Neural, Parallel, and Scientific Computations 19 (2011) 1-20

COOPERATIVE GAMES IN MARKETING: A DIFFERENTIAL GAME APPROACH

N. G. MEDHIN AND WEI WAN

Department of Mathematical Science, North Carolina State University
Raleigh, North Carolina 27695, USA
Department of Mathematical Science, Claflin University
Orangeburg, South Carolina 29115, USA

ABSTRACT. Cooperation is not applicable in a constant-sum differential game because there is no possibility of mutual gain from cooperation. However, in other types of games cooperation is possible. The fate of a game being noncooperative or cooperative is generally determined by the 'scope' of the game, which means how many people will be included in the game, and the status of the players. In reality it is possible to have both cooperative and noncooperative games, or just one of them, in different situations/phases of the same game. In this paper we will set up a cooperative differential game model for competition in marketing. There are many ways to define the solution of a cooperative differential game. Different definitions of 'solution' will precipitate totally different models. After defining the solution for our cooperative differential game model, we develop an algorithm based on the idea of evolutionary computation, and then draw conclusions about general cooperative differential games.

AMS (MOS) Subject Classification. 91A12, 91A23, 91A80, 91B50, 91B60

Keywords: cooperative differential games, numerical methods, marketing

1. INTRODUCTION

Our motivation to study cooperative differential game is based on two facts: first is that in real marketing competition, cooperation may be seen more often than pure competition; second, the methods for solving cooperative differential game model is limited in literatures. In the literature there are few papers dealing with cooperative differential games for market competition, or how to solve this type of models. It is challenging to solve cooperative differential game analytically, even in very special and simple models. The difficulty in application of this model comes from its solvability. Furthermore, there are many ways to define the solution of cooperative differential game since there are many ways for competitors to cooperate. And different ways to define the solution will lead to different ways to solve it. In this paper we set up a general cooperative differential game model for competition in the final stage of a product life cycle, and then proceed to derive optimality conditions for the model, at last develop a numerical algorithm to solve it. Our algorithm is based on Evolutionary Computation. This may be the first exploration of this way to solve cooperative differential game models. The difficulties to solve our model is that we need solve a functional minimization problem with constraints coming from two different spaces: one is $\mathbb{C}^1$, the other is $\mathbb{R}^n$. Thus, our investigation of using Evolutionary Computation

Received November 23, 2010

to solve cooperative differential games can also be extended to solve similar types of constrained optimization problem.

Based on numerical results, we analyze the differences between cooperative and noncooperative differential games, and draw some useful practical guidelines. For example, cooperation in competition is better than noncooperative competition although how to distribute benefit is another problem in a cooperative differential game. At last we bring out another problem in real competition, which is how to choose a game. In reality, there are many factors that determine or affect the choice of a game. And the choice of a game is a dynamic process determined by the 'scope' of the game. Figuring out or choosing the 'scope' of the game is more important than just solving a game.

2. COOPERATIVE DIFFERENTIAL GAME

When we talk about any mathematical model, we should define what we mean by the 'solution' of the model clearly. In a noncooperative game, we can define the *saddle point* as the solution of the game. In a cooperative zero-sum game, we can still define the *saddle point* as the solution of the game. However, in most cooperative variable-sum games, a solution is much harder to define. Different solutions are defined based on practical need. Regardless of the definition, we interpret the solution of a cooperative game as a way to coordinate the players' decisions effectively.

A typical optimization problem in Operations Research consists of an objective function, which represents the goal or payoff of decision makers, with some constraints.

The cooperative differential game we consider is such that each competitor has the following objective function:

$$J_i(u_i) = \int_0^T g_i(t, x_1(t), \cdots, x_n(t), u_1(t), \cdots, u_n(t))dt + h_i(x_1(T), \cdots, x_n(T))$$

and the dynamics of the state are governed by the system of differential equations:

$$\begin{cases} \dot{x}_1 = & f_1(t, x_1(t), \cdots, x_n(t), u_1(t), \cdots, u_n(t)) \\ & \vdots \\ \dot{x}_i = & f_i(t, x_1(t), \cdots, x_n(t), u_1(t), \cdots, u_n(t)) \\ & \vdots \\ \dot{x}_n = & f_n(t, x_1(t), \cdots, x_n(t), u_1(t), \cdots, u_n(t)) \\ x_i(0) & \textit{given} \end{cases}$$

Now suppose that these **n** players cooperate under some agreement, then at first we are concerned with the following questions:

1. Under what conditions are these players willing to cooperate? That is, why would a player want to cooperate? If a player gets 'less' than what he gets in a noncooperative game, then there is no incentive for him to cooperate. This is called *individual rationality axiom.* More discussion about individual rationality can be found in Jorgensen, Martin-Herran and Zaccour (2003) [9]. Based on this rationality axiom, we can imagine that a necessary condition for existence of a cooperative game is that each player should get at least what he can get

in the noncooperative case. Later we will define *Pareto solution* of cooperative differential game on the basis of this condition.

2. What will be an appropriate payoff functional for a cooperative differential game? There are various ways of defining the solution of a cooperative game. One way is to consider the payoff functional:

$$\begin{aligned} J_0(u_1,\cdots,u_n) &= \max\Big\{\int_0^T g_i(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t))dt \\ &+ h_i(x_1(T),\cdots,x_n(T)), i=1,\cdots,n\Big\} \end{aligned}$$

Because of cooperation, we will take the **n** players as one decision maker. And we accept the weighted sum of the individual payoffs as the new payoff functional in a cooperative game. This was first introduced by Leitmann (1974) [1]:

$$\begin{aligned} J_0(u_1,\cdots,u_n) &= \sum_{i=1}^n c_i \int_0^T g_i(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t))dt \\ &+ \sum_{i=1}^n c_i h_i(x_1(T),\cdots,x_n(T)) \end{aligned}$$

where $\mathbf{c}_i$ is interpreted as a 'bargaining weight'.

We will maximize/minimize the objective function J_0 subject to the system of differential equations:

$$\begin{cases} \dot{x}_1 = & f_1(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t)) \\ & \vdots \\ \dot{x}_i = & f_i(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t)) \\ & \vdots \\ \dot{x}_n = & f_n(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t)) \\ x_i(0) & \textit{given} \end{cases}$$

In this formulation we can expect to face the problem of solving a typical optimal control problem.

3. How will the optimal cooperative objective function value be divided among these players? One way is to divide the optimal payoff $\mathbf{J}_0^*$ among the players in proportion to their contribution.

So we can see that different answers to the above questions will lead us to totally different set-up of the problem. The above questions and our answers to them form the foundation for the basic frame-work of our problem. Given $\mathbf{C} = \{\mathbf{c_1}, \mathbf{c_2}, \cdots \mathbf{c_n}\}$, we have the following optimal control problem:

$$\begin{cases} \min_{u_1,\cdots,u_n} J_0 & = \sum_{i=1}^n c_i \int_0^T g_i(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t))dt \\ & +c_i \sum_{i=1}^n h_i(x_1(T),\cdots,x_n(T)) \\ \quad s.t. & \\ \qquad \dot{x}_1 = & f_1(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t)) \\ & \vdots \\ \qquad \dot{x}_i = & f_i(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t)) \\ & \vdots \\ \qquad \dot{x}_n = & f_n(t,x_1(t),\cdots,x_n(t),u_1(t),\cdots,u_n(t)) \\ & \\ & x_i(0) = x_{i0}, \quad i = 1,\cdots,n-1 \end{cases}$$

By solving the above problem, we can get the optimal objective function value $J_0^*(c_1,\cdots,c_n)$. Using this relation we can define a mapping:

$$\mathbf{T} : \mathbb{R}^n \mapsto \mathbb{R}.$$

Then, our task is find out what $\min \mathbf{T}(\mathbf{c_1},\cdots,\mathbf{c_n})$ is and a minimizing vector $\mathbf{C} = [\mathbf{c_1},\mathbf{c_2},\cdots,\mathbf{c_n}]$.

