Proceedings of Student-Faculty Research Day, CSIS, Pace University, May 5th, 2017

A Novel Approach for Library Materials Acquisition using Discrete Particle Swarm Optimization

Ana Wu and Daniel A. Sabol

Seidenberg School of CSIS, Pace University, Pleasantville, New York
Ana.wu@pace.edu Dsabol@pace.edu

***Abstract*—The academic library materials acquisition problem is a challenge for librarian, since library cannot get enough funding from universities and the price of materials inflates greatly. In this paper, we analyze an integer mathematical model by considering the selection of acquired materials to maximize the average preference value as well as the budget execution rate under practical restrictions. The objective is to improve the Discrete Particle Swarm Optimization (DPSO) algorithm by adding a Simulate Annealing algorithm to reduce premature convergence. Furthermore, the algorithm is implemented in multiple threaded environment. The experimental results show the efficiency of this approach.**

Keywords: Library Material Acquisition, Discrete Particle Swarm Optimization, Simulated Annealing, Multithreading, Algorithm Design.

I. INTRODUCTION

With the rapid development of technology, the Internet has become a necessary part of everyday life. Innumerable electronic books, publications, and teaching videos are accessible everywhere on the Internet with little to no cost. Meanwhile, with the popularization of smart phones and e-book readers, it is convenient for users to access the electronic materials anywhere and anytime. With these changes, the traditional library turns out to be not necessary for students. Consequently, library administrators prefer to build library with better design or more equipment, but not spend money to purchase books [2].

Meanwhile, in the past decades, the prices of books have increased dramatically. According to a study by the Student Public Interest Research Group, textbook prices in the U.S. have climbed 864% since 1978, compared to 257% rise in consumer price index (CPI) [10]. Due to the unreasonable prices, a number of universities cannot purchase sufficient book, and they are only affording textbooks for courses, which will be borrowed and read by students.

Moreover, the budgets to purchase materials decreased, and this kind of dilemma happens everywhere. According to the American Research Libraries (ARL) [2], as a percentage of overall university expenditures, libraries have been steadily losing ground. The percentage of university funds spent on libraries has declined over the past 30 years from a high of 3.7% in 1984 to just under 1.8% in 2011. Referring to this trend, the budgets of libraries will keep decreasing in future. What's worse, libraries have to spend a large portions of their budget on the maintenance cost of buildings and equipment.

Finally, the requirements of books vary a lot based on majors. Some hot majors, such as computer science, may require more books for students. Students may not only need textbooks, but also need some hand-on books for their better study. As a result, it is extremely important for libraries to find out the method of purchasing materials with limited cost while benefit more students.

Consequently, with the growth of electronic resources, and the shrinking of library budgets, the price inflation of library materials, and the uneven distribution of different majors, the Library Materials Acquisition (LMA) problem has become a challenge for librarians. Similarly, the academic library of Pace University is encountering similar pressure from LMA. Sabol[1] notes that "as an Academic Librarian seeing the trends directly, we will continue to have major shifts in our models of purchase and selection. This will also be seen as open access to items will become available which will decrease the library's value".

II. PROBLEM STATEMENT

The academic library is positioned to acquire materials for multiple majors, including Business, Sciences, Nursing,

[1] Sabol, D.A. (2016) Evening & Weekend Reference Librarian. Mortola Library – Pace University

Education, etc, with different budgets and requirements. Each individual librarian has their own budget max limitation for their preferred materials. Meanwhile, each individual department has a *preference value*, ranges from 0 to 1 inclusively for each material indicating the interests of departments. Higher *preference value* means that the department has higher interest in the material. The library also should take overlap situations into consideration. Since one material may be required by many departments, the library is able to arrange their funds in a proportionate allocation. In this case, the acquisition cost should be apportioned by all recommending departments in proportion to their *preference values* for specific materials. Meanwhile, higher *preference value* for the material also indicates that the department is willing to spend more cost for the material, vice versa. From the perspective of library, to meet the various requirements from all the departments and to balance the amounts of materials in each major in the library, each material belongs to a specified category, which is limited to a range by the acquisition librarian.

From the view of each department, they expect higher *average preference values* representing the satisfaction to the result, based on the decision made by each librarian. Nevertheless, the main purpose of academic library is to satisfy all departments' requirements and to support student assignments. The processing includes two aspects. The first one is to determine that which materials should be required and the amounts that each department should pay for each material. The second one is to meet the constrains of material amounts in each category and budgets of each department.

The problem solved in this paper is to help librarian get higher *average preference values* for all materials, while expend all available funds. Otherwise, a low budget execution rate may lead to an amount cut in next purchase periods. The aim of this paper is to help acquisition librarian select materials to be acquired in order to maximize the *average preference value* as well as the budget execution rate under the restrictions, which are departmental budgets and the amount of materials in each category.

