

A Binary U-Turning Ant Colony Optimization Algorithm for the 0–1 Knapsack Problem: Parameter Sensitivity and Comparative Evaluation

Saman M. Almufti^{1,*}, Renas Rajab Asaad², Chen Jun³, Yang Hao⁴, Wang Fang⁵, S. Sharan Jeev⁶

^{1,2}Department of Information Technology, Technical College of Informatics-Akre, Akre University for Applied Sciences, Nineveh Governorate, Iraq.

³Department of Engineering, Shanghai Tech Solutions, Shanghai, China.

⁴Department of Engineering, Dragon Peak Electronics, Sichuan, China.

⁵Department of Production, Golden Dragon Manufacturing, Zhejiang, China.

⁶Department of Cybersecurity, University of Texas at Dallas, Richardson, Texas, United States of America.
saman.almufti@gmail.com¹, renas.beda89@gmail.com², chen.jun@shanhaitech.cn³, yang.hao@dragonpeak.com.cn⁴,
wang.fang@goldendragon.com.cn⁵, dal905518@utdallas.edu⁶

Abstract: This paper applies the two-phase U-Turning Ant Colony Optimization (U-TACO) paradigm for the symmetric traveling salesperson problem to the binary 0–1 knapsack problem. The proposed formulation uses capacity-feasible constructive sampling, a normalized profit-to-weight heuristic, partial-subset U-turn reinforcement, elite and global-best pheromone deposits, and feasible add-and-swap improvement. Researchers used 30 diagnostic instance runs and 30 Pisinger instance runs (knapPI_1_100_1000_1) to evaluate eight parameter configurations, for a total of 480 algorithm runs. Each instance's configurations used the same seed set. Researchers verified the optimal diagnostic and benchmark results using exact dynamic programming and the Pisinger reference solution. Each design yielded earnings of 698 and 9,147, with 100% feasibility and no relative percentage variance. Analysis focused on iteration-to-optimal and runtime because the final objective values were equal. S4 had the lowest descriptive composite rank and reached the optimum in 1,500 iterations on the diagnostic instance, while S8 had the lowest mean runtime (0.037051 s) but took 4,933 iterations. Friedman tests demonstrated substantial configuration effects for both measures, and Holm-adjusted Wilcoxon tests identified the main pairwise differences. In the first iteration, all configurations met the Pisinger optimum; therefore, it was mostly an implementation and feasibility check. Direct numerical comparisons were limited to research utilizing the same instance, while a 30-method evidence matrix documented diverse techniques. S4 had the best convergence-cost balance and was kept as the working configuration. Scalability and superiority claims require evaluation across more Pisinger classes and item counts.

Keywords: 0–1 Knapsack Problem; Ant Colony Optimization; Binary Combinatorial Optimization; Parameter Sensitivity; Nonparametric Analysis; Pisinger Benchmark.

Received on: 17/09/2025, **Revised on:** 06/11/2025, **Accepted on:** 07/01/2026, **Published on:** 03/06/2026

Journal Homepage: <https://www.fmdbpublish.com/user/journals/details/FTSCIS>

DOI: <https://doi.org/10.69888/FTSCIS.2026.000763>

Cite as: S. M. Almufti, R. R. Asaad, C. Jun, Y. Hao, W. Fang, and S. S. Jeev, “A Binary U-Turning Ant Colony Optimization Algorithm for the 0–1 Knapsack Problem: Parameter Sensitivity and Comparative Evaluation,” *FMDB Transactions on Sustainable Critical Infrastructures*, vol. 1, no. 2, pp. 73–88, 2026.

Copyright © 2026 S. M. Almufti *et al.*, licensed to Fernando Martins De Bulhão (FMDB) Publishing Company. This is an open access article distributed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/), which permits copying, remixing, redistributing, transformation, and building upon the material in any medium so long as the original work is properly cited.

*Corresponding author.

1. Introduction

The 0–1 knapsack problem (KP) selects a subset of items to maximize total profit, subject to a capacity limit. Although its mathematical statement is compact, instance difficulty varies substantially with the profit-weight correlation, the magnitude of the coefficients, and the capacity ratio. Pisinger showed that these structural properties can be more influential than item count alone [1]. The associated benchmark collections are therefore useful for distinguishing performance on convenient instances from performance across systematically different instance classes [2]. Ant colony optimization (ACO) is well suited to constructive combinatorial search because pheromone values retain collective information about decisions observed in high-quality solutions, while heuristic information guides local selection. A direct transfer from routing to the knapsack problem is nevertheless inadequate.

The representation must encode binary inclusion; capacity feasibility must be preserved or repaired; item desirability must be defined without edge distances; and pheromone concentration must be controlled to avoid reducing the search to a deterministic greedy procedure. U-TACO was originally proposed for the symmetric traveling salesperson problem as a two-phase extension of ACO. A U-turning ant constructs a partial route, returns along the same path, and uses that partial experience to initialize pheromone information before conventional ants construct complete tours [3]; [4]. Subsequent TSPLIB studies compared U-TACO with elephant herding optimization and Tabu Search. U-TACO obtained optimal or near-optimal tours and consistently outperformed Tabu Search in the reported comparisons. In contrast, elephant herding optimization achieved the best result on a larger number of instances [5]; [6]. In the present formulation, a partial route is replaced by a feasible partial item subset, a fraction of capacity represents route length, edge pheromone is replaced by item-selection pheromone, and complete-tour construction becomes feasible subset completion.

Previous studies have addressed the binary KP using continuous ACO [7], modified artificial bee colony search [8], binary honey badger optimization [9], bat and cuckoo-search variants [10]; [11], population-based simulated annealing [12], binary mantis-search and related hybrid methods [13], binary grey wolf optimization [14], binary Aquila optimization [15], quantum-inspired differential evolution [16], binary artificial jellyfish search [17], binary Kepler optimization [18], and transfer-function-based binary Pufferfish and comparator algorithms. These studies use different instance suites, repair rules, transfer functions, evaluation budgets, and reporting conventions; consequently, results obtained under different protocols should not be combined into a single numerical ranking. This study has four objectives. It first defines a capacity-feasible U-TACO formulation for the 0–1 KP. It then evaluates eight parameter configurations using a matched seed set and exact-solution verification. The analysis reports objective statistics, iteration-to-optimum, runtime distributions, Friedman tests, and Holm-adjusted Wilcoxon comparisons. Finally, it separates identical-instance numerical comparisons from a broader evidence matrix covering published methods evaluated under different protocols.

2. Related Work

2.1. Origin and Prior Evaluation of U-TACO

The original U-TACO framework was introduced in a 2015 master's paper as an ACO-based method for the traveling salesperson problem. Its characteristic agent, the U-turning ant (Uant), does not construct a complete tour. It visits a limited number of cities, returns to its starting point along the same partial path, and deposits pheromone that guides the subsequent complete-tour phase. The paper also examined Great Deluge and 2-opt improvement; these procedures belong to the historical TSP implementation and are separate from the partial-tour principle [3]. The subsequent STSP paper described U-TACO as a two-phase procedure. During the first phase, Uants construct partial tours and initialize the pheromone matrix through deposition and evaporation.

During the second phase, conventional ants use the initialized matrix to construct complete tours. Uants therefore provide directional information rather than final solutions [4]. The transition rule retains the Ant System combination of pheromone intensity and heuristic information; the difference between Uants and complete ants lies in construction length and return behavior. Figure 1 presents the original workflow reported in Toaza and Esztergar-Kiss [4]. The TSPLIB95 experiments considered partial-tour lengths equal to one-quarter, one-half, and three-quarters of the number of cities.

