Swarm and Evolutionary Computation ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Regular Paper

Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems

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

[a] Faculty of Engineering, Dayalbagh Educational Institute, Agra 282010, India
[b] Institut für Informatik, Kiel University, 24098 Kiel, Germany

ARTICLE INFO

Article history:
Received 11 May 2015
Received in revised form
28 August 2015
Accepted 23 September 2015

Keywords:
Combinatorial Optimization
Large Size Quadratic Knapsack Problem
Parallel Algorithm
Quantum Inspired Evolutionary Algorithm

ABSTRACT

Quadratic Knapsack Problem (QKP), an extension of the canonical simple Knapsack Problem, is NP Hard in the stronger sense. 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. A few attempts have been made to solve large size instances of QKP due to its complexity. Quantum Inspired Evolutionary Algorithm (QIEA) provides a generic framework that has often been carefully tailored for a given problem to obtain an effective implementation. In this work, an improved and parallelized QIEA, dubbed IQIEA-P is presented. Several additional features make it more balanced in exploration and exploitation and thus have better applicability. Computational experiments are presented on large QKP instances of 1000 and 2000 items. The improvements are inherently parallelizable and, therefore, good speedups are obtained on a multi-core machine. No parallel algorithm is available for QKP. The solutions provided by QIEA-P are competitive with those obtained from the state of the art algorithm.

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. Without loss of generality matrix P is considered symmetric such that $p_{ij}=p_{ji}$ for all i and j. Hence, additional profit achieved if both items i and j are selected is considered as p_{ij} rather than $p_{ij}+p_{ji}$, for $j > i$. QKP is to find a subset of items whose total weight is not more than the knapsack capacity c such that the overall profit is maximized. If x_j is binary variable which is equal to 1 if jth item is selected and 0 otherwise, the problem is formulated as follows.

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

The KP is a particular case of QKP which arises when $p_{ij}=0$ for all $i \neq j$. The *Clique* problem, is another particular case of QKP, which requires checking whether, for a given integer k, a given undirected graph $G=(V, E)$ contains a complete subgraph on k nodes. 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. *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. Nevertheless, it has 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].

* 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, Pipal Mandi, Agra 282003, India.

http://dx.doi.org/10.1016/j.swevo.2015.09.005
2210-6502/

Gallo et al. [1] introduced QKP and presented a method to derive upper bounds using upper planes. Several attempts have been made to solve QKP [3]. Some recent ones are as follows. Pisinger et al. [11] presented an algorithm based on Lagrangian relaxation\decomposition and aggressive reduction. It has been shown to solve some large-sized instances with 1500 binary variables. Le´tocart et al. [12] presented another algorithm based on a re-optimization technique to accelerate the resolution of each independent sequence of 0–1 linear knapsack problems induced by the Lagrangian relaxation\decomposition. Computational results for randomly generated instances of 600 binary variables were presented. Large-sized benchmark instances of Pisinger et al. [11] (1500 binary variables) and Le´tocart et al. [12] (600 binary variables) were randomly generated and tested. They have not been recorded by the authors [13].

Existing standard deterministic approaches like CPLEX can not solve large instances of QKP. Several studies on heuristic and meta-heuristic methods have also been made in the literature. These provide satisfactory solutions for QKP within reasonable time. Some effective heuristic and meta-heuristic methods applied in last few decades to solve QKP are given in Table 1. From the table, it is clear that state of the art method applied on large QKP instances is GRASP and Tabu Search proposed recently by Yang et al. [13]. A GRASP and tabu search method [13] solves larger instances of size 1000 and 2000 variables. No parallel algorithm exists which solves large size QKP.

Evolutionary Algorithms (EA), inspired by natural selection, mimic iterative evolutionary processes with a set of solutions encoded in a population. The population evolves based on the rule of "survival of the fittest" [14]. The computational challenges are faced due to problem difficulty and size, the complexity of fitness function, and distribution characteristics of solution space, and also on runtime efficiency of stochastic search [15]. EA's are considered inherently parallelisable [16].

QIEA refers to a subclass of EA where representation and evolution is implemented based on concept of Quantum computing. Similar to EA, QIEA exhibit the property of inherent parallelism embedded in the evolutionary process. Some attempts have been made in literature to utilize parallel implementations of QIEA for simple KP [17,18].

In this work, an improved and parallelized QIEA, dubbed IQIEA-P is presented with several additional features to make it more balanced in exploration and exploitation and also have better applicability to different types of combinatorial optimization problems. The improvements are inherently parallelizable and, therefore, good speedups are obtained on a multi-core machine. This is the first attempt to parallelise the QIEA for QKP. This attempt in fact presents the first parallel algorithm to solve large instances of QKP. Computational experiments are presented on large QKP instances used by Yang et al. [13] (1000 and 2000 binary variables) which have been obtained on request. Quality of solutions provided by IQIEA-P is competitive to best known results. Parallelization provides good speedup.

The rest of the paper is organized as follows. The basic concept of QIEA is explained in Section 2. In Section 3 the proposed IQIEA-P is presented. The primary differences of the strategy used to improve QIEA in present work as compared to earlier QIEAs are discussed. A comparison of IQIEA-P with sequential version (IQIEA) is done in Section 4. Conclusions and future work are discussed in Section 5.

2. Quantum inspired evolutionary algorithm (QIEA)

The QIEA introduced in [28] is population-based stochastic evolutionary algorithm. It uses the qubit, a vector, to represent the probabilistic state of individual. Each qubit is represented as $q_i = \begin{bmatrix}\alpha_i \\ \beta_i\end{bmatrix}$, where α_i, β_i are complex numbers so that $|\alpha_i|^2$ is the probability of state being 0 and $|\beta_i|^2$ is the probability of state being 1 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 Q, 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 P is generated randomly, the ith bit P_i being 1 with probability Q_i^2 independent of other bits. 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 qubits 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 [28].

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

$$\begin{bmatrix}\alpha_i^{t+1} \\ \beta_i^{t+1}\end{bmatrix} = \begin{bmatrix}\cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i)\end{bmatrix} \begin{bmatrix}\alpha_i^{t} \\ \beta_i^{t}\end{bmatrix} \quad (2)$$

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 best solution. The value for $\Delta\theta_i$ is chosen to be 0.01 when observed solution is not better than best solution found till the time of observation.

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.

QIEA of [25] (dubbed here QIEA-QKP) could give near optimal solutions for QKP benchmark instances of size up to 200 binary variables in reasonable time. Following improvements have been made in the original QIEA to obtain QIEA-QKP.

- To update the qubit individuals the rotation gate used is slightly different, which may assign different rotation angle to different qubits depending on the bits of observed current solution and best solution.
- A rudimentary local search technique is used which generate n solutions in neighborhood of solutions (provided by observation of qubit individuals) and keep the best.
- Migration Operator is removed.

A host of QIEA-based attempts have been reported in the literature that utilizes QIEAs for the solution of a wide variety of problems [31,32]. QIEAs, of course, are not a "one-size fits all" solution. The No Free lunch Theorem prohibits that. However, a particular QIEA has to be designed for the problem at hand to achieve high performance with respect to the state-of-art algorithms for the problem. The primary strengths and weaknesses of the QIEAs are briefly discussed in Table 2. Many of the weaknesses are shared with other EAs as well.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

Table 1
Effective Heuristic and Meta-heuristic methods applied on QKP.

Methods	Authors (Year)	Instances	Significant achievements	Brief description of results
Linearization and exchange (LEX) heuristic	Hammer and Rader (1997) [19]	Instances of 100 variables	Achieved profit more than 99% of optimal in less than 1 s.	Presented error of solutions obtained from optimal and computational time to solve.
Greedy genetic algorithm (GGA)	Julstrom (2005) [20]	Sample BS benchmark instances of 100 and 200 variables	Solved to optimality with probability 0.902 using around 15000 Function Evaluations (FES) on an average.	Presented number of times it finds optimum within 50 runs for GA. Also both the generations required to reach optimal and time in seconds to reach optimal are reported. The FES required per generation are mentioned.
Mini-swarm algorithm	Xie and Liu (2007) [21]	All [a]BS benchmark instances 100 and 200 variables	Solved to optimality with probability 0.949 requiring 1 s on an average	Reported number of hits made to optimal in 100 runs and average execution time per run in seconds
Artificial Bee Colony (ABC) algorithm	Pulikanti and Singh (2009) [22]	All [a]BS benchmark instances 100 and 200 variables	Solved to optimality with probability 0.987 requiring 22 s on an average	Reported quality of solution in terms of number of hits made to optimal and average value obtained in 100 runs for each problem and computational cost in terms of minimum and average time to reach optimal
Artificial fish swarm algorithm (AFSA)	Azad et al. (2011,2014) [23,24]	Sample [a]BS benchmark instances 100 and 200 variables	Reported to return optimal with probability of around 0.5 requiring 12500 FES and 35.5 s on an average.	Presented Number of successful runs and average function value. Average FES and average time required in seconds per run out of 30 runs and 50 runs have also been reported
Quantum Inspired Evolutionary Algorithm (QIEA)	Patvardhan et al. (2012) [25]	Sample [a]BS benchmark instances 100 and 200 variables	Solved to optimality with probability 0.944 using around 140,000 FES on an average per trial	Reported average iterations required to optimal, number of hits made to optimal and average function value in 50 runs for each problem. The FES required per iteration are mentioned.
GRASP and Tabu search	Yang et al. (2013) [13]	[a]BS Benchmark Instances of sizes 100, 200, 300 binary variables. Benchmark Instances of 1000, 2000 binary variables	Solved 1000 and 2000 variable instances with average gap of 1.403 from known tightest Upper bound with average probability of 0.916 and relative percentage deviation from best as 0.0008 taking 288.39 s on an average. The comparison on the results of [a]BS benchmark instances of size 100 and 200 binary variables shows that Grasp and Tabu outperforms then state of the art.	Presented gap of best solution from upper bound and hits made to best and relative percenatge deviation (rpd) from best in 100 runs. Also reported the average time required to solve in seconds.