Definition 2.1. *Argmin*$\{\mathbf{T}(c_1,\cdots,c_n)\}$ is defined as **Pareto solution** of the cooperative differential game.

The functional $\mathbf{T}$ is complicated and has no explicit expression, so direct search seems a good choice of minimizing it. In every direct search method, the key issue is how to generate variations of the parameter vectors. Once some variation is generated, we can make a choice to accept or decline based on some criteria(objective function). In all standard direct search methods, the new generated parameter will be accepted if and only if it makes our objective value better. This is called *greedy criteria*. The weak point of this criteria is the risk of falling into a local minimum/maximum. However, evolutionary computation can help ameliorate this problem.

3. MODEL DEVELOPMENT

We proceed to consider competition in the final stage of a product life cycle, where **n**-companies and **one** product are involved. The market managers use controls/advertising to minimize cost. Because the sale will decrease to zero in the long run, managers will not consider market share as their objectives. We use the index $i = 1,2,\cdots,n$ to represent these **n** companies. The main notations are as follows:

$\mathbf{x}_i(t)$	Market share of company i at time t.
$\mathbf{u}_i(t)$	Control/Advertising of company i at time t.
ρ	Natural sale decrease factor without advertising.
κ	Effectiveness of advertising on the sales decrease rate.
$\mathbf{a}_i$	Effectiveness of control/advertising of company i.
δ_i	Advertising cost parameter for company i.
$\mathbf{p}$	Price of the product.

The sale of each company will be affected by a natural decrease rate which will be affected by advertising, and competition driven by advertising. Each company wants to minimize cost. So each company's objective function is:

$$J_i(u_i) = \int_{t_0}^{T} e^{-rt}[\frac{\delta_i}{2}u_i^2(t) - px_i(t)]dt, \quad i = 1, 2, \cdots, n$$

and the dynamics are

$$\begin{bmatrix} \frac{dx_1(t)}{dt} \\ \vdots \\ \frac{x_n(t)}{dt} \end{bmatrix} = \begin{bmatrix} a_1u_1(t) \\ \vdots \\ a_nu_n(t) \end{bmatrix} - \begin{bmatrix} f+G & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f+G \end{bmatrix} \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix}$$

where $f = \rho(1 - \kappa \sum_{i=1}^n u_i(t))$, $G = \sum_{i=1}^n a_iu_i(t)$ and $x_i(0)$ are given. Detail explanation about how the objective function and dynamics come can be seen in [4].

In a cooperative differential game, we will take these **n** companies as a whole. Then, we pretend as if there is a 'higher' manager, who will control the activities of these **n** companies and maximize a combined benefit. This combined objective function is set up by taking a convex combination of the **n** objective functions. Then, the cooperative differential game model for these **n** companies is as follows:

$$\begin{cases} \min_{c_1,\cdots,c_n} \min_{u_1,\cdots,u_n} & J_0 = \sum_{i=1}^n c_i \int_0^T e^{-rt}[\frac{\delta_i}{2}u_i^2(t) - px_i(t)]dt \\ s.t. & \\ \frac{dx_1(t)}{dt} = & a_1u_1(t) - (f+G)x_1(t) \\ & \vdots \\ \frac{dx_i(t)}{dt} = & a_iu_i(t) - (f+G)x_i(t) \\ & \vdots \\ \frac{dx_n(t)}{dt} = & a_nu_n(t) - (f+G)x_n(t) \\ & \\ x_i(0) = & x_{i0}, \quad i = 1, \cdots, n-1 \\ \sum_{i=1}^n c_i = & 1, \quad c_i \geq 0 \end{cases}$$

If we fix a vector $\mathbf{C} = [\mathbf{c_1}, \cdots, \mathbf{c_n}]$, then we face a typical optimal control problem. So, it seems that given one $\mathbf{C}$, we can solve one typical optimal control problem and get a corresponding optimal objective function value $\mathbf{J_0(C)}$. In order to make this idea make sense, we need following theorem:

Theorem 3.1.

$$\textit{Suppose} \quad f : \Omega \times \Omega \mapsto \mathbf{R}, \quad \textit{then} \quad \min_{x\in\Omega} \min_{y\in\Omega} f(x,y) \equiv \min_{(x,y)\in\Omega\times\Omega} f(x,y).$$

Proof. It is obvious that $\min_{x\in\Omega}\min_{y\in\Omega} f(x,y) \geq \min_{(x,y)\in\Omega\times\Omega} f(x,y)$. Assume that $\min_{x\in\Omega}\min_{y\in\Omega} f(x,y)$ attains minimum at (x_1, y_1), and $\min_{(x,y)\in\Omega\times\Omega} f(x,y)$ attains minimum at (x_2, y_2). And assume that $f(x_1, y_1) > f(x_2, y_2)$. Since $f(x_2, y_2)$ is global minimum, we have $f(x_2, y_2) = \min_y f(x_2, y)$. And because $f(x_1, y_1) = \min_y f(x_1, y)$, therefore $\min_y f(x_1, y) > \min_y f(x_2, y)$ according to assumption $f(x_1, y_1) > f(x_2, y_2)$. Hence, $\min_x \min_y f(x,y) \leq \min_y f(x_2, y) < \min_y f(x_1, y)$, which implies
$\min_{(x,y)\in\Omega\times\Omega} f(x,y)$ cannot attaint minimum at (x_1, y_1). So it is contradictory, and we get the equality. □

By the above theorem, we can search for the optimal solution of the above differential game through following steps: given any vector $\mathbf{C}$, we solve an optimal control problem and get a J_0. We will continue till we find such a $\mathbf{C}$ such that J_0 attains minimum.

First, we take any $\mathbf{C}$ and we draw the necessary conditions for the above optimal control problem. The Hamiltonian for above optimal control problem is:

$$H = e^{-rt}\left[\sum_{i=1}^{n} c_i(\frac{\delta_i}{2}u_i^2 - px_i)\right] + \sum_{j=1}^{n}\lambda_j[a_j u_j - (f+G)x_j]$$

Minimizing H with respect to u_i:

$$\frac{\partial H}{\partial u_i} = e^{-rt}c_i\delta_i u_i + a_i\lambda_i - (a_i - \rho k)\sum_{j=1}^{n}\lambda_j x_j$$

Solving $\frac{\partial H}{\partial u_i}$ for u_i explicitly:

$$u_i = \frac{e^{rt}}{c_i\delta_i}\left[(a_i - \rho k)\sum_{j=1}^{n}\lambda_j x_j - a_i\lambda_i\right], \quad i = 1, \cdots, n$$

The costate system comes from:

$$\dot{\lambda}_i = -\frac{\partial H}{\partial x_i} = \lambda_i(f+G) + e^{-rt}pc_i, \quad i = 1, \cdots, n$$

So in order to solve the above optimal control problem, we should solve the following boundary value problem (BVP):

$$\begin{bmatrix} \dot{x}_i \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = -\begin{bmatrix} f+G & \cdots & 0 \\ 0 & \ddots & 0 \\ \vdots & f+G & \vdots \\ 0 & \ddots & 0 \\ 0 & \cdots & f+G \end{bmatrix}\begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} a_1u_1 \\ \vdots \\ a_iu_i \\ \vdots \\ a_nu_n \end{bmatrix}$$

$$\begin{bmatrix} \dot{\lambda}_1 \\ \vdots \\ \lambda_i \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} f+G & \cdots & 0 \\ 0 & \ddots & 0 \\ \vdots & f+G & \vdots \\ 0 & \ddots & 0 \\ 0 & \cdots & f+G \end{bmatrix}\begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_i \\ \vdots \\ \lambda_n \end{bmatrix} + pe^{-rt}\begin{bmatrix} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{bmatrix}$$

with boundary conditions:

$$\begin{bmatrix} x_1(0) \\ \vdots \\ x_i(0) \\ \vdots \\ x_n(0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ \vdots \\ x_{i0} \\ \vdots \\ x_{n0} \end{bmatrix}$$

$$\begin{bmatrix} \lambda_1(T) \\ \vdots \\ \lambda_i(T) \\ \vdots \\ \lambda_n(T) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Thus, given a vector $\mathbf{C} \in \mathbb{R}^n$, we can solve an optimal control problem to get its optimal objective value by solving a two point boundary value problem (TBVP). We now define this relation to be a functional:

$$T : \mathbf{R}^n \mapsto \mathbf{R}$$

However, the explicit formula for $\mathbf{T}$ does not exist or is unknown, and the feasible domain is large. Thus, we cannot solve this optimization problem by the usual methods from convex optimization or optimization techniques related to gradient. Therefore we choose evolutionary computation. And in the appendix, an algorithm [4] is given to solve above optimal control where $\mathbf{C}$ is a constant vector. This algorithm will be called by our evolutionary algorithm.