Tests on Lin105, Eil51, and Pr107 led the authors to use one-half of the city count in the principal experiments; the reported setting applied 50 Uant iterations for pheromone initialization and 20 complete-ant iterations. In a 17-instance comparison, elephant herding optimization reached the optimum on more instances, while U-TACO matched it on several cases and produced competitive near-optimal tours [5]. A later ten-instance comparison among elephant herding optimization, U-TACO, and Tabu Search again ranked elephant herding optimization first overall. Still, U-TACO remained substantially closer to the TSPLIB optima than Tabu Search across all reported instances [6]. These findings support partial-route initialization as a useful search mechanism, but they do not establish its universal superiority.

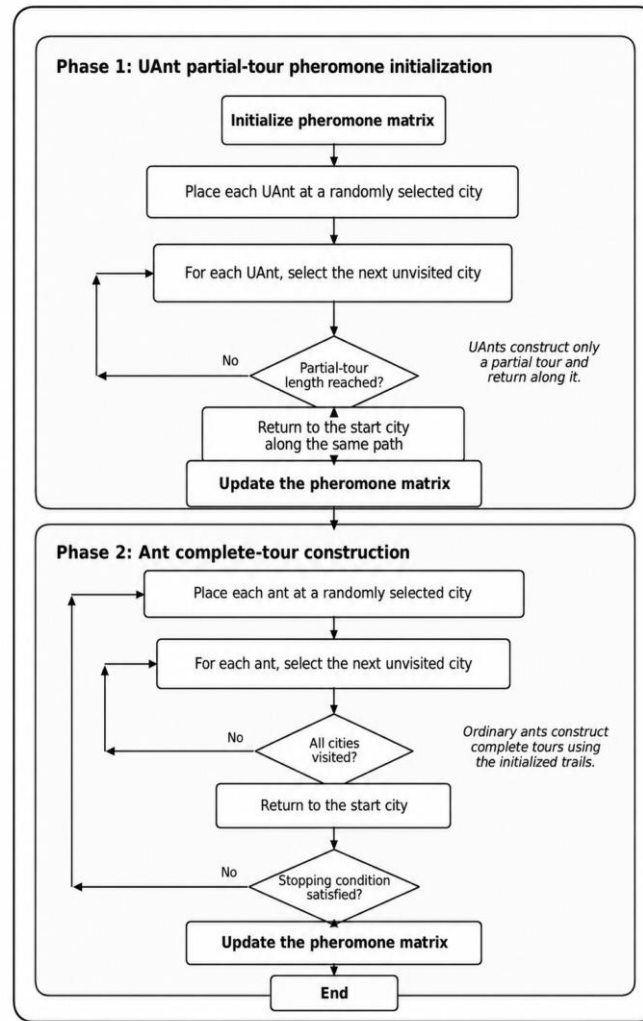


Figure 1: Two-phase structure of the original U-TACO procedure, adapted and redrawn from

2.2. Constructive and Repair-Based Binary Metaheuristics

Binary metaheuristics commonly use constructive generation, continuous-to-binary mapping, or feasibility repair. Constructive methods add feasible items sequentially. Transfer-function methods map continuous positions to binary decisions. Repair-based methods permit infeasible vectors and then remove or replace items to restore feasibility. Continuous ACO for the 0–1 KP follows a constructive formulation [7]. The modified artificial bee colony method introduced problem-specific operations for binary selection [8], and the binary honey badger method adapted its movement mechanism to standard KP instances [9]. Transfer-function approaches retain the movement equations of a continuous optimizer but require an explicit binary mapping. This structure is used in binary grey wolf optimization [14], binary Aquila optimization [15], binary Kepler optimization [18], and comparative implementations of Pufferfish, particle swarm, secretary-bird, and arithmetic optimization. Performance depends jointly on the original movement rule and the selected binarization scheme. Because independent bit decisions may violate capacity, these methods also require a feasibility rule or repair operator.

2.3. Hybridization, Local Improvement, and Adaptive Control

Hybrid binary algorithms often combine population searches with local refinement, repair, or adaptive control. The adaptive genetic algorithm modifies crossover behavior according to search conditions. Hybrid-encoding cuckoo search combines alternative representations with repair [11], while population-based simulated annealing joins trajectory-based acceptance with a population of candidates [12]. Quantum-inspired differential evolution uses a probabilistic representation and grey wolf guidance [16]. Binary Kepler optimization also includes hybrid variants intended to improve diversification and local refinement [18]. These designs seek to limit premature convergence without removing local refinement. U-TACO addresses the same issue through partial-subset credit assignment. A completed subset is used to assess whether its early decisions were

productive, but U-turn pheromone is deposited only on the partial subset. The main colony combines that signal with complete-solution reinforcement and feasible add-and-swap improvement.

2.4. Research Gap

The literature includes many binary KP algorithms, but parameter studies often summarize results only by final objective values. When every configuration reaches the optimum, final-value statistics cannot distinguish their computational behavior. Iteration-to-optimum, runtime variability, and paired nonparametric comparisons provide additional information. Numerical comparison also requires identical item vectors, capacity, objective direction, and stopping conditions. The present study applies these two principles by examining computational measures and by separating direct from protocol-different comparisons.

3. Research Results and Discussion

Let n denote the number of candidate items. Item i has profit p_i and weight w_i , while x_i equals one when item i is selected and zero otherwise. The objective is:

$$\max_{\mathbf{x}} f(\mathbf{x}) = \sum_{i=1}^n p_i x_i \quad (1)$$

subject to the capacity constraint:

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

and the binary domain:

$$x_i \in \{0,1\}, \quad i = 1, 2, \dots, n \quad (3)$$

Feasibility is preserved during construction. An item is eligible only when its inclusion is satisfactory (2). Overweight solutions are therefore not evaluated; the feasibility rate is 100% by construction, and no penalty coefficient is required.

3.1. Proposed U-TACO Method

3.1.1. Binary Representation and Heuristic Information

Each ant constructs a binary vector by selecting items from the current feasible candidate set. Item desirability combines pheromone intensity τ_i with a normalized profit-to-weight heuristic. The heuristic is defined as:

$$\eta_i = \frac{p_i/w_i}{\max_j (p_j/w_j)} + \varepsilon \quad (4)$$

For the currently feasible set F , roulette-wheel selection uses:

$$P_i = \frac{\tau_i^\alpha \eta_i^\beta}{\sum_{j \in F} \tau_j^\alpha \eta_j^\beta}, \quad i \in F \quad (5)$$

where α controls the influence of pheromones and β controls the influence of heuristics, before roulette-wheel sampling, a bounded multiplicative perturbation is applied to reduce deterministic tie behavior while preserving the relative information supplied by pheromone and item efficiency.

3.1.2. U-Turning Reinforcement

The method preserves the defining separation of the original U-TACO formulation: a UA_{nt} phase derives pheromone guidance from partial constructions, and a main-ant phase constructs complete candidate solutions [3]; [4]. For the knapsack problem, a UA_{nt} builds a feasible partial subset until its accumulated weight reaches a specified fraction of capacity. The subset then completes the solution using the same constructive rule and refines it with local search. The completed profit assesses the

quality of the early decisions, but the U-turn deposit is assigned only to items selected before the partial-capacity threshold. This separates early-subset assessment from complete-solution reinforcement. The binary formulation does not use Great Deluge or 2-opt; it uses capacity-preserving additions and one-for-one exchanges because they operate directly on item subsets.