In the view of computation complexity theory, the library materials acquisition problem is a generalized version of the knapsack problem which is NP-hard. Currently, many heuristic optimization algorithms are proposed to solve NP-hard problems, such as Simulated Annealing, Tabu Search, and Particle Swarm Optimization algorithm. These algorithms work by finding approximate optimal solutions with limited time and impressive performance. In this project, we will use those three algorithms to solve the LMA problem and compare their results and performances.

III. Literature Review

Over the past few decades, researches on LMA problem have been designed and implemented with a number of models and approaches. Dating back to 1983, Beilby and Mott Jr. demonstrated a lexicographic linear goal programming methodology to solve the allocation problem [1]. In 1996, Kenneth Wise applied the lexicographic linear goal programming methodology to a practical project which contains 90 funds representing books and periodicals in 45 subject disciplines at the University of Tennessee, Knoxville, and resulted in the successful distribution of $3.5 million while taking into consideration ten goals or variables ranging from circulation to number of faculty and students [11]. Then in 2000, Kenneth Wise and Perushek continued to improve the goal programming mode by taking more goals into account and used to illustrate the solution of a library acquisition allocation problem [12].

To solve the LMP problem and meet the practical requirement, which is to obtain a good solution within a reasonable time, Tsu-Feng Ho applied simulated annealing, genetic algorithm, and tabu search. Their goal was to select materials to achieve a maximum total preference under the constraints, in which the acquisition for each category of material is deter-mined by a predefined budget [4]. Later, in 2010, Tsu-Feng Ho presented a mode that maximized the average preference values of acquired subjects and first formulated the problem by means of mathematical programming. Then they designed a Particle Swarm Optimization (PSO) to resolve the problem, and at last got the result which showed the algorithm can optimally solve problems within a reasonable amount of time [5].

Yi-Ling Wu and Tsu-Feng Ho improved the particle swarm optimization algorithm by designing an initialization algorithm and a penalty function, and employing scout particles to enhance the exploration within the solution space. They presented an integer programming model of the LMA problem by considering how to select materials in order to maximize the average preference and the budget execution rate under some practical restrictions, including departmental budget and limitation of the number of materials in each category and language. The results showed that the proposed PSO is an effective approach for the problem [13].

PSO is an evolutionary computation scheme originally created in 1995 by James Kennedy [7]. It was an algorithm modeled on swarm intelligence to find a solution to an optimization problem in a search space or model and predicts social behavior in the presence of objectives. Typical examples include problems which require the ordering or arranging of discrete elements, as in scheduling and routing problems. For those discrete optimization problems, Kennedy and Eberhart introduce a binary particle swarm optimization, DPSO, in 1997 [8]. In DPSO, they change the concept of velocity from adjustment of the position to the probability that a bit in a solution is 1.

Furthermore, there a number of research working on multi-threading in optimization. Kuo-Yang Tu and Zhan-Cheng Liang developed an exact PSO model whose particles simultaneously interact with each other [14]. They separated particles into several subgroups, and implemented the communication among the subgroups by parallel computation modes. Samuel Williams et al. [15] examined sparse matrix-vector multiply on fields of multicore designs. They also presented several optimization strategies especially effective for the multicore environment. We share similar abstract mode with them to analyze LMA problem. To increase the performance, we apply multithreading implementation into our algorithm design.

IV. Problem Formulation

In mathematical modeling for this problem, we refer to the integer modeling built by Yi-Ling Wu [9]. At the Mortola Library of Pace University, we have *m* departments and *n* materials. Each material *i* $(i \in [1, n])$ is associated with a cost c_i and a preference value p_{ij} recommended by each department *j* $(j \in [1, m])$ and ranges from 0.0 to 1.0. If a material is acquired by more than one departments, the acquisition cost would be apportioned by all recommending departments in proportion to their preference values. For instance, if a material with cost 100 is acquired by two department, Education and Business, with preference 0.3 and 0.9 respectively, each department should pay 25 (100*(0.3/(0.3+0.9))), and 75 (100*(0.9/(0.3+0.9))) respectively. Therefore, each department *j* has an actual cost a_{ij} in material *i*. Each department *j* also owns an amount B_{ij} of budget limitation. No budget exceeding its limits is accepted by any department. Meanwhile, each material is specified to a category and a specific category should be restricted into a range to meet the acquisition requirements from all the departments and to balance the amounts of materials in each major in library. Considering that we have a set of *q* categories, and each category *k* $(k \in [1, q])$ is associated to CU_k, which is the upper bound on the number of materials in category *k*, and CL_k, which is the lower bound on the number of materials in category *k*. For material *i*, b_{ik} denotes if material *i* belongs to category *k*. If b_{ik}=1, then material *i* belongs to category *k;* b_{ik}=0 otherwise. For material *i* and department j, x_{ij} denotes if material *i* is acquired by department *j* from which the cost will be charged. If x_{ij}=1, then department *j* will be charged by material *i*; x_{ij}=0 otherwise.

