G.D. Smith, N.C. Steele, R.F. Albrecht (eds.)
Artificial Neural Nets and Genetic Algorithms 3
Springer-Verlag Wien New York 1998
pp. 250-254

A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem

Carlos Cotta, José Mª Troya

Departamento de Lenguajes y Ciencias de la Computación
Complejo Politécnico (2.2.A.6), Campus de Teatinos
29071 - Málaga (SPAIN)
{ccottap, troya}@lcc.uma.es

Abstract. A hybrid genetic algorithm based in local search is described. Local optimisation is not explicitly performed but it is embedded in the exploration of a search metaspace. This algorithm is applied to a NP-hard problem. When it is compared with other GA-based approaches and an exact technique (a branch & bound algorithm), this algorithm exhibits a better overall-performance in both cases. Then, a coarse-grain parallel version is tested, yielding notably improved results.

1 Introduction

Genetic algorithms have been traditionally considered as robust techniques, easily applicable to almost any domain. This robustness is not only an advantage, but also a major point of weakness. As shown in recent research, (the so-called *No Free Lunch Theorem* [1]) no algorithm can be expected to outperform any other one (including random search) when averaged over all possible problems. In other words, an algorithm is as good for a problem (or problem domain) as the problem-dependent knowledge it incorporates.

For that reason, hybrid genetic algorithms (i.e., genetic algorithms using specialised non standard mechanisms) have been often defined to deal with problems that would be very hard for a blind black-box search algorithm. These hybrid algorithms have been proved to be very effective tools in many situations (see [2]). Problem-dependent knowledge is incorporated by means of appropriate representations and specialised operators. In that sense, the use of operators performing local search is a widely used technique. These operators frequently play an essential rôle (e.g., in Mühlenbein's parallel genetic algorithm [3]). In this work, we study a genetic algorithm based in local search. Unlike other hybrid approaches, local optimisation is not achieved via an improvement heuristic (like hill climbing) but by means of a construction heuristic. This implies that the search does not take place in the solution space but in a solution metaspace (the problem space).

2 Problem-Space Search

Problem-space exploration was first suggested by Storer *et al.* [4]. In short, it consists in the intelligent generation of different starting points for a construction heuristic. Since a construction heuristic H is a function that returns a feasible solution for a given problem instance, every new starting point defines a new problem instance for which a solution is generated (see Figure 1). All these solutions the heuristic provides are evaluated with respect to original data. Thus, the alternative problem instances are only used to appropriately modify the behaviour of the heuristic.

New problem instances are generated perturbing original data by means of an appropriate problem-specific procedure. This procedure is parameterized by a perturbation vector D that indicates how to modify each problem datum. Assuming that the heuristic H provides good solutions for the class of problems being solved, it is expected that the magnitude $\|D\|$ of the perturbation vector D needed to find the optimum be not very large.

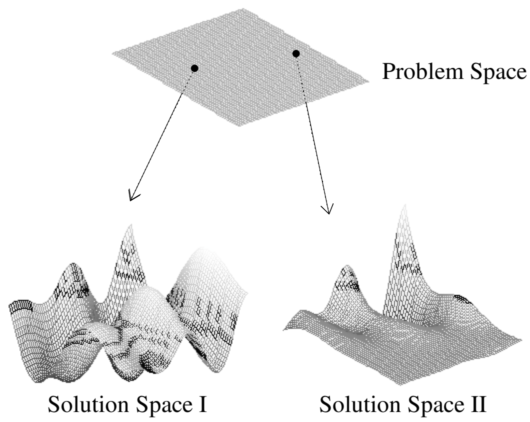

Fig. 1. Every point in the Problem Space defines a different solution space. The goal is to find a fitness landscape that optimally matches the requirements of a given heuristic.

G.D. Smith, N.C. Steele, R.F. Albrecht (eds.)
Artificial Neural Nets and Genetic Algorithms 3
Springer-Verlag Wien New York 1998
pp. 250-254

Notice that, strictly speaking, the search is done in an auxiliary space (the perturbation space) that yields a problem space when composed with original data.