3.1.3. Local Improvement and Pheromone Update

Each complete subset is improved by repeated feasible additions, followed by the best improvement one-for-one exchange. The main colony is ranked by objective value, an elite subset deposits quality-normalized pheromone, and the global-best solution receives an additional deposit. Evaporation and pheromone truncation are as follows:

$$\tau_i \leftarrow \min\{\tau_{\max}, \max[\tau_{\min}, (1 - \rho)\tau_i + \Delta\tau_i]\} \quad (6)$$

where ρ is the evaporation rate, and $\Delta\tau_i$ is the accumulated deposit. The limits $\tau_{\min}$ and $\tau_{\max}$ prevent complete loss of an item and excessive pheromone concentration. The search stops at the maximum iteration count, when the verified target is attained, or when an optional stagnation criterion is met. The target is used only for early termination and does not enter construction, local search, or pheromone calculation.

3.2. Algorithmic Procedure

3.2.1. Algorithm 1: U-TACO for the 0–1 knapsack problem

- Input item profits, item weights, capacity, parameter set, stopping criteria, and random seed.
- Initialize item pheromones within $[\tau_{\min}, \tau_{\max}]$ and compute normalized profit-to-weight heuristics.
- Construct U-turning partial subsets while maintaining feasibility.
- Complete and locally improve each partial subset; evaluate its profit.
- Deposit U-turning pheromone only on items selected before the partial-capacity threshold.
- Construct the main colony of complete feasible subsets by roulette-wheel sampling.
- Apply feasible addition and best one-for-one exchange for the specified number of local-search passes.
- Rank the colony and update the incumbent global best subset.
- Evaporate pheromone; add elite and global-best deposits; truncate pheromones to their bounds.
- Record the best-so-far objective and the first iteration at which the verified optimum is reached.
- Terminate when a stopping condition is satisfied; otherwise return to Step 3.
- Output the best binary vector, profit, weight, convergence record, and runtime.

4. Experimental Design

4.1. Instances and Exact Verification

Two instances were used. The first is a synthetic 30-item diagnostic instance intended to expose differences in convergence and computational effort. The second, knapPI_1_100_1000_1, is a 100-item instance from the Pisinger collection with a published reference solution. Exact dynamic programming independently verified the diagnostic optimum, while the benchmark solution verified the Pisinger objective [1]; [2] (Table 1).

Table 1: Test instances and verified reference solutions

Instance	Items	Capacity	Optimum	Optimum Weight	Purpose
Diagnostic instance	30	400	698	400	Parameter discrimination
knapPI 1 100 1000 1	100	995	9,147	985	Published benchmark validation

The diagnostic optimum selects items {1, 2, 3, 4, 15, 17, 20, 21, 24, 27, 30}. The Pisinger optimum selects items {7, 11, 14, 24, 26, 31, 33, 38, 39, 49, 54, 61}.

4.2. Parameter Configurations

Table 2 presents eight possible parameter sets (S1-S8). These parameters include the number of ants, maximum iterations, α , β , ρ , and elite fraction. Each set is intentionally crafted to match a specific search approach, such as cautious, balanced, explorative, exploitative, or computationally lean.

Table 2: Main colony and pheromone parameters

Set	Ants	Max. iter.	α	β	ρ	Elite Fraction
S1 Conservative	30	200	0.8	1.5	0.1	0.15
S2 Balanced	50	300	1.0	2.0	0.2	0.2
S3 Explorative	60	350	0.8	1.5	0.3	0.2
S4 Exploitative	50	300	1.5	3.0	0.15	0.25
S5 High evaporation	50	300	1.0	2.0	0.4	0.2
S6 Strong U-turn	50	300	1.0	2.0	0.2	0.2
S7 Large colony	80	400	1.0	2.0	0.2	0.2
S8 Computationally lean	25	150	1.0	2.0	0.25	0.2

These variants let you evaluate how different parameter values affect the algorithm's overall performance and the effectiveness of the optimization process (Table 3).

Table 3: U-turning and local-improvement parameters

Set	Global-Best Weight	U-Turn Ants	Partial-Capacity Ratio	U-Turn Weight	Local-Search Passes
S1 Conservative	1.5	15	0.4	0.1	2
S2 Balanced	2.0	25	0.5	0.15	3
S3 Explorative	1.5	35	0.4	0.2	3
S4 Exploitative	3.0	25	0.55	0.15	4
S5 High evaporation	1.5	25	0.5	0.15	3
S6 Strong U-turn	2.0	40	0.6	0.3	3
S7 Large colony	2.0	40	0.5	0.15	3
S8 Computationally lean	2.0	12	0.5	0.15	1

The configurations represent conservative search (S1), a balanced setting (S2), stronger exploration (S3), stronger exploitation (S4), high evaporation (S5), stronger U-turn reinforcement (S6), a larger colony (S7), and a computationally lean setting (S8). All configurations used $Q = 1$, $\tau_{\min} = 0.05$, and $\tau_{\max} = 20$.

4.3. Repetition Policy and Evaluation Measures

Researchers ran each configuration 30 times on each instance, using seeds 1–30, for 480 runs. Researchers used the same seed labels for all configurations on a given instance, allowing us to treat the run index as a block in the repeated-measures analysis. Reported measures include best, mean, median, worst, standard deviation, mean relative percentage deviation (RPD), success rate, feasibility rate, iteration-to-optimum, and runtime. A descriptive composite rank was calculated as the mean of the objective, runtime, and convergence ranks; it is used only as a summary measure. RPD is defined as:

$$RPD = 100 \times \frac{f_{\text{opt}} - f_r}{f_{\text{opt}}} \quad (7)$$

where f_{opt} is the verified optimum and f_r is the objective obtained in run r . The success rate is:

$$\text{Success rate}(\%) = 100 \times \frac{N_{\text{opt}}}{R} \quad (8)$$

where N_{opt} is the number of runs attaining the verified optimum and $R = 30$; the Friedman test was applied to iteration-to-optimum and runtime, with run index used as the blocking factor. Significant omnibus results were followed by Wilcoxon signed-rank tests with Holm correction. An objective-value test is undefined when all observations are identical.

4.4. Execution and Reproducibility

The supplementary archive contains the MATLAB implementation, the eight-configuration driver, the exact dynamic-programming verifier, the Pisinger parser, the benchmark file, the run-level data, the summary Tables 2 and 3, and the Figure 1 outputs. Researchers produced the precomputed Tables 2 and 3 using an independent validation implementation that mirrors

the MATLAB construction, local search, pheromone update, seed, and stopping rules, as a licensed MATLAB runtime was unavailable in the execution environment used to generate the results.

Researchers checked objective values, selected subsets, feasibility status, and iteration-to-target against exact or published reference solutions. Runtime is interpreted only as a within-environment measure and should be regenerated in MATLAB on the hardware used for final reporting.

5. Results and Discussion

5.1. Diagnostic-Instance Solution Quality

All 240 diagnostic runs achieved the exact profit of 698 with a total weight of 400. For every configuration, the best, mean, median, and worst objectives were 698; standard deviation and RPD were zero; and success and feasibility were 100% (Table 4).

Table 4: Diagnostic-instance results over 30 runs per configuration

	Mean	Std.	RPD (%)	Success (%)	Mean Iter.	Max. Iter.	Mean Time (S)	Composite Rank
S1 Conservative	698	0	0.000	100	4.633	11	0.055634	7.0
S2 Balanced	698	0	0.000	100	1.833	3	0.045337	4.0
S3 Explorative	698	0	0.000	100	1.933	3	0.064453	8.0
S4 Exploitative	698	0	0.000	100	1.500	2	0.041864	1.0
S5 High evaporation	698	0	0.000	100	1.800	4	0.044823	2.0
S6 Strong U-turn	698	0	0.000	100	1.800	3	0.046936	3.0
S7 Large colony	698	0	0.000	100	1.700	3	0.057409	4.0
S8 Computationally lean	698	0	0.000	100	4.933	11	0.037051	4.0

Final objective values therefore do not discriminate among the configurations in this instance. The remaining analysis considers convergence speed and computational effort (Appendix A).

