Journal of Computational and Applied Mathematics 285 (2015) 86–99

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

Solving the 0–1 Quadratic Knapsack Problem with a competitive Quantum Inspired Evolutionary Algorithm

C. Patvardhan [a], Sulabh Bansal [a,*,1], A. Srivastav [b]

[a] Faculty of Engineering, Dayalbagh Educational Institute, Agra 282005, India

[b] Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany

ARTICLE INFO

Article history:
Received 16 November 2013
Received in revised form 25 November 2014

Keywords:
Combinatorial optimization
Quadratic Knapsack Problem
Quantum Inspired Evolutionary Algorithm

ABSTRACT

Quadratic Knapsack Problem (QKP) extends the canonical simple Knapsack Problem where the value obtained by selecting a subset of objects is a function dependent not only on the value corresponding to individual objects selected but also on their pair-wise selection. QKP is NP Hard in stronger sense i.e. no pseudo-polynomial time algorithm is known to exist which can solve QKP instances. QKP has been studied intensively due to its simple structure yet challenging difficulty and numerous applications. Quantum Inspired Evolutionary Algorithm (QIEA) belongs to the class of Evolutionary Algorithms and exhibits behaviour of an Estimation of Distribution Algorithm (EDA). QIEA provides a generic framework that has to be carefully tailored for a given problem to obtain an effective implementation. Thus, several forms of QIEA exist in the literature. These have been successfully applied on many hard problems. A new QIEA, QIEA-PSA is proposed with improved exploration and exploitation capabilities. Computational experiments on these benchmarks show that QIEA-PSA is improved significantly both in terms of the quality of solutions and speed of convergence on several benchmark QKP instances. The ideas incorporated are general enough and can be utilized with advantage on other similar and not so similar problems.

1. Introduction

The 0/1 Quadratic Knapsack Problem (QKP) is a generalization of the 0/1 Knapsack Problem (KP) introduced by Gallo et al. [1]. Given n items to be filled in a Knapsack where w_j is the positive integer weight of jth item, c is a positive integer Knapsack capacity and an $n \times n$ nonnegative integer matrix $P = (p_{ij})$ is given, where p_{jj} is a profit achieved if item j is selected, and, for $j > i$, $p_{ij} + p_{ji}$ is the additional profit achieved if both items i and j are selected. QKP is to find a subset of items whose total weight is not more than Knapsack capacity c such that the overall profit is maximized. x_j is a binary variable which is equal to 1 if jth item is selected and 0 otherwise, the problem is formulated as follows.

$$\begin{aligned} &\text{Maximize:} && \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}x_i x_j \\ &\text{Subject to:} && \sum_{j=1}^{n} w_j x_j \le c \\ &x_j \in \{0,1\} && j \in \{1,\ldots,n\}. \end{aligned} \tag{1}$$

* Corresponding author.
E-mail addresses: cpatvardhan@gmail.com (C. Patvardhan), sulabh.bansal.78@gmail.com (S. Bansal), asr@informatik.uni-kiel.de (A. Srivastav).
[1] Permanent Residential Address: 1/25, Hazuri Bhawan, Peepal Mandi, Agra 282003, India.

http://dx.doi.org/10.1016/j.cam.2015.02.016
0377-0427/

Without loss of generality matrix P is considered symmetric such that $p_{ij} = p_{ji}$ for all i and j. To simplify, another formulation of problem due to the above assumption is popular where additional profit achieved if both items i and j are selected is considered as p_{ij} rather than $p_{ij} + p_{ji}$, for $j > i$. Some researchers formulate the problem thus as follows.

$$\begin{aligned} &\text{Maximize:} && \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}x_i x_j \\ &\text{Subject to:} && \sum_{j=1}^{n} w_j x_j \le c \\ &x_j \in \{0, 1\}, && j \in \{1, \ldots, n\}. \end{aligned} \tag{2}$$

The KP is a particular case of QKP which arises when $p_{ij} = 0$ for all $i \neq j$. Clique problem, is also a particular case of QKP, which calls for checking whether a given undirected graph $G = (V, E)$, contains a complete subgraph on k nodes for a given integer k. The popular optimization version of Clique, called Max Clique, calls for an induced complete subgraph with a maximum number of nodes. The Max Clique, can be solved using a QKP algorithm by using binary search. The Max Clique is not only NP-hard in strong sense but is one of the hardest combinatorial optimization problems. Same properties apply to QKP as well. Pseudo polynomial time algorithms exist for KP but no such algorithm exists for QKP. QKP is thus considered much more difficult than the simple KP [2,3].

QKP is thus a challenging problem. It has yet been studied widely due to its generality and wide applicability in several areas like facility location problems [4,5], compiler design [6], finance [7], VLSI design [8] and weighted maximum b-clique problem [9,10].

Gallo et al. [1] introduced QKP and presented a method to derive the upper bound using upper planes. Several attempts have been made to solve QKP [3]. The exact algorithms suffer from high time complexity.

Several studies on heuristic and meta-heuristic methods also exist which provide satisfying solutions for QKP within reasonable time. Hammer and Rader presented the linearization and exchange (LEX) heuristic [11] which is able to compute very close to optimal (profit more than 99% of optimal) in less than 1 s for instances up to 100 binary variables. Julstrom [12] presented studies on various greedy and genetic algorithms and has shown a greedy genetic algorithm (GGA) to solve a sample of Billionet and Soutif's benchmark instances [13] (BS benchmark instances) with up to 200 variables to optimality in 902 out of 1000 trials using around 15000 Function Evaluations (FES) on an average per trial. A Mini-Swarm algorithm is proposed by Xie and Liu [14] and an artificial bee colony (ABC) algorithm by Pulikanti and Singh [15] which are shown to solve all BS benchmark instances with up to 200 binary variables to optimality with high probability (0.949 and 0.987 respectively) in a very reasonable time. The ABC algorithm has been shown to outperform Mini-Swarm in [15]. Recently Azad et al. presented the solution of QKP using an artificial fish swarm algorithm (AFSA) [16,17]. In [16] AFSA is compared on the basis of FES with MINLP solver for the results on 40 BS Benchmark instances of size 100 variables. In [17] a comparison of simplified AFSA is provided with GGA [12] and shown to solve BS Benchmark instances of size 200 with higher probability. However, the results of ABC [15] are better both in terms of computation time and probability to find optimum.

Patvardhan et al. [18] presented an improved QIEA for QKP (dubbed QIEA-PPA in this paper). They compared their results with Naïve EA and GGA [12] on BS benchmark instances and reported better performance in terms of number of optimal solution hits.

Hammer and Rader [11] presented the solution quality in terms of error from optimal observed and computational time required by the heuristic. Julstrom [12] presented the quality of solution in terms of error from optimal for heuristics and hits made to optimal within 50 runs for GGA. The computational effort is reported as time in seconds for heuristics while for GGA both the generations required to reach optimal and time in seconds to reach optimal are reported. Xie and Liu [14] reported number of hits made to optimal in 100 runs and average execution time per run in seconds. Pulikanti and Singh [15] reported number of hits made to optimal and average value obtained in 100 runs for each problem for quality of solution and minimum and average time to reach optimal. Azad et al. in [16] presented average FES and average time required in seconds per run out of 30 runs for computational effort while number of successful runs and average function value for quality of solution. In [17] the average time to reach optimal in 50 runs is reported for computational effort while number of successful runs and average function value for quality of solution. Patvardhan et al. [18] reported average iterations required to optimal, number of hits made to optimal and average function value in 50 runs for each problem.

In this paper, an attempt has been made to design a QIEA dubbed QIEA-PSA (QIEA-Patvardhan–Sulabh–Anand) for the Quadratic Knapsack Problem (QKP). QIEA-PSA successfully balances the conflicting objectives of providing near-optimal solutions while keeping the computation cost low. On several benchmark QKP problems, QIEA-PSA delivers performance which is substantially better than EAs, other QIEAs, ABC and AFSA implementations. The results in this paper are presented for 100 BS benchmark instances of sizes 100, 200 and 300 binary variables.