3 The 0-1 Multiple Knapsack Problem

The 0-1 Multiple Knapsack Problem (the 0-1 MKP after this) is a well-known member of the NP-hard class [5]. It is also referenced as the 0-1 integer programming problem or the 0-1 linear programming problem.

3.1 Definition of the Problem

The 0-1 MKP is a generalisation of the 0-1 simple Knapsack problem. In the latter, a set of objects $O = \{o_1, \ldots, o_n\}$ and a knapsack of capacity C are given. Each object o_i has an associated profit p_i and weight w_i. The objective is to find a subset $S \subseteq O$ such that the weight sum over the objects in S does not exceed the knapsack capacity and yields a maximum profit. The 0-1 MKP involves m knapsacks of capacities $c_1, \ldots, c_m$. Every selected object must be placed in all m knapsacks. Moreover, the weight of an object o_i is not fixed, but it has a different value in each knapsack. Again, the goal is to obtain the maximum profit.

3.2 A Construction Heuristic

The greedy approach is a typical construction heuristic for the 0-1 simple Knapsack problem. This algorithm first calculates the profit density $\delta_i = p_i/w_i$ of every object and sorts them by decreasing values of this ratio. Then, objects are successively taken and included in the knapsack if they fit in it. To generalise this heuristic for the 0-1 MKP, the profit density of every object in every knapsack is calculated, and only the lowest value for each object is considered (i.e., $\delta_i = min(c_j \cdot p_i/w_{ij}) = = p_i/max(w_{ij}/c_j),\ 1 \leq j \leq m$). This is better than taking $\delta_i = avg(c_j \cdot p_i/w_{ij})$ since the latter could hide extreme values of the weight ratio c_j/w_{ij} for a certain knapsack.

4 The Hybrid Genetic Algorithm

A genetic algorithm for the 0-1 MKP performing problem-space search may be designed using the greedy construction algorithm described above. Each individual in the population represents a perturbation vector defining a new problem instance.

4.1 Perturbing data

New problem instances may be obtained from a given one modifying the profit p_i of each object. This is an appropriate procedure because all instances generated this way have the same set of feasible solutions, although the evaluation of every solution is different. Thus, each perturbation vector D is a list of n numbers $\{d_1, \ldots, d_n\}$ defining a problem instance in which the object o_i has a profit of p_i+d_i and the same original weights. The magnitude of the perturbation is problem-dependent and must be large enough to allow the optimum to be generated. Let ξ be the amount of profit that has to be added to the object with lowest profit density so it becomes the one with the highest ratio. Taking d_i from $[-\xi, +\xi]$ suffices to ensure that any feasible solution may be generated by the heuristic.

4.2 Experimental results

Experiments have been carried out with an elitist generational genetic algorithm, using a population size of $\mu = 100$, a crossover rate of $p_c=0.9$, a mutation rate of $p_m=0.01$ and proportional selection. The crossover operator is Radcliffe's n-dimensional R^3 operator [6], i.e., it picks at each locus a random value in the range defined by the corresponding parents' alleles. The mutation operator replaces a component of D by a random value.

For comparison purposes with [7] we have chosen a benchmark composed of nine problems taken from the OR-Library by Beasley [8]. Each run of the genetic algorithm involves $2 \cdot 10^4$ evaluations, and a total number of 100 runs are executed for each test problem. Results are shown in table 1.

Table 1. Experimental results obtained by the GA.

Problem	*n*	*m*	Optimum	Avg.	%Opt. found
Knap15	15	10	4015	4015.0	100%
Knap20	20	10	6120	6119.4	94%
Knap28	28	10	12400	12400.0	100%
Knap39	39	5	10618	10609.8	60%
Knap50	50	5	16537	16512.0	46%
Sento1	60	30	7772	7767.9	75%
Sento2	60	30	8722	8716.5	39%
Weing7	105	2	1095445	1095386.0	40%
Weing8	105	2	624319	622048.1	29%

These results clearly improve those referenced in [7] (using a binary encoding and a graded penalty term in the fitness function). They are much better not only in average but also in the number of times the optimum is found. For example, the larger instances (Sento1,

G.D. Smith, N.C. Steele, R.F. Albrecht (eds.)
Artificial Neural Nets and Genetic Algorithms 3
Springer-Verlag Wien New York 1998
pp. 250-254

Sento2, Weing7, Weing8) are solved to optimality at least five times more often (with the number of evaluations being a factor of ten lower).