4. **EVOLUTIONARY COMPUTATION**

In nonlinear programming problems, the classical methods such as gradient descent, Newton method, and hill climbing work well in some specific problems. In this paper we employ evolutionary computation. Later, we can see evolutionary computation may be the only way to solve our optimization problem.

4.1. **Operators of Evolutionary Computation.**

As is well known, any evolutionary computation technique will consider the following factors:
1) Representation of individuals in a selected population.
2) A set of operators to generate the next generation.
3) The rate of evaluating the fitness of individuals.

Mutation and recombination operator will be used on the current population to generate a new population. Mutation operator usually means the creation of a new solution from one and only one parent. However, recombination operator creates new individuals by recombining some parts from several individuals. In the following we will design the mutation and recombination operation used in our algorithm.

The general form of mutation can be in the following form:

$$C^k(t+1) = m(C^k(t))$$

where $\mathbf{C}^k(t)$ represents individual $\mathbf{k}$ at generation $\mathbf{t}$, and $\mathbf{m}(\cdot)$ represents mutation operator. In our problem, we will take real representation for each individual, that is,

$$C^k(t) \in \left\{ (c_1^k, \cdots, c_n^k) \in \mathbf{R}^n \mid \sum_{i=1}^{n} c_i^k = 1, c_i > 0 \right\}$$

We will consider two kinds of mutations. The first is

$$C^k(t+1) = m(C^k(t)) = C^k(t) + M^k$$

where $M^k = [M_1^k, M_2^k, \cdots, M_n^k]$ is a random vector with $\sum_{i=1}^n M_i^k = 0$, and M_i^k is random variable from uniform distribution $\mathbf{U}[-a_i, +a_i]$, where $0 < a_i < \min\{c_i^k(t)\}$. Intuitively speaking, we randomly perturb an individual locally to generate a next generation. This kind of perturbation is so small that it will keep all generated points in feasible domain. From the following *Figure 1*, we can visualize this type of mutation in 3-*dimensional* space. This type of mutation may lead to possible entrapment in a local extremum. In our algorithm we use it to look for local extremum. We hope such mutation could find individuals with better fitness locally. Our second mutation will help us look in a larger area.

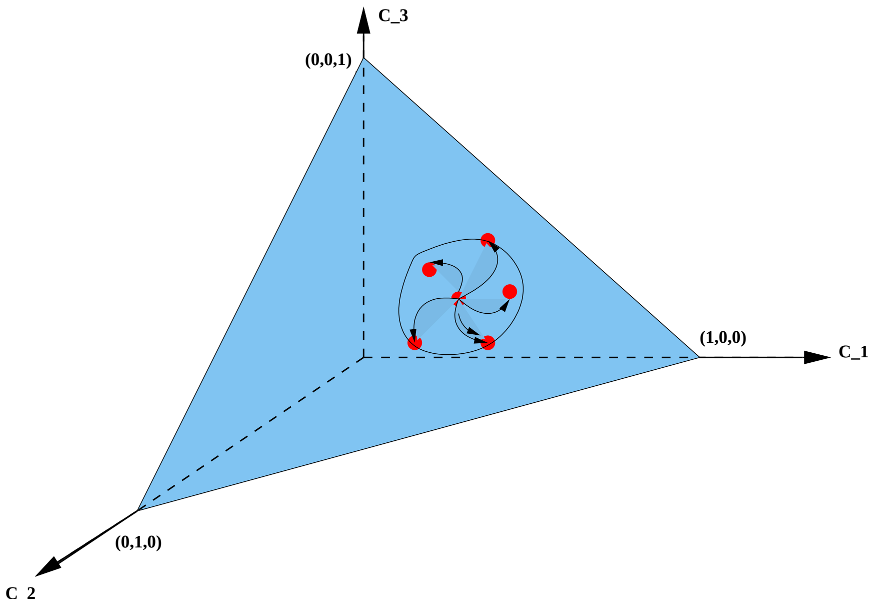

FIGURE 1. First kind of Mutation

The second kind of mutation we will use is following:

$$\begin{aligned} C^k(t+1) &= m(C^k(t)) \\ &= m((c_1^k, c_2^k, \cdots, c_n^k)) \\ &= (c_{i_1}^k, c_{i_2}^k, \cdots, c_{i_n}^k) \end{aligned}$$

where the vector $[\mathbf{i_1}, \mathbf{i_2}, \cdots, \mathbf{i_n}]$ is random permutation of the integers $[\mathbf{1}, \mathbf{2}, \cdots, \mathbf{n}]$. With this type of mutation, we can search in a larger area of the feasible domain instead of local search. We can visualize this type of mutation in 3-*dimension* in *Figure 2*.

Following this mutation we proceed to implement recombination. In biological systems recombination/crossover is a phenomena which happens between pairs of chromosomes. Two chromosomes are paralleled together, and broken into some fragments at some places. These fragments are exchanged and linked into a new pair of chromosomes. The recombination between chromosomes contributes to the diversity of biological systems. Evolutionary computation mimics this process to form the new generation. The following is the typical crossover introduced by Holland (1975) [8]. First, two individuals are selected from the population of parents based on some rules.

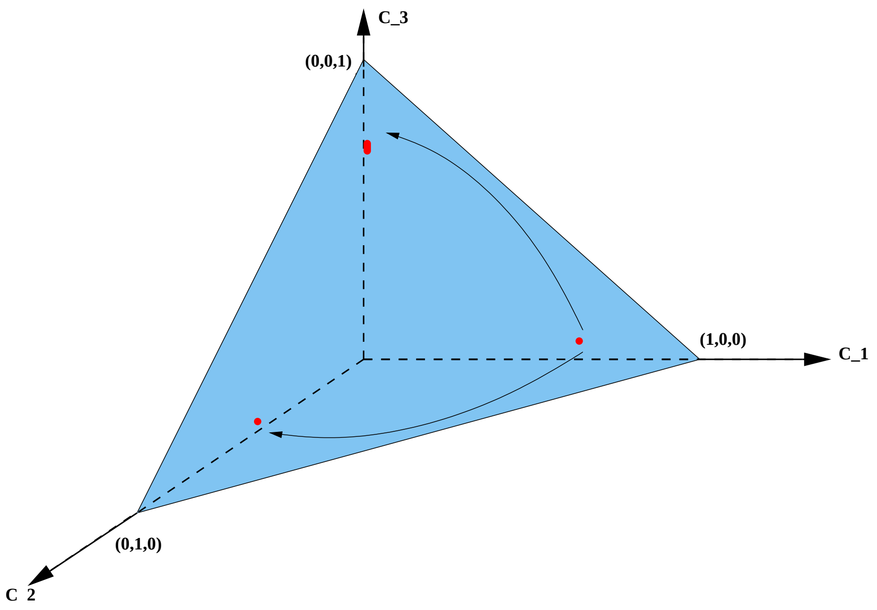

FIGURE 2. Second kind of Mutation

Second, crossover points are chosen based on some rule/s to break these two individual into several segments. Third, these segments from the parents are exchanged to form a new generation. Suppose

$$C^1(t) = [c_1(t), c_2(t), \cdots, c_k(t), c_{k+1}(t) \cdots, c_n(t)]$$

$$C^2(t) = [c'_1(t), c'_2(t), \cdots, c'_k(t), c'_{k+1}(t) \cdots, c'_n(t)]$$

The crossover point is randomly chosen from $\mathbf{U}(2, n-1)$, then we get a new pair of individuals:

$$C^1(t+1) = [c_1(t+1), c_2(t+1), \cdots, c_k(t+1), c'_{k+1}(t+1) \cdots, c'_n(t+1)]$$

$$C^2(t+1) = [c'_1(t+1), c'_2(t+1), \cdots, c'_k(t+1), c_{k+1}(t+1) \cdots, c_n(t+1)]$$