A comparison with GGA [12], QIEA-PPA [18] and AFSA [16] shows that proposed QIEA-PSA outperforms them in terms of both the number of hits made to optimal and number of function evaluations required to reach to optimal. A comparison with ABC algorithms on BS benchmark instances for sizes up to 200 variables shows that QIEA-PSA outperforms ABC and mini swarm both in terms of probability to obtain optimal and average time to compute optimal. This paper presents a comparison with best of the results reported at different places in the literature on the basis of different parameters for QKP. A comparison is presented with results of ABC [15] in terms of time to reach optimal, number of hits made to optimal

and standard deviation observed in value when 100 runs were performed for all 80 problems of size 100 and 200 variables. A comparison is presented with results of GGA [12] and QIEA-PPA [18] in terms of average FES required and number of hits made to optimal when 50 runs were performed for selected 20 problems of size 100 and 200 variables. A comparison is presented with results of AFSA in [16] in terms of average FES required, number of hits made to optimal and average computational time taken in 30 runs for selected 20 problems of size 100 variables.

The rest of the paper is organized as follows. The QIEA framework used here is explained in Section 2. The proposed QIEA-PSA is presented in Section 3 along with an explanation of the various features incorporated in QIEA-PSA and their motivation. Computational performance of QIEA-PSA is presented in Section 4. Conclusions are presented in Section 5.

2. Quantum Inspired Evolutionary Algorithm (QIEA)

The QIEA introduced in [19] belongs to a class of population-based stochastic evolutionary algorithms. It uses the qubit, a vector, to represent the probabilistic state of individual. Each qubit is represented as $q_i = \begin{bmatrix}\alpha_i \\ \beta_i\end{bmatrix}$, α_i, β_i are complex numbers so that $|\alpha_i|^2$ is the probability of state being 1 and $|\beta_i|^2$ is the probability of state being 0 such that $|\alpha_i|^2 + |\beta_i|^2 = 1$. For the purpose of QIEAs, α_i and β_i are assumed to be real. Thus, a qubit string with n bits represents a superposition of 2^n binary states and provides an extremely compact representation of entire space.

The process of generating binary strings from the qubit string, Q, is known as observation. To observe the qubit string Q, a string consisting of the 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 the square of Q_i and 1 otherwise. 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 the best solution, in subsequent iterations. A quantum gate is utilized for this purpose so that qubits retain their properties [19].

One such gate, used by the QIEAs presented in this work, is the Rotation Gate, which updates the qubits as follows:

$$\begin{bmatrix}\alpha_i^{t+1} \\ \beta_i^{t+1}\end{bmatrix} = \begin{bmatrix}\cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i)\end{bmatrix} \begin{bmatrix}\alpha_i^{t} \\ \beta_i^{t}\end{bmatrix} \tag{3}$$

where, α_i^{t+1} and β_i^{t+1} denote probabilities for ith qubit in $(t+1)$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 QIEA. Observing a qubit string 'n' times yields 'n' different solutions because of the probabilities involved. The 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. This sequence of steps continues; these ideas can be easily generalized to work with multiple qubit strings. A simple QIEA implementation used in the present work is given in Fig. 1.

Notations-

$Q(t)$: Qubit population in tth iteration.

$P(t)$: population of binary solutions in tth iteration.

$B(t)$: population of best solutions found till tth iteration.

q_j^t: jth individual in $Q(t)$.

p_j^t: jth individual in $P(t)$.

b_j^t: jth individual in $B(t)$.

b: best solution observed so far.

MaxIterations: maximum number of iteration set as termination criterion by the user.

n: Number of items to be considered in problem.

t: the current iteration.

- Qubits in $Q(t)$ are initialized with $1/\sqrt{2}$ and best binary solution b with zeros (lines 1 and 2). Before starting iterations individuals of $Q(t)$ are observed and the solutions are repaired to make them feasible (lines 3 and 4). $B(t)$ and $P(t)$ are initialized by these feasible solutions (line 5). The QIEA iterates MaxIterations times through the tasks described in lines 6–21. The following tasks are repeated η_1 times first (lines 10–19).
  - The qubits in $Q(t)$ are collapsed and repaired η_2 times to form feasible solutions in $P(s)$ and the corresponding best solutions are retained in $P(t)$ (lines 11–15).
  - Individual at every position in $B(t)$ is replaced by corresponding individual in $P(t)$ if found better (line 16).
  - The best solution from among b and $B(t)$ is moved into b (line 17).
  - **Local Update**: Each individual in $Q(t)$ is updated based on the corresponding best solution in $B(t)$ (line 18).
- Each iteration restarts after updating the individuals in $Q(t)$ globally (line 20) based on the best solution observed so far.

```
    Procedure QIEA
 1  t ← 0;
 2  Initialize (Q(t)); Initialize b;
 3  Make P(t) from Q(t);
 4  Repair P(t);
 5  copy P(t) to B(t);
 6  while ( t<MaxIterations)
 7  {
 8     t ← t+1;
 9     for r from 0 to η1 do{
10           for s from 0 to η2 do{
11             Make P(s) from Q(t);
12             Repair P(s);
13             Evaluate P(s);
14             for each j ∈ {1, ... , n}if ( p_j^s better than p_j^t ) p_j^t ← p_j^s;
15           } //*for s*//
16           for each j ∈ {1, ... , n} if ( p_j^t is better than b_j^t) b_j^t← p_j^t;
17           for each j ∈ {1, ... , n}if (b_j^t is better than b) { b← b_j^t ; bqbit←q_j^t;}
18           for each j ∈ {1, ... , n} Update q_j^t based on b_j^t;
19     } //*for r*//
20  for each j ∈ {1, ... , n}Update q_j^t based on b;
21  } //*while*//
```

Fig. 1. Pseudo-code for QIEA framework used in this paper.

QIEA was shown to be competitive for search and optimization problems. Subsequently QIEAs were shown to be a form of Estimation of Distribution Algorithms (EDAs) that implicitly create the probability distributions that have higher probability of generating better solutions on the basis of past experience [20,21]. There has been greater recent interest in the QIEAs as the fundamental basis of their working is well understood and documented. A host of QIEA-based attempts have been reported in the literature that utilize QIEAs for the solution of a wide variety of problems [21].

The main strengths of the QIEAs are as follows.

(i) QIEAs have better representation power using qubits to enable use of smaller populations (ideally even a size of 1) [22,23]. Smaller populations require lesser computation during search.
(ii) QIEAs have an EDA style functioning with implicit determination of distributions leading to better solutions [20,21].
(iii) QIEAs provide an extremely flexible framework that can be adapted for the solution of both real-parameter function optimization problems as well as combinatorial optimization problems [21,23]. This makes them very versatile in their applicability.
(iv) The QIEA framework also provides the flexibility necessary for the inclusion of features appropriate for a given problem towards delivering better search performance [21].
(v) QIEA inherently favours exploration of the search space initially gradually shifting towards exploitation as the search progresses which is a desirable aspect [23].
(vi) There is a possibility of utilizing one of the several termination criteria appropriate for the problem at hand [24].

QIEAs, of course, are not a "one-size fits all" solution. The No Free lunch Theorem prohibits that. The conspicuous limitations of QIEAs are as follows. Many of these are shared with other EAs as well.

(i) Slow convergence may result from use of small qubit rotations. Use of large qubit rotations may cause the algorithm to miss a good solution completely.
(ii) Inclusion of features promoting faster convergence may cause the algorithm to get stuck in local optima.
(iii) Slow convergence limits the problem sizes that can be tackled using QIEAs.
(iv) Implementation of QIEAs, just as other EAs, is more an art to enable balance of computational effort devoted to exploration and exploitation that is required for good search performance.

Thus, although it is well known that QIEAs are competitive for a large variety of problems, a particular QIEA has to be designed for the problem at hand to achieve high performance that is competitive with respect to the state-of-art algorithms for the problem. QIEA-PSA is an attempt towards this end for QKP.

3. QIEA-PSA

As stated above, QIEA provides broad framework with scope for enhancements for rapid solution of a specific problem. Here, the modified algorithm designed for QKP is named as QIEA-PSA. The steps taken to improve QIEA are as follows.

(i) Initializing the Best Solution "*b*" with a good solution obtained using a heuristic.
(ii) Using the Sort Orders of items as domain knowledge in the following steps of QIEA
  (a) Initializing the Qubit Individuals to depict the better estimations of distribution models.
  (b) Fortifying the repair function to improve quality of solutions.

```
    Function SolveGreedy()  /* returns GreedySolution*/
1   GS ← {1,...,n}, w←∑_{i=0}^{n} w_i;
2   forever  do {
3       Let i | min_{i ∈ GS}(RPD_i^{GS});
4       if (w > C) { GS ← GS / i; w ← w- w_i;}
5       else { ImproveLocal(GS); return (GS);}
6   } //* forever *//
```

Fig. 2. Pseudo-code for SolveGreedy.

```
    Function SortGreedy() /*returns GreedyOrder*/
1   PS ← Φ;j←1;
2   Let i | max_{0≤i≤n}( p_ii/w_i);
3   GO[j] ← i;
4   PS ← PS ∪ i
5   for j from 2 to n{
6       Let i | max_{i ∉ PS} RPD_i^{PS}; GO[j] ← i; PS ← PS ∪ i;
7   }//* for j *//
8   return(GO);
```

Fig. 3. Pseudo-code for SortGreedy.