As discussed in Section II, the objective is to select materials to be acquired in order to maximize the average preference value as well as the budget execution rate under the constraints. The objective function is mathematically formulated as the following:

$$O(x) = \rho * \left(\frac{\sum_{j=1}^{m} \left(\sum_{i=1}^{n} x_{ij} p_{ij} / \sum_{i=1}^{n} x_{ij} \right)}{m} \right) + (1-\rho) * \left(\frac{\sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} a_{ij}}{\sum_{j=1}^{m} B_j} \right) \qquad (1)$$

In Equation (1), ρ is a float positive number which ranges from 0.0 to 1.0, inclusively, to control the importance between maximum average preference value and maximum budget execution rate.

In this problem, constrains include the budget limitation of each department and the amount limitation of each category. The mathematical formulas are as follows:

$$\sum_{i=1}^{n} a_{ij} \le B_j \quad \text{for } 1 \le j \le m \qquad (2)$$

$$a_{ij} = \left(\frac{x_{ij} p_{ij}}{\sum_{\hat{j}=1}^{m} x_{i\hat{j}} p_{i\hat{j}}} \right) * c_i \quad \text{for } 1 \le i \le n,\ 1 \le j \le m \qquad (3)$$

$$\sum_{i=1}^{n} \left(\sum_{j=1}^{m} x_{ij} \right) b_{ik} \le CU_K \quad \text{for } 1 \le k \le q \qquad (4)$$

$$\sum_{i=1}^{n} \left(\sum_{j=1}^{m} x_{ij} \right) b_{ik} \ge CL_K \quad \text{for } 1 \le k \le q \qquad (5)$$

Equation (2) confines the budget limitation constrain for each department. Equation (3) shows the actual expenses of material *i* apportioned by department *j* according to the proportion of the preference. Equation (4) and (5) are used to abide by the lower bound and upper bound specified on the number of materials in each category.

As discussed in Section III, each particle has a fitness value which is calculated by the objective function and constrains. This value indicates how well the solution solves the problem. In this problem, constrains are depicted by penalty function which is defined as following,

$$\begin{aligned} C(x) = {} & \sum_{j=1}^{m} \max\{0, \frac{\sum_{i=1}^{n} x_{ij} e_{ij} - B_j}{B_j}\} \\ & + \sum_{k=1}^{q} \max\{0, \frac{\sum_{i=1}^{n} z_i b_{ik} - CU_k}{|\sum_{i=1}^{n} z_i b_{ik} - CU_k|} \\ & + \sum_{k=1}^{q} \max\{0, \frac{CL_k - \sum_{i=1}^{n} z_i b_{ik}}{|CL_k - \sum_{i=1}^{n} z_i b_{ik}|} \end{aligned} \qquad (6)$$

in which z_i denotes whether material *i* is acquired. z_i = 1 means at least one department required material i, otherwise, no department requires it. Equation for z_i is shown as below:

$$\begin{aligned} & \sum_{j=1}^{n} x_{ij} \ge 1 \Rightarrow z_i = 1 \\ & else \Rightarrow z_i = 0 \end{aligned} \qquad (7)$$

Fitness value is calculated by the objective function and penalty function as Equation (8). In the feasible solution which meets all constrains, penalty value must be 0, so that the fitness value is exactly the same as objective value. Otherwise, in other infeasible solution, the penalty value is larger than 0, so that the fitness value will be smaller than its objective value.

$$F(x) = O(x) - C(x) \tag{8}$$

V. ALOGRITHM FORMULATION

In this section, we present the formulas of the PSO algorithm and the DPSO algorithm.

A. *PSO Formulation*

In PSO, a candidate solution is represented as a particle with position P in a D-dimensional space. In each step of iteration, each particle has a position. For particle s, P_s^t denotes the solution found by particle s at iteration t. V_s^t denotes the velocity of particle s in iteration t, where velocity represents a change in the position. Each particle s also maintains its "pbest", which is introduced in Section III, representing the position of its previous local best performance in a vector. The "nbest" represents the best previous position of any particle in the neighborhood of s, called neighborhood best. Neighborhoods can be defined in numerable ways, and most implementations prefer two ways. The first way is that evaluating a particle i in a neighborhood consisting of itself, particle $i-1$, and particle $i+1$, with arrays wrapped so $i=1$ is beside $i=N$ [13]; the second way is that evaluating all the particles in the same neighborhood, which means "nbest" is the global best one. An iteration comprises evaluation of each particle, then stochastic adjustment of its velocity in the direction of "pbest" and "nbest". Thus in the original PSO, velocity and position of particle s in iteration $t+1$ can be calculated by following formula:

$$V_s^{t+1} = V_s^t + c_1 r_1 (pbest - P_s^t) + c_2 r_2 (nbest - P_s^t) \tag{9}$$