This kind of crossover is well suited when an individual is represented in binary form. When using crossover, we need to consider another issue: the proportion of parents, that will be undergoing crossover. This is called *crossover rate*, $\mathbf{p_c} \in [\mathbf{0}, \mathbf{1}]$. Usually, there are some commonly used crossover rates, $[0.45, 0.95]$ (Grefenstette, 1988) [6].

In continuous optimization, the "individuals" are usually real-valued functions. Thus, suppose $X(t), Y(t)$ are two points from the population of parents. Approximately representing $X(t), Y(t)$ in decimal form, a pair $X(t+1), Y(t+1)$ is created as follows (Reed et. al. (1967)) [10]:

$$X(t) = x_1.x_2x_3 \cdots x_kx_{k+1} \cdots x_n$$

$$Y(t) = y_1.y_2y_3 \cdots y_ky_{k+1} \cdots y_n$$

Then, choosing crossover points in the same way as above, we get

$$X(t+1) = x_1.x_2x_3 \cdots x_ky_{k+1} \cdots y_n$$

$$Y(t+1) = y_1.y_2y_3 \cdots y_kx_{k+1} \cdots x_n$$

The following graph(*Figure 3*, T. Back, 2000 [14]) illustrates the above crossover process. However, in our problem, the constraint $\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{c_i(t)} = \mathbf{1}$, on the population prevents us from using this kind of crossover, so we'd like to choose a convex recombination to generate a new offspring as follows:

$$\begin{aligned} C^{k'}(t+1) = &\ R(C^k(t), C^j(t)) \\ = &\ \alpha C^k(t) + (1-\alpha)C^j(t) \end{aligned}$$

were, $\mathbf{R}(\cdot,\cdot)$ is recombination operator, and α is randomly chosen from the interval $[0,1]$ from some distribution. It is obvious that $\sum_{i=1}^{n} c_i^{k'}(t+1) = 1$, which means the new offsprings are kept in the feasible domain.

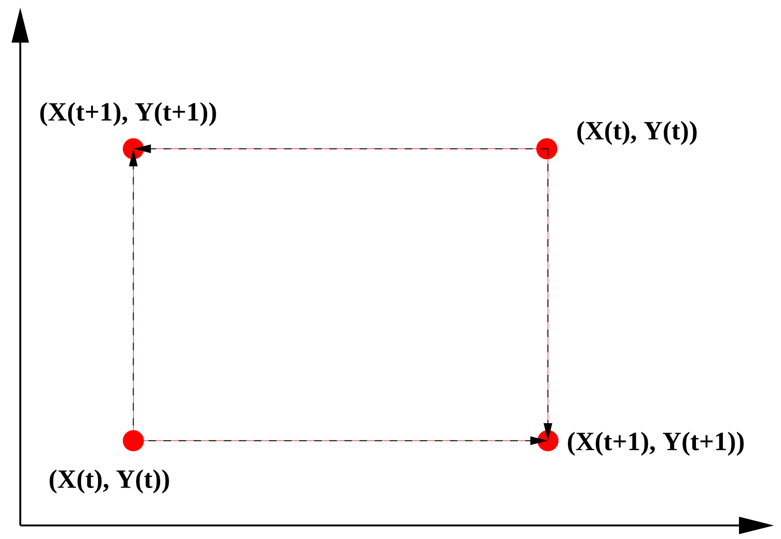

FIGURE 3. Crossover in real representation

4.2. **Selection Process of Evolutionary Computation.** After we use all of the operators to generate new individuals, we get a new generation. Then, the new generation will be subjected to a process of selection. All or part of the new generation may replace all or part of the parent generation that gave rise to it. The selection process affects the efficiency of the evolutionary computation, and it provides the driving force behind an evolutionary algorithm. When we compare different selection methods, we use a general term-'selection pressure'. 'Selection pressure' is another way of expressing 'selection criteria'. We can expect that the stricter the 'selection criteria' is, the faster the rate of convergence. In the literature there are two measures to analyze selection methods: *take-over time* and *selection intensity*. 'Take-over time' means that the average time required for the best individual in some generation from birth to be replaced by others under the effect of selection. Goldberg, Deb 1991 [7] first introduced this concept. 'Selection intensity' is defined in terms of the average fitness before and after selection. There are two typical ways of selections. One way is choosing some best individuals to reproduce according to their fitness. This is often seen in Genetic Algorithm (GA). That is, GA will begin with population of μ and select λ according to fitness to produce new generation with size of λ. However, in Evolutionary Strategies (ES) or Evolutionary Programming (EP), all individuals are allowed to reproduce. Then, choosing some best individuals from new produced individuals to make up next generations. So, at the beginning, ES/EP will initiate μ people, and produce λ people, and truncate λ people out according to their fitness and make a new generation μ. In our evolutionary algorithm to solve our model, we will adopt the way of ES/EP to generate new offspring and make selection.

5. DESIGN OF ALGORITHM

An individual in our model is a vector $\{(c_1, \cdots, c_n) \in \mathbf{R}^n | \sum_{i=1}^{n} c_i = 1, c_i \geq 0\}$, so we have a constrained optimization problem. Usually there are two ways to deal with constraints. One is using mutation operator that generates feasible solutions. Another way to deal with constraints is using a penalty function which eliminates infeasible offsprings from becoming part of the next generation. In our algorithm we will adopt the first way to make an offspring be in the feasible domain. And in each generation, the size of population stays the same, which is μ.

We take the fitness function as the functional:

$$\mathbf{T} : \mathbf{R}^n \mapsto \mathbf{R}$$

defined in section 2. Given a vector $\mathbf{C}^k(t) \in \mathbb{R}^n$, evaluating the fitness function involves solving an optimal control problem.

The process of evolutionary computation is as follows. Our algorithm begins with a 'big' population with size $\mu + \lambda$:

$$\mathbf{P(0)} = \{\mathbf{C^1(0)}, \mathbf{C^2(0)}, \cdots, \mathbf{C}^{\mu+\lambda}\mathbf{(0)}\},$$

where $\mathbf{C}^k(0)$ represents one individual, or a potential solution to the problem, which are randomly generated in the feasible domain. Then, each individual will be evaluated by a fitness function to give its fitness value.