(iii) Improving the local best solutions using local search.
(iv) Corrective step when solutions appear to be stuck in a local optimum.
(v) Re-initialization of Qubit individuals.
(vi) Replacing the non-performing qubit individuals by best.
(vii) Exploit locally before exploring globally.

The details are explained below.

3.1. Initializing best solution "b" with a good solution obtained using a heuristic

The "update" procedure modifies qubit individuals during evolution so that they represent a probability distribution model which leads to generation of solutions closer to best solutions obtained so far. Initializing the best solution with a better solution (rather than any random solution) ensures that the better distribution models are found earlier. The greedy solution is a fairly good solution and the QIEA initializes the best solution with it. A greedy solution can be formed for QKP relatively quickly using a simple constructive heuristic. Starting with an infeasible solution which has all the items included in the Knapsack the greedy algorithm removes items iteratively till the solution becomes feasible. Each time an item having minimum relative profit density is removed from the solution. The solution thus generated is improved using the local search procedure described in Section 3.3. Let a set P represent a partial solution such that $i \in P$ iff ith item is included in the Knapsack. The relative profit density (RPD_i^P), of an item i with respect to partial solution P, is computed as $(p_{ii} + \sum_{j \in P/\{i\}} p_{ij})/w_i$. Let GreedySolution be a set representing the solution and GS be the set used in function as partial solution. The greedy heuristic used in QIEA-PSA is described in Fig. 2.

3.2. Using sort orders as domain knowledge

Orderings of the elements named GreedyOrders are computed which provide knowledge about their priority for inclusion in the Knapsack. These are used to initialize the qubit individuals (Section 3.2.1) and also while repairing infeasible solutions (Section 3.2.2).

Starting with an empty solution suppose elements are added iteratively without any constraint on capacity, such that during each iteration an item, having maximum RPD with respect to the partial solution formed so far, is added. The order of items in which they are added to the solution in such a way is the sort order named as GreedyOrder. The process is depicted in Fig. 3. The item having the maximum diagonal profit can be selected as the first item with the rest being selected as stated. The GreedyOrder so generated may not provide the optimal sequence of items to be selected. Different choice of the first item in step 3 leads to a different order. Thus, multiple orders are generated by taking different items as the first item instead of item 1 of output of process shown in Fig. 3.

3.2.1. Initializing the qubit individuals to depict the better estimations of distribution models

The qubit individuals are initialized based on the GreedyOrders as shown in Fig. 4. The list of sorted items is seen as divided in 3-parts based on order of their preferences; first part contains items having high preference for selection in Knapsack, second part contains items having medium preference and third contains the items having low preference. Hence,

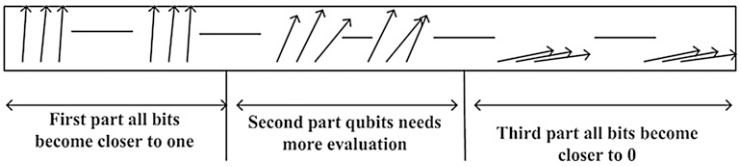

Fig. 4. Initialization of qubits. Qubits for items shown are sorted in a GreedyOrder from left to right.

```
    Procedure RepairGreedy (x, GreedyOrder)
1     knapsack-overfilled ← false;
2     weight ← ∑_{j=1}^{n} w_j x_j;
3     if (weight >  C)  knapsack-overfilled ← true;
4     while (knapsack-overfilled) {
5       for i from GreedyOrder[n] to GreedyOrder[1]
6         { if (x_i=1) {x_i ← 0; weight← weight – w_i;} }
7       if (weight ≤ C) knapsack-overfilled ← false;
8     }//*while*//
9     for j from GreedyOrder[1] to GreedyOrder[n] {
10      if ((x_j=0) and (weight+w_j ≤ C)){ x_j ←1; weight← weight +w_j;}
11    } //*for j*//
```

Fig. 5. Pseudo-code for RepairGreedy.

qubits for items lying in first part (third part) are assigned values closer to 1 (0) so that they have high (low) probability of collapsing to value 1. Qubits for items lying in the second part are the ones which primarily require more search for convergence to either 0 or 1, hence medium values between 0 and 1 are assigned to them. As a result, QIEA-PSA starts exploiting the favourable area in solution space represented by such initialized qubit individuals. Initializing qubits based on a single GreedyOrder would limit the exploitation to a single area thus multiple GreedyOrders are used in round-robin fashion to initialize all the qubit individuals.

3.2.2. Fortifying the repair function to improve quality of solutions

QIEA uses a simple "repair" function after it observes the qubits through the "make" procedure to make the observed solution feasible. In QIEA-PSA, the repair function is fortified to improve the quality of solutions while making them feasible. This improves the speed of convergence. A GreedyOrder, as explained above, determines the relative change in profit when an item is included or excluded from the Knapsack. Accordingly, in each repair step, items closest to the end of a GreedyOrder are removed and items closest to the beginning are added as necessary. The pseudo-code is given in Fig. 5.

If the same order is used each time in repairing all the infeasible solutions, the results would be almost alike every time. Thus, multiple orders are used in repair function in a round-robin manner. Such a policy helps in maintaining diversity in the solutions generated by the repair function.

3.3. Improving the local best solutions

The local best solutions are further improved using a local "improve" function also referred in Section 3.1. It tries to explore the quality of all solutions in the vicinity of a local best solution and selects the best. It iteratively executes passes as long as gain in profit is observed. The *i*th element (if not in solution) is either **included** or **replaced** by an item *j* already in solution after each pass. The action of inclusion (or replacement) is performed for an *i* (or pair *i* and *j*) which results in maximum gain in overall profit of the solution. In other words all actions (inclusion and replacement) pertaining to combinations of *i* and *j* are first observed and only the change which gives best gain in profit is made effective in a pass. Let *P* be a feasible solution, the pseudo-code of the procedure used to improve profit of *P* is given in Fig. 6.

The function defined above is very expensive in terms of computational effort it requires and moreover it tries to convert a probably worse solution to the nearest best solution and thus increases the chance of generating the same solution many times. So a lighter version, **RandImproveLocal**, is also implemented where the forever loop in step 1 of ImproveLocal is executed a fixed number of times instead, each time randomly picking an element from a list of not included items in step 3. So to reduce the computational effort and moreover the duplication of effort this lighter version is performed on half of the individuals.

3.4. Mutation of solutions as a corrective step taken when they appear to be stuck in local optimum

EAs suffer from tendency of getting stuck in local optima. All the modifications described above help the algorithm to exploit the search space around the greedy solutions increasing the speed of convergence but also the tendency to get stuck

```
    Procedure ImproveLocal(P)
 1  forever do{
 2      bg←0;m_i=-1;m_j=-1
 3      for ∀i ∉ P {
 4        if (w_k ≤ C − ∑_{k∈P} w_k) {
 5            g = p_ii + ∑_{k∈P}(p_k p_i);
 6           if (g > bg) { bg ← g;m_i=i;}
 7        }// if*//
 8        else{
 9           for ∀j ∈ P do
10              g = p_ii − p_jj + ∑_{k∈P/j}(p_k p_i − p_k p_j);
11              if (g > bg){ bg ← g; m_i=i;m_j=j;}
12           }//*for j*//
13        }//*else*//
14      }//*for i*//
15      if (bg = 0) exit;
16      P ← P ∪ m_i; if(m_j≠-1) P ← P/ m_j;
17  } //* forever *//
```

