Soft Comput
DOI 10.1007/s00500-013-1203-7

FOCUS

# A quantum-inspired Tabu search algorithm for solving combinatorial optimization problems

Hua-Pei Chiang · Yao-Hsin Chou · Chia-Hui Chiu · Shu-Yu Kuo · Yueh-Min Huang

**Abstract** In this study, we propose a novel quantum-inspired evolutionary algorithm (QEA), called quantum inspired Tabu search (QTS). QTS is based on the classical Tabu search and characteristics of quantum computation, such as superposition. The process of qubit measurement is a probability operation that increases diversification; a quantum rotation gate used to searching toward attractive regions will increase intensification. This paper will show how to implement QTS into NP-complete problems such as 0/1 knapsack problems, multiple knapsack problems and the traveling salesman problem. These problems are important to computer science, cryptography and network security. Furthermore, our experimental results on 0/1 knapsack problems are compared with those of other heuristic algorithms, such as a conventional genetic algorithm, a Tabu search algorithm and the original QEA. The final outcomes show that QTS performs much better than other heuristic algorithms without premature convergence and with more efficiency. Also on multiple knapsack problems and the traveling salesman problem QTS verify its effectiveness.

**Keywords** Quantum computing · Combinatorial optimization · Quantum-inspired evolutionary algorithm · Tabu search · Knapsack problem

Communicated by A. Castiglione.

H.-P. Chiang · Y.-M. Huang
Department of Engineering Science, National Cheng Kung University, Tainan, Taiwan, ROC

Y.-H. Chou (✉) · C.-H. Chiu · S.-Y. Kuo
Department of Computer Science and Information Engineering, National Chi Nan University, Puli, Taiwan, ROC
e-mail: yhchou@ncnu.edu.tw

## 1 Introduction

Quantum-inspired evolutionary algorithm (QEA) is a special case of the estimation of distribution algorithm (EDA), where quantum probability amplitudes are used with probabilistic models to describe the promising areas of decision space, and to guide the exploration of the global optimal. It differs from other EDAs in that it has more than one probability variable that can be used simultaneously, while also enhancing traditional evolutionary computing.

The most successful case of using quantum-inspired evolutionary computing was by Han and Kim (2002). In recent years, QEA can be used to deal effectively with some combinatorial optimization problems, such as the traveling salesman problem, the knapsack problem, the assignment problem, etc (Talbi et al. 2004; Wanga et al. 2007; Chou et al. 2010; Narayan and Patvardhan 2009; Li and Ling 2007; Chou 2012). Inspired by this, we proposed that a combination of QEA (Talbi et al. 2004; Wanga et al. 2007; Chou et al. 2010; Narayan and Patvardhan 2009; Li and Ling 2007; Merkle and Hellman 1978; Benny and Ronald 1988; Simon and Willi 2011; Han and Kim 2000, 2002, 2004; Li et al. 2009; Jiao et al. 2008; Li and Wang 2007) and traditional methods of Tabu search (TS), called quantum-inspired Tabu search (QTS), could significantly improve the performance of evolutionary algorithms. Our preliminary version of this work has been published in Chou et al. (2011). Indeed, this paper attempts to detail our proposed QTS towards a comprehensive solution and to demonstrate their advanced performance through further simulations.

TS was proposed and developed by Glover (1986, 1989, 1990), and has seen practical implementation in a wide variety of fields (Hu 1992; Chelouah and Siarry 2000). A distinguishing feature of TS is the use of the memory mecha-

Published online: 27 December 2013

nism, which allows for economical and effective searching of the solution space. In addition, TS combined with quantum computing characteristics can overcome the shortcomings of simple TS. The process of qubits measurement is a probability operation that increases diversification, and a quantum rotation gate is used to search toward attractive regions that increase intensification.

The 0/1 knapsack problems, multiple knapsack problems and the traveling salesman problem are problems in combinatorial optimization. Those problems are NP-complete thought to be computationally difficult to solve in general and often arise in studied fields such as computer science, complexity theory and combinatorics. In addition to those fields, those problems also apply in cryptography such as hiding information or key cryptosystem (Merkle and Hellman 1978; Benny and Ronald 1988; Simon and Willi 2011).

The rest of this paper is organized as follows. Section 2 describes the principle of quantum computing. Section 3 briefly introduces background information of Tabu search. Section 4 introduces QEA and QTS algorithm proposed in Sect. 5. In Sect. 6 we will show the experiment results and conclusions given in Sect. 7.

## 2 Quantum computing principles

The smallest information unit in today's digital computers is a single bit, which at any given time is either in state "1" or "0". The corresponding analogue on a quantum computer is represented by a quantum bit or *qubit*. Similar to classical bits, a qubit may be in the basis state "0" or "1", but may also be in any superposition of both states. Additionally, the act of measuring (or observing) a qubit will project the quantum system onto one of its basis states.

A quantum bit state $|\psi\rangle$ can be represented as (1):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \tag{1}$$

where $\alpha$ and $\beta$ are complex numbers, and $|\alpha|^2$ and $|\beta|^2$ represent the probability that the qubit will be found in the "0" state and "1" state, respectively. The sum of $|\alpha|^2$ and $|\beta|^2$ will follow the rule of probability. and can be represented as

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2}$$

If there are $n$ quantum bits in a quantum system, we can represent all of them in the following form:

$$\left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{array}\right]. \tag{3}$$

According to (2), for each $|\alpha|^2 + |\beta|^2 = 1, i = 1, 2, \ldots n$.

Table 1 Observation of a qubit string

| No. qubit | 1 | 2 | 3 | 4 | 5 | … | N |
|---|---|---|---|---|---|---|---|
| $R$ | 0.16 | 0.32 | 0.9 | 0.46 | 0.28 | … | 0.82 |
| $\lvert\beta\rvert^2$ | 0.5 | 0.75 | 0.3 | 0.81 | 0.27 | … | 0.62 |
| $x$ | 0 | 0 | 1 | 0 | 1 | … | 1 |

A key point when considering quantum systems is that they can compactly convey information on a large number of possible system states. In classical bit strings, a string of length $n$ can only represent one of the $2^n$ possible states. However, a quantum system of $n$ qubits can represent a total of $2^n$ dimensions. This means that even a short qubit can convey information on many possible system states. The representation of quantum bits as (3) more easily expresses any superposition of the states, and it is more suitable than other classified representations when applied to generating populations of evolutionary algorithms.

The process of generating binary strings from the qubit string $\beta$ is called measurement (observation). Let us have a binary solution $x$ by observing the states of $\beta$, where $x = x_1, x_2, \ldots, x_n$. At any generation in the binary string of length $n$, $x$ is formed by selecting either 0 or 1 for each bit by comparing the probability, either $|\alpha|^2$ or $|\beta|^2$, with $R$. In a quantum computer, the act of observing a quantum state collapses it to a basis state. However, this collapse into a single state does not occur in QTS since it operates on a classical computer, rather than a quantum computer. Besides, to observe (measure) a Q-bit string $\beta$, a string consisting of the same number of random numbers between 0 and 1 is generated, and the elements of string x are set to 0 if R is less than the square of $\beta$; Otherwise, $x$ is set to 1. For example, Table 1 shows the observation process.

## 3 Tabu search

TS is a meta-heuristic algorithm that can be used for solving combinatorial optimization problems. It is based on a greedy local search algorithm that explores the solution space beyond local optimality and adopts a memory structure which is an imitation of human behavior, and uses past experiences to improve decision making in daily life. Once a potential solution has been determined, it will be marked as "tabu", so the algorithm will not visit it repeatedly.