Then, we sort $\mathbf{C}^k(0), k = 1, \cdots, \mu + \lambda$ by fitness. Then, we will choose deterministically the best μ individuals as the first generation

$$\mathbf{P(1)} = \{\mathbf{C^1(1)}, \mathbf{C^2(1)}, \cdots, \mathbf{C}^{\mu}\mathbf{(1)}\}.$$

After this, mutation and recombination process will generate $\mu + \lambda$ new individuals. These $\mu + \lambda$ individuals come in several ways:

1. We always directly move approximately $p_0\%$ of the best parents in $P(1)$ to the next generation.
2. Mutation 1. We will choose $\mu * p_1\%$ from $P(1)$ with best fitness values and apply the first kind of mutation described in section 3. We want to search for individual with better fitness values in the neighborhood of the $\mathbf{C}^k(1)$'s. Using the first type mutation, we hope to find local minimum.
3. Mutation 2. We will randomly choose $\mu * p_2\%$ individuals from $P(1)$ to apply the second type of mutation. In this way, we can search in a large area of feasible domain and escape from local minimum.
4. Recombination/crossover. We will take $\mu * p_3\%$ individuals from $P(1)$ randomly, and do convex combination between any two individuals as follows:

$$a_i C^k + (1 - a_i) C^j$$

where $\mathbf{a}_i$ is generated from Uniform distribution $\mathbf{U[0,1]}$. Then the constraints $\sum_{i=1}^{n} c_i^k = 1$ are always satisfied.

5. Immigration. In order to keep searching in a large feasible, we generate randomly some individuals.

Once we use the above selection, mutation and recombination to generate $\mu+\lambda$ individuals, we will use fitness function to evaluate the new population $\mathbf{C^1}, \mathbf{C^2}, \cdots, \mathbf{C}^{(\mu+\lambda)}$. From this new population of size $\mu+\lambda$ we select μ of them with best fitness to move them into the second generation of population:

$$\mathbf{P(2)} = \{\mathbf{C^1(2)}, \mathbf{C^2(2)}, \cdots, \mathbf{C}^{\mu}\mathbf{(2)}\}.$$

This process will continue till

$$\min\{\mathbf{T}(\mathbf{C^1}(\mathbf{t})), \cdots, \mathbf{T}(\mathbf{C}^{\mu}(\mathbf{t}))\}$$

change little or maximum iteration is arrived at. Then, this minimum value and the related vector $\mathbf{C^{k*}}(\mathbf{t})$is the solution of our model. We summarize the above process as follows:

```
$t = 0$

$Initialize: P(t) = \{C^1(t), C^2(t), \cdots, C^{\mu+\lambda}(t)\}$

$Evaluate: \{\mathbf{T}(C^1(t)), \mathbf{T}(C^2(t)), \cdots, \mathbf{T}(C^{\mu+\lambda}(t))\}$

$Select: P(1) := \{C^1(1), C^2(1), \cdots, C^{\mu}(1)\}$

$Iterate: While\ \ STOP\ \ conditions\ \ not\ \ satistied$

{
$Mutate,\ Recombine,\ Immigrate\ \ to\ \ generate\ \ a\ \ population\ \ of\ \ size\ \ \mu+\lambda$
$Evaluate: \{T(C^1(t)), T(C^2(t)), \cdots, T(C^{\mu+\lambda}(t))\}$
$Select\ \ to\ \ get\ \ P(t+1) = \{C^1(t+1), C^2(t+1), \cdots, C^{\mu}(t+1)\}$
$t = t+1$
}
```

From the above evolutionary process we can see that:

1. The size of the population is always kept same, that is μ.
2. Each individual in population $\mathbf{P(t)}$ has a chance to reproduce.
3. Deterministically each generation will keep the best individuals of the previous generation, so in each iteration the objective value J_0 will be non-increasing. In fact, we expect it to decrease.
4. In our problem, the efficiency of solving the optimal control problem is a key factor to determine whether we can use Evolutionary Algorithm to solve it successfully. An efficient algorithm to solve the optimal control problem is necessary. We will call the algorithm [4] presented in the appendix, to evaluate the fitness function.
5. The iteration process will continue till the a predetermined convergence criteria is met, or a specified number of iterations has taken place.

Now we convert the above framework of evolutionary process into executable algorithm which is used to solve our model.

Algorithm 5.1 :

Step 1:

Randomly generate 60 individuals: $\{C^i, i = 1, \cdots, 60\}$.

Step 2:

Evaluate $\{C^i, i = 1, \cdots, 60\}$ by fitness function. Set $t = 1$. Select 20 of the $\{C^i, i = 1, \cdots, 60\}$ to make up the first generation of population $\{C^i(t), i = 1, \cdots, 20\}$ by fitness.

Step 3:

Convex recombination between pairs of $\{C^1(t), C^2(t), \cdots, C^6(t)\}$ to generate 15 individuals.

Mutation 1. Randomly choose 12 individuals from $\{C^i(t), i = 1, \cdots, 20\}$, and randomly permute $\{C^j(t), j = i_1, \cdots, i_{12}\}$'s n coordinates to generate 12 new individuals.

Mutation 2. Taking the 3 best individuals $\{C^j(t), j = i_1, i_2, i_3\}$ from $\{C^i(t), i = 1, \cdots, 20\}$. Generating random vectors $\{M^i = [M^{i_1}, \cdots, M^{i_n}], i = 1, \cdots, 18\}$, where $\sum_{k=1}^{n} M^{i_k} = 0$, then generating 18 new individuals by adding a random vector such as M^i to one of the $C^i(t)$ so that $C^i(t+1) = C^i(t) + M^i$.

Keep the best 3 individuals from $\{C^i(t), i = 1, \cdots, 20\}$ into next generation.

Immigrate 12 individuals, i.e., introduce 12 new individuals that are feasible.

Now we get a new population: $\{C^1, \cdots, C^{60}\}$, where 15 come from Step 3, 12 from mutation 1, 18 from Mutation 2, 3 best individuals from the original 20, and 12 newly introduced.

Step 4:

From the population of the new 60 select 20 best individuals to make up the new generation $\{C^i(t+1), i = 1, \cdots, 20\}$.

Set $t = t + 1$. If stopping criteria is not met, go back to step 3.

6. **ANALYSIS OF RESULTS**

In our experiment, we will take $n = 3$, which means cooperative differential game involving three players. We will adopt the following coefficients:

$$\delta_1 = 20; \delta_2 = 18; \delta_3 = 21;$$
$$a_1 = 0.008; a_2 = 0.010; a_3 = 0.007;$$
$$\kappa = 0.01; \rho = .6; p = 5; r = .1;$$

Looking at $\delta_\mathbf{i}, \mathbf{i} = \mathbf{1}, \mathbf{2}, \mathbf{3}$, we can see that company 2's cost for control/advertisment is lowest. By looking at $\mathbf{a_i}, \mathbf{i} = \mathbf{1}, \mathbf{2}, \mathbf{3}$, we can see that company 2's control/advertisment has the most attractiveness to the customers. So we can say that company 2 is the 'biggest' one.

We solve our problem using Algorithm 5.1 above. We run the program twice with different stopping criteria. The stopping criteria in the first case is: *Maximum number of iterations* $= 10^2$ *and difference of objective values in two consecutive iterations* $\leq 10^{-9}$. The stopping criteria in the second case is: *Maximum of iterations* $= 10^3$ *and difference of objective value in two consecutive iterations* $\leq 10^{-12}$. The following are the results and analysis.

Running time in the two cases are

$$2.83 * 10^3 \approx 48mins, \quad 3.25 * 10^4 \approx 9.05hours$$

respectively. In the second case, we use stricter criteria.

Optimal C vector in two cases are

	c_1	c_2	c_3
Case 1	0.02329017280686	0.96422023625082	0.01248959094232
Case 2	0.02213275535129	0.97438357690981	0.00348366773889

Table 1. Optimal **C** from different stopping criteria

The manner of decrease of the objective value J_0 can be found in *Figure 4*. Our problem is a minimization problem, and from the figure, we can see that the objective value is always non-increasing, because in our evolution algorithm, we always move the best individual directly into the next generation.

Optimal state trajectories can be found in *Figure 5*. The sale of the product still decreases to zero as time goes. However, from *Figure 9*, where we compare the state trajectories between cooperative and noncooperative differential games, in the same setting, we can find something interesting. For the 'biggest' company, the rate of decrease of sale's rate in the cooperative case is slower than that in the non-cooperative case. However, in the case of the other two companies, the rates of their sales' rates decrease faster in the cooperative case. The phenomenon will be explained in the analysis of control trajectories.

