ATLANTIS PRESS
Advances in Computer Science Research, volume 44
3rd International Conference on Wireless Communication and Sensor Network (WCSN 2016)

Discrete Particle Swarm Optimization Algorithms for the Prize-Collecting Call Control Problem on Lines

Qian-Na Cui
School of Mathematics and Statistics, Yunnan University
Kunming, 650500, P.R.China
cuiqianna@126.com

Qiao-Yan Zhu
School of Mathematics and Statistics, Yunnan University
Kunming, 650500, P.R.China
zqy12356@qq.com

Xing Wu
School of Mathematics and Statistics, Yunnan University
Kunming, 650500, P.R.China
wuxing123@163.com

***Abstract*—In this paper,we propose two discrete particle swarm optimization algorithms to solve a prize-collecting call control problem on lines based on the classical particle swarm optimization algorithm. The two algorithms combine simulated annealing and genetical methods with particle swarm optimization respectively. Then, we do lots of experiments to compare the two discrete particle swarm optimization algorithms we addressed, in which we compare the results and iteration time and the convergence of two algorithms under some given variables. Finally, computational results show that the proposed discrete particle swarm optimization with genetic algorithm is very efficient and can be quickly obtained good results.**

Keywords-DPSO algorithm; prize-collecting; call control; lines

I. INTRODUCTION

In recent years, much attention has been paid to the unsplittable flow problem (UFP). Simply summarized the definition of this problem: given an undirected graph where each edge has its capacity and a set of vertex pairs with a positive demand and a positive profit. The goal is to find a subset with the maximum profit form the given set so that each chosen pair from the subset can satisfy its full demand without exceeding its capacity.

Bansal et al. [1] considered the UFP on a line, they simply introduced that the UFP is NP-hard. To solve this problem, they added dynamic programming into linear programming relaxation to avoid a $\Omega(n)$ integrality gap generated by linear programming. They presented the first polynomial time $O(\log n)$ approximation algorithm for the problem without any assumptions, which broke the previous conclusion that is no polynomial time $O(n)$ approximation algorithm was known in the past.

Bonsma et al. [2] studied the UFP on a path, they presented a constant-factor (7+ ε) approximation algorithm for this problem, which improved on the previous ratio of $O(\log n)$. They also showed a 2+ ε -approximation algorithm with slightly violated the capacities, then they proved that this problem is strongly NP-hard. Based on their results, another group gave a new improvement. Anagnostopoulos et al. [3] implied a 2+ ε approximation for the same problem without violating any capacities, which was better than the 7+ ε ratio for any constant $\varepsilon > 0$. Batra et al. [4] studied the same problem, they obtained a PTAS when the the ratio of a task's profit and demand lie in a constant range. They also obtained a PTAS, in which they were allowed to shorten the paths of the tasks.

UFP was widely used in various fields, for which studies did a lot of researches about its actual problems, such as the problem of resource allocation. Bar-Noy et al.[5] presented approximation algorithms for solving resource allocation and scheduling problems, which factors applied to many problems such as dynamic storage allocation and so on. Li.et al. [7] considered another problem about UFP, which is the ring loading problem with penalty cost. In that paper, they showed a 1.58-approximation algorithm for the demand unsplittable case. Later, motivated by the research of prize-collecting Steiner tree problem [8], Li. et al. [6] introduced the prize-collecting call control (PCCC) problem on lines, for which they designed a 1.58-approximation algorithm using a randomized rounding technique.

