Adaptive Operator Selection with Reinforcement Learning

Rafet Durgut[a], Mehmet Emin Aydin[b], Ibrahim Atli[c]

[a] *Bandirma Onyedi Eylul University, Dept. of Computer Engineering, Bandirma, Turkey*
rdurgut@bandirma.edu.tr
[b] *UWE Bristol, Dept. of Computer Science and Creative Technologies, Bristol, UK*
mehmet.aydin@uwe.ac.uk
[c] *Ankara Yildirim Beyazit University, Dept. of Computer Engineering, Ankara, Turkey*
iatli@ybu.edu.tr

Abstract

Operator selection plays a crucial role in the efficiency of heuristic-based problem solving algorithms, especially, when a pool of operators is used to let algorithms dynamically select operators to produce new candidate solutions. A sequence of selected operators forms up throughout the search which impacts the success of the algorithms. Successive operators in a bespoke sequence can be complementary and therefore diversify the search while randomly selected operators are not expected to behave in this way. State of art adaptive selection schemes have been proposed to select the best next operator without considering the problem state in the process. In this study, a reinforcement learning algorithm is proposed to embed in a standard artificial bee colony algorithm for taking the problem state on board in operator selection process. The proposed approach implies mapping the problem states to the best fitting operators in the pool so as to achieve higher diversity and shape up an optimum operator sequence throughout the search process. The experimental study successfully demonstrates that the proposed idea works towards higher efficiency. The state of art approaches are outperformed with respect to the quality of solution in solving Set Union Knapsack problem over 30 benchmarking instances.

Keywords: Artificial Bee Colony, Adaptive Operator Selection, Reinforcement Learning, Q-Learning, Clustering-based Q-Learning

*Corresponding author
Email address: rdurgut@bandirma.edu.tr (Rafet Durgut)

1. Introduction

Optimisation is an unavoidable discipline in engineering studies, which attracts high interest for solving many real-world problems as an interdisciplinary field of engineering [1], mathematics [2], economics [3], computer science [4], industrial production process [5] etc. Among a wide range of optimisation techniques, metaheuristic-based optimisation approaches are developed as nature-inspired problem solving algorithms and considered within the umbrella of artificial intelligence [6]. Studies related to metaheuristics and heuristic optimisation have been very popular and tend to increase significantly in recent years.

One of the essences of metaheuristic approaches is to handle the dilemma of randomness and the use of gained experience within the search conducted by the algorithms. That is known to be exploration and exploitation dilemma that helps keep a balance between the use of randomness and gained experience in order to solve the problems fruitfully and to achieve higher efficiency. However, it is not trivial to sustain such a balance in between for all problem types, even if it is secured for a particular problem type, due to the differences in the natures of the problems. This usually requires user-defined procedures and problem-specific functional embeddings. One of the generalised approaches to avoid problem-specific obstacles and difficulties is to use binary representations and binary problem solving with various binary operators. Metaheuristic approaches impose the use of single operators in generating new solutions while conducting searches. On the other hand, evolutionary and swarm intelligence algorithms enforce multiple operators such as genetic operators and local search routines. A wide range of studies demonstrated that recruiting multiple operators offers a better payoff with respect to efficiency [7, 8]. Studies reveal that harmonic use of complementary operators helps improve the diversity of search, but, a well-customised switch scheme will be required to achieve higher efficiency. For instance, evolutionary algorithms impose a sequence of operators for this purpose while some other approaches such as variable neighborhood search reinforce a systematic change between two operators to let the agents switch search neighborhood. Another cluster of approaches can be named as adaptive methods with which the operators have opted on merit-based [9, 10] while the merits gained through the number of successfully use.

Machine learning has started attracting so much attention in use across all industries with its proven success in data modeling and predictive analytics. [11] overviews the use of machine learning for empowering metaheuristics in a wider perspective and integrates it with metaheuristic and evolutionary algorithms. It is known that reinforcement learning (RL) approaches are successfully used

in online training to opt to take the best action depending on environmental circumstances [12]. However, implementing RL algorithms in generalised form requires structural credit assignment and handling mechanisms which can be quite challenging. One of the ideas for generalising the online experience gained by RL algorithms to be exploited in later times and stages is elaborated and discussed in [13]. Their method implies combining a Q-Learning algorithm with clustering algorithm for structural credit handling and draws attention with its the success and simplicity. A recent work by [14] proposes the use of RL to adopt the best fitting mutation strategies in a differential evolutionary algorithm which is typically used in design problems. It is clear that a comprehensive study has not been seen yet on use of RL in building adaptive operator selection schemes in order to furnish artificial bee colony (ABC) algorithms for solving binary problems. In this paper, we aim to seek the utilisation of RL approaches in developing highly efficient adaptive operator selection schemes to be embedded in evolutionary and swarm intelligence algorithms , particularly in ABC algorithms, for problem solving. Following successful use of merit-based adaptive operator selection schemes in ABC algorithms for solving binary problems [9], we propose to use a bespoke Q-Learning algorithm combined with a Hard C-Mean clustering algorithm [15] to construct and refine an adaptive operator selection scheme to let ABC algorithm recruit multiple state-of-art binary operators to solve set union knapsack problem (SUKP) [10].

The paper is organized as follows: Section 2 provides the background context and related literature while Section 3 introduces the proposed algorithm with details of RL-based adaptive operator selection scheme. Section 4 elaborates the details of the experimental study to demonstrate the success of the proposed method in comparison with the state-of-the-art problem solving approaches for SUKP. Finally, in Section 5, the conclusion has been drawn and future work directions have been provided.

2. Background and Related Works

The aim of this section is to brief the facilitating background of this study and to overview the relevant works. ABC algorithm has been used as the metaheuristic and the swarm intelligence algorithm embedded with the adaptive operator selection scheme developed with a customised reinforcement learning.

2.1. Artificial Bee Colony Algorithms

Swarm intelligence algorithms have been originally developed for either discrete or continuous problems, and then implemented for the other types of problems; ant colony optimisation (ACO) [16] is originally developed for combinatorial problems, e.g. traveling salesman problem while particle swarm optimisation (PSO) [17] and artificial bee colony (ABC) algorithms are developed for continuous problems [18]. In order to implement a search algorithm for a problem type that is not originally developed for, additional conversion procedures are embedded into the algorithms, which applies to ABC, too. Two approaches are mainly devised for implementing ABC to solve combinatorial problems. The commonly used approach is to keep the continuous variables in use with an operation to map the variable in use to a binary vector and vice versa [19]. ABC algorithm has been commonly used across many industries to solve so many problems including combinatorial and binary problems. Since this main aim of this research is to emphasise developing adaptive operator selection using one of state-of-art machine learning approaches, further literature details have not been considered for an update but a bespoke algorithm is introduced in Section-2.2. The idea of this study has been built upon the recent works done with ABC in the form used in [9].

2.2. Adaptive Operator Selection

Operator selection is one of the functionalities needed by search algorithms that use multiple alternative operators. As explained before, the process follows rules for achieving higher efficiency in search while it is not trivial to have a generic rule that works well with all types of problems. The nature of the problems can affect the structure of the search space and hence enforce operators to show different behaviours. It can be highly dynamic, non-stationary, or can be moderately dynamic and stationary, etc. Therefore, many methods have been proposed in the literature for selecting the right operator at a certain moment based on some strategies from a pool containing multiple operators. It is called adaptive operator selection (AOS). This is a challenging problem because the characteristics of the operators used and their joint effects are not known. In other words, an operator that performs well at the beginning might act poorly over time or vice versa. Operator selection involves a two-stage process; credit assignment and operator selection [20, 21]. The first stage is for building the credibility of operators from their recent history of success while the second stage is about selecting the right operator based on the credits gained. The characteristics of AOS algorithms proposed in the literature vary in the way how each is utilised.

