Enhanced Quantum Evolutionary Algorithms for Difficult Knapsack Problems

C. Patvardhan[1], Apurva Narayan[1], and A. Srivastav

[1] Faculty of Engineering,
Dayalbagh Educational Institute,
Agra – 282005
cpatvardhan@hotmail.com, apurvanarayan@gmail.com
[2] Mathematisches Seminar, Christian-Albrechts-Universitat Zu Kiel, Kiel Germany
asr@numerik.uni-kiel.de

Abstract. Difficult knapsack problems are problems that are expressly designed to be difficult. In this paper, enhanced Quantum Evolutionary Algorithms are designed and their application is presented for the solution of the DKPs. The algorithms are general enough and can be used with advantage in other subset selection problems.

Keywords: Evolutionary Algorithms ,Quantum ,knapsack.

1 Introduction

The classical knapsack problem is defined as follows: Given a set of n items, each item j having an integer profit p_j and an integer weight w_j, the problem is to choose a subset of the items such that their overall profit is maximized, while the overall weight does not exceed a given capacity c. The problem may be formulated as the following integer programming model

$$\text{(KP)} \qquad \text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \le c, \tag{2}$$

$$x_j \in \{0,1\}, \quad j = 1,\ldots,n. \tag{3}$$

where the binary decision variables x_j are used to indicate whether item j is included in the knapsack or not. Without loss of generality it may be assumed that all profits and weights are positive, that all weights are smaller than the capacity c, and that the overall weight of the items exceeds c.

The standard knapsack problem (SKP) is *NP*-hard in the weak sense, meaning that it can be solved in pseudo-polynomial time through dynamic programming. The SKPs are quite easy to solve for the most recent algorithms [1-5]. Various branch-and-bound algorithms for SKPs have been presented. The more recent of these solve a core problem, i.e. an SKP defined on a subset of the items where there is a large probability of finding an optimal solution. The MT2 algorithm [1] is one of the most advanced of these algorithms. Realizing that the core size is difficult to estimate in

A. Ghosh, R.K. De, and S.K. Pal (Eds.): PReMI 2007, LNCS 4815, pp. 252 – 260, 2007.

advance, Pisinger [5] proposed to use an expanding core algorithm. Martello and Toth [3] proposed a special variant of the MT2 algorithm which was developed to deal with hard knapsack problems. One of the most successful algorithms for SKP was presented by Martello, Pisinger and Toth [7]. The algorithm can be seen as a combination of many different concepts and is hence called Combo.

The instances do not need to be changed much before the algorithms experience a significant difficulty. These instances of SKP are called the Difficult Knapsack Problems (DKP) [6]. These are specially constructed problems that are hard to solve using the standard methods that are employed for the SKPs. Pisinger [6] has provided explicit methods for constructing instances of such problems.

Heuristics which employ history of better solutions obtained in the search process, viz. GA, EA, have been proven to have better convergence and quality of solution for some "difficult" optimization problems. But, still, problems of slow/premature convergence remain and have to be tackled with suitable implementation for the particular problem at hand. Quantum Evolutionary Algorithms (QEA) is a recent branch of EAs. QEAs have proven to be effective for optimization of functions with binary parameters [8, 9].

Although the QEAs have been shown to be effective for SKPs in [8], their performance on the more difficult DKPs has not been investigated. Exact branch and bound based methods are not fast for DKPs. This provides the motivation for attempting to design newer Enhanced QEAs (EQEAs) with better search capability for the solution of the DKPs. In this paper, such EQEAs are presented. The QEAs are different from those in [8] as they include a variety of quantum operators for executing the search process. The computational performance of the EQEAs is tested on large instances of nine different varieties of the DKPs i.e. for problems up to the size of 10,000. The results obtained are compared with those obtained by the greedy heuristic. It is seen that the EQEAs are able to provide much improved results.

The rest of the paper is organized as follows. In section 2 we describe the various types of DKPs reported in the literature. A brief introduction to the QEAs and some preliminaries regarding the QEAs are provided in section 3. The EQEA is given in the form of pseudo-code in section 4. The computational results are provided section 5. Some conclusions are derived in section 6.