In our paper, we do a research about (PCCC) problem on lines proposed by Li. et al. [6], which is prize-collecting call control problem on lines, the description of this problem is as follows. Given an line $G=(V,E)$ and a set of K pairs of vertexes $\mathscr{P}=\{P_k|k=1,2,\ldots,K\}$, in which each pair of vertexes is representative as a request path. Each request path P_k has a positive demand d_k

and a positive penalty cost p_k. The objective of PCCC problem is to find a subset $\mathscr{P}'$ from $\mathscr{P}$ so that the sum of the total demand $\sum_{k:P_k\in\mathscr{P}'} d_k$ and the total penalty cost $\sum_{k:P_k\in\mathscr{P}\setminus\mathscr{P}'} p_k$ is minimum. The integer linear program (ILP) of this problem is as follows:

$$\begin{cases} min \quad f = L + \sum_{k=1}^{K} p_k(1-x_k) \\ \sum_{k:P_k\in\mathscr{P}} d_k x_k \le L \\ x_k \in \{0,1\} \end{cases}$$

in which for each $P_k \in \mathscr{P}$, P_k is either accepted if $x_k \equiv 1$ or rejected once $x_k \equiv 0$ and L is the maximum load of accepted paths, f is simply represent as the objective function. Li. et al. [6] proved that the PCCC problem on lines is NP-hard.

To solve above problem, we propose two algorithms based on classical particle swarm algorithm. Kennedy and Eberhart [9] proposed particle swarm optimization algorithm in 1995, which is intended to gain a global optimal value. The principles of PSO algorithm can be described as follows [9]. It is assumed that the search space is n-dimension and population size is M, in which the position and velocity of particle i in the t th iteration is defined by $X_i^t = \left(x_{i1}^t, x_{i2}^t, \mathrm{L}\ , x_{in}^t\right)$ and $V_i^t = \left(v_{i1}^t, v_{i2}^t, \mathrm{L}\ , v_{in}^t\right)$ where $i = 1, 2, \mathrm{L}\ , M$. The flying speed and position of particle i in the j th $\left(j = 1, 2, \mathrm{L}\ , n\right)$ subspace of the t th iteration are calculated according to:

$$v_{ij}^{t+1} = wv_{ij}^t + c_1 r_1\left(p_{ij}^t - x_{ij}^t\right) + c_2 r_2\left(g_j^t - x_{ij}^t\right) \tag{1}$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \tag{2}$$

where w, called weight, is a constant to control the impact of the previous velocities on the current velocity. c_1 and c_2 are acceleration factors, which can impact the moving trail of particles, r_1 and r_2 are uniformly distributed random variables in [0,1], g_j can be representative as the best history position in the whole population, p_{ij} is the best history position of the current particle.

II. THE ALGORITHM OF DPSO

Due to the continuous character of PSO, Kashan et al. [10] proposed a discrete particle swarm optimization algorithm for scheduling parallel machines. We use this algorithm methods efficiently to propose two DPSO algorithms and then solve the PCCC problem with them.

A. *The Proposed DPSO-GA Algorithm*

We set k numbers of dimensions are presented each for one of k request calls, hence, a solution for the problem is represented by an array whose length is equal to the number of non-zero elements of k −dimention array. The location of a particle in the i th dimension represents the i th path to be accepted or rejected, in which the i th path is accepted if the i th value is 1, or rejected if the if the i th value is 0. We draw lessons from Kashan et al. [10] and propose a DPSO algorithm with the basis on GA method to solve the PCCC problem on lines, which is called DPSO-GA.

The process of generating a new position for selecting an optimal solution in the given particle swarm is depicted as follows [10]:

$$V_i^{t+1} = V_i^t \oplus (R_1 \otimes (P_i^t \ominus X_i^t)) \oplus (R_2 \otimes (P_g^t \ominus X_i^t)) \tag{3}$$

$$X_i^{t+1} = X_i^t \oplus V_i^{t+1} \tag{4}$$

where R_1 and R_2 are 1-by-k arrays compromising 0 or 1 elements. These random arrays are generated from a Bernoulli distribution in which the probability of getting 1 is equal to 0.5. V_i^t and X_i^t are the i th particle current velocity and position arrays, respectively. P_i^t and P_g^t are the i th particle best position and the global best position in history. The subtraction operator $(\ominus)$ and the multiply operator $(\otimes)$ used by Kashan et al. [10] in equation (3) and (4) are also our operators. The add operator $(\oplus)$ is a crossover operator that method is typically used in genetic algorithms, which method is the only different point with Kashan et al. [10]. First, it randomly generates two cross points from particle chain A. Next, it exchanges the sub-chain between the two points and two new chains obtained. Then, we select one of the new chains randomly to be the result of the add operator.