Credit assignment is generally conducted using fitness value and ranks. The quality of an operator can be determined in several ways. The simplest way is to utilize the latest fitness value [22] or average fitness values within a time window that indicates improvements in solution quality [23]. [24] proposes taking maximum improvement in fitness values obtained over the chosen time window by operators. However, when the environments are not stationary, [25] proposes a weighted average update. Using raw fitness improvements as the quality directly has a large impact on robustness. In order to increase the robustness in credit assignments, the work by [26] utilizes a normalization scheme. Another approach is to use the latter credit assignment method, namely rank-based credit assignment [27, 24]. Meanwhile, a similar approach proposed by [28], so-called Compass method, takes both the fitness improvement and population diversity into account. In [24], sum of ranks and area under curve methods are utilized for aiming at robustness.

After updating the qualities of operators using credit assignments, the next step is to determine the way to select operators. The most popular and promising algorithms are Probability Matching (PM) [29], Adaptive Pursuit (AP) [30] as the probability-based method and multi-armed bandit-based (MAB) methods [31, 26, 21]. While PM and AP make selection according to credit values, multi-armed bandit-based methods like Upper Confidence Bound (UCB) additionally take number of selections into account [32]. Credit values obtained over a certain range are also used in some dynamic MAB versions besides using current credit values [31, 26]. Although these algorithms have a similar logic while obtaining credits and producing selection criterion, the optimisation procedure is usually handled with different method such as ant colony optimisation [33], evolutionary algorithms [34, 20] and [21].

2.3. Reinforcement Learning

Reinforcement learning is a modern machine learning paradigm which implies gaining experience on run-time and using it to learn the relationship between input and output sets. It is particularly useful for modern robotic applications as well as softbot systems to let robotic and software agents smartly interact with their living/operating environments; this enables them to take the environmental events as stimuli to decide the best reaction to the environment, in response to what is ongoing within the environment [35].

The main principle behind reinforcement learning is to let agents learn how to act optimally in order to maximise the environmental reward generated in response to their actions. Let an agent

be capable of taking a set of actions, A, to respond to ongoing activities within the environment, E, where it lives in, and let the environment be represented with a set of states, S, with which the status of the environmental instances/episodes are perceived. The agent decides to take an action $a \in A$ in response to the environmental state, $s \in S$. This causes a change within the environment as the result of the action a taken and an indicator of the payoff of the change made, $r \in \mathbb{R}^+$, is produced to reward the agent's action. Each agent lives within the environment and aims to maximise the rewards, $r(s, a)$, gained per instance/episode, represented with the pair of (s, a), and accumulates the rewards towards maximum gain. That is:

$$\max \quad R = \sum_{t=0}^{\infty} r_t(s, a) \tag{1}$$

where t represents the time index, $r_t(s, a)$ is the payoff gained as a result of taking action a in response to state s at time t.

RL has been successfully used in training the robotic and/or software agents, which are employed in various purposes including games, in various forms from simpler cases to complex problems [36]. Recently, deep learning versions of RL have been developed and made available to handle and solve complex, dynamic, online and real-time problems [36]. This success history has triggered its use in operator selection as part of the heuristic optimisation briefed above. It would facilitate building more conscious adaptive selection schemes to take inputs into account while selecting the operators and crediting each aftermath of conducting each operation.

2.4. Set Union Knapsack Problem

This study has been conducted with solving set union knapsack problem (SUKP), which is one of well-known non-polynomial (NP) hard combinatorial optimisation problems [37]. SUKP is a special case of generic knapsack problem, which involves items to be optimally composed in subsets so as to gain the maximum benefit. Given a set of n elements, $U = \{u_i | i = 1, .., n\}$ with a non-negative weight set, $W = \{w_i | i = 1, .., n\}$ and a set of m items, $\mathcal{S} = \{U_j | j = 1, .., m\}$ with a profit set, $\mathcal{P} = \{p_j > 0 | j = 1, .., m\}$, a subset of $A \subseteq \mathcal{S}$ is sought to be found such that it maximises the profit subject to that the sum of the weights of selected items not to exceed the capacity constraint,

C. The formal structure of the problems is as follows:

$$\max \quad P(A) = \sum_{j \in A} p_j \tag{2}$$

$$\text{s.t.} \quad W(A) = \sum_{i \in \bigcup_{j \in A} U_j} w_i \leq C, \quad A \subseteq \mathcal{S} \tag{3}$$

The problem is represented in real numbers and needs to be represented in binary form to enable binary operators in search algorithms as in binary ABC [9]. Following the approach introduced by [38], in order to do that a binary vector, $B = \{b_j | j = 1, .., m\} \in \{0, 1\}$, is defined to be used as the decision variable, where $b_j = 1$ if an item is selected, $b_j = 0$, otherwise. The new model can be reformulated as follows:

$$\max \quad f(B) = \sum_{j=1}^{m} b_j p_j \tag{4}$$

$$\text{s.t.} \quad W(A_B) = \sum_{i \in \bigcup_{j \in A_B} U_j} w_i \leq C \tag{5}$$

The main goal is to find the best binary vector, B, which provides the subset of items with the maximum profit.

2.5. Related Works

This study relates to adaptive operator selection, artificial bee colonies, reinforcement learning and set union knapsack problem, while the main focus goes on development of building adaptive operator selection schemes using reinforcement learning. This is why the most relevant works have been chosen from the literature on adaptive operator selection and reinforcement learning while there is hardly any study conducted exactly matches the scope of this study. Binary artificial bee colony algorithms have been credited in Section 2.1 and while literature on adaptive operator selection is briefed in Section 2.2. The state-of-art algorithms in solving set union problems have been used for comparative analysis in Section 4.

Operator selection in an adaptive way has been studied since early 1990s particularly applied to evolutionary algorithms in order to determine the rates of genetic operators to select each dynamically over time [22]. The significant progress seems to be achieved over last decade on which

the operator selection schemes have been devised to help improve the efficiency of evolutionary algorithms applied to multi objective optimisation problems [20, 39]. The approaches mentioned in Section 2.2 have been extensively used in various forms to solve a number of problems. All of these approaches emphasise on how to promote the most successful operators and to rank them on success/merit basis, which enforces to pay more attention on how to credit/discredit the operators. Although a considerable level of success has been achieved through these studies, the situational information such as problem state has not been considered by any of state-of-the-art studies. None of the adaptive approaches reported above have considered the problem state and/or the history of previous acts while selecting an operator to produce new solutions.

Another group of studies have conducted research on how to best select the mutation operators to use in differential evolution algorithms, where multiple genetic operators have been made available to opt in action. Few studies propose to take the information collected from fitness landscapes into account for operator selection. The authors of [40] use an exploratory fitness landscape approach to device a prediction machine to let select the operators, while [7] uses fitness landscape and performance measures in operators selection without any structured learning process, and [41] propose to use fitness landscape analysis measures to train a dynamic regression model for online learning to build operator selection scheme. On the other hand, a very recent study reported in [14] imposes a reinforcement learning algorithm embedded in differential evolution algorithm is similar to what [7] proposes in order to help select the best fitting mutation strategy subject to given circumstances of parametric status. The study is attempted using a Q-Learning algorithm based on Q-Table, which produces good results if the finite number of states is moderately low so as to tabulate each against each action. Relatively larger state spaces cannot afford to work with Q-Table-based learning approaches.

