On the adoption of Metaheuristics for Solving 0-1 Knapsack Problems

Yang Qiu
Department of Computer Science
Wenzhou-Kean University
Wenzhou 325060, CHINA
yangq@kean.edu

Haomiao Li
Department of Computer Science
Wenzhou-Kean University
Wenzhou 325060, CHINA
haomiaol@kean.edu

Xiao Wang
Department of Computer Science
Wenzhou-Kean University
Wenzhou 325060, CHINA
xiaowa@kean.edu

Omar Dib*
Department of Computer Science
Wenzhou-Kean University
Wenzhou 325060, CHINA
odib@kean.edu

***Abstract*—0-1 knapsack problem is a classical Non-deterministic Polynomial Hard (NP-Hard) problem in combinatorial optimization. It has a wide range of applications in real life, such as the distribution of goods in logistics companies, capital calculation, storage, and distribution, etc. This paper studies the adoption of three meta-heuristic approaches for solving the 0-1 knapsack problem. A novel quantum-inspired Tabu Search (QTS) that combines a classical Tabu Search (TS) and Quantum-inspired Evolutionary Algorithm (QEA), Ant Colony swarm intelligence Algorithm (ACO), and Genetic Algorithm (GA) with augmented fitness function. Based on empirical analysis of computer simulations and comparative demonstrations, we show that applying such metaheuristic results in high efficiency in terms of the quality of obtained solutions, computational time, and robustness when dealing with the underlying 0-1 knapsack optimization problem.**

Keywords—0-1 knapsack; Tabu Search; Quantum-inspired Evolutionary Algorithm; Genetic Algorithm; Ant Colony Optimization

I. INTRODUCTION

A. 0-1 Knapsack Problem

0-1 knapsack Problem is a classical NP-hard problem, theoretically described as given n items and a backpack; each object has two attributes: price V_i and weight W_i. And the backpack capacity is C. The objective is to maximize the total value of the items *n* transferred into the backpack while respecting the latter capacity [9]. There are only two states for each item, not loaded into the backpack (0), loaded into the backpack (1). Abstractly speaking, it is to solve the combinatorial optimization problem of maximizing the value of the objective function (Total value of the contents n of the backpack) under the given constraint (Backpack capacity C remains unchanged). Formally, the problem may be defined as follows:

$$\max \sum_{i=1}^{n} v_i x_i \quad (1)$$

$$s.t. \quad \sum_{i=1}^{n} w_i x_i \leq C \quad (2)$$

$$x_i \in [0,1], \quad i = 1,2,\ldots,n \quad (3)$$

Objective Function (1): Maximize the value V_i of the items in the bag.
Constraint Condition (2): Capacity limit of the package C.
In formula (3): the binary decision variable $x_i = 1$ means loading item n_i in package, $x_i = 0$ means not loading n_i.

Solving the 0-1 knapsack problem can be divided into deterministic (exact) and heuristic algorithms. Exact algorithms include Branch and X, dynamic programming, and recursive algorithms, et al. [4]. Such approaches can find the optimal solution(s), but they are often only suitable for small-scale problems due to their high computational time that grows exponentially with the number of items. On the other hand, heuristic algorithms include greedy approaches, genetic algorithms, simulated annealing, tabu search, etc.[20], which are generally used for solving large and more complex problems. Greedy algorithms have good climbing performance, but they usually converge to a locally optimal solution. While Greedy Algorithms have strong dependencies on the initial starting point, the optimization result of the genetic algorithm remains relatively good even with poor initial conditions. Practically speaking, a GA requires less information about the problem, has strong robustness against uncertainty, and is suitable for solving dynamic and complex optimization problems. However, a GA might encounter some difficulties due to the intricate design of the objective function and its high computational cost, resulting in slow convergence speed. [14]. Moreover, GA might achieve mediocre performance when dealing with optimization problems with lots of constraints. As each metaheuristic has its pros and cons, we aim in this paper to empirically analyze the performance of meta-heuristic approaches of different nature and characteristics for solving the 0-1 knapsack problem.