5.2. Convergence Behavior

S4 reached the optimum within two iterations in every run. S2, S3, S6, and S7 reached it within three iterations, whereas S1 and S8 required up to 11 iterations (Figure 2).

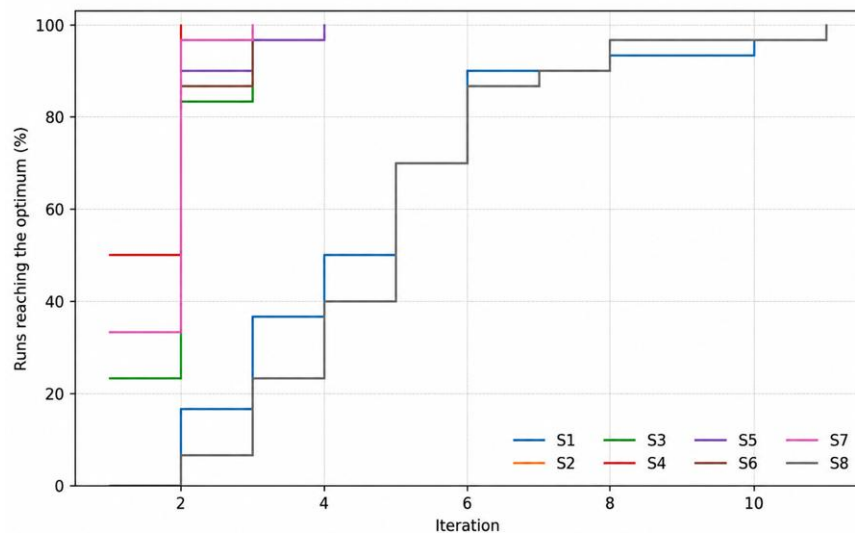


Figure 2: Empirical cumulative percentage of diagnostic runs attaining the optimum by iteration

This pattern is consistent with the smaller colonies and lower local search effort observed in the leaner settings, although the experiment does not isolate the effect of any single parameter (Figure 3).

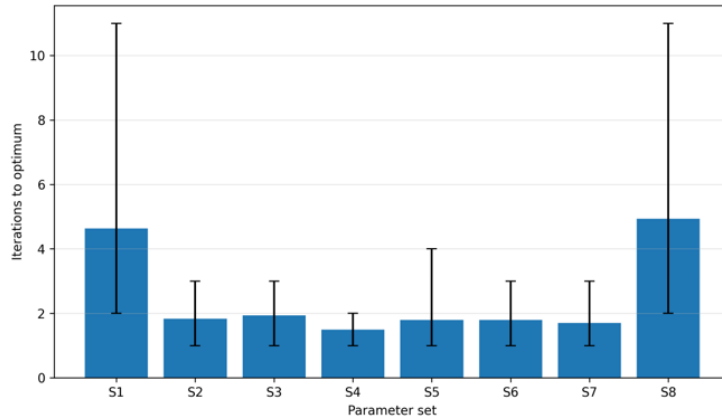


Figure 3: Mean iteration-to-optimum with the observed minimum-to-maximum range

S4 had the lowest mean iteration-to-optimum (1.500). S7 followed at 1.700, while S5 and S6 each required 1.800 iterations. S4 combines relatively strong pheromone and heuristic influence, stronger global-best reinforcement, and four local-search passes. These settings favor rapid refinement once promising subsets have been identified.

5.3. Runtime and Efficiency Trade-off

S8 had the lowest mean runtime (0.037051 s), while S3 had the highest (0.064453 s). S4 averaged 0.041864 s and also had the lowest iteration-to-optimum. Runtime increased when researchers completed additional iterations before target-based termination (Figure 4).

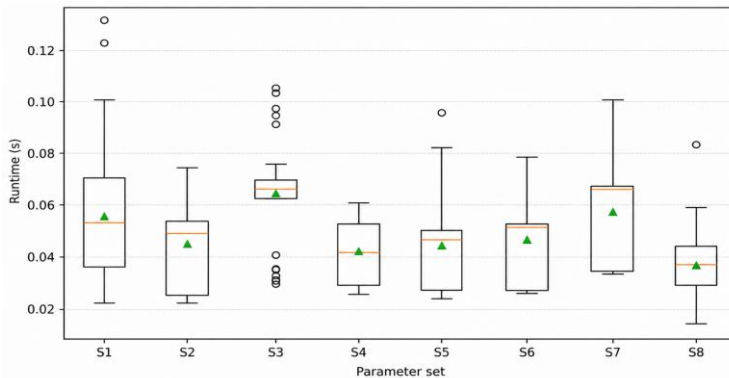


Figure 4: Runtime distributions for the diagnostic instance

Figure 5 shows the trade-off between mean runtime and mean iteration-to-optimum.

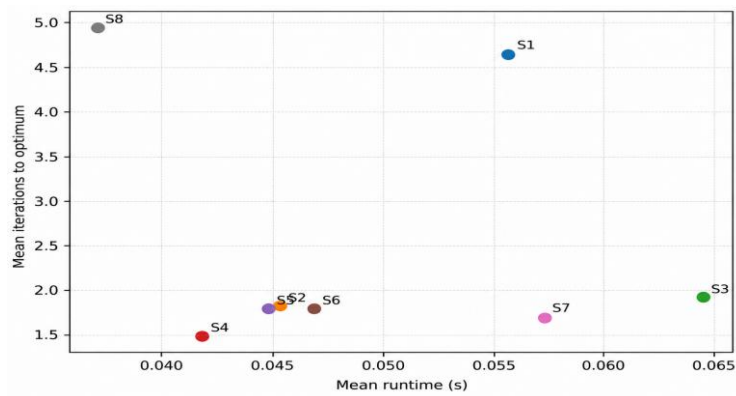


Figure 5: Mean runtime versus mean iteration-to-optimum on the diagnostic instance

S4 combines the lowest iteration count with the second-lowest mean runtime. S8 is faster in absolute runtime but requires substantially more iterations. The larger colony or stronger exploration settings used by S3 and S7 increased computational work without improving final accuracy on this instance (Figure 6).

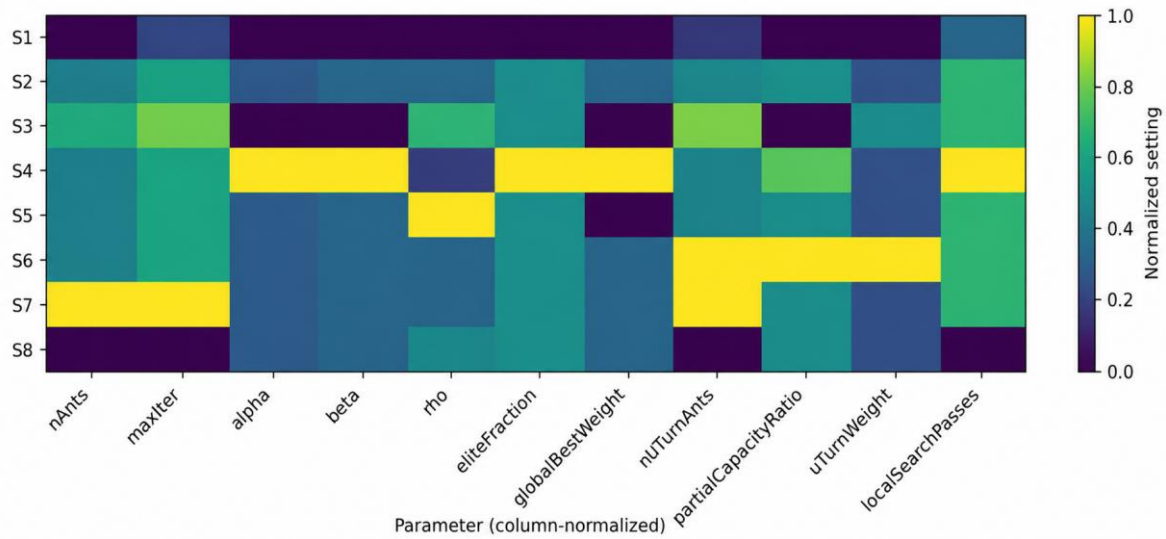


Figure 6: Column-normalized parameter values for the eight configurations

No single parameter explains the full ranking because several settings change simultaneously. S4 combines strong heuristic and pheromone influence, high global-best reinforcement, and the greatest local-search effort. S8 reduces colony size, U-turning effort, iteration budget, and local-search passes. The resulting profiles match the observed convergence-runtime trade-off.