[a] BS Benchmark Instances means Instances [26] referred by Billionet and Soutif [27].

Table 2
Strengths and weaknesses of Quantum Inspired Evolutionary Algorithm.

Strengths	Weaknesses
1. QIEAs have better representation power using qubits to enable use of smaller populations (ideally even a size of 1) [29,30]. Smaller populations require lesser computation during search.	1. 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.
2. QIEAs have an Estimation of Distribution Algorithm (EDA) style functioning with implicit determination of distributions leading to better solutions [31,32].	2. Inclusion of features promoting faster convergence may cause the algorithm to get stuck in local optima.
3. 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 [32,30]. This makes them very versatile in their applicability	3. Slow convergence limits the problem sizes that can be tackled using QIEAs.
4. The QIEA framework also provides the flexibility necessary for the inclusion of features appropriate for a given problem towards delivering better search performance [32].	4. 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.
5. QIEA inherently favors exploration of the search space initially gradually shifting towards exploitation as the search progresses which is a desirable aspect [30].	
6. There is a possibility of utilizing one of several termination criteria appropriate for the problem at hand [33].	

3. Proposed parallel improved QIEA (IQIEA-P)

The proposed framework of IQIEA-P, as shown in Fig. 1, is implemented on a machine having multi-core host CPU processor using OpenMP®. Number of individuals to be processed on a single thread or process depends on capability of machine and total number of individuals required for effective performance on the given instance.

The IQIEA-P executes through several phases described as follows.

I. *Parallel generation of sort orders.* The input elements are considered in an order based on the knowledge about their priority for inclusion in the knapsack, as described in Section 3.1. Multiple sort orders are considered for QKP in order to provide requisite diversity in the population of solutions. Consideration of Multiple sort orders implies additional computational burden. To mitigate this, these sort orders are processed simultaneously on parallel threads or processes.

II. *Parallel Initialization and exploitation of the individuals.* Steps during initialization viz., parallel initialization of Qubit individuals (described in Section 3.1.1), faster exploitation of some qubit individuals in parallel (described in Section 3.3), parallel initialization of local best solutions are implemented as separate parallel sections performed in sequence one after another.

III. *Parallel execution of independent portions of IQIEA-P:* There are several independent portions in the proposed algorithmic structure (Section 3.2) with steps such as mutation, re-initialization, purge, improvement in repair and specialized local search (described in Sections 3.1 and 3.4 through 3.7) embedded in them. These are executed on separate threads or processes. These large parallel portions result in high speedups in IQIEA-P.

3.1. Multiple greedy sort orders evaluated in parallel

The QIEAs in the form of qubit individuals, estimate the probability of the bits in an individual to be set to 1 (or 0) in an optimal solution on the basis of the history of solutions generated during evolution. Some good heuristic can be used to estimate the relative preferences of items to be selected while initializing the qubit individuals and also when repairing the infeasible solutions to reduce the task of QIEA ahead. The priority order of items to be selected in a particular problem instance dubbed GreedySortOrder, is computed as shown in Fig. 2. The psedo-code SortGreedy is explained as follows.

Starting with an empty solution, elements are added iteratively. The item that provides maximum individual profit is selected as the first item. During subsequent iterations, the item having maximum Relative Value Density (RVD) with respect to the partial solution available before the iteration is added. If P is a partial solution such that $k \in P$ implies kth item is selected. The relative value density of any item $i \in \{1,\ldots,n\}$ with respect to P, RVD_i^P is computed as $(p_{ii} + \sum_{j \in P/i} p_{ij})/w_i$.

Items selected according to the GreedySortOrder described above may not provide the optimal solution. However, if a different first item is chosen in step 3a different selection results. Thus, multiple orders are generated by choosing different items as first item in other generated orders (MultipleSortGreedy). These are then used to initialize different qubit individuals and repair different observed solutions.

The computation cost involved in finding a GreedySortOrder is high. However, other orders can be generated independently and, so, are generated in parallel in IQIEA-P. In each of such orders determined by MultipleSortGreedy the first element is chosen randomly from the first 30% of elements in the GreedySortOrder. The rest of the items are selected in the same manner as in GreedySortOrder.

3.1.1. Initializing the qubit individuals using MultipleSortGreedy

The qubit individuals are initialized as shown in Fig. 3. The items are divided in 3-parts based on where they lie in the sequence – the first part contains items having high preference for selection in knapsack, second part having medium preference and third having low preference. Hence, qubits for items lying in first part (third part) are assigned values closer to 1 (0). Qubits for items lying in second part necessarily require more search for convergence to either 0 or 1, hence medium values between 0 and 1 are assigned to them. As a result, IQIEA-P starts search in areas of the solution space favored by such initialized qubit individuals.

Initializing qubits based on a single sort order GO obtained from GreedySortOrder would limit the exploitation to a single area. Thus, multiple sort orders GO_i, $i \in 1,\ldots r$, are used in round-robin fashion to initialize all the qubit individuals.

The process of initialization of different qubit individuals is performed in parallel as they can be initialized independent of each other.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

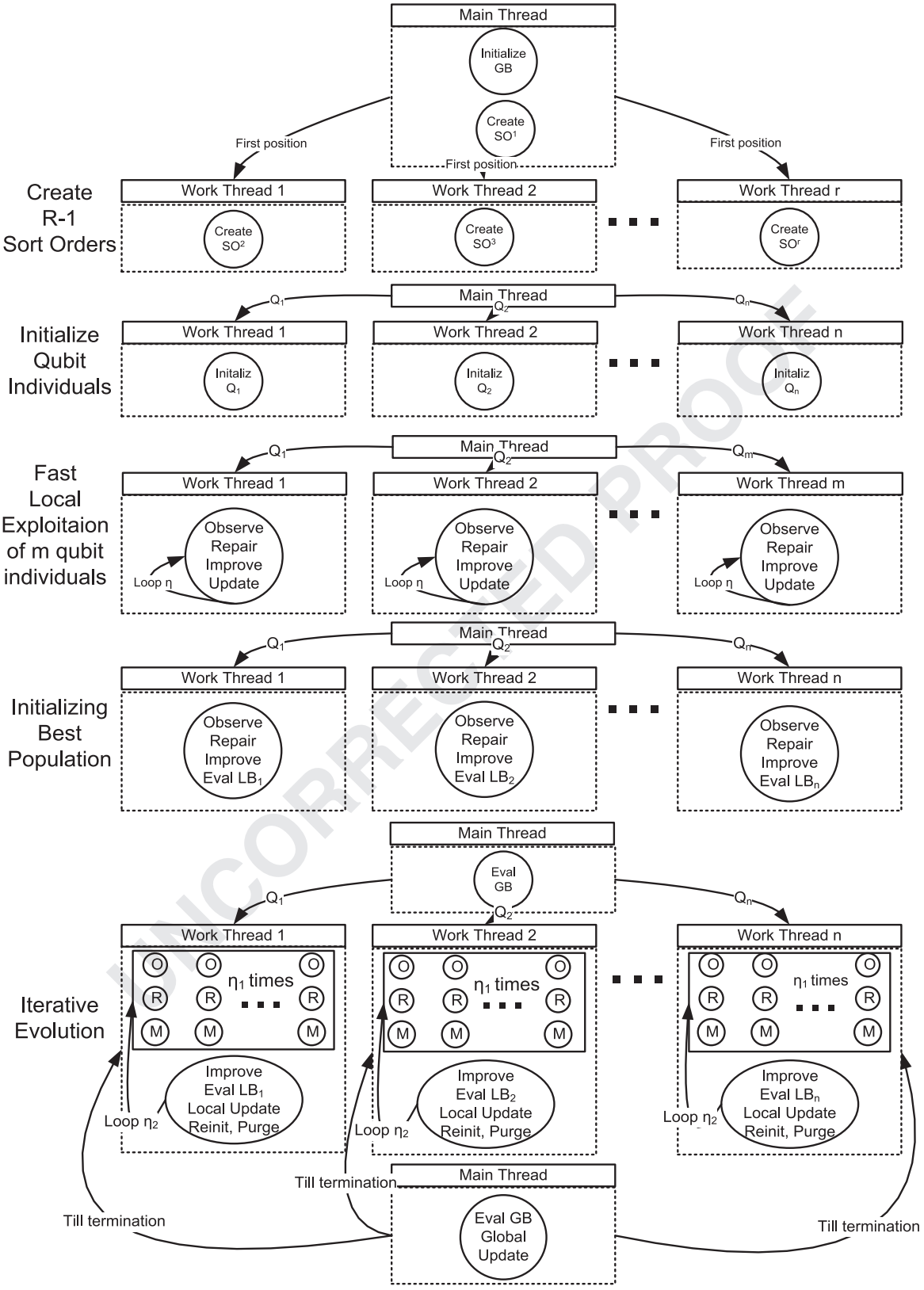

Fig. 1. The IQIEA-P framework. (O, R, M refers to operations Observe, Repair and Mutate respectively). (Note: GB: Global Best Solution, LB_i: *i*th Local Best Solution, Q_i: *i*th Qubit individual [$i \in 1,\ldots,n$], SO^j: *j*th sort order $j \in 1,\ldots,r$}).

3.1.2. *Repairing observed solutions using MultipleSortGreedy*

A QIEA uses a simple repair function to make the observed solutions feasible. In IQIEA, during each repair step, items having lower preference according to GreedySortOrder are removed and items having higher preference are included whenever required. This improves the quality of solution obtained from each repair and hence improves the speed of convergence. The pseudo-code is given in Fig. 4.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

```
Function SortGreedy() /*returns GO*/
1    PS ← Ø;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} RVD_i^PS; GO[j] ← i; PS ← PS ∪ i;
7    }// * for j *//
8    return(GO);
```

Fig. 2. Pseudo-code for SortGreedy returning GreedySortOrder.

