Available online at www.sciencedirect.com
SCIENCE DIRECT•
ELSEVIER Computers and Mathematics with Applications 49 (2005) 1655–1668
An International Journal
computers & mathematics with applications
www elsevier com/locate/camwa

# An Application of Swarm Optimization to Nonlinear Programming

YING DONG, JIAFU TANG*, BAODONG XU AND DINGWEI WANG
Department of Systems Engineering
Key Laboratory of Process Industrial Automation of MOE
School of Information Science and Engineering
Northeastern University
Shenyang 110004, P.R. China
dy_neu@sina.com jftang@mail.neu.edu.cn

*(Received March 2004; revised and accepted February 2005)*

**Abstract**—Particle swarm optimization (PSO) is an optimization technique based on population, which has similarities to other evolutionary algorithms It is initialized with a population of random solutions and searches for optima by updating generations Particle swarm optimization has become the hotspot of evolutionary computation because of its excellent performance and simple implementation. After introducing the basic principle of the PSO, a particle swarm optimization algorithm embedded with constraint fitness priority-based ranking method is proposed in this paper to solve nonlinear programming problem By designing the fitness function and constraints-handling method, the proposed PSO can evolve with a dynamic neighborhood and varied inertia weighted value to find the global optimum The results from this preliminary investigation are quite promising and show that this algorithm is reliable and applicable to almost all of the problems in multiple-dimensional, nonlinear and complex constrained programming. It is proved to be efficient and robust by testing some example and benchmarks of the constrained nonlinear programming problems. © 2005 Elsevier Ltd. All rights reserved

**Keywords**—Particle swarm optimization, Nonlinear programming, Global optimization, Evolutionary algorithm, Priority-based ranking.

## 1. INTRODUCTION

Nonlinear programming (NLP) is a mathematical programming technique where the objective function is nonlinear, or one or more of the constraints have nonlinear relationship or both. It is paid attention in past years as an important branch of operations research and has wide applications in the areas of military, economics, engineering optimization, and management science [1]. There are many traditional methods in the literature for solving nonlinear programming problems. However, most of them may solve NLP only on an approximate basis and assume

This paper is jointly financial supported by the National Natural Science Foundation (70471028, 70301007, 70431003) of the People's Republic of China together with the Excellent Youth Teacher Program of Ministry of Education (MOE) (2002-350), Program for New Century Excellent Talents in University (NCET-04-0280) of MOE, Key Project of Science and Technology of MOE (104064), Key Laboratory of Process Industrial Automation of MOE in NEU and Liaoning Provincial Natural Science Foundation (20022019)

that goal and constraints are differentiable [2]. Recently, based on strict optimization theory and algorithms, many researchers have proposed some new stochastic optimization methods and intelligent algorithm, such as the genetic algorithm (GA) [3–5], analog neural networks [6], chaos optimization algorithm [7], ant colony algorithm [8], line-up competition algorithm [9], and various hybrid methods [10–12]. However, each method has its own suitable application scope and constraints condition. So far, there is not a method for determining the global optimal solution to the general nonlinear programming problem.

Recently, a global optimization algorithm of population-based search, called particle swarm optimization (PSO) was proposed [13,14]. PSO is a kind of random search algorithm that simulates natural evolutionary process and performs good characteristic in solving some difficult optimization problems and hence, received many attentions since its origination. The PSO has been successfully applied to a wide range of applications, such as optimization problem, traveling salesman problem, job scheduling, etc. Because PSO has very deep intelligent background, it is suitable for science computation and engineering applications.

One of the important factors that particle swarm optimization is attractive is simple that there are very few parameters to adjust. It can achieve the optimal or near-optimal solutions in a rather short time without enormous iterative computations in digital implementation. A thorough mathematical foundation for the methodology was not developed so far with the step of the algorithm; however, it has been proven to be very effective for application.

The particle swarm optimization has been found to be robust and fast in solving nonlinear, non-differentiable, multimodal problems, but it is still in its infancy, particularly its search rates are commonly lower and sometimes bring more computation when solving some difficult optimization problems. A lot of work and research are needed. This paper focuses on the application of PSO to a type of nonlinear programming problem. When applying PSO to NLP, how to configure the fitness function and formulate and evaluate the infeasible particle as well as construct search schemes are important issues. By introducing the concept of constraint fitness and objective fitness, a constraint fitness priority-based ranking method is developed to evaluate the particles during the search. To speed the search processes and overcome the disadvantages of the global and local search, a dynamic neighborhood operator is proposed. By embedding the constraint fitness priority-based ranking method and dynamic neighborhood operator, a special PSO is developed to solve nonlinear programming problems. The superior performance of the proposed PSO will be demonstrated by solving several NLP testing problems.

The rest of this paper is organized as follows. Section 2 explains the basic idea and overall procedure of basic PSO. To apply the framework of basic PSO to solve NLP, a constraint fitness priority-based ranking method and dynamic neighborhood operator are developed in Section 3. The overall scheme of the proposed particle swarm optimization for solving NLP is presented in Section 4. Finally, simulation results and analysis of some examples and the conclusions are given in Sections 5 and 6, respectively.

## 2. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Kennedy and Eberhart in 1995 [13,14]. It exhibits common evolutionary computation attributes including:

(1) it is initialized with a population of random solutions,
(2) it searches for optima by updating generations, and
(3) potential solutions, called particles, are then "flown" through the problem space by following the current optimum particles.