5.4. Nonparametric Statistical Analysis

The Friedman tests detected significant differences among configurations for iteration-to-optimum (Friedman $\chi^2 = 119.213$, $p = 1.12 \times 10^{-22}$) and runtime (Friedman $\chi^2 = 43.467$, $p = 2.71 \times 10^{-7}$). A final-objective test could not be performed because every observation equaled 698 (Table 5).

Table 5: Friedman tests across the eight configurations

Metric	$\chi^2 F$	df.	p-value	Interpretation
Iterations to Optimum	119.213	7	1.118E-22	Significant differences among configurations
Runtime	43.467	7	2.710E-07	Significant differences among configurations
Final Objective	—	7	—	All observations equal the optimum; an objective-value test is undefined.

Relative to S4, S1 and S8 required significantly more iterations, with rank-biserial effect sizes of 1.000. Relative to S8, runtimes were significantly larger for S1, S3, S6, and S7 (Table 6).

Table 6: Significant Holm-adjusted Wilcoxon comparisons

Metric	Reference	Comparator	Median Excess Over Reference	Rank-Biserial Effect Size	Holm-Adjusted p
Iterations	S4	S1	2.500	1.000	1.205E-05
Iterations	S4	S8	3.000	1.000	1.167E-05
Runtime	S8	S1	0.014804 s	0.630	9.320E-03
Runtime	S8	S3	0.029607 s	0.841	7.427E-05
Runtime	S8	S6	0.008337 s	0.578	1.866E-02
Runtime	S8	S7	0.026158 s	0.725	1.538E-03

Differences between S8 and S2, S4, or S5 were not significant after Holm correction; therefore, the observed mean runtime advantage of S8 was not uniform across all paired runs.

5.5. Validation on the Pisinger Instance

All 240 runs on knapPI_1_100_1000_1 attained the reference objective of 9,147 in the first iteration. Every configuration returned the same 12-item subset with total weight 985, leaving 10 units of unused capacity. The standard deviation and RPD were 0, and both success and feasibility were 100%. This result provides a consistency check for objective calculation, feasibility handling, and benchmark parsing, but it offers little discrimination among parameter configurations (Table 7).

Table 7: Results on knapPI_1_100_1000_1 over 30 runs per configuration

Set	Mean	Std.	RPD (%)	Success (%)	First Optimum Iteration	Mean Time (S)	Time CV (%)
S1 Conservative	9147	0	0.000	100	1	0.011408	10.19
S2 Balanced	9147	0	0.000	100	1	0.023945	7.50
S3 Explorative	9147	0	0.000	100	1	0.032393	8.12
S4 Exploitative	9147	0	0.000	100	1	0.026212	7.79
S5 High evaporation	9147	0	0.000	100	1	0.022214	10.07
S6 Strong U-turn	9147	0	0.000	100	1	0.026128	6.29
S7 Large colony	9147	0	0.000	100	1	0.035861	7.73
S8 Computationally lean	9147	0	0.000	100	1	0.008488	11.83

Because every configuration terminated after the first iteration, runtime mainly reflects the colony size, the effort required to U-turn, and the local search work. S8 was fastest at 0.008488 s, followed by S1 at 0.011408 s, while S7 was slowest at 0.035861 s (Figure 7).

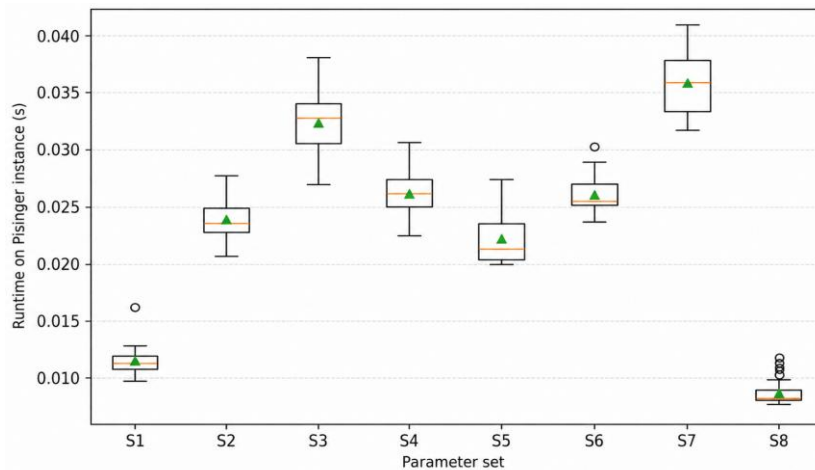


Figure 7: Runtime distributions on knapPI_1_100_1000_1

This result does not establish S8's general superiority; it only indicates that the lean setting performs less redundant work when the reference objective is reached immediately.

5.6. Comparison with Existing Algorithms

5.6.1. Common-Instance Numerical Comparison

A direct numerical comparison requires identical item profits, weights, capacity, objective direction, and feasibility definition. Table 8 therefore compares U-TACO with results reported for knapPI_1_100_1000_1. Crawford et al. reported 31-run statistics for four algorithms under two binarization configurations. Erdoğan reported the best outcome from ten runs for three fixed-crossover genetic algorithms and one adaptive genetic algorithm.

Table 8: Direct comparison on knapPI_1_100_1000_1

Algorithm/Configuration	Best	Mean	Worst	Std.	RPD of Mean (%)	Runs
-------------------------	------	------	-------	------	-----------------	------

U-TACO S1-S8 (this study)	9,147	9,147.000	9,147.000	0.000	0.000	30 per set
POA, S1-STD	9,147	9,147.000	9,147.000	0.000	0.000	31
PSO, S1-STD	9,147	9,147.000	9,147.000	0.000	0.000	31
SBOA, S1-STD	9,147	9,147.000	9,147.000	0.000	0.000	31
AOA, S1-STD	9,147	9,147.000	9,147.000	0.000	0.000	31
POA, V1-STD	9,147	9,012.935	8,817.000	140.421	1.466	31
PSO, V1-STD	9,147	8,806.355	8,382.000	211.194	3.723	31
SBOA, V1-STD	9,147	9,004.161	8,817.000	159.063	1.562	31
AOA, V1-STD	9,147	8,769.613	8,259.000	219.035	4.126	31
GA with single-point crossover	9,147	n.r.	n.r.	n.r.	n.r.	10
GA with two-point crossover	9,147	n.r.	n.r.	n.r.	n.r.	10
GA with uniform crossover	9,147	n.r.	n.r.	n.r.	n.r.	10
Adaptive genetic algorithm	9,147	n.r.	n.r.	n.r.	n.r.	10

Because the latter study does not report corresponding means, researchers treat its entries as best-value evidence rather than mean-performance equivalents.