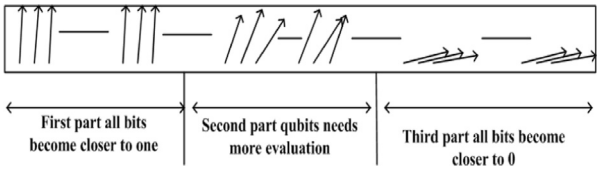

Fig. 3. Initialization of qubits. Items, for which qubits are shown, are sorted in a sort order from left to right.

```
Procedure RepairGreedy (x, GO )
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 GO[r] to GO[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 GO[1] to GO[r] {
10       if ((x_j=0) and (weight + w_j ≤ C)){
11          x_j ←1; weight← weight +w_j; }
12     } //*for j*//
```

Fig. 4. Pseudo-code for RepairGreedy.

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 to maintain diversity in the repaired solutions.

Repairing the solutions generated from different qubit individuals can be done in parallel and so the overhead is shared among the threads executing in parallel.

3.2. Parallelisable balanced structure

The structure of QIEA introduced by Han and Kim [28] is modified here. The proposed structure, as shown in Fig. 5, provides better balance between exploitation and exploration capabilities and enhances the parallelisability.

Local best solutions are replaced by either global best solution or neighborhood local best solutions after global migration period or local migration period in the original structure. In the proposed structure the local best solutions are never replaced by global best solution. These solutions are used as attractors in the Q-gate to update the qubit individuals at two different levels. The qubits in the proposed structure are updated using local best solution and the global best solution separately. This modification benefits the evolution primarily in two ways as follows.

- In the modified structure multiple attractors are maintained. All of which may not be same as the best solution found so far globally. Using global best solution as attractor most of the time (as is evident in simple QIEA where global migration converts all attractors to be same as the global best) leads to early convergence of the qubit individuals to probably a sub-optimal solution. Different local attractors for different qubit individuals guide them to search different search areas in parallel. The idea of using multiple attractors or non-elitist attractor has also been used in previous attempts to avoid local optima. In versatile QIEA proposed by Platel et al. [34] the attractor had to be replaced by the currently generated solution unconditionally. So the attractor changes more frequently than the original QIEA and may become inferior to the best solution found so far i.e. a non-elitist attractor. The QIEA proposed by Lasse et al. [35] maintains a population of attractors selected using tournament selection performed on a set of generated good solutions.
- The proposed algorithm not only uses multiple attractors but also uses different attractors to update each qubit individual at the same time. The multiple local best solutions are used as attractors to preserve the local traits. The global best solution preserves the information about the global optimum of the evolving population found so far. Preserving these separately for a longer time helps in steady and controlled evolution. It is also a pivotal feature of memetic computation, considered an effective way of hybridizing the population based search techniques [36–39].

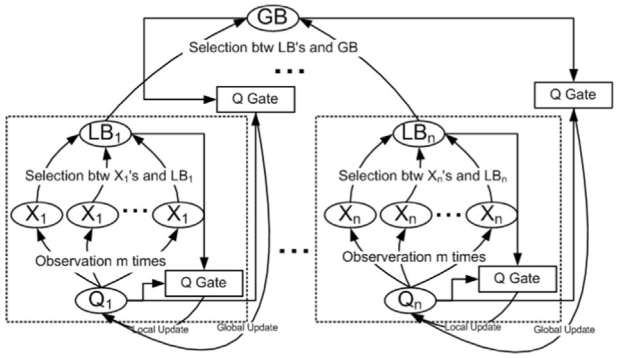

Fig. 5. Structure of IQIEA-P with balanced exploration and exploitation.

In the proposed structure multiple observations are made for a qubit individual at a time before selecting local best solution and updating qubit individual. This helps in many ways viz.,

- It helps in better exploitation of the areas represented by the qubit individuals.
- It helps to assess the health of individuals in current qubit population. For example if multiple observations made on a qubit individual lead to same solution every time then the qubit individual may be considered as the one which has converged to extremes and may not help in further evolution.

A larger proportion of work can be performed independently in an iteration of QIEA in the proposed structure. More observations are performed for a qubit individual followed by several rotations towards local best solution before synchronising to select the global best solution and rotating towards it (global rotation). The dotted regions in Fig. 5 indicate the portions of algorithm that can be executed in parallel.

3.3. Faster exploitation of systematically Initialized Qubit Individuals

The sort orders as described in Section 3.2 help to initialize the qubit individuals to have better estimation of probability

distribution in order to avoid redundant exploitation of solution space. The iterations of QIEA when executed on initialized qubit individual further improve these estimations. On the other hand, executing QIEA for few number of iteration also performs a quick scan of promising regions of search space. It may help in solving the easier instances quickly. Hence, the basic steps of QIEA listed below are performed on a quarter of the qubit individuals for small number of times (here empirically taken as quarter of total iterations performed in main QIEA) before initializing the population of local best solutions.

- Make
- RepairGreedy
- ImproveLocal
- Update

Another quarter of qubit individuals are then initialized to the qubit individuals as they appear after the local exploitation. This task can be executed as independent for each individual hence performed in parallel.

3.4. Specialized local search procedure for QKP to improve intermediate solutions

The local best solutions are improved using a local search function specially designed for QKP. It tries to explore the quality of all solutions in the neighborhood 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 (replacement) is performed for the individual (pair) which results in maximum gain in profit. Let *P* be a feasible solution, the pseudo-code of procedure used to improve profit of *P* is given in Fig. 6.

The function ImproveLocal is computationally expensive and might lead to the local optimum. RandImproveLocal is a lighter version which performs comparison of only randomly selected actions (inclusion and replacement) before replacing. It reduces the computational effort and helps avoiding local optima.

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

Fig. 6. Pseudo-code for ImproveLocal.

RandImproveLocal is performed on half of the population of individuals and ImproveLocal on the other half. These local search functions are executed independently for different individuals by parallel threads.

3.5. Mutation of solutions appearing to be stuck in local optimum

All EAs suffer from tendency of getting stuck in local optima. Here the chances are higher as both the repair and improve procedures try to narrow down the search to the local best solutions. To overcome this problem, a generated solution is mutated when found close to the previously recorded global best solution i.e. Hamming distance between them is small. This is done to divert the search to a different region. 2–3 bits in the solution vector are randomly selected and changed to 0. Elements with better RVD are then iteratively included in solution as long as the solution remains feasible.

This operator improves diversity without increasing the computational effort too much. It also helps to explore the solution space around a current solution such that no local optimal in vicinity is missed. This improves the chances of finding optimal in case it is in the vicinity.

This function is performed independently on different solutions so the computational overhead posed by the mutation is shared among the threads working on different qubit individuals in parallel.

3.6. Re-initialization of qubit individuals

After some generations, if qubits of an individual converge all observations would yield the same solution. Such qubit strings are re-initialized to restore diversity. Qubit individuals, which generate same solution more than 3 times out of 5 are re-initialized. The qubits are set to $1/\sqrt{2}$ in the first half of the search and, in the second half, they are re-initialized using sort orders as given in Section 3.1.1.

3.7. Purging the non-performing qubit individuals stochastically (StochasticPurge)

The qubit individuals need to provide some improved solution within specified number of iterations (say, 5). At the end of these iterations, the qubit individual is purged with probability of 0.5 if fitness of the best solution generated by it is worse than the average of the worst and best solution generated so far globally. On purging, it is replaced by the best qubit individual found so far.

3.8. Initializing global best (GB) using heuristic and specialized local search

The initial global best solution (GB) is set to the solution found using local search procedure described in Section 3.4 on the initial Greedy Solution found using pseudo-code in Fig. 7. This helps in

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

Fig. 7. Pseudo-code for SolveGreedy.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

```
Procedure IQIEA-P
1   Initialize no. of sort orders and population size
2   SolveGreedy(GSolution),
3   MultipleSortGreedy(GreedySortOrders, Nsortorder) In Parallel
4   t ← 0, C_orginal ← C, b ← GSolution, we← GSolution, be← 0
5   InitializeGreedy( q_j^t, GreedySortOrders) for each j ∈ {1, … , n} In Parallel
6   Faster Exploitation of quarter of qubit individuals in Parallel as described in section 3.3
7   Copy updated quarter qubits in step 5 to another quarter in Parallel
8   Initialize b_j^t for each j ∈ {1, … , n} using observations on Q(t) in Parallel
9   while ( t<MaxIterations) {
10    for j from 0 to n do in Parallel{
11      for r from 0 to η_1 do{
12        dupcnt=0;
13        for s from 0 to η_2 do{
14          Observe Q(t) obtaining P(s)
15          RepairGreedy (P(s),GreedySortOrders)
16          for each j ∈ {1, … , n} if (HamDistance( p_j^s,b)<2) Mutate( p_j^s)
17          Evaluate P(s)
18          for each j ∈ {1, … , n} if ( p_j^s better than p_j^t ) then p_j^t ← p_j^s
19          if ( p_j^s is same as p_j^{s-1} ) then dupcnt=dupcnt+1;
20        } //*for s*//
21        if (dupcnt satisfy criterion) ReInitialize(q_j^t);
22      }//*for r*//
23      for j ∈ {1, … , n/2} ImproveLocal p_j^t ;for j ∈ {n/2, … , n} RandImproveLocal p_j^t
24      for each j ∈ {1, … , n} if ( p_j^t is better than b_j^t) then b_j^t← p_j^t
25      for each j ∈ {1, … , n} Update q_j^t based on b_j^t
26      StochasticPurge (B(t),be,we, Q(t),bqbit)
27    } //*for j*//
28    for each j ∈ {1, … , n} if (b_j^t is better than b) then b← b_j^t , bqbit ← q_j^t
29    for each j ∈ {1, … , n} Update q_j^t based on b in Parallel
30    t ← t+1; Q(t) ← Q(t-1); B(t) ← B(t-1);
31  } // *while*//
```

Fig. 8. Pseudo-code for IQIEA-P.

solving the easy instances faster. Moreover, initializing the global attractor with a good solution rather than a random solution helps in faster convergence to a better solution.