The particle swarm concept originated as a simulation of a simplified social system. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. The authors use the term *swarm* in accordance with a paper by Millonas [11], who developed his models for applications in artificial life, and articulated five basic principles of swarm intelligence. The term *particle* was selected as a compromise. It could be argued that the population members are massless and volumeless, and thus could be called "points", it is felt that velocities and accelerations are more appropriately applied to particles, even if each is defined to have arbitrarily small mass and volume [13].

Each particle keeps track of its coordinates in the problem space, which are associated with the best solution (fitness) it has achieved so far. This value is called *pBest*. Another "best" value that is tracked by the *global* version of the particle swarm optimization is the overall best value, and its location obtained so far by any particle in the population. This location is called *gBest*.

The particle swarm optimization concept consists of, at each step, changing the velocity (accelerating) each particle toward its *pBest* and *gBest* locations (global version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pBest* and *gBest* locations.

The updates of the particles are accomplished according to the following equations. Equation (1a) calculates a new velocity for each particle (potential solution) based on its previous velocity ($v_{id}$), the best location it has been achieved ($p_{id}$ or *pBest*) so far, and the global best location ($p_{gd}$ or *gBest*) the population has been achieved. Equation (1b) updates individual particle's position in solution hyperspace. The two random numbers $c_1$ and $c_2$ in (1a) are independently generated in the range [0,1]. The use of the inertia weight $w$ provides improved performance in a number of applications [15],

$$v_{id} = w * v_{id} + c_1 * \text{rand}() * (p_{id} - x_{id}) + c_2 * \text{Rand}() * (p_{gd} - x_{id})\,, \tag{1a}$$

$$x_{id} = x_{id} + v_{id} \tag{1b}$$

The acceleration constants $c_1$ and $c_2$ in equation (1a) represent the weighting of the stochastic acceleration terms that pull each particle towards *pBest* and *gBest* positions. Thus, adjustment of these constants changes the amount of "tension" in the system. Low values of them allow particles to roam far from target regions before being tugged back, while high value results in abrupt movement toward, or past through target regions.

Particle's velocities on each dimension are confined to a maximum velocity $v_{\max}$ which is a parameter specified by the user. If the sum of accelerations would cause the velocity on that dimension to exceed $v_{\max}$, then the velocity on that dimension is limited to $v_{\max}$.

There are two versions of the PSO: one is local version and another is global version. With local version, particles only have information of their own and their neighbors' bests, rather than that of the entire group. Instead of moving toward a kind of stochastic average of *pBest* and *gBest* (the best location of the entire group), particles move toward points defined by *pBest* and *lBest* which is the index of the particle with the best evaluation in the particle's *neighborhood*,

$$v_{id} = w * v_{id} + c_1 * \text{rand}() * (p_{id} - x_{id}) + c_2 * \text{Rand}() * (p_{ld} - x_{id})\,. \tag{1c}$$

Global version is faster but might easily converge to trap into local optimum. Taking into account the characteristic of these two versions, one can use global version to get quick result in the first phase of search and use local version to refine the search finally.

From the above discussion, one can learn that there are two key steps when applying PSO to optimization problems, i.e., the representation of the solution and the fitness function configuration. Unlike GA, PSO doesn't need complex encoding and decoding processes and special genetic operators, it take real numbers as particles in the aspects of representation solution.

From the procedure, one can learn that PSO shares many common points with GA, for example, they start with a group of a randomly generated population, evaluate the population with fitness values, update the population and search for the optimum with random techniques, and do not guarantee optimal.

Distinguished from GA, PSO does not need genetic operators like crossover and mutation. Particles update themselves with internal velocity. They also have memory, which is important to the algorithm. Compared with genetic algorithm (GAs), the information sharing mechanism in PSO is significantly different. In GAs, chromosomes share information with each other, so the whole population moves like a one group towards an optimal area. In PSO, only *gBest* (or *lBest*) gives out the information to others, hence, it is a one-way information sharing mechanism. The evolution only looks for the best solution. Compared with GA, all the particles tend to converge to the best solution quickly even in the local version in most cases [16].

The main disadvantage of the above basic PSO is that it is difficult to keep the diversity of population, and to balance local and global search and hence it may result in local optimal solutions. Besides, their search rates are commonly lower and sometimes need more computation when solving some difficult optimization problems. A modified method is proposed in this paper to deal with the above disadvantages of PSO in solving nonlinear programming problems.

## 3. CONSTRAINT FITNESS PRIORITY-BASED RANKING METHOD

### 3.1. Canonical Form of NLP

NLP problems with $n$ variables and $m$ constraints may be written as the following canonical form,

$$\begin{aligned} \max \quad & f(\mathrm{x}) = f(x_1, x_2, \dots, x_n), \\ \text{s.t.} \quad & g_i(\mathrm{x}) \le 0, \qquad i = 1, 2, \dots, m_1, \\ & h_i(\mathrm{x}) = 0, \qquad i = m_1 + 1, m_1 + 2, \dots, m. \end{aligned} \tag{2}$$

### 3.2. Formulation and Evaluation of Infeasible Particles

The handling with the system constraints, especially the measurement and evaluation of illegal particle are key techniques with PSO in solving NLP. In the implementation process, particles randomly generated at the beginning and/or generated by internal velocity during the evolutionary process usually violate the system constraints resulting in infeasible particles. Therefore, the handling of system constraints, particularly nonlinear equation constraints, and the measurement and evaluation of infeasible particles are of research interest. Currently, several methods have been developed to deal with system constraints. These methods mainly consider preserving feasibility of solutions, penalty strategies and searching for feasible solutions, and they have several drawbacks. To cope with constrained problems with evolutionary computation, four different approaches can be adopted,

(1) rejection of infeasible individuals,
(2) repair of infeasible individuals,
(3) replacement of individuals by their repaired versions, and
(4) penalty function methods.