TS is an iterative procedure designed for optimizing (minimizing or maximizing) the objective function $f(x)$, which is subject to $x \in X$, where $X$ is the set of feasible solutions. The basis of TS may be described as: TS begins as a neighborhood search, which is similar to a hill climbing strategy, proceeding iteratively until a chosen termination criterion is satisfied, such as that no improving neighbor solutions can be found, or ends when it achieves the maximum iteration. Each solution must exist with associated neighborhood solutions $N(x) \subset X$, and the search progresses changes from a solution x to each current neighborhood solution $x' \in N(x)$ by an operation called a move.

To avoid being trapped in a local optimality, or cycling in the solution exploration, TS uses a special memory structure which records the solutions that have recently been visited. This short term memory mechanism prohibits certain moves, even if they are improvements, in the Tabu list $T$. No records in $T$ can be used to form a next feasible solution, unless they fit "Aspiration Criteria". The "Aspiration Criteria" (AC) allow better solutions to be chosen, even if they have been tabooed. The length of $T$ has no fixed standard as yet. Suppose $T$ follows the policy of FIFO; the larger the $T$ set, the longer will be the prohibited time of the move in $T$.

Simple TS is divided into four phases: first, initialize the problem; second, move; third, update; and fourth, test termination criterion. The algorithm's goal is to minimize $f(x)$, described below.

**Tabu Search Algorithm**

```
MAXITER : the maximum number of iterations
$x'$ ← produce an initial solution $x$
initialize tabu list T

1. for i = 1 to MAXITER do
2.   identify Neighborhood set $N$
3.   identify Candidate set $C = N - T + AC$
4.   find the best $x$ from $C$
5.   if $f(x) > f(x')$ then
6.     $x' \leftarrow x$
7.   end if
8.   update $T$ with FIFO policy
9. End for
```

One major drawback in simple TS is that, in most cases, it will find a local optimum rather than a global optimum, thus both intensification and diversification strategies should be considered for improving the robustness, effectiveness and efficiency of simple TS. Therefore, in order to improve the performance of the algorithm, we construct a more powerful neighborhood structure, which is a combination of quantum computing concepts and principles and simple TS.

## 4 Quantum-inspired evolutionary algorithm

QEA is a probabilistic and population-based algorithm, shown below.

**Quantum-inspired Evolutionary Algorithm**

```
1.  $t \leftarrow 0$
2.  initialize $Q(t)$
3.  make $P(t)$ by measure $Q(t)$
4.  repair $P(t)$
5.  evaluate $P(t)$ and store the best solutions into $B(t)$
6.  while (not termination-condition) do
7.    t ← t + 1
8.    make $P(t)$ by measure $Q(t-1)$
9.    repair $P(t)$
10.   evaluate $P(t)$ and $B(t-1)$
11.   update $Q(t)$
12.   store the best solutions among $B(t-1)$ and $P(t)$
      into $B(t)$
13.   store the best solution $b$ among $B(t)$
14. End while
```

Each population is produced by measuring a string of qubits as:

$$q = \left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{array}\right]$$

where n is the quantity of *qubits*, and satisfies the basic condition, $|\alpha_k|^2 + |\beta_k|^2 = 1, k = 1, 2, \ldots, n$ at $t$ iteration and $m$ population size, and can be expressed as $Q(t) = \{q_1^t, q_2^t, \ldots, q_m^t\}$.

In the beginning, at $t = 0$, $Q(0)$ are initialized with $\frac{1}{\sqrt{2}}$, which means the linear superposition of individual possible states with the same probability. In the iteration, we can get $P(t) = \{s_1, s_2, \ldots, s_m\}$, where $s^i = \{x_1^i, x_2^i, \ldots, x_n^i\}$, $i = 1, 2, \ldots, m$, by measuring $Q(t-1)$, each $x$ is set to either "0" or "1". We then evaluate the fitness function value of all the $s$ in $P(t)$ and $B(t-1)$ to store the better solutions into $B(t) = \{s_1, s_2, \ldots, s_m\}$, where there are also binary strings.

An important step in QEA is to *update* $Q$, which uses a rotation matrix to change the value of a *qubit*. The polar plot is shown in Fig. 1. The operation is represented as

$$\begin{bmatrix} \alpha' \\ \beta' \end{bmatrix} = \begin{bmatrix} cos(\triangle\,\theta) & -sin(\triangle\,\theta) \\ sin(\triangle\,\theta) & cos(\triangle\,\theta) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

where $\triangle\,\theta$ in the function is decided by their fitness value and two reference bits. For instance, the $k$th *qubit* in the $i$th solution $q_k^i \in Q$ has to update, the $\triangle\,\theta$ is set according to the fitness value of $s^i \in P$ and two bits, $x_k^i \in P$ and $b_k^i \in B$. The $Q$ update procedure allowd the solution to tend toward the best required solution. Finally, $b$ records a global best solution. Until the termination condition is satisfied, the procedure is looped in the *while*.

Update $Q(t)$ is employed by slight *rotation* through each iteration, making the search range diversified to begin, and then slowly more intensified. This is why we follow this method to combine slight *rotation* with Simple TS to

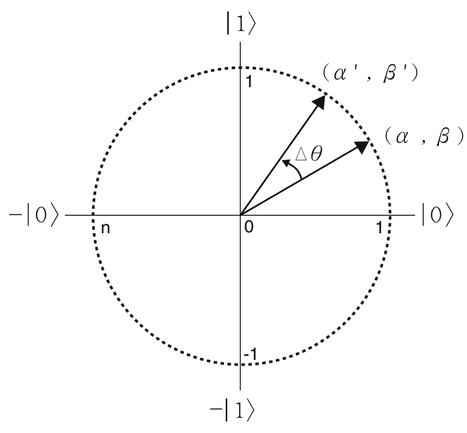

**Fig. 1** Polar plot of the rotation operation of a *qubit*

improve both the traditional TS and QEA. In our algorithm, we replace the $B$ of QEA with another means of updating quantum states. Fortunately, this combination not only prevents premature convergence, but also more quickly obtains the costly optimal solution.

## 5 Quantum-inspired Tabu search

The QTS evolutionary algorithm is based on TS. QTS includes two important added components: the diversification and the intensification strategies from QEA. For example, the diversification strategy looks for neighborhood solutions by measuring the diversity of the search towards unexplored regions, which appears random and in deterministic in the results, while the operation of the new move mechanism, called *move-gate*, uses a two-dimensional quantum rotation gate to intensify the search toward attractive regions more thoroughly. The algorithms are described below, and further explanations for solving the 0/1 knapsack problem are given in the next section.

**Quantum-inspired Tabu Search Algorithm (QTS)**

```
1.  $t \leftarrow 0$
2.  initialize $Q(t)$
3.  initialize the fitness $b$ of best solution $s^b$
4.  **while** (not termination-condition) **do**
5.     t ← t + 1
6.     produce neighborhood Set $N$ by multiple measure $Q(t-1)$
7.     repair $s \in N$ and evaluate $f(s)$
8.     select the best solution $s^b$ and the worst solution $s^w$ among $N$
9.     store the fitness of best solution into $b$
10.    update Tabu list $T$
11.    update $Q(t)$
12. **End while**
```

The detailed description of each step is described as follows.