B. *The Proposed DPSO-SA Algorithm*

Simulated Annealing (SA) is a random-search technique proposed by Metropolis [11], whose major advantage is an ability of suddenly jumping with probability to avoid becoming trapped in local minima. So, we can add the SA into DPSO algorithm to solve the PCCC problem on lines efficiently, called DPSO-SA algorithm, it continues the updating equation (3) and (4). Before the updating of the position and the velocity vector, given an initial temperature T_0, we compute the temperature T_i by $e^{-f\left(p_i^t\right)-f\left(p_g^t\right)/T(0)}$ [11], the algorithm

repeats the updating operator until reaching the temperature balance T_i [11]. More important, the simulated annealing operator is $T_{i+1} = CT_i$ where $C \in (0,1)$ until satisfying the convergence conditions.

DPSO-SA pseudo code.

```
Begin
  for i: 1 to swarm size
    X(i,:): generate a particle at
random;
    V(i,:): generate a particle at
random;
    P(i,:)=X(i,:);
  end for
  Pg=X(1,:);
  for i:2 to swarm size
    find the X(j,:) which satisfies the
min(f);
  end for
    Pg=X(j,:);
  given a initial temperature T(0);
  t=1;
    for i:1 to swarm size

T(i)=exp(-(f(P(i,:))-f(Pg))/T(0));
    end for
  generating new solution X', obtaining
the value:
  f(X')-f(X);
  if
min{1,exp(-(f(X')-f(X))/T(i))}>random
[0,1]
    accepting the solution X';
  else
    break;
  end if
  for i:1 to swarm size
    V(i,:):update the i-th particle
velocity vector by (3);
    X(i,:):update the i-th particle
position vector by (4);
    if min{f(X(i,:))}<min{f(P(i,:))}
      P(i,:)=X(i,:);
    else
      P(i,:)=P(i,:);
    end if
  end for
  if min{f(Pg)}>min{f(P(i,:))}
    Pg=P(i,:);
  end if
  T(0)=T(0)*C;
  t=t+1;
End
```

III. Computational Experiments

The two DPSO algorithms we proposed are better or not to solve effectively the PCCC problem on lines, we do lots of experiments to prove it. We will give 100 particles to each algorithm and let the two programs run 100 times for each experiment in MATLAB to reach results, and use the finial result and the total running time to compare what we want to. In all following tables, we let "result" and "time" be representatives.

A. *Experiments of DPSO-SA*

Due to the optimal value obtained by SA is different when its parameters changed a little, we do some experiments to obtain the best parameters. An experimental frame-works, namely E1, is considered each of them having two influence factors: the initial temperature T_0 and the annealing function $T_{i+1} = CT_i$. Theoretically, the annealing velocity is no more quicker than $T_i = T_0 / (1 + \ln(i))$, in other words, T_i is diminishing once $i > 2$, so we design C is an alterable constant in $(0,1)$. In order to obtain the best result, insuring the initial temperature big enough is of great importance, then we decide that the initial temperature is similar to the function value of initial global best position $f(p_g)$, that is to say let T_0 be the relative to the $f(p_g)$, for which we definite $T_0 = rf(p_g)$ where r is a changeable constant.