Among of them, the penalty function methods are particularly promising, as evidenced by recent developments [2]. Based on the above analysis in this paper, a constraint fitness priority-based ranking method is constructed to solve system constraints.

Fitness function stands for particle's fitness level to environment. In this section, two kinds of fitness function are introduced, one is optimal fitness function $F_{\mathrm{obj}}(\mathrm{x})$ for objective function, and the other is constraint fitness function $F_{\mathrm{con}}(\mathrm{x})$ for constraints. $F_{\mathrm{obj}}(\mathrm{x})$ and $F_{\mathrm{con}}(\mathrm{x})$ are defined as follows.

DEFINITION 1. *The function $F_{\mathrm{obj}}(\mathrm{x})$ is defined as objective fitness function at point $x$,*

$$F_{\mathrm{obj}}(\mathrm{x}) = f(\mathrm{x}). \tag{3}$$

DEFINITION 2. *The constraint fitness function $F_i(\mathrm{x})$ is defined to be the fitness level of point* x *to the constrained condition (i).*

The following two methods are suggested for evaluating the infeasible particles according to different forms of constrained condition.

For inequality constraint $g_i(\mathrm{x}) \le 0$,

$$F_i(\mathrm{x}) = \begin{cases} 1, & g_i(\mathrm{x}) \le 0, \\ 1 - \dfrac{g_i(\mathrm{x})}{g_{\max}(\mathrm{x})}, & g_i(\mathrm{x}) > 0, \end{cases} \tag{4}$$

where $g_{\max}(\mathrm{x}) = \max\{g_i(\mathrm{x}),\ i = 1, 2, \dots, m_1$.

For equality constraint $h_i(\mathrm{x}) = 0$,

$$F_i(\mathrm{x}) = \begin{cases} 1, & h_i(\mathrm{x}) = 0, \\ 1 - \dfrac{|h_i(\mathrm{x})|}{h_{\max}(x)}, & h_i(\mathrm{x}) \ne 0, \end{cases} \tag{5}$$

where $h_{\max}(\mathrm{x}) = \max\ \{h_i(\mathrm{x}),\ i = m_1 + 1, m_1 + 2, \dots, m\}$.

DEFINITION 3. *The function $F_{\mathrm{con}}(\mathrm{x})$ is defined as total constraint fitness function at point* x.

Based on the above definition, taking into account the equality constraints, the weighted constraint function, denoted by $F_{\mathrm{con}}(\mathrm{x})$, can be constructed as follows,

$$F_{\mathrm{con}}(\mathrm{x}) = \sum_{i=1}^{m} w_i F_i(\mathrm{x}), \quad \sum_{i=1}^{m} w_i = 1, \qquad 0 \le w_i \le 1, \quad \forall i. \tag{6}$$

Each $w_i$, is the weight for constraint $i$, which can be generated randomly in order to assure the particle's diversity. The function reveals the relationship between the point x and the feasible domain $Q$; if $F_{\mathrm{con}}(x) = 1$, it indicates that $\mathrm{x} \in Q$. On the contrary, if $0 < F_{\mathrm{con}}(\mathrm{x}) < 1$, and the smaller the $F_{\mathrm{con}}(\mathrm{x})$ is, the performance of x "belonging to" $Q$ becomes worse, i.e., x is further away from the feasible domain $Q$. Hence, the total constraint fitness function represents the fitness level of point x to the feasible domain $Q$

### 3.2.1. Constrained-condition's handling

In light of the idea of PSO, particle flies through the problem space by following the current optimum solution with the best fitness value. So, the basic idea for handling constraints may be described as follows. First, all particles are ranked according to their fitness value. During the ranking process, the highest constraint fitness value according to (4) is in the first position. If two particles have the same constraint fitness values then their objective fitness value are compared. The one with better objective fitness value is in prior to the other one.

In comparison with common penalty function, the advantage of this method is that the fitness values of the feasible particles are always better than infeasible ones. So, we can obtain the feasible particles rather easily in the iteration process. Using evolutionary operation, the optimal feasible particle is achieved based on feasible particles and better infeasible particles. In this way, feasible area and optimal solution are combined together while the optimal objective function need not required to be greater than zero. Another advantage of the method is that there is no need to adjust weighted factor between constraint and objective fitness. So, it is easier to apply to real world problems.

From the above discussion, one knows that there are two levels of competition in the algorithm. One is the survival competition inside constraints. The best one survives in every generation. The other one is the competition between constraints and objective. According to the values of their objective function and constraint fitness value, all particles are ranked to form a line-up. The best particle is in the first position in the line-up, while the worst in the final position. The value of fitness function of the first position is updated continually. As a result, the optimal solution is approached rapidly during the implementation. So, the algorithm can be called competition PSO (referred CPSO hereafter) in this paper.

### 3.2.2. Dynamic neighborhood operator

Since both global version and local version have their own advantages and disadvantage respectively, one can use them both in the algorithm while global version is used to get quick result and local version can refine the search space.

From the definition of the fitness function, one can learn that there must have better particles in the neighbor of the good fitness value. Hence, a dynamic neighborhood is presented. In each generation, after calculating distances to every other particle, each particle finds its new neighbors. Among the new neighbors, each particle finds the local best particle as the *lBest*. At the beginning of the generation, a particle's neighborhood is defined as its own, i.e., neighborhood includes only one particle. As the generation increases, neighborhood scope will grow larger and larger and at the end will contain the whole feasible domain. The problem is how to define the distance and how to define the local best particle.