1. $Q(0) = \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix}\begin{vmatrix} \alpha_2 \\ \beta_2 \end{vmatrix}\begin{matrix} \cdots \\ \cdots \end{matrix}\begin{vmatrix} \alpha_n \\ \beta_n \end{vmatrix}$, where $n$ is the quantity of *qubits*, all $\alpha$ and $\beta$ are initialized with $\frac{1}{\sqrt{2}}$, as in Han and Kim (2002). This represents that this *qubit* will have the same probability of collapsing into either a "0" or "1" state.
2. The best fitness $b$ is used to record the best solution achieved through the evolutionary algorithm. It is initialized as 0.
3. The remaining steps, involving lines 4 to 12 of Algorithm QTS are executed cyclically until the termination condition is satisfied.
4. Neighborhood solution $N$ (line 6 of Algorithm QTS) is produced by multiple measurements of $Q(i-1)$ $m$ times, and forms a binary matrix:
$$N = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_n^1 \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^m & x_2^m & \dots & x_n^m \end{bmatrix},$$
where each bit $x_k^j$, $k = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$ using the probability of $|\alpha_k|^2$ or $|\beta_k|^2$ to compare with a random number $R \in [0, 1]$, when $R > |\beta_k|^2$, set $x_k^j = 1$, otherwise $x_k^j = 0$. We may not get the same results from the same measurement performed repeatedly on the same *qubit*, but the smaller the gap between $|\alpha|$ and $|\beta|$ was, the more divergent neighborhood solutions we obtained.
5. Repair $s \in N$ so that it can conform to feasible domain, then evaluate its fitness value with the objective function $f(s)$ (line 7 of Algorithm QTS).
6. Of all the neighborhood solutions, choose the solution $s \in N$ which has best fitness value stored into $s^b$, and which has worst fitness value stored into $s^w$. These two solutions are to be used as reference bits for the *update* $Q(i)$ step (line 8 of Algorithm QTS).
7. Update $b$ by line 9 of Algorithm QTS. If current iteration has a fitness value better than $b$, $b$ will be replaced.
8. Update tabu list $T$ by line 10 of Algorithm QTS. In QTS, $T$ is used to taboo *qubits* prohibited from changing their original state, which differs from general common $T$, which is used to records recent moves. Therefore, tabu size is dynamic. The tabu conditions are shown in Table 2. Compare two binary strings $s^b$ and $s^w$. When the kth bits $s_k^b$ and $s_k^w$ are different, it means that *qubit* $Q_k$ was placed in $T$.
9. The $Q$ update process is described by lines 11 of Algorithm QTS. When *qubit* $q \in Q$ changes its original state, it should correspond to Table 2. When $q$ is not in $T$, it applies the *move-gate* $U(\triangle\ \theta)$ =

**Table 2** Move-gate lookup table

| $s_k^b$ | $s_k^w$ | $Q_k \in T$ | $\triangle\,\theta$ |
|---|---|---|---|
| $Q_k$ locates in first or third quadrant | | | |
| 0 | 0 | True | 0 |
| 0 | 1 | False | $-\theta$ |
| 1 | 0 | False | $+\theta$ |
| 1 | 1 | True | 0 |
| $Q_k$ locates in second or fourth quadrant | | | |
| 0 | 0 | True | 0 |
| 0 | 1 | False | $+\theta$ |
| 1 | 0 | False | $-\theta$ |
| 1 | 1 | True | 0 |

$k = 1, 2, \ldots n$, $n$ is number of $qubit$

$\begin{bmatrix} cos(\triangle\,\theta) & -sin(\triangle\,\theta) \\ sin(\triangle\,\theta) & cos(\triangle\,\theta) \end{bmatrix}$ to rotate to a new state, which may acquire better neighborhood solutions by measuring $q$ in the next iteration. The $\triangle\,\theta$ should be designed in compliance with the application problem, but is usually set to a small value to prevent premature convergence. Table 2 gives the value of $\triangle\,\theta$ that should be used in the $Q$ update procedure.

## 6 Experimental results

### 6.1 Solving the 0/1 knapsack problem

The knapsack problem is a kind of combinatorial optimization problem, and is an appropriate example to demonstrate the effectiveness of QTS algorithm. The knapsack problem can be described as: Given a set of $n$ items, each kind of item $k$ has a weight $w_k$ and a profit $p_k$. Select a subset of items so that the total weight is less than or equal to the capacity $C$ of the knapsack $\sum_{k=1}^{n} w_k x_k \leqslant C$, and obtain the maximum total profit by objective function $f(x) = \sum_{k=1}^{n} p_k x_k$. If the $k$th item is selected for the knapsack, $x_k = 1$, otherwise $x_k = 0$. The main parts of the pseudo-codes in QTS algorithm for the knapsack problem are listed as follows.

Suppose $n$ items can be chosen to put in the knapsack, each item represented by a $qubit$, expressed as

$$Q(0) = \left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \end{array}\right],$$

where all $\alpha$ and $\beta$ are initialized with $\frac{1}{\sqrt{2}}$. The best fitness value $b$ is initialized to 0. After the above initial procedures, the remaining steps enter an iterative loop until the maximum number of iterations is reached.

The **Measure**$(x)$ procedure can produce one solution $s = \{x_1, x_2, \ldots, x_n\}$, where all $x$ are either 1 or 0 to indicate which item is selected or not. In QTS, the neighborhood search method is used by repeatedlt measuring the $Q(t-1)$ $m$ times in order to obtain neighborhood set $N = \{s_1, s_2, \ldots, s_m\}$.

**Procedure Measure (x)**

```
 1. begin
 2.   for k=1 to n do
 3.     R ∈ random number U[0,1]
 4.     if $R < |\beta|^2$ then
 5.       $x_k \leftarrow 1$
 6.     else
 7.       $x_k \leftarrow 0$
 8.     End if
 9.   End for
10. End
```

Then, repairing all of the solutions in $N$, we use the same **Repair**$(s)$ procedure as in Han and Kim (2002). If the total weight exceeds the limit, it will repeatedly discard one item until total weight is less than the limit. On the other hand, it will also repeatedly include one item until the limit is matched. The profit by fitness function is then evaluated, while $f(s)$ is better than the current fitness value, $b$ will be updated.

Next, selecting the best and the worst profit of neighborhood solutions into $s^b$ and $s^w$ respectively:

$$s^b \leftarrow max\{f(s^j), \forall j\}$$
$$s^w \leftarrow min\{f(s^j), \forall j\}$$

The final step is update quantum individual $qubit$ by applying move-gate. According to Table 2, when $s_k^b$ and $s_k^w$ are the same, the $\Delta\theta$ will be set to 0. This means they cannot change their original state, and also indicates that the $k$th item was marked as taboo. However, the rest of the $qubits$ should execute the **Update Q**$(q_k)$. The procedure implements the same rotation matrix, but not in the same way as QEA.

**Procedure Update Q (k)**

```
1. begin
2.   Determine $\Delta\theta$ by Table 2
3.     $q_k = \begin{bmatrix} cos(\triangle\,\theta) & -sin(\triangle\,\theta) \\ sin(\triangle\,\theta) & cos(\triangle\,\theta) \end{bmatrix} \begin{bmatrix} \alpha_k \\ \beta_k \end{bmatrix}$
4. End
```

Assuming the $k$th item should be updated, the procedure with two reference bits $s_k^b$ and $s_k^w$ updates $q_k$. Referring to Fig. 1, it can be seen that $q_k$ will tend to the $s_k^b$ state. The aim is to force $qubit$ states to approach better regions, and to avoid bad regions. Furthermore, only $n$-$qubit$ needs to be updated in QTS, which differs from QEA which needs $n$ by $m$-$qubit$. We believe that these are the main reason why QTS is more effective and efficient than QEA, since our algorithm merely simulates quantum characteristics and implements them on a conventional computer. When a quantum state has been measured, its superposition state can be maintained, and supplied for the next iteration to use.