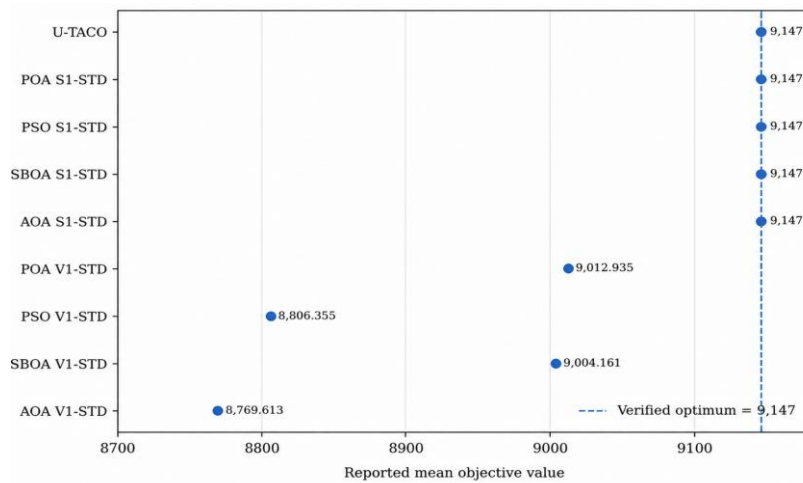


Figure 8: Reported mean objective values on knapPI_1_100_1000_1; the dashed line denotes the verified optimum

Figures 8 and 9 include only methods for which a mean objective is available. Runtime is excluded because hardware, implementation language, and stopping conditions differ.

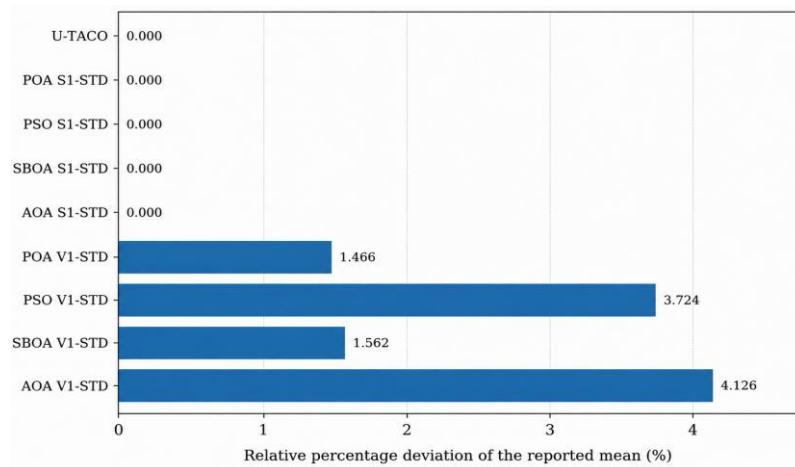


Figure 9: RPD of the reported mean objective values from the verified optimum of 9,147

The mean objective of U-TACO and each S1-STD configuration equaled 9,147. The V1-STD configurations reached 9,147 in their best runs but had lower means and nonzero mean RPD. The genetic-algorithm variants also reported a best value of 9,147, although they did not report mean performance. The common-instance evidence therefore places U-TACO among the zero-deviation mean results for this instance. This conclusion is instance-specific and should not be extended to the complete Pisinger suite.

5.6.2. Evidence Matrix For 30 Published Algorithms and Configurations

The evidence matrix shows three recurrent design choices. Transfer functions convert continuous movement into binary decisions, repair operators restore feasibility after generating overweight solutions, and local or adaptive operators increase refinement (Table 9).

Table 9: Published knapsack algorithms and configurations

No.	Algorithm/Configuration	Class	Benchmark	Published Outcome	Comparability	Ref.
1	Pufferfish Optimization Algorithm, S1-STD	Binary swarm	knapPI_1_1 00_1000_1	Mean 9147; standard deviation 0; RPD 0%	Direct numerical	[18]
2	Particle Swarm Optimization, S1-STD	Binary swarm	knapPI_1_1 00_1000_1	Mean 9147; standard deviation 0; RPD 0%	Direct numerical	[18]
3	Secretary Bird Optimization Algorithm, S1-STD	Binary swarm	knapPI_1_1 00_1000_1	Mean 9147; standard deviation 0; RPD 0%	Direct numerical	[18]
4	Arithmetic Optimization Algorithm, S1-STD	Binary arithmetic	knapPI_1_1 00_1000_1	Mean 9147; standard deviation 0; RPD 0%	Direct numerical	[18]
5	Pufferfish Optimization Algorithm, V1-STD	Binary swarm	knapPI_1_1 00_1000_1	Mean 9012.935; RPD 1.466%	Direct numerical	[18]
6	Particle Swarm Optimization, V1-STD	Binary swarm	knapPI_1_1 00_1000_1	Mean 8806.355; RPD 3.723%	Direct numerical	[18]
7	Secretary Bird Optimization Algorithm, V1-STD	Binary swarm	knapPI_1_1 00_1000_1	Mean 9004.161; RPD 1.562%	Direct numerical	[18]
8	Arithmetic Optimization Algorithm, V1-STD	Binary arithmetic	knapPI_1_1 00_1000_1	Mean 8769.613; RPD 4.126%	Direct numerical	[18]
9	GA with single-point crossover	Evolutionary	knapPI_1_1 00_1000_1	Best of 10 runs = 9147	Direct best-value	[16]
10	GA with two-point crossover	Evolutionary	knapPI_1_1 00_1000_1	Best of 10 runs = 9147	Direct best-value	[16]
11	GA with uniform crossover	Evolutionary	knapPI_1_1 00_1000_1	Best of 10 runs = 9147	Direct best-value	[16]
12	Adaptive Genetic Algorithm	Evolutionary	knapPI_1_1 00_1000_1	Best of 10 runs = 9147	Direct best-value	[16]
13	Continuous Ant Colony Optimization	Constructive swarm	0–1 KP benchmark suite	Dedicated ACO formulation evaluated on KP instances	Protocol differs	[7]
14	Modified Artificial Bee Colony	Swarm	0–1 KP benchmark suite	Modified ABC evaluated on low- and high-dimensional KP instances	Protocol differs	[8]
15	Binary Honey Badger Algorithm	Binary swarm	27 KP instances	Evaluated on 15 small and 12 large instances	Protocol differs	[9]
16	Phase Angle-Modulated Bat Algorithm	Binary swarm	12 KP instances	Reported improvements in optimization ability and execution time	Protocol differs	[10]
17	Improved Hybrid-Encoding Cuckoo Search	Binary swarm	0–1 KP benchmark suite	Hybrid encoding and repair evaluated for KP	Protocol differs	[11]