Fig. 6. Pseudo-code for ImproveLocal.

```
    Procedure QIEA-PSA
 1  Initialize(Nsortorder, Wlambda)
 2  SolveGreedy(GSolution), MultipleSortGreedy(GreedyOrders, Nsortorder)
 3  t ← 0, C_orginal ← C, b ← GSolution, we← GSolution, be← 0
 4  InitializeGreedy ( q_j^t, GreedyOrders) for each j ∈ {1, ..., n}
 5  InitializeGreedy ( bqbit, GreedyOrders)
 6  Make P(t) from Q(t)
 7  RepairGreedy (P(t),GreedyOrders)
 8  copy P(t) to B(t)
 9  while ( t<MaxIterations) {
10     for r from 0 to η_1 do
11        for s from 0 to η_2 do
12           Make P(s) from Q(t)
13           RepairGreedy (P(s),GreedyOrders)
14           for each j ∈ {1, ..., n} if (HamDistance( p_j^s,b)<2) Mutate( p_j^s)
15           Evaluate P(s)
16           for each j ∈ {1, ..., n} if ( p_j^s better than p_j^t ) then p_j^t ← p_j^s
17        } //*for s*//
18        for j ∈ {1, ..., n/2} ImproveLocal p_j^t ;for j ∈ {n/2, ..., n} RandImproveLocal p_j^t
19        for each j ∈ {1, ..., n} if ( p_j^t is better than b_j^t) then b_j^t← p_j^t
20        for each j ∈ {1, ..., n} if (b_j^t is better than b) then b← b_j^t , bqbit ← q_j^t
21        for each j ∈ {1, ..., n} Update q_j^t based on b_j^t
22        StochasticPurge (B(t),be,we, Q(t),bqbit)
23     } //*for r*//
24     for each j ∈ {1, ..., n} Update q_j^t based on b
25  } // *while*//
```

Fig. 7. Pseudo-code for QIEA-PSA.

in local optima. To combat this problem, if a new solution generated is seen to be close to the global best solution found so far, it is mutated. During mutation 2–3 bits in the solution vector are randomly selected and changed to 0. Elements with better RPD are then iteratively included in solution as long as the solution remains feasible. To check closeness of two solutions, Hamming distance between them is calculated. Such an operator improves diversity without increasing the computational effort. It helps to explore the solution space around a current solution so that local optimal in vicinity is not missed. This improves the chances of finding optimal in case it is in the vicinity of the converging solution.

3.5. Re-initialization of qubit individuals

It may happen even after applying mutation as explained in Section 3.4 that all the solutions generated from a qubit individual are still same after a sequence of generations. It clearly indicates that such a qubit individual has converged and further observations made on it will not yield any better solutions. So instead of wasting cycles on such a qubit string, its re-initialization is done to restore diversity. But, reinitializing it again in the same way as described in Section 3.2.1 may not make any difference. Thus, each qubit in individuals which generates the same solution for more than 3 times out of 5 is

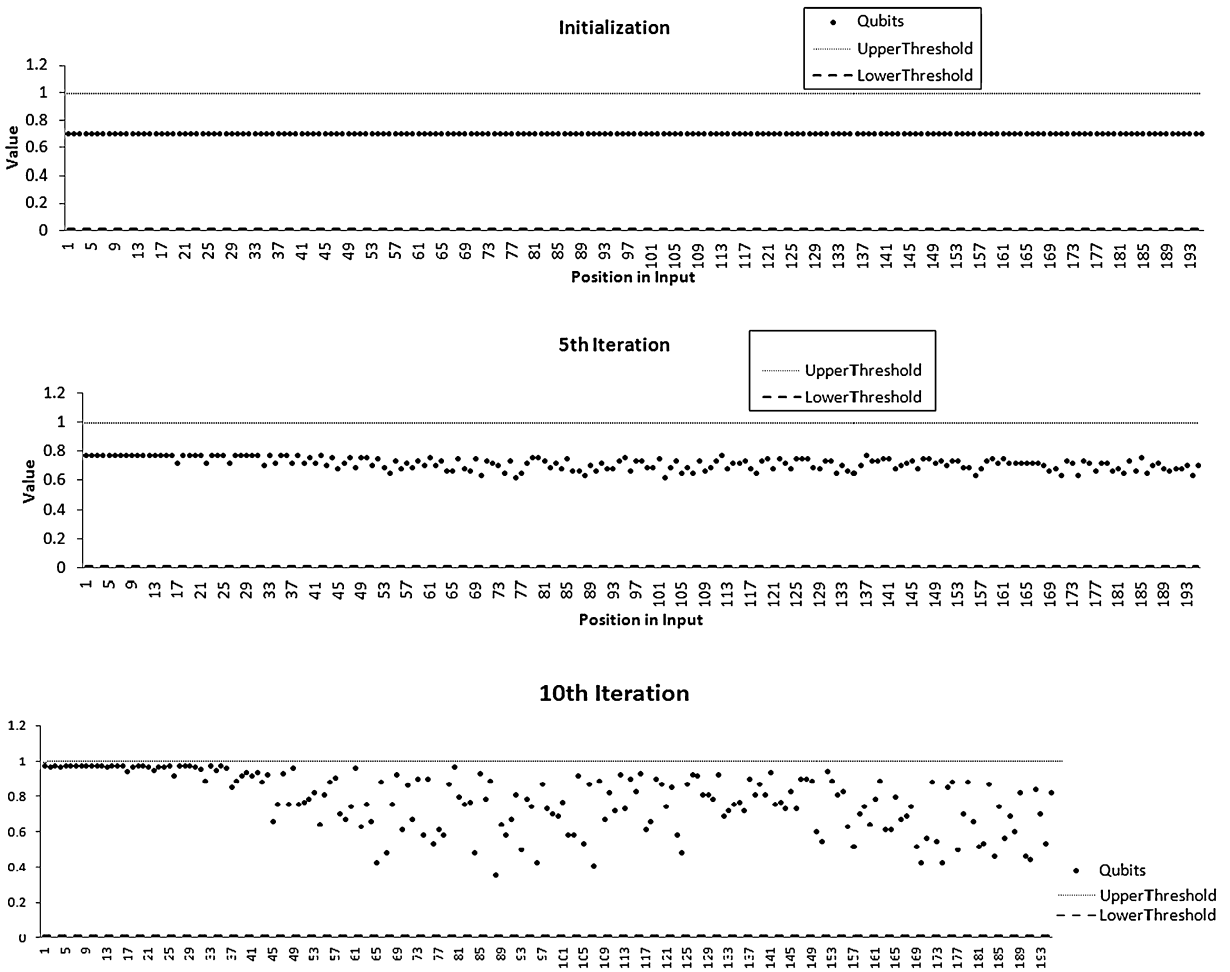

Fig. 8. Status of qubits during evolution in QIEA.

set to $1/\sqrt{2}$. It increases the diversity of solutions explored through qubit individuals using same computational effort and thus improves chances to find the optimal.

3.6. Replacing the non-performing qubit individuals by best (StochasticPurge)

The qubit individuals are assigned a lifetime to perform. If a qubit individual is found to perform less than average after that, there remains 50% chance for it to survive. In case it ceases to exist it is replaced by the best performer found so far.

3.7. Exploit locally before exploring globally

QIEA has the property that it updates the qubits over the time period such that they represent better estimation of distribution models. The sort orders as described in Section 3.2 are used to initialize the qubit individuals in order to improve the exploitation in favourable area of solution space. On the same lines basic steps of QIEA are used to exploit these areas intensively before starting the exploration globally in QIEA-PSA. The resulting qubit individuals favour the smaller solution area closer to better solutions. Thus, basic steps of QIEA are used to further update half of the total qubits individuals initialized according to step 3.2 before feeding them to actual QIEA-PSA. Basic steps of QIEA-PSA listed as follows are performed on half of the qubit individuals for small number of times (empirically set as 15 for this paper) before starting QIEA-PSA on the entire population.

- Make
- RepairGreedy
- ImproveLocal
- Update.

Since adding this feature increases the computational time required for initiation of the algorithm we report its results separately calling it as QIEA-PSA2 while the version of algorithm which does not use this feature is named as QIEA-PSA1.