In our experiments, the knapsack capacity is set to $C = \frac{1}{\sqrt{2}} \sum_{k=1}^{n} w_k$. We conduct two experiments, CASE I and CASE II. In the first case of the 0/1 knapsack problem, we set $w_k \in [1, 10]$, and let $p_k = w_k + 5$, as in Han and Kim (2002). It can be seen from Table 3 that the heavier items always obtain less profit per unit weight, but in the real world, this is not always the case. For this reason we also perform an experiment to test case II: $p_k = w_k + l_k$, where $l_k \in [0, 5]$, as in Wanga et al. (2007).

**Procedure Repair (s)**

```
1.  begin
2.    $s = \{x_1, x_2, \cdots, x_n\}$
3.    Set the knapsack-overfilled $false$
4.    if $\sum_{k=1}^{n} w_k x_k > Capacity$ then
5.      set the knapsack-overfilled $true$
6.    End if
7.    while (knapsack-overfilled) do
8.      select random $j$th item from the knapsack
9.      $x_j \leftarrow 0$
10.     if $\sum_{k=1}^{n} w_k x_k \leq Capacity$ then
11.       set the knapsack-overfilled $false$
12.     End if
13.   End while
14.   while (not knapsack-overfilled) do
15.     select random $j$th item from the knapsack
16.     $x_j \leftarrow 1$
17.     if $\sum_{k=1}^{n} w_k x_k > Capacity$ then
18.       set the knapsack-overfilled $true$
19.     End if
20.   End while
21.   $x_j \leftarrow 0$
22. End
```

We experiment with three different knapsack problems with 100, 250 and 500 items, respectively, in QTS, and compare the results with those of other evolutionary algorithms including quantum-inspired electromagnetism-like mechanism (QEM) (Chou et al. 2010), QEA, TS and genetic algorithms (GA). The parameter settings are as follows (Table 4).

**Table 3** Unreasonable setting in 0/1 knapsack problem (CASE I)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $w_k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $p_k$ | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $p_k \diagdown w_k$ | 6 | $\cdots$ | $\cdots$ | $\cdots$ | 2 | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | 1.5 |

**Table 4** The parameters of experiments for the 0/1 knapsack problem

| | QTS | QEM | QEA | TS | GA |
|---|---|---|---|---|---|
| Population size | 10 | 10 | 10 | 10 | 1,000 |
| $\Delta\theta$ | $0.01\pi$ | X | $0.01\pi$ | X | X |
| Cross over | X | X | X | X | 0.65 |
| Mutation | X | X | X | X | 0.05 |

**Table 5** Experimental results of the 0/1 knapsack problem (CASE I)

| Items | Profit | GA | TS | QEA | QTS |
|---|---|---|---|---|---|
| 100 | $b.$ | 580.9 | 606.2 | 617.3 | 619.6 |
| | $m.$ | 577.4 | 603.4 | 604.1 | 613.4 |
| | $w.$ | 543.0 | 536.3 | 543.1 | 544.9 |
| t (sec/run) | | 0.6407 | 0.5256 | 0.5821 | 0.4429 |
| 250 | $b.$ | 1,400.9 | 1,494.0 | 1,532.3 | 1,546.5 |
| | $m.$ | 1,394.1 | 1,485.6 | 1,476.0 | 1,514.8 |
| | $w.$ | 1,336.3 | 1,324.3 | 1,336.2 | 1,338.5 |
| t (sec/run) | | 0.7449 | 0.5353 | 0.9266 | 0.6739 |
| 500 | $b.$ | 2,766.5 | 2,981.9 | 3,030.3 | 3,098.4 |
| | $m.$ | 2,754.8 | 2,960.7 | 2,906.6 | 3,003.8 |
| | $w.$ | 2,666.8 | 2,635.9 | 2,667.8 | 2,673.0 |
| t (sec/run) | | 0.9712 | 0.5654 | 1.5083 | 1.0064 |
| 100 | $b.$ | 580.9 | 606.2 | 617.3 | 619.6 |
| | $m.$ | 577.4 | 603.4 | 604.1 | 613.4 |
| | $w.$ | 543.0 | 536.3 | 543.1 | 544.9 |
| t (sec/run) | | 0.6407 | 0.5256 | 0.5821 | 0.4429 |
| 250 | $b.$ | 1,400.9 | 1,494.0 | 1,532.3 | 1,546.5 |
| | $m.$ | 1,394.1 | 1,485.6 | 1,476.0 | 1,514.8 |
| | $w.$ | 1,336.3 | 1,324.3 | 1,336.2 | 1,338.5 |
| t (sec/run) | | 0.7449 | 0.5353 | 0.9266 | 0.6739 |
| 500 | $b.$ | 2,766.5 | 2,981.9 | 3,030.3 | 3,098.4 |
| | $m.$ | 2,754.8 | 2,960.7 | 2,906.6 | 3,003.8 |
| | $w.$ | 2,666.8 | 2,635.9 | 2,667.8 | 2,673.0 |
| t (sec/run) | | 0.9712 | 0.5654 | 1.5083 | 1.0064 |

Each algorithm runs 100 samples, and the average profit of CASE I is shown in Table 5, and of CASE II in Table 6. The profits (average best $b.$, average mean $m.$ and worst $w.$) of QTS are much better than those of the others. It can be seen in Figs. 2, 3, 4, 5, 6 and 7 that when the number of items increases, the profit of QTS progressively increases. In addition, QTS most rapidly determines the best solution.

## 6.2 Solving the 0/1 multiple knapsack problem

If there is more than one knapsack constraints in a 0/1 knapsack problem, it can be generalized to a multiple knapsack problem (MKP) (Martello and Toth 1990). The MKP can be

**Table 6** Experimental results of the 0/1 knapsack problem (CASE II)

| Items | Profit | GA | TS | QEA | QTS |
|---|---|---|---|---|---|
| 100 | *b*. | 444.8 | 469.6 | 484.9 | 485.6 |
| | *m*. | 441.9 | 467.0 | 470.1 | 478.8 |
| | *w*. | 415.3 | 409.0 | 414.6 | 416.3 |
| t (sec/run) | | 0.6130 | 0.4967 | 0.4491 | 0.3984 |
| 250 | *b*. | 1,068.4 | 1,152.6 | 1,192.0 | 1,207.6 |
| | *m*. | 1,063.3 | 1,145.1 | 1,139.1 | 1,175.6 |
| | *w*. | 1,018.3 | 1,002.5 | 1,017.9 | 1,018.4 |
| t (sec/run) | | 0.7376 | 0.514 | 0.8835 | 0.6187 |
| 500 | *b*. | 2,115.6 | 2,309.4 | 2,355.3 | 2,425.0 |
| | *m*. | 2,108.0 | 2,290.5 | 2,244.2 | 2,334.0 |
| | *w*. | 2,043.8 | 2,014.9 | 2,044.1 | 2,043.2 |
| t (sec/run) | | 0.9503 | 0.6754 | 1.5708 | 0.9265 |