B. Heuristic and Meta-heuristic Algorithm

The pillar point of any Heuristic Strategy (algorithm) is to give a solution to the problem within an acceptable computational time and space; however, such a solution is not guaranteed to be the optimal one, and can only be determined as a feasible solution.

The design of the algorithm largely depends on the developer's experience. There are no standard guidelines to create a specific heuristic for a problem. Developers only need to obtain the feasible solution and then get closer to the optimal solution until a local optimum is found. This leads to a heuristic algorithm that is highly dependent on the problem itself. Even though greedy heuristics are very

popular, they often lead to a feasible solution falling into a local optimum instead of a global optimum. To overcome this shortcoming, the idea of a meta-heuristic was introduced.

A Meta-heuristic algorithm is a derivative of a heuristic strategy. The main difference is that a metaheuristic is not explicitly designed for one problem, thereby expanding its applicability to a wide range of problems [5]. Meta-Heuristic algorithms generally do not pursue the greedy paradigm during the search process and can even accept worse solutions than the current optimal solution, such as Tabu Search (TS) and Simulated Annealing (SA), which expands the decision space. Due to that, there is a greater probability of handling local minimum regions in the search space and obtaining the optimal solution as the search space is extended and carefully explored [3].

II. Related Work

A. Dynamic Programming and Greedy Random Adaptive Search

At present, when the data volume is small, dynamic programming is one of the classical algorithms to find the optimal solution for knapsack problems. The core idea of dynamic programming is to avoid computing the same subproblem multiple times by memorizing its solution, and thereby improve the overall efficiency of the search algorithm. When the input volume is small, the efficiency of dynamic programming approaches is linear $O(n* W)$ with the number of items n and the total capacity of the knapsack W. However, as the data pair grows, dynamic programming often requires significant time and space overhead. Differently, Greedy Random Adaptive Search Algorithm (GRASP) is a multi-start iterative process of searching for local optimal solutions [14]. Each iteration consists of two stages: one is the construction stage to produce a feasible solution; the other is the local search stage to find the optimal local solution starting from the initial state. More specifically, GRASP works by greedily and randomly constructing a solution before the algorithm iteration and then judging whether it is feasible or not. If it is not feasible, it enters the Repair function for correction, performs a local search for the feasible solution, and updates the optimal solution.

B. Metaheuristic Algorithm

The meta-heuristic algorithm is proposed relative to the optimization algorithm. The meta-heuristic algorithm gives an approximate optimum of the problem at an acceptable time and space cost [8]. There are many branches and extensions of meta-heuristic algorithms. Tabu Search (TS), Genetic Algorithm (GA), Simulated Annealing (SA), such non-deterministic algorithms are prevalent in the application of knapsack problems. Take Tabu Search (TS) as an example, except for naive TS; there is also a tabu search algorithm based on double tabu lists and a Quantum-Inspired Tabu Search (QTS). TS and QEA were combined in QTS. The idea was created by Han and Kim [11]. Unlike single solution metaheuristics, the Genetic Algorithm (GA) is a random search population-based optimization algorithm based on biological evolution and molecular genetics. GAs are self-organizing, self-adapting, and learning [13]. They are parallel, do not require derivation or other auxiliary knowledge, and emphasize probability conversion rules. It is feasible to use a GA to solve the knapsack problem. Still, the traditional GA has shortcomings such as "premature convergence," dealing with a large number of constraints, handling multiple objectives, and deciding on uncertain variables [1][15].

III. Algorithms Description

A. Tabu Search

Tabu Search is an extension of local domain search and a global step-by-step optimization algorithm. It involves concepts such as Neighborhood Structure, Tabu List, Tabu Length, Neighbor Candidate, and Aspiration Criterion. "Tabu" is a term that refers to prohibits repeating the work that has been previously done, which simulates the "memory" function in human intelligence [19] by recording information (e.g., local optimums) related to the search process in a Tabu List. Such tabu components could not be selected again, and therefore avoided in further generations to continue searching in more promising regions in the search space. By branching out from the local optimum, the algorithm can explore more decision space. In addition, it reduces the risk of falling into a local optimum when searching for local neighborhoods [17]. Next, to prevent skipping the optimal solution, the Aspiration Criterion is introduced to release some "good" tabu status, and the global optimization is finally realized. Focus on knapsack problem; the TS algorithm process can be briefly described as two steps:

$STEP1$: Initialization: Choose an initial solution x_0 and a tabu table $\mathrm{H} \rightarrow \varnothing$.