$$P_s^{t+1} = P_s^t + V_s^t \tag{10}$$

In Equation (9), c_1 and c_2 are positive numbers. c_1 denotes the cognition learning rate which means the influence rate of individual experience, and c_2 denotes the social learning rate which means the influence rate of neighbors' experience. Meanwhile, r_1 and r_2 are random positive float numbers generated for each particle and range from 0.0 to 1.0. If position of particle s in iteration t , P_s^t, is less than its local "pbest", a positive number $c_1 * r_1$ will be added into velocity. Similarly, if position of particle s in iteration t is less than neighborhood "nbest", a positive number will be added into velocity. In Equation (10), if velocity increases, the position of particle will be closer to "pbest" and "nbest". Therefore, if $c1$ and $c2$ are set relatively high, the particles seem to be sucked into the current best solution quickly. If $c1$ and $c2$ are set relatively low, the particles seem to swirl around the goal, then realistically approaching it.

The PSO algorithm also limits velocity of particle by a value V_{max}. The velocity of each particle is kept with the range [$-V_{max}$, V_{max}]. V_{max} parameter needs to be setup carefully since it influences the balance between exploration and exploitation.

Specifying a high V_{max} increases the range explored by a particle. To better balance the exploration and exploitation, several variants of PSO algorithm have been proposed [9]. A widely used method is to better control the scope of the search to reduce the importance of V_{max}. For this purpose, an *inertia weight (w)* to the velocity was introduced by Eberhart and Shi [12]. The Following modification of Equation (9) is proposed:

$$V_s^{t+1} = wV_s^t + c_1 r_1 (pbest - P_s^t) + c_2 r_2 (nbest - P_s^t) \tag{11}$$

The steps to implement PSO algorithm are listed as below:

1. Initialize position and velocity for each particle randomly;
2. Start loop
   a) For each particle, evaluate the fitness value;
   b) For each particle, compare the fitness value with its "pbest". if a better solution occurs, update "pbest";
   c) For each particle, identify the best neighbor, get the "nbest", and update the velocity by Equation (3);
   d) For each particle, update the position by Equation (2). Check all positions in this step, if there is a position better than current global best, update global best.
   e) If a criterion is met, such as the maximum number of iterations, exit loop;
3. End loop

B. *DPSO Formulation*

For discrete optimization problems, a binary particle swarm optimization (DPSO) was purposed by Kennedy and Eberhart. DPSO changes the concept of velocity from adjustment of the position to the probability that a bit in some solution will be 1. The velocity is squashed in sigmoid function as shown below:

$$S(V_s^t) = \frac{1}{1 + e^{-(V_s^t)}} \tag{12}$$

The equation to get position for each particle is updated as below:

$$\begin{aligned} &if\ (random() < S(V_S^t)) then\ P_S^t = 1; \\ &else\ P_S^t = 0; \end{aligned} \tag{13}$$

By Equation (12) and (13), $s(V_s^t)$ is a float value which ranges from 0.0 to 1.0. Smaller $s(V_s^t)$ *value* means low probability to be 1.0, larger $s(V_s^t)$ value means high probability to be 1.0. For example, if $s(V_s^t) = 0.1$, there is a ten percent chance that the bit will is 1.0, and ninety percent chance it is 0.

In PSO, a high V_{max} increases the range explored by a particle, but the situation in the DPSO is opposite. Smaller V_{max} allows a mutation rate in DPSO. For instance, when velocity equals 6.0, the $s(V_s^t)$ will be 0.9975. If V_{max} is largely higher than 6.0, the position value P_s^t will almost always be 1.0 after velocity bigger than 6.0. As a result, a smaller V_{max} is more preferable to DPSO.

VI. Implementation

This section details how to tackle the library materials acquisition problem by discrete particle swarm optimization algorithm, and how to combine simulated annealing into DPSO algorithm to avoid the premature convergence problem, which is a challenging problem faced by DPSO algorithms through the optimization process.

A. *Motivational Example*

For easily understanding the problem and the solution, we present a simple example to describe the whole citations as following. Suppose we have five materials (Book1 to Book5), three departments (Computer Science, Business, and Art), and three categories (Science, Art and Social). The input data will be formatted as following structures.