2 Difficult Knapsack Problems (DKPs)

Several groups of randomly generated instances of DKPs have been constructed to reflect special properties that may influence the solution process in [6]. In all instances the weights are uniformly distributed in a given interval with *data range with* R =1000. The profits are expressed as a function of the weights, yielding the specific properties of each group. Nine such groups are described below as instances of DKPs. The first six are instances with large coefficients for profit and the last three are instances of small coefficients for profits.

(i) Uncorrelated data instances: p_j and w_j are chosen randomly in [1,R]. In these instances there is no correlation between the profit and weight of an item.

(ii) Weakly correlated instances: weights w_j are chosen randomly in [1,R] and the profits p_j in [w_j- R/10 , w_j + R/10] such that p_j >= 1. Despite their name, weakly correlated instances have a very high correlation between the profit and weight of an

item. Typically the profit differs from the weight by only a few percent.

(iii) Strongly correlated instances: weights w_j are distributed in [1,R] and $p_j = w_j+R/10$. Such instances correspond to a real-life situation where the return is proportional to the investment plus some fixed charge for each project.

(iv) Inverse strongly correlated instances: profits p_j are distributed in [1,R] and $w_j=p_j+R/10$. These instances are like strongly correlated instances, but the fixed charge is negative.

(v) Almost strongly correlated instances: weights w_j are distributed in [1,*R*] and the profits p_j in [w_j + R/10 – R/500, w_j + R/10 + R/500]. These are a kind of fixed-charge problems with some noise added. Thus they reflect the properties of both strongly and weakly correlated instances.

(vi) Uncorrelated instances with similar weights: weights w_j are distributed in [100000 , 100100] and the profits p_j in [1, 1000].

(vii) Spanner instances *(v,m)***:** These instances are constructed such that all items are multiples of a quite small set of items — the so-called spanner set. The spanner instances span(*v,m)* are characterized by the following three parameters: *v* is the size of the spanner set, *m* is the multiplier limit, and any distribution (uncorrelated, weakly correlated, strongly correlated, etc.) of the items in the spanner set may be taken. More formally, the instances are generated as follows: A set of *v* items is generated with weights in the interval [1,*R*], and profits according to the distribution. The items (p_k,w_k) in the spanner set are normalized by dividing the profits and weights with *m*+1. The *n* items are then constructed, by repeatedly choosing an item $[p_k,w_k]$ from the spanner set, and a multiplier *a* randomly generated in the interval [1,*m*]. The constructed item has profit and weight (a . p_k , $a.w_k$).The multiplier limit was chosen as $m = 10$.

(viii) Multiple strongly correlated instances mstr(k_1,k_2,d)**:** These instances are constructed as a combination of two sets of strongly correlated instances.Both instances have profits $p_j := w_j + k_i$ where k_i= 1 , 2 is different for the two instances. The multiple strongly correlated instances mstr(k_1,k_2,d) are generated as follows:

The weights of the *n* items are randomly distributed in [1,*R*]. If the weight w_j is divisible by *d*, then we set the profit $p_j := w_j + k_1$ otherwise set it to $p_j := w_j + k_2$. The weights w_j in the first group (i.e. where pj = wj + k_1) will all be multiples of *d*, so that using only these weights at most *d* [*c/d*] of the capacity can be used. To obtain a completely filled knapsack some of the items from the second distribution need to be included.

(ix) Profit ceiling instances pceil(d)**:** These instances have the property that all profits are multiples of a given parameter *d*. The weights of the *n* items are randomly distributed in [1,*R*], and the profits are set to $p_j = d[w_j/d]$. The parameter *d* was chosen as *d*=3.

3 QEA Preliminaries

QEA is a population-based probabilistic Evolutionary Algorithm that integrates concepts from quantum computing for higher representation power and robust search.

Qubit

QEA uses qubits as the smallest unit of information for representing individuals. Each

qubit is represented as $q_i = \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}$. State of the qubit ($q_i$) is given by $|\Psi_i\rangle = \alpha_i|0\rangle + \beta_i|1\rangle$, α_i and β_i are complex numbers representing probabilistic state of qubit, i.e. $|\alpha_i|^2$ is probability of the state being 0 and $|\beta_i|^2$ is the probability of the state being 1, such that $|\alpha_i|^2+|\beta_i|^2=1$. For purposes of QEA, nothing is lost by regarding α_i and β_i to be real numbers. Thus, a qubit string with n bits represents a superposition of 2^n binary states and provides an extremely compact representation of the entire search space.