Optimal control trajectories can be found in *Figure 6*. First let us analyze the problem intuitively without computation. Company 2 is the 'strongest' company, whose cost of control is lowest, and control's effectiveness is biggest. Thus, if you were in the group with the 'strongest' player, what will you do? It is logical to let the 'strongest' player use his control as much as possible. From our computational result, we can see this situation does happen. Company 2's control is bigger than that in the noncooperative case, and bigger than zero all the time (*Figure 7*). However, company 1's and company 3's controls assume negative values in some interval. The negative values of the controls in our marketing model means that company 1 and company 3 will not invest in advertising, but increase the price of product in some sense. So this can explain why the state trajectories of company 1 and company 3 will decrease faster in the cooperative case. The faster-decrease of sale will not hurt them but benefit them, because of the higher price of the product. This situation will happen only in cooperative case. And it can be explained and found in reality. In the market of oligarchy, the lone strongest company will adopt any suitable control to maximize its own profit without considering the interest of customers. Our cooperative game model can explain this rational of people. That is, the strongest company may consider itself to be a single company formed by combining the three companies in the best possible proportion for maximum profit. If (u_1, u_2, u_3) is the corresponding control, then the strongest company, which is 2, in the present case, will use strategy u_2. In this way he may decrease his cost of advertising.

Optimality of the C vector. Sometimes it is difficult to determine whether a solution of evolutionary algorithm is optimal or not. In our case, we can at least say the $\mathbf{C}$ we have found is 'almost' optimal solution. Based on the above analysis and common sense, we can expect that the more efficient company will have bigger value of $\mathbf{c_i}$. In the above optimal $\mathbf{C}$, we get $\mathbf{c_2} \gg \mathbf{c_1} > \mathbf{c_2}$, which means in the convex combination of objective functions of three companies, company 2 takes the biggest part of the whole objective function. So we can say this $\mathbf{C}$ is approaching the true optimal solution.

Other Observation relating to noncooperative game. Here we'd like to compare results in cooperative case with that of noncooperative case. We set up a noncooperative differential game model based on the same assumptions as in the previous section. Then, each company has the following problem:

$$\begin{cases} \min_{u_i} J_i = & \int_0^T e^{-rt}[\frac{\delta_i}{2}u_i^2(t) - px_i(t)]dt \\ s.t. & \\ \dot{x}_1(t) = & a_1u_1(t) - (f+G)x_1(t) \\ \vdots & \\ \dot{x}_i(t) = & a_iu_i(t) - (f+G)x_i(t) \\ \vdots & \\ \dot{x}_n(t) = & a_nu_n(t) - (f+G)x_n(t) \\ & \\ x_i(0) = & x_{i0}, \quad i = 1, \cdots, n-1 \end{cases}$$

Suppose $\mathbf{J_i^*}, \mathbf{i} = \mathbf{1}, \cdots, \mathbf{n}$ are optimal objective values for the above noncooperative differential game model, and the optimization is based on *Nash Equilibrium*.
Suppose that $[\mathbf{c_1^*}, \mathbf{c_2^*}, \cdots, \mathbf{c_n^*}], \mathbf{u_1^*}, \mathbf{u_2^*}, \cdots, \mathbf{u_n^*}, \mathbf{x_1^*}, \mathbf{x_2^*}, \cdots, \mathbf{x_n^*}$ constitute optimal solution for our cooperativedifferential game model above. From the numerical results, we have the relation:

$$J_0^* = \sum_{i=1}^n c_i^* \int_0^T e^{-rt}[\frac{\delta_i}{2}u_i^{*2}(t) - px_i^*(t)]dt < \sum_{i=1}^n c_i^* \cdot J_i^*$$

That is, optimal objective value of cooperative differential game is always less than the same convex combination of the solution of the noncooperative differential game.

Efficiency of evolutionary computation. We have mentioned that one of the criteria to evaluate efficiency of an evolutionary algorithm is to see how fast its fittest individual is replaced. Let $\mathbf{r_e} = \frac{\textbf{number of times of a most fit individual is replaced}}{\textbf{number of iterations}}$.
In *Figure 10*, we have the graph of $J_0(i) - J_0(i-1)$. In our evolutionary algorithm, $r_e \approx 1$. This is because we have used two kinds of mutation operator, which do search locally and globally separately. The first one is doing local perturbation, by which probabilistically we can always decrease the objective value in local search as the objective function and state equations are smooth locally.

7. CONCLUSION

Cooperation is a fact of life in modern economy. In this paper, we analyzed cooperative differential game model in the final stage of product life cycle. Evolutionary algorithm was used for the necessary numerical computation. In the literature, usually researchers use analytical methods to solve cooperative differential game models

