Efficient Swarm Algorithms to Constrained Resource Allocation

Peng-Yeng Yin
Department of Information Management
National Chi Nan University
Nantou, Taiwan
pyyin@ncnu.edu.tw

Jing-Yu Wang
Department of Information Management
National Chi Nan University
Nantou, Taiwan
wjykino@hotmail.com

Abstract—**Resource management is the effective deployment for an organization's resources. It deals with classification, allocation, stocking, processing, storage, and valuation for the shared resources when they are needed. Among many managerial tasks, the constrained resource allocation problem has challenging many practitioners involved in manufacturing operations. This problem seeks to find an optimal allocation of a limited amount of resource to a number of manufacturing activities for optimizing organization's objective subject to the resource constraints. Most existing methods use mathematical programming techniques, but they are defaced in deriving exact solutions for large-scale problems with reasonable time. A viable alternative is to use swam algorithms which can obtain approximate solutions to real-world intractable problems. This paper presents two swarm algorithms embodying the adaptive resource bound technique for conquering the constrained nonlinear resource allocation problem. Experimental results manifest that the proposed methods are more effective and efficient than a genetic algorithm-based approach. The convergence behavior of the proposed methods is analyzed by observing the variations of population entropy. Finally, a worst-case analysis is conducted to provide a reliable performance guarantee.**

Keywords-particle swarm optimization; ant colony optimization; metaheuristic; resource allocation

I. Introduction

Resource management is the central process to ensure the organization's competence against its rivals. It deals with classification, allocation, stocking, processing, storage, and valuation for an organization's resources when they are needed. Among these tasks, *resource allocation problem* (RAP) has been targeted by many researchers due to its everlasting importance. The constrained RAP seeks for an optimal allocation of resources to a number of activities for optimizing organization's objective subject to operational constraints. For instance, project budgeting [1] allocates a given amount of money to a number of projects for maximizing the net present value (NPV) or internal rate of return (IRR), software testing [2] allocates a number of programmers with varying skills to achieve maximum reliability of the software, task allocation [3] allocates a given number of program modules to a number of processors for minimizing the incurred cost, health care financing [4] allocates a fixed amount of medical resource across competing programs promising improved health for patients. For a comprehensive survey the reader is referred to [5].

There exist many solution methods for tackling distinct versions of RAP. We briefly summarize them as follows. (1) Integer linear programming [6] and mixed integer linear programming [4] have been used to formulate and solve the linear RAP with discrete or continuous resource. (2) Basso and Peccati [1] proposed a dynamic programming (DP) algorithm with an efficient pruning procedure for solving the portfolio optimization problem in project financing. Morales et al. [7] presented three parallel DP algorithms using pipeline, dominancy, and resource parallelism to conquer the curse of dimensionality. (3) Bretthauer and Shetty [8] solved the RAP subproblems using a branch-and-bound tree via a one-dimension search for the optimal Lagrange multiplier of the constraint. Bretthauer and Shetty [9] further improved the algorithm by incorporating the pegging method, which iteratively reduces the size of the relaxed subproblems containing variables not satisfying their bounds, for solving the problem more efficiently. (4) Since exact algorithms could be very time expensive for large-scaled problems, an alternative for solving the RAP is to find approximate solutions with reasonable computational time. Dai et al. [2] proposed a genetic algorithm for allocation of software testing resource. The chromosome is represented by a list of modular testing times to be allocated and the objective is to maximize the system reliability with the minimum testing cost. Hou and Chang [10] presented a genetic algorithm for allocating a number of products among plants such that the incurred production cost is minimized.

The motivations of this research are two-fold. *First*, we observe that most of existing methods for tackling the RAP are based on mathematical programming techniques which may fail to derive exact solutions with reasonable time for problems of large size. For the RAP practitioners, they would like to obtain a feasible and quality, although not optimal, solution when the problem size is large. An alternative for obtaining approximate solutions is using metaheuristic algorithms. *Second*, the development of metaheuristic computation has been flourishing during the last decade. Many metaheuristic paradigms such as genetic algorithm (GA) [11], simulated annealing (SA) [12], tabu search (TS) [13], ant colony optimization (ACO) [14], and particle swarm optimization (PSO) [15] have been applied to solve benchmark NP-hard problems. Encouraged by their successful applications, we investigate the feasibility of using metaheuristic algorithms for solving the RAP.