DEFINITION 4. *dist*$[l]$ *stands for the distance of the current particle from the* $l^{th}$ *particle in the fitness value space of the constraint function.*

The algorithm used to search for local optima in each generation is defined as follows.

(1) Calculate the distances dist$[l]$.
(2) Find the nearest $k$ particles as the neighbors of the current particle based on the distances calculated above. ($k$, the neighborhood size.)
(3) Find the local optima among the neighbors in terms of the fitness value of the objective function.

## 4. OVERALL SCHEME OF CPSO FOR NONLINEAR PROGRAMMING

The basic idea of CPSO may be described as follows. First, randomly produce an initial population with the size of *popsize* particles. The information of infeasible particle is embedded into the fitness function in order to measure the degree to which the infeasible particles are away from the feasible domain so that they could not be rejected in later generation processes. Hence, the CPSO ensures the optimum to be obtained from both sides of the feasible and infeasible domains. In the process of iteration, for a particle $x_i \notin Q$, give it a less fitness value so that it may have less chances than others to be selected as the *pBest* or *lBest* in the later generations. As the generation increases, the individuals with less fitness values die out gradually, namely, the individuals $x_i \in Q$ with less value and individuals $x_i \notin Q$ die out gradually, and the particles maintained in the population are the particles with a high value of objective function. After a number of generations, the particle's objective function values reach the optimal or near optimal from the two sides of feasible domain.

### 4.1. PSO Setting

Randomly produce an initial population with a definite number $m$ including velocity and position. This might assure the solution's diversity since it is randomly generalized.

The typical range is 20–40 for the number of particles. Actually, for most of the problems ten particles is large enough to get good results. For some difficult or special problems, one can try 100 or 200 particles as well.

Dimension of particles and range of particles are all determined by the problem to be optimized. One should specify different ranges for different dimension of particles.

Inertia weight controls the impact of previous historical values of particle velocity on its current one. A larger inertia weight pressures towards global exploration (searching new area) while a smaller inertia weight pressures toward fine-tuning the current search area. Without the first part in (1a), then the "flying" particles' velocities are only determined by their current positions and their best positions in history. The velocities itself is memoryless. Suitable selection of the inertia weight provides a balance between global and local exploration and exploitation, and results in less iteration on average to find an optimal solution. At the beginning of the search process, a larger inertia weight can be used and decreases step by step as the fitness value increases so as to improve the local search ability.

Hence, using the method of Shi [20], i.e., inertia weight decreased linearly from about 0.9 to 0.4 during a run,

$$f\text{InerWt} = ((f\text{INITWT} - 0.4) * (G_{\max} - G_n)/G_{\max}) + 0.4, \tag{7}$$

where $G_{\max}$ is the maximum generation prespecified by the user and $f$INITWT is defined as initial weight with the value of 0.9.

The acceleration constants $c_1$ and $c_2$ in equation (1a) represent the weighting of the stochastic acceleration terms that pull each particle toward *pBest* and *gBest* positions. Thus, adjustment of these constants changes the amount of "tension" in the system. Low values allow particles to roam far from target regions before being tugged back, while high value results in abrupt movement toward, or past, target regions.

If $c_1 = 0$, then the particles have no cognition ability, which represents the private behavior of the particle itself. It can explore the new search space under the reciprocity within the particles. If $c_2 = 0$, then there is no social share information, which represents the collaboration among the particles. It is equivalent to run $m$ particles independently at the same time for population because of no interaction between the particles. It is very hard to obtain the optimal solution.

In this paper, $c_1$ and $c_2$ are set to equal to 2. However, other settings were also used by many researchers.

Max-velocity $v_{\max}$ determines the maximum change one particle can take during a run. Therefore, $v_{\max}$ is an important parameter. It determines the resolution, or fineness, with which regions between the present position and the target (best so far) positions are searched. If $v_{\max}$ is too high, particles might fly through good solutions. If $v_{\max}$ is too small, on the other hand, particles may not explore sufficiently beyond locally good regions. In fact, in this case, they could easily be trapped in local optima and unable to move far enough to reach a better position in the problem space.

In this paper, $v_{\max}$ is set at about 10% of the dynamic range of the variable to each dimension.

According to the degree of precision required, the maximum number of iterations the PSO executes and the minimum error requirement can be adopted as stop condition

## 4.2. Overall Procedure of the CPSO

In summary, the overall procedure of the CPSO can be described by pseudo code as follows:

```
For each particle
  Initialize particle
End
Do
  For each particle
    Calculate constraint fitness value and objective fitness value
    Ranking the fitness value according to (4)
    If the fitness value is better than the best fitness value (pBest) in history
      Set current value as the new pBest
  End
  Calculate neighborhood size according to the method described in 3.2.2
  Choose the particle with the best fitness value of all the particles or neighborhood
  as the gBest or lBest
For each particle
      Calculate particle velocity according to equation (1a) or (1c)
```

```
        Update particle position according to equation (1b)
    End
While stop criteria is not satisfied
```

## 5. NUMERICAL EXAMPLES AND ANALYSIS

To demonstrate the effectiveness of the CPSO, some test examples with linear constraints, nonlinear constraint, and nonconvex constraint, particularly testing problems with large size are given in this section. The simulation analyses of the testing problems are conducted in three parts. Part 1 tries to take a comparison of the CPSO and GA on variant types of NLP problems. Parts 2 and 3 aim to demonstrate the sensitivity analysis and the performance of the CPSO on large size problems, respectively.