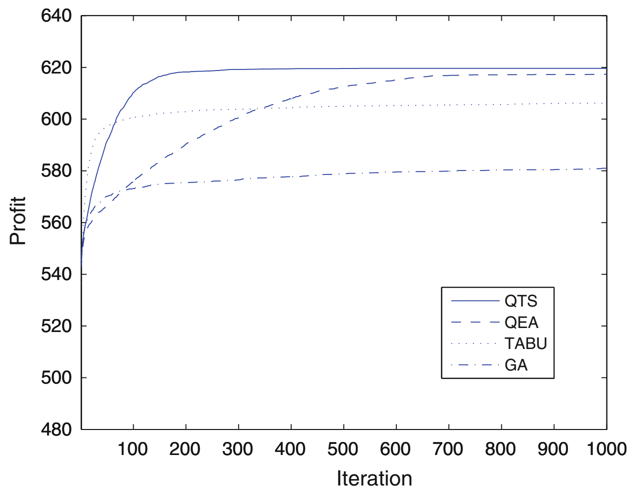

**Fig. 2** 100 items (CASE I)

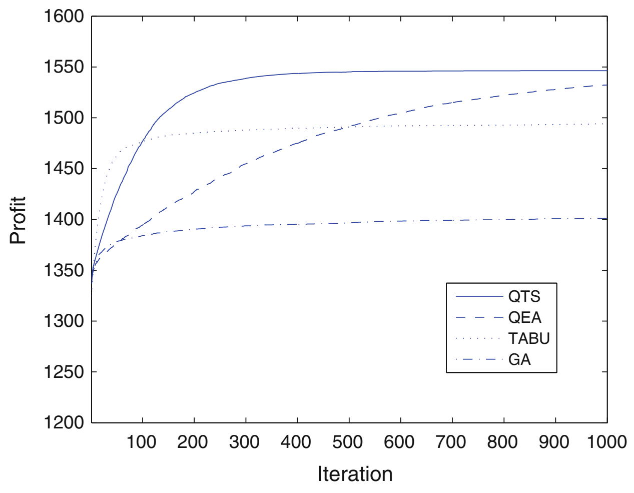

**Fig. 3** 250 items (CASE I)

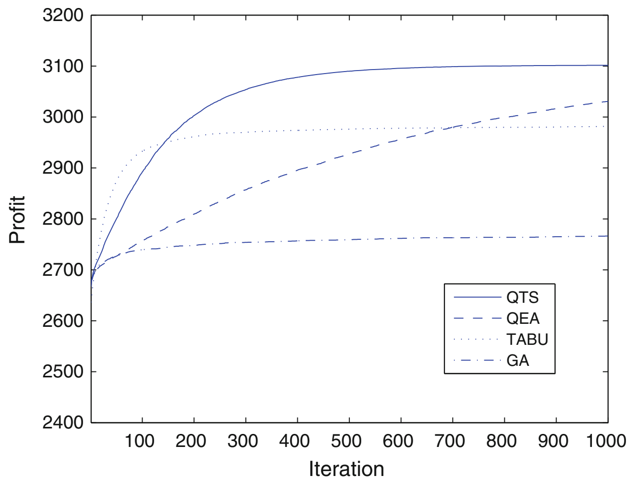

**Fig. 4** 500 items (CASE I)

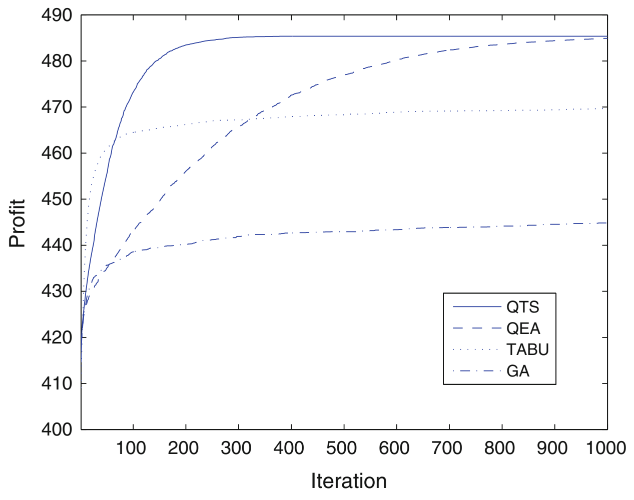

**Fig. 5** 100 items (CASE II)

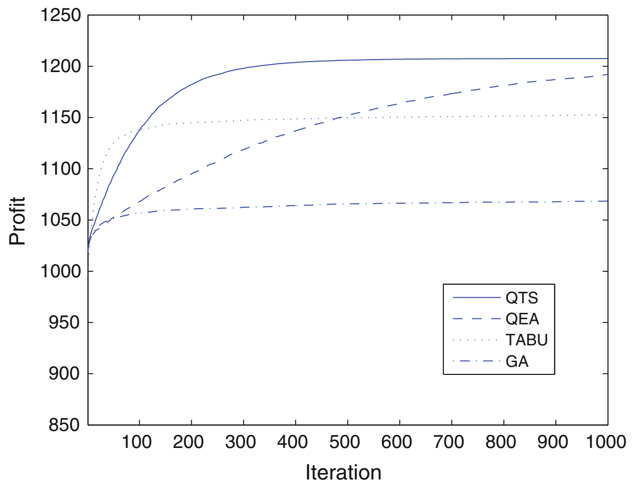

**Fig. 6** 250 items (CASE II)

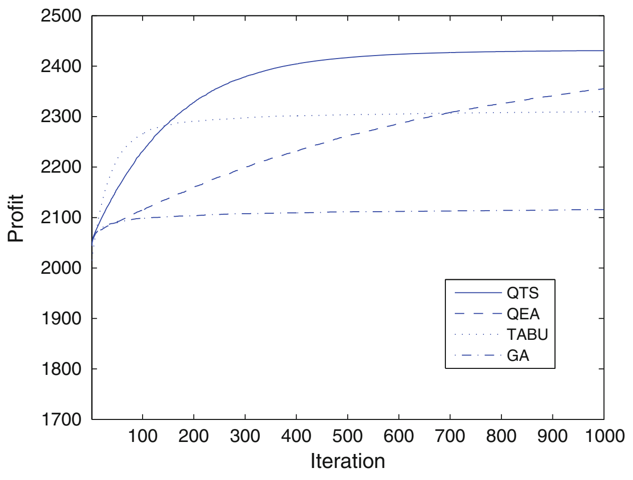

**Fig. 7** 500 items (CASE II)

defined as: given a set of $n$ items and a set of $m$ knapsacks, with

$$p_j = \text{profit of item } j,$$
$$w_j = \text{weight of item } j,$$
$$ci = \text{capacity of knapsack } i,$$

select a combination of items that acquires maximum profit, with various resource capacity constraints, and each subset being allowed to be assigned to a different knapsack whose capacity is not less than the total weight of items in the subset. Formally, maximize $z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$, subject to $\sum_{j=1}^{n} w_j x_{ij} \leq C_i, i \in M = 1, 2, \ldots, m, x_{ij} = 0 \; or \; 1, j \in N = 1, 2, \ldots, n$, where

$$x_{ij} = \begin{cases} 1 \text{ if } item\ j\ is\ assigned\ to\ knapsack\ i, \\ 0 \text{ otherwise.} \end{cases}$$

Therefore, without loss of generality, it is usual to assume that coefficients $p_j$, $w_j$ and $c_i$ are positive integers,

$$w_j \leq \max_{i \in M}\{c_i\} \quad for\ j \in N,$$
$$c_i \geq \min_{j \in N}\{w_j\} \quad for\ i \in M,$$
$$\sum_{j=1}^{n} w_j > c_i \qquad for\ i \in M$$

when $m = 1$, MKP reduces to the 0/1 (single) knapsack problem discussed in the previous section, and QTS can effectively solve it. Furthermore, we test some standard problems which are available from the OR-Library (http://people.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html) and (ZIB MP-Testdata 1993) in order to evaluate the performance of QTS in solving the MKP (Table 7). The results, given in Table 8, show, for each instance, the number of items and constraints (const.), the known optimal solution (opt. sol.), the maximum number of iterations required to find the best solution, the number of optimal solutions found, average profit with the execution of 10 runs, and the size of Neighbor solutions $N$. Moreover, certain instances must use the best solution $b$ and $s^w$ for updating $qubits$ to be able to find the better solution; in such situations, the instance will add the suffix "a".

