Parallel Multi-Population Particle Swarm Optimization Algorithm for the Uncapacitated Facility Location Problem using OpenMP

Dazhi Wang, Chun-Ho Wu, Andrew Ip, Dingwei Wang, and Yang Yan

Abstract— **Parallel multi-population Particle Swarm Optimization (*PSO*) Algorithm using *OpenMP* is presented for the Uncapacitated Facility Location (*UFL*) problem. The parallel algorithm performed asynchronously by dividing the whole particle swarm into several sub-swarms and updated the particle velocity with a variety of local optima. Each sub-swarm changes its best position so far of to its neighbor swarm after certain generations. The parallel multi-population *PSO* (*PMPSO*) algorithm is applied to several benchmark suits collected from *OR*-library. And the results are presented and compared to the result of serial execution multi-population *PSO*. It is conducted that the parallel multi-population *PSO* is time saving, especially for large scale problem and generated more robust results.**

I. Introduction

THE Particle Swarm Optimization (*PSO*) is one of the recent metaheuristics invented by Eberhart and Kennedy [1] based on the metaphor of social interaction and communication such as bird flocking fish schooling. In *PSO*, the potential solutions, so-called particles, move around in a multi-dimensional search space with a velocity, which is constantly updated by the particle's own experience and the experience of the particle's neighbors (*local* version) or the experience of the whole swarm (*global* version). *PSO* has been successfully applied to a wide range of applications such as function optimization, neural network training, task assignment, and scheduling problems.

Parallel optimization methods have been given more concentration recently. A unified method of parallel Genetic Algorithm (*GA*) is presented for the representative problems [1]. Parallel Strategy is quoted to the Ant Colony Algorithm (*ACO*), the influence of asynchronous and synchronous parallel styles to *ACO* are designed and compared [3]. A parallel recombinative Simulated Annealing (*PRSA*) is presented to the parallel problem of Single Instruction Multiple Data and Multiple Instruction Multiple Data [4]. A parallel particle swarm algorithm designed to solve a kind of combinatorial optimization problem was presented to overcome the heavy computational time disadvantage of general serial algorithm. It was also applied to hot rolling planning [5].

Computer engineers and researchers have concentrated on accelerating scientific applications on high performance workstations and large scale systems. For this purpose, they have introduced various kinds of architectures, compilers, and parallel programming models and their APIs. One of the efforts to provide easy programming models to users on shared memory multiprocessors systems is *OpenMP* [6]. Currently *OpenMP* is a popularly used and standard API in many areas, especially in scientific applications.

Location problems is one of the most widely studied problems in NP-hard [7] combinatorial optimization problems thus there is a very rich literature in operation research (*OR*) [8]. In addition, the bank account location problem, network design, vehicle routing, distributed data and communication networks, computer network design, cluster analysis, machine scheduling, economic lot sizing, portfolio management are some instances without facilities to locate problems that is modeled as an *UFL* problem in the literature.

UFL problems have been studied and examined extensively by various attempts and approaches. All important approaches relevant to *UFL* problems can be classified into two main categories: exact and metaheuristics based algorithms. There is a variety of exact algorithms to solve the *UFL* problem, such as branch and bound [9], linear programming, Lagrangean relaxation [10], dual approach (*DUALLOC*) of Erlenkotter [11] and the primal-dual approaches of Körkel [12]. Although *DUALLOC* is an exact algorithm, it can also be used as a heuristic to find good solutions. It is obvious that since the *UFL* problem is NP-hard [7] exact algorithms may not be able to solve large practical problems efficiently. There are several studies to solve *UFL* problem with metaheuristics. Some of recent studies are tabu search [13, 14], genetic algorithms [15], neighborhood search [16], and simulated annealing [17]. And a continuous particle swarm optimization algorithm with local search is presented by Mehmet and Ali [18].