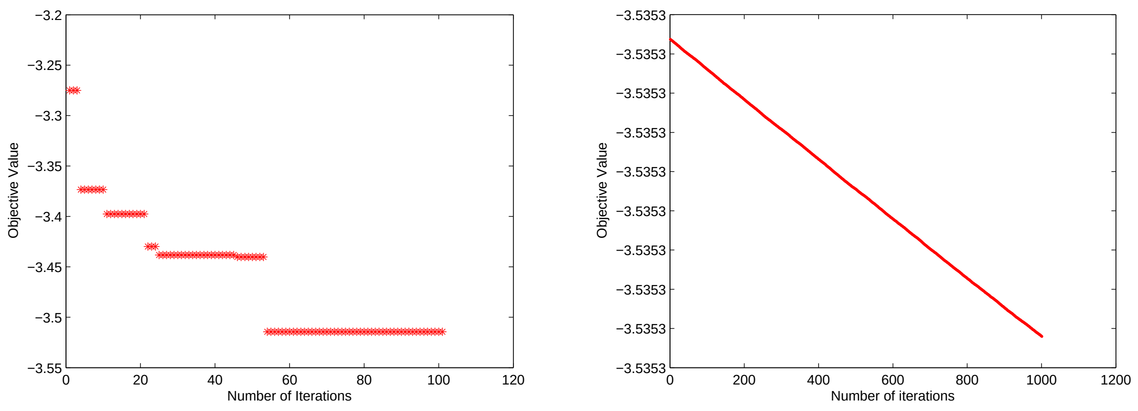

FIGURE 4. The search process for optimal objective value

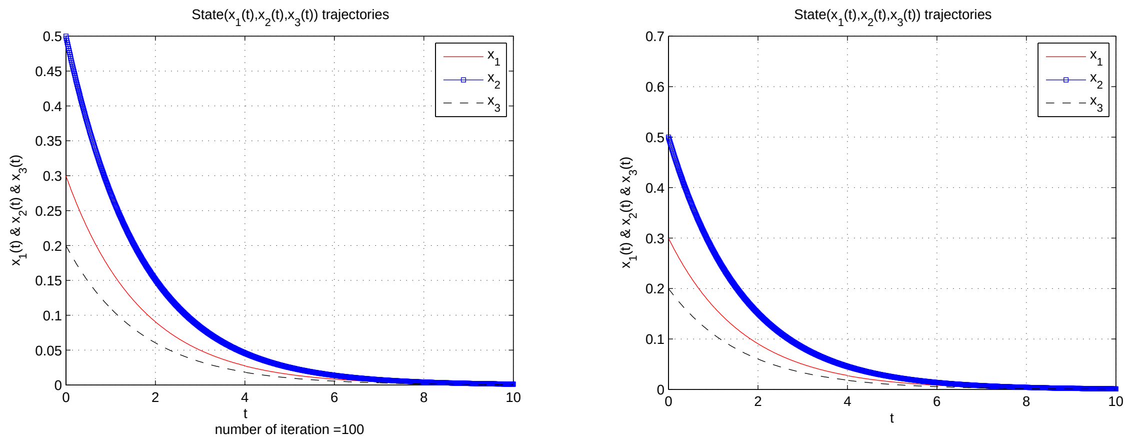

FIGURE 5. Optimal state trajectories

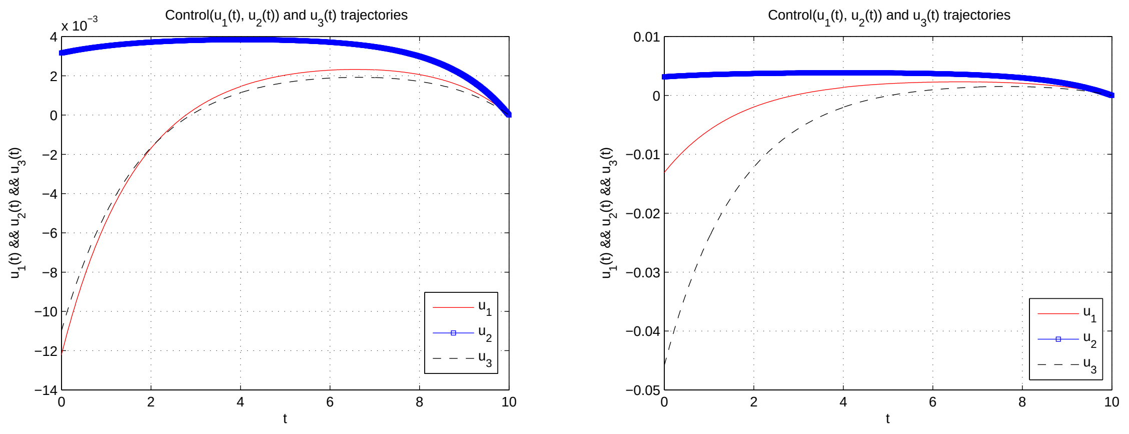

FIGURE 6. Optimal control trajectories

because of the simplicity of their models. There is no efficient way to solve cooperative differential game if the model is complicated.

Evolutionary computation is an efficient optimization computation method modelled on the biological evolutionary processes. The nature of evolutionary process in

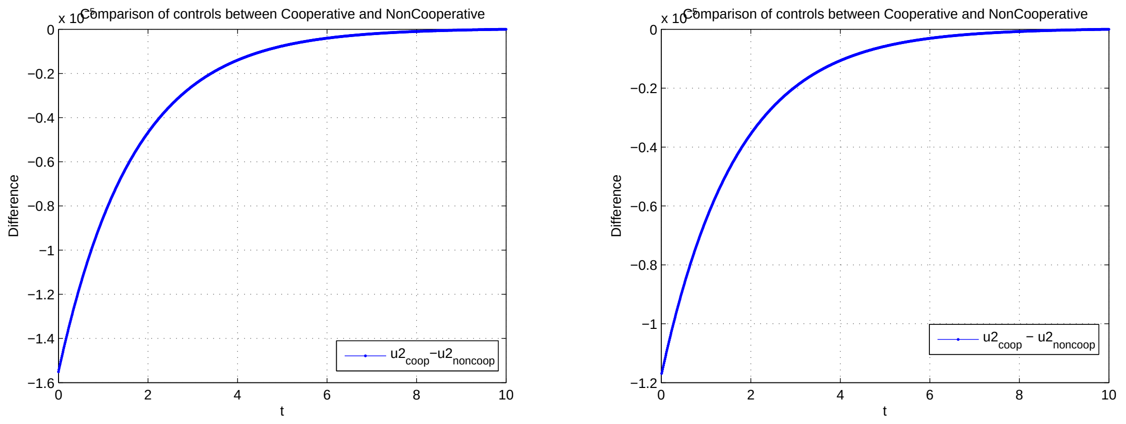

FIGURE 7. Comparison of control(Company 2)

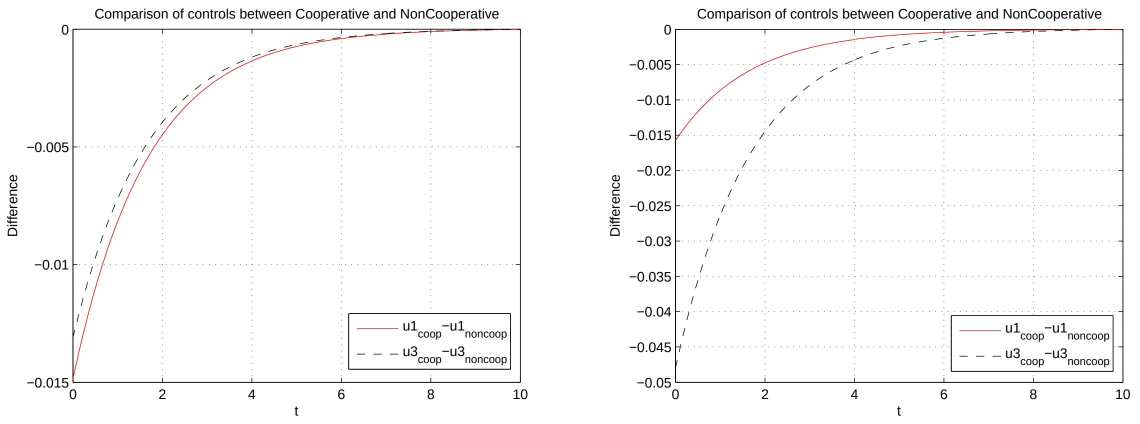

FIGURE 8. Comparison of control(Company 1,3)

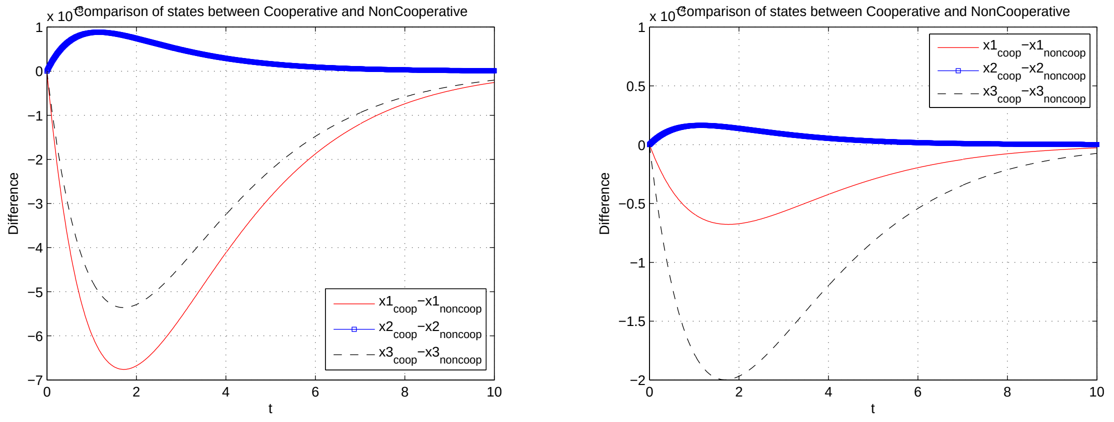

FIGURE 9. Comparison of state

biological system is the process of choosing the 'best' to live and continue on. So we mimic this biological process in numerical optimization. There are many kinds of evolutionary computation. And their difference comes from the differences of the