The simple constructive heuristic for finding Greedy Solution works as follows. 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. An item having minimum RVD is removed from solution each time. GS is a set representing the greedy solution.

The complete pseudo-code for Improved QIEA implemented for parallel execution is presented in Fig. 8. Following notations are used.

***Q*(*t*):**	Qubit population in *t*th iteration.
***P*(*t*):**	population of binary solutions in *t*th iteration.
***B*(*t*):**	population of best solutions found till *t*th iteration.
q_j^t	*j*th individual in $Q(t)$.
p_j^t**:**	*j*th individual in $P(t)$.
b_j^t**:**	*j*th individual in $B(t)$. Same as LBs (Fig. 6).
***b*:**	best solution observed so far. Same as GB.
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

MultipleSortGreedy (GreedySortOrders, Nsortorder): SortGreedy() is executed Nsortorders times, to obtain that many sort orders in GreedySortOrders, with different choice of first element. Here the first sort order is generated by executing the function exactly as coded in SortGreedy(), while *n*th sort order, for $n > 1$, is generated with the first element chosen randomly from 30% initial elements of 1st sort order.

InitializeGreedy (q_j^t, GO): GO is 2-D array containing r sort orders in total such that GO[i] refers to *i*th 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 r].

ReInitialize(q_j^t): For initial half iterations all qubits of q_j^t are set to $1/\sqrt{2}$. For rest of the iterations q_j^t is initialized using **InitializeGreedy.**

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 p_j^s and *b*.

Mutate(p_j^s): Mutates the solution p_j^s as described in Section 3.5.

Update q_j^t based onb_j^t: Rotates the qubits q_j^t towards bits in b_j^t as explained earlier and defined in [28] using the rotation angle as 0.01.

StochasticPurge ($B(t)$, be, we, $Q(t)$, bqbit): Sets be and we to the best and worst values in $B(t)$, be and we. Replaces the qubit individual q_j^t by bqbit if value of b_j^t is below (be+we)/2.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

Table 3
Improved performance of IQIEA-P over IQIEA [13] for QKP Instances of size 1000 and 2000 variables.

n_d_i	Best Known	Best achieved	IQIEA			IQIEA-P			Speed Up
			RPD	SR	*t*(s)	RPD	SR	*t*(s)	
1000_25_1	6172407	6172407	0	100	0.23	0	100	0.146	**1.55**
1000_25_2	229941	229941	0	100	252.50	0	100	17.824	**14.17**
1000_25_3	172418	172418	0	100	176.58	0	100	14.042	**12.57**
1000_25_4	367426	367426	0	100	4.49	0	100	2.735	**1.64**
1000_25_5	4885611	4885611	0	100	169.97	0	100	18.023	**9.43**
1000_25_6	15689	15689	0	100	3.88	0	100	2.794	**1.39**
1000_25_7	4945810	4945810	0	100	151.42	0	100	10.635	**14.24**
1000_25_8	1710198	1710198	0	100	372.25	0	100	38.016	**9.79**
1000_25_9	496315	496315	0	100	4.12	0	100	2.693	**1.53**
1000_25_10	1173792	1173792	0	100	348.17	0	100	32.684	**10.65**
1000_50_1	5663590	5663590	0	100	159.79	0	100	15.852	**10.08**
1000_50_2	180831	180831	0	100	102.38	0	100	9.837	**10.41**
1000_50_3	11384283	11384283	0	100	94.84	0	100	6.966	**13.62**
1000_50_4	322226	322226	0	100	145.05	0	100	11.512	**12.60**
1000_50_5	9984247	9984247	0.000856	3	387.02	0.000849	6	16.492	**23.47**
1000_50_6	4106261	4106261	0	100	218.19	0	100	26.888	**8.11**
1000_50_7	10498370	10498370	0	100	46.27	0	100	6.197	**7.47**
1000_50_8	4981146	4981146	0.005167	1	1256.97	0.004656	1	101.504	**12.38**
1000_50_9	1727861	1727861	0	100	3.63	0	100	2.561	**1.42**
1000_50_10	2340724	2340724	0	100	458.21	0	100	38.948	**11.76**
1000_75_1	11570056	11570056	9.46E−05	72	774.56	0.00012	76	57.801	**13.40**
1000_75_2	1901389	1901389	0	100	343.47	0	100	35.460	**9.69**
1000_75_3	2096485	2096485	0	100	410.67	0	100	33.272	**12.34**
1000_75_4	7305321	7305321	0	100	256.12	0	100	19.986	**12.81**
1000_75_5	13970240	13970240	0.002566	10	753.26	0.002346	16	64.035	**11.76**
1000_75_6	12288738	12288738	0	100	73.70	0	100	10.512	**7.01**
1000_75_7	1095837	1095837	0	100	252.09	0	100	20.654	**12.21**
1000_75_8	5575813	5575813	0	100	739.34	0	100	65.745	**11.25**
1000_75_9	695774	695774	0	100	121.20	0	100	12.471	**9.72**
1000_75_10	2507677	2507677	0	100	3.38	0	100	2.586	**1.31**
1000_100_1	6243494	6243494	0.000377	64	1094.06	0.000283	68	96.208	**11.37**
1000_100_2	4854086	4854086	9.06E−06	99	702.72	1.81E−05	96	80.487	**8.73**
1000_100_3	3172022	3172022	0	100	279.49	0	100	31.601	**8.84**
1000_100_4	754727	754727	0	100	91.81	0	100	13.756	**6.67**
1000_100_5	18646620	18646620	0.00029	33	485.40	0.000297	35	44.278	**10.96**
1000_100_6	16018298	**16020232**	0	100	96.02	0	100	11.393	**8.43**
1000_100_7	12936205	12936205	0	100	141.42	0	100	15.528	**9.11**
1000_100_8	6927738	6927738	0.00038	60	927.10	0.000366	68	84.706	**10.94**
1000_100_9	3874959	3874959	0	100	3.03	0	100	2.563	**1.18**
1000_100_10	1334494	1334494	0	100	389.75	0	100	36.037	**10.82**
2000_25_1	5268188	5268188	0	100	3546.56	0	100	451.067	**7.86**
2000_25_2	13294030	13294030	0	100	1201.36	0	100	186.651	**6.44**
2000_25_3	5500433	5500433	0	100	4431.71	1.09E−06	100	509.844	**8.69**
2000_25_4	14625118	14625118	0	100	13.282	0	100	13.286	**1.00**
2000_25_5	5975751	5975751	0	100	21.041	0	100	21.035	**1.00**
2000_25_6	4491691	4491691	0	100	2821.36	0	100	317.178	**8.90**
2000_25_7	6388756	6388756	0	100	41.13	0	100	20.841	**1.97**
2000_25_8	11769873	11769873	0	100	16.235	0	100	16.236	**1.00**
2000_25_9	10960328	10960328	0	100	1454.65	0	100	187.348	**7.76**
2000_25_10	139236	139236	0	100	24.982	0	100	24.982	**1.00**
2000_50_1	7070736	7070736	0	100	4454.17	0	98	470.694	**9.46**
2000_50_2	12587545	12587545	4.13E−06	98	3731.55	0	98	501.619	**7.44**
2000_50_3	27268336	27268336	0	100	14.263	0	100	14.263	**1.00**
2000_50_4	17754434	17754434	1.97E−05	50	3488.15	1.97E−05	50	396.971	**8.79**
2000_50_5	**16805490**	16805433	0.003103	0	6220.76	0.003108	0	589.315	**10.56**
2000_50_6	23076155	23076155	1.46E−05	74	2076.65	1.8E−05	70	243.050	**8.54**
2000_50_7	28759759	28759759	0.00668	1	2070.20	0.006818	1	201.994	**10.25**
2000_50_8	1580242	1580242	0	100	826.81	0	100	116.759	**7.08**
2000_50_9	26523791	26523791	7.54E−05	88	2149.12	7.48E−05	94	232.279	**9.25**
2000_50_10	24747047	24747047	0	100	1203.69	0	100	169.517	**7.10**
2000_75_1	25121998	25121998	0.000175	68	5570.12	0.000198	52	571.806	**9.74**
2000_75_2	12664670	12664670	0	100	3186.01	0	100	341.354	**9.33**
2000_75_3	43943994	43943994	0	100	1276.77	0	100	128.524	**9.93**
2000_75_4	37496613	37496613	1.04E−06	97	2199.43	1.39E−06	98	234.359	**9.38**
2000_75_5	24834948	24834948	0	100	3655.59	0	100	395.601	**9.24**
2000_75_6	45137758	45137758	0	100	12.759	0	100	12.759	**1.00**
2000_75_7	25502608	25502608	0	100	3432.80	0	100	340.182	**10.09**
2000_75_8	10067892	10067892	0	100	2714.10	0	100	291.290	**9.32**
2000_75_9	14171994	14171994	0.000256	63	6288.70	0.000255	68	646.902	**9.72**
2000_75_10	7815755	7815755	2.24E−05	98	3860.89	8.29E−05	94	394.449	**9.79**
2000_100_1	37929909	37929909	0	100	3075.48	0	100	321.090	**9.58**
2000_100_2	33647322	**33648051**	0.000127	65	6483.84	0.000184	66	607.696	**10.67**
2000_100_3	29952019	29952019	0.000225	30	5010.64	0.000256	26	529.829	**9.46**