Observation

The process of generating binary strings from the qubit string, Q, is called Observation. To observe the qubit string (Q), a string consisting of same number of random numbers between 0 and 1 (R) is generated. The element P_i is set to 0 if R_i is less than square of Q_i and 1 otherwise. Table1 represents the observation process.

Table 1. Observation of qubit string

i	*1*	*2*	*3*	*4*	*5*	.....	*Ng*
Q	0.17	0.78	0.72	0.41	0.89	.....	0.36
R	0.24	0.07	0.68	0.92	0.15	.....	0.79
P	1	0	0	1	0	.....	1

Updating qubit string

In each of the iterations, several solution strings are generated from Q by observation as given above and their fitness values are computed. The solution with best fitness is identified. The updating process moves the elements of Q towards the best solution slightly such that there is a higher probability of generation of solution strings, which are similar to best solution, in subsequent iterations. A Quantum gate is utilized for this purpose so that qubits retain their properties [8]. One such gate is rotation gate, which updates the qubits as

$$\begin{bmatrix} \alpha_i^{t+1} \\ \beta_i^{t+1} \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i^{t} \\ \beta_i^{t} \end{bmatrix}$$

where, α_i^{t+1} and β_i^{t+1} denote probabilities for i^{th} qubit in $(t+1)^{\text{th}}$ iteration and $\Delta\theta_i$ is equivalent to the step size in typical iterative algorithms in the sense that it defines the rate of movement towards the currently perceived optimum.

The above description outlines the basic elements of QEA. The qubit string, Q, represents probabilistically the search space. Observing a qubit string 'n' times yields 'n' different solutions because of the probabilities involved. Fitness of these is computed and the qubit string, Q, is updated towards higher probability of producing strings similar to the one with highest fitness. The sequence of steps continues. The above ideas can be easily generalized to work with multiple qubit strings. Genetic operators like crossover and mutation can then be invoked to enhance the search power further.

4 Enhanced Quantum Evolutionary Algorithm

The algorithm is explained succinctly in the form of a pseudo-code below.

Notation

- Max = number of items in DKP, Cap = Capacity of knapsack
- Profit = Array of profit by selecting each item
- Weight = Array of weight by selecting each item
- Gsol = DKP Solution by Greedy heuristic, GProfit = Profit of items in Gsol
- P, ep ,rp : Strings of Quantum bits used in search
- NO1 = Operator used to evolve ep towards best solution found so far
- NO2 = Operator used to evove rp randomly
- NO3 = Rotation operator used to evolve p
- bestcep = String produced by operator NO1 giving best result on observation
- bestcrp = Quantum string produced by NO2 giving best result on observation
- Best = Best profit found by EQEA of solution maxsol

Algorithm EQEA

1	Initialize iteration number t=0, cap, profit, weight , max
2	Sort items in descending order of profit/weight
3	Find greedy solution G with profit Gcost.
4	Best = Gcost , Maxsol = G
5	Initialize for every 'k' If G[k] ==1 p[k] =ep[k]=rp[k]=0.8 else p[k] =ep[k]=rp[k]=0.2
6	Best Cap = ep , Best Cap = rp /*Initialization*/
7	Observe p[k] to get solution with cost tcost ; if(tcost > Best){Best = tcost; Maxsol = tsol;}
8	while (termination_criterion != TRUE) Do Steps 9-15

9	Apply NO1 to p to generate the current rp i.e crp;
	for(i=0;i<num1;i++)
	{ Observe crp to obtain solution rsol with cost rcost;
	if(rcost > Best)
	{Crossover rsol with Maxsol to obtain tsol with cost tcost;
	If(tcost>rcost) {rsol = tsol;rcost = tcost}
	bestcrp = crp;}
	If(rcost >Best) {Best = rcost ; maxsol = rsol;}