### 5.1. Comparison of the CPSO and GA on Some Benchmarking Problems

The test problems P1–P5 are selected as benchmarking examples for comparison [17]. Among of them, test P1 is an interval optimization problem, P2 includes inequality constraints, P3 is a multimodal problem, P4 and P5 are nonconvex linear problem and penalty hard problem respectively, while P6 is an exponential function and search process is rather complex [18]. These testing problems represent variant types of NLP problems.

$$\text{P1}: \begin{cases} \min & 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141, \\ \text{s.t.} & 0 \le 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \le 92, \\ & 90 \le 80.51249 + 0.0071371x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \le 110, \\ & 20 \le 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \le 25, \\ & 78 \le x_1 \le 102, \\ & 33 \le x_2 \le 45, \\ & 27 \le x_i \le 45, \qquad i = 3,4,5, \end{cases}$$

$$\text{P2}: \begin{cases} \min & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 \\ & + (x_5-3)^2 + 1(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7) + 45, \\ \text{s.t.} & 105 - 4x_1 - 5x_2 + 3x_7 - 4x_8 \ge 0, \\ & -3(x_1-2)^2 - 4(x_2-3)^2 - 2x_3^2 + 7x_4 + 120 \ge 0, \\ & -10x_1 + 8x_2 + 17x_7 - 2x_8 \ge 0, \\ & -2x_1^2 - 2(x_2-1)^2 + 2x_1x_2 - 14x_5 + 6x_6 \ge 0, \\ & 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \ge 0, \\ & -5x_1^2 - 8x_2 - (x_3-6)^2 + 2x_4 + 40 \ge 0, \\ & 3x_1 - 6x_2 - 12(x_9-8)^2 + 7x_{10} \ge 0, \\ & -0.5(x_1-8)^2 - 2(x_2-4)^2 - 3x_5^2 + x_6 + 30 \ge 0, \\ & -10 \le x_i \le 10, \qquad i = 1,\dots,10, \end{cases}$$

$$\text{P3} . \begin{cases} \max & \sin^3(2\pi x_1)\sin^3(2\pi x_2), \\ \text{s.t.} & x_1^2 - x_2 + 1 \le 0, \\ & 1 - x_1 + (x_2-4)^2 \le 0, \\ & 0 \le x_1 \le 10, \qquad 0 \le x_2 \le 10, \end{cases}$$

$$\mathrm{P4}:\begin{cases}\min & (x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2+10x_5^6+7x_6^2+x_7^4\\ & -4x_6x_7-10x_6-8x_7,\\ \text{s.t.} & 127-2x_1^2-3x_2^4-x_3-4x_4^2-5x_5\geq 0,\\ & 282-7x_1-3x_2-10x_3^2-x_4+x_5\geq 0,\\ & 196-23x_1-x_2^2-6x_6^2+8x_7\geq 0,\\ & -4x_1^2-x_2^2+3x_1x_2-2x_3^2-5x_6+11x_7\geq 0,\\ & -10\leq x_i\leq 10 \qquad i=1,\ldots,7,\end{cases}$$

$$\mathrm{P5}:\begin{cases}\min & x_1+x_2+x_3,\\ \text{s.t.} & 1-0.0025(x_4+x_6)\geq 0,\\ & 1-0.0025(x_5+x_7-x_4)\geq 0,\\ & 1-0.01(x_8-x_5)\geq 0,\\ & x_1x_6-833.33252x_4-100x_1+83333.333\geq 0,\\ & x_2x_7-1250x_5-x_2x_4+1250x_4\geq 0,\\ & x_3x_8-1250000-x_3x_5+2500x_5\geq 0,\\ & 100\leq x_1\leq 10000,\ 1000\leq x_i\leq 10000, \quad i=2,3, 10\leq x_i\leq 1000, \ i=4,\ldots,8,\end{cases}$$

$$\mathrm{P6}:\begin{cases}\min & -a_1\exp\left(-a_2\sqrt{\frac{1}{n}\sum_{j=1}^{2}x_j^2}\right)-\exp\left(\frac{1}{n}\sum_{j=1}^{2}\cos(a_3x_j)\right)+a_1+e,\\ \text{s.t.} & -5\leq x_j\leq 5 \qquad j=1,2.\end{cases}$$

P6 is Ackley function [19] and the optimum is 0. Constants are set as follows,

$$a_1=20, \qquad a_2=0.2, \qquad a_3=2\pi, \qquad e=2.71282.$$

The comparison results of the worst, the mean and the best solutions for the testing problems by GA and the CPSO are given in Table 1. Due to the CPU times the algorithms GAs run are neglected in literature [17–19], the last column presents the CPU time when the algorithm CPSO terminated at the best solution using IBM T42 with Pentium-M 1.5 GHz processor and 512 M memory.

CPSO—the results as Gmax=1000, popsize=50, $c_1=c_2=2$; GA—the results in Xie [17] except P6.

The results shown in Table 1 that the CPSO is always superior to GA except for Test P1 from the best solution perspective and for the Test P2 from the worst and mean solution perspective

as marked in bold style. That is to say, there are five out of six testing problems that show the CPSO is better than GA either from the best solution, mean solution or the worst solution. From the perspective of CPU time the algorithm runs, one can also find that the algorithm CPSO is effective. It can be concluded from the simulation of benchmarking problems that the CPSO is more effective at least not worse than GA.