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

Table 3 (*continued*)

n_d_i	Best Known	Best achieved	IQIEA			IQIEA-P			Speed Up
			RPD	SR	*t*(s)	RPD	SR	*t*(s)	
2000_100_4	26949268	26949268	4.45E−07	94	5375.68	4.45E−07	98	602.483	**8.92**
2000_100_5	22041715	22041715	0.000971	3	7930.55	0.000957	6	861.113	**9.21**
2000_100_6	18868887	18868887	5.83E−05	29	6770.63	5.05E−05	20	715.625	**9.46**
2000_100_7	15850597	15850597	0.00025	10	5809.62	0.000249	18	650.508	**8.93**
2000_100_8	13628967	13628967	0	100	2870.46	0	100	292.190	**9.82**
2000_100_9	8394562	8394562	0	100	4017.08	0	100	369.983	**10.86**
2000_100_10	4923559	4923559	5.42E_05	97	2012.40	3.62E−05	96	193.325	**10.41**
Average			**0.000272**	**85.09**	**1670.70**	**0.000266**	**85.24**	**178.77**	**9.346**

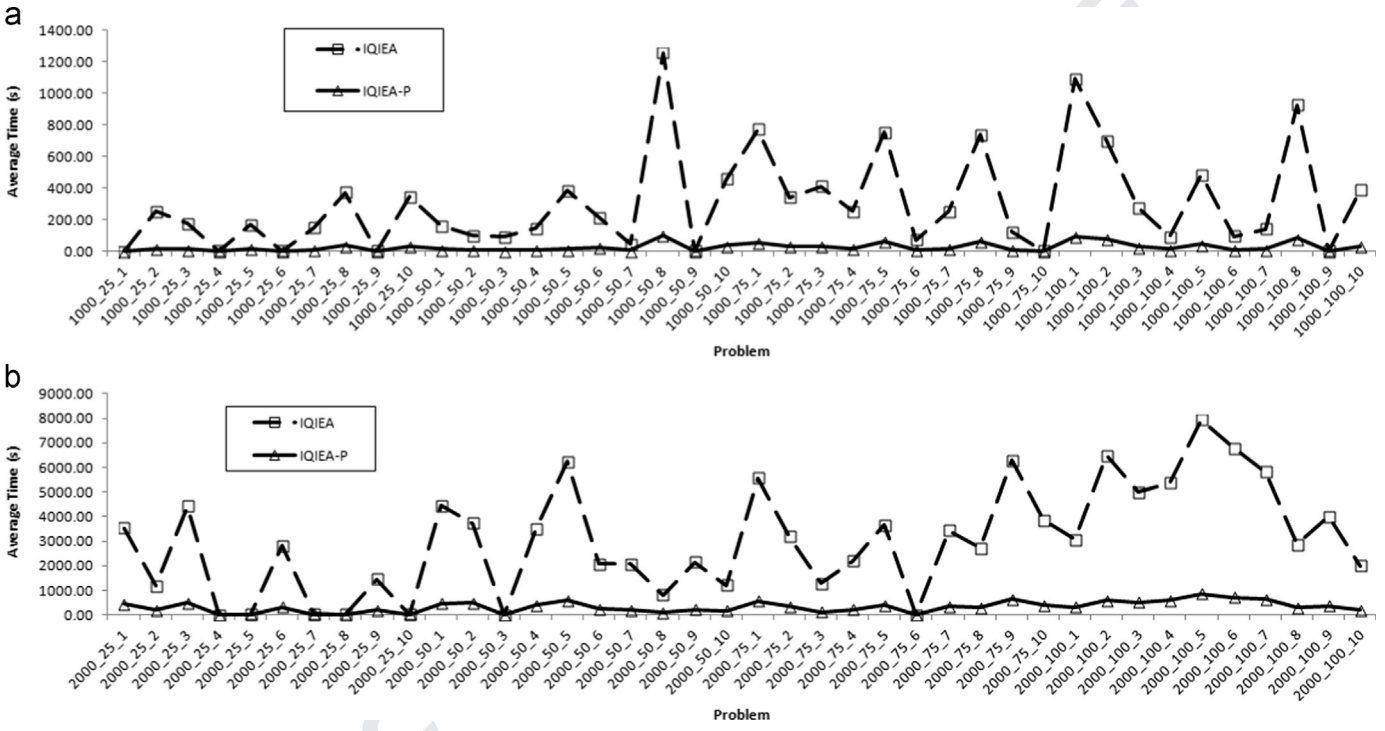

Fig. 9. Comparing time taken by IQIEA, IQIEA-P and GRASP Tabu Search for instances having (a) 1000 variables (b) 2000 variables.

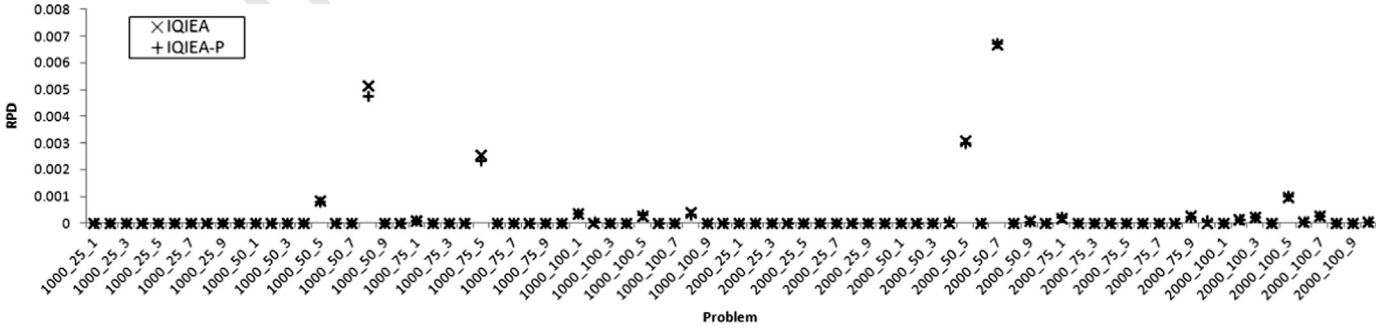

Fig. 10. Comparing Quality of solutions obtained from IQIEA and IQIEA-P in terms of RPD.

The novel features introduced in IQIEA-P as compared to prior attempts made on improving QIEA are summarized as follows.

- It uses a novel technique to initialize the qubit individuals. The qubit individuals which represent an estimation of distribution model have been initialized using the knowledge of preference of items to be selected in a solution based on a heuristic/s. These qubit individuals are then expected to generate solutions in the regions around the heuristic solution/s. This has resulted in to better solutions searched in less time in QIEAs.
- It has a different structure having benefits as described in Section 3.2.
- The various operators like local search, re-initialization, stochastic purge and mutation has been carefully knitted together

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

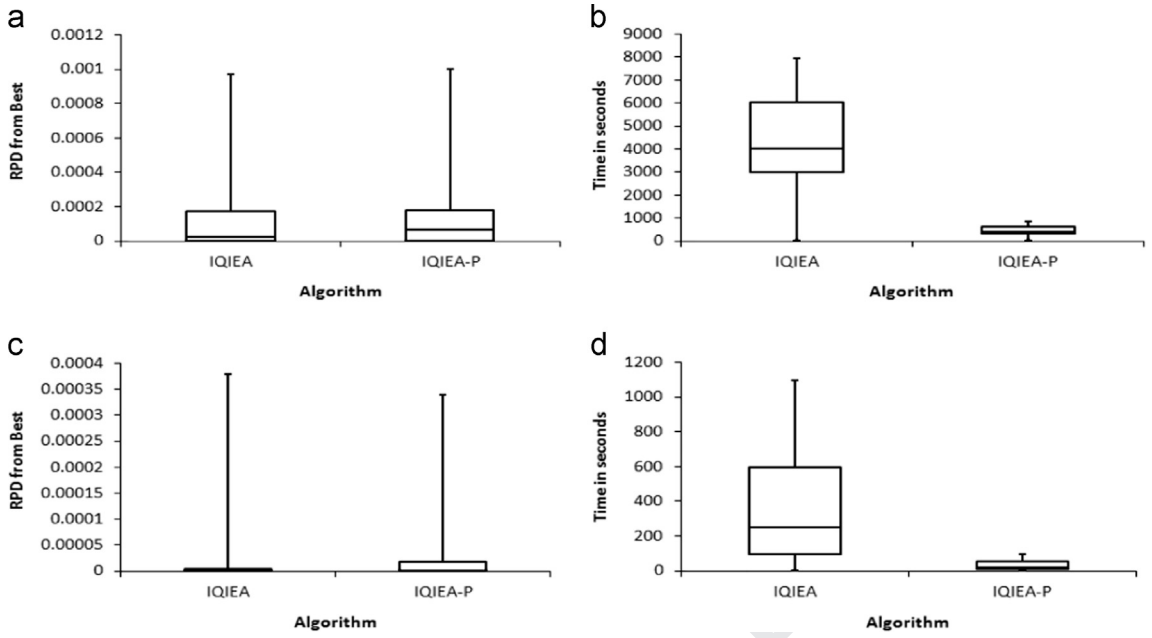

Fig. 11. Comparison of IQIEA and IQIEA-P with the help of box-plots drawn for time taken and RPD observed using different algorithms. (a) Box-plots for time(s) taken to solve instances of size 1000. (b) Box-plots for time(s) taken to solve instances of size 2000. (c) Box-plots for RPD to solve instances of size 1000. (d) Box-plots for RPD to solve instances of size 2000.

Table 4
Comparing Performance of IQIEA-P on selected instances using different number of threads while execution on basis of RPD of obtained solutions and time taken.

Problem ↓	RPD			Time (s)		
Threads →	5	12	24	5	12	24
1000_25_2	0	0	0	51.223	28.478	52.570
1000_50_5	0.000865	0.000677	0.000808	56.269	24.479	13.580
1000_75_1	0.000196	9.6E−07	3.75E−05	162.452	92.111	51.756
1000_100_8	0.00043	0.00043	0.000215	240.520	162.166	112.073

with primary steps of QIEA in order to help the search to exploit and explore the search space in a more balanced manner.
- The improvements are such that more independent sub tasks can be identified to be executed on parallel architecture.