10	Repeat step 9 with NO2 on p to obtain bestcep, ecost and esol;
11	Apply NO3 on p

12	For(i = 0 ; i < max; i++)
	{qmax[i] = findmax(bestcep[i], bestcrp[i],p[i]);
	qmin[i] = findmix(bestcep[i], bestcrp[i],p[i]);
	if(p[i] > 0.5)
	{{if((maxsol[i]==1)and (bestcep[i]==1)and (bestcrp[i]==1)) p[i]=qmax[i];}
	else {if((maxsol[i]==0)and (bestcep[i]==0)and (bestcrp[i]==0)) p[i]=qmin[i];}}

13	Settled = number of p[i]'s with $\alpha_i > 0.98$ or $\alpha_i < 0.02$
14	t =t +1
15	If((t > iter _count) or (settled > 0.98 * max)) termination _ criterion = TRUE

Algorithm EQEA starts with the greedy solution and initializes the qubit strings in accordance with the greedy solution as in step 4. Observe operation in step 5 is a modified form of observation described in section 3. In this, the solution string resulting from the observation is checked for violation of capacity constraint and repaired, if necessary, using a greedy approach i.e. selected items are deselected in increasing order of profit/weight till capacity constraint is satisfied. The crossover operator is a simple two-point crossover with greedy repair of constraint violations.

Evolving Qubits

In EQEA, the qubit is evolved state of the qubit, which is a superposition of state 0 and 1, is shifted to a new superposition state. This change in probability magnitudes $|\alpha|^2$ and $|\beta|^2$ with change in state is transformed into real valued parameters in the problem space by two neighborhood operators.

Neighborhood Operator 1 (NO1) generates a new qubit array crp from the rp. An array R is created with max elements generated at random such that every element in R is either +1 or -1. Let ρ_k be the k^{th} element in R. Then θ_k^t is given by

$$\theta_k^t = \theta_k^{t-1} + \rho_k * \delta \tag{4}$$

where, δ is alteration in angle and θ_k^t is the rotated angle given by $\arctan(\beta_k^t / \alpha_k^t)$.

δ is randomly chosen in the range $[0, \theta_k^{t-1}]$ if $\rho_k = -1$ and in the range $[\theta_k^{t-1}, \pi/2]$ if $\rho_k = +1$.

The new probability amplitudes, α_k^t, β_k^t are calculated using rotation gate as

$$\begin{bmatrix} \alpha_{ijk}^t \\ \beta_{ijk}^t \end{bmatrix} = \begin{bmatrix} \cos\delta & \sin\delta \\ -\sin\delta & \cos\delta \end{bmatrix} \begin{bmatrix} \alpha_{ijk}^{t-1} \\ \beta_{ijk}^{t-1} \end{bmatrix} \tag{5}$$

Neighborhood Operator 2 (NO2) NO2 works just as NO1 except that it generates a point between ep and BEST. It is primarily utilized for exploitation of search space.

The rationale for two neighborhood operators is as follows. NO1 has a greater tendency for exploration. NO2 has a greater tendency towards exploitation because, as the algorithm progresses, the values of ep converge towards BEST. Table 2 shows the frequency of use of NO1 and NO2.

Table 2. Frequency of use of NO1 and NO2

Stage of search	Proportion of NO1 (%)	Proportion of NO2 (%)
First one-fifth iterations	90	10
Second one-fifth iterations	70	30
Third one-fifth iterations	50	50
Forth one-fifth iterations	30	70
Last one-fifth iterations	10	90

The neighborhood operators thus evolve new quantum strings from the existing strings. This approach removes all disadvantages of binary representation of real numbers while, at the same time, balances exploration and exploitation in the sense that it adopts the "step-size" from large initially to progressively smaller size.

Updating Qubit String(NO3)

In the updating process, the individual states of all the qubits in p are modified so that probability of generating a solution string which is similar to the current best solution is increased in each of the subsequent iterations. Amount of change in these probabilities is decided by Learning Rate, $\Delta\theta$, and is taken as 0.001π. Updating process is done as explained in section 2. Table 3 presents the choice of $\Delta\theta$ for various conditions of objective function values and i^{th} element of p and BEST in t^{th} iteration. F(p) is the profit of the current solution observed from p. F(BEST) is the profit of maxsol.

