Dr Ken Brown (University College Cork, Ireland), and Andy Turner (University of Leeds, UK) for their comments.

The comments from the editor and anonymous reviewers were also helpful in improving the quality of the paper.

References

[1] A.E. Eiben, P.-E. Raue, Z. Ruttkay, GA-easy and GA-hard constraint satisfaction problems, pp. 267–283, in: Constraint Processing, Selected Papers, 1995.
[2] B.M. Smith, S.A. Grant, Modelling exceptionally hard constraint satisfaction problems, pp. 182–195, in: Principles and Practice of Constraint Programming, 1997.
[3] S. Bistarelli, P. Codognet, F. Rossi, Abstracting soft constraints: framework, properties, examples, Artificial Intelligence 139 (2) (2002) 175–211.
[4] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint satisfaction and optimization, JACM: Journal of the ACM 44 (1997).
[5] M.C. Cooper, T. Schiex, Arc consistency for soft constraints, Artificial Intelligence 154 (1–2) (2004) 199–237.
[6] E. Tsang, Foundations of Constraint Satisfaction, Academic Press, 1993.
[7] J.-P. Rennard, Handbook of Research on Nature Inspired Computing for Economy and Management, Idea Group, Inc, 2006.
[8] S. Lucas, G. Kendall, Evolutionary computation and games, IEEE Computational Intelligence Magazine 1 (1) (February 2006) 10–18.
[9] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, USA, 1992.
[10] C.A. Coello Coello, A survey constraint handling techniques used evolutionary algorithms, Technical Report Technical Report Lania-RI-9904, Laboratorio Nacional de Informtica Avanzada, 1999.
[11] Z. Michalewicz, A survey of constraint handling techniques in evolutionary computation methods, in: Evolutionary Programming, MIT Press, Cambridge, MA, 1995, pp. 135–155.
[12] T.P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Transactions on Evolutionary Computation 4 (3) (2000) 284–294.
[13] G.E. Liepins, M.D. Vose, Representational issues in genetic optimization, Journal of Experimental and Theoretical Artificial Intelligence 2 (1990) 101–115.
[14] G.E. Liepins, W.D. Potter, A genetic algorithm aproach to multiple-fault diagnosis, in: L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp. 237–250.
[15] Z. Michalewicz, C.Z. Janikow, GENOCOP: a genetic algorithm for numerical optimization problems with linear constraints, Communications of the ACM 39 (12es) (1996).
[16] E.W. Saad, D.V. Prokhorov, D.C. Wunsch III, Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks, IEEE Transactions on Neural Networks 9 (November (6)) (1998) 1456–1470.
[17] B.G.W. Craenen, A.E. Eiben, J.I. van Hemert, Comparing evolutionary algorithms on binary constraint satisfaction problems, IEEE Transaction on Evolutionary Computation 7 (5) (2003).
[18] A. Muthoo, Bargaining Theory and Applications, first edition, Cambridge University Press, Cambridge, UK, 1999.
[19] P. Faratin, C. Sierra, N.R. Jennings, Negotiation decision functions for autonomous agents, International Journal of Robotics and Autonomous Systems 24 (3–4) (1998) 159–182.
[20] A. Rubinstein, Perfect equilibrium in a bargaining model, Econometrica 50 (1) (1982) 97–110.
[21] R. Hinterding, Z. Michalewicz, A.E. Eiben, Adaptation in evolutionary computation: a survey, in: IEEECEP: Proceedings of the IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, 1997.
[22] W.B. Langdon, R. Poli, Foundations of Genetic Programming, Springer-Verlag, 2002.
[23] N. Jin, Equilibrium selection by co-evolution for bargaining problems under incomplete information about time preferences, in: D. Corne, et al. (Eds.), Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 3, Edinburgh, UK, 2005, pp. 2661–2668.
[24] N. Jin, E. Tsang, Co-adaptive strategies for sequential bargaining problems with discount factors and outside options, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, 2006.
[25] E.P.K. Tsang, J. Li, J.M. Butler, EDDIE beats the bookies, Software: Practice and Experience 28 (10) (1998) 1033–1043.
[26] E. Tsang, P. Yung, J. Li, EDDIE-automation, a decision support tool for financial forecasting, Decision Support Systems 37 (4) (2004) 559–565.
[27] S.S. Alexander, The Random Character of Stock Market Prices, Chapter Price Movement in Speculative Markets: Trend or Random Walks, no. 2, MIT Press, Cambridge, MA, 1964, pp. 338–372.
[28] R.J. Sweeney, Some new filter rule tests: methods and results, Journal of Financial and Quantitative Analysis 23 (1988) 285–300.
[29] W. Brock, J. Lakonishok, B. LeBaron, Simple technical trading rules and the stochastic properties of stock returns, Journal of Finance 47 (1992) 1731–1764.
[30] E.F. Fama, M.E. Blume, Filter rules and stock-market trading, Journal of Business 39 (1) (1966) 226–241.
[31] A. Fisher, The Mathematical Theory of Probabilities, vol. 1, McMillan, New York, 1923.
[32] P.D. Turney, Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm, Journal of Artificial Intelligence Research 2 (1995) 369–409.

[33] P. Chan, S. Stolfo, Scaling learning by meta-learning over disjoint and partially replicated data, in: Proceedings of the Ninth Florida AI Research Symposium, 1996, pp. 151–155.
[34] C. Nedellec, Rouveirol, Pruning decision trees with misclassification costs, in: ECML, vol. 1398 of Lecture Notes in Computer Science, Springer, 1998, pp. 131–136.
[35] T. Fawcett, F. Provost, Adaptive fraud detection, Data Mining and Knowledge Discovery 1 (3) (1997) 291–316.
[36] C.C. Bojarczuk, H.S. Lopes, A.A. Freitas, Discovering comprehensible classification rules by using genetic programming: a case study in a medical domain, in: W. Banzhaf, J.M. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M.J. Jakiela, R.E. Smith (Eds.), GECCO, Morgan Kaufmann, 1999, pp. 953–958.
[37] N.I. Nikolaev, V. Slavov, Inductive genetic programming with decision trees, in: 9th European Conference on Machine Learning, Prague, Czech Republic, (April 1997), pp. 3–26.