$STEP2$: Iteration: Until a termination condition is satisfied.

otherwise, choose the candidate set $Can(x_i)$ in neighbor list $N(H, x)$ of x_i;

choose a solution x_i, with the best evaluation value in the candidate set, if $x_0 = x_i$; update list H, repeat Step2.

B. Quantum-inspired Tabu Search

Quantum-inspired Tabu Search (QTS) combines Quantum-inspired Evolutionary Algorithm (QEA) and the traditional Tabu Search (TS). The implementation of QTS in this work is based on this research paper [6].

Because of the characteristics of qubit and quantum superposition, quantum probability amplitudes and probabilistic models have been adopted to describe the promising areas of decision, avoid naïve tabu algorithm as it is easy to fall into local optimum, and ameliorate the convergence speed [12]. The QTS algorithm is given in Fig.1:

Algorithm 1: Quantum-inspired Tabu Search (QTS)

initialization: initialize solution s^i, fitness b, Quantum population $Q(0)$;
while ! *Termination Rule* **do**
  $i \leftarrow i + 1$
  Multiply the measurements of $Q(i-1)$ to obtain neighborhood Set N
  Check $s \in N$ and $f(s)$
  Update b
  Get best and worst local solution, s^{lb} and s^{lw}
  Update Tabu-List TT
  Update quantum individual $Q(i)$ by s^{lb} and s^i
  Update local solution s^i by $Q(i)$
  Update quantum individual $Q(i)$ by $s^{lw}$and s^i
  Update local solution s^i
end

1. Pseudocode of QTS

C. Ant Colony Optimization Algorithm

According to [7], "It has been proved that the ACO algorithm takes on a better optimization performance in solving optimization problems." for our Knapsack problem, [7]. In detail, it has n nodes (cargoes), and there are n directed lines segments starting from node i $(i = 1, 2\ldots, n)$, n connected to node $i + 1$. There are

two numbers W_j(the volume of the $j-th$ cargo) and P_j (the value of the $j-th$ cargo) on $j\ (j=1,2\ldots,\ n)$ as the parameters in [10]. For solving 0-1 knapsack problem, the path choice of ant is as follows: let $\tau_{ij}(T)$ is a directed line segment at $t\ (t=1,2\ldots,\ n)$. The pheromone on each directed line segment at the initial time is $\tau_{ij}(0)=C$ (C is a small normal constant). At time t, the generated m ants are placed on randomly selected nodes, and then each ant independently sets a directed line segment according to the pheromone and heuristic factors on the path and moves to the next node (cargo) until it cannot move forward. At time t, ant k, $(k=1,2\ldots,\ m)$ passes through the line segment from node $i\ (i=1,2\ldots,\ m)$ The probability $p_{ij}(T)$ of transition from i to $j\ (j=i+1)$ is [16]:

$$p_u^k(t)=\frac{\left[\tau_u(t)\right]^{\alpha}\ .\ \left[\eta_u(t)\right]^{\beta}}{\sum_s\left[\tau_{is}(t)\right]^{\alpha}\ .\ \left[\eta_{is}(t)\right]^{\beta}} \tag{4}$$