To the best knowledge of the authors, a comprehensive approach using reinforcement learning (RL) to take the problem state, i.e. input data, on board in operator selection has not been studied before for the purpose of developing adaptive operator selection schemes. Such an RL-based approach is expected to help agents learn on the fly how to select the best fitting operator for the given problem state while conducting a search for best solution. In the next sections of this paper, a reinforcement learning approach has been explained with which the search algorithm will gain situational knowledge through online training to get assisted in selecting the best fitting operators. A Q-Learning algorithm has been merged with a clustering algorithm to help generalise

the situational information as in [13] and learn to select the best operator subject to the situational information including the problem state. To the best knowledge of the authors, this is the first study considers such an approach to train the search agents online to select the best operator within an artificial bee colony algorithm devised to solve set union knapsack problem represented in binary form.

3. Operator selection with Reinforcement Learning

The operator selection schemes studied in the literature mainly devised on merit basis with which the selection process relies on the gained reputation over success history. Another rising stream of the studies bring forward the use of fitness landscape measures to set up basis for building efficient selection schemes. As indicated in previous sections, the problem state does not play a major role in the operator selection process while an efficient selection mechanism cannot avoid disregarding the problem state from the selection process. It is well-known that reinforcement learning (RL) emerges as a very promising online/active learning approach assisting immediate exploitation of cumulatively gained experience within the purpose of use, which can be a decision making or a prediction. The essence of RL is to evaluate an input set that represents the problems state and take the best action in its capacity to date, accordingly, while the environment, which holds the agents, let them be trained, and produces payoff signals upon any action taken by agents to pass into the agents in order to let them consider for gaining experience and learning how to select actions in response to the input set as the situational data.

This study aims to propose a more conscious selection process developed based on reinforcement learning approach implemented into a distance-based clustering algorithm in which the distance in between the input set and the fine-tuned cluster centers is estimated and made reference index in operator selection. The idea of setting up a selection scheme based on clusters is discussed and implemented in machine learning studies. Reinforcement learning is known to be very useful in handling dynamically changing environment and for solving dynamic problems, particularly for operating within unknown dynamic environments. One of earlier studies proposes embedding reinforcement learning in a distance-based clustering algorithm, namely hard-c-means algorithm, to train agents to select the best scheduling operator subject to dynamic production environments to solve dynamic scheduling problems [13]. Inspiring of this study, a reinforced-clustering algorithm is put together to optimise the cluster centers so that the problem states can be classified with

optimised clusters, where each cluster will correspond to an operator. The algorithm will impose selecting the cluster center, i.e. operator, closer to the input set in distance. This will facilitate a selection scheme more conscious with problem state.

3.1. Operator Selection

Given a set of operators, $\mathcal{O} = \{o_i | i = 1, .., |\mathcal{O}|\}$, and a binary vector of inputs, $\mathbf{x}_t = \{x_{t,j} | t \in T, j = 1, .., |\mathbf{x}_t|\}$, where T is the time/iteration count, $x_{t,i} \in \{0, 1\}$, $\mathbf{x}_t \subseteq \mathcal{S}$ and $\mathcal{S}$ represents the complete problem state space. An operator $o_i \in \mathcal{O}$ works as a function, $f(\mathbf{x}_t)$, to produce a neighbouring state, $\mathbf{x}_{t+1}$, that is evaluated with a measure, so called fitness, represented as $F(\mathbf{x}_t) : \mathbf{x}_t \to \mathbb{R}$. The operators are attached with an indicator, $\mathcal{Q} = \{Q_i | i = 1, .., |\mathcal{O}|\}$, to measure the credit gained throughout the search process upon each successful production, which is usually decided if $F(\mathbf{x}_{t+1}) < F(\mathbf{x}_t)$ for minimisation cases, opposite otherwise. The operators are selected subject to an ϵ policy in which an operator is selected randomly if the random number drawn is lower than the threshold of ϵ, otherwise the operators selection scheme imposes its rule to identify an operator.

$$i_t* = \begin{cases} i_r & \text{if } \epsilon < rand \\ \arg\max_{i \in |\mathcal{O}|} \{Q_{i,t}\} & \text{otherwise} \end{cases} \tag{6}$$

where $rand$ is the random number, $i_r \in |\mathcal{O}|$ randomly selected, and i_t* is to be taken as the index for o_i, i.e. selected operator. Once operator o_i is selected and applied to $\mathbf{x}_t$, its credit level is updated with $Q_{i,t+1} = \delta Q_{i,t} + (1 - \delta) r_{i,t}$, where $0 \leq \delta \leq 1$ is a discount rate and $r_{i,t}$ is the reward allocated to o_i for its success level on $\mathbf{x}_t$ state.

Rewarding is the way to allocate credit to a chosen operator upon its success. Each selected operator is evaluated, once applied, with respect to the level of success; success or failure. A reward value, $r_{i,t}$, is decided/calculated and assigned to the selected operator, o_i, in order to update its credibility for the next runs. The reward can be determined/calculated based on either instant results or on some calculated statistics such as the average or extreme value over a pre-set time window. Each statistic is recalculated/updated after delivery of each operation over the last W number of iterations, i.e. run of an operation. The success level per operation can be estimated based on either the value of an objective function, i.e. a value in real number, or the success status, i.e. success or failure. An objective function-driven reward calculation is adopted in this study as

detailed in the next section.

An instant reward can be calculated with $r_{i,t} = F(\mathbf{x}_{t+1}) - F(\mathbf{x}_t)$ once o_i applied to $\mathbf{x}_t$ at time t, where $F(\mathbf{x}_{t+1})$ is the calculated objective value of the resulted state, $\mathbf{x}_{t+1}$ once o_i applied to $\mathbf{x}_t$ and $F(\mathbf{x}_t)$ is the known objective value of the current solution, $\mathbf{x}_t$. It is known that reward calculation with immediate results, i.e. instant case, can cause degeneration or disruption in the later stages [26]. An idea to overcome these possible issues is to normalise the difference between $F(\mathbf{x}_{t+1})$ and $F(\mathbf{x}_t)$ with the factor of the rate between attained objective value and the global best value as follows:

$$r_{i,t} = \frac{F(\mathbf{x}_{t+1})}{gbest_t}(F(\mathbf{x}_{t+1}) - F(\mathbf{x}_t)) \tag{7}$$

where $gbest_t$ is the global best fitness value known to the time t. As indicated before, an instant reward may not be useful all the time, while average reward, $\bar{r}_{i,t}$ or extreme reward, $\hat{r}_{i,t}$, are two alternative options to be used, which can be calculated with the following equations, Eq 8 and Eq 9, respectively:

$$\bar{r}_{i,t} = \frac{1}{W} \sum_{j=t}^{(t+W)} r_{i,j} \tag{8}$$

$$\hat{r}_{i,t} = \max_j \{r_{i,j} | j = t, .., (t+W)\} \tag{9}$$

In both of the cases, the time window in size of W is considered to be a sliding window throughout of learning and search process.

3.2. Q-Learning combined with clustering

Q-Learning algorithm is one of very commonly used and highly reputed reinforcement learning algorithms, which uses a data quality indicator, $Q(s, a)$, to measure the credit of applying action a to state s. The algorithm works iteratively to accumulate experience gained on the fly and immediately use it. The underlying logic conforms to Markovian Decision Processes (MDP) in which a decision is made as to the result of the previous decision. Q-Learning lets agents online learn how to optimally act by selecting the best action $a \in A$ given the environmental state $s \in S$. Once action a is taken for state s, a new state, $s' \in S$ is generated, then the environment produces

a payoff value, $r \in \mathbb{R}$ in response to a new state to reinforce the agent on the result of its action.

$$Q(s,a) = Q(s,a) + \alpha\Big(r + \gamma \max_{a} Q(s',a) - Q(s,a)\Big) \tag{10}$$

where α is the learning rate and γ is the discount rate, which is applied to the expected maximum Q value of the next step. This process is repeated as long as the operation is alive. Further details can be found in [42] and [35].