## 5.2. Sensitivity Analysis of the CPSO

In this section, two testing examples are selected from the literatures [5,19] to demonstrate the CPSO is effective from the sensitivity perspective The testing P7 [5] and P8 [19] are given as

Table 1 Comparison results of test problems using CPSO and GA

| Testing problems | Type | Algorithm | Best solution | Worst solution | Mean solution | CPU time(s) |
|---|---|---|---|---|---|---|
| P1 | Min | CPSO | −30664.7 | −30656.1 | −30662.8 | 1 |
| | | GA | −30665.5 | −30236.6 | −30533.8 | —— |
| P2 | Min | CPSO | 24.80818 | 26.18273 | 25.40852 | 1.5 |
| | | GA | 25.237 | 26.110 | 25.237 | —— |
| P3 | Max | CPSO | 0.9999 | 0.9999 | 0.9999 | 0.8 |
| | | GA | 0.95825 | 0.95825 | 0.95825 | —— |
| P4 | Min | CPSO | 680.667 | 680.9047 | 680.7867 | 1 |
| | | GA | 680.67 | 683.74 | 681.62 | —— |
| P5 | Min | CPSO | 7114.84 | 7225.979 | 7222.389 | 0.5 |
| | | GA | 7115.00 | 7384.29 | 7795.93 | —— |
| P6 | Min | CPSO | 0.005451 | 0.005451 | 0.005451 | 0.1 |
| | | GA | 0.005456 | —— | —— | —— |

$$\text{P7}: \begin{cases} \max & f(x) = -2x_1^2 + 2x_1x_2 - 2x_2^2 + 4x_1 + 6x_2, \\ \text{s.t.} & 2x_1^2 - x_2 \le 0, \\ & x_1 + 5x_2 \le 5, \\ & x_1, x_2 \ge 0, \end{cases}$$

$$\text{P8}: \begin{cases} \max & f(x) = \dfrac{x_1^2x_2^2x_3^2}{2x_1^3x_3^2 + 3x_1^2x_2^2 + 2x_2^2x_3^3 + x_1^3x_2^2x_3^2}, \\ \text{s.t.} & x_1^2 + x_2^2 + x_3^2 \ge 1, \\ & x_1^2 + x_2^2 + x_3^2 \le 4, \\ & x_1, x_2, x_3 > 0. \end{cases}$$

Tang [5] solved the above problem using hybrid genetic algorithm combining the penalty function and gradient direction search with the best solution being (0.658,0.868) and the best objective being 6.61305 (optimal solution is 6.613086).

For testing P8, Liu [19] got a better solution using real number coding after 50 iterations. The best solution is (.8692552, .5345225, 1.313627) with objective function being 0.15373.

There are several parameters in the CPSO, among which the population size and maximum number of generations are important ones to affect the effectiveness when the algorithm is implemented. To show sensitivity of the CPSO with parameters, the simulations are conducted when the population size *popsize* are set as 10, 20, 30, and 40, and maximum number of generation *NG* as 20, 50, and 1000, respectively. Table 2 presents the simulation results of the CPSO for the testing P7 and P8 under all 12 combinatorial cases of the population size and maximum generations.

It can be seen from Table 2 that the best solution is slightly affected by the population size and is greatly affected by the maximum generation *NG* until it reaches a suitable level, such as NG = 50. At the same time compared with GA, CPSO can achieve the global optimization very rapidly. From Table 2, one can also observe that the solution is almost satisfied when *popsize* is 30, and the best solution can be obtained when iteration reaches 50 in most of the cases.

It can be observed from the simulation experiences with other values of population sizes and number of generations that, given a specified population size, the value of the best objective function is becoming better as the number of the iterations increases, and *vice versa*, given a specified generation number the objective function will be improved also, however slightly with the increase of the population sizes. The computation experiences show that, in general, 500 iterations would be enough for the testing problem since an insignificant improvement of the objective function is observed when the iteration number increases from 500 to 1000.

Table 2 The best solutıon under dıfferent populatıon sızes and maximum generatıons of CPSO

| Popsize | NG | Test P7 | | Test P8 | |
|---|---|---|---|---|---|
| | | Best solutıon | Best objectıve | Best solutıon | Best objective |
| 10 | 20 | 6550682, .8663893 | 6 60344714 | .865603, .5098256, 1.265862 | .15372290 |
| | 50 | .6588718, 8682256 | 6 61308426 | .8692601, 5344981, 1 31409 | 15372999 |
| | 1000 | .6588723, 8682255 | 6.61308528 | 8692552, 5345225, .313627 | .15373001 |
| 20 | 20 | .6587232, 8681039 | 6.61266253 | .8776503, 544467, 1 318814 | 15372804 |
| | 50 | .6588687, .8682261 | 6.61307538 | 869847, .5351846, 1 314006 | 15373000 |
| | 1000 | 6588723, .8682255 | 6.61308528 | .8692552, .5345225, 1.313627 | 15373001 |
| 30 | 20 | 6587971, .868145 | 6.61265623 | 8591962, 5379595, 1 323435 | 15372619 |
| | 50 | .6588702, .8682253 | 6.61307982 | 8708594, 5354301, 1.314003 | 15372989 |
| | 1000 | 6588723, .8682255 | 6 61308528 | 8692552, 5345225, 1 313627 | 15373001 |
| 40 | 20 | .6576564, .8681041 | 6 61103395 | .8778075, 5398821, 1.320782 | 1537285 |
| | 50 | 6588691, 8682251 | 6 61307918 | 8692278, 5345769, 1 313367 | .1537300 |
| | 1000 | .6588723, .8682255 | 6 61308528 | 8692552, 5345225, 1 313627 | .1537300 |

To reflect the convergence of algorithm CPSO to the optimal solution with the generations, the left part of the Table 3 presents the best solutions of the testing P7 varying with the iterations numbered from 1 to 20. It can be observed that the best objective is becoming improved greatly in the beginning iterations and then slightly with the increase number of iterations and gradually reaches a near optimal solution.