Among them: η_{ij} indicates that ants select directed line segments $\alpha[i,j]$, $\eta_{ij}=p_i/W_j$, P_i is the value of the object, W_j is the volume of the object, η. The larger the i, the higher the probability of selecting item j is; J_k (i) represents the set of directed line segments that ant K at node i can currently select. J_k (i) = {1,2, …, n }, the tabu list of ants k, records the current directed line segments, α, and β represent the relative importance of pheromone and heuristic factor on the directed line segment respectively. Different from as algorithm, the selected directed line segment can only be added into the Tabu if the constraints are satisfied when solving knapsack problems. The change in pheromone concentration is shown in the following formula [18]:

$$\tau ijt+n=1-\rho*\tau ij\cdot t+\Delta\tau ij \tag{5}$$

$$\Delta\tau ij=k=1m\Delta\tau ijk \tag{6}$$

D. Genetic Algorithm

In a genetic algorithm, to represent a solution for the 0-1 knapsack problem, firstly we construct a binary code with *n* bits as a chromosome and each item as an independent gene fragment. For example, 0100 means to put x_2 in the backpack. Then it will randomly generate *p* populations; the number of populations is considered as the number of chromosomes. In the knapsack problem, we use the total value of the item as a function of fitness, calculated by the following formula:

$$T=\sum s_i x_i \tag{7}$$

$$Fitness=\begin{cases}\sum v_i x_i, & T\le C\\ \sum v_i x_i-\alpha*(T-C), & T>C\end{cases} \tag{8}$$

When the total weight of the population is within the volume of the backpack, we use their total value as their population fitness. However, when the total weight of the population exceeds the volume C of the backpack, we enforce a penalty function. In this formula, alpha is an integer > 1. In our code implementation, we set the penalty function alpha to 2. For selection, we use a probability proportional to fitness to determine the number of individuals copied into the next generation population. First, we calculate the total fitness of all individuals in the group $\Sigma_{i=1}^{n}f(a)=\Sigma_{i=1}^{n}fitness$. Secondly, calculate the relative fitness of each individual, denoted by p.

$$p(ai)=\frac{f(a)}{\Sigma_{i=1}^{n}f(a)}*100\%.$$

It is the probability that each individual is inherited into the next generation population. We divide the interval for each individual in [0,1] according to the relative fitness and generate a random number between 0 and 1. Based on the probability, the individual is selected as the parent and crossed with the individual with the highest fitness. Then We adopt a single-point crossover method. In this step, the individual with the highest fitness has a certain probability of not performing the crossover operation but directly producing offspring. The offspring have the same genes as their parents. Finally, we reverse the original gene value of the mutation point according to a certain probability.

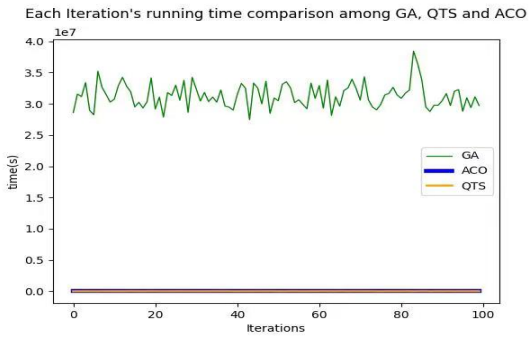
IV. RESULT

To assess the performance of the aforementioned algorithms, we have implemented them to solve state-of-the-art knapsack instances. All simulations and implementations of QTS, ACO, GA are coded in Python and executed on an Intel Core i7-7700 HQ CPU @2.80 GHz with 16GB of RAM using Windows 10. The general ideas of the algorithms and data are from the literatures [6] [11]. The basic paraments are set in Fig. 2:

Parameter Setting:
Knapsack object with $n=250,\ 500$;
Items attributes: price V_i and weight W_i are random generated;
Package capacity $c=\Sigma_{i=1}^{1}w_i x_i$, $i=1,2,\ldots,n$;
Maximum number of iterations $ittr=100$;
Neighbor Population $N=[10,\ 50]$;
Run times of program r=100, the result is getting the average best;
Quantum angle parameter θ=0.01π in QTS;
For GA, probabilities of crossover, variation rate, and mating rate: 2, 0.4, and 1.
For ACO, control coefficient of Pheromones α and control coefficient of visibility β: 3 and 2