Q-Learning algorithm works based on immediate feedback gathered online for exact situations, which need to be generalised for use in a wider scope, however, the algorithm as is does not provide such a structure for this purpose. Q Table is the most and earlier used data structure proposed to sustain learned experience [35]. However, Q Table is not sufficiently scalable with respect to problem size. Functional approximation approaches including artificial neural networks [12] and deep learning have been implemented to generalise the experience to efficiently cover unseen situations.

Clustering algorithms have been considered to generalise situational experience integrating Q-Learning algorithm into the clustering procedure. Hard-C-Means algorithm is one of distance-based unsupervised clustering algorithm, a kind of k-means algorithm versioned to be a base for fuzzy-C-Means algorithm used in fuzzy rule generation[43]. It has been merged with Q-Learning and converted to reinforcement learning algorithm as proposed in [13], where it is reported that the core idea has been successfully implemented for dynamic production scheduling problems. However, it was originally implemented with a simple binary reinforcement mechanism to feed the algorithm while this study extends the approach with implementing a statistics-based reinforcement mechanism as mentioned in Section 3.1.

Hard C-Means algorithm originally works as an unsupervised learning algorithm, similar to K-Means, but, it is intervened to align with Q-Learning so as to serve as a reinforcement learning algorithm. In order to do that the number of cluster centers are fixed to a certain number and the membership to each center is fine-tuned through online learning. Let $\mathcal{C} = \{\mathbf{c}_i | i = 1, .., |\mathcal{O}|\}$ be a set of clusters, where each is defined as $\mathbf{c}_i = \{c_{i,j} | i = 1, .., |\mathcal{O}|, j = 1, .., |\mathbf{x}_t|\}$ to represent an operation, o_i, with $|\mathbf{x}_t|$. The data structure for cluster centers, $\mathbf{c}_i \in \mathcal{C} \quad \forall i \in \mathcal{O}$, is inserted into Q-Learning algorithm to let the quality indicator and the cluster centers work side-by-side and feed each other once required. All clusters, $\mathbf{c}_i \in \mathcal{C} \quad \forall i \in |\mathcal{O}|$, are initialised to a 0 vector, and

then updated based on credits/rewards gained by each operator, o_i. The centers of the clusters are updated upon each successful selection of the corresponding operators, while they are selected randomly when the clusters have overlapping centers. The centers are updated per dimension with $c_{i,j} \leftarrow \big(c_{i,j} + x_{t,j}\big)/t$, where i is the index for corresponding operator, j is the dimension index for input data, i.e. j^{th} element of input vector, and t is the iteration count.

A typical iteration of the combined algorithm works as follows: (i) the learning agent perceives a state of $\mathbf{x}_t$ at iteration t, (ii) selects an operator, o_i, using Eq. 6, applies to $\mathbf{x}_t$ and produces $\mathbf{x}_{t+1}$, (iii) the environment responds to the change in the state with the action taken, o_i, with a reward of r, where an extreme reward approach (Eq 9) through a sliding window, W is used for this purpose (iv) the center for cluster $\mathbf{c}_i$ is updated with $c_{i,j} = \big(c_{i,j} + x_{t,j}\big)/t$ if the reward is larger than 0, $r > 0$, no change is applied otherwise, (v) the quality indicator, $Q(\mathbf{x}_t, o_i)$ is updated with the following equation:

$$Q(\mathbf{x}_t, o_i) = Q(\mathbf{x}_t, o_i) + \alpha\Big(r + \gamma E(Q(\mathbf{x}_{t+1}, o_i)) - Q(\mathbf{x}_t, o_i)\Big) \tag{11}$$

where α and γ are learning and discount rates as in Eq. 10, while $E(Q(\mathbf{x}_{t+1}, o_i)) = \|\mathbf{x}_{t+1} - \mathbf{c}_i\|$, as the expected value for quality indicator of state $\mathbf{x}_{t+1}$ and action o_i, which is defined to be a distance measure in between the state $\mathbf{x}_{t+1}$ and cluster center $\mathbf{c}_i$. The expected value $E(.)$ is commonly defined to be $max(.)$ as in Eq 10. However, it can be redefined to adapt to circumstances of the domain and the implementation. In this study, $E(.)$ is defined to be based on minimum Euclidean distance between a problem state, i.e. an input vector, and the cluster centers as mentioned above.

The procedure of this learning process is sketched in pseudo-code provided in Algorithm 1.

3.3. ABC algorithm embedded with combined Q-Learning (RLABC)

Q-Learning merged with hard C-means clustering algorithm has been developed to serve as an adaptive operator selection scheme learning on-fly. It lets the operator selection engine self-build up on-fly in which it gains experience through exploration and uses the collected data to assist in learning how to react to the environmental circumstances. Figure 1 illustrates the logic behind how the system works and the components interact towards achieving a built-up operator selection scheme working adaptively. The system is architected of three main interacting components; *selection engine*, *evaluator*, and *learner*. The *selection engine* (i) collects input data from the solution pool, i.e. bee swarm, (ii) passes relevant information to *the evaluator* and receives advice for the

Algorithm 1 General overview of Q-Learning algorithm merged with clusters

Operator Selection
if there are non-rewarded operators **then**
    Choose o_i randomly
else
    Calculate probabilities of operators using Eq. 6
    Choose o_i using Roulette-Wheel selection
end if
Operator Evaluation
Execute o_i and get r
if $r > 0$ **then**
    Update c_i of applied o_i
    Add r to operator o_i
end if
Credit Assignment
Get total rewards for current iteration
Update Q using Eq. 11

best operator from the operator pool, (iii) chooses the operator (a single operator each time) and produces new solutions, and then forwards all relevant information to *the learner*, while *evaluator* proposes the best operator based on the most up-to-date credit levels using Eq. 6. Meanwhile, *learner* runs one iteration of the learning algorithm, i.e. Q-Learning combined with clustering, to update the credit level of chosen operator and the centroid details of the corresponding cluster using Eq. 11, then feeds *evaluator* with the most up-to-date credit and centroid values.

The adaptive operator selector (AOS), composed of the three components sketched in Figure 1 and explained above, has been embedded in a standard artificial bee colony (ABC) algorithm, labeled with *RLABC* henceforth, to solve SUKP problems as indicated before. The pseudo-code of *RLABC* is illustrated in Algorithm 2. Once AOS and ABC parameters including all algorithmic details are initialised, the employed bees take a position to start conducting the search, as the first phase, in which an operator from the operator pool is selected using selection scheme and a neighbouring bee per active bee to pair with is identified using the roulette-wheel approach. The selected operator takes the problem states from the active bee and its neighbour to produce a new solution candidate. If the fitness of the freshly produced solution is better than the solution held by the active bee, then the old solution held is replaced with the new one, and the operator is rewarded accordingly. If the produced solution is worse, then, it is discarded and the count indicating the successive failure record of the active bee is counted up by 1.

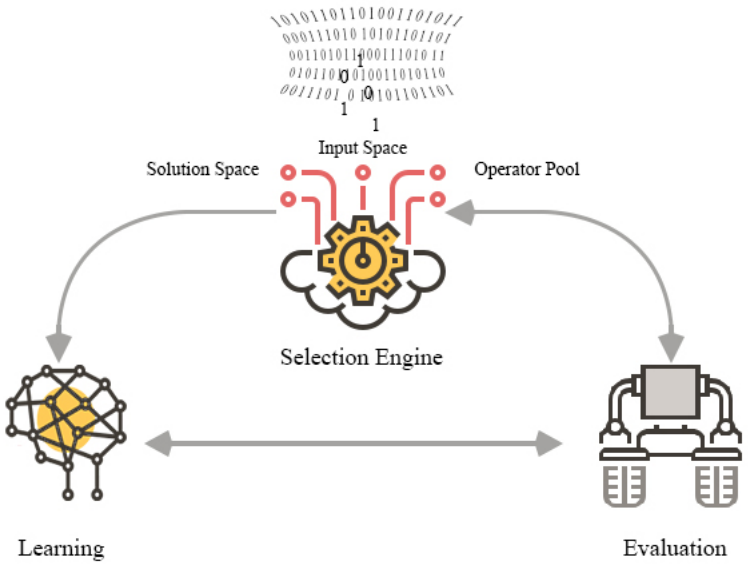


Figure 1: The mechanics of building adaptive operator selector with reinforcement learning

The next phase of ABC is to run the onlooker bees in which the bees are activated based on the probabilities calculated with the bias of fitness values. All relevant actions including operator selection, fitness and reward calculations are repeated. Once a bee is found not contributing more than a certain number of trails, it is moved/deleted out of the swarm/population and a newborn bee with fresh randomly generated solution is added to the swarm. At the final stage of an iteration of ABC, (i) the rewards are allocated to each of recruited operators so as to consider an updated version in the next iteration, (ii) the global best solution produced by the swarm is tested and updated, and (iii) the stopping criteria is checked if it is the time to stop, otherwise moves to the next generation.