5 The Exact Solution: Branch & Bound

An exact method has been applied to the above test problems to obtain a measure of the effort that it is required to optimally solve them. Since the search space is composed of 2^n points for a problem with n objects (the number of knapsacks does not enlarge it but may make the feasible region be smaller) a simple Branch & Bound (B&B) algorithm (as described in [9]) is unable to solve most of the problems above. Therefore, a more sophisticated version of the algorithm using the linear-programming relaxation [10] has been considered. This B&B algorithm solves the LP-relaxed version of the problem and then branches on the object with highest profit that has a fractional value, generating two subproblems in which that object is forced to be included/excluded respectively.

As shown in Figure 2, ~10^3 subproblems must be solved to obtain the optimal solution for the most difficult problem (Sento2). This is not a very high value, suggesting that the above problems are relatively *easy*. For that reason, harder problem instances were generated as shown in [9], i.e. randomly picking the weights w_{ij} from the interval [0,1], setting all knapsack capacities to n/3 and using object profits to define three levels of difficulty (from easiest to hardest): $p_i \in [0,1]$, $p_i \in [.45,.55]$, and p_i=.5. Results are shown in table 2.

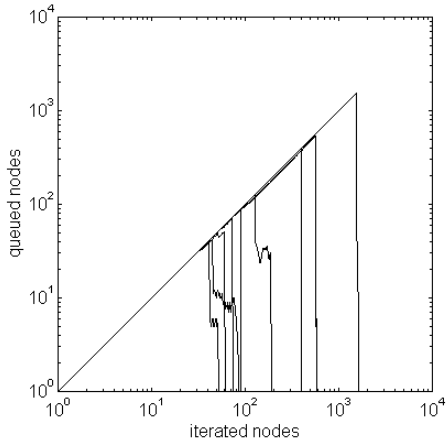

Fig. 2. Queued vs. Iterated nodes (OR-Library problems).

Table 2. B&B results for the generated problems.

n $(=m)$	$p_i \in [0,1]$	$p_i \in [.45,.55]$	p_i=.5
	Iterations		
20	67	6925	37373
30	90	37259	295210
40	80	>250000	-
	Max. queue length		
20	64	2163	7383
30	89	6997	50872
40	77	>150000	-

It can be seen that the B&B algorithm is unpractical for homogeneous (p_i=.5) or nearly homogeneous ($p_i \in [.45,.55]$) problems of size around 30-40.

6 A Distributed Version of the Hybrid GA

Finally, a coarse-grain parallel version of the hybrid genetic algorithm following the island model [12] (i.e., k genetic algorithms running in parallel, periodically interchanging some individuals) is tested. This model is used to study the improvement that can be achieved in both the *easy* and *hard* problem instances discussed above.

The experiments carried out so far have been realised with 2 and 4 genetic algorithms. The population size is μ=100/k and one individual migrates every 20 generations to a random subpopulation (no fine tuning has been attempted). As shown in Figures 3 and 4, the quality of the results (in terms of how often the optimum is found) is notably improved.

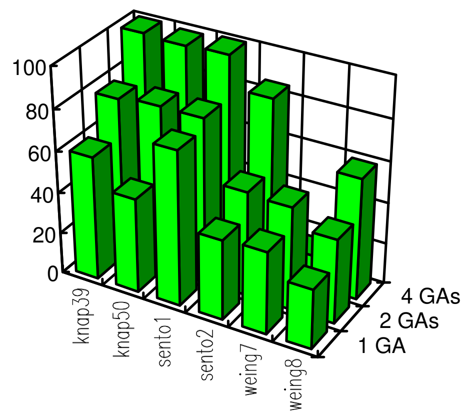

Fig. 3. Number of times the optimum was found using a distributed genetic algorithm (results for the OR-Library problems).