In an *UFL* problem there are a number of sites, n and a number of customers, m .Each site has a fixed cost fc_i . There is a transport cost from each site to each customer c_{ij} . There is no limit of capacity for any candidate site and the whole demand of each customer has to be assigned to one site. We are asked to find the number of sites to be established and specify those sites such that the total cost will be minimized (1). The mathematical formulation of *UFL* can be stated as follows [7]:

Manuscript received December 2, 2007. This work is supported by the Key Program of National Natural Science Foundation of China Grant No. 70431003, the Science Fund for Creative Research Group of National Natural Science Foundation of China Grant No. 60521003 and National Science and Technology Support Plan Grant No. 2006BAH02A09.

Dazhi Wang is with the School of Information Science and Engineering, Northeastern University, Shenyang, Liaoning Province 110004 China (phone: 086-024-83684032; e-mail: wongdz@gmail.com).

Chun-Ho Wu and Andrew Ip are with the Department of Industrial and Systems Engineering The Hong Kong Polytechnic University, Kowloon, Hong Kong (email: jack.wu@polyu.edu.hk; mfwhip@inet.polyu.edu.hk).

Dingwei Wang and Yang Yan are with the School of Information Science and Engineering, Northeastern University, Shenyang, Liaoning Province 110004 China (e-mail: dwwang@mail.neu.edu.cn; yanyangmail@163.com).

978-1-4244-1823-7/08/$25.00©2008 IEEE

$$Z = \min(\sum_{j=1}^{m}\sum_{i=1}^{n} c_{ij} \cdot x_{ij} + \sum_{i=1}^{n} fc_i \cdot y_i) \quad . \tag{1}$$

Subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \quad . \tag{2}$$

$$0 \le x_{ij} \le y_i \in \{0;1\} \quad . \tag{3}$$

where

$$i = 1,\cdots,n;\ j = 1,\cdots,m;$$

x_{ij} : the quantity supplied from facility i to customer j;

y_i : whether facility i is established ($y_i = 1$) or not ($y_i = 0$).

Constraint (2) makes sure that all demands have been met by the open sites, and (3) is to keep integrity. Since it is assumed that there is no capacity limit for any facility, the demand size of each customer is ignored, and therefore (2) established without considering demand variable.

The organization of this paper is as follows. Section II introduces the proposed parallel multi-population *PSO* for *UFL*. In Section III the experimental results are provided and Section IV presents the conclusion driven.

II. Parallel Multi-Population PSO Algorithm for UFL

A. A Pure PSO Algorithm

The PSO algorithm for the *UFL* problems considers each particle based on three key vectors; position (X_i), velocity (V_i), and open facility (Y_i). $X_i = [x_{i1}, x_{i2}, x_{i3}, \cdots, x_{in}]$ denotes the i^{th} position vector in the swarm, where x_{ik} is the position value of the i^{th} particle with respect to the k^{th} dimension $(k = 1,2,3,\cdots,n)$. $V = [v_{i1}, v_{i2}, v_{i3}, \cdots, v_{in}]$ denotes the i^{th} velocity vector in the swarm, where v_{ik} is the velocity value of the i^{th} particle with respect to the k^{th} dimension. $Y = [y_{i1}, y_{i2}, y_{i3}, \cdots, y_{in}]$ represents the opening or closing facilities identified based on the position vector(X_i), where y_{ik} represents the opening or closing the k^{th} facility of the i^{th} particle. For an n problem, each particle contains n *-facility* number of dimensions.

Initially, the position (x_{ij}) and velocity (y_{ij}) vectors are generated randomly and uniformly as continuous sets of values between $(-10.0, +10.0)$ and $(-4.0, +4.0)$ respectively that is consistent with the literature [19]. The position vector $X_i = [x_{i1}, x_{i2}, x_{i3}, \cdots, x_{in}]$ corresponds to the continuous position values for *n* facilities, but it does not represent a candidate solution, a particle, the position vector is converted to a binary variable, $Y_i \leftarrow X_i$, which is also a key element of a particle. In other words, a continuous set is converted to a discrete set for the purpose of creating a candidate solution, particle. The fitness of the i^{th} particle is calculated by using open facility vector (Y_i). For simplicity, $f_i(Y_i \leftarrow X_i)$ is from now on be denoted with f_i .