4. Experimental Results

4.1. RLABC implementation

ABC algorithm has been implemented to solve SUKP problems using multiple binary operators selected with an adaptive selection scheme built with Q-Learning algorithm as explained before. A typical solution, i.e. a problem state $\mathbf{x}_t \in \mathcal{S}$, is represented with a binary vector, $\mathbf{x}_t = \{x_{t,j} | x_{t,j} \in \{0,1\}, t \in T,\ j = 1,..,|\mathbf{x}_t|\}$, where $t \in T$ is the time/iteration count, $|\mathbf{x}_t|$ is the solution size,

Algorithm 2 Reinforcement Learning Based Artificial Bee Colony

```
Initialization of ABC and AOS
while Termination criteria is not met do
    Employed Bee Phase
    for each bee in colony do
        Select operator (o_i)
        Produce and evaluate candidate solution
        Operator Evaluation using Algorithm 1
        if Candidate is better than current bee then
            Replace current bee with candidate
        else
            Increment trial number of current bee
        end if
    end for
    Onlooker Bee Phase
    Calculate selection probability of each bee (P)
    for i ← 1 to n do
        Choose current bee according to P
        Select operator (o_i)
        Produce and evaluate candidate solution
        Operator Evaluation using Algorithm 1
        if Candidate is better than current bee then
            Replace current bee with candidate
        else
            Increment trial number of current bee
        end if
    end for
    Scout Bee Phase
    if trial number of any bee has exceeds the limit then
        Replace the first scout bee with random valid solution
    end if
    Update credits of operators
    Memorize the best solution
end while
```

which is scaled with the benchmark problem size. As the algorithm is implemented for SUKP benchmarks, the solution size is set to the total number of items to select the best subset among, which is $|\mathbf{x}_t| = m$. Suppose that we are given 10 items, $m = 10$, each includes 5 different elements, then, $\mathbf{x}_2 = \{0, 1, 0, 1, 1, 1, 0, 0, 1, 1\}$ would be a typical solution, say at time 2, $t = 2$. Each swarm of bees will be a collection of such solutions to be used for search. Meanwhile, new solutions will be generated with operators, which are selected from a pool of operator. The action space, A, consists of the set of operators, $A = \mathcal{O}$. The operators make up the pool to select, $o_i \in \mathcal{O}$, are taken from the state-of-art binary operators collected from literature. The set of operators is formed of $\mathcal{O} = \{binABC, ibinABC, disABC\}$ taken from [44], [45] and [46] as explained in [9] and [10]. The algorithm solves 2 problems inherently; (i) it seeks for the best set of items to place in a knapsack, while (ii) it also looks for the set of best operator to be paired with each solution, $< \mathbf{x}_t, o_i >$. In the end, the best solution for the problem, $\mathbf{x}_T^*$ and the best sequence of pairs, $\mathfrak{S} = \{< \mathbf{x}_t, o_i > | \mathbf{x}_t \in \mathcal{S}, o_i = \arg\max_{i \in |\mathcal{O}|}\{Q_{i,t}\}, t \in T, i \in |\mathcal{O}|\}$ are expected to be gained.

SUKP problem is a capacity constrained problem that requires feasibility check once applied operators. Solutions generated with the operators can end up infeasible, which need to be either enforced to be eliminated/discredited with penalty functions, or can be repaired for feasibility with some functions. Repairment functions gained more attraction in the state-of-the-art literature for knapsack problems; a greedy approach, so called S-GROA, has been proposed by [47] as part of solving SUKP problems.

The algorithm imposes a greedy approach in which the items with higher values are prioritised to keep within the knapsack when the capacity is exceeded and similarly the items with higher values from outside are forced to select when the knapsack is filled. The main logic behind S-GROA approach is to find an infeasible solution first and to create a zero vector next. The items included in infeasible solution are ranked with respect to the total density of the items rated with the benefit value of the items, $\frac{p_i}{\sum_{i \in U_i}(w_j/d_j)}$, where p_j is the benefit of item i, w_j and d_j are the weight and the frequency of element j. Then, the blank items are moved in the blank solution until it fills to the capacity.

4.2. Test Problems

This study has been conducted to demonstrate the gain/benefit of using reinforcement learning in developing adaptive operator selection schemes, which are embedded within a standard ABC

Table 1: The SUKP instances

Set_1					Set_2					Set_3				
ID	m	n	**w**	**y**	ID	m	n	**w**	**y**	ID	m	n	**w**	**y**
PI_1_1	100	85	0.10	0.75	PI_2_1	100	100	0.10	0.75	PI_3_1	85	100	0.10	0.75
PI_1_2	100	85	0.15	0.85	PI_2_2	100	100	0.15	0.85	PI_3_2	85	100	0.15	0.85
PI_1_3	200	185	0.10	0.75	PI_2_3	200	200	0.10	0.75	PI_3_3	185	200	0.10	0.75
PI_1_4	200	185	0.15	0.85	PI_2_4	200	200	0.15	0.85	PI_3_4	185	200	0.15	0.85
PI_1_5	300	285	0.10	0.75	PI_2_5	300	300	0.10	0.75	PI_3_5	285	300	0.10	0.75
PI_1_6	300	285	0.15	0.85	PI_2_6	300	300	0.15	0.85	PI_3_6	285	300	0.15	0.85
PI_1_7	400	385	0.10	0.75	PI_2_7	400	400	0.10	0.75	PI_3_7	385	400	0.10	0.75
PI_1_8	400	385	0.15	0.85	PI_2_8	400	400	0.15	0.85	PI_3_8	385	400	0.15	0.85
PI_1_9	500	485	0.10	0.75	PI_2_9	500	500	0.10	0.75	PI_3_9	485	500	0.10	0.75
PI_1_10	500	485	0.15	0.85	PI_2_10	500	500	0.15	0.85	PI_3_10	485	500	0.15	0.85

algorithm equipped with a number of binary operators to solve SUKP problems. SUKP is a special case of generic knapsack problems that are proven to be one of the known NP-hard combinatorial problems. The problem has been modeled as a binary problem as seen in Section 2.4. The test problems chosen in this study are collected from recently published literature.

He et al. [47] have introduced 30 benchmarking problem instances of SUKP as tabulated in Table 1 with all configuration details, where 3 different configurations presented varying with comparative status of m and n; (i) $m > n$, (ii) $m < n$, and (iii) $m = n$), while $\mathbf{w} \in \{0.10, 0.15\}$ and $\mathbf{y} \in \{0.75, 0.85\}$ representing the density of elements and the rate between the capacities and the sum of weights of elements, respectively. As seen, each set of problem instances includes 10 instances varying with $m, n, \mathbf{w}$ and $\mathbf{y}$ values. More details can be found in [47, 48].