**Table 7** Results obtained of TSP by QTS

| Test | burma14 | | Ulysses16 | | Gr17 | | Gr21 | |
|---|---|---|---|---|---|---|---|---|
| | Result | Dev. | Result | Dev. | Result | Dev. | Result | Dev. |
| 1 | 3,323 | 0 | 6,747 | 0 | 2,085 | 0 | 2,707 | 0 |
| 2 | 3,323 | 0 | 6,747 | 0 | 2,085 | 0 | 2,707 | 0 |
| 3 | 3,323 | 0 | 6,747 | 0 | 2,131 | 0.02 | 2,940 | 0.086 |
| 4 | 3,323 | 0 | 6,763 | 0.002 | 2085 | 0 | 2,998 | 0.107 |
| 5 | 3,323 | 0 | 6,753 | 0.0008 | 2090 | 0.002 | 2,707 | 0 |
| 6 | 3,323 | 0 | 6,758 | 0.002 | 2090 | 0.002 | 3,017 | 0.115 |
| 7 | 3,323 | 0 | 6,747 | 0 | 2,085 | 0 | 2,801 | 0.035 |
| 8 | 3,323 | 0 | 6,747 | 0 | 2,090 | 0.002 | 2,707 | 0 |
| 9 | 3,323 | 0 | 6,747 | 0 | 2,090 | 0.002 | 2,707 | 0 |
| 10 | 3,323 | 0 | 6,763 | 0.002 | 2,085 | 0 | 2,709 | 0.0007 |
| Opt. sol. | 3,323 | | 6,747 | | 2,085 | | 2,707 | |
| Mean cost | 3,323 | | 6,751.9 | | 2,091.6 | | 2,800 | |
| $\Delta\theta$ | 0.001 | | 0.001 | | 0.001 | | 0.001 | |
| Size of N | 100 | | 4,000 | | 4,000 | | 4,000 | |
| Max iter. | 1,000 | | 2,000 | | 2,000 | | 4,000 | |
| Mean t (s) | 5.52 | | 253.31 | | 258.08 | | 320.20 | |

The results indicate that QTS was able to effectively solve MKP with a low number of items, implying that the number of constraints does not have a significant influence on the performance of QTS. However, if the number of items increases, QTS cannot easily obtain the optimal solution.

## 6.3 Solving the traveling salesman problem

The traveling salesman problem (TSP) is a well-known NP-complete combinatorial optimization problem. These problems are widely considered unsolvable using polynomial time algorithms. In this section, we will explain how QTS is used to solve TSP. The problem can be stated as follows:a set of $n$ cities $C = \{c_1, c_2, \ldots, c_n\}$ and distances $D = \{(c_i, c_j) : c_i, c_j \in C, i \neq j\}$ for each pair of cities,find the shortest tour that visits each city exactly once and then returns to the starting city.

The encoding method we used is similar to Wanga et al. (2007). Quantum population can be represented as an $n \times c$ matrix as follows:

$$Q(t) = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1c} \\ q_{21} & q_{22} & \cdots & q_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1} & q_{n2} & \cdots & q_{nc} \end{bmatrix}$$

**Table 8** Experimental result of 0/1 knapsack problem (CASE II)

| Inst. | Const.# | Item# | Opt. sol. | Opt.# | Mean | $\Delta\theta$ | N | Iter.# | t (s) |
|---|---|---|---|---|---|---|---|---|---|
| $knap10_a$ | 10 | 10 | 87,061 | 10 | 87,061 | 0.01 | 10 | 10 | 0.02 |
| $knap15_a$ | 10 | 15 | 4,015 | 10 | 4,015 | 0.01 | 10 | 70 | 0.06 |
| $knap20_a$ | 10 | 20 | 6,120 | 10 | 6,120 | 0.01 | 50 | 60 | 0.11 |
| $knap28_a$ | 10 | 28 | 12,400 | 10 | 12,400 | 0.01 | 100 | 80 | 0.22 |
| $knap39_a$ | 5 | 39 | 10,618 | 4 | 10,608 | 0.001 | 500 | 3,000 | 22.82 |
| $knap50_a$ | 5 | 50 | 16,537 | 10 | 16,537 | 0.001 | 500 | 800 | 12.43 |
| hp1 | 4 | 28 | 3,418 | 10 | 3,418 | 0.001 | 100 | 1,000 | 3.11 |
| hp2 | 4 | 35 | 3,186 | 10 | 3,186 | 0.001 | 200 | 1,500 | 9.01 |
| pb1 | 4 | 27 | 3,090 | 10 | 3,090 | 0.001 | 30 | 1,000 | 3.37 |
| pb2 | 4 | 34 | 3,186 | 10 | 3,186 | 0.001 | 100 | 1,000 | 3.37 |
| pb4 | 2 | 29 | 95,168 | 10 | 95,168 | 0.01 | 150 | 2,000 | 9.12 |
| pb5 | 10 | 20 | 2,139 | 10 | 2,139 | 0.001 | 30 | 1,000 | 0.71 |
| pb6 | 30 | 40 | 776 | 10 | 776 | 0.01 | 60 | 1,000 | 9.68 |
| pb7 | 30 | 37 | 1,035 | 10 | 1,035 | 0.01 | 60 | 500 | 2.95 |
| sent01 | 30 | 60 | 7,772 | 10 | 7,772 | 0.001 | 100 | 3,500 | 47.24 |
| sent02 | 30 | 60 | 8,722 | 10 | 8,722 | 0.001 | 100 | 1,500 | 16.05 |
| weing1 | 2 | 28 | 141,278 | 10 | 141,278 | 0.01 | 100 | 100 | 0.27 |
| weing2 | 2 | 28 | 130,883 | 10 | 130,883 | 0.01 | 100 | 100 | 0.27 |
| weing3 | 2 | 28 | 95,677 | 10 | 95,677 | 0.01 | 100 | 200 | 0.68 |
| weing4 | 2 | 28 | 119,337 | 10 | 119,337 | 0.01 | 100 | 150 | 0.56 |
| weing5 | 2 | 28 | 98,796 | 10 | 98,796 | 0.01 | 100 | 100 | 0.33 |
| weing6 | 2 | 28 | 130,623 | 10 | 130,623 | 0.001 | 100 | 600 | 1.85 |
| $weing7_a$ | 2 | 105 | 1,095,445 | 2 | 1,095,400 | 0.0001 | 100 | 20,000 | 297.3 |
| $weing8_a$ | 2 | 105 | 624,319 | 1 | 620,660 | 0.0003 | 50 | 15,000 | 260.0 |
| weish01 | 5 | 30 | 4,554 | 10 | 4,554 | 0.01 | 100 | 100 | 0.24 |
| weish02 | 5 | 30 | 4,536 | 10 | 4,536 | 0.01 | 100 | 100 | 0.30 |
| weish03 | 5 | 30 | 4,115 | 10 | 4,115 | 0.01 | 100 | 100 | 0.48 |
| weish04 | 5 | 30 | 4,561 | 10 | 4,561 | 0.01 | 100 | 100 | 0.40 |
| weish05 | 5 | 30 | 4,514 | 10 | 4,514 | 0.01 | 100 | 100 | 0.54 |
| weish06 | 5 | 40 | 5,557 | 10 | 5,557 | 0.05 | 200 | 200 | 2.13 |
| weish07 | 5 | 40 | 5,567 | 10 | 5,567 | 0.01 | 200 | 100 | 0.90 |
| weish08 | 5 | 40 | 5,605 | 10 | 5,605 | 0.01 | 200 | 120 | 0.94 |
| weish09 | 5 | 40 | 5,246 | 10 | 5,246 | 0.01 | 100 | 150 | 0.79 |
| weish10 | 5 | 50 | 6,339 | 10 | 5,246 | 0.01 | 100 | 250 | 1.58 |
| weish11 | 5 | 50 | 5,643 | 10 | 5,643 | 0.01 | 100 | 350 | 3.21 |
| weish12 | 5 | 50 | 6,339 | 10 | 6,339 | 0.01 | 100 | 250 | 1.59 |
| weish13 | 5 | 50 | 6,159 | 10 | 6,159 | 0.01 | 100 | 350 | 2.26 |
| weish14 | 5 | 60 | 6,954 | 10 | 6,954 | 0.01 | 100 | 250 | 2.02 |
| weish15 | 5 | 60 | 7,486 | 10 | 7,486 | 0.001 | 200 | 2,500 | 63.26 |
| $weish16_a$ | 5 | 60 | 7,289 | 10 | 7,289 | 0.001 | 200 | 1,800 | 11.87 |
| weish17 | 5 | 60 | 8,633 | 10 | 8,633 | 0.01 | 200 | 100 | 0.55 |
| weish18 | 5 | 70 | 9,580 | 10 | 9,580 | 0.001 | 200 | 1,200 | 9.44 |
| weish19 | 5 | 70 | 7,698 | 10 | 7,698 | 0.001 | 200 | 3,000 | 30.11 |
| $weish20_a$ | 5 | 70 | 9,450 | 10 | 9,450 | 0.001 | 200 | 1,800 | 14.12 |
| $weish21_a$ | 5 | 70 | 9,074 | 8 | 9,046.2 | 0.001 | 200 | 2,000 | 18.37 |
| $weish22_a$ | 5 | 80 | 8,947 | 10 | 8,947 | 0.001 | 200 | 2,500 | 19.96 |

