Particle Swarm Optimization for the Multidimensional Knapsack Problem

Fernanda Hembecker, Heitor S. Lopes*, and Walter Godoy Jr.

Federal University of Technology Paraná (UTFPR),
Av. 7 de setembro, 3165 80230-901, Curitiba (PR), Brazil
fernanda@denes.com.br, hslopes@pesquisador.cnpq.br, godoy@utfpr.edu.br

Abstract. The multidimensional 0/1 knapsack problem is a classical problem of discrete optimization. There are several approaches for solving the different variations of such problem, including mathematical programming and stochastic heuristic methods. This paper presents the application of Particle Swarm Optimization (PSO) for the problem, using selected instances of ORLib. For the instances tested, results were very close or equal to the optimal solution known, even considering that the parameters of PSO were not optimized. The analysis of the results suggests the potential of a simple PSO for this class of combinatorial problems.

1 Introduction

There are problems with important practical applications concerned with the search of the "best" configuration or set of parameters to achieve some objective criteria. Such problems are generally referred to as optimization problems. The knapsack problem is a classical optimization problem and has many practical applications in industry, transportation and logistics. A simple and general description of the problem is as follows. Given a set of objects with corresponding values and costs (weights), select some of them to put in a container (knapsack), without extrapolating its capacity, in such a way that maximize the sum of values. This is the basic definition of the problem. However, there are several variations, such that [10]:

- Single knapsack problem: all objects must be put in a single container;
- Multidimensional knapsack problem: more than one container is available;
- Multiple-choice knapsack problem: objects are clustered into subsets and at most one object can be selected;
- Bounded knapsack problem: there is a limited number of objects available to be selected.

This paper is related to the Multidimensional Knapsack Problem (MKP), possibly, the most widely known version of the problem [2]. This same problem

* This work was partially supported by the Brazilian National Research Council – CNPq, under research grant no. 305720/2004-0 to H.S. Lopes.

B. Beliczynski et al. (Eds.): ICANNGA 2007, Part I, LNCS 4431, pp. 358–365, 2007.
© Springer-Verlag Berlin Heidelberg 2007

is also referred as: Multiconstraint, Multi-Knapsack or 0/1 Multidimensional Knapsack Problem.

Many problem-independent and domain-specific heuristics have been developed for optimization problems and, in particular, to the MKP that is NP-complete. The quest for a computational algorithm that are effective both in terms of processing time and quality of the solutions is the motivation of this work. Therefore, we propose a methodology for solving MKP using a relatively recent evolutionary computation technique, the Particle Swarm Optimization (PSO), proposed by Kennedy and Eberhart [7]. This problem has been explored by using genetic algorithms [2],[6],[8], and there are evidences that PSO can be useful for it [4]. PSO, in fact, has been applied successfully to several problems like pattern recognition, classification, scheduling, mobile robotics, image processing, and others [5].

2 Particle Swarm Optimization

PSO is a heuristic method for optimization, inspired in the behavior of social agents found in nature. This behavior can be observed in bird flocking, bee swarming, and fish schooling, for instance.

The computational model is population-based. Agents, or particles, change their position (state) in the multidimensional search space of the problem, according to their own experience and the influence of the neighboring particles. Each particle has a limited store capability, keeping track only of information about its current position, speed and quality (fitness of the solution regarding the problem), as well as its best position ever visited ("best particle solution" – *pbest*). Amongst the swarm of particles, the one with best quality is referred as "the best global solution" – *gbest*. Alternatively, only the neighborhood of the particle is considered, that is, the ("best local solution" – *lbest*. At each time tick, particles move, influenced by both *pbest* and *gbest*, to a new position in the search space. This is an iterative process, repeated until a stop condition is met, usually a predefined number of iterations. Whenever a better solution than the previous is found, *gbest* is updated. This procedure is similar to the principle of elitism, common in other evolutionary computation paradigms, since throughout iterations the best solution is conserved. However, there is a subtle difference: *gbest* is updated is a reference for all particles in the same iteration (in a genetic algorithm, this would be similar to say that all individuals would mate with the best individuals).

It is interesting that *pbest* would be a point with good fitness and also located quite far from *gbest* in the search space, so as to improve diversity. In PSO, like other population-based heuristics, maintaining diversity throughout iterations is often a challenge, and it is a necessary condition to assure a satisfactory exploration of the search space. When many *pbest* 's are somewhat close to the *gbest*, there will be a particle crowding and the search stagnates. A mechanism to avoid the consequences of this unavoidable convergence is the explosion of the swarm, sending particles to random positions and keeping *gbest*.