Table 1. Cost and category for each material

Materials	Book1	Book2	Book3	Book4	Book5
Cost	\$100	\$45	\$70	\$60	\$38
Category	Science	Art	Science	Art	Social

Table 2. Budget for each department

Department	Computer science	Business	Art
Budget	\$550	\$880	\$660

Table 3. Preference value for each material from each department

Preference value	Computer science	Business	Art
Book1	0.7	0.3	0
Book2	0	0.4	0.5
Book3	0.4	0.7	0.6
Book4	0.5	0	0.9
Book5	0.1	1	0.3

In Table 1, each book is associated with a cost and a specific category. We can use two arrays to represent the costs and categories for all materials respectively. In Table 2, each department holds a budget. We can also use an array to represents the budget for each department. In Table 3, each department has a preference value, which ranges from 0.0 to 1.0 inclusively, for each material to indicate the interest for each material. We can use a 5 * 3 float matrix to represent. All the arrays and matrix mentioned above can be shown in data structure as following:

```
float[] cost = {100, 45, 70, 60, 38};
int category = {0, 1, 0, 1, 2};
int budget = {550, 880, 660};
float[][] preference = {{0.7, 0.3, 0},
                        {0, 0.4, 0.5},
                        {0.4, 0.7, 0.6},
                        {0.5, 0, 0.9},
                        {0.1, 1, 0.3}};
```

Note that cost array, budget array and preference matrix correspond to c_i, B_j, and p_{ij} which was mentioned in Section IV respectively. Category array is also related to c_{ik} which is also mentioned in Section IV. We can use category array to get the value of c_{ik}. For example, in category array above, the category of Book1 is signed as 0, which means Book1 belongs to category 0. Therefore, for Book1, we can get c_{00} equals to 1, c_{01} and c_{02} all equal to 0.

The solution, also depicted as the position of each particle, for above example can be represented by a 5 * 3 binary matrix which is shown as following.

```
int[][] position = {{0, 1, 0},
                    {1, 0, 0},
                    {0, 1, 1},
                    {1, 1, 0},
                    {0, 0, 0}};
```

Each entry in the position matrix indicates whether material *i* is acquired by department *j*. Note that each entry corresponds to the decision variable x_{ij} which was mentioned in Section IV.

As mentioned in Section IV, the position of each particle represents one candidate solution. Therefore, the position in DPSO algorithm for this simple example can be represented by the two-dimension array, *position*, as well. Meanwhile, the velocity for each particle can also be represented by a 5 * 3 float matrix, *velocity*. For example, the velocity matrix may be the following:

```
float[][] velocity = {{0.8, 1.5, 4.0},
                      {-1.5, 0.0, 4.0},
                      {-4.0, 4.1, 2.0},
                      {3.1, -1.1, 2.0},
                      {-3.0, 5.0, 2.0}};
```

Using Equation (12) and (13), we can calculate the possibility of each entry to be 1 based on $postion_{i,j}$ and $velocity_{i,j}$.

B. *Algorithm Initialization*

The initial velocity is generated randomly for each particle. Meanwhile, the initial position, which is also a feasible solution, is generated by following steps for each particle.

1. Let $k = 1$. Randomly select a material i in the k_{th}.
2. If the material i is acquired, select another one.
3. If the material i is not acquired, randomly select a department j. Check whether *position[i][j]* equals to 1.
4. If *position[i][j]* does not equal to 1, set position[i][j] to 1.
5. If *position[i][j]* equals to 1, go to step 3 to select another j.
6. If there is no j left, go to step 1 and add k by 1.

C. *Avoiding premature convergence*

The drawback of PSO algorithm is that the particle swarm may prematurely converge. In DPSO, each particle moves iteratively by following the best solution found by itself and its neighborhoods. As a result, all particles may converge to the current best solution, which is a local optimal solution in another word.

