Applied Soft Computing 26 (2015) 378–389

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem

Mingchang Chih*

System Development Center, National Chung-Shan Institute of Science and Technology, Lungtan Township, Taoyuan 32546, Taiwan, ROC

ARTICLE INFO

Article history:
Received 16 May 2014
Received in revised form 20 October 2014
Accepted 22 October 2014
Available online 31 October 2014

Keywords:
Combinatorial optimization
Self-adaptive check and repair
Multidimensional knapsack problem
Particle swarm optimization
Pseudo-utility ratio
OR-Library

ABSTRACT

The multidimensional knapsack problem (MKP) is a combinatorial optimization problem belonging to the class of NP-hard problems. This study proposes a novel self-adaptive check and repair operator (SACRO) combined with particle swarm optimization (PSO) to solve the MKP. The traditional check and repair operator (CRO) uses a unique pseudo-utility ratio, whereas SACRO dynamically and automatically changes the alternative pseudo-utility ratio as the PSO algorithm runs. Two existing PSO algorithms are used as the foundation to support the novel SACRO methods, the proposed SACRO-based algorithms were tested using 137 benchmark problems from the OR-Library to validate and demonstrate the efficiency of SACRO idea. The results were compared with those of other population-based algorithms. Simulation and evaluation results show that SACRO is more competitive and robust than the traditional CRO. The proposed SACRO-based algorithms rival other state-of-the-art PSO and other algorithms. Therefore, changing different types of pseudo-utility ratios produces solutions with better results in solving MKP. Moreover, SACRO can be combined with other population-based optimization algorithms to solve constrained optimization problems.

1. Introduction

The 0–1 multidimensional knapsack problem (MKP) is a well-known NP-hard optimization problem. MKP is one of the most intensively studied discrete programming problems [1]. Many practical problems are commonly modeled as MKPs [2,3]. MKP can be mathematically formulated as follows:

$$\max \quad z = \sum_{j=1}^{n} c_j y_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} y_j \le b_i, \quad i = 1, 2, \ldots, m, \tag{2}$$

$$y_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n, \tag{3}$$

where n is the number of items and m is the number of knapsack constraints, with the capacity b_i for $i = 1, 2, \ldots, m$. Each item j requires a_{ij} units of resource consumption in the ith knapsack and yields c_j units of profit upon inclusion. The goal is to find an item subset that yields the maximum profit without exceeding the resource capacity. All entries are naturally nonnegative.

MKP is a combinatorial optimization problem [4]. From the perspective of computation, the different proposed algorithms for approaching the MKP can be broadly grouped into two classes: exact algorithms and heuristic/metaheuristic algorithms. Exact techniques included the Lagrangian methods, surrogate relaxation techniques, special enumeration techniques and reduction schemes, and the branch-and-bound methods. Exact algorithms are not practical for MKP because the search space grows exponentially with the problem size and exhaustive search is infeasible. Therefore, several modern heuristic algorithms, such as simulated annealing [5], Tabu search [6], genetic algorithms [7], ant colony algorithms [8], particle swarm optimization (PSO) [9,10], and other heuristics [11], were developed to solve MKP.

Many works have not successfully proved that crude heuristic algorithms were an effective tool for the MKP. Most of them cannot solve large-scale problems effectively and efficiently. Very recently, some works [12,13] proposed and designed heuristics with new repair operator particularly for large-scale problems by restricting the algorithms to search only the feasible space. Due to the significance of the MKP in academic research and real applications, it is important to develop novel algorithms with satisfactory performances.

* Tel.: +886 37331862
E-mail address: d927805@oz.nthu.edu.tw

http://dx.doi.org/10.1016/j.asoc.2014.10.030
1568-4946/© 2014 Elsevier B.V. All rights reserved.

PSO is a promising alternative to other population-based optimization algorithms [14–18] for solving combinatorial optimization problems. This conventional heuristic approach was designed by Kennedy and Eberhart [19] to optimize various continuous nonlinear functions. PSO is a random search algorithm that simulates natural evolutionary processes to solve complex optimization problems. The PSO prototype is restricted to the real number space. However, many optimization problems are set in a space with discrete or qualitative distinctions between variables. Kennedy and Eberhart [20] developed a discrete version of the algorithm to address this problem.

PSO has been successfully applied to various areas, such as the traveling salesman problem [21], inspection policy [22], classification problem [23], economic statistical control chart design [24], reliability networks [25], multi-modal problems [26], feature selection [27], multi-objective optimization [28], supplier selection [29], production design and manufacturing [30], structure design [31], vehicle crashworthiness [32], and MKP [33].

This study proposes a self-adaptive repair mechanism to convert infeasible solutions to feasible ones for PSO. The proposed mechanism and algorithms are compared with state-of-the-art PSO algorithms to solve the MKP using the available test datasets in the OR-Library [34].

The rest of the paper is organized as follows: Section 2 provides the basic concepts of PSO algorithm. The proposed self-adaptive repair mechanism and PSO algorithms are presented in Sections 3 and 4, respectively. Section 5 summarizes the simulation and evaluation results of the proposed algorithms on the test datasets. The conclusion is presented in Section 6.