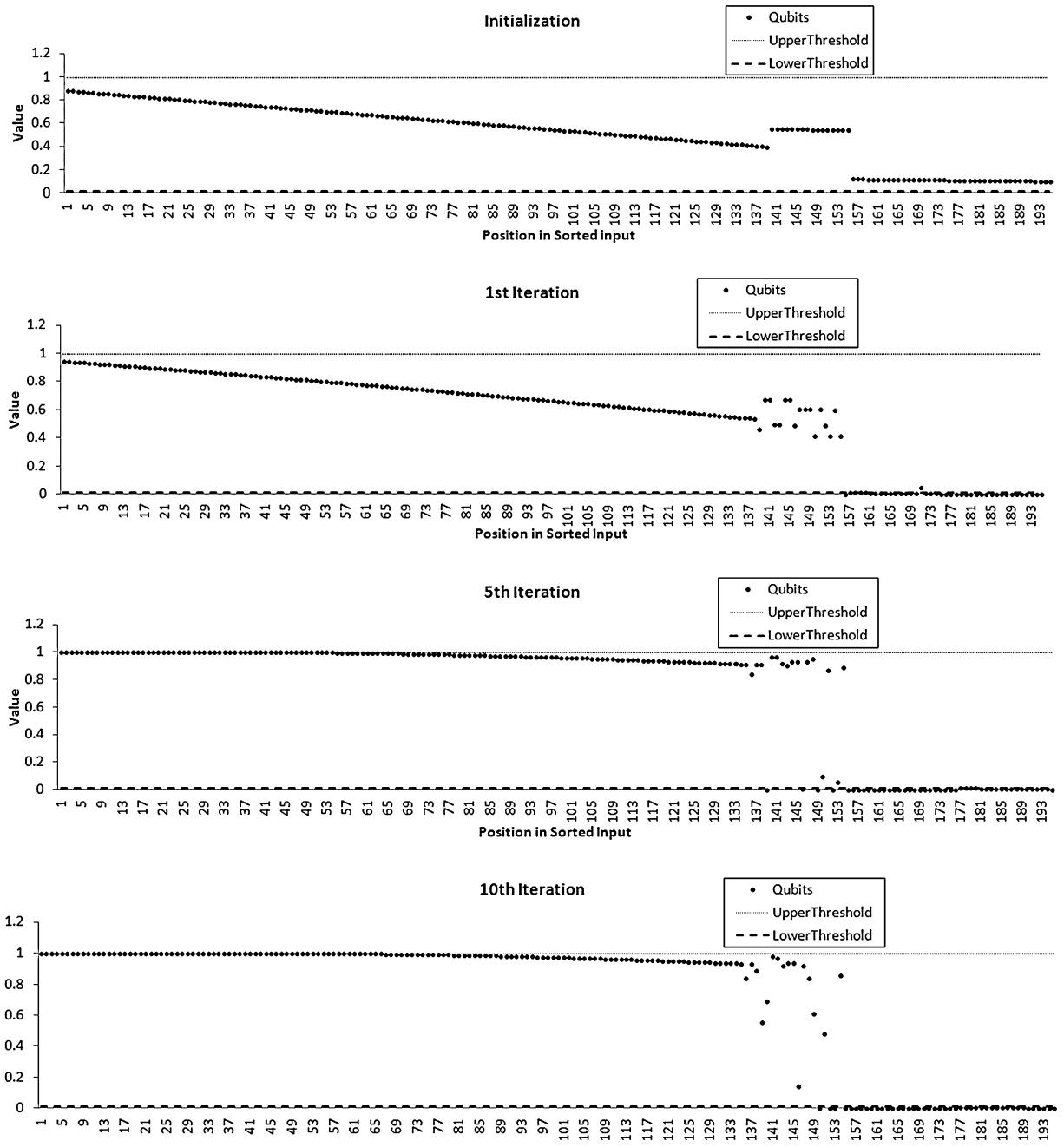

Fig. 9. Status of qubits during evolution in QIEA-PSA.

The complete pseudo-code for QIEA-PSA is presented in Fig. 7. As in pseudo-code for QIEA, $Q(t)$ is the qubit population after tth iteration, $P(t)$ is the population of individual solutions, $B(t)$ is the set of best solutions found so far corresponding to each individual, C is the capacity of the Knapsack. Individuals in $Q(t)$, $P(t)$ and $B(t)$ are referred by q_j^t, p_j^t and b_j^t respectively for each $j \in n$; b refers to the global best solution found so far and bqbit refers to the qubit individual which generated b. In the following a brief description for the procedures called in QIEA-PSA is mentioned for which complete pseudo-code is not provided.

MultipleSortGreedy (GreedyOrders, Nsortorder): Executes Nsortorders times the function SortGreedy(), to obtain that many sort orders in GreedyOrders, with different choice of first element. Here the first sort order is generated by executing the function exactly as coded in SortGreedy(), while nth sort order, for $n > 1$, is generated with the first element chosen as nth element in 1st sort order.

Table 1
Comparison of GGA, QIEA-PPA and QIEA-PSA on BS Benchmark Instances.

Problem	Opt	GGA			QIEA-PPA			QIEA-PSA1			QIEA-PSA2		
		Hits	Min FES	AVG FES	Hits	Min FES	AVG FES	Hits	Min FES	AVG FES	Hits	Min FES	AVG FES
100_25_1	18558	50	375	2269	50	4250	182716	50	20	3177.94	50	1210	2537.38
100_25_2	56525	50	75	187	50	50	50	50	1	1	50	1	1
100_25_3	3752	36	625	9203	46	4850	289376	50	82	90.62	50	2	7.34
100_25_4	50382	23	175	4849	50	50	50	50	5	5860.12	50	1212	28536.04
100_25_5	61494	50	25	54	50	50	50	50	1	1	50	1	1
100_25_6	36360	50	375	1330	50	50	50	50	1	1	50	1	1
100_25_7	14657	50	275	486	50	50	33228	50	22	194.88	50	2	17.84
100_25_8	20452	50	275	411	50	50	50	50	81	1095.44	50	2	179.98
100_25_9	35438	37	225	11777.5	50	150	25434	50	35	118.26	50	2	18.18
100_25_10	24930	50	325	582	38	6050	192471	50	1	1	50	1	1
200_100_1	937149	50	1550	46990	50	1100	54816	50	82	84.88	50	2	6.04
200_100_2	303058	50	2950	10362	49	1500	207634	50	148	1010.22	50	11	97.52
200_100_3	29367	50	1250	1940	50	3700	153232	50	25	97.74	50	2	8.78
200_100_4	100838	50	1250	2082	50	100	100	50	1	1	50	1	1
200_100_5	786635	50	1650	4928	41	1900	267374	50	82	89.18	50	2	8.08
200_100_6	41171	50	1150	1392	50	100	100	50	1	1	50	1	1
200_100_7	701094	50	1850	19666	40	700	90540	50	82	84.24	50	2	4.08
200_100_8	782443	6	107950	157100	41	5100	567602	49	82	5613.1224	50	2	34.72
200_100_9	628992	50	1550	5716	42	2500	523990	50	82	91.08	50	2	8.2
200_100_10	378442	50	3950	17932	47	7700	221086	50	1	1	50	1	1
Sum		**902**			**944**			**999**			**1000**		

InitializeGreedy (q_j^t, GO): Where q_j^t is jth qubit individual in a population $Q(t)$ and GO is a 2-D array containing 'n_s' sort orders in total such that GO[i] refers to ith sort order. The procedure initializes the qubit individual q_j^t as explained in Section 3.1 using the sort order number GO[j mod n_s].

Make $P(t)$ from $Q(t)$: The procedure collapses the qubit individuals in $Q(t)$ observing solution individuals in $P(t)$.

HamDistance (p_j^s, b): Returns hamming distance between two binary strings.

Mutate (p_j^s): Mutates the solution p_j^s as described in Section 3.4.

Update q_j^t based on b_j^t: Rotates the qubits q_j^t towards bits in b_j^t as explained earlier in Section 2 and defined in [19] using the rotation angle as 0.01.

StochasticPurge ($B(t)$, be, we, $Q(t)$, bqbit): 'be' and 'we' refer to the best and worst values evolved so far in $B(t)$. Replaces the qubit individual q_j^t by bqbit with probability 0.5 if value of b_j^t is below (be + we)/2.

The maximum number of iterations in the algorithm is controlled using a global constant, MaxIterations.

4. Results and discussion

The experiments are done on Intel® Xeon® Processor E5645 having specifications as 12M Cache, 2.40 GHz, 5.86 GT/s Intel® QPI. The machine uses Red Hat Linux Enterprise 6 as operating system.

On the basis of empirical investigation η_1 and η_2 are set to 5 and population size is set to 160. Most of the problems are solved to optimality in almost all 100 runs within 60 iterations so MaxIterations is taken as 60 for the purpose of this presentation. Problems are named as *n_d_i* which specifies parameters size_density_seed of an instance.