Table 1. Framework E1

	K	d_k	p_k	r	C
E1	100	$U[1,3]$	$U[1,5]$	0.5,0.7,1.0,1.2,1.5	0.2, 0.4,0.5, 0.7,0.9

We can see the experimental results about DPSO-SA with $K=100$ in Table2. Clearly, when the value of r is 0.5, 0.7, 1.0, 1.2, 1.5, the two values change separately. The fluctuating values are 18, 8, 20, 33, 16 and 0.4990, 0.0760, 0.9050, 0.8890, 0.7950, so the two vales fluctuate least when $r=0.7$. Similarly, the changeable range is least when $C=0.7$, the two values only change 2 and 0.1410. In general, the two parameters result into slightly alteration, so we will choose the two best parameters to do the following experiments, that is to say $r=0.7$ and $C=0.7$.

B. *Contrastive Experiments between DPSO-GA and DPSO-SA*

Analysis of results and time. In this part, we study the comparison on the effectiveness of the DPSO-GA algorithm and the DPSO-SA algorithm by do a mount of experiments. Three experimental frameworks, namely E2, E3, E4, are all considered having three influence factors: the value of K, demand (d_k), and the penalty (p_k). Table 3 clearly presents a summary of all experimental frameworks.

TABLE 2. RESULTS FOR EXPERIMENT E1

K	*r*	*C*	*result*	*time*	*K*	*r*	*C*	*result*	*time*
		0.2	216	63.5240			0.2	227	63.5390
		0.4	227	63.8040			0.4	230	63.6790
100	0.5	0.5	234	63.6790	100	0.7	0.5	227	64.3810
		0.7	223	63.4300			0.7	227	63.3680
		0.9	222	63.3050			0.9	222	63.3050
		0.2	226	63.8510			0.2	203	63.4490
		0.4	221	63.6500			0.4	215	63.5700
100	1.0	0.5	206	64.1160	100	1.2	0.5	227	63.9750
		0.7	226	63.3990			0.7	225	63.2890
		0.9	226	63.2110			0.9	236	63.0860
		0.2	234	63.4920					
		0.4	218	63.8660					
100	1.2	0.5	222	63.5540					
		0.7	228	63.4140					
		0.9	229	63.0710					

TABLE 3. FRAMEWORK OF COMPARISON EXPERIMENTS

	K	d_k	p_k
E_2	40,100,150,180,300,500,1000	$U[1,20]$	$U[1,10]$
E_3	100	$U[20,50], U[1,100], U[100,200], U[100,800]$	$U[1,20], U[20,50]$
E_4	50, 120	$U[20,50]$	$U[1,6], U[8,15], U[20,60], U[50,100]$

TABLE 4. RESULTS FOR EXPERIMENT E2

			DPSO-GA		DPSO-SA	
K	d_k	p_k	*result*	*time*	*result*	*time*
40			197	6.6450	193	16.0520
100	$U[1,20]$		424	62.9300	473	63.2430
150			841	115.1810	854	116.5570
180		$U[1,10]$	1130	152.7030	1019	155.8240
300			1792	348.8240	1829	356.2110
500			3166	824.7430	3450	858.1210
1000			6827	$3.0043e+003$	6922	$3.0153e+003$

TABLE 5. .RESULTS FOR EXPERIMENT E3

			DPSO-GA		DPSO-SA	
K	d_k	p_k	*result*	*time*	*result*	*time*
100	$U[20,50]$	$U[1,20]$	1407	62.7900	1330	63.4140
	$U[1,100]$		1328	62.3850	1465	63.3830
	$U[100,200]$		2655	62.3530	3286	63.8360
	$U[100,800]$		4964	62.2590	7959	63.3050
100	$U[20,50]$	$U[20,50]$	2995	62.1970	3079	63.4140
	$U[1,100]$		3171	62.2910	3042	63.3200
	$U[100,200]$		4961	62.4320	4605	62.8360
	$U[100,800]$		7884	61.9480	8257	63.1490