4. Results and discussion

The computational experiments are performed on Dell HPCC having one DELL PowerEdge R710 Rack Server as master blade and 24 DELL PowerEdge M610 Blade Servers as compute nodes. Each blade uses Intel®Xeon® processor (5500 and 5600 series) @2.67 Ghz. The IQIEA-P was executed on the cluster through 24 threads spawned at a time.

Max Iterations is set to 60,η_1 and η_2 are set to 5 and population size is set to 320. The instances are referred to as n_d_i where n means the size of problem, d means the density of profit matrix and i mean seed value ranging from 1 to 10.

While reporting the results the following notations are used. The column best known lists the best value known from the current state of the art algorithm [13]. The column best achieved shows the best value achieved using IQIEA or IQIEAP (since the best solution provided by both of them is of same quality). If value obtained in a run for an instance is v and best achieved value for the instance is v^B then RPD (average of relative percentage deviation of v from best value over total number of runs) is calculated as $\text{Average}_{\text{totalruns}}((v^B -)v*100/v^B)$.

Performance of IQIEA-P is compared with IQIEA on larger Instances of size 1000 and 2000 instances of [13]. Table 3 shows the comparison with obtained speedups. The quality of solutions obtained from IQIEA-P and IQIEA is similar, while IQIEA-P takes significantly lesser time to reach best solution. The comparison between IQIEA and IQIEA-P is presented on the basis of average RPD, value of best solution achieved; average time taken to compute the best solution per run out of 100 runs and SR, strike rate as the number of hits made to the best known value out of 100 runs. The average speed up in IQIEA-P over IQIEA for each problem is mentioned.

Some observations made out of comparison between results obtained by IQIEA-P and IQIEA are as follows.

- The overall average time observed in IQIEA to execute one run of all 80 instances is 1670.30 while it is around 178seconds in IQIEA-P, giving an average speedup of 9.4.
- Maximum speedup obtained for an instance is 23.4 when 24 threads are used on a population of size 320. This implies that the algorithm is capable of providing maximum speedup if appropriate numbers of processors are available and larger populations are required to solve the problem.
- Instances for which the speedup is small are the ones where the best solution is obtained with execution of the parallel portion for small number of iterations.
- Quality of solutions obtained using sequential and parallel version is similar. Though comparing average of RPD and SR shows that parallelisation leads to improvement in quality of solutions. The overall quality of solutions is better than the current state of the art. The best solution obtained from IQIEA and IQIEA-P is better in two and worse in one case as compared to best solution provided by the state of the art technique [13] for the benchmark problems. For all other problems IQIEA and IQIEA-P provided similar solutions as the current state of the art.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

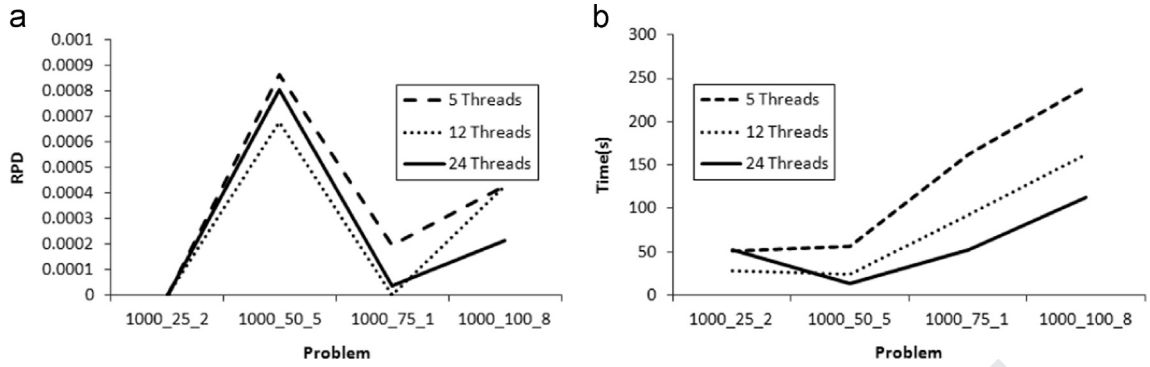

Fig. 12. Comparing Performance of IQIEA-P on selected instances using different number of threads while execution. (a) RPD obtained in IQIEA-P when executed with different number of threads. (b) Time taken by IQIEA-P when executed with different number of threads.

Table 5
Impact on Value of RPD obtained from various versions of IQIEA-P formed by excluding only one of the features proposed using selected instances.

IQIEA-P Without → Problem ↓	Initial fast exploitation	Attractor initialization	Local search	Mutation	Modification in structure	Purges	Q-bit individual re-initialization	Sort orders	IQIEA-P
1000_25_2	0.0003866	0.0156562	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0034792	**0.0000000**
1000_50_5	0.0008502	0.0012186	0.0008502	0.0009215	0.0007857	0.0007323	0.0008925	0.0045961	**0.0008079**
1000_75_1	0.0001604	0.0005954	0.0000346	0.0001594	0.0002314	0.0000845	0.0001844	0.0058964	**0.0000375**
1000_100_8	0.0006448	0.0004298	0.0005373	0.0001075	0.0000000	0.0004298	0.0004298	0.0339264	**0.0002149**

- The results show that as the size and complexity of problem increases it can be dealt by the parallel implementation using appropriate size of population and number of processors.

Fig. 9 plots a comparison of average time taken in seconds by IQIEA and IQIEA-P. The high speed up obtained in parallel over sequential version of IQIEA is explicitly visible in all instances from plots of Fig. 9. The speedup is higher when time taken by sequential version is high confirming the utility of parallelization.

Fig. 10 plots a comparison of quality of solutions obtained in terms of RPD. According to Fig. 10 quality of solutions obtained using IQIEA and IQIEA-P is similar. Fig. 11 shows the box plots for time taken and RPD observed using different algorithms under consideration for instances of size 1000 and 2000 separately.

An analysis of the impact of changing the number of threads on the performance of IQIEA-P has been studied. Few selected problems viz., 1000_25_2, 1000_50_2, 1000_75_1 and 1000_100_8 has been chosen for this study. For this study the observation is done only for 25 runs on each problem and a comparison is done between three implementations with number of threads as 5, 12 and 24. Table 4 shows the average values of RPD and Time taken in seconds by these implementations over 25 runs. The graphs have also been plotted for both the RPD and Time taken by different implementations in Fig. 12. The observations made from the results are as follows.

- The quality of solutions obtained in implementations using 5 threads is considerably lower than using 12 and 24 threads. Quality of solutions obtained, using 12 threads and 24 threads, is competitive.
- The computational effort, in terms of time taken to compute the solutions, required in implementation which uses 24 threads is significantly lesser than other implementations which use 5 and 12 threads.

Further study has been done to analyze the impact of each feature that has been incorporated in original QIEA of Han and Kim to design the proposed IQIEA-P. Same four problems viz., 1000_25_2, 1000_50_2, 1000_75_1 and 1000_100_8 has been selected for this study too. The different features that have been proposed in IQIEA-P and considered in this study are as follows.

- Initial fast exploitation
- Attractor initialization
- Local search
- Mutation
- Modification in structure
- Purges
- Q-bit Individual re-initialization
- Sort orders

Twenty five different runs of each of the nine different versions are performed on four problems mentioned above for this study. The nine versions include the proposed IQIEA-P and eight others such that each one of them is formed by removing only one feature from IQIEA-P. The comparison between nine different versions has been studied on the basis of impact observed on four parameters of performance viz., RPD and StdDev for quality of solutions and FES and Time taken for computational effort required to compute the solutions.

In Table 5, the values of RPD observed during execution of different versions are shown. The line graphs have been plotted in Fig. 13. The values in Table 5 clearly shows that if any of the feature of using sort orders or the feature of initializing the global attractor is removed from IQIEA-P, the resulting algorithm performs very badly in terms of RPD observed. Since the impact due to these two features outperform the impact due to other features, considering them while plotting graphs makes it difficult to compare the impact between other features. Hence, these two features have not been considered while plotting the graph in Fig. 13 so that the comparison of impact due to other features can be shown effectively.

In Table 6, the values of StdDev observed during execution of different versions are shown. The line graphs have been plotted in

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

Fig. 14. The values in Table 6 clearly shows that if any of the feature of using sort orders or the feature of initializing the global attractor is removed from IQIEA-P, the resulting algorithm performs very badly in terms of StdDev observed. Hence, these features has not been considered while plotting the graph in Fig. 14, so that the comparison of impact due to other features can be shown effectively.

The observations from Tables 5 and 6 and Figs. 13 and 14 clarify some important points. These help to do a comparative study on impacts the incorporation of different features in IQIEA-P have on the quality of solutions obtained. These observations are listed as follows.

- Making use of sort orders in controlling the search process through different ways have the highest impact on quality of solutions obtained.
- Initialization of attractor is next and similarly important in finding good solutions.
- All other features have a competitive impact on the quality of solution obtained using the proposed IQIEA-P. No one of them is observed to outperform others for all considered cases.
- It is clear from graphs in Figs. 14 and 15 that IQIEA-P is impacted negatively due to removing any of the feature in general apart from a few exceptions.
- So it shows that the features incorporated in the IQIEA-P balances each other and hence resulting comprehensive IQIEA-P provides better results generally.