Behaviour of qubit convergence is observed in modified QIEA without including the features mentioned after Section 3.3. Plots are drawn to elicit the status of qubit in bqbit as observed after initialization, 5th iteration and 10th iteration of QIEA in Fig. 8 for instance 200_75_1 in [13]. The qubit position (numbers ranging from 1 to 200 representing the corresponding item in the input individual) is depicted on the x-axis whereas the qubit value is on the y-axis. Similar plots are drawn for QIEA-PSA in Fig. 9 where the qubit number (numbers ranging from 1 to 200 representing the corresponding item in the sorted in GreedyOrder) is depicted on the x-axis whereas the qubit value is on the y-axis.

Following points are observed from the results presented.

(i) Till the 10th iteration most of the qubits in QIEA are between 0 and 1 and thus far from convergence while most of the qubits in QIEA-PSA get settled to either value 0 or 1 within 10 iterations.
(ii) The qubits in a sorted input of QIEA-PSA are divided into three parts which are initialized by the values in decreasing order as is clear from the graph.
(iii) During evolution in the first (last) part the qubits initialized close to 1 (close to 0) move further towards 1 (0).
(iv) Only the middle part of qubits goes through a considerable turbulence i.e. qubits closer to 1 move towards 0 and vice versa.

Table 2
Comparison of ABC and QIEA-PSA on BS Benchmark Instances of size 100 binary variables.

Problem	Opt	ABC				QIEA-PSA1				QIEA-PSA2			
		Hits	SD	MinT	AvgT	Hits	SD	MinT	AvgT	Hits	SD	MinT	AvgT
100_25_1	18558	100	0	0.02	0.039	100	0	0.015	0.180	100	0	0.572	0.690
100_25_2	56525	100	0	0	0.017	100	0	0.001	0.001	100	0	0.001	0.001
100_25_3	3752	100	0	0	0.011	100	0	0.014	0.020	100	0	0.014	0.015
100_25_4	50382	100	0	0.01	0.018	100	0	0.012	0.130	100	0	0.161	0.669
100_25_5	61494	100	0	0	0.008	100	0	0.001	0.001	100	0	0.001	0.001
100_25_6	36360	100	0	0.02	0.024	100	0	0.001	0.001	100	0	0.001	0.001
100_25_7	14657	100	0	0.01	0.019	100	0	0.014	0.051	100	0	0.014	0.022
100_25_8	20452	100	0	0.02	0.023	100	0	0.015	0.096	100	0	0.014	0.092
100_25_9	35438	100	0	0.02	0.04	100	0	0.014	0.021	100	0	0.013	0.016
100_25_10	24930	100	0	0.02	0.037	100	0	0.002	0.002	100	0	0.002	0.002
100_50_1	83742	100	0	0.01	0.032	100	0	0.012	0.017	100	0	0.013	0.013
100_50_2	104856	100	0	0	0.02	100	0	0.016	0.017	100	0	0.013	0.014
100_50_3	34006	100	0	0.01	0.023	100	0	0.019	0.114	100	0	0.016	0.047
100_50_4	105996	100	0	0	0.012	100	0	0.001	0.001	100	0	0.001	0.001
100_50_5	56464	100	0	0.02	0.031	100	0	0.001	0.001	100	0	0.001	0.001
100_50_6	16083	100	0	0.01	0.022	100	0	0.002	0.002	100	0	0.002	0.002
100_50_7	52819	100	0	0.01	0.028	100	0	0.001	0.001	100	0	0.001	0.001
100_50_8	54246	100	0	0.01	0.027	100	0	0.017	0.032	100	0	0.013	0.017
100_50_9	68974	100	0	0.01	0.025	100	0	0.001	0.001	100	0	0.001	0.001
100_50_10	88634	100	0	0.01	0.036	100	0	0.015	0.019	100	0	0.013	0.015
100_75_1	189137	100	0	0	0.002	100	0	0.001	0.001	100	0	0.001	0.001
100_75_2	95074	100	0	0.02	0.075	100	0	0.001	0.001	100	0	0.001	0.001
100_75_3	62098	100	0	0.02	0.025	100	0	0.002	0.002	100	0	0.002	0.002
100_75_4	72245	100	0	0.02	0.072	100	0	0.013	0.025	100	0	0.013	0.016
100_75_5	27616	100	0	0.01	0.018	100	0	0.065	0.087	100	0	0.547	0.576
100_75_6	145273	100	0	0.01	0.016	100	0	0.001	0.001	100	0	0.001	0.001
100_75_7	110979	100	0	0.01	0.029	100	0	0.013	0.019	100	0	0.013	0.014
100_75_8	19570	100	0	0.01	0.016	100	0	0.002	0.002	100	0	0.002	0.002
100_75_9	104341	100	0	0.02	0.044	100	0	0.014	0.026	100	0	0.013	0.020
100_75_10	143740	100	0	0.01	0.016	100	0	0.011	0.014	100	0	0.012	0.012
100_100_1	81978	100	0	0.02	0.036	100	0	0.002	0.002	100	0	0.002	0.002
100_100_2	190424	100	0	0.01	0.019	100	0	0.013	0.019	100	0	0.013	0.016
100_100_3	225434	100	0	0	0.011	100	0	0.001	0.001	100	0	0.001	0.001
100_100_4	63028	100	0	0.01	0.022	100	0	0.002	0.002	100	0	0.002	0.002
100_100_5	230076	100	0	0	0.009	100	0	0.001	0.001	100	0	0.001	0.001
100_100_6	74358	100	0	0.01	0.032	100	0	0.002	0.002	100	0	0.002	0.002
100_100_7	10330	100	0	0	0.007	100	0	0.002	0.002	100	0	0.002	0.002
100_100_8	62582	100	0	0.01	0.023	100	0	0.002	0.002	100	0	0.002	0.002
100_100_9	232754	100	0	0	0.028	100	0	0.001	0.001	100	0	0.001	0.001
100_100_10	193262	100	0	0.01	0.123	100	0	0.015	0.015	100	0	0.012	0.013
Sum		**4000**				**4000**				**4000**			

The implementation of the feature described in Section 3.7, i.e. updating half of the qubit individuals using QIEA before initialization, may increase computational time taken by the algorithm. The version of algorithm which uses this feature is reported as QIEA-PSA2 while the version of algorithm which does not use this feature is named as QIEA-PSA1.

QIEA-PPA [18] and GGA [12] presented the results on 20 of the BS benchmark instances [13] referred in [25]. Table 1 presents the comparison of QIEA-PSA with QIEA-PPA [18] and GGA [12] on the number of hits and the number of function evaluations (FES) required to reach the best solution in 50 trials made for each instance. The results of GGA and QIEA-PPA reported the generations with fixed FES. If the FES per generation is f, and generations reported for an instance to reach optimal is g, the FES to optimal is calculated as $(g * f + f/2)$.

Table 1 shows that although QIEA-PPA is better than GGA in terms of the number of hits made to optimal solution, but in terms of FES required to reach the solution QIEA-PPA does not win over GGA. Average FES taken, per run out of 30 for the instances reported, by GGA is 14962.83, by QIEA-PPA is 140497.5, by QIEA-PSA1 is 880.74 while by QIEA-PSA2 is 1573.56. It is clear from Table 1 that both versions of QIEA-PSA outperform QIEA-PPA and GGA in terms of both the number of hits made to optimal and average FES taken to reach the optimal. If the solution obtained after initialization (solution obtained from function SolveGreedy) is optimal, then no evolution is required. The FES count is shown as 1 for such instances.

The ABC [15] algorithm has been shown to outperform Mini-Swarm [14] both in terms of probability to reach the optimal and time taken to compute them. Mini-Swarm provides 7590 hits to optimal out of 8000 trials for 100 and 200 object instances while ABC hits optimal 7892 times. (Note: Pulikanti & Singh for ABC [15] reported the optimal value for one of the instances (200_50_6) as 426771 probably by mistake while it is 426777).

Tables 2 and 3 present a comparison of both versions of proposed QIEA-PSA with ABC [15] on the number of hits made to optimal values, standard deviation (SD) and time (minimum MinT and average AvgT) to reach the best solution in 100 trials per instance of benchmark instances [13] with up to 200 binary variables.

Table 3
Comparison of ABC and QIEA-PSA on BS Benchmark Instances of size 200 binary variables.