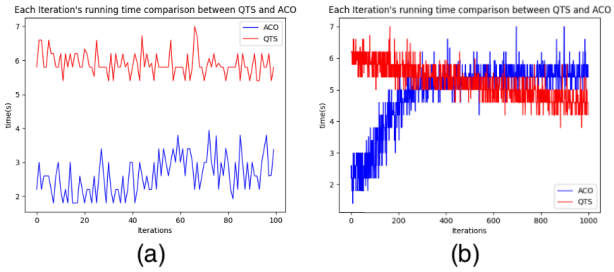
2. Basic paraments settings

First, we compare the running time for each iteration of QTS, GA, and ACO. We used the same data as the problem's input. In Figure 3, we use $n=500, ittr=100$, running ACO, GA, QTS codes simultaneously and comparing the running time of each iteration. To compare the running time of algorithms, the result is shown in Fig. 3:



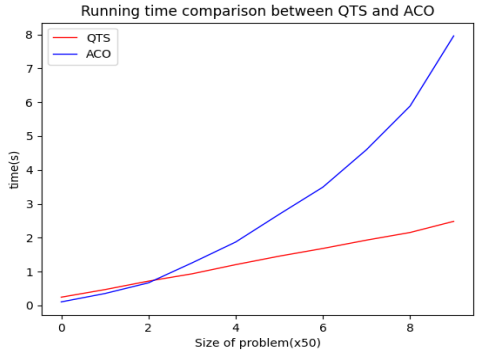
3. Comparison of the running time of QTS, GA and ACO at each iteration

Experimentations have proven that compared to QTS and ACO, GA's running time at each iteration is too high, around 30 seconds. However, here we are only comparing the case of $ittr = 100$ and $n = 500$, and the running time for each iteration is also not equal to the overall running time. When the three algorithms run simultaneously, the comparison of ACO and QTS is not evident due to the long GA iteration time. To make the experiment clearer, we separate GA, and the results are shown in Fig. 4:



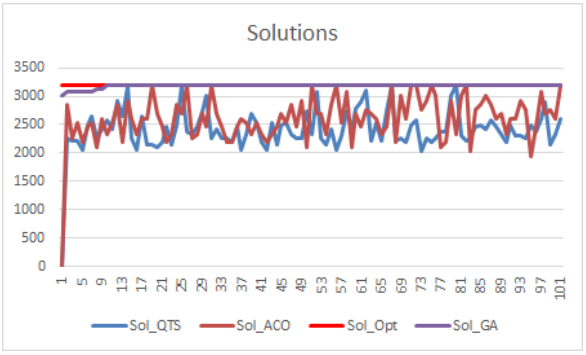
4. Comparison of ACO and QTS for running time each iteration

From Fig. 4(a), during 100 iterations, each iteration of ACO is faster than QTS, and both variations are generally stable. The range of fluctuation is almost equal. Increasing the iterations from 100 to 1000, as shown in Fig. 4(b), indicates that the running time of QTS decreases gradually with the whole algorithm, and the opposite is true for ACO. Hence, when problem size n is small, ACO is better than QTS, but for larger n, the QTS performance is better. Then we tried to gradually increase the scale of the input problem, from 50 to 500, to compare the running time difference between QTS and ACO. We input the same data, and the captivity is the sum of total weight divided by 10, iterations equally 100 times. The result is shown in Fig. 5:



5. Comparison of running time of QTS and ACO as scale increase

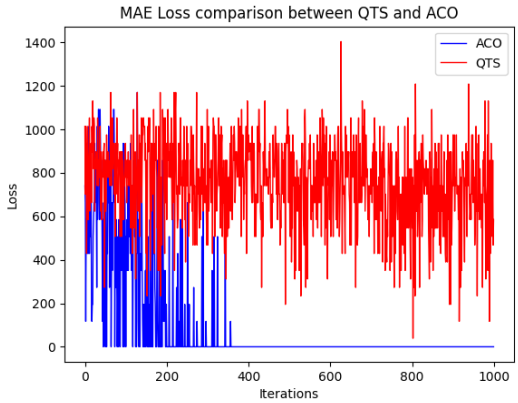
With the increase of scale, QTS's time increases linearly, but ACO's running time increases exponentially. That is, the running time of ACO is too high when the input size increases. After assessing the running time, we compared the local solutions among QTS, GA, and ACO. The same data is used, and the result is shown in Fig. 6:



6. Comparison of QTS, GA, and ACO for local solutions

As can be seen from Fig. 6, when the GA algorithm runs 100 times, the approximate optimal solution range is 3003-3198. The fitness function will remain unchanged when the GA reaches the highest objective value, so the algorithm cannot find a better approximate optimal solution. In most cases, the approximate optimal solution of this data set is 3198. According to the running result, the worst deviation of GA is 6.09%. In addition, GA only needs 13 iterations to get the highest fitness, and the fitness does not fluctuate anymore, while QTS and ACO still fluctuate after finding the approximate optimal solution. This shows that the convergence speed and stability of GA are much higher than QTS and ACO. However, according to Figure 3, it can be seen that the time required by the GA for one iteration is much higher than them, so the total time needed for GA is higher than QTS and ACO.

Moreover, the stability of ACO's algorithm is better than QTS, finding more approximate optimal solutions in 100 iterations and with fewer fluctuations than QTS. This may be related to the rule about picking the local optimal solution at each iteration. GA will get its optimized solution in only about 10 iterations, then the local solution frozen at the top 100 iterations is not necessarily finished. However, QTS and ACO must complete 100 iterations before they go to the next step, and here comes the problem of algorithmic stability of GA and ACO. To make it more straightforward, we separate GA and do the same experiment and calculate the loss of ACO and QTS:



7. Comparison of QTS and ACO for MAE loss

In the beginning, the ACO's loss fluctuated a lot, but after 400 iterations, it reduced to 0, at the time it found its optimized solution. QTS's loss still did not reduce to 0, but

the trend was declining. However, the reduced trend of ACO is more significant than QTS.

V. ANALYSIS

A. Tabu Search and QTS

First, in terms of running time, the implementation of Tabu Search and QTS on the knapsack problem is ordinary. But the 0-1 knapsack problem is an NP-Hard problem, and there is no polynomial-time solution that can completely control the error. Because it requires a small error, it is necessary that the global optimal solution must be obtained, but this is against the realization of the heuristic algorithm. In theory, the running time of the algorithm should be exponential.

Then there is the setting of the parameters in the algorithm. Because of the limitations of the python language, n is not easy to set too large, resulting in too long running time. Second, the number of iterations $ittr$ and the selection of the optimal solution directly affect the efficiency of the algorithm. For $ittr$, the following numbers have been tested: 50, 100, 200,500. After extending the calculation time, observing the results, and finding no improvements, we set $ittr = 100$ as the main parameter. Compared with the current parameter settings, the above experimental data has reached the optimal solution. Finally, summarize the effects and characteristics of QTS. Compared with other algorithms, the QTS algorithm is mainly different from the introduction of qubit (quantum bit concept). Qubit is primarily used in two aspects. First, when the algorithm is initialized to generate the initial solution, it is not generated randomly, each item is brought into the quantum gate-move matrix, and the solution with a more significant value is calculated (normal object n is converted to qubit $Q(i)$). This way, the initial solution generated is closer to the optimal solution in the quantum sense [6]. Second, it is used for the state of the object in the backpack after the *fit repair* process. In the update, the concept of gate-move is used again to determine when an item n_i leaves the backpack. The use of this qubit increases the concept of each item from a mathematical perspective, thereby expanding the decision space, so the efficiency of operation and accuracy is higher than that of standard algorithms.