4.3. Parameter Tuning

The parametric setting is one of the major steps to conduct in experimental studies, which has also a significant impact upon the success of problem-solving approaches. Therefore, the first step of experimental studies is usually conducting a comprehensive parameter tuning. The parametric study has been conducted in 2 phases in this study; the first phase is to identify the parametric settings applicable to all operator selection schemes used in this study including the proposed approach, the second is to single out the best performing adaptive selection scheme among the state-of-the-art approaches so as to compare with the proposed approach throughout the entire experimentation.

The first phase has been carried out for all related parameters required including γ, used to normalise the distances measure as part of learning-based operator selection scheme, experimenting with one of commonly used problem instance by the state-of-the-art studies, PI_2_6. The best parametric level has been sought for the type of *reward* whether to be *extreme* ($\hat{r}_{i,t}$) or *average*

Table 2: Parameter tuning for proposed approach

				γ=0.1		γ=0.3		γ=0.5		γ=0.9	
Reward	W	ϵ	α	Best	Mean	Best	Mean	Best	Mean	Best	Mean
Avg ($\bar{r}_{i,t}$)	5	0.1	0.1	10788	10647.2	10860	10674.57	11169	10678.57	10933	10674.17
			0.5	11410	10686.93	10848	10630.77	11161	10681.03	10933	10691.17
			0.9	11233	10736.3	10945	10692.77	11113	10725.13	11064	10665.43
		0.2	0.1	11237	10683.33	10968	10653.93	11179	10683.9	11064	10693.6
			0.5	10859	10677.33	10793	10658.87	11054	10697.63	11057	10677.27
			0.9	11187	10689.3	11159	10680.8	11093	10695.33	10851	10672.37
	25	0.1	0.1	11093	10679.77	11007	10688.33	10867	10665.13	10889	10675.27
			0.5	11043	10699.73	11067	10628.93	10953	10671.73	11064	10707.37
			0.9	10793	10658.63	11138	10710.03	10968	10669.3	11043	10736.97
		0.2	0.1	10782	10650.9	10847	10643.33	10889	10663.5	11064	10678.13
			0.5	11113	10694.83	10918	10640.7	10959	10693.73	10923	10645.57
			0.9	11007	10703.23	11081	10685.67	10788	10660.7	11251	10683.63
Ext ($\hat{r}_{i,t}$)	5	0.1	0.1	11261	10689.73	10808	10662.3	11282	10682.4	11093	10730.3
			0.5	11108	10703.03	10937	10685.33	11064	10703	11077	10695.33
			0.9	11064	10695.93	10782	10668.47	10835	10700.03	10804	10658.1
		0.2	0.1	10945	10687.27	11113	10700.27	11132	10682.97	11039	10668.8
			0.5	10857	10647.3	11410	10701.93	11151	10685.7	11233	10714.77
			0.9	11205	10681.53	11007	10682.93	11113	10710.93	11077	10699.3
	25	0.1	0.1	10951	10712.93	11425	10716.57	11005	10684.87	11113	10684.57
			0.5	11178	10726.13	10788	10676.67	11344	10682.37	11410	10740.37
			0.9	10860	10661.37	11344	**10770.5**	10860	10682.27	10925	10648.7
		0.2	0.1	10968	10688.63	11113	10723.33	10914	10660.87	10831	10642.53
			0.5	10808	10663.6	11178	10702.57	10980	10665.43	10925	10694.57
			0.9	11196	10697.87	11237	10684.33	10793	10629.03	11425	10694.13

($\bar{r}_{i,t}$), the best size of time window, W, in iterations, the minimum likelihood of an operator to be selected, ϵ, and α as the adaptation coefficient. This preliminary study for fine-tuned parameters has been conducted over 30 repetitions and summarised with *best* and *mean* statistics in Table 2. The best parametric setting is concluded that γ is 0.3, W is 25 iterations, *reward* is *extreme* ($\hat{r}_{i,t}$), ϵ is 0.1 and α is 0.5. This parametric setting has been used across the whole study reported in this article.

The second phase of the parametric study has been conducted to identify the best adaptive approach among the renown alternatives; probability matching (PM), adaptive pursuit (AP) and upper confidence bound (UCB). The purpose of this phase is to identify the best-performing adaptive selection scheme to compare with the proposed approach by this study. The results of this phase are tabulated in Table 3, where PM is concluded as the best performing scheme among the three approaches, where the performance of each approach is indicated with *Best* and *Mean* statistics. PM clearly performs better with respect to *Mean* measure, hence, it has been chosen as the best performing adaptive approach to use for comparisons purposes.

The next step in experimentation is to run the binary ABC algorithm embedded with an adaptive operator selection scheme for all 3 sets of SUKP benchmark instances introduced in Table 1. The results of *PM*, proposed approach, labeled as *RLABC*, and random selection scheme, labeled as

Random, have been summarised in Table 4 with *Best*, *Mean* and *St.Dev.* statistics; each is calculated over 100 repetitions. *Random* has been chosen to clearly indicate that the adaptive operator selection schemes make a significant difference. The results suggest that *RLABC* outperforms *PM* in most of the instances while *Random* produces the worst results as expected.

In order to ascertain if the differences are statistically sound, Friedman test has been conducted for the results tabulated in Table 4 and test results are presented in Table 5, where the averaged ranks calculated for all three selection schemes presented with respect to *Best* and *Mean* measures are indicated against the three sets of problem instances (i.e. Set_1, Set_2, and Set_3). Meanwhile, the corresponding p-values of the columns are added to the bottom line of the table, which proves the significance of the difference in the performances. The averaged ranks provided in Table 4 suggest that *RLABC*, as the best performer, holds all lowest rank values across the columns with respect to *Mean*, while all *Best* values except of Set_1 appear the lowest by *RLABC*, where *PM* remains the first runner up. The ranks of *Random* are the highest. It is important to make clear that the higher the rank the lower the performance in this statistical test-based evaluations. Meanwhile, it is observed that the ranks remain very close to one another with respect to *Best* measure and the corresponding p-values are calculated over the threshold value of 0.05 for 95% confidence level, which means the differences are not significant in terms of *Best* measure. On the other hand, the ranks with respect to *Mean* measure are clearly different as their p-values are far below the threshold, 0.05. This is due to fact that all algorithms hit the best known or optimum solutions time-to-time, but, not as many as *RLABC* does.

RLABC has been investigated with respect to the efficiency of the operators and the journey of their reward hunts throughout the search process reward. As explained above, the main idea is to let the algorithm chose the best fitting operator to the circumstances of the problem state. While seeking for the optimum solution, the algorithm tries to build the best sequence of operators paired with the solutions, e.g., $< \mathbf{x}_t, o_i >$. After each operation is applied, the reward for that operator is calculated and allocated accordingly. The profile of the reward hunting per operator has plotted in Figure 2, where the most reward is accumulated by *ibinABC* and then by *disABC* while *binABC* remains as the worst, in this respect. That is very much in line with the capabilities of the operators. It is known that *ibinABC* dominates *binABC* since it is an extension of *binABC* and inherits all capabilities while *disABC* remains competent and complementary to both. Therefore, the *ibinABC* hunts the majority of the rewards, while *disABC* competes with that and *binABC* remains much