Problem	Best	ABC				QIEA-PSA1				QIEA-PSA2			
		Hits	SD	MinT	AvgT	Hits	SD	MinT	AvgT	Hits	SD	MinT	AvgT
200_25_1	204441	100	0	0.08	0.118	100	0	0.003	0.003	100	0	0.003	0.003
200_25_2	239573	100	0	0.03	0.044	100	0	0.091	0.095	100	0	0.116	0.119
200_25_3	245463	100	0	0	0.047	100	0	0.091	0.105	100	0	0.111	0.115
200_25_4	222361	100	0	0.07	0.141	100	0	0.091	0.098	100	0	0.113	0.116
200_25_5	187324	100	0	0.12	0.207	100	0	0.003	0.003	100	0	0.003	0.003
200_25_6	80351	100	0	0.25	0.341	83	6.04	0.149	2.680	100	0	0.163	1.880
200_25_7	59036	100	0	0.14	0.286	100	0	0.322	0.834	100	0	0.182	1.212
200_25_8	149433	100	0	0.22	0.36	100	0	0.122	0.417	100	0	0.117	0.213
200_25_9	49366	100	0	0.12	0.15	100	0	0.012	0.012	100	0	0.013	0.013
200_25_10	48459	100	0	0.15	0.201	100	0	0.013	0.013	100	0	0.013	0.013
200_50_1	372097	100	0	0.15	0.201	100	0	0.003	0.003	100	0	0.003	0.003
200_50_2	211130	57	9.9	0.36	5.073	100	0	0.643	2.384	100	0	0.188	4.189
200_50_3	227185	100	0	0.26	0.344	100	0	0.130	0.284	100	0	0.126	0.154
200_50_4	228572	100	0	0.15	0.2	100	0	0.007	0.007	100	0	0.007	0.007
200_50_5	479651	100	0	0	0.047	100	0	0.001	0.001	100	0	0.001	0.001
200_50_6	**426777**	**100**	**0**	**0.04**	**0.084**	**100**	**0**	**0.098**	**0.113**	**100**	**0**	**0.111**	**0.117**
200_50_7	220890	100	0	0.28	0.339	100	0	0.135	0.903	100	0	0.151	0.825
200_50_8	317952	100	0	0.15	0.196	100	0	0.103	0.290	100	0	1.434	1.568
200_50_9	104936	100	0	0.13	0.167	100	0	1.092	2.376	100	0	7.025	8.740
200_50_10	284751	100	0	0.16	0.286	100	0	0.123	0.294	100	0	0.124	0.210
200_75_1	442894	35	101.8	0.24	7.096	100	0	0.192	2.286	100	0	0.248	2.054
200_75_2	286643	100	0	0.2	0.495	99	0.3	1.730	5.705	100	0	3.997	5.522
200_75_3	61924	100	0	0.07	0.103	100	0	0.125	0.415	100	0	0.138	0.299
200_75_4	128351	100	0	0.17	0.204	100	0	0.012	0.012	100	0	0.012	0.012
200_75_5	137885	100	0	0.09	0.125	100	0	0.012	0.013	100	0	0.012	0.012
200_75_6	229631	100	0	0.25	0.301	100	0	0.686	0.872	100	0	0.182	2.960
200_75_7	269887	100	0	0.24	0.351	100	0	0.195	0.644	100	0	0.136	0.743
200_75_8	600858	100	0	0.08	0.12	100	0	0.002	0.002	100	0	0.002	0.002
200_75_9	516771	100	0	0.16	0.284	100	0	0.117	0.131	100	0	0.124	0.132
200_75_10	142694	100	0	0.17	0.194	100	0	0.013	0.013	100	0	0.013	0.013
200_100_1	937149	100	0	0	0.043	100	0	0.109	0.111	100	0	0.121	0.124
200_100_2	303058	100	0	0.29	0.643	100	0	0.601	0.850	100	0	0.163	0.605
200_100_3	29367	100	0	0.02	0.039	100	0	0.110	0.148	100	0	0.137	0.146
200_100_4	100838	100	0	0.1	0.129	100	0	0.014	0.014	100	0	0.014	0.014
200_100_5	786635	100	0	0.08	0.127	100	0	0.112	0.121	100	0	0.124	0.130
200_100_6	41171	100	0	0.04	0.062	100	0	0.014	0.014	100	0	0.015	0.015
200_100_7	701094	100	0	0.16	0.34	100	0	0.116	0.119	100	0	0.122	0.125
200_100_8	782443	100	0	0.12	1.477	99	3.8	0.116	0.677	100	0	0.125	0.152
200_100_9	628992	100	0	0.16	0.984	100	0	0.118	0.135	100	0	0.120	0.128
200_100_10	378442	100	0	0.29	0.35	100	0	0.009	0.009	100	0	0.009	0.009
Sum		**3892**				**3981**				**4000**			

Both ABC and QIEA-PSA hit optimal 100% times for 100 object instances. For 200 object instances ABC hits optimal solution 3892 times within 4000 trials while QIEA-PSA1 hits 3981 times and QIEA-PSA2 hits 4000 times. For 100 object instances QIEA-PSA1 takes less or equal time per run on an average as compared to ABC for 32 instances while QIEA-PSA2 takes less or equal time as compared to ABC in 35 instances out of 40 instances reported. QIEA-PSA2 takes less or equal time than QIEA-PSA1 in 37 instances out of 40 instances reported. For 200 object instances QIEA-PSA1 takes less or equal time per run on an average than ABC for 24 instances while QIEA-PSA2 takes less or equal average time than ABC for 26 instances out of 40 instances reported. QIEA-PSA2 takes less or equal time as compared to QIEA-PSA1 for 25 instances out of 40.

So it is clear that both QIEA-PSA1 and QIEA-PSA2 provide better results than ABC in all problems taking less or equal computational effort in more than 60% of problems.

Table 4 presents results of both versions of QIEA-PSA on instances of size 300 binary variables, it shows that both versions of QIEA-PSA are able to solve 20 instances with very high probability to optimality; QIEA-PSA1 with 0.965 (1993 hits in 2000 trials) and QIEA-PSA2 with 0.96 (1992 hits in 2000 trials). QIEA-PSA2 provides 100% success rate for all except one problem while QIEA-PSA1 misses 100% success rate in 3 problems. QIEA-PSA2 takes less FES than QIEA-PSA1 in 18 out of 20 instances (excluding only 300_25_8 and 300_50_4). QIEA-PSA2 takes less time than QIEA-PSA1 in 13 out of 20 instances.

QIEA-PSA2 gives better results than QIEA-PSA1 for more number of problems. QIEA-PSA1 beats QIEA-PSA2 in terms of time taken to reach optimal in some cases primarily because of extra initialization effort.

Table 5 presents a comparison of performance of QIEA-PSA with MINLP solver and AFSA as presented in [16] on the basis of hits to optimal, average FES and average time (in sec) required to reach optimal out of 30 runs. It is clear that QIEA-PSA outperforms MINLP in terms of the number of FES required and AFSA in terms of optimal count, average time and average number of FES required to reach the optimal.

Table 4
Performance of QIEA-PSA on BS Benchmark Instances of size 300 binary variables.

Problem	Opt	QIEA-PSA1					QIEA-PSA2				
		Hits	Avg	SD	AvgT	Avg FES	Hits	Avg	SD	AvgT	Avg FES
300_25_1	29140	100	29140	0	0.048	1	100	29140	0	0.048	1
300_25_2	281990	99	281989.83	1.7	3.617	10630.535	100	281990	0	3.734	1629.04
300_25_3	231075	100	231075	0	0.027	1	100	231075	0	0.027	1
300_25_4	444759	99	444758.53	4.7	1.032	3105.2626	100	444759	0	0.519	39.22
300_25_5	14988	100	14988	0	3.374	15132.11	100	14988	0	4.493	8938.97
300_25_6	269782	100	269782	0	1.821	2407.36	100	269782	0	2.894	539.06
300_25_7	485263	100	485263	0	0.525	281.66	100	485263	0	0.528	33.84
300_25_8	9343	100	9343	0	1.044	2179.39	100	9343	0	3.887	2489.93
300_25_9	250761	95	250760.5	2.19	13.013	48520.779	92	250760.2	2.73	16.373	30509.5
300_25_10	383377	100	383377	0	0.013	1	100	383377	0	0.013	1
300_50_1	513379	100	513379	0	2.223	3449.77	100	513379	0	1.867	181.64
300_50_2	105543	100	105543	0	0.045	1	100	105543	0	0.046	1
300_50_3	875788	100	875788	0	0.538	232.15	100	875788	0	0.560	50.54
300_50_4	307124	100	307124	0	4.890	6449.96	100	307124	0	23.021	10027.32
300_50_5	727820	100	727820	0	0.957	1256.1	100	727820	0	0.693	64.82
300_50_6	734053	100	734053	0	0.013	1	100	734053	0	0.013	1
300_50_7	43595	100	43595	0	1.235	1785.3	100	43595	0	3.688	1037.66
300_50_8	767977	100	767977	0	0.567	143.54	100	767977	0	0.514	16.62
300_50_9	761351	100	761351	0	0.012	1	100	761351	0	0.012	1
300_50_10	996070	100	996070	0	0.005	1	100	996070	0	0.005	1
Sum		**1993**					**1992**				

Table 5
Comparison of MINLP, bAFSA and QIEA-PSA on hits and average FES in 30 runs.

