Journal of Computer Science 5 (8): 544-547, 2009
ISSN 1549-3636

An Improved Genetic Algorithm Based on Adaptive Repair Operator for Solving the Knapsack Problem

Sunanda Gupta and M.L. Garg
School of Computer Science and Engineering,
Shri Mata Vaishno Devi University, Katra, J and K, India

Abstract: Problem statement: Knapsack problem is a typical NP complete problem. During last few decades, Knapsack problem has been studied through different approaches, according to the theoretical development of combinatorial optimization. **Approach:** In this study, modified evolutionary algorithm was presented for 0/1 knapsack problem. **Results:** A new objective_func_evaluation operator was proposed which employed adaptive repair function named as repair and elitism operator to achieve optimal results in place of problem specific knowledge or domain specific operator like penalty operator (which are still being used). Additional features had also been incorporated which allowed the algorithm to perform more consistently on a larger set of problem instances. **Conclusion/Recommendations:** This study also focused on the change in behavior of outputs generated on varying the crossover and mutation rates. New algorithm exhibited a significant reduction in number of function evaluations required for problems investigated.

Key words: Knapsack problem, genetic algorithm, adaptive repair operator

I. INTRODUCTION

Knapsack problem is a well known and well studied problem in combinatorial optimization being widely used in areas like network planning, network routing, parallel scheduling and budgeting[1]. Mathematically the 0-1 Knapsack problem may be formulated as:

$$\text{Maximize}: \quad \sum_{j=1}^{n} p_i x_i$$

$$\text{Subject to}: \quad \sum_{j=1}^{n} w_j x_j \le c$$

$$xj = 0 \text{ or } 1 \text{ and } j = 1, 2, ...n$$

where, x_j the number of each kind of item is restricted to one or zero indicating its presence or absence in the knapsack. c is the capacity of the knapsack into which n types of objects may be placed. The object of type i has a profit p_i and weight w_i, associated with it.

Since the Knapsack problem is NP problem, various approaches presently available such as dynamic programming, backtracking, branch and bound etc. are not very useful for solving it. These exact algorithms have a running time that is bounded by an exponential function of length of input data, thus it is very difficult to obtain the exact solutions in case of many large scale knapsack instances which come from practical applications[2]. Hence, for those large scale instances, it has to rely on heuristic algorithms to obtain the near optimal solutions to them.

Amongst the Heuristic algorithms for knapsack problem, genetic algorithm is an effective method to solve the knapsack instances approximately. Genetic algorithm is a search technique to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics[3,4]. They work on the Darwin's principle of natural selection and survival of the fittest. Evolutionary algorithm produces initial population with individuals selected at random. The fitness value of each individual of the population is worked out. If it does not satisfy the goal criterion then it is improved through crossover and mutation operators. Because of the incompatibility between the speed of convergence and the action for searching for the best solution, the simple evolutionary algorithm always behave slowly in convergence, easily in prematurity and plunges into local optimizations. The Evolutionary algorithm with adaptive repair operator is proposed in this study. It

Corresponding Author: Sunanda Gupta, School of Computer Science and Engineering, Shri Mata Vaishno Devi University, Katra, J and K, India

leads the search direction of the population by collecting some better individuals from every generation by using the elitism operator and hence results in improving the searching efficiency. Furthermore, the computational cost implications of using adaptive repair function in the algorithm have also been evaluated.

Evolutionary algorithm with adaptive repair operator: A modified algorithm called KNAP-GA is described in this section for solving KP using elitist GAs. The structure of the proposed KNAP-GA is presented in Fig. 1.

Here P represents the population $\{P_0, P_1, P_2 \ldots P_{max}\}$ where each population comprises of m entries $\{S_1, S_2, \ldots S_m\}$ and each entry $S_i = \{J_k$ such that $1 \leq k \leq n$ and $J_k = \{0, 1\}\}$. Let $J = \{J_1, J_2, J_3, \ldots \ldots J_n\}$ represents the set of unique items that are available . The selected items are represented by value 1 and the not selected ones are represented by value 0. The fitness values of the chromosomes in a population are evaluated based on the profits associated with the items. The best fit chromosome generated from a given population is stored in S^*. The crossover and mutation rate are varied to study the impact of varying these operators on the outputs obtained. The working, the characteristics and the need of objective_func_evaluation operator is explained in the preceding section. Each generation of the population yields a set of unique items that would result in maximum profit, thus modifying the solution vector in each step by proceeding towards the best fit solution.