G.D. Smith, N.C. Steele, R.F. Albrecht (eds.)
Artificial Neural Nets and Genetic Algorithms 3
Springer-Verlag Wien New York 1998
pp. 250-254

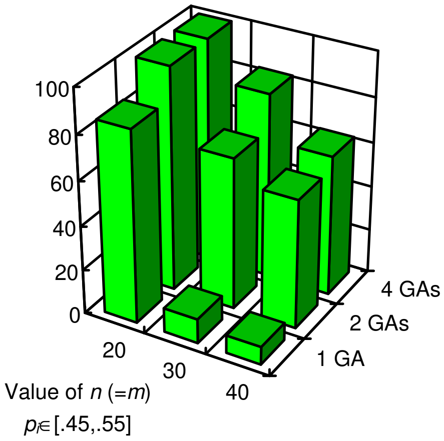

Fig. 4. Number of times the optimum was found using a distributed genetic algorithm (results for the randomly generated problem instances).

It must be noted that the B&B-*hard* instances were solved to optimality in 100 out of 100 runs both for the easiest ($p_i \in [0,1]$) and the homogeneous case (p_i=.5) with a sequential genetic algorithm. Thus, a genetic algorithm seems to be more appropriate than B&B for this kind of problems.

7 Conclusions

A hybrid genetic algorithm using problem-space search has been presented and applied to the 0-1 Multiple Knapsack Problem. This technique is appropriate when a problem-dependent construction heuristic and a perturbing algorithm are available. The resulting hybrid algorithm performs better than other GA-based approaches on problems taken from the literature. Moreover, it consistently solves to optimality randomly-generated problem instances that are very hard for an exact (and specialised) technique.

Two important advantages of this method can be stressed: on the one hand, the results are assured to be at least as good as those of the construction heuristic as long as a null perturbation vector is inserted into the initial population. On the other hand, perturbation vectors are usually points in a *n*-dimensional numerical domain. This allows using techniques oriented to continuous parameter optimisation (e.g., evolution strategies [13]) for the resolution of discrete-domain problems without needing to define artificial encodings. Furthermore, the use of the construction heuristic avoids needing to handle unfeasible solutions.

References

[1] Wolpert D H and Macready W G, “No Free Lunch Theorems for Search”, Sante Fe Institute Technical Report SFI-TR-95-02-010, 1995

[2] Davis L (ed.), “Handbook of Genetic Algorithms”, Van Nostrand Reinhold Computer Library, NY, 1991

[3] Mühlenbein H, “Parallel genetic algorithm, population dynamics and combinatorial optimization”, *Proceedings of the Third International Conference on Genetic Algorithms*, Schaffer H (ed.), Morgan Kaufmann, San Mateo, 1989, pp. 416-421

[4] Storer R H, Wu S D and Vaccari R, “New search spaces for sequencing problems with application to job shop scheduling”, *Management Science*, 38, 1992, pp. 1495-1509

[5] Garey M and Johnson D, “Computers and intractability: A guide to the theory of NP-Completeness”, Freeman and Co., San Francisco, 1979

[6] Radcliffe N J, “Forma Analysis and Random Respectful Recombination”, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Belew R K and Booker L B (eds.), Morgan Kaufmann Pub., San Mateo CA, 1991, pp. 222-229

[7] Khuri S, Bäck T and Heitkötter J, “The Zero/One Multiple Knapsack Problem and Genetic Algorithms”, *Proceedings of the ACM Symposium of Applied Computation*, ACM Press, 1993

[8] Beasley J E, “OR-Library: Distributing Test Problems by Electronic Mail”, *Journal of Operational Research Society*, 41(11), 1990, pp. 1069-1072

[9] Horowitz E and Sahni S, “Fundamentals of Computer Algorithms”, Computer Science Press, 1978

[10] Wiston W L, “Operations Research. Applications and algorithms”, Duxbury Press, Belmont CA, 1993

[11] Peterson C and Södeberg B, “Artificial Neural Networks”, *Modern heuristic techniques for combinatorial problems*, Reeves C R (ed.), Advanced Topics in Computer Science, Oxford Scientific Publications, 1993, pp. 197-242

[12] Chipperfield A and Fleming P, “Parallel Genetic Algorithms”, *Parallel and Distributed Computing Handbook*, Zomaya A Y (ed.), McGraw-Hill Series on Computing Engineering, 1996, pp. 1118-1143

[13] Rechember I, “Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution”, Frommann-Holzboog Verlag, Stuttgart, 1973