TABLE 6. .RESULTS FOR EXPERIMENT E4

			DPSO-GA		DPSO-SA	
K	d_k	p_k	*result*	*time*	*result*	*time*
50	$U[20,50]$	$U[1,6]$	254	22.8540	257	22.5420
		$U[8,15]$	673	23.1350	612	22.6510
		$U[20,60]$	1566	23.1660	1571	22.4950
		$U[50,100]$	1654	23.0720	1830	22.5730
120	$U[20,50]$	$U[1,6]$	1007	81.3380	969	82.5870
		$U[8,15]$	1870	81.4170	1871	82.2440
		$U[20,60]$	4017	81.2770	3992	82.9300
		$U[50,100]$	4722	83.1630	5074	82.9920

In table 4, we can see that the all results obtained by the DPSO-SA algorithm is bigger than that of the DPSO-GA algorithm expect the case of value $K=180$, especially, the difference of result is 284 between the two DPSO algorithms, so the DPSO-GA is better to get the optimal result. In addition, the total time of DPSO-GA is shorter than that of the DPSO-SA, the comparison is the most outstanding at value $K=40$. But beyond that, there is a very obvious disadvantage both in the two DPSO algorithms that is the total time of the two DPSO algorithms will increase to about 3000 seconds which is about 50 minutes when the request calls is up to 1000. So, the two algorithms will cost too much time to solve the actual problem when request calls is a large number.

Table 5 shows the results for experiment E3, and performs the changing fact of results and times when the demand increases. We can see that the result of DPSO-SA algorithm is more than that of DPSO-GA clearly once the demand increases to $U[100,800]$. However, there are two results bigger than that of the DPSO-SA, which show that the DPSO-GA algorithm is not stable.

In the Table 6, it is shown that the different results between DPSO-GA algorithm and DPAO-SA algorithm when the penalty increased. We can see that the influence of penalty's supremum is not be neglected, in which the results of the two algorithms are the most significant once the penalty's supremum is up to 100. Therefore, the DPAO-GA is better to solve the PCCC problems with higher penalty.

Analysis of convergence. For more effective comparison on the two algorithm, we use their convergence to do it. First, we let three variables be $K=100$, $d_k \in U[1,20]$, $p_k \in U[1,10]$, the first two pictures of Fig.1. performs the convergence that DPSO-GA has better convergence rate, in which the fit curve of DPSO-GA is in an iterative smooth before 30 times but the other is after 30times. Then we replace the value of K, let K be 50. The last two pictures of Fig. 1 show that the convergence of DPSO-GA begins to converge for 20 iterations, the other begins for 40 iterations.

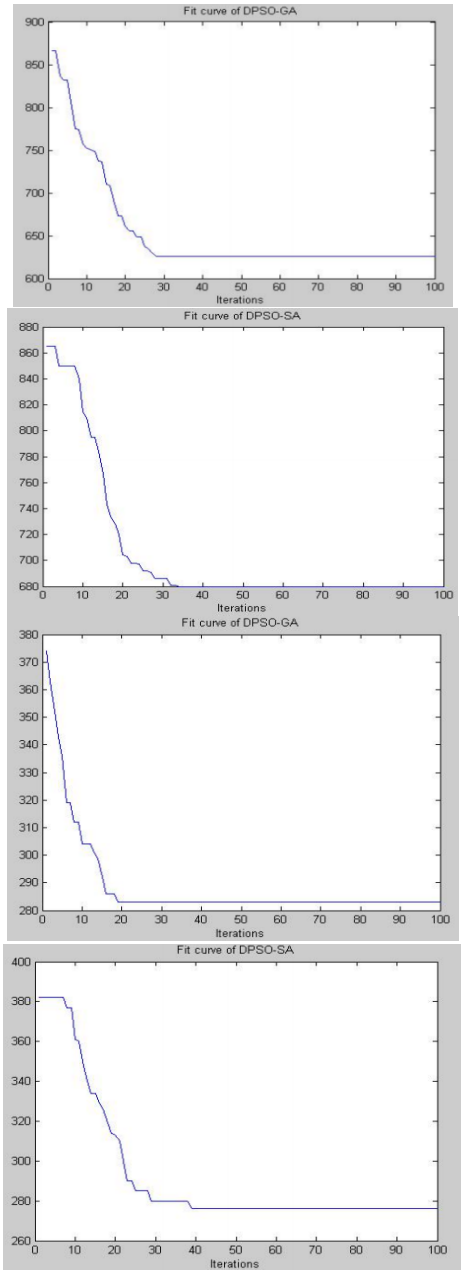

Figure 1. Fit curve with $K = 100$ and $K = 50$.