Objective_func_evaluation operator Δ: The objective_func_evaluation is a function which evaluates the fitness of chromosomes in the new population. For the Knapsack problem, we calculate the fitness of each chromosome by summing up the benefits of the items that are included in the knapsack, restricting the total weight of the selected items to the capacity of knapsack. If the total weight of the items in the chromosome generated by the crossover and mutation procedure is greater than the capacity of the knapsack then that solution is infeasible and the Repair Operator and the elitism come into play. The Objective_func_evaluation operator is basically a combination of Elitism and Repair operator which is simple to implement and provides us with similar results as obtained with other operators which need complex computations. The outline of the algorithm is given in Fig. 2.

1. Set t: = 0;
2. Initialize P (t):= $\{S_1, S_2 \ldots S_M\}$, such that $S_i = \{J_k$ where $1 \leq k \leq n$, $J_k = \{0,1\}$ /* n is the number of unique jobs */
3. Evaluate P (t):= $\{f(S_1), f(S_2) \ldots f(S_M))$; $f(S_1) = \sum$fitness J_i where $1 \leq i \leq n$ | only if $J_i = 1$};
4. Find $S^* \in$ P (t) such that $f(S^*) \leq f(S)$, for all $S \in$ P (t) /* S^* stores the most fit chromosome*/
5. **while** $t < t_{max}$ **do**
6. select (S_i, S_j): = ϕ(P(t)) /* ϕ = binary tournament operator */
7. crossover C: = Ω_c (S_i, S_j); /* Ω_c = uniform crossover operator */
8. mutate $C \leftarrow \Omega_m(C)$ /* Ω_m = mutation operator */
9. **if** C ≡ any $S \in$ P(t) **then** discard C and go to step 6
10. **end if**
11. evaluate $f(C)$
12. if $f(C) > f(S^*)$ then
13. $S^* \leftarrow C$; /* update best fit chromosome found*/
14. **end if**
15. Apply *objective_func_evaluation operator* Δ to evaluate the unfit chromosome and accordingly apply the repair operator and elitism operator to convert them into fit chromosomes.
16. $t \leftarrow t+1$;
17. end while
18. return S, $f(S^*)$

Fig. 1: Algorithm for KNAP-GA

1. Count= 0;
2. **for** j = 1 to n {n represents the number of chromosomes in a population}
   If (Chromosome [j]. fitness > Capacity) **then**
   Count = Count + 1
   end if
3. **If** Count < (population size × 0.02) **then**
   Elitism
   {
   The unfit chromosome is replaced by the most fit chromosome of the previous generation, the second unfit chromosome replaced by the second most fit chromosome of previous generation and so on.
   }
   else
   Repair Operator
   {
   /* It makes all the bits of the chromosome equal to zero from the point when the fitness exceeds the capacity.*/
   Let R = the accumulated weight of the chromosome S of length *l* and C = capacity of the Knapsack.
   1. **Begin**
   2. Initialize R= $\Sigma w_j S[j]$ for $1 \leq j \leq 1$
   3. Initialize j = 1
   4. **While** ((R > C) and (j ≥ 1)) **do**
   5. **If** (S [j] = 1) **then**
   6. set S [j] = 0 and R= R - w_j
   7. **end if**
   8. j = j - 1
   9. **end while**
   10. **End**
   }

Fig. 2: Objective_func_evaluation operator Δ

MATERIALS AND METHODS

The test data represents various instances of the 0/1 Knapsack problem as available in the literature[1]. The data sets consist of varying correlation types between profits and weights.