**Table 8** continued

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $weish23_a$ | 5 | 80 | 8,344 | 8 | 8,343.4 | 0.001 | 200 | 3,000 | 26.03 |
| $weish24_a$ | 5 | 80 | 10,220 | 10 | 10,220 | 0.001 | 500 | 1,800 | 18.25 |
| $weish25_a$ | 5 | 80 | 9,939 | 7 | 9,934.2 | 0.001 | 500 | 3,000 | 31.73 |
| $weish26_a$ | 5 | 90 | 9,584 | 7 | 9,574.4 | 0.001 | 500 | 3,000 | 60.68 |
| $weish27_a$ | 5 | 90 | 8,919 | 10 | 8,919 | 0.001 | 200 | 2,000 | 22.60 |
| $weish28_a$ | 5 | 90 | 9,492 | 8 | 9,487.4 | 0.001 | 200 | 3,000 | 32.47 |
| $weish29_a$ | 5 | 90 | 9,410 | 9 | 9,406.8 | 0.001 | 500 | 2,500 | 63.20 |
| $weish30_a$ | 5 | 90 | 11,191 | 10 | 11,191 | 0.001 | 300 | 2,000 | 20.03 |

where $n$ is the number of cities and $c$ is the minimum number of bits used to represent all cities, which satisfies $2^c \geq m$. We can get a binary matrix $X = \{x_{11}, \ldots, x_{1c}; x_{21}, \ldots, x_{2c}; x_{n1}, \ldots, x_{nc}\}$ by measuring a $Q(t)$. Each binary string $\{x_{i1}, \ldots, x_{ic}\}$ is the visited sequence of the $i$th city, where $i = 1, 2, \ldots, n$. By sorting all binary strings we can obtain a visited sequence $S_i = \{s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_n\}$. We then evaluate the total length of each sequence $S_i$ with object function $f(s) = -\sum_{j=1}^{n-1} d(s_j, s_{j+1}) + d(s_n, s_1)$, where $d(s_i, s_j)$ is the distance between cities $s_i$ and $s_j$.

In order to avoid *qubits* becoming convergent before QTS finds the optimal solution, we add a mutation mechanism for updating $Q$ to change the probability distribution to escape local optimum, with mutation probability usually set to 0.1∼0.2.

The performance of our proposed algorithm for TSP is examined by four benchmark problems from (TSPLIB 1997). Two examples are shown in Figs. 8 and 9. Table 7 gives the experimental results, which show that QTS was unable to efficiently and effectively deal with the small-scale TSP. We