18	Population-Based Simulated Annealing	Trajectory/population	0–1 KP benchmark suite	Population-based SA evaluated for KP	Protocol differs	[12]
19	Binary Mantis Search Algorithm	Binary swarm	KP01 and MKP suites	Average fitness, Friedman rank, and cost reported	Protocol differs	[13]
20	Binary Quadratic Interpolation Optimization	Binary mathematical	KP01 and MKP suites	Average fitness, Friedman rank, and cost reported	Protocol differs	[13]
21	Binary Differential Evolution	Evolutionary	KP01 and MKP suites	Average fitness, Friedman rank, and cost reported	Protocol differs	[13]
22	Hybrid Mantis Search Algorithm with repair	Hybrid binary swarm	KP01 and MKP suites	Repair-enhanced variant evaluated on medium and large instances	Protocol differs	[13]
23	Hybrid Quadratic Interpolation Optimization	Hybrid binary mathematical	KP01 and MKP suites	Reported as the strongest alternative among the proposed variants	Protocol differs	[13]
24	Hybrid Differential Evolution with repair	Hybrid evolutionary	KP01 and MKP suites	Repair-enhanced variant evaluated on medium and large instances	Protocol differs	[13]
25	Binary Grey Wolf Optimizer	Binary swarm	Multidimensional KP suite	Dedicated binary GWO evaluated for knapsack	Protocol differs	[14]
26	Binary Aquila Optimizer	Binary swarm	0–1 KP benchmark suite	Dedicated BAO evaluated for 0–1 KP	Protocol differs	[15]
27	Quantum-Inspired DE with Grey Wolf Optimizer	Quantum/evolutionary hybrid	High-dimensional 0–1 KP	Compared with QEA and discrete DE	Protocol differs	[16]
28	Binary Artificial Jellyfish Search	Binary swarm	40 KP instances	Reached the optimum in 97.5% of reported problems	Protocol differs	[17]
29	Binary Kepler Optimization Algorithm	Binary swarm	20 KP instances	Compared on instances with 4-75 variables	Protocol differs	[18]
30	Hybrid Binary Kepler Optimization Algorithm	Hybrid binary swarm	20 KP instances	Reported improvement over BKO and competing optimizers	Protocol differs	[18]

U-TACO is primarily constructive: it maintains feasibility during generation and assigns pheromone credit to early feasible subsets. Its local search uses feasible additions and exchanges rather than post hoc repair

5.7. Parameter Selection and Interpretation

Under the tested protocol, S4 provided the most favorable balance between convergence and runtime. It had the lowest mean iteration-to-optimum, reached the target within two iterations in every diagnostic run, and had the second-lowest mean runtime. After Holm correction, its runtime did not differ significantly from that of S8. S4 is therefore selected as the working configuration for subsequent experiments, but this choice remains provisional because the present instances provide limited difficulty variation. The partial-capacity ratio can be interpreted in relation to the original STSP studies. Earlier U-TACO experiments found that a half-tour often balanced insufficient pheromone information against excessive commitment to a nearly complete route [3]; [4].

S4 uses a partial-capacity ratio of 0.55, which is close to that historical setting. Tour length and capacity fraction represent different structures, however, and the ratio must be retested across additional knapsack classes. The selected working values are 50 ants, 300 maximum iterations, $\alpha = 1.5$, $\beta = 3.0$, $\rho = 0.15$, elite fraction = 0.25, global-best weight = 3.0, 25 U-turn ants, partial-capacity ratio = 0.55, U-turn weight = 0.15, and four local-search passes. S8 remains a lower-cost alternative, for instance, with a strong greedy structure. S3 and S7 increased computational effort without improving accuracy in the present experiments.

6. Conclusion

This paper formulated a binary U-TACO method for the 0–1 knapsack problem and evaluated eight parameter configurations over 480 runs. The adaptation preserves the original separation between partial UAnt guidance and complete-ant construction, while replacing TSP routes, edge pheromone, Great Deluge, and 2-opt with feasible item construction, item-level pheromone, and add-and-swap refinement. Researchers maintain feasibility throughout construction and assess partial-subset quality by completion before assigning pheromone credit to early decisions. Every configuration attained the verified objective on both instances, with zero RPD, zero objective variance, and 100% feasibility. The diagnostic instance nevertheless revealed significant differences in computational behavior. S4 reached the optimum after 1.500 iterations on average and provided the best convergence-cost balance, whereas S8 had the lowest mean runtime but required 4.933 iterations. On knapPI_1_100_1000_1, all configurations reproduced the reference objective of 9,147 and weight of 985 in the first iteration. The common-instance comparison places U-TACO among the methods reporting a zero-deviation mean on the selected Pisinger instance. This evidence does not support a general superiority claim. Researchers retain S4 as the working configuration for subsequent experiments, and S8 as a lower-cost alternative when per-run computation is prioritized. Conclusions concerning scalability and comparative performance require a broader set of Pisinger instances and controlled ablation studies.

6.1. Limitations and Further Evaluation

The study has five principal limitations. First, the 30-item diagnostic instance is small, and all configurations eventually attain its optimum; it distinguishes convergence behavior but does not establish scalability. Second, every configuration solves the selected 100-item Pisinger instance in the first iteration and therefore serves mainly as a correctness check. Third, the eight configurations alter several parameters simultaneously, so the experiment is a sensitivity comparison rather than a factorial analysis. Fourth, runtime comes from an independent validation implementation and is meaningful only in that execution environment. Fifth, direct external comparison is possible only for studies reporting the identical instance and compatible statistics. Further evaluation should include multiple Pisinger correlation classes and item counts of 100, 200, 500, 1,000, and larger. Report a fixed evaluation budget alongside target-based early stopping, and use an exact solver to verify each reference objective. Ablation experiments should isolate U-turn reinforcement, global-best reinforcement, and add-and-swap improvement. A factorial or response-surface design would also provide a more defensible account of parameter interactions.

Appendix A. Complete Diagnostic Iteration Results

Table A1 gives the number of iterations to the optimum over 30 independent runs for each of the eight parameter sets (S1-S8). The results show that most configurations, especially S2-S7, generally find the optimum in fewer iterations. In comparison, S1 and S8 exhibit greater variation, implying less consistent convergence behavior.

Table A1: Iteration at which the optimum was first reached in each diagnostic run

Run	S1	S2	S3	S4	S5	S6	S7	S8
1	2	1	3	2	1	2	2	4
2	2	3	3	1	2	2	1	6
3	6	2	2	2	2	2	2	5
4	11	3	2	2	2	1	2	3
5	5	2	2	1	2	1	2	5
6	6	1	2	1	1	3	2	5
7	4	2	2	2	2	1	2	4
8	2	3	2	1	3	3	2	6
9	5	1	1	2	1	1	2	8
10	5	1	1	2	1	2	1	6
11	3	2	2	1	2	1	1	3
12	3	2	2	1	2	3	2	5
13	4	3	3	2	2	2	2	6
14	8	2	2	2	2	2	2	5
15	3	2	1	1	2	1	1	5
16	4	2	2	2	2	2	2	2
17	6	2	1	2	3	2	2	4
18	3	2	2	1	2	2	1	3
19	4	1	1	2	1	2	1	5

20	5	2	2	1	2	2	1	8
21	6	1	1	2	1	1	1	4
22	5	1	2	1	1	2	2	6
23	3	1	3	1	1	2	2	7
24	3	2	2	1	2	2	2	3
25	2	3	2	2	2	1	2	3
26	6	2	3	1	4	2	2	4
27	6	1	1	1	1	1	1	5
28	10	2	2	2	2	2	3	5
29	5	1	2	1	1	1	2	2
30	2	2	2	2	2	3	1	11

Appendix A.1: Complete Diagnostic Runtime Results

Table A2 shows runtime values obtained from 30 independent runs for eight parameter sets (S1–S8). The results indicate that computation time varies considerably by configuration and run. The runtime statistics provide a basis for assessing and comparing the computational efficiency and consistency of each parameter set.