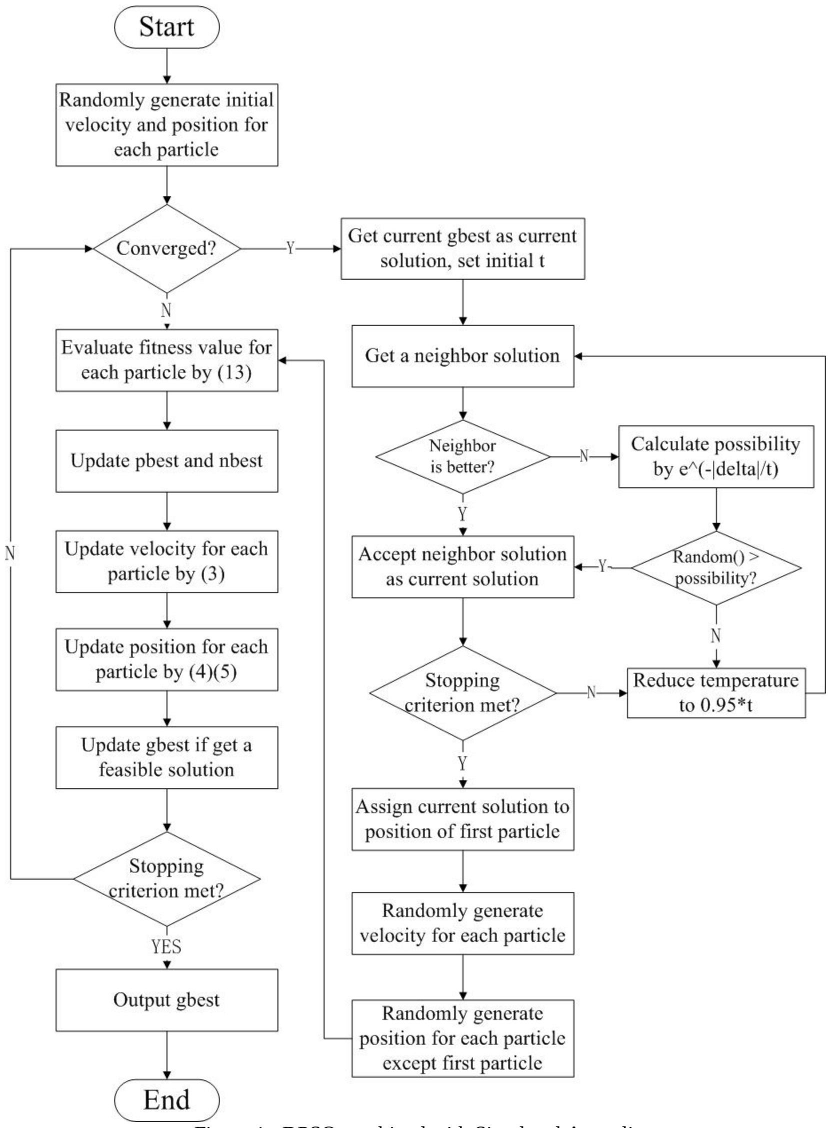
Start
Randomly generate initial velocity and position for each particle
Converged?
Y
N
Get current gbest as current solution, set initial t
Evaluate fitness value for each particle by (13)
Get a neighbor solution
Update pbest and nbest
Neighbor is better?
N
Calculate possibility by e^(-|delta|/t)
Y
Update velocity for each particle by (3)
Accept neighbor solution as current solution
Y
Random() > possibility?
N
Update position for each particle by (4)(5)
Stopping criterion met?
N
Reduce temperature to 0.95*t
Y
Update gbest if get a feasible solution
Assign current solution to position of first particle
Stopping criterion met?
YES
Randomly generate velocity for each particle
Output gbest
Randomly generate position for each particle except first particle
End

Figure 1. DPSO combined with Simulated Annealing

Whether the particle swarm is converged can be checked by the current velocity for each particle. If the particle swarm is converged, each dimension in velocity of each particle nearly equals to V_{max} or $-V_{max}$. Once all dimensions in all velocities are the same and equal to V_{max} or $-V_{max}$, all positions will be the same as the best solution we have got thus far. In this situation, all particles are converged to one single particle and trapped into a current best solution. There is no diversity in each individual particle, and all particles keep the same solution point.

To avoid premature convergence, we combine a Simulated Annealing (SA) algorithm into DPSO to enhance the exploration range. The basic concept is to use the best solution we get from DPSO as an initial solution of SA. Then we use the new best solution obtained from SA as initial position of DPSO.

The basic procedure is depicted in Figure 1. When convergence occurs, we use the current global solution, “gbest”, as the initial solution for SA algorithm. Suppose that the result solution we get from SA algorithm is “sabest”. After SA ends, we assign “sabest” to the position of the first particle. Meanwhile, we reset the positions and velocities by randomly generating positions for all particles except the first particle, and randomly generating velocities for all particles. The whole procedure consists of cycles of move, converge, and dispatch, until meeting the stopping criteria. The stopping criteria is the iteration times of DPSO and SA. It effects the performance of our approach, and we will set different values to it in the experiments.

D. Multithreading

To improve the performance of optimization, we implement our approach into multithreading environment.

We create several threads to run particles' self execution, including generate and update the velocity and position of itself, evaluate the fitness value, calculate possibility, and run SA program. In the main process, we check whether the whole algorithm is converged, dispatch tasks to threads, broadcast current best solution to threads, and control the iterations of DPSO+SA.

First, each thread randomly generates initial velocity and position for the particles arranged to it. Then the main process summarizes the solutions from all the threads to get the current best one, and broadcast the initial t to all the threads.

Each particle evaluates the fitness value and updates the "pbest", "nbest", velocity, and position value based on the information sent from the main process. Each particle keeps doing this process until get stopping notification from main process. The main process checks whether the algorithm is converged every round threads finishing their calculation.

Once the main process finds the algorithm converged, it will notify threads not to do DPSO, but SA algorithm. The SA algorithm is run at threads, not in form of particle. Each thread takes the same solution from DPSO as the input to run SA algorithm, but they may get different results. The main process selects the best one after collecting all the results from threads to be the initial input for the next round DPSO.