Correlation types:

Data set Uncorrelated (UC)
w_i = (uniformly) random (1…v) and
p_i = (uniformly) random (1…v)

Data set Weakly Correlated (WC)
w_i = (uniformly) random (1…v) and
p_i = w_i + (uniformly) random (-r..r)

Data set Strongly Correlated (SC)
w_i = (uniformly) random (1…v) and
p_i = w_i + r

The experiments have been conducted without sorting the items on p_i/w_i values. Knapsack problem instances were studied on data sets with 100, 250 and 500 items for each of the three correlation types and the three types of Knapsack capacities. However, due to space limitation we are presenting only the case with 250 items.

KNAP-GA was implemented on Pentium-4 (1.7 Ghz) and the results were compared with those obtained by the dynamic programming algorithm. We ran 100 instances of the random sets. Each table entry is the average of 30 runs.

RESULTS

The effects of varying crossover and mutation rates have been studied and is presented in Table 1. When the crossover and mutation rates are both set to 0, the population has obviously contained copies of the strategies randomly generated at the beginning. In other words, with no with no crossover or mutation, all the children looked exactly like one of their parents.

However, on setting crossover rate equal to zero and varying the mutation rate it was observed that the mutation rate of n/100 (where n can be anything between 0-9) yields the best results. Due to space constraint we have shown only 3 instances of the mutation rate i.e., 03, 0.003, 0.70. Experimental results do not show the case when mutation is turned off, because it does not matter much what the crossover rate is when mutation is turned off as mutation is a function which is an integral part of crossover operator and if mutation is off, it yields the same results as when crossover and mutation are set equal to zero.

The second observation worth noticing from Table 1 is that crossover rate between 60 and 70 yields best solution with mutation rate having negligible impact on the output. On examining the three cases, without Elitism and Repair operator, with Repair operator and with elitism and Repair operator, we observed that the mere presence of Repair operator improved the final solution than the one that we were getting in the absence of Elitism and Repair operators and by clubbing the Elitism and Repair operators i.eby using objective_func_evaluation operator Δ, considerable improvement in the final solution has been found.

Table 1: Comparison of varying crossover and mutation rates

		c = 0	c = 0.30	c = 0.50	c = 0.60	c = 0.70	c = 0.90
Uncorrelated data							
Without elitism and repair operator	m = 0.003	2650	2607	2669	2669	2701	2625
	m = 0.030	2702	2791	2676	2810	2679	2761
	m = 0.700	2525	2540	2469	2981	2613	2490
With repair operator	m = 0.003	2661	2681	2716	2650	2771	2678
	m = 0.030	2663	2653	2806	2690	2667	2614
	m = 0.700	2576	2528	2590	2604	2611	2389
With elitism and repair operator	m = 0.003	3308	2254	3333	3310	3346	3365
	m = 0.030	3333	3318	3330	3336	3337	3323
	m = 0.700	3210	3225	3288	3265	3280	3265
Weakly correlated data							
Without elitism and repair operator	m = 0.003	2741	2717	2753	2771	2799	2783
	m = 0.030	2674	2684	2698	2784	2743	2706
	m = 0.700	2694	2642	2649	2733	2763	2705
With repair operator	m = 0.003	2654	2764	2807	2812	2741	2734
	m = 0.030	2777	2802	2782	2705	2833	2834
	m = 0.700	2684	2674	2725	2743	2756	2613
With elitism and repair operator	m = 0.003	3379	3406	3369	3378	3321	3347
	m = 0.030	3406	3394	3413	3444	3403	3407
	m = 0.700	3269	3250	3219	3250	3270	3202
Strongly correlated data							
Without elitism and repair operator	m = 0.003	2300	2336	2321	2326	2408	2345
	m = 0.030	2310	2304	2326	2351	2368	2370
	m = 0.700	2269	2254	2252	2272	2263	2258
With repair operator	m = 0.003	2311	2325	2354	2338	2334	2365
	m = 0.030	2349	2302	2318	2366	2358	2320
	m = 0.700	2253	2254	2281	2260	2246	2247
With elitism and repair operator	m = 0.003	2332	2372	2386	2404	2383	2379
	m = 0.030	2377	2393	2408	2404	2416	2382
	m = 0.700	2251	2249	2295	2284	2330	2283