Table 3: Comparison on parameter tuning phase

Parameters				AP		PM		UCB		RLABC	
R	W	ϵ	α	Best	Mean	Best	Mean	Best	Mean	Best	Mean
Avg ($\bar{r}_{i,t}$)	5	0.1	0.1	10998	10658.63	11113	10719.8	10948	10650.2	10860	10674.57
			0.5	11012	10676.30	10961	10692.13	10892	10686.57	10848	10630.77
			0.9	10735	10638.30	11023	10693.8	11054	10693.07	10945	10692.77
		0.2	0.1	11039	10670.40	10889	10700.8	11410	10697.23	10968	10653.93
			0.5	11113	10667.50	11000	10697.07	11081	10712.23	10793	10658.87
			0.9	10793	10651.83	**11425**	**10713.0**	11057	10697.7	11159	10680.8
	25	0.1	0.1	11178	10709.37	11093	10679.13	10793	10667.37	11007	10688.33
			0.5	11106	10704.50	11081	10670.27	11113	10657.9	11067	10628.93
			0.9	11025	10672.17	11051	10652.43	11106	10663.37	11138	10710.03
		0.2	0.1	11093	10662.50	11305	10687.93	10889	10677.67	10847	10643.33
			0.5	11082	10682.70	10851	10667.5	11025	10692.53	10918	10640.7
			0.9	11425	10708.63	11139	10700.73	10804	10669.37	11081	10685.67
Ext ($\hat{r}_{i,t}$)	5	0.1	0.1	11007	10657.13	11222	10704.13	11039	10684.9	10808	10662.3
			0.5	11046	10683.53	11064	10724.37	11113	10689.3	10937	10685.33
			0.9	10949	10690.63	11178	10692.93	10835	10666.57	10782	10668.47
		0.2	0.1	11093	10722.43	11093	10684.7	11093	10656.2	11113	10700.27
			0.5	11081	10691.70	11139	10666.73	10931	10660.97	11410	10701.93
			0.9	10949	10634.17	11057	10671.23	10953	10657.57	11007	10682.93
	25	0.1	0.1	11078	10683.90	11132	10691.23	11251	10693.33	11425	10716.57
			0.5	11425	10691.93	11251	10697.4	11113	10708.83	10788	10676.67
			0.9	10990	10707.43	11410	10723.53	10953	10688.03	**11344**	**10770.5**
		0.2	0.1	10968	10681.27	10860	10654.13	11093	10715.7	11113	10723.33
			0.5	11093	10695.57	10990	10700.33	11178	10724.33	11178	10702.57
			0.9	10966	10662.50	11037	10704.23	11132	10713.1	11237	10684.33

weaker.

4.4. Comparative Discussions

The proposed approach, *RLABC*, has been compared with the state-of-art methods, which recently appeared in the literature. Comparative results have been plotted in Figure 3, 4 and 5, where the plots for each algorithm is presented in a color code. Also, they are presented in Table A.7, A.8 and A.9, where the solutions by the state-of-the-art methods and the proposed approach are tabulated separately for benchmarking sets of instances, Set_1, Set_2 and Set_3. The results have been demonstrated with respect to *Best*, *Mean* and *St. Dev* measures standing for the maximum objective value found, the mean and standard deviations values calculated over 30 repetitions, respectively.

The state-of-the-art methods chosen for comparisons are variants of evolutionary algorithms and variants of swarm intelligence algorithms including genetic algorithms (GA) [37], differential evolution (binDE) [49], artificial bee colony algorithms, (BABC, ABCBin) [19, 50] and particle swarm optimisation (GPSO) [48]. All comparative results have been taken from [47, 48], while all ABC algorithmic parameters have been kept the same as the parametric values reported in [47] to keep the comparisons fair. Also, the parameters for operators chosen to take part of *RLABC*

Table 4: The comparison of results for operator selections on the SUKP instances

	RLABC			PM			Random		
Benchmarks	Best	Mean	St.Dev	Best	Mean	St.Dev	Best	Mean	St.Dev
PI_1_1	13283	**13066.6**	55.6908	13251	13056.3	43.8665	13167	13046.7	22.3411
PI_1_2	12274	**12143.8**	74.6502	12274	12137	60.2143	12274	12091.4	81.9539
PI_1_3	13502	**13275.5**	105.159	13405	13266.9	110.006	13405	13178.6	125.441
PI_1_4	14215	**13660.9**	259.819	14215	13640.4	223.115	14215	13488.4	251.618
PI_1_5	11319	10678.2	191.816	11411	**10703**	162.833	11086	10609.1	160.568
PI_1_6	12245	**11742.1**	298.458	12245	11711.1	292.62	12245	11518.4	349.268
PI_1_7	11294	**10757.6**	219.13	11244	10733.1	228.858	11244	10563.7	239.852
PI_1_8	10267	10070.9	127.911	10328	**10071.6**	120.141	10175	9983.42	173.584
PI_1_9	11495	**11229.9**	146.957	11546	11195.3	142.392	11584	11111.2	160.866
PI_1_10	9707	9325.42	161.332	10194	**9361.02**	181.602	10022	9264.62	198.5
PI_2_1	14044	**13936**	82.6673	14044	13920.2	90.6535	14044	13892	95.6615
PI_2_2	13508	13419.8	104.02	13508	**13434**	66.3042	13508	13381.8	103.871
PI_2_3	12374	**11941.4**	226.143	12350	11890.8	211.051	12350	11721.2	232.911
PI_2_4	12063	11688.7	171.281	12317	**11691.9**	187.367	11975	11523.2	219.258
PI_2_5	12677	12575	136.064	12713	**12583.8**	126.988	12644	12499.5	162.942
PI_2_6	11425	10683.3	136.514	11093	**10688.4**	118.312	11178	10635.7	133.911
PI_2_7	11249	**10910.8**	151.625	11310	10861.1	139.574	11282	10789.3	160.374
PI_2_8	10500	**9881.76**	242.734	10725	9880.29	283.202	10355	9754.36	227.384
PI_2_9	10902	**10669.6**	71.0826	10885	10637.9	82.2207	10902	10597.9	122.507
PI_2_10	10194	**9857.12**	224.868	10176	9819	160.244	10176	9682.7	183.942
PI_3_1	12045	**11650.4**	191.287	12020	11590.3	170.981	12020	11504.8	176.992
PI_3_2	12369	**12172.8**	214.534	12369	12156.7	183.883	12369	12013.4	264.661
PI_3_3	13614	**13315.1**	127.952	13609	13307.1	136.457	13500	13215	135.312
PI_3_4	11298	**10849.9**	145.333	11298	10817.5	139.872	10973	10730	143.184
PI_3_5	11538	**11246.9**	183.863	11538	11217.2	191.406	11538	11141.7	169.399
PI_3_6	11502	11024.8	232.097	11590	**11042.9**	216.742	11377	10887.4	232.341
PI_3_7	10180	9941.43	76.98	10397	**9955.62**	104.965	10176	9890.62	76.3551
PI_3_8	9960	**9462.28**	165.694	9865	9439.48	140.503	9831	9364.28	150.273
PI_3_9	10846	**10640.6**	108.112	11018	10603.6	125.988	10823	10519.9	133.872
PI_3_10	9964	**9434.52**	155.976	9686	9389.84	124.552	9788	9283.56	163.665

Table 5: The statistical analysis of results for operator selections on the SUKP instances

	Set_1		Set_2		Set_3	
	Best	Mean	Best	Mean	Best	Mean
RLABC	1.6	1.3	1.5	1.2	1.3	1.4
PM	1.4	1.7	1.5	1.8	1.5	1.6
Random	1.9	3.0	2.0	3.0	2.3	3.0
p-value	0.34	0.0003	0.118	0.0002	0.26	0.0005