Table 3. Calculation of $\Delta\theta$ for t^{th} iteration

Fitness	*Elemental Values*	$\Delta\theta$
X	$p_i = BEST\ i$	0
F(BEST)>F(p)	$p_i > BESTi$	0.001π
	$p_i < BESTi$	-0.001π
F(BEST)<F(p)	$p_i > BESTi$	-0.001π
	$p_i < BESTi$	0.001π
F(p)=F(BEST)	$p_i > BEST_i$	0
	$p_i < BEST_i$	0

The updating process is illustrated in figure 2 for k^{th} element for t^{th} iteration i.e. changes in state of k^{th} qubit and corresponding change in probability amplitudes. Findmax finds the maximum of the three arguments whereas findmin finds the minimum of the three arguments.

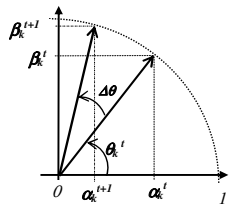

Fig. 2. Updating k^{th} element of qubit string Q

5 Results of Computational Experiments

The computational experiments have been performed on the above mentioned nine types of DKPs for the data range R = 1000 and the number of items ranging from 500 to 10000. For each instance type a series of $H = 9$, $c = \frac{h}{H+1}\sum_{j=1}^{n} w_j$, h = 1,., . . .H

instances is performed. The EQEA performed better in most of the cases where the problem did not get settled up with a definite structure. The table 4 shows computational results. AGP is the average profit in solutions obtained by the greedy algorithm whereas AQP is the average profit of solutions by the EQEA algorithm.

Table 4. Shows the relative values of solutions by Greedy and Quantum Evolutionary methods

DKP Grp.	Problem size = 500			Problem size = 5000			Problem size = 10,000		
	AGP	AQP	# EQEA better	AGP	AQP	# EQEA better	AGP	AQP	# EQEA better
1	48438	61018	7	483294	563412	8	984889	1158681	5
2	67582	69037	5	672893	688997	4	1363381	1395080	5
3	50076	61948	6	506102	620607	6	1015016	1246752	5
4	44036	44919	6	445676	451673	1	896304	906715	1
5	50792	62752	5	507447	621695	6	1021859	1251110	7
6	55633	57705	3	553570	584196	4	1127432	1177977	4
7	25255	31255	7	253710	310182	7	508955	622955	5
8	37893	43051	7	382858	430121	4	757420	856606	4
9	142162	153300	4	1437554	1541309	3	2897585	3106700	1

The Third column in each shows the number of problems (out of 9 tried for various capacities of the knapsack) in which EQEA gives better solution than Greedy method for that group. It is seen that the EQEA provides a considerable improvement in several instances.

6 Conclusions

An enhanced QEA is presented for Difficult knapsack problems. EQEA shows good performance even for large instances of the problems. EQEA can be used with advantage in any subset slection problems apart from the DKPs.

Acknowledgments

The authors are extremely grateful to the Revered Chairman, Advisory Committee on Education, Dayalbagh for constant guidance and encouragement.

References

[1] Martello, S., Toth, P.: A new algorithm for the 0-1 knapsack problem. Management Science 34, 633–644 (1988)

[2] Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. J. Wiley, Chichester (1990)

[3] Martello, S., Toth, P.: Upper bounds and algorithms for hard 0-1 knapsack problems. Operations Research 45, 768–778 (1997)
[4] der Heide, F.M.a.: A polynomial linear search algorithm for the n-dimensional knapsack problem. Journal of the ACM 31, 668–676 (1984)
[5] Pisinger, D.: An expanding-core algorithm for the exact 0-1 knapsack problem. European Journal of Operational Research 87, 175–187 (1995)
[6] Pisinger, D.: Where are the Hard Knapsack Problems, Technical Report, 2003-8, DIKU, University of Copenhagen, Denmark
[7] Martello, S., Pisinger, D., Toth, P.: Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science 45, 414–424 (1999)
[8] Han, K.H., Kim, J.H.: Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization. IEEE Transactions on Evolutionary Computation 6(6), 580–593 (2002)
[9] Han, K.H., Kim, J.H.: Quantum-Inspired Evolutionary Algorithms with a New Termination Criterion, Hε Gate, and Two-Phase Scheme. IEEE Transactions on Evolutionary Computation 8(2), 156–169 (2002)