B. Ants Colony Optimization

As a variant of genetic algorithm, Ants Colony Optimization first came up to solve TSP problems. However, as a metaheuristic algorithm, Ant Colony Optimization has been adapted to cope with0-1 knapsack problems. When visualizing the TSP problem, we get an undirected graph that can be incompletely connected. Each edge can be seen as the path to a different city. When all traveled, the algorithm stops. For 0-1 knapsack problems, it can be seen as a fully connected undirected graph. Each edge can be seen as the choice for the following item put into the knapsack. When the remaining captivity of the knapsack cannot contain any remaining items, the algorithm stops. Compared with other algorithms for 0-1 knapsack problems, ACO can find out optimized solutions. Compared with QTS, ACO can find a better solution. However, the cost of obtaining an excellent solution is paid at the running time level. When the size of the problem is small, the running time does not have a noticeable difference. However, with the increase in the input size, ACO will need more time to solve the problem. That is, ACO's running time increases exponentially with the instance magnitude. So, considering the time complexity, ACO turns out to be not suitable for large-scale problems. Meanwhile, according to Abdullah [2], "One of the approaches converges faster not because of the pheromones but because of the randomization process," the randomization process can be optimized by importing pre-trained pheromone concertation and keeping optimize based on it [2].

C. Genetic Algorithm

The key to GA to solve knapsack problem is the design of the fitness function. For fitness function, the two most used techniques are as follows: when the total weight exceeds the ability, the individual is directly eliminated; and when the total weight exceeds the capacity, the fitness is set to 1. Therefore, in theory, compared with other meta-heuristic algorithms, GA runs faster, but it is easy to fall into the local optimal solution. However, in this research, we redesign the punishment method. Our test results find that the optimized GA algorithm has higher convergence performance in obtaining near-optimal solutions. Still, at the same time, the algorithm also needs more running time. This shows that part of the algorithm running time is sacrificed due to the redesign of the fitness function. Still, compared with the traditional fitness function, the optimized GA algorithm has a better solving ability. In addition, the choice of algorithm parameters significantly affects the quality of the obtained solutions. In this study, we carried out control variables for the number of crossover nodes, mutation rate, and initial population size and only set the conventional values of variables through experience. In the research process, we found that the change of variables significantly impacts the iteration time, so tuning all the parameters remains an issue for the algorithms.

VI. FUTURE WORK

This section will discuss the current difficulties in solving the knapsack problem with meta-heuristic algorithms and our future research directions. The 0-1 knapsack problem is a classic NP-hard problem; thus, its solving can either be accomplished using exact or heuristic algorithms. An exact algorithm can find the optimal solution, but it is only suitable for small scale, However, meta-heuristic approaches are generally used to solve large-scale problems and can provide a feasible solution within an acceptable computational tie. However, the degree of deviation from the optimal solution may not be predicted in advance for metaheuristics. Therefore, although TABU, QTS, ACO, and GA have obvious time advantages for large-scale input, those methods still face common problems to be solved. One of the most crucial problems is premature convergence that leads to sub-optimal solutions. Our experimental results show that the meta-heuristic algorithms cannot guarantee robust results for every execution unless a tedious tuning process for the parameters must be performed. Therefore, to improve efficiency, we will use machine learning models for the parameters tuning process in the future. In addition, we will further explore how different heuristic algorithms can work together to solve advanced versions of the knapsack problem.

VII. CONCLUSION

Our research explores the performance of ACO, GA, QTS, three meta-heuristic algorithms for solving 0-1 knapsack questions. We observe some attributes under the experimental results. GA has always spent less than 20 iterations on its iterative nature, either getting stuck in a typical local optimum or achieving the global optimum. In addition, the GA computational time at each iteration is more significant than ACO, QTS. In contrast, ACO and

QTS must stop until getting termination conditions. However, their local solution is constantly fluctuating, performed poorly in generating global optimum. Next, the time complexity of QTS is lower as the problem size increases, whereas ACO does the opposite. Moreover, GAs could converge prematurely and become trapped in local optimum regions, ACO and QTS have the more powerful ability in local search, and ACO converges earlier than QTS. That is because QTS puts much computational time into converting individuals to the qubit representation. Overall, GA performs better in generating optimal solutions. ACO performs better when the question size is small, and QTS performs better when the input size increases.