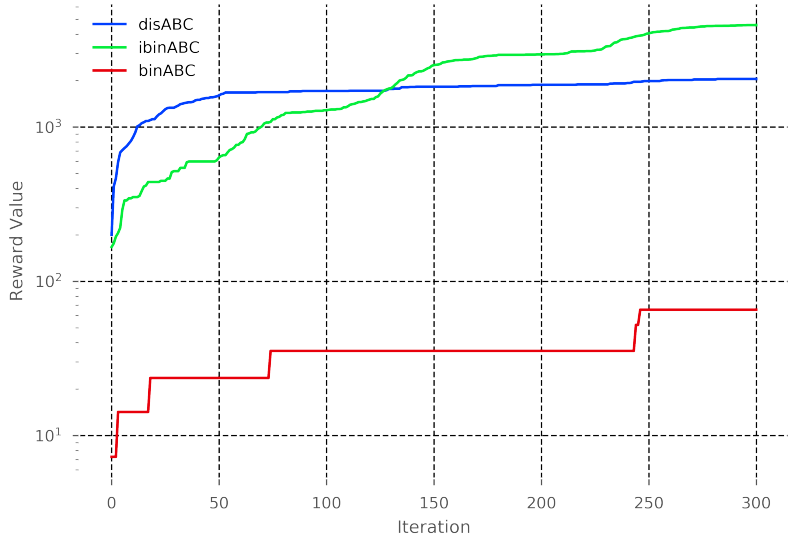
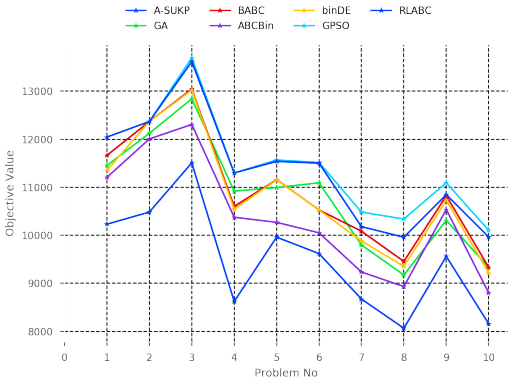


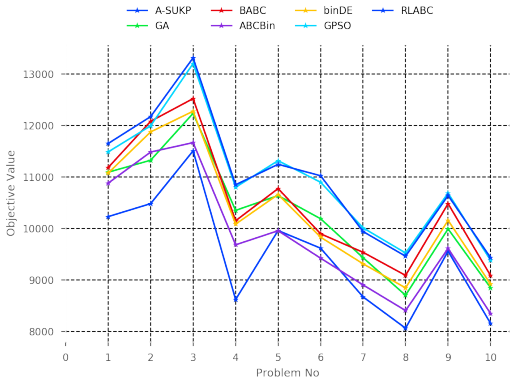
Figure 2: Cumulative reward gained by each operator presented in log scale

have been mirrored from original studies. The results plotted in Figure 3, 4 and 5 and presented in Tables A.7,A.8 and A.9 suggest that the proposed approach, *RLABC*, overwhelmingly performs better than all rival algorithms with respect to *Mean* and *St Dev* measures across the majority of benchmarking instances, while the first runner up algorithm appears to be GPSO, which competes with *RLABC* mainly in terms of *Best* measure and occasionally of other measures. *RLABC*'s competition with GPSO can be seen over the plots presented in Figure 3(a), 4(a) and 5(a) with respect to *Best* measures, while *RLABC*'s strength over GPSO is seen on Figure 3(b), 4(b) and 5(b) in which they look nearly overlapping due to the scale of the plot and and the strength in performance of both over other rivals. In fact, as can be observed from Tables A.7, A.8 and A.9, *RLABC* overwhelmingly solves the instance problems of Set_1 and Set_3 better; winning 8 cases out of 10, while solves 6 out of 10 cases in Set_2. This suggests that although GPSO wins some cases, the robustness does not sound promising.

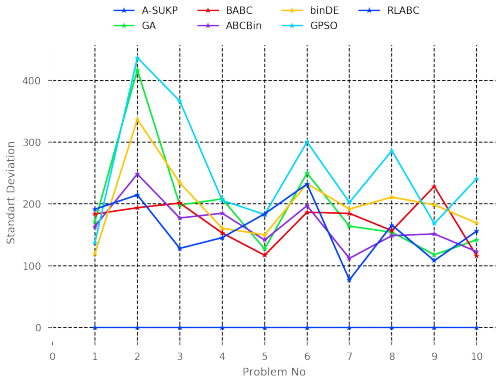
The significance of the results indicate the success of proposed approach over the state-of-the-art methods has been statistically investigated with Friedman test and the results are tabulated in Table 6. As clearly seen, *RLABC* achieves the lowest averaged rank with respect to *Mean*, while competes with GPSO in terms of *Best* measure. The p-values demonstrate the significance in 95%



(a) *Best* values

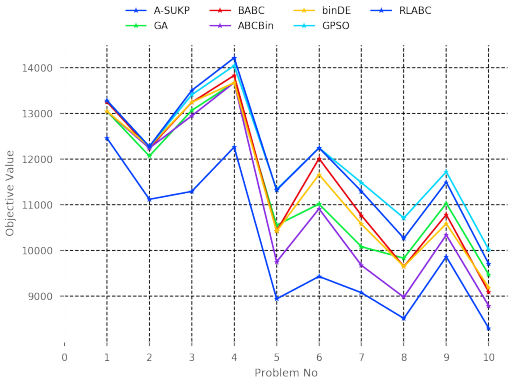


(b) *Mean* values

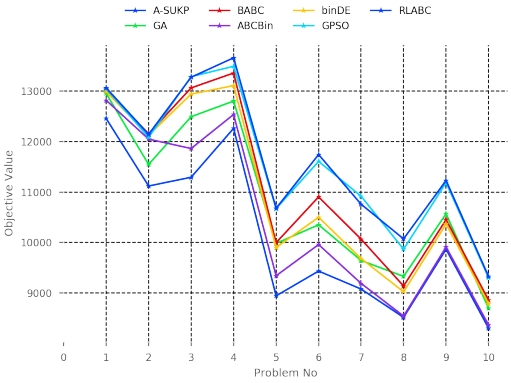


(c) *St. Dev* values

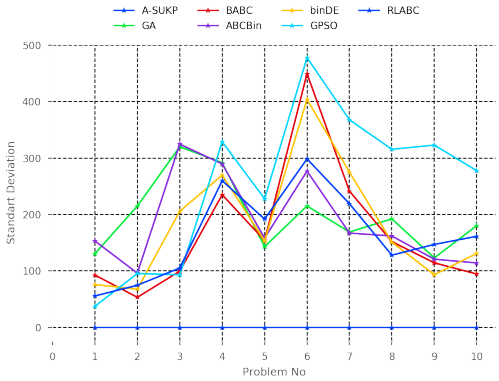
Figure 3: Comparative results for benchmarks in Set_1 by all algorithms



(a) *Best* values

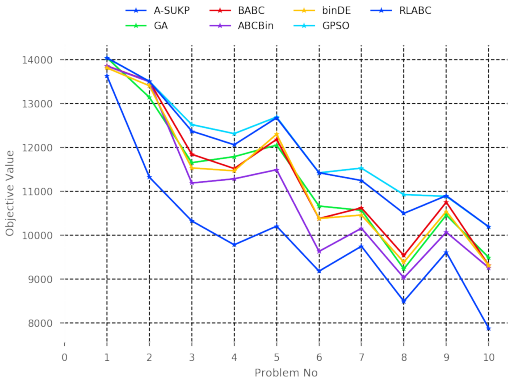


(b) *Mean* values

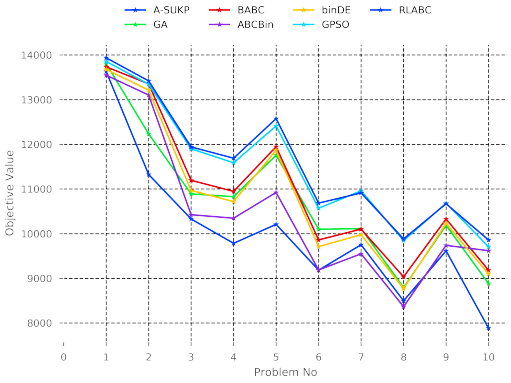


(c) *St. Dev* values