VII. Experimental Setup and Result

The values of the parameters in DPSO were set as particle amount = 50, inertia weight for velocity = 1.0, cognition learning rate c_1 = 2.0, social learning rate c_2 = 2.0, max velocity V_{max} = 6.0. Iteration times varies from case to case. The stopping criteria were defined as completing defined iteration times. In the experiment, datasets with different sizes were tested, as shown in Table 6. Each dataset was tested 50 times in both a purposed algorithm and standard DPSO algorithm. All the programs were implemented in Java and run on a PC with an Intel Core i7-4810MQ 2.80GHz CPU and 16G RAM. We recorded the average objective values and program execution times for all three datasets in each algorithm, which are shown in Table 7.

First, we used exhaustive algorithm to get the theoretically optimal results for these three cases, which are also listed in Table 7. As shown in Table 7, DPSO+SA spent less time and more optimal than DPSO in Case I. Meanwhile, in Case II and III, using DPSO+SA can always obtain more optimal solution than using DPSO alone, but the execution time of DPSO+SA is a bit higher than that of using DPSO.

Table 6. Three Test Data Sets

Case I	20 materials, 3 departments, 3 categories
Case II	50 materials, 3 departments, 3 categories
Case III	100 materials, 10 departments, 10 categories

Table 7. The Average Performance of DPSO and DPSO+SA

Datasets	DPSO		DPSO+SA		Optimal
	Objective value	Execution time (ms)	Objective value	Execution time (ms)	Objective value
Case I	0.869009	616.92	0.893129	505.98	0.91985
Case II	0.834884	1275.94	0.840621	1496.7	0.84892
Case III	0.810837	6799.98	0.858569	7143.18	0.87275

Then we implement our DPSO+SA into multithreading environment. Table 8 shows the results of running DPSO+SA in single thread and multiple threads.

Table 8. The Performance of running DPSO+SA in single and multiple threads

Datasets	Single Thread		Multiple Threads	
	Result	Time (ms)	Result	Time (ms)
Case I	0.893129	505.98	0.895892	451.26
Case II	0.840621	1496.7	0.847823	1298.55
Case III	0.858569	7143.18	0.860986	6564.87

Compared with the optimal result, as shown in Table 7, the objective values of DPSO+SA are 47.43%, 40.87%, 34.7% higher than those of DPSO alone. The object values of DPSO+SA in multithreading are 10.34%, 8.68%, and 17.04% closer to the optimal ones than in single thread. As shown in Figure 2, DPSO+SA was proven with higher performance than DPSO alone in most situations. In Case II and III, the results of using DPSO+SA are more stable than using DPSO alone, and they are closer to the theoretically optimal result those of using DPSO.

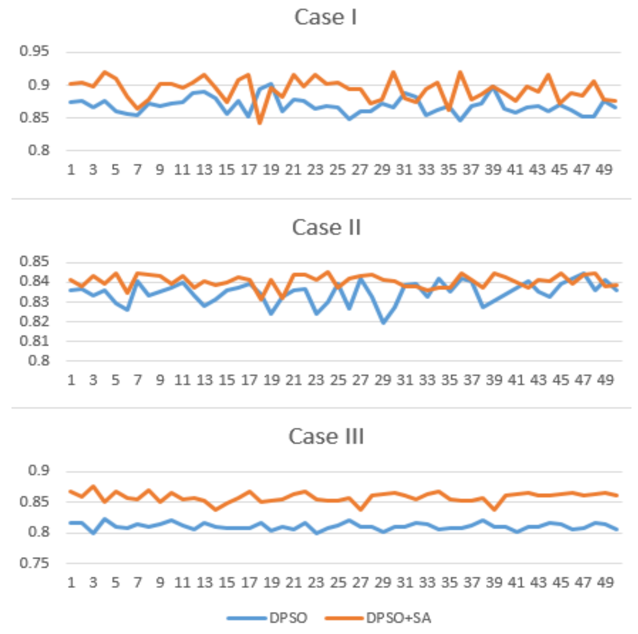

Figure 2. The Comparison Results of DPSO and DPSO+SA in three Cases

Then we compared the performance of running DPSO+SA in single thread and multiple threads. The results are shown in Figure 3. Running DPSO+SA in multithreading environment is more stable than that in single thread.

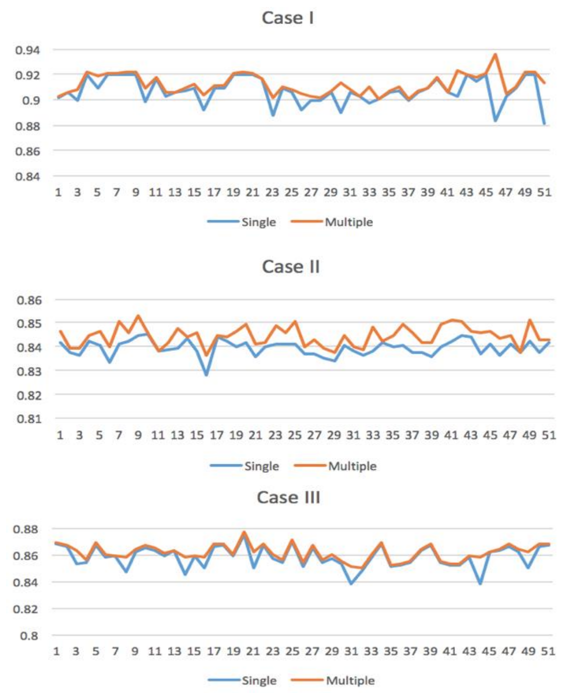

Figure 3. The Performance of running DPSO+SA in single and multiple threads under three Cases