REFERENCES

1. Ali, M. Z., Awad, N. H., Suganthan, P. N., Shatnawi, A. M., & Reynolds, R. G. (2018). An improved class of real-coded Genetic Algorithms for numerical optimization☆. Neurocomputing, 275, 155-166.
2. Alzaqebah, A., & Abu-Shareha, A. A. (2019). Ant colony system algorithm with dynamic pheromone updating for 0/1 knapsack problem. International Journal of Intelligent Systems and Applications, 10(2), 9.
3. Abdel-Basset, M., El-Shahat, D., & Sangaiah, A. K. (2019). A modified nature inspired meta-heuristic whale optimization algorithm for solving 0–1 knapsack problem. International Journal of Machine Learning and Cybernetics, 10(3), 495-514.
4. Al Etawi, N. A., & Aburomman, F. T. 0/1 KNAPSACK PROBLEM: GREEDY VS. DYNAMIC-PROGRAMMING.
5. Braik, M., Sheta, A., & Al-Hiary, H. (2021). A novel meta-heuristic search algorithm for solving optimization problems: capuchin search algorithm. Neural Computing and Applications, 33(7), 2515-2547.
6. Chou, Y. H., Chiu, C. H., & Yang, Y. J. (2011, July). Quantum-inspired tabu search algorithm for solving 0/1 knapsack problems. In Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (pp. 55-56).
7. Deng, W., Xu, J., & Zhao, H. (2019). An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. IEEE access, 7, 20281-20292.
8. Ezugwu, A. E., Pillay, V., Hirasen, D., Sivanarain, K., & Govender, M. (2019). A comparative study of meta-heuristic optimization algorithms for 0–1 knapsack problem: Some initial results. IEEE Access, 7, 43979-44001.
9. Feng, Y. H., & Wang, G. G. (2018). Binary moth search algorithm for discounted {0-1} knapsack problem. IEEE Access, 6, 10708-10719.
10. Fidanova, S. (2020, December). Hybrid Ant Colony Optimization Algorithm for Multiple Knapsack Problem. In 2020 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE) (pp. 1-5). IEEE.
11. Han, K. H., & Kim, J. H. (2002). Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE transactions on evolutionary computation, 6(6), 580-593.
12. Kuo, S. Y., & Chou, Y. H. (2017). Entanglement-enhanced quantum-inspired tabu search algorithm for function optimization. IEEE Access, 5, 13236-13252.
13. Mohammadpour, M., Minaei, B., Parvin, H., & Rahimizadeh, K. (2021). Improved Genetic Algorithm Based on Critical Self-Organization and Gaussian Memory for Solving Dynamic Optimization Problems. Soft Computing Journal, 9(1), 56-91.
14. Resende, M. G., & Ribeiro, C. C. (2019). Greedy randomized adaptive search procedures: Advances and extensions. In Handbook of metaheuristics (pp. 169-220). Springer, Cham.
15. Soukaina, L., Mohamed, N., Hassan, E. A., & Boujemâa, A. (2018, May). A hybrid genetic algorithm for solving 0/1 knapsack problem. In Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications (pp. 1-6).
16. Shu, Z., Ye, Z., Zong, X., Liu, S., Zhang, D., Wang, C., & Wang, M. (2021). A modified hybrid rice optimization algorithm for solving 0-1 knapsack problem. Applied Intelligence, 1-19
17. Venkateswarlu, C. (2021). A Metaheuristic Tabu Search Optimization Algorithm: Applications to Chemical and Environmental Processes.
18. Yang, X., Zhou, Y., Shen, A., Lin, J., & Zhong, Y. (2021, June). A hybrid ant colony optimization algorithm for the knapsack problem with a single continuous variable. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 57-65).
19. Zhang, Y. (2018). Design and application of course scheduling system based on tabu search method. Electronic Design Engineering. Vol. 26 No.16. Aug 2018.
20. Zhang, L., & Lv, J. (2018). A heuristic algorithm based on expectation efficiency for 0-1 knapsack problem. International Journal of Innovative Computing, Information and Control, 14(5), 1833-1854.