DISCUSSION

The objective_func_evaluation operator incorporating the new adaptive repair operator has been found to be more cost effective than the existing techniques that have been used in the past to handle chromosomes yielding infeasible solutions. As per the philosophy Richardson *et al*.[5], (employing penalty method) infeasible bred strings (or chromosomes) were allowed to join the population however by reducing the string strength by adding penalty terms to the fitness. The farther the string is from feasibility, the higher is the penalty term. In the other approach flags were used to signal the feasibility of final solution. If after several runs on the same problem instance, the flag consistently indicated an infeasible string then one had to either bias the random number generator so as to produce strings in which the number of zeros is greater than the number of one's[6] or use some other heuristic such as greedy one, to generate a solution[7]. Both the methods, the penalty method or using flags increase the computation overhead because in addition to estimating the fitness of chromosomes in a population the track of unfit chromosomes had to be kept by assigning varying penalties and again estimating the strength in each iteration. Whereas if flags were used then one had to keep track of the status of the flag amongst several problem instances and if the same value persists then greedy approach is used thus increasing computational cost multi folds.

The algorithm employing objective_func_evaluation operator Δ provides a much efficient approach of reaching an optimal solution by using a simple to implement adaptive repair operator which incorporates a strategy of making all bits of the chromosome equal to zero from the point where the fitness exceeds the capacity and by employing elitism to get a small percentage of the best chromosomes from the old population to the new population. Thus the evolutionary algorithm with adaptive repair operator proposed in this study, leads the search direction of the population by collecting the few best individuals from every generation by using the elitism operator. It has resulted in improving the search efficiency.

CONCLUSION

The results obtained with the newly designed genetic operators in algorithm are encouraging, on the practical data sets. Rather than augmenting the genetic algorithm with domain specific knowledge, we have introduced a fitness function employing adaptive repair operator and elitism which is simple to use. In order to further the results, application of the developed KNAP-GA to real life problems, hybridization of local search techniques with other heuristic or meta heuristic techniques for solving the Knapsack problem may be studied. Future work may address whether the proposed algorithm can be applied to other constraint optimization problems such as the maximum clique or the degree constrained minimum spanning tree problems.

VII. REFERENCES

1. Martello, S. and P. Toth, 1990. Knapsack Problems: Algorithms and Computer Implementation. John Wiley and Sons, New York, ISBN: 0-471-92420-2, pp: 296.
2. Li, K.L., G.M. Dai and Q.H. Li, 2003. A genetic algorithm for the unbounded knapsack problem. Proceedings of the 2nd International Conference on Machine Learning and Cybernetics, No. 2-5, IEEE Xplore Press, USA., pp: 1586-1590. DOI: 10.1109/ICMLC.2003.1259749
3. Holland, J.H., 1975. Adaptation in Natural and Artificial Systems: Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. University of Michigan Press, ISBN: 0472084607, pp: 183.
4. Goldberg, D.E., 1989. Genetic Algorithm in Search Optimization and Machine Learning. 1st Edn., Addison-Wesley, New York, ISBN: 10: 0201157675, pp: 432.
5. Richardson, J.T., M.R. Palmer, G. Liepins and M. Hillard, 1989. Some guidelines for genetic algorithms with penalty functions. Proceedings of the 3rd International Conference on Genetic Algorithms, (ICCA'89), Morgan Kaufmann Publishers Inc., San Francisco, CA., USA., pp: 191-197. http://portal.acm.org/citation.cfm?id=657233
6. Khuri, S., T. Back and J. Heitkotter, 1994. The zero/one multiple knapsack problem and genetic algorithms. Proceeding of the ACM Symposium on Applied Computing, Mar. 06-08, ACM Press, Phoenix, Arizona, United States, pp: 188-193. http://portal.acm.org/citation.cfm?id=326619.326694
7. Moret, B.M.E. and H.D. Shapiro, 1991. Algorithms from P to NP, Design and Efficiency. Benjamin Cummings, Menlo Park, CA., USA., ISBN: 0-8053-8008-6, pp: 576.