In order to ascertain how to drive an open facility vector from position vector, an instance of 5-facility problem is illustrated in Table 1. Position values are converted to binary variable using following formula:

$$y_i = \left\lfloor \left| x_i \bmod 2 \right| \right\rfloor . \tag{4}$$

In equation (4) a position value is first divided by 2 and then the absolute value of the remainder is floored; and the obtained integer is taken as an element of the open facility vector. For example, fifth element of the open facility vector, y_3 , can be found as follows:

$$\left\lfloor \left| -7.34 \bmod 2 \right| \right\rfloor = \left\lfloor \left| -1.34 \right| \right\rfloor = \left\lfloor 1.34 \right\rfloor = 1 .$$

An illustration of derived open facility vector from position vector for a 5-facility to 6-customer problem is given in TABLE I.

For each particle in the swarm, a personal best, $P_i = [p_{i1}, p_{i2}, p_{i3}, \cdots, p_{in}]$, is defined, whereby p_{ik} denotes the position value of the i^{th} personal best with respect to the k^{th} dimension. The personal bests are determined just after

TABLE I
AN ILLUSTRATION OF DERIVED OPEN FACILITY VECTOR FROM POSITION VECTOR FOR A 5-FACILITY TO 6-CUSTOMER PROBLEM

Symbol	Particle Dimension(k)				
i_{th} Particle Vectors	1	2	3	4	5
Position Vector(X_i)	3.22	-0.73	-7.43	2.3	-3.5
Velocity Vector(V_i)	-0.73	2.33	-1.33	3.56	2.35
Open Facility Vector(Y_i)	1	0	1	0	1

generating Y_i vectors corresponding to the fitness values. In every generation, t , the personal best of each particle is updated if a better fitness value is obtained. Regarding the objective function, $f_i(Y_i \leftarrow X_i)$, the fitness value for the personal best of the i^{th} particle, P_i , is denoted by f_i^{pb} . The personal best vector is initialized with position vector $(P_i = X_i)$ at the beginning. Where $P_i = [p_{i1}, p_{i2}, p_{i3}, \cdots, p_{in}]$ is the position vector and the fitness values of the personal bests are equal to the fitness of position, $f_i^{pb} = f_i$.

Then, the best particle with respect to fitness value in the whole swarm is selected with the name global best and denoted as $G_i = [g_1, g_2, g_3, \cdots, g_n]$. The global best, $f_g = f(Y \leftarrow G)$, can be obtained as the whole swarm, $f_g = \min\{f_i^{gb}\}$, with its corresponding position vector, X_g ,which is to be used for $G = X_g$, where $G = [g_1 = x_{g1}, g_2 = x_{g2}, g_3 = x_{g3}, \cdots, g_n = x_{gn}]$ and $Y_g = [y_{g1}, y_{g2}, y_{g3}, \cdots, y_{gn}]$ denotes the Y_i vector of the global best found.

Afterwards, the velocity of each particle is updated based on its personal best and global best in the following way (5):

$$v_{ik}^{(t+1)} = (w \cdot v_{ik}^{(t)} + c_1 r_1 (p_{ik}^{(t)} - x_{ik}^{(t)}) + c_2 r_2 (g_k^{(t)} - x_{ik}^{(t)})) . \tag{5}$$

where, w is the inertia weight to control the impact of the previous velocity on the current one. In addition, r_1 and r_2 are random numbers between $[0,1]$, c_1 and c_2 are the learning factors, which are also called social and cognitive parameters respectively. The next step is to update the position with (6).

$$x_{ik}^{(t+1)} = x_{ik}^{(t)} + v_{ik}^{(t+1)}. \quad (6)$$

After getting position values updated for all particles, the corresponding open facility vectors are determined with the fitness values in order to start a new iteration if the predetermined stopping criterion is not satisfied. So the *PSO* algorithm for *UFL* problem is elaborated in the pseudo code given in Fig.1.

```
Begin
    Initialize particles positions
    For each particle
        Find open facility vector (4)
        Evaluate (1)
        Do {
            Find the personal best and the global best
            Update velocity (5) and position (6) vectors
            Update open facility vector (4)
            Evaluate (1)
        } While (Termination)
End
```

Fig. 1. The Pseudocode of PSO

B. Multi-Population PSO (MPSO) Algorithm for UFL

1) Particle Initialization and Sub-population Division Method:

Initially, N particles were randomly generated (including their position and velocity). In order to maintain the diversity of the sub-population and avoid the extreme centralization and decentralization, the following method was used to divide the particles into sub-populations: First, calculated the fitness values of the N particles, and sort them by their fitness, and numbered the particles with $1 \sim N$; then, put the number i particles into the $i(\mathrm{mod})M+1$ sub population.