Figure 4 shows the execution time of running DPSO+SA in single and multiple threads. From the comparison, we can obviously get that running in multithreading environment is faster than running in single thread. Furthermore, with the increasing of complexity, the execution time of running in multithreading is more stable than that in single thread.

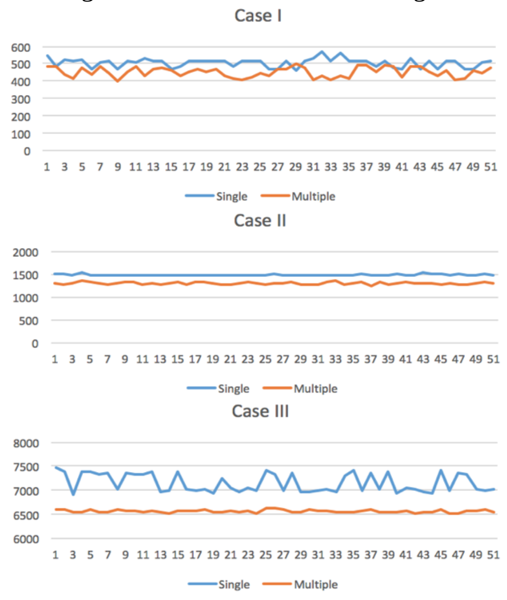

Figure 4. The Execution Time of running DPSO+SA in single and multiple threads under three Cases

VIII. Conclusion

In this paper, we proposed a combinational algorithm combining standard DPSO with SA algorithm to solve the LMA problem. Furthermore, we implemented our approach in multithreading environment to improve the efficiency. Three test data sets were conducted to demonstrate the effectiveness of our proposed algorithm. The experimental results show that our approach reached better solution and lower execution time than DPSO algorithm and DPSO+SA in single thread.

IX. References

[1] M. H. Beilby and T. H. Mott, "Academic library acquisitions allocation based on multiple collection development goals," *Computers & Operations Research*, vol. 10, pp. 335--343, 1983.

[2] P. DAVIS, "Libraries Receiving a Shrinking Piece of the University Pie," 2012.

[3] R. C. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *evolutionary computation, 2001. Proceedings of the 2001 Congress on*. vol. 1, ed, 2001, pp. 81--86.

[4] T.-F. Ho, S. J. Shyu, and Y.-L. Wu, "Material acquisitions in academic libraries," in *Asia-Pacific Services Computing Conference, 2008. APSCC'08. IEEE*, ed: IEEE, 2008, pp. 1465--1470.

[5] T.-F. Ho, S. J. Shyu, B. M. Lin, and Y.-L. Wu, "An evolutionary approach to library materials acquisition problems," in *Intelligent Systems (IS), 2010 5th IEEE International Conference*, ed, 2010, pp. 450-455.

[6] http://www.arl.org/, "Library Expenditure as \% of Total University Expenditure," 2013.

[7] J. Kenndy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*. vol. 4, ed, 1995, pp. 1942--1948.

[8] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*. vol. 5, ed, 1997, pp. 4104--4108.

[9] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, pp. 33-57, 2007.

[10] N. Tafuri. (2014). *Library Materials Price Index*. Available: http://www.ala.org/alctsnews/features/librarymaterials2014

[11] K. Wise and D. Perushek, "Linear goal programming for academic library acquisitions allocations," *Library Acquisitions: Practice & Theory*, vol. 20, pp. 311-327, 1996.

[12] K. Wise and D. Perushek, "Goal programming as a solution technique for the acquisitions allocation problem," *Library \& Information Science Research*, vol. 22, pp. 165--183, 2000.

[13] Y. L. Wu, T. F. Ho, S. J. Shyu, and B. M. Lin, "Discrete particle swarm optimization with scout particles for library materials acquisition," *ScientificWorldJournal*, vol. 2013, p. 636484, 2013

[14] Tu Kuo-Yang, and Zhan-Cheng Liang. "Parallel computation models of particle swarm optimization implemented by multiple threads." Expert Systems with Applications 38.5 (2011): 5858-5866.

[15] Williams, Samuel, et al. "Optimization of sparse matrix–vector multiplication on emerging multicore platforms." Parallel Computing 35.3 (2009): 178-194.