In the classical PSO model, the movement of the i-th particle is defined by (1), where its next position in the search space (X_i^{t+1}) is updated using the current position and a speed term (V_i^t):

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

In fact, the speed term actually does not have the dimension of velocity. It could be better defined as ΔX_i but, for the sake of simplicity, it is called speed, following [5]. The speed term, in turn, is defined according to (2):

$$V_i^t = c_1.r_1.\Delta_{pbest} + c_2.r_2.\Delta_{gbest}\,, \tag{2}$$

where: V_i^t is the current speed of the i-th particle; r_1 and r_2 are random values in the range [0..1]; c_1 and c_2 are the weights of *pbest* and *gbest*, respectively (in percentage); Δ_{pbest} and d_{gbest} are the distance between the current position and *pbest* and the current position and *pgbest*, respectively. The speed term, that is, the updating rate of the current position, is directly proportional to the distance between the current position to *pbest* and *gbest*. Therefore, within few iterations the particle will be attracted to either *pgbest* or *gbest*.

The speed term controls the amount of global and local exploration of the particle (that is, the balance between exploration and exploitation). A high speed facilitates global exploration, while small speed will encourage local search. A user-defined upper bound (V_{max}) is established to limit the maximum speed of particles. Figure 1 shows graphically the elements that influence the position of a particle in the hypersurface of the search space (in this case, a bi-dimensional space).

According to a psychological interpretation of PSO [5], the swarm of particles is like a population of individuals. Then, the two terms of (2) represent the cognitive and the social components of a particle's behavior. The former leads the particle to repeat its own past successful behaviors, while the latter makes it follows the others'. There are no default values for weights c_1 and c_2; sometimes

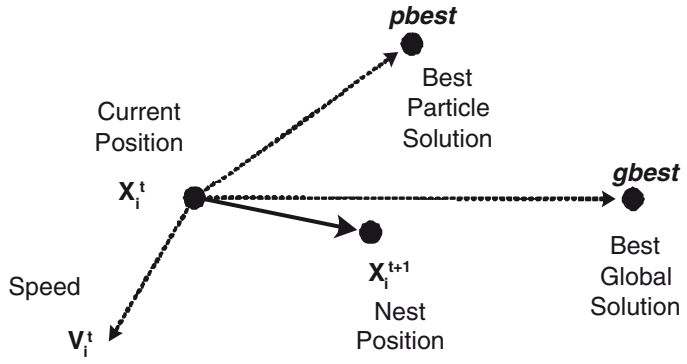

Fig. 1. Illustration of how the position of a particle is updated in PSO

they are set identical and sometimes they are set asymmetrical. It is commonly accepted that those weights are problem-dependent and this seems to be an open subject for further research [3].

3 Methodology

A description of the MKP includes n objects and m knapsacks with specific capacities c_j $(j = 1, \ldots, m)$. There are binary variables x_i $(i = 1, \ldots, n)$ that are set to 1 if the i-th object is selected to be put in the knapsacks, and 0, otherwise. Every object has a satisfaction value p_i $(i = 1, \ldots, n)$ and a specific weight w_{ij} for each knapsack. Therefore, the optimization problem is defined as follows [8],[10]:

$$maximize \quad \sum_{i=1}^{n} p_i.x_i, \tag{3}$$

$$subject \quad to \quad \sum_{j=1}^{m} w_{ij}.x_i \leq c_j. \tag{4}$$

The objective of the MKP is to fill the knapsacks with the most valuable objects without extrapolating their capacities. Therefore, a particle should be represented as a possible solution for the MKP. In words, a particle is a binary vector, where each element indicates whether or not an object is selected to be included in the knapsacks. The length of the vector depends on the number of objects available for the selection, that is, it represents the n-dimensional search space. It should be noted that, in this formulation, a given selected object is to be included in all knapsacks.

The evaluation of a possible solution is given by a fitness function. This function is what PSO will optimize. The fitness function was defined before (3), but the constraint (4) is not directly dealt by the algorithm, and so, unfeasible solutions can be found during search. This policy is frequently used in evolutionary algorithms because during evolution towards the best global solution, the algorithm can pass through regions of unfeasible solutions. In PSO, a given particle is dynamically attracted by the social and the cognitive components, and an unfeasible particle now can be changed to a feasible one later. A particle representing an unfeasible solution is allowed to exist in the swarm, but a penalty will be imposed to the fitness of such particle. This penalty is proportional to the total amount of excess in the knapsacks.