2) Updating Principles:

(5) described the classic velocity and position updating principle of *PSO* algorithm. For the *MPSO* algorithm, taken the cooperation of the sub-populations into account, the principle (5) was improved:

$$\begin{aligned} v_{ik}^{(t+1)} = (w \cdot v_{ik}^{(t)} + c_1 r_1 (p_{ik}^{(t)} - x_{ik}^{(t)}) + c_2 r_2 (g_k^{(t)} - x_{ik}^{(t)}) \\ + c_3 r_3 (p_{jik}^{(t)} - x_{ik}^{(t)})) \end{aligned} \quad (7)$$

where, $p_{jik}^{(t)}$ is the best position of particle i in sub-population j, r_3 is random numbers between $[0,1]$, c_3 is the learning factors, which are also called cognitive parameters respectively. The *MPSO* algorithm for *UFL* problem is elaborated in the pseudo code is illustrated in Fig.2. The serial *MPSO* algorithm for *UFL* problem is elaborated in the pseudo code.

C. Parallel MPSO Algorithm using OpenMP for UFL

OpenMP is a library that supports parallel programming on shared memory architecture platform and is more suitable for multi core processors. The parallelization execution can be achieved by the directive compiler instruction. *OpenMP* provides a fork-and-join execution model in which a program begins execution as a thread. This thread executes sequentially until a parallelization directive for a parallel region is found. At this time, the thread creates a set of threads and becomes the master thread of the new group. All threads execute the statements until the end of the parallel region. All threads need to synchronize at the end of parallel constructs. The advantage of *OpenMP* is that an existing code can be easily parallelized by placing *OpenMP* directives instruction.

In our parallel *MPSO*, we design that the updating procedure for all sub swarms are executed on two threads, namely thread 0 and thread 1. Since the total number of sub-swarms is 5 and then thread 0 is aiming for the execution of swarm0 and swarm2; thread 1 for swarm1, swarm3, and swarm4.

```
Begin
    Initialize particles positions
    For each particle
        Find open facility vector (4)
        Evaluate (1)
        Divide the particles into Sub-populations
    For each subpopulation
    For each particle
        Do {
            Find the personal best and the global best
            Update velocity (7) and position (6) vectors
            Update open facility vector (4)
            Evaluate (1)
        } While (Termination)
End
```

Fig. 2. Pseudo code of *MPSO* algorithm for *UFL* problem.

III. Experimental Results

This experimental study has been completed in two stages; serial *MPSO*, parallel *MPSO*. Experimental results provided in this section are carried out with two algorithms over 15 benchmark problems that are taken from *OR* Library [20], a collection of bench marks for *OR* studies.

The benchmarks are introduced in Table 2 with their sizes and the optimum values. Although the optimum values are known, it is really hard to hit the optima for large size problems. Since the main idea is to test the performance of the proposed serial *MPSO* algorithm and parallel *MPSO* algorithm with *UFL* benchmarks, the results are provided in Table 2 with regard to solution quality indexes: Hit to Optimum Rate (*HR*).

HR provides the ratio between the number of runs yielded the optimum and the total numbers of the experimental trials. The higher *HR* the better quality of the solution is.

All algorithms were coded in C++. The serial MPSO algorithm was run on an Intel 2.66 *GHz* PC with 512*MB* memory. And parallel MPSO algorithm was run on an Intel Core Duo 1.83 *GHz* PC with 512*MB* memory.

For the serial and parallel *MPSO*, the number of the particles was set to 100, the number of subpopulation was 5, the size of the population was set to the fixed number 200, inertia weight was $w = 0.729$, the social and cognitive parameters were taken as $c_1 = c_2 = 9$, $c_3 = 0.065$, and the iteration number was 2000.