In Table 7, the average values of FES taken by different versions to calculate the solutions are shown. The line graphs have been plotted in Fig. 15. Table 8 shows the average values for Time taken to compute the solution in seconds. The line graphs have also been

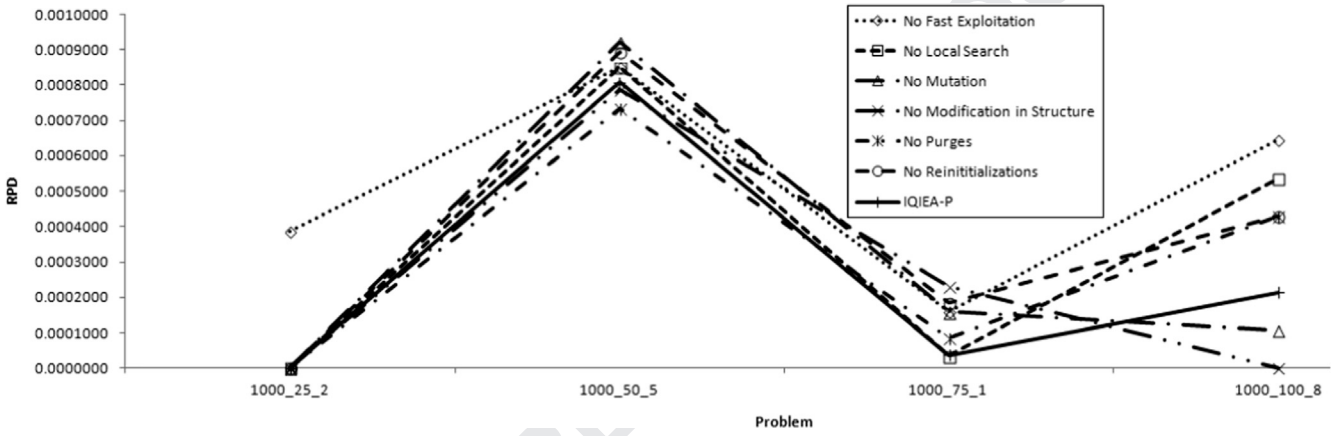

Fig. 13. Comparing Impact of removing some selected features, on Quality of Solutions obtained, from IQIEA-P. Different Line Graphs are plotted for the obtained values of RPD in IQIEAP and several versions of IQIEA-P each without only one of the feature as mentioned in Legend. The features of Sort Orders and Initialization of the Attractor have not been considered in this graph as they clearly have very high impact.

Table 6
Impact on Value of StdDev obtained from various versions of IQIEA-P formed by excluding only one of the features proposed using selected instances.

IQIEA-P Without → Problem ↓	Initial fast exploitation	Attractor initialization	Local search	Mutation	Modification in structure	Purges	Q-bit individual re-initialization	Sort orders	IQIEA-P
1000_25_2	2.67	22.11	0	0	0	0	0	0	0
1000_50_5	21.33	40.94	21.33	0	43.09	34.57	40.80	35.24	31.05
1000_75_1	55.29	69.88	11.63	55.33	54.51	29.33	47.42	368.08	12.63
1000_100_8	33.50	35.31	35.31	22.33	0	35.31	35.31	312.86	29.54

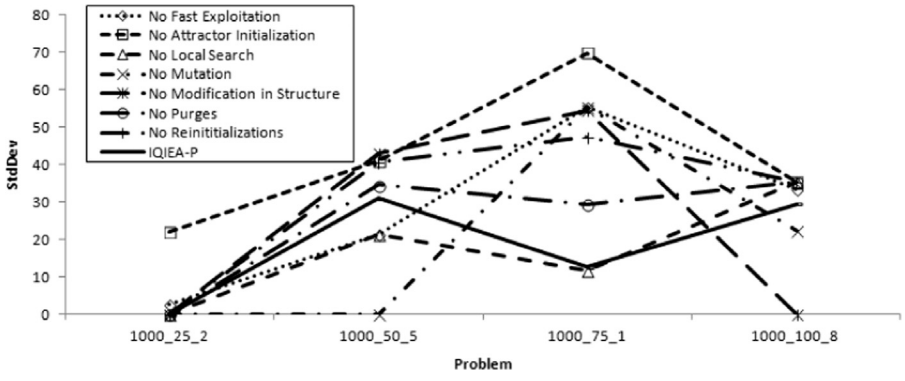

Fig. 14. Comparing Impact of removing some selected features, on Quality of Solutions obtained, from IQIEA-P. Different Line Graphs are plotted for the obtained values of StdDev in IQIEAP and several versions of IQIEA-P each without only one of the feature as mentioned in Legend. The features of Sort Orders and Initialization of the Attractor have not been considered in this graph as they clearly have very high impact.

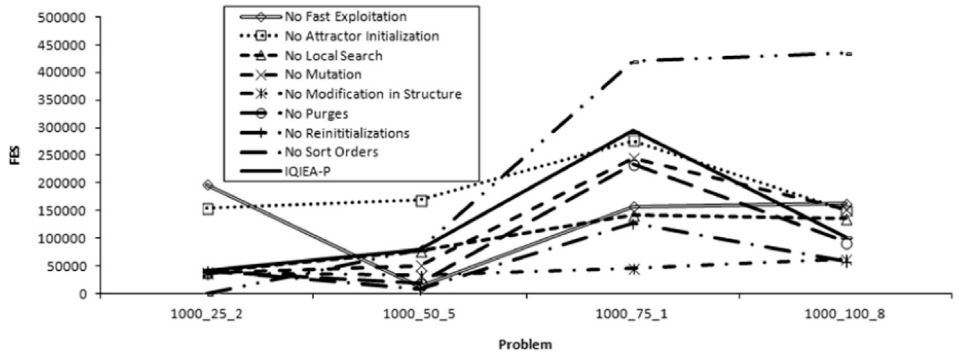

Fig. 15. Comparing Impact of removing some selected features, on computational effort, from IQIEA-P. Different Line Graphs are plotted for the values of FES required in IQIEAP and several versions of IQIEA-P each without only one of the feature as mentioned in legend.

Table 7
Impact on value of FES taken by various versions of IQIEA-P formed by excluding only one of the features proposed using selected instances.

IQIEA-P Without → Problem ↓	Initial fast exploitation	Attractor initialization	Local search	Mutation	Modification in structure	Purges	Q-bit individual re-initialization	Sort orders	IQIEA-P
1000_25_2	197396.6	154836.7	39774.33	38311	37821	38564.33	40914.33	1	42421
1000_50_5	11331	169311.2	77681	50784	33056.44	18447.22	8803.667	80321	79318.33
1000_75_1	157781	276886.7	141167.7	245457.7	46101.11	233851	127107.7	420154.3	293991
1000_100_8	162187.7	150576.7	135101	151851	61734.33	92934.33	57337.67	435924.3	102671

Table 8
Impact on Time taken by various versions of IQIEA-P formed by excluding only one of the features proposed using selected instances.

IQIEA-P Without → Problem ↓	Initial fast exploitation	Attractor initialization	Local search	Mutation	Modification in structure	Purges	Q-bit individual re-initialization	Sort orders	IQIEA-P
1000_25_2	52.57044	43.17558	18.93942	20.99801	21.4534	20.46704	20.6706	2.835458	52.57044
1000_50_5	13.57999	46.98485	22.79089	21.17384	16.71649	13.85619	13.79521	55.434	13.57999
1000_75_1	51.75579	70.16949	46.21626	73.76166	24.83407	63.77632	39.61631	196.7863	51.75579
1000_100_8	112.0733	90.08667	76.69523	91.93855	71.343	83.94629	72.24878	207.4989	112.0733

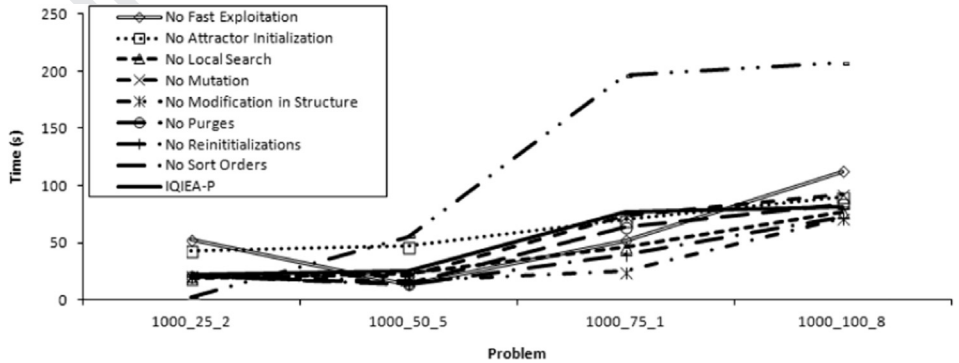

Fig. 16. Comparing Impact of removing some selected features, on computation effort, from IQIEA-P. Different Line Graphs are plotted for the values of time taken in s by IQIEAP and several versions of IQIEA-P each without only one of the feature as mentioned in Legend.

drawn to in Fig. 16 for the comparison between the impact different features have on IQIEA-F. Following observations are made.

- The removal of features heuristic based features, like making use of sort orders and initialization of attractor, has very high impact on the computation effort required by the algorithms in terms of both the time taken to compute and FES done during evolution for it. So it shows these features not contribute to improve the quality in proposed IQIEA-P but also to reduce the computational effort required for that.
- All other features have a random effect on reduction of the computational effort required by the IQIEA-P. But as observed

Table 9
Analyzing the overall impact of each feature on the performance of IQEA-P while solving four problems i.e. 1000_25_2, 1000_50_2, 1000_75_1 and 1000_100_8 (numbered 1–4 resp.) through the values of four parameters viz., RPD (R), StdDev(S), FES(F) and Time (T). A + sign indicate the feature has positive impact, A–sign indicate that the feature has negative impact while the sign = indicate that the feature has no impact on the performance of IQEA-P based on the observations made in above experiments.

	Initial Fast Exploitation				Attractor Initialization				Local Search				Mutation				Modif. in Structure				Purges				Q-bit Indi. Re-initialize				Sort Orders			
	R	S	F	T	R	S	F	T	R	S	F	T	R	S	F	T	R	S	F	T	R	S	F	T	R	S	F	T	R	S	F	T
1	+	+	+	+	+	+	+	+	=	=	−	−	=	=	−	−	=	=	−	−	=	=	−	−	=	=	−	−	+	=	+	−
2	+	−	−	−	+	+	+	+	+	−	−	−	+	−	−	−	−	+	−	−	+	+	−	−	+	+	−	−	+	+	+	+
3	+	+	−	−	+	+	−	+	+	−	−	−	+	+	−	−	+	+	−	−	-	+	−	−	+	+	−	−	+	+	+	+
4	+	+	+	+	+	+	+	+	+	+	+	+	−	−	+	+	−	−	−	+	+	+	−	+	+	+	−	−	+	+	+	+

earlier they have a considerable on improving the quality of solutions.