The remainder of this paper is organized as follows. Section 2 formulates the addressed RAP problem. Section 3 presents the proposed swarm algorithms with adaptive resource bounds for tackling the problem. Section 4 reports the comparative performances, convergence analysis, and worst-case analysis. Finally, we conclude in Section 5.

II. Problem Formulation

Given Q units of discrete resource and T activities, the problem is to find an optimal resource allocation to these activities such that the total costs entailed by performing the activities are minimized. The quantity of resource allocated to activity i is constrained in the range $[a_i, b_i]$. The cost function $f_i(x_i)$ is an integer nonlinear function which is dependent upon the quantity x_i of resource the activity i consumes. Formally, the RAP problem considered in this sequel is formulated by the following integer program.

$$\text{Min} \qquad J(\boldsymbol{X}) = \sum_{i=1}^{T} f_i(x_i), \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{T} x_i = Q, \tag{2}$$

$$0 \le a_i \le x_i \le b_i \le Q;\ \forall i = 1,2,...,T, \tag{3}$$

$$x_i \in integer\ .$$

The objective function (1) represents the total cost resulted from the resource allocation decision $\boldsymbol{X} = \{x_i\}_{1\le i\le T}$. Constraint (2) guarantees that the sum of allotted resource is equivalent to the total units of available resource. Constraint (3) states that the quantity x_i of resource allotted to activity i is constrained between the lower bound a_i and upper bound b_i.

III. Proposed Method

A. Adaptive Resource Bounds

As seen from Constraint (2) of the problem formulation, the quantities of resource allotted to different activities are correlated because all units of resource should be exactly used out during the execution of activities. If we allocate too many units of resource to a certain activity, the remaining resource may be insufficient for performing the other activities. On contrary, if we allocate too few quantities of resource to a certain activity, the amount of remaining resource may be larger than what can be maximally consumed by the other activities. Therefore, exploring all possible resource allocations between the pre-specified fixed resource bounds will yield many infeasible solutions violating Constraint (2) and impair the efficiency of the algorithm.

Two remedies to the above problem are presented as follows. The *feasibility checking rule* checks whether there exist any feasible solutions for the given problem instance. There exist no feasible solutions to the underlying problem if either $\sum_{i=1}^{T} a_i > Q$ or $\sum_{i=1}^{T} b_i < Q$ holds since Constraint (2) can never be satisfied in these conditions. If the given problem instance passes the feasibility checking rule, the *adaptive resource bound updating rule* guides the algorithm to construct a feasible solution. It sequentially allocates resource to the activity and adapts the resource bounds for the next activity. Let Q' be the remaining units of resource (initially $Q' = Q$) and assume that we have allocated resource to the first i activities. We adapts the resource upper bound for the (i+1)th activity as follows.

$$b'_{i+1} = \min\left\{\left(Q' - \sum_{l=i+2}^{T} a_l\right),\ b_{i+1}\right\},\quad i = 0, 1, \ldots, T\text{-}2. \tag{4}$$

The updating for the above resource upper bound is because the (i+1)th activity can only consume at most $Q' - \sum_{l=i+2}^{T} a_l$ for satisfying the minimum resource requests of the rest of the activities, and it cannot exceed the original resource upper bound b_{i+1} of the (i+1)th activity neither. Similarly, the resource lower bound for the (i+1)th activity is updated by

$$a'_{i+1} = \max\left\{\left(Q' - \sum_{l=i+2}^{T} b_l\right),\ a_{i+1}\right\},\quad i = 0, 1, \ldots, T\text{-}2. \tag{5}$$

After allocating resource to the first T - 1 activities with respect to the adaptive resource bounds, the remaining resource should be entirely given to the last activity in order to ensure that all units of the resource have been completely consumed.

B. PSO-based Method

The PSO [15] is inspired by the observations for bird flocking and fish schooling. A number of birds/fish flock synchronously, change direction suddenly, and scatter and regroup together. Each individual, called a particle, benefits from the personal best experience of its own (*pbest*) and that of any members of the swarm (*gbest*) observed so far during the search for food. In what follows, we apply the PSO with the adaptive resource bounds for solving the nonlinear RAP.

Each particle position P_i corresponds to a feasible resource allocation satisfying all constraints. Formally, $P_i = (p_{i1}, p_{i2}, ..., p_{iT})$ where $a'_j \le p_{ij} \le b'_j$, $\forall j = 1, 2, ..., T-1$ and $p_{iT} = Q - \sum_{j=1}^{T-1} p_{ij}$. The particle representation requests an allocation which allots p_{ij} units of resource to the jth activity and the last activity consumes the remaining resource. The PSO initializes a swarm of particles at random. In each iteration, particle i adjusts its velocity v_{ij} and position p_{ij} through each activity dimension j by referring to the personal best position ($pbest_{ij}$) and the swarm's best position ($gbest_j$) as follows.

$$v_{ij} = K[v_{ij} + c_1 r_1(pbest_{ij} - p_{ij}) + c_2 r_2(gbest_j - p_{ij})] \quad (6)$$

and

$$p_{ij} = p_{ij} + v_{ij} \quad (7)$$

where c_1 and c_2 are the self- and socio-cognition coefficients, r_1 and r_2 are random real numbers drawn from $U(0, 1)$, and K is the constriction factor. Clerc and Kennedy [16] has shown that the use of a constriction factor is needed to insure convergence of the PSO, and it is determined by

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad (8)$$

where $\varphi = c_1+c_2$, $\varphi>4$. Typically, φ is set to 4.1 and K is thus 0.729.

However, the fly of the particles is constrained by the adaptive resource bounds. We set particle position p_{ij} to a'_j if it passes over the adaptive lower bound, and to b'_j if it exceeds the adaptive upper bound.

The swarm intelligence of the PSO is managed by *pbest* and *gbest*. To identify them, the particle *fitness* should be evaluated. We define the fitness function of particle P_i by $fitenss(P_i) = 1/J(P_i)$, where $J(P_i)$ is computed by the objection function (1) using P_i as the decision for the resource allocation. Thus, the smaller the total cost incurred by P_i, the higher the fitness is, and vice versa. Here we propose a *bounding criterion* to expedite the computation for the fitness. We observe that the fitness value of a particle is only used for determination of $pbest_i$ and *gbest*, but not directly used for velocity update. Since $J(P_i)$ is a monotonically increasing function, we can use the fitness of the incumbent $pbest_i$ as a fitness bound (which should not be confused with the resource bound) and terminate the fitness evaluation of the *i*th particle when the intermediate fitness value has exceeded this bound. Also, only those $pbest_i$ that have been updated in the current iteration need to be compared with *gbest* for its possible updating. The use of bounding criterion can save the computational time significantly.

In all of the experiments, we terminate the program when our algorithm has experienced a specified number of computations for the fitness function.

C. ACO-based Method

The ant colony optimization (ACO) [14] is inspired by the research on the real ant foraging behavior. Ethologists observed that ants can construct the shortest path from their colony to the feeding source through the use of pheromone trails. An ant leaves some quantities of pheromone on the ground and marks the path by a trail of this substance. The next ant will sense the pheromone laid on different paths and choose one with a probability proportional to the amount of pheromone on it. The ant then traverses the chosen path and leaves its own pheromone. This is a positive feedback process which favors the path along which more ants previously traversed.

To apply ACO for solving the RAP, the problem must be represented by an appropriate graph along which the ant moves to construct candidate solutions. We represent an RAP instance by a (T+2)-layered graph. The first layer (start) and the last layer (sink) consist of only one node with which the ant starts and terminates its traversal. The intermediate T layers represent the allowable resource allocation quantities for the T tasks, respectively. Therefore, for the *i*th layer the allowable resource allocation quantity ranges from a'_i to b'_i. An ant constructs a candidate solution by traversing a path which starts from the start-node, visits each layer sequentially by selecting a node with an allowable quantity, and finally terminates at the sink-node.

The node transition rule determines the probability which is dependent on two factors: *pheromone* and *visibility*. Without loss of generality, assume that the ant is currently positioned at the *j*th node of the *i*th layer and is going to select a node, say, the *k*th node, from the (*i*+1)th layer. We use pheromone τ_{ijk} to exploit the historical traversal experiences for all the ants and determines the desirability for traversing the edge from a global optimization view. On the other hand, the visibility $\eta_{i+1,k}$ has nothing to do with the ant traversal experiences and only considers the problem-specific static information, which corresponds to the greediness about selecting a node from a local heuristic view. In particular, the visibility value for selecting the *k*th node from the (*i*+1)th layer is defined as

$$\eta_{i+1,k} = \frac{s}{f_{i+1}(k)}, \ i = 0, 1, \ldots, T\text{-}2; k = a_{i+1}, \ldots, b_{i+1}, \quad (9)$$

where s is a small constant. That is $\eta_{i+1,k}$ grows inversely proportional to $f_{i+1}(k)$. The smaller the cost $f_{i+1}(k)$ incurred by allocating k quantities of resource to the (*i*+1)th task, the higher the value of $\eta_{i+1,k}$.

We define the node transition probability p_{ijk} with which the ant at the *j*th node of the *i*th layer moves to the *k*th node of the (*i*+1)th layer as follows.

$$p_{ijk} = \frac{\tau_{ijk}^{\alpha}\eta_{i+1,k}^{\beta}}{\sum_{l=a'_{i+1}}^{b'_{i+1}} \tau_{ijl}^{\alpha}\eta_{i+1,l}^{\beta}} \quad (10)$$

where α and β are ACO parameters controlling the relative importance ratio between τ_{ijk} and $\eta_{i+1,k}$. Before activating the next iteration, the quantity of pheromone on each edge is updated by the following pheromone updating rule,

$$\tau_{ijk} \leftarrow (1-\rho)\tau_{ijk} + \sum_{t=1}^{m} \Delta\tau^{t} \tag{11}$$

where $\rho \in (0,1)$ is the evaporation rate of pheromone trails, m is the size of ant population, $\Delta\tau^{t}$ is calculated by

$$\Delta\tau^{t} = \frac{s}{J(X_t)} \tag{12}$$

$J(X_t)$ is computed by the objection function (1) using X_t as the decision for the resource allocation.

IV. EXPERIMENTAL RESULTS

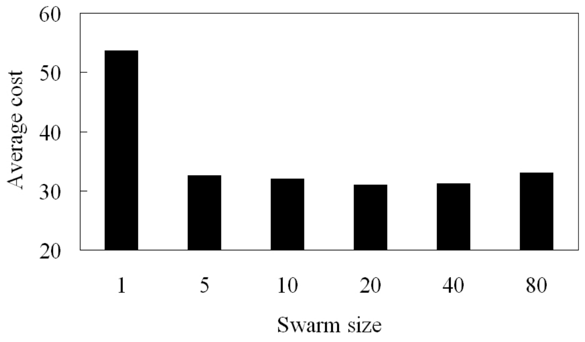
In this section, we analyze the properties of the proposed swarm algorithms and present the comparative performances with competing algorithms. The platform for conducting the experiments is a PC with a 2.4 GHz CPU and 256 MB RAM. All programs are coded in C++ language.

A. Synergism

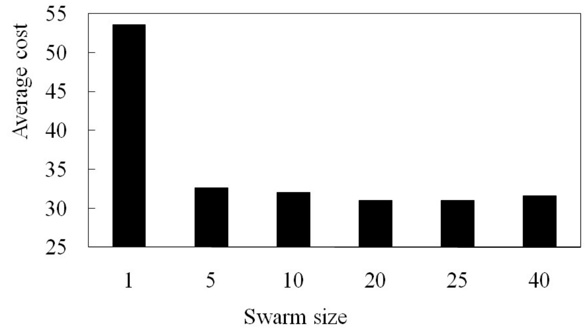
An interesting property about swarm algorithms is whether there exists an optimal size for the swarm. Given a fixed computation resource, the larger the swarm size, the smaller the number of evolutionary iterations can be performed for each particle/ant, and vice versa. We fix the number of times for evaluating the objective function when performing our algorithm with swarms of different size. As shown in Fig. 1, the average cost is minimal when the swarm size equals 20 for both the PSO- and ACO-based algorithms. In other words, there does exist an optimal swarm size for the given problem instance such that the synergism among different individuals is maximized.

B. Convergence

The convergence of the swarm algorithms can be analyzed by deriving the information entropy contained in the swarm. Here, we propose the information *entropy* for measuring the convergence of *pbest* in PSO and of node transition probability in ACO, because the evolution trajectories of PSO and ACO are substantially drawn by the value of *pbest* and the node transition probability. Fig. 2 shows a typical run for the variations of entropy value as the number of evolutionary iterations increases. It is observed for both swarm algorithms that the entropy value drops drastically at the initial period, then decreases gradually and finally stagnates. This demonstrates that the swarm intelligence is greatly enriched during the initial stage where the swarm scatters in the entire search space; but as the evolution becomes mature, the distributed awareness is resorting to the centralized awareness.

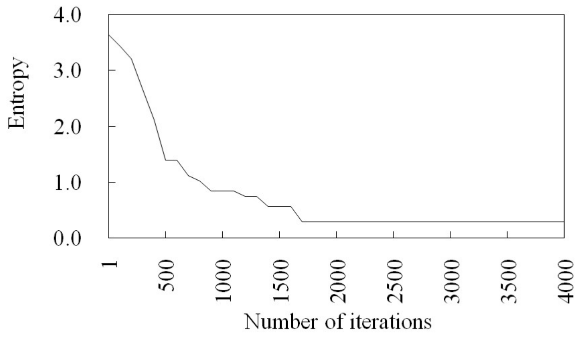


(a) PSO

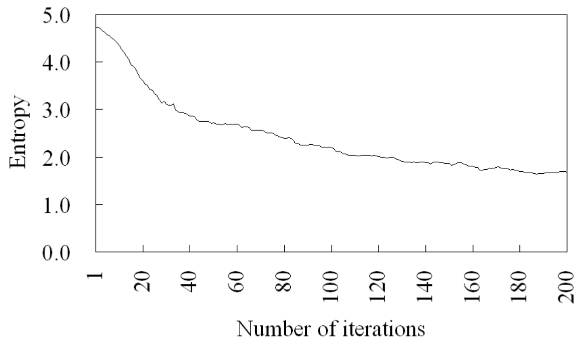


(b) ACO

Fig. 1 Average costs obtained by using swarms of different size.



(a) PSO



(b) ACO

Fig. 2 The entropy as the number of iterations increases.

C. Comparative Performances

The performance of the proposed swarm algorithms is evaluated by competing with an existing GA-based approach [10] and a naïve exhaustive search. We report the average cost and the needed mean CPU time over five independent

TABLE I
COMPARATIVE PERFORMANCES

Q	T	Δ	PSO		ACO		GA		Exhaustive	
			cost	time	cost	time	cost	time	cost	time
100	5	10	17	1301	17.8	48	17.8	3078	17	0.001
	5	20	12	1582	12	152	13.0	3982	12	0.001
	5	30	5	2012	5	326	10.5	4556	5	0.001
200	10	10	26	2954	26	234	38.0	7439	26	0.001
	10	20	12	3565	12	440	25.3	21317	12	8812
	10	30	13.5	3865	13.4	518	29.0	27132	13	142294
300	15	10	32.4	4085	32	416	47.0	194642	32	76159
	15	20	21.6	4927	22	604	60.0	285676	-	> 2 days
	15	30	17.2	6609	17	1007	-	> 2 days	-	-
400	20	10	40.6	5938	40	610	-	-	-	-
	20	20	28.1	7210	28.2	1047	-	-	-	-
	20	30	31.2	9927	31.2	2327	-	-	-	-

runs of each algorithm for every problem instance, while the exhaustive search is executed one time for deriving the exact solution. For practical concerns, all competing algorithms are terminated if the execution time exceeds 48 hours, and we discard such cases for further comparison.

We generate a set of testing problems with diverse computational complexity by varying the number of activities (T) and the mean allowable range of allocated resource units ($\Delta = \sum_{i=1}^{T}(b_i - a_i)/T$) for all activities. The value of T ranges from 5 to 20, and for each value of T, we set Δ to 10, 20, and 30, respectively. The value of Q is given appropriately for various values of T and Δ in order to assure existence of feasible solutions. It is seen from Table 1 that, for the small- and median-sized problems, both PSO and ACO can derive solutions that are equal or very close to the exact solutions as reported by exhaustive search, while the solutions obtained by the GA-based approach are far from the optima. As for the large-sized problems, the solutions obtained by using the PSO and ACO are also significantly better than those obtained by the GA-based approach. This is mainly due to the adaptive resource bounds technique for avoiding producing infeasible solutions and thus significantly reducing the searching space. On the other hand, the GA-based method does not actually reduce the searching space, when infeasible solutions are produced it modifies them to feasible solutions by adjusting resource allocation between activities to ensure that the resource allocation does not violate resource bound constraints.

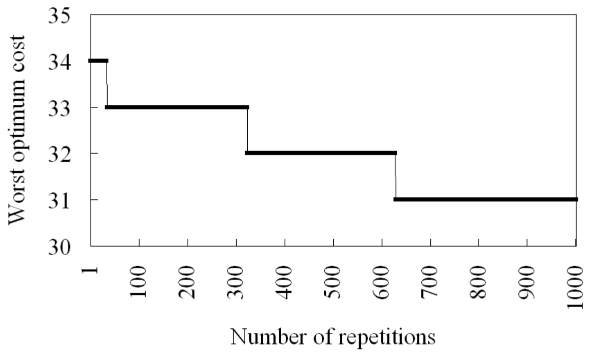
As for the computational efficiency, Table 1 also illustrates that the CPU times consumed by the exhaustive search method grow exponentially with the problem size, and it fails to solve large-scaled problems. The computational times consumed by GA also grow at a fast rate and exceed 48 hours for the last four instances. On the other hand, the computational time used by the PSO and ACO methods as the problem size increases is still in an acceptable range, thought the ACO-based method seems to be computationally faster than the PSO-based method. In summary, both the proposed swarm algorithms can derive quality solutions against scalable problems.

D. Worst-case Analysis

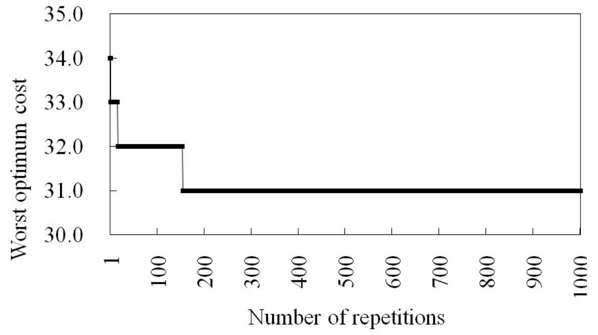
Since swarm algorithms are stochastic methods, each separate run of the same program could yield different result. It is critical to analyze the worst-case performance one may obtain if the swarm algorithm is adopted for tackling the RAP. The worst-case performance is analyzed as follows. *First*, the program is executed for a specific number of repetitive runs and each run will output an optimal solution. *Second*, the worst-case analysis is set up as the worst solution we could get from those optimal solutions. Fig. 3 shows the worst-case analysis where the swarm algorithms are executed to solve the problem instance with (Q, T, Δ) = (400, 20, 30) for 1,000 times. The curve shows that about 97% of the repetitive runs can obtain a quality solution with cost less than or equal to 33, meaning that we can be confident in using the swarm algorithms for solving the RAP. When the user requests a worst-case guarantee for the derived solution with cost no more than 33, he/she can obtain such a solution by executing the swarm algorithms for no more than 30 repetitive runs (3% of 1,000 runs).

V. CONCLUSIONS

The constrained resource allocation problem (RAP) seeks for an allocation of a fixed amount of resource to a number of activities such that the objective is optimized and the resource constraints are met. RAP has many applications, including product allocation, resource distribution, project budgeting, software testing, health care resource allocation, just to name a few. Different versions of problem formulations have been proposed in accordance with various applications. This paper addressed the nonlinear RAP with integer decision variable constraint. The proposed swarm algorithms are built on the foundation of PSO and ACO with the adaptive resource bounds technique in order to construct feasible solutions. Simulation results show that the swarm algorithms derive comparable solutions to the exact solutions obtained using an exhaustive search method. The swarm algorithms also outperform a GA-based approach which yields worse solutions and needs more computational time. The general



(a) PSO



(b) ACO

Fig. 3 The worst optimum cost vs. the number of repetitive runs of the PSO algorithm.

properties and the convergence behaviors of the swarm algorithms are analyzed. Finally, a performance guarantee is provided through the worst-case analysis.

ACKNOWLEDGMENT

This work is supported in part by the National Science Council of Taiwan under grant NSC 98-2410-H-260-018-MY3.

REFERENCES

[1] A. Basso and L. A. Peccati, Optimal resource allocation with minimum activation levels and fixed costs, *European Journal of Operational Research*, vol. 131, 2001, pp. 536-549.

[2] Y. S. Dai, M. Xie, K. L. Poh, and B. Yang, Optimal testing –resource allocation with genetic algorithm for modular software systems, *The Journal of Systems and Software*, vol. 66, 2003, pp. 47-55.

[3] A. Ernst, H. Hiang and M. Krishnamoorthy, Mathematical programming approaches for solving task allocation problems, *Proc. of the 16th National Conf. Of Australian Society of Operations Research*, 2001.

[4] A. A. Stinnett and A. D. Paltiel, Mathematical programming for the efficient allocation of health care resource, *Journal of Health Economics*, vol. 15, 1996, pp. 641-653.

[5] T. Ibaraki and N. Katoh, "Resource allocation problems: algorithmic approaches," MIT Press, Boston, 1988.

[6] S. Birch and A. Gafni, Cost effectiveness analysis: Do current decision rules lead us where we want to be?, *Journal of Health Economics*, vol. 11, 1992, pp. 279-296.

[7] D. Morales, F. Almeida, F. Garcia, J. L. Roda, and C. Rodriguez, Design of parallel algorithms for the single resource allocation problem, *European Journal of Operational Research*, vol. 126, 2000, pp. 166-174.

[8] K. M. Bretthauer and B. Shetty, The nonlinear resource allocation problem, *Operations Research*, vol. 43, 1995, pp. 670-683.

[9] K. M. Bretthauer and B. Shetty, A pegging algorithm for the nonlinear resource allocation problem, *Computers & Operations Research*, vol. 29, 2002, pp. 505-527.

[10] Y. C. Hou and Y. H. Chang , A new efficient encoding mode of genetic algorithms for the generalized plant allocation problem, *Journal of Information Science and Engineering*, vol. 20, 2004, pp. 1019-1034.

[11] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," Addison Wesley, Reading, Massachusetts, 1997.

[12] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi, Optimization by simulated annealing, *Science*, vol. 220, 1983, pp. 671-680.

[13] F. Glover, Tabu search - Part I, ORSA *Journal of Computing*, vol. 1, 1989, pp. 190-206.

[14] M. Dorigo, "Optimization, learning, and natural algorithms," Ph.D. Thesis, Dip. Elettronica e Informazione, Politecnico di Milano, Italy, 1992.

[15] J. Kennedy and R.C. Eberhart, Particle swarm optimization, *Proceedings IEEE Int'l. Conf. on Neural Networks*, IV, 1995, pp. 1942-1948.

[16] M. Clerc and J. Kennedy, The particle swarm explosion, stability, and convergence in a multidimensional complex space, *IEEE Transaction on Evolutionary Computation*, vol. 6, 2002, pp. 58-73.