The performance of parallel *MPSO* algorithm seems much more impressive than serial *MPSO* with respect to the indexes of solution quality. *HR* index was higher for all experimental instances. And the best and worst obtained by parallel *MPSO* algorithm were better than the serial *MPSO* algorithm. Although for the large scale problems (capA, capB and capC),

even parallel *MPSO* still can not find the optimum listed in *OR* Library for its iteration is limited, but the best solutions get by parallel *MPSO* is better than by serial *MPSO*.

Compared with the serial *MPSO* algorithm, the parallel *MPSO* using *OpenMP* method can also saved more time. Thus the serial *MPSO* algorithm using *OpenMP* method overcame the heavy computation time disadvantage (see Fig.3).

TABLE II
EXPERIMENTAL RESULTS GAINED WITH SERIAL *MPSO* AND PARALLEL *MPSO*

			Serial *MPSO*			Parallel *MPSO*		
Problem	Size	Optimum	*HR*	Worst	Best	*HR*	Worst	Best
*cap*71	16×50	932615.75	1.00	932615.75	932615.75	1.00	932615.75	932615.75
*cap*72	16×50	977799.40	1.00	977799.40	977799.40	1.00	977799.40	977799.40
*cap*73	16×50	1010641.45	1.00	1010641.49	1010641.45	1.00	1010641.45	1010641.45
*cap*74	16×50	1034976.98	1.00	1034977.00	1034977.00	1.00	1034977.00	1034977.00
*cap*101	25×50	796648.44	0.73	797508.75	796648.44	0.77	797508.75	796648.44
*cap*102	25×50	854704.20	0.97	854704.19	854704.20	1.00	855971.75	854704.20
*cap*103	25×50	893782.11	0.53	895027.19	893782.11	0.53	895027.19	893782.11
*cap*104	25×50	928941.75	0.97	934587.00	928941.75	1.00	928941.75	928941.75
*cap*131	50×50	793439.56	0.10	801801.25	793439.56	0.07	800314.19	793439.56
*cap*132	50×50	851495.13	0.30	858537.56	851495.13	0.13	860138.06	851495.13
*cap*133	50×50	893076.71	0.17	901990.69	893076.71	0.33	899963.00	893076.71
*cap*134	50×50	928941.75	0.40	938630.56	928941.75	0.40	942121.38	928941.75
capA	100×1000	17156454.48	0.00	25244264.00	18555544.00	0.00	22328480.00	17535908.00
capB	100×1000	12979071.58	0.00	15138593.00	13448056.00	0.00	14871166.00	13347948.00
capC	100×1000	11505594.33	0.00	127927455.00	11895037.00	0.00	13196681.00	11955365.00

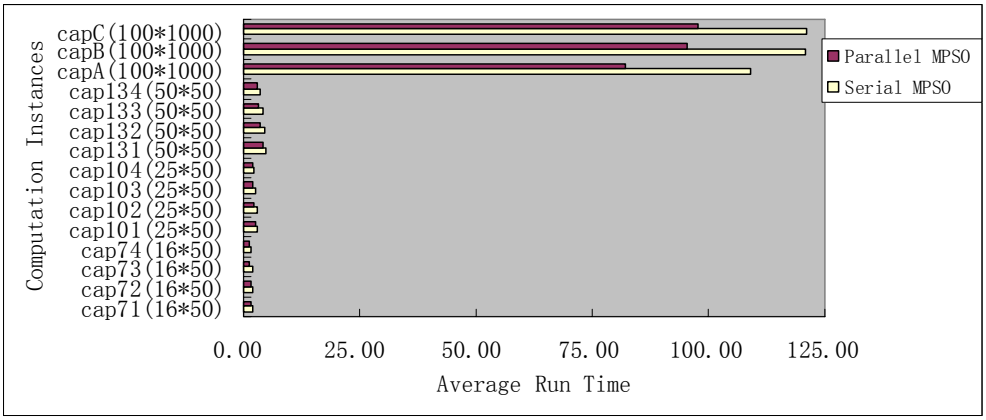

Fig 3. The comparison of average run time between Parallel multi-population *PSO* and Serial multi-population *PSO*

IV. CONCLUSION

In this paper, a parallel multi-population *PSO* using *OpenMP* method applied to *UFL* problems. The algorithm has been tested on several benchmark problem instances and optimal results are obtained in a reasonable computing time. The results of parallel *MPSO* are compared with PSO and serial *MPSO*. It is concluded that parallel *MPSO* is better than the compared methods and generating more robust results. In addition, compared with serial *MPSO*, parallel *MPSO* saved more time.