IV. CONCLUSION

To our knowledge, the application of PSO algorithm in solving the PCCC problem on lines was proposed in this paper. We proposed two DPSO algorithms and then compare optimal results, running time and convergence of the two algorithms. Also, we found that DPSO-GA outperforms DPSO-SA in all of the experimental frameworks. We do some experiments to compare the results and iteration time of the two DPSO algorithms, we found that it is not about good or bad about the two algorithms, it is all about certain point. In addition, the two DPSO algorithms is not perfect, which have many disadvantages, thus, finding preferable ways to solve the PCCC problems is interesting in the future. Also, extending our two DPSO algorithms for solving other problems is attractive.

ACKNOWLEDGMENTS

The research was supported by the National Nature Science Foundation of China [No. 11301466] and the Natural Science Foundation of Yunnan Province of China [No.2014FB114].

REFERENCES

[1]. N. Bansal, Z. Friggstad, R. Khandekar and M.R. Salavatipour.: A Logarithmic Approximation for Unsplittable Flow on Line Graphs. In: Proceeding of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1–15 (2014)

[2]. P. Bonsma, J.Schulz and A. Wiese.: A Constant-factor Approximation Algorithm for Unsplit-table Flow on Paths. J. SIAM J. Compute. 43, 767–799 (2014)

[3]. A. Anagnostopoulos, F. Grandoni, S. Leonardi and A. Wiese.: A Mazing 2+ε Approximation for Unsplittable Flow on a Path. In: Proceeding of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 26–41 (2014)

[4]. J. Batra, N. Garg, A. Kumar, T. Mo¨mek and A. Wiese.: New Approximation Schemes for Unsplittable Flow on a Path. In: Proceeding of the Twenty-Sixth Annual ACM-SIAM Sym-posium on Discrete Algorithms, pp. 47–58 (2015)

[5]. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor and B. Schieber.: A Unified Approach to Approximating Resource Allocation and Scheduling. J. ACM. 48, 1069–1090 (2001)

[6]. W. Li, J. Li, L. Guan and Y. Shi.: The Parize-Collection Call Control Problem on Weighted Lines and Rings. J. Rai. Oper. Rea. 50, 39–46 (2016)

[7]. W. Li, J. Li, L. Guan.: Approximation Algorithms for the Ring Loading Problem with Penalty Cost. J. Inf. Pro. Let. 114, 56–59 (2014)

[8]. A. Archer, M.H. Beteni, M.T. Hajianhayi and H. Karloff.: Improved Approximation Algo-rithms for Prize-Collecting Steiner Tree and TSP. J. SIAM J. Cmpute. 40, 309–332 (2011)

[9]. J. Kennedy, RC. Eberhart.: Particle Swarm Optimization. In: Proceeding of IEEE International Conference on Neural Networks, pp. 1942–1948. IEEE Serverce Center, Piscataway (1995)

[10]. A. H. Kashan and B. Karimi.: A Discrete Particle Swarm Optimization Algorithm for Scheduling Parallel Machines. J. Comprter and Industrial Engineering. 56, 216–223 (2009)

[11]. N. Metropolis, AW. Rosenbluth, MN. Rosenbluth, AH. Teller and E. Teller.: Equation of State Calculations by Fast Computing Machines. J. Chem. Phys. 21, 1087–1090 (1953)