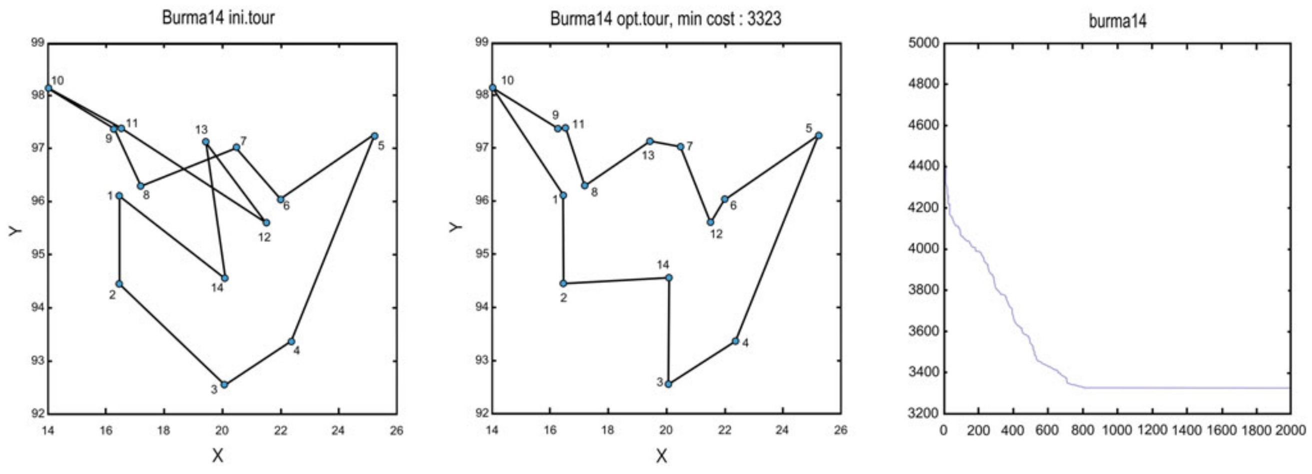

**Fig. 8** Results of optimal solution in Burma14

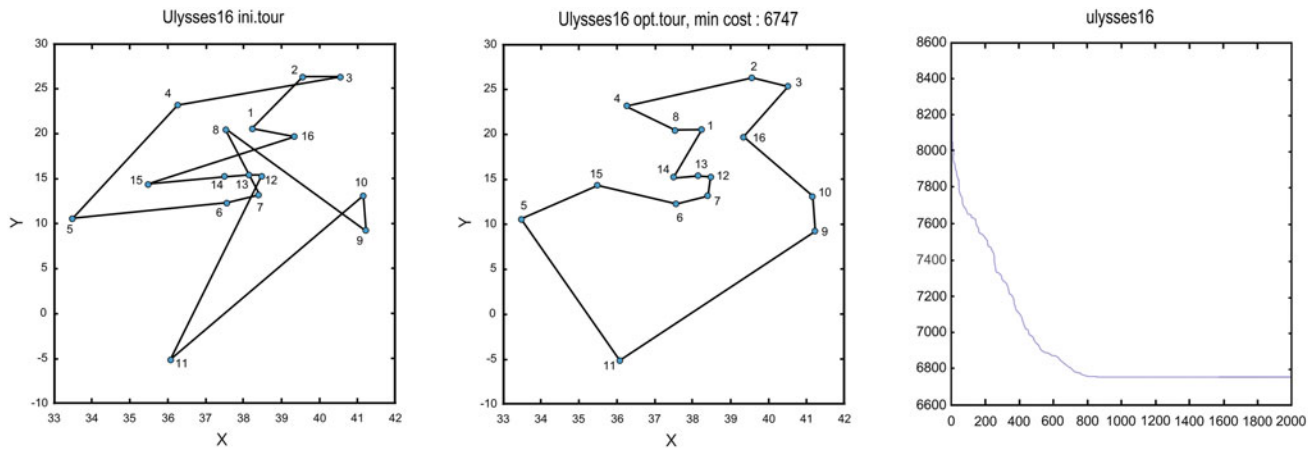

**Fig. 9** Results of optimal solution in Ulysses16

also calculate deviation as dev. of each test result and optimal solution. With the exception of Burmal14, QTS must be given a larger number of neighboring solutions to have a greater chance of finding the optimal solution, as with Ulysses16, Gr17 and Gr21.

## 7 Conclusions

This paper combines classical Tabu search with the features of quantum computation to propose a novel QEA, named QTS. We use QTS to solve 0/1 knapsack problems, multiple knapsack problems and the traveling salesman problem, which are problems can be used in computer science, cryptography and so on. Although quantum computers are still developing, we simulate a similar mechanism on a classical computer.

First, we maintain a population of Q to record qubit states, then measuring Q will obtain binary strings which are possible solutions. We also propose a new operation of tabu list, which is well suited to solving the knapsack problem for TS or QTS, and helps the algorithm avoid trapping in a local optimality. The repair procedure keeps the possible solutions in the feasible domain. After obtaining possible solutions, we use the best and worst solution as the reference bits to update the qubits original state. This gradually forces the possible solutions toward the elite solutions, and makes the whole procedure more precise and efficient.

Generally, the abovementioned steps can help us get closer to the optimal solutions for combinatorial optimization problems. The results of the 0/1 knapsack problem experiments show that QTS does perform better than other heuristic algorithms, both in terms of time and the optimal solution. However, the results of the MKP and TSP experiments have not much great effect from QTS, with the increase in quantity of items or cities resulting in a decrease in the rate of finding the optimal solution. In the future work, QTS may use more neighbor solutions and generations to effectively find a global optimal solution.

**Acknowledgments** This research was partly funded by the National Science Council of the ROC under grants NSC 100-2221-E-260-038, NSC 101-2221-E-260-033 and NSC 102-2221-E-260-017.

## References

Beasley J OR-LibraryG http://people.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html

Benny C, Ronald LR (1988) A knapsack-type public key cryptosystem based on arithmetic in finite fields. IEEE Trans Inf Theory 34(5):901–909

Chelouah R, Siarry P (2000) Tabu Search applied to global optimization. Eur J Oper Res 123(2):256–270

Chou YH, Chang CC, Chiu CH, Lin FJ, Yang YJ, Peng ZY (2010) Classical and quantum-inspired electromagnetism-like mechanism for solving 0/1 knapsack problems. In: Proceedings 2010 IEEE International Conference on Systems Man and Cybernetics, SMC 2010, 10–13 Oct 2010. pp 3211–3218

Chou YH, Yang YJ, Chiu CH (2011) Classical and quantum-inspired Tabu search for solving 0/1 knapsack problem. In: Proceedings 2011 IEEE International Conference on Systems Man and Cybernetics, SMC 2011, pp 1364–1369, 9–12 Oct 2011

Chou YH, Chen CY, Chiu CH, Chao HC (July 2012) Classical and quantum-inspired electromagnetism-like mechanism and its applications. IET Control Theory Appl 6(10):1424–1433

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549

Glover F (1989) Tabu search: part I. ORSA J Comput 1(3):190–206

Glover F (1990) Tabu search: part II. ORSA J Comput 2(1):4–32

Han KH, Kim JH (2000) Genetic quantum algorithm and its application to combinatorial optimization problem. In: Proceedings 2000 Congress on Evolutionary Computation, vol 2. IEEE Press, Piscataway, NJ, pp 1354–1360

Han KH, Kim JH (2004) Quantum-inspired evolutionary algorithms with a new termination criterion, hc gate, and two phase scheme. IEEE Trans Evol Comput 8(2):156–169

Han KH, Kim JH (2002) Quantum-inspired evolutionary algorithm for a class of combinatorial. IEEE Trans Evol Comput 6:580–593

Hu N (1992) Tabu search method with random moves for globally optimal design. Int J Numer Methods Eng 35(5):1055–1070

Jiao LC, Li YY, Gong MG, Zhang XR (2008) Quantum-inspired immune clonal algorithm for global optimization. IEEE Trans Syst Man Cybern Part B 38(5):1234–1253

Li BB, Wang L (2007) A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling. IEEE Trans Syst Man Cybern Part B 37(3):576–591

Li BB, Ling W (June 2007) A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling. IEEE Trans Syst Man Cybern Part B Cybern 37(3):576–591

Li Y, Zhao J, Jiao L, Wu Q (2009) Quantum-inspired evolutionary multicast algorithm. In: Proceedings Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on, pp 1496–1501, 11–14 October 2009

Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, Chichester

Merkle R, Hellman ME (1978) Hiding information and signatures in trapdoor knapsacks. IEEE Trans Inf Theory 24(5):525–530

Narayan A, Patvardhan C (2009) A novel quantum evolutionary algorithm for quadratic Knapsack problem. In: Proceedings Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on, 11–14 October 2009. pp 1388–1392

Simon K, Willi M (2011) Cryptanalysis of the knapsack generator In: Joux A (Ed) Proceedings of the 18th international conference on Fast software encryption (FSE'11), vol 6733. Springer, Berlin pp 1888-198

Standard TSPLIB (1997) http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Talbi H, Draa A, Batouche M (2004) A new quantum-inspired genetic algorithm for solving the travelling salesman problem. IEEE ICIT 3:1192–1197

Wanga Y, Fenga XY, Huanga YX, Pub DB, Zhoua WG, Lianga YC, Zhou CG (2007) A novel quantum swarm evolutionary algorithm and its applications. In: Neurocomputing. pp 633–640

ZIB MP-Testdata (1993) http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/index.html