4 Computational Experiments

For the computational experiments, we used several instances of MKP found in [1]. These instances were divided into two groups: the first one corresponded to series "sento" [12] and "weing" [15], and the second group, to "weish" [13]. For all experiments reported in this section, the PSO was run for 300 iterations.

The first group of experiments had 10 problems with either 2 or 30 knapsacks and 28, 60 or 105 objects. For these experiments, results are shown in Table 1. In this table, n is the number of objects and m is the number of knapsacks.

Table 1. Comparison of results obtained by PSO and the optimal known solutions

problem	m	n	optimum	best	abs.diff.
sento1	30	60	7772	7725	0.605%
sento2	30	60	8722	8716	0.069%
weing1	2	28	141278	138927	1.664%
weing2	2	28	130883	125453	4.149%
weing3	2	28	95677	92297	3.533%
weing4	2	28	119337	116622	2.275%
weing5	2	28	98796	93678	5.180%
weing6	2	28	130623	128093	1.937%
weing7	2	105	1095445	1059560	3.276%
weing8	2	105	492347	492347	0.000%

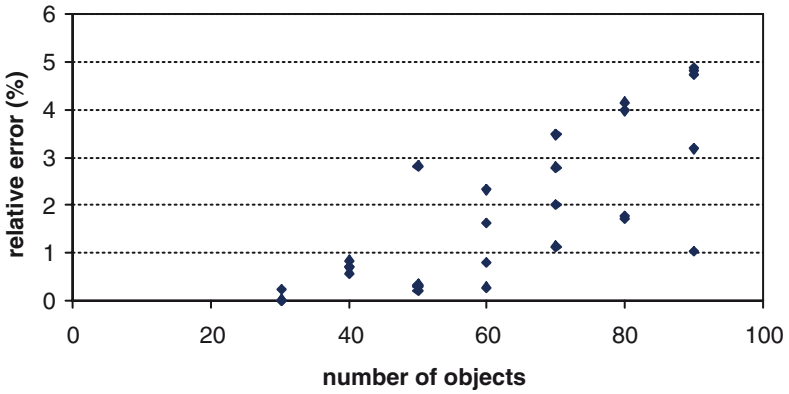

Fig. 2. Performance of PSO regarding the complexity of the problem

Each experiment was repeated 100 times with different random seeds and the best solution found by PSO is shown in the table. Therefore, a total of 1,000 experiments were run in this group.

The second group of experiments was to investigate the behavior of PSO regarding the complexity of the problem. In this particular case, complexity is understood as the number of possible combinations of objects $\times$ knapsacks. The second group of experiments had 30 problems, always with 5 knapsacks, but objects ranged from 30 to 90. Figure 2 shows the performance of PSO regarding the relative difference between the best value found in 100 independent runs and the optimum known value. As mentioned before, for this group of experiments the number of knapsacks is constant (5) and the number of objects increases.

To investigate the convergence of the algorithm, we analyzed the experiments of four different instances with the same number of knapsacks: weish2, weish12, weish23, weish28, having 30, 60, 80, 90 objects, respectively. Figure 3 shows

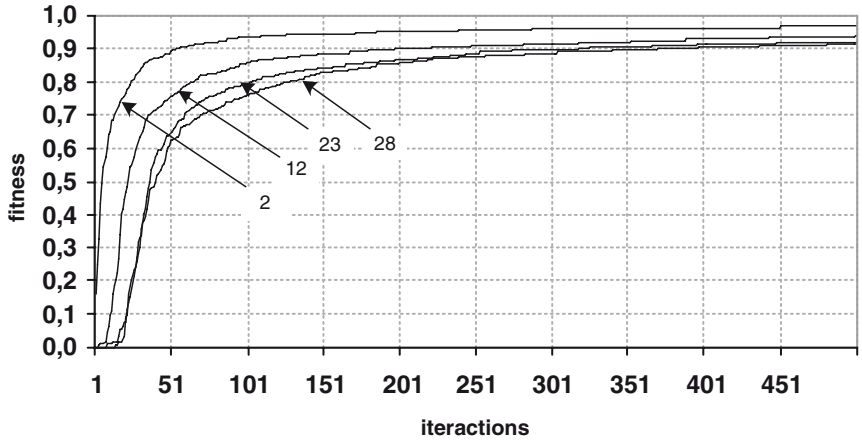

Fig. 3. Evolution of solutions for some instances of the MKP

the evolution of the best solution found by PSO at each iteration for the four instances (numbered as 2, 12, 23 and 28) in 500 iteractions. Each point of the curves is the average of the best solution found in 100 independent runs with random seeds.