Table 9 shows the overall picture of this analysis. To use the space more efficiently in this table each of the parameter has been further abbreviated as follows: RPD is named as R, StdDev as S, FES as F, and Time is named as T. The four problems viz., 1000_25_2, 1000_50_2, 1000_75_1 and 1000_100_8 has also been referred using numbers from 1 to 4. If removing a feature the value of any parameter viz RPD, StdDev, FES or Time reduces then it is considered to have a negative impact or in other words it is considered to contribute negatively for the improvement of IQIEA-P. However, if any of these values is observed to increase than it is considered to have positive impact. If no change is observed than it is considered to have no impact. Following final observations are made from the comprehensive analysis.

- As expected the features of using Sort Orders and Initialization of Attractor have highest contribution in the improvement of performance of IQIEA-P in all aspects.
- Almost all other features have positive contribution in improvement of the quality of solutions obtained. This improvement may be either in RPD which measures closeness of solutions to best known solution or StdDev which measures the consistency of obtaining same solution in different runs. There are a few exceptions (marked in bold) for this where a feature contributed badly in terms both of these parameters while solving an instance.

The Evolutionary algorithm is a population based search technique considered slow in terms of time they take to converge due to large population size and large number of function evaluations required. Moreover the problem of getting stuck in local optima has to be handled carefully without increasing much of the computational effort to obtain effective performance from EAs. QIEAs also require higher time due to large number of observations and rotations required apart from issues common to EAs.

On the other hand the EA's use simple operations which can be applied directly or with little modification to other similar or not so similar problems, hence an EA (or QIEA in particular) with generalized frame work capable to provide effective and competitive performance is appreciated.

So the objective accomplished here is that a QIEA, the population based search technique generally considered slow, is improved through the help of parallelization with significant speedup. The presented algorithm has additional advantage that it is scalable. Bigger and more complex problems can still be solved in reasonable time by having more threads in parallel.

5. Conclusions and future work

The NP-hard optimization problem QKP is difficult in the sense that no pseudo-polynomial time algorithm exists to solve it. Very few attempts have been made in literature to solve large QKP instances. This work presents an improved parallel QIEA, IQIEA-P. The structure and features are applied to improve its capability to exploit and explore the solution space as well as to enhance its capability to be parallelised. These ideas to improve QIEA are applicable to solve other similar or not so similar problems.

The comparison of performance of IQIEA-P with its sequential version, IQIEA, shows that the proposed parallelisation in IQIEA-P provides high speedup. Hence, IQIEA-P can be used to tackle problem within reasonable time by increasing the number of individuals and hence the number of parallel threads at software level and providing sufficient cores at hardware level.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005

The results presented shows that the proposed algorithm IQIEA-P provide solutions competitive to state of the art approach for large QKP benchmark instances (size 1000 and 2000 binary variables).

Implementing other models for parallelization of IQIEA with its modified structure which allow larger population sizes keeping reasonable time limit may be taken as future work to obtain results for larger instances and/or with improved quality.

Uncited reference

[40].

Acknowledgments

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] Alberto Caprara, David Pisinger, Paolo Toth, Exact solution of the Quadratic Knapsack Problem, INFORMS J. Comput. vol. 11 (no. 2) (1999) 125–137.
[3] David Pisinger, The quadratic knapsack problem-a survey, Discret. 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 Internal Report, Technical Report, 1975.
[6] E. Johnson, A. Mehrotra, G. Nemhauser, Min-cut clustering, Meth. 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 , no.
[9] G. Dijkhuijen, U. Faigle, A cutting-plane approach to he edge-weighted maximal clique problem, Eur. J. Oper. Res. 69 (1993) 121–130.
[10] K. Park, K. Lee, S. Park, An extended formulation approach to the edge weighted maximal cliqur problem, Eur. J. Oper. Res. 95 (1996) 671–682.
[11] W. David Pisinger, Anders Bo Ramussen, Rune Sandvik, Solution of large Quadratic Knapsack Problems through aggressive reduction, INFORMS J. Comput. 19 (2) (2007) 280–290.
[12] L. Létocart, A. Nagih, G. Plateau, Reoptimization in Lagrangian methods for the 0-1 quadratic knapsack problem, Comput. Oper. Res. 39 (2012) 12–18.
[13] Z. Yang, G. Wang, F. Chu, An effective GRASP and tabu search for the 0-1 quadratic knapsack problem, Comput. Oper. Res. 40 (2013) 1176–1185.
[14] D.E. Goldberg, Genetic Algorithms in Search, Optimization And Machine Learning, Addison-Weley Longman Publishing Co, Boston, MA, USA, 1989.
[15] P.,S. Oliveto, J. He, X. Yao, Time complexity of evolutionary algorithms for combinatorial optimization: a decade of result, Int. J. Autom. Comput. 4 (3) (2007) 281–293.
[16] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Trans. Evolut. Comput. 6 (5) (2002) 443–462.
[17] K. Han, K. Park, C. Lee, J. Kim, Parallel quantum-inspired genetic algorithm for combinatorial optimization prblem, in: Proceedings CEC, vol. 2, 2001, pp. 1422–1429.
[18] R. Nowotniak, J. Kucharski, GPU-based tuning of quantum-inspired genetic algorithm for a combinatorial optimization problem, Bull. Pol. Acad. Sci.: Tech. Sci. 60 (2) (2012) 323–330.
[19] P.L. Hammer, D.J. Rader, Efficient methods for solving quadratic 0-1 knapsack problem, INFOR vol. 35 (1997) 179–182.
[20] B.A. Julstrom, Greedy, genetic and greedy genetic algorithms for the quadratic knapsack problem, in: Genetic and evolutionary computation conference, Washingtom DC, USA, 2005, pp. 607–614.
[21] X. Xie, J. Liu, A Mini-Swarm for the quadratic knapsack problem, in: IEEE Swarm Intelligence Symposium, Honolulu, USA, 2007, pp. 190–197.
[22] S. Pulikanti, A. Singh, An Artificial Bee Colony Algorithm for the Quadratic Knapsack Problem, in: ICONIP 2009, Part II, LNCS 5864, 2009, pp. 196–205.
[23] Md. Abul Kalam Azad, Maria A.C. Rocha, M.G.P. Fernandes, Solving 0-1 Quadratic Knapsack Problem with a Population-based Artificial Fish Swarm Algorithm, in: International Conference on Applied and Computational Mathematics, 2012.
[24] Md. Abdul Kalam Azad, Maria A.C. Rocha, Edite M.G.P. Fernandes, A simplified binary artificial fish swarm algorithm for 0-1 quadratic knapsack problems, J. Comput. Appl. Math. 259 (2014) 897–904 [Online] ⟨http://www.sciencedirect.com/science/article/pii/S0377042713005074⟩.
[25] 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.
[26] A. Billionet, E. Soutif, QKP Instances, 2004. [Online] ⟨http://cedric.cnam.fr/~soutif/QKP/QKP.html⟩.
[27] A. Billionet, E. Soutif, An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem, Eur. J. Oper. Res. 157 (2004) 565–575.
[28] Kuk-Hyun Han, Jong-Hwan Kim, Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, IEEE Trans. Evolut. Comput. 6 (6) (2002) 580–593.
[29] K. Han, J. Kim, On setting the parameters of quantum-inspired evolutionary algorithm for practical application, in: Proceedings CEC, 2003, pp. 178–184.
[30] K.-H. Han, On the Analysis of the Quantum-inspired Evolutionary Algorithm with a Single Individual, in: IEEE Congress on Evolutionary Computation, Vancouver, Canada, 2006.
[31] M.D. Platel, Stefan Schliebs, Nikola Kasabov, Quantum-inspired evolutionary algorithm: a multimodel EDA, IEEE Trans. Evolut. Comput. 13 (6) (2009) 1218–1232.
[32] Gexiang Zhang, Quantum-inspired evolutionary algorithms: a survey and empirical study, J. Heuristics 17 (3) (2011) 303–351.
[33] K. Han, J. Kim, Quantum-inspired evolutionary algorithms with a new termination criterion, h-epsilon gate, and two-phase scheme, IEEE Trans. Evolut. Comput. 8 (2) (2004) 156–169.
[34] M.D. Platel, S. Schliebs, N. Kasabov, A versatile quantum-inspired evolutionary algorithm, in: Proceedings CEC, 2007, pp. 423–430.
[35] Kliemann Lasse, Kliemann1 Ole, C. Patvardhan, Sauerland Volkmar, Srivastav Anand, A New QEA Computing Near-Optimal Low-Discrepancy Colorings in the Hypergraph of Arithmetic Progressions, in: SEA 2013, 2013, pp. 67–78.
[36] P. Moscato, On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms, California Instiute of Technology, Pasadena, CA, Technical Report 826, 1989.
[37] P. Moscato, C. Cotta, A. Mendes, Memetic algorithms, in: New Optimization Techniques in Engineering, Berlin Heidelberg, 2004, pp. 53–85.
[38] Y.S. Ong, M.H. Lim, X. Chen, Memetic computation: Past, present & future, in: IEEE Computational Intelligence Magazine, 5, 2, 2010, pp. 24–31.
[39] Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim, Kay Chen Tan, A multi-facet survey on memetic computation, IEEE Trans. Evolut. Comput. 15 (5) (2011) 591–607.
[40] P. Arpaia, D. Maisto, C. Manna, A quantum-inspired evolutionary algorithm with a competitive variation operator for multiple-fault diagnosis, Appl. Soft Comput. 11 (8) (2011) 4655–4666.

Please cite this article as: C. Patvardhan, et al., Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm and Evolutionary Computation (2015), http://dx.doi.org/10.1016/j.swevo.2015.09.005