Particularly, to illustrate effectiveness of constraint fitness priority based ranking method in the PSO, a penalty-based evaluatıon is embedded as fitness in the PSO (referred as penalty-PSO in this paper) is ıntroduced as comparison counter. Penalty function used in the penalty-PSO is

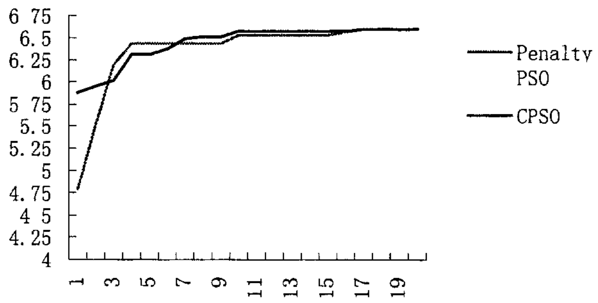

Fıgure 1 Relation between generation and objectıve of P7

defined as in [18]. Their comparison results of the testing P7 are shown in Table 3 and Figure 1, from which one can see that CPSO is more effective and superior than penalty-PSO.

## 5.3. Performance of the CPSO on Large Size Problem

In order to test the performance of the CPSO on large size problem, an integrated logistics decision problem selected from [20] is introduced as a testing example and the two layer decomposition method (TLD) [20] is given as comparison counter.

The integrated logıstic decision problem (IDM-M) is to determine simultaneously the assignment of annual production quantity and lot size at the suppliers and the annual shipment amounts and order quantity from the suppliers to the destinations to meet the demands with minimum total costs in a production and distribution network with multiple suppliers, multiple destinations, and multiple products. It is formulated as a large scale NLP that is neither convex nor concave with number of $2mL(1+n)$ decision variables and of $(nL+2mL+m+L]$ inequality and equality

Table 3. The best solutions of the testing P7 varying with generation number using CPSO and penalty-PSO

| | CPSO | | Penalty-PSO | |
|---|---|---|---|---|
| Gn | Best Solution | Best Objective | Best Solution | Best Objective |
| 1 | (.5005807, .833949) | 5 87554 | (.1527652, 8667169) | 4.795114 |
| 2 | ( 515465, .8359336) | 5.94883 | (.3787496, .571216) | 5 515699 |
| 3 | (.5277445, 837571) | 6.010272 | (.3128578, 7889293) | 6 195228 |
| 4 | (.5499627, 8806564) | 6.312963 | (.0455868, .495745) | 6.436533 |
| 5 | ( 547456, 8187297) | 6 312963 | (.3585616, .4849982) | 6 436533 |
| 6 | ( 6162313, 8259138) | 6 373346 | (.2041051, .1326934) | 6.436533 |
| 7 | ( 5665962, .786211) | 6.490521 | (.3759846, 9222181) | 6 436533 |
| 8 | ( 6093932, 8157683) | 6.512958 | ( 3242176, 8928556) | 6 436533 |
| 9 | (.6066412, 8475772) | 6.512958 | (.5849726, 830576) | 6 436533 |
| 10 | (.6001881, 8761123) | 6 576381 | ( 3825417, .8651791) | 6 532469 |
| 11 | ( 5794246, 8664396) | 6.576381 | (.5018676, 8389922) | 6.532469 |
| 12 | (.6430827, 8623704) | 6 576381 | (.579161, .8313762) | 6.532469 |
| 13 | (.6327764, .8685131) | 6 576381 | (.548866, .8366776) | 6 532469 |
| 14 | (.6390524, 8710732) | 6 576381 | ( 526365, .8729435) | 6 532469 |
| 15 | ( 6472884, .8700998) | 6.576381 | (.5365801, .8576724) | 6.532469 |
| 16 | ( 6514064, 8696131) | 6 584051 | (.6367062, 8612269) | 6 565932 |
| 17 | (.6498362, .8699648) | 6.595132 | ( 6228947, 8707712) | 6.598411 |
| 18 | (.6491703, 8694377) | 6 595132 | (.655944, .8660849) | 6.598411 |
| 19 | (.6504188, 8696761) | 6.595132 | ( 6523901, 8601647) | 6 598411 |
| 20 | ( 656019, 8687224) | 6 603044 | (.6568159, .8682291) | 6.601675 |

constraints, where $m$, $n$, and $L$ are numbers of suppliers, destinations, and products involved. For example, for 10 suppliers, 10 destinations, and 50 products, the model has 11000 decision variables. Due to being a large scale NLP problem, an effective heuristics TLD is developed in [20] to solve it.

The comparison results of the IDM-M with different sizes using the CPSO and TLD are show in Table 4. It can be shown from the Table 4 that the CPSO is slightly more effective than the TLD from the perspective of the best solutions and the CPU time required.