It should be noted that [2] present a genetic algorithm for the MKP using the instances used in our work, and they always found the optimum. However, to achieve such performance, they needed to process 10^4 chromosomes, thus requesting a large computational effort, not comparable with the PSO implementation.

5 Conclusions and Future Work

In table 1, the average performance of PSO for the ten case studies considered was 2.269% of the known optimum. This shows that PSO can perform well for this class of combinatorial problem, even for large instances. That is, PSO seems to be efficient in navigating the hypersurface of the search space and finding good solutions (and, sometimes, the best solution) independently of the initialization and the trajectory of particles.

The second group of experiments aimed at identifying how the performance of PSO degraded as the difficulty of the problem increased. For the particular range of experiments done, we observed that PSO tends to decrease the average performance almost linearly, as shown in figure 2. However, more experiments have to be done so as to confirm this tendency for even harder instances. Also, it can be seen in the graphics that, for the same degree of difficulty, different performances are achieved, depending on the specific nature of each instance.

Figure 3 indicates that, the harder the instance, the longer PSO takes to converge. In the figure, it can be observed a positive derivative in the curves,

suggesting that more iterations, besides 500, are needed to converge. This is specially true for the harder instances.

It is a matter of fact that PSO is sensitive to its control parameters, particularly for hard combinatorial problems with large search space [14]. Recall that no serious attempt was done to optimize the running parameters for the PSO, what could improve the performance achieving better results. Therefore, future work will focus on fine-tuning the PSO parameters for given classes of problems, by using some kind of adaptive strategy [11]. Recent literature have shown that a PSO hybridized with a local search technique certainly can achieve better results than a "pure" PSO, independently of the problem [9]. Therefore, further improvement of the system will be towards the hybridization with some local search technique.

In general, the use of a problem-independent heuristics, such as PSO, gives robustness and efficiency to the exploration of the search space of difficult problems. We believe that this is a promising method for solving several classes of combinatorial problems.

References

1. Beasley, J.E.: ORLib – Operations Research Library. [*http : //people.brunel.ac.uk/˜mastjjb/jeb/orlib/files/mknap2.txt*] , (2005).
2. Beasley, J.E., Chu, P.C.: Genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics* **4** (1998) 63–86.
3. Clerc, M.: The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: *Proc. IEEE Congress on Evolutionary Computation*, vol. 3, pp. 1951–1957, (1999).
4. Eberhart, R.C., Shi, Y.: Comparison between genetic algorithms and particle swarm optimization. In: *Proc. 7^{th} Annual Conference on Evolutionary Programming*, pp. 611–616, (1998).
5. Eberhart, R.C., Shi, Y.: Particle swarm optimization: developments, applications and resources. In: *Proc. Congress on Evolutionary Computation*, Seoul, Korea, vol. 1, pp. 81–86 (2001).
6. Hoff, A., Løkketangen, A., Mittet, I.: Genetic algorithm for 0/1 multidimensional knapsack problems. In: *Proc. Norsk Informatikkonferanse*, Molde, Norway, (1996).
7. Kennedy, J., Eberhart, R.C.: *Swarm Intelligence*. Morgan Kaufmann, San Francisco, USA (2001).
8. Khuri, S., Bäck, T., Heitkoetter, J.: The zero/one multiple knapsack problem and genetic algorithm. In: *Proc. ACM Symposium on Applied Computing*, Phoenix, USA, pp. 188–193 (1994).
9. Lopes, H.S., Coelho, L.S.: Particle swarm optimization with fast local search for the blind travelling salesman problem. In: *Proc. 5^{th} Hybrid Intelligent Systems Conference*, Rio de Janeiro, Brazil, pp. 245–250 (2005).
10. Martelo, S., Toth P.: *Knapsack Problems – Algorithms and Computer Implementations*. John Wiley & Sons, New York, USA (1990).
11. Maruo, M.H., Lopes, H.S., Delgado, M.R.B.S.: Self-adapting evolutionary parameters: encoding aspects for combinatorial optimization problems. In: Raidl, G.R., Gottlieb, J. (eds.), *Evolutionary Computation for Combinatorial Problems*, LNCS, v. **3448** (2005) pp. 154–165.

12. Senyu. S., Toyoda, Y.: An approach to linear programming with 0-1 variables. *Management Science* **15** (1967) B196–B207.
13. Shih, W.: A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society* **30** (1979) 369–378.
14. Trelea, I.C.: The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters* **85** (2003) 317–325.
15. Weingartner, H.M., Ness, D.N.: Methods for the solution of multi-dimensional 0/1 knapsack problems. *Operations Research* **15** (1967) 83–103.