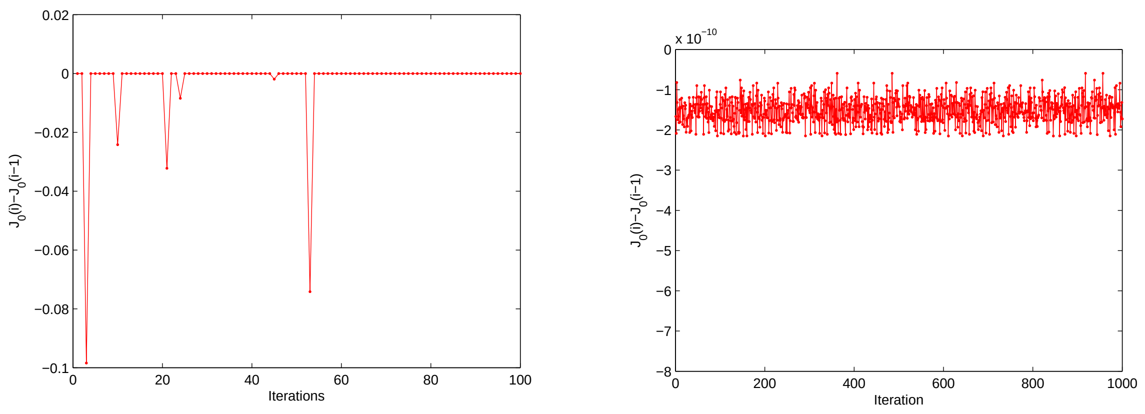

FIGURE 10. Changing of objective value

operators. The key factors to employ in evolutionary algorithm are: 1) choosing the right operators for the specific problem. 2) efficiency of fitness function to evaluate individuals. The characteristics of our problem are: 1) it is a constrained optimization problem, which requires that we should choose some special mutation and recombination operator. 2) There is no explicit formula for the functional to be optimized. This functional is complicated. Thus, we cannot use routine optimization technique based on gradient methods. Evolutionary computation is the only way out. By choosing appropriate operators and using algorithm in the **Appendix**, which solves the optimal control problem efficiently, we set up Algorithm 5.1 to solve the cooperative differential game efficiently. And it is our view that our's is the most general algorithm to solve cooperative differential game in current literature.

By exploiting the specific cooperative differential game model in marketing, we found general rules for the game.

1. In a fixed group, cooperation is always more beneficial than noncooperation for the whole group, and then more beneficial to each player based on *rational axiom*. This is common sense in our real experience, and is shown by our model.
2. Choice of cooperative or noncooperative game is more difficult in solving a specific game. This is because, in reality, there are many factors we should consider before we choose to play a specific kind of game. These considerations are
   a) How many players will be involved in the game? Are they the same kind of players? For example, in our model if we take the customer as a player, then we have a zero-sum game. It will not be helpful to play cooperative game in this case.
   b) The different status between different players as a factor: For the 'strong' player, it may not always be beneficial to play cooperative game. Leader-follower game is better than cooperative game for the leader.
   c) In reality, in the process of playing a game it is not unusual that some players deviate from one type of game to another. For example, a 'strong' player may change from cooperative game to a leader-follower game. Or, some players may cooperate to defend against others. Thus, how to keep all players in the same kind of game is an important issue in reality. It is true that all players may not want to stay with a certain type of game if they perceive it is suboptimal to

do so. Thus, they may opt to play noncooperative game instead of cooperative game.
d) How long will the game last? This will affect the choice of different types of game too.

Of course there may be many other factors that determine or affect the choice of a game. We propose define all these factors as the 'scope' of the game. That is, it is important to know how many players are involved and their status. The choice of a game is a dynamic process determined by the 'scope' of the game. When we play a game in reality, figuring out or choosing the 'scope' of the game is more complicated and is a key issue to success in the game. We may need to open another research topic on this.

8. APPENDIX

The following algorithm is used to solve the BVPs in section 2, then solve the optimal control problem.
Algorithm8.1 :

```
1. Guessing $x_i(T)^{(0)}, i = 1, \cdots, n$, and using the result from steepest descent algorithm.
2. Solving ODE backward using $RK4$(Runge-Kutta) by $x_i(T)^{(0)}, m_{ij}(T)$
3. For $i = 1, \cdots, n$
   generate $\Delta x_i(T)$ randomly from Uniform Distribution in $(-\epsilon, +\epsilon)$.
   Use $[x_1(T)^{(0)}, \cdots, x_i(T)^{(0)} + \Delta x_i(T), \cdots, x_n(T)^{(0)}, m_{ij}(T)]$ to solve the ODE backward.
   Calculate $\Delta F_j$, and $\frac{\Delta F_j}{\Delta x_i(T)}$, for $j = 1, \cdots, n$
   End. Then, proceed to get $J_0$
4. Solve $J_0 \cdot y_0 = F(X^0(T))$ for $y_0$.
5. Update $x_i(T)^{(0)}$ by $X^1(T) = X^0(T) - y_0$, and let $k = 1$
6. While $\| F_j \| > \varepsilon$ for $j = 1, \cdots, n$ and $k \leq MaxIteration$, proceed as follows
   For $i = 1, \cdots, n$
   Generating $\Delta x_i(T)$ randomly from Uniform Distribution in $(-\epsilon, +\epsilon)$.
   Using $[x_1(T)^{(k)}, \cdots, x_i(T)^{(k)} + \Delta x_i(T), \cdots, x_n(T)^{(k)}, m_{ij}(T)]$ to solve ODE backward.
   Calculate $\Delta F_j$, and $\frac{\Delta F_j}{\Delta x_i(T)}$, for $j = 1, \cdots, n$
   End. Then, get $J_k$.
   Solve $J_k \cdot y_k = F(X^k(T))$ for $y_k$
   Update $x_i(T)^{(k)}$ by $X^{k+1}(T) = X^k(T) - y_k$
   Using $[x_1(T)^{(k+1)}, \cdots, x_i(T)^{(k+1)}, \cdots, x_n(T)^{(k+1)}, m_{ij}(T)]$ to solve ODE backward.
   Calculate $F_j(X(T)^{(k+1)})$
   End of While statement.
7. Calculate $u_i(t)$ and $J_i$
```

REFERENCES

[1] Leitmann G., *Cooperative and Non-cooperative Many Players Differential Games.*, Springer-Verlag, New York, 1974.

[2] Medhin, N. G., Wan Wei, *Multi-New Product Competition in Duopoly: A Differential Game Analysis.*

[3] Medhin, N. G., Wan Wei, *Multi-New Product Competition in Duopoly: A Differential Game Analysis.*

[4] Medhin, N. G., Wan Wei, *Competition in the last stage of Product Life-cycle.*

[5] Medhin, N. G., Wan Wei, *Leader-Follower Games in Marketing: A Differential Game Approach.*

[6] FitzPatrick,J. M.,Grefensette, J. J. *Genetic Algorithms in noisy environments*, Machine Learning, Vol.3, pp. 101–120, 1988.

[7] Goldberg D.E., Del.K.*A comparative analysis of selection schemes used in genetic algorithms*, in Foundations of Genetic Algorithms, G. J. E. Rawlins, Ed., Morgan Kaufman, 1991.

[8] Holland, J. H., *Adaptation in natural and artificial systems*, Ann Arbor, MI: University of Michigan Press, 1975.

[9] Jorgensen S., G. Martin-Herran and G. Zaccour, *Agreeability and Time-consistency in Linear-State Differential Games.*, Journal of optimization theory and applications, Vol.119, No. 1, pp. 49–63, 2003.

[10] Reed J., R. Toombs, and N.A. Barricelli. *Simulation of biological evolution and machine learning. i. selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing.* Journal of Theoretical Biology, 17, 1967, pp. 319–342.

[11] Schwefe H.P., *Evolution and Optimum Seeking*, John Wiley, 1995.

[12] Stalford H.L., *Criteria for Pareto-Optimality in Cooperative Differential Games*, Journal of Optimization Theory and Applications. Vol.9, No. 6, 1972

[13] Steffen Jorgensen, Georges Zaccour, *Differential Games in Marketing*, Kluwer Academic Publishers, 2004

[14] Thomas Back, David B Fogel, and Zbigniew Michalewicz*Evolutionary Computation: Basic Algorithm and Operators*, Institute of Physics Publishing, Bristol and Philadelphia, 2000

[15] Rainer Storn, Kenneth Price, *Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces*, 1995