	MINLP solver		bAFSA			QIEA-PSA1			QIEA-PSA2		
	Opt	FES	Hits	Avg FES	AvgT	Hits	Avg FES	AvgT	Hits	Avg FES	AvgT
100_25_1	18558	8364	6	14183	2.51	30	1957.43	0.124	30	2513.07	0.670
100_25_2	56525	706	18	11062	1.47	30	1	0.001	30	1	0.001
100_25_3	3752	1713	6	10050	1.3	30	89.3	0.019	30	8.27	0.016
100_25_4	50382	61	12	18301	2.47	30	7351.8	0.186	30	30246.27	0.807
100_25_5	61494	5130	26	3912	0.57	30	1	0.001	30	1	0.001
100_50_1	83742	7062	22	13716	1.9	30	79.1	0.017	30	2.57	0.013
100_50_2	104856	10071	2	13851	1.88	30	89.2	0.017	30	15.4	0.015
100_50_3	34006	17786	11	14564	2.05	30	1228.47	0.102	30	69.03	0.049
100_50_4	105996	157	27	5937	0.91	30	1	0.001	30	1	0.001
100_50_5	56464	7478	15	7815	1.35	30	1	0.001	30	1	0.001
100_75_1	189137	7	30	490	0.07	30	1	0.001	30	1	0.001
100_75_2	95074	173495	5	20643	2.81	30	1	0.001	30	1	0.001
100_75_3	62098	3447	19	8432	1.56	30	1	0.002	30	1	0.002
100_75_4	72245	301913	8	10426	1.85	30	95.6	0.025	30	9.83	0.016
100_75_5	27616	14023	16	10288	1.39	30	892.3	0.087	30	1277.47	0.575
100_100_1	81978	38198	11	12391	1.75	30	1	0.002	30	1	0.002
100_100_2	190424	37144	6	12133	1.9	30	106.6	0.019	30	24.4	0.016
100_100_3	225434	1590	6	20968	2.66	30	1	0.001	30	1	0.001
100_100_5	230076	1370	19	24659	3.1	30	1	0.002	30	1	0.002
100_100_6	74358	13569	22	14255	2.01	30	1	0.002	30	1	0.002
Sum		**643284**	**287**	**248076**	**35.51**	**600**	**11900.8**	**0.608**	**600**	**34177.31**	**2.189**

5. Conclusions

This work presents a QIEA which is improved to make it competitive with state-of the art approaches for a difficult combinatorial optimization problem i.e. QKP. Improvements introduced in QIEA are better initialization, judicious use of local search, re-initialization, mutation, purge operation and local exploitation before global exploration. The endeavour has been to create effective mix of these features in an effort towards balancing the exploration and exploitation which remains the overarching goal in any attempted implementation of QIEA or, in general, population based search algorithm.

The improvements introduced are generic to be considered for other similar and not so similar problems.

Acknowledgements

Authors are grateful to Department of Science and Technology, India (DST) and Deutsche Forschungsgemeinschaft, Germany (DFG) for the support under project No. INT/FRG/DFG/P-38/2012 titled "Algorithm Engineering of Quantum Evolutionary Algorithms for Hard Optimization Problems".

References

[1] G. Gallo, P. Hammer, B. Simeone, Quadratic Knapsack problems, Math. Program. Study 12 (1980) 132–149.
[2] A. Caprara, D. Pisinger, P. Toth, Exact solution of the quadratic Knapsack problem, INFORMS J. Comput. 11 (2) (1999) 125–137.
[3] D. Pisinger, The quadratic Knapsack problem—a survey, Disc. Appl. Math. 155 (2007) 623–648.
[4] J. Rhys, A selection problem of shared fixed costs and network flows, Manag. Sci. 17 (1970) 200–207.
[5] C. Witzall, Mathematical methods of site selection for electronic message system (EMS), NBS Int. Rep., Tech. Rep. 1975.
[6] E. Johnson, A. Mehrotra, G. Nemhauser, Min-cut clustering, Math. Program. 62 (1993) 133–151.
[7] D.L. Laghhunn, Quadratic binary programming with applications to capital budgetting problems, Oper. Res. 18 (1970) 454–461.
[8] C.E. Ferreira, A. Martin, C.C. deSouza, Formulations and valid inequalities for node capacitated graph partitioning, Math. Program. 74 (1996) 247–266.
[9] G. Dijkhuijen, U. Faigle, A cutting-plane approach to he edge-weighted maximal clique problem, European J. Oper. Res. 69 (1993) 121–130.
[10] K. Park, K. Lee, S. Park, An extended formulation approach to the edge weighted maximal clique problem, European J. Oper. Res. 95 (1996) 671–682.
[11] P.L. Hammer, D.J. Rader, Efficient Methods for solving quadratic 0–1 Knapsack problem, INFOR 35 (1997) 179–182.
[12] B.A. Julstrom, Greedy, genetic and greedy genetic algorithms for the quadratic Knapsack problem, in: Proceedings of the GECCO'05, pp. 607–614 (2005).
[13] A. Billionet, E. Soutif, (2004) QKP Instances. [Online]. http://cedric.cnam.fr/~soutif/QKP/QKP.html.
[14] X. Xie, J. Liu, A Mini-Swarm for the quadratic Knapsack problem, in: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 190–197 (2007).
[15] S. Pulikanti, A. Singh, An artificial bee colony algorithm for the quadratic Knapsack problem, in: Proceeding of the ICONIP'09, Part II, in: LNCS, vol. 5864, Springer-Verlag, Heidelberg, 2009, pp. 196–205.
[16] M.A.K. Azad, M.A.C. Rocha, M.G.P. Fernandes, Solving 0–1 Quadratic Knapsack Problem with a Population-based Artificial Fish Swarm Algorithm, in: Proceedings of the International Conference on Applied and Computational Mathematics (2012).
[17] M.A.K. Azad, M.A.C. Rocha, E.M.G.P. Fernandes, A simplified binary artificial fish swarm algorithm for 0–1 quadratic Knapsack problems, J. Comput. Appl. Math. (2013) [Online]. http://www.sciencedirect.com/science/article/pii/S0377042713005074.
[18] C. Patvardhan, P. Prakash, A. Srivastav, A novel Quantum-Inspired Evolutionary Algorithm for the quadratic Knapsack problem, Int. J. Math. Oper. Res. 4 (2) (2012) 114–127.
[19] K.-H. Han, J.-H. Kim, Quantum-Inspired Evolutionary Algorithm for a class of combinatorial optimization, IEEE Tran. Evol. Comput. 6 (6) (2002) 580–593.
[20] M.D. Platel, S. Schliebs, N. Kasabov, Quantum-Inspired Evolutionary Algorithm: A multimodel EDA, IEEE Tran. Evol. Comp. 13 (6) (2009) 1218–1232.
[21] G. Zhang, Quantum-Inspired Evolutionary Algorithms: a survey and empirical study, J. Heur. 17 (3) (2011) 303–351.
[22] K.-H. Han, J.-H. Kim, On setting the parameters of quantum-inspired evolutionary algorithm for practical application, in: Proceedings of the CEC'03, pp. 178–184 (2003).
[23] K.-H. Han, On the analysis of the quantum-inspired evolutionary algorithm with a single individual, in: Proceedings of the CEC'06, Vancouver, Canada, 2006.
[24] K.-H. Han, J.-H. Kim, Quantum-Inspired Evolutionary Algorithms with a new termination criterion, h-epsilon gate, and two-phase scheme, IEEE Trans. Evol. Comput. 8 (2) (2004) 156–169.
[25] A. Billionet, E. Soutif, An exact method based on Lagrangian decomposition for the 0–1 quadratic Knapsack problem, European J. Oper. Res. 157 (2004) 565–575.