Table A2: Runtime in seconds for each diagnostic run

Run	S1	S2	S3	S4	S5	S6	S7	S8
1	0.024209	0.023246	0.091424	0.052499	0.029532	0.054158	0.065339	0.029837
2	0.024207	0.074594	0.103425	0.026030	0.046870	0.053228	0.033527	0.044526
3	0.072228	0.049146	0.069980	0.052708	0.052737	0.051982	0.067683	0.037913
4	0.131911	0.068377	0.073893	0.048990	0.055114	0.026446	0.067010	0.022692
5	0.057379	0.049382	0.064771	0.025654	0.046988	0.027368	0.069398	0.036998
6	0.068429	0.023087	0.062490	0.031875	0.027533	0.078358	0.067181	0.037183
7	0.045831	0.048590	0.075684	0.058000	0.047545	0.027240	0.066264	0.028892
8	0.022334	0.068025	0.062507	0.029082	0.082231	0.078168	0.065559	0.044415
9	0.056478	0.026098	0.031058	0.054929	0.027543	0.027315	0.066402	0.058742
10	0.057254	0.024292	0.032390	0.056470	0.025507	0.052593	0.034257	0.052231
11	0.035117	0.049915	0.068349	0.032276	0.053284	0.026096	0.034382	0.025702
12	0.037926	0.048668	0.067723	0.029105	0.047860	0.077332	0.065454	0.037057
13	0.047321	0.072616	0.097366	0.058354	0.046263	0.051645	0.066195	0.043750
14	0.100729	0.052370	0.064421	0.052784	0.048134	0.051658	0.066989	0.037774
15	0.039552	0.053576	0.029915	0.028794	0.051996	0.026369	0.033979	0.037433
16	0.049851	0.050073	0.063438	0.054472	0.046946	0.052796	0.067085	0.015139
17	0.071208	0.057812	0.030876	0.051855	0.073759	0.053482	0.066238	0.029693
18	0.035546	0.049243	0.066353	0.033169	0.049584	0.050637	0.034335	0.022834
19	0.046921	0.025459	0.035339	0.055897	0.024507	0.052448	0.034501	0.037232
20	0.060813	0.048596	0.062229	0.032607	0.048966	0.051239	0.034497	0.059058
21	0.072331	0.023691	0.034592	0.052183	0.026833	0.026421	0.034164	0.029428
22	0.062218	0.023455	0.065631	0.030450	0.026491	0.052397	0.068346	0.044426
23	0.035215	0.023568	0.094991	0.029224	0.024765	0.050411	0.068818	0.050989
24	0.039165	0.050506	0.068370	0.034587	0.050475	0.052022	0.066810	0.022270
25	0.023285	0.071195	0.067974	0.060482	0.045270	0.026445	0.067326	0.022189
26	0.071078	0.053962	0.105255	0.027421	0.095933	0.051562	0.069588	0.029516
27	0.072823	0.025979	0.040466	0.027769	0.026602	0.026358	0.035964	0.038325
28	0.122832	0.054686	0.066259	0.049066	0.044975	0.050198	0.100591	0.037712
29	0.060851	0.022441	0.067757	0.029719	0.024219	0.026202	0.068987	0.014759
30	0.023988	0.047458	0.068653	0.049473	0.046219	0.075501	0.035406	0.082828

Acknowledgement: The authors gratefully acknowledge Technical College of Informatics-Akre, Shanghai Tech Solutions, Dragon Peak Electronics, Golden Dragon Manufacturing, and the University of Texas at Dallas for their valuable support, guidance, research resources, and institutional assistance.

Data Availability Statement: The data for this study are available upon request from the corresponding author.

Funding Statement: This manuscript and research paper were prepared without any financial support or funding.

Conflicts of Interest Statement: The authors have no conflicts of interest to declare. This work represents a new contribution by the authors, and all citations and references are appropriately included based on the information utilized.

Ethics and Consent Statement: This research adheres to ethical guidelines and obtains informed consent from all participants. The authors implemented confidentiality measures to safeguard participant privacy.

References

1. D. Pisinger, "Where are the hard knapsack problems?" *Comput. Oper. Res.*, vol. 32, no. 9, pp. 2271–2284, 2005.
2. D. Pisinger, "David Pisinger's Optimization Codes," *Department of Computer Science, University of Copenhagen*, 2026. [Accessed by 11/04/2026].
3. Y. Wang and Z. Han, "Ant colony optimization for traveling salesman problem based on parameters optimization," *Applied Soft Computing*, vol. 107, no. 8, p. 107439, 2021.
4. B. Toaza and D. Esztergár-Kiss, "A review of metaheuristic algorithms for solving TSP-based scheduling optimization problems," *Applied Soft Computing*, vol. 148, no. 11, pp. 1–24, 2023.
5. C. Changdar, G. Mahapatra, and R. K. Pal, "Solving 0–1 knapsack problem by continuous ACO algorithm," *Int. J. Comput. Intell. Stud.*, vol. 2, no. 3–4, pp. 333–349, 2013.
6. J. Cao, B. Yin, X. Lu, Y. Kang, and X. Chen, "A modified artificial bee colony approach for the 0–1 knapsack problem," *Appl. Intell.*, vol. 48, no. 8, pp. 1582–1595, 2018.
7. G. O. Büyüköz and H. Haki, "Binary honey badger algorithm for 0–1 knapsack problem," *J. Intell. Syst. Theory Appl.*, vol. 6, no. 2, pp. 108–118, 2023.
8. J. Dong, Z. Wang, and J. Mo, "A phase angle-modulated bat algorithm with application to antenna topology optimization," *Appl. Sci.*, vol. 11, no. 5, pp. 1–19, 2021.
9. Y. Feng, K. Jia, and Y. He, "An improved hybrid encoding cuckoo search algorithm for 0–1 knapsack problems," *Comput. Intell. Neurosci.*, vol. 2014, no. 1, pp. 1–9, 2014.
10. N. Moradi, V. Kayvanfar, and M. Rafiee, "An efficient population-based simulated annealing algorithm for 0–1 knapsack problem," *Eng. Comput.*, vol. 38, no. 3, pp. 2771–2790, 2022.
11. M. Abdel-Basset, R. Mohamed, S. Saber, I. M. Hezam, K. M. Sallam, and I. A. Hameed, "Binary metaheuristic algorithms for 0–1 knapsack problems: Performance analysis, hybrid variants, and real-world application," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 36, no. 6, pp. 1–25, 2024.
12. K. Luo and Q. Zhao, "A binary grey wolf optimizer for the multidimensional knapsack problem," *Appl. Soft Comput.*, vol. 83, no. 10, p. 105645, 2019.
13. E. Baş, "Binary Aquila Optimizer for 0–1 knapsack problems," *Eng. Appl. Artif. Intell.*, vol. 118, no. 2, p. 105592, 2023.
14. Y. Wang and W. Wang, "Quantum-inspired differential evolution with grey wolf optimizer for 0–1 knapsack problem," *Mathematics*, vol. 9, no. 11, pp. 1–21, 2021.
15. G. Yildizdan and E. Baş, "A novel binary artificial jellyfish search algorithm for solving 0–1 knapsack problems," *Neural Process. Lett.*, vol. 55, no. 7, pp. 8605–8671, 2023.
16. M. Abdel-Basset, R. Mohamed, I. M. Hezam, K. M. Sallam, A. M. Alshamrani, and I. A. Hameed, "A novel binary Kepler optimization algorithm for 0–1 knapsack problems: Methods and applications," *Alexandria Eng. J.*, vol. 82, no. 11, pp. 358–376, 2023.
17. B. Crawford, Á. Paz, R. Soto, Á. Peña Fritz, G. Astorga, F. Cisternas-Caneo, C. P. Toledo Mac-Lean, F. Solís-Piñones, J. L. Arce, and G. Giachetti, "Binary Pufferfish Optimization Algorithm for Combinatorial Problems," *Biomimetics*, vol. 11, no. 1, pp. 1–60, 2026.
18. K. Erdoğan, "An Adaptive Genetic Algorithm for the 0–1 Knapsack Problem," *Smyrna J. Natural Data Sci.*, vol. 1, no. 1, pp. 34–40, 2025.

Publisher's Note: The publisher remains impartial concerning jurisdictional claims in published maps and institutional affiliations. Responsibility for the content rests entirely with the authors and does not necessarily reflect the publisher's perspectives.