REFERENCES

[1] R. C. Eberhart and J. Kenned, "A new optimizer using particle swarm theory," *In Proc. Of the 6th International Symposium on Micro Machine and Human Science*. Piscataway , N. J ., USA, 1995. pp: 39-43

[2] C. Erick, "A survey of parallel genetic algorithms," *Calculateurs Parallels*, 1998, 10 (2) : 141-711.

[3] B. Bernd, E. K. Gabriel and S. Christine, *Parallelization strategies for the ant system*. Vienna , Austria : University of Vienna , 1997.

[4] S. W. Mahfoud and D. E. Goldberg, "Parallel recombinative simulated annealing: a genetic algorithm," *Parallel Computing*, 1995, 21 (1): 1-28.

[5] J. Zhao, W. Wang and X. J. Pan, "Parallel particle swarm algorithm & its application in hot rolling planning," *Computer Integrated Manufacturing Systems*, vol. 13, no. 4, pp. 698-703, 2007.(in Chinese)

[6] OpenMP Forum, http://www.openmp.org/, OpenMP: A Proposed Industry Standard API for shared Memory Programming, October 1997.

[7] G. Cornuéjols, G. L. Nemhauser and L. A. Wolsey, "The uncapacitated facility location problem," *Discrete Location Theory*, Wiley-Interscience, New York, pp. 119-171, 1990.

[8] P. B. Mirchandani and R. L. Francis, *Discrete Location Theroy Wiley-Interscience*, New York, 1990.

[9] A. Klose, "A branch and bound algorithm for an UFLP with a side constrain," *International Transactions in Operational Research*, vol. 5, no. 2, pp. 155-168, 1998.

[10] J. Barcelo, E. Fernandez and K. Jörnsten, "Computational results from a new Lagrangean relaxation algorithm for the capacitated plant location problem," *European Journal of Operational Research*, vol. 53, no. 1, pp. 38-45, 1991.

[11] D. Erlenkotter, "A dual-based procedure for uncapacitated facility location," *Operational Research*, vol. 26, pp. 992-1009, 1976.

[12] M. Körkel, "On the exact solution of large-scale simple plant location problems," *European Journal of Operational Research*, vol. 39, no. 2, pp. 157-173, 1983.
[13] K. S. Al-Sultan and M.A. Al-Fawzan, "A tabu search approach to the uncapacitated facility location problem," *Annals of Operations Research*, vol. 86, pp. 91-103, 1999.
[14] M. Laurent and P. V. Hentenryck, "A simple tabu search for warehouse location," *European Journal of Operational Research*, vol. 157, no. 3, pp. 576-591, 2004.
[15] J. H. Jaramillo, J. Bhadury and R. Batta, "On the use of genetic algorithms to solve location problems," *Computers and Operations Research*, vol. 29, no. 6, pp. 761-779, 2002.
[16] D. Ghosh, "Neighborhood search heuristics for the uncapacitated facility location problem," *European Journal of Operational Research*, vol. 150, no. 1, pp. 150-162, 2003.
[17] M. E. Aydin and T. C. Fogarty, "A distributed evolutionary simulated annealing algorithm for combinatorial optimization problems," *Journal of Heuristics*, vol. 10, pp. 269-292, 2004.
[18] M. Sevkli and A. R. Guner, "A continuous particle swarm optimization algorithm for uncapacitated facility location problem," *In Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS)*. Springer-Verlag, Berlin Heidelberg, pp. 316-323, 2006.
[19] Y. Shi and Eberhart R, "Parameter selection in particle swarm optimization," *In Evolutionary Programming VIZ: Pro. EP98*. Springer-Verlag, New York , pp.591-600, 1998.
[20] Brasley J E: OR-Library:http://people.runel.ac.uk/~mastjjb/jeb/info.html (2005).