Table 4. Comparison Results between TLD and CPSO for different size of examples (Popsize = 30, Gmax = 50).

| Problem size ($m * n * L$) | Cost | | | CPU Time (sec) | |
|---|---|---|---|---|---|
| | CPSO | TLD [20] | Diff(%) | CPSO | TLD [20] |
| 5*5*5 | 672,716 | 674,670 | 0.28 | 5 | 5 |
| 5*5*10 | 1,292,585 | 1,304,090 | 0.89 | 49 | 51 |
| 5*10*10 | 2,0481,25 | 2,075,570 | 1.34 | 116 | 121 |
| 5*10*20 | 4,406,440 | 4,488,400 | 1 86 | 204 | 216 |
| 5*10*50 | 9,743,172 | 9,988,700 | 2 52 | 365 | 398 |
| 10*10*10 | 2,154,144 | 2,190,550 | 1 69 | 175 | 189 |
| 10*10*20 | 4,376,652 | 4,469,000 | 2.11 | 293 | 312 |
| 10*10*50 | 9,622,824 | 9,899,000 | 2.87 | 746 | 829 |

## 6. CONCLUSIONS

In this paper, a new CPSO algorithm with embedding constraint fitness priority-based ranking method and dynamic neighborhood operator is proposed for the nonlinear programming problem. It is the first time that, have been proposed for formulating and measuring the infeasible point.

In comparison with the basic particle swarm optimization, the CPSO performers the following characteristics,

(1) embedding the information of illegal particle into the evaluation process to develop new kinds of evaluation function,

(2) it can converge to the optimum from both sides of the feasible domain and the infeasible domain.

The theoretical analysis and the comparison results obtained by using GA and CPSO and sensitivity analysis of the CPSO as well as performance of the CPSO on largest size problems show that the CPSO is proved to be an effective one for solving NLP, especially where a significant degree of nonlinearity is present.

## REFERENCES

1 M.S. Bazarra and L M Shetty, *Nonlinear Programming· Theory and Algorithms*, pp 124–159, 373–378, John Wiley and Sons, New York, (1979).

2 R Y K Fung, J.F Tang and D. Wang, Extension of a hybrid genetic algorithm for nonlinear programming problems with equality and inequality constraints, *Computers and Operations Research* **29**, 261–274, (2002)

3 Y. Li and M. Gen, Non-linear mixed integer programming problems using genetic algorithm and penalty function, In *Proceedings of 1996 IEEE Int Conf. on SMC*, 2677–2682, (1996)

4 Y. Takao, M. Gen, T Takeaki and Y. Li, A method for interval 0-1 number non-linear programming problems using genetic algorithm, *Computers and Industrial Engineering* **29**, 531–535, (1995).

5. J F. Tang, D Wang *et al.*, A hybrid genetic algorithm for a type of nonlinear programming problem, *Computers Math. Applic.* **36** (5), 11–21, (1998)

6. J Su, A Hu and Z He, Solving a kind of nonlinear programming problems via analog neural networks, *Neurocomputing* **18**, 1–9, (1998).

7 Z.L Wang, L Qiu, Q Fu and C. Liang, Application of chaos optimization algorithm to nonlinear constrained programming (in Chinese), *Journal of North China Institute of Water Conservancy and Hydroelectric Power* **23** (2), 1–3, (2002)

8 M Dorigo, V. Maniezzo and A. Colori, Ant system Optimization by a colony of cooperating agents, *IEEE Trans On System, Man, and Cybernetics* **26** (1), 28–41, (1996).

9 L.X. Yan and D. Ma, Global optimization of non-convex nonlinear programs using line-up competition algorithm, *Computers and Chemical Engineering* **25**, 1601–1610, (2001)

10. F. Glover *et al*, Genetic algorithm and tabu search· Hybrid for optimizations, *Computers and Operations Research* **22**, 111–134, (1995).

11 T C Lin *et al.*, Applying the genetic approach to simulated annealing in solving some NP-hard problems, *IEEE Trans On SMC* **23**, 1752–1766, (1993).

12 R. Ostermark, Solving a nonlinear non-convex trim loss problem with a genetic hybrid algorithm, *Computers and Operations Research* **26**, 623–635, (1999)

13. J Kennedy and R Eberhart, Particle swarm optimization, In *Proc. IEEE Int Conf. on Neural Networks*, 1942–1948, Piscataway, NJ, (1995)

14 R Eberhart and J Kennedy, A new optimizer using particle swarm theory, In *Proc. 6th Int Symposium on Micro Machine and Human Science*, 39–43, Nagoya, Japan, (1995)

15 Y Shi and R. Eberhart, A modified particle swarm optimizer, *Proc IEEE Int Conf on Evolutionary Computation*, 69–73, (1998).

16. J. Kennedy and W.M. Spears, Matching algorithms to problems An experimental tests of the particle swarm and some genetic algorithms on the multimodal problem generator, In *Proc IEEE Int Conf on Evolutionary Computation*, pp 78–83, Anchorage, AK, (1998).

17 X.F Xie, W J Zhang, J Ruan and Z.L Yang, Genetic algorithm for constrained nonlinear programming problems (in Chinese), *Computer Engineering and Application* **21**, 64–67, (2002).

18 T Back, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, (1996)

19 B D Liu and R. Zhao, *Stochastic Programming and Fuzzy Programming* (in Chinese), pp 26–33, Tsinghua, Beijing, (2001).

20 J F Tang, K L. Yung and A.W.H. Ip, Heuristics-based integrated decision for logistic networks systems, *Journal of Manufacturing Systems* **23** (1), 1–13, (2004).

21 M M Millonas, Swarms, phase transition, and collective intelligence, In *Artificial Life III*, (Edited by C G Langton), Addison Wesley, Massachusetts, (1994)

22 P N Suganthan, Particle swarm optimizer with neighbourhood operator [A], In *Proc of the Congress on Evolutionary Computation [C]*, pp 1958–1962, Washington, DC, (1999)
23 Y.F. Shi and R. Eberhart, Parameter selection in particle swarm optimization, In *Proc. of the 7th Annual Conf on Evolutionary Programming*, pp 591–600, (1998)