2. Background

PSO was first developed by Kennedy and Eberhart [19] based on the metaphors of social interaction and communication (e.g., fish schooling and bird flocking). PSO uses the collaboration between simple search agents called particles in a population to find the optima in a particular search space, and it is effective in optimizing difficult multidimensional problems in different fields [35,36].

PSO combines local and global searches to obtain high search efficiency. Its algorithm is initialized in a population of random particles with random positions and velocities inside the problem space. PSO subsequently searches for optima by updating successive generations. Each particle is updated during each iteration according to the two "best" values. The first value, called *pbest*, is the best solution (fitness) achieved by the particle. The other "best" value tracked by PSO is the current best value obtained by any particle in the population. This best value is the global best and is designated as *gbest*. When a particle considers parts of the population as its topological neighbors, the best value is a local best, i.e., *lbest*. After the two best values are identified, the particle updates its velocity and position according to the following equations in continuous PSO:

$$v_{ij}^t = w \cdot v_{ij}^{t-1} + c_1 \cdot Rand() \cdot (p_{ij}^{t-1} - x_{ij}^{t-1}) + c_2 \cdot Rand() \cdot (g_{ij}^{t-1} - x_{ij}^{t-1}), \quad (4)$$

and

$$x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t, \quad (5)$$

where v_{ij}^t is the velocity of ith particle at jth dimension (decision variable) in tth iteration; x_{ij}^{t-1} is the current particle (solution); p_{ij}^{t-1} and g_{ij}^{t-1} are *pbest* and *gbest*, respectively; and *Rand*() is a random number between [0,1]. c_1 and c_2 are learning factors, and the value of $(c_1 + c_2)$ is usually limited to 4 [35,36]. PSO is affected by several parameters, such as the number of population members (m), cognition learning factor (c_1), social learning factor (c_2), inertia weight (w), and number of iterations or CPU time.

The optimum solution in population-based optimization methods is determined by gaining proper control of global and local explorations. Shi and Eberhart [37] found a significant improvement in the performance of PSO with a linearly varying inertia weight over generations. The mathematical representation of this method is given by

$$w = (w_{\max} - w_{\min})\frac{t_{\max} - t}{t_{\max}} + w_{\min}. \quad (6)$$

Ratnweera et al. [38] introduced time-varying acceleration coefficients (TVACs) in PSO with learning factors. TVACs enhance the global search in the early stages of optimization and encourage the particles to converge toward the global optima at the end of the search. TVACs can be mathematically represented as

$$c_1 = (c_{1f} - c_{1i})\frac{t-1}{t_{\max}} + c_{1i}, \quad (7)$$

$$c_2 = (c_{2f} - c_{2i})\frac{t-1}{t_{\max}} + c_{2i}. \quad (8)$$

Shi and Eberhart [37] observed that the optimal solution for most problems could be improved by varying the value of w from 0.9 to 0.4 throughout the search. Ratnweera et al. [38] found that the optimum solution for most benchmarks could be improved by changing c_1 from 2.5 to 0.5 and c_2 from 0.5 to 2.5 throughout the search.

Chih et al. [33] combined the time-varying inertia weight (w) and time-varying learning factors (c_1, c_2) of TVACs with their proposed PSO algorithms. The proposed velocity updating equation can be expressed as:

$$v_{ij}^t = w \cdot v_{ij}^{t-1} + \left\{(c_{1f} - c_{1i})\frac{t-1}{t_{\max}} + c_{1i}\right\} \cdot Rand() \cdot (p_{ij}^{t-1} - x_{ij}^{t-1}) + \left\{(c_{2f} - c_{2i})\frac{t-1}{t_{\max}} + c_{2i}\right\} \cdot Rand() \cdot (g_{ij}^{t-1} - x_{ij}^{t-1}) \quad (9)$$

PSO was initially introduced to optimize various continuous nonlinear functions; therefore, the major obstacle of using PSO in practical applications is its continuous nature. Kennedy and Eberhart [20] developed a discrete binary version of PSO called BPSO to resolve this drawback. The particle in BPSO is characterized by a binary solution presentation, and the velocity must be transformed toward the change in probability for each binary dimension to take a value of one.

The position update equation has been recently introduced into BPSO [33,39]. It is based on the position update equation for direct continuous optimization. If the velocity bounds are $-V_{\max}$ and $V_{\max}$, then the term $x_{ij}^{t-1} + v_{ij}^t$ is bound between $(0 - V_{\max} = -V_{\max})$ and $(1 + V_{\max})$ because x_{ij}^t in Eq. (5) must have a value of 0 or 1. Therefore, the position update equation can be represented as:

$$x_{ij}^t = \begin{cases} 1, & \text{if } U(-V_{\max}, 1 + V_{\max}) < x_{ij}^{t-1} + v_{ij}^t; \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

$$U(0, 1) = \frac{U(-V_{\max}, 1 + V_{\max}) + V_{\max}}{(1 + 2V_{\max})}.$$

Eq. (10) can be rewritten as

$$x_{ij}^t = \begin{cases} 1, & \text{if } U(0, 1) < \dfrac{x_{ij}^{t-1} + v_{ij}^t + V_{\max}}{1 + 2V_{\max}}, \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

where $U(0, 1)$ is a random number between [0,1], and $U(-V_{\max}, 1 + V_{\max})$ is a random number between $[-V_{\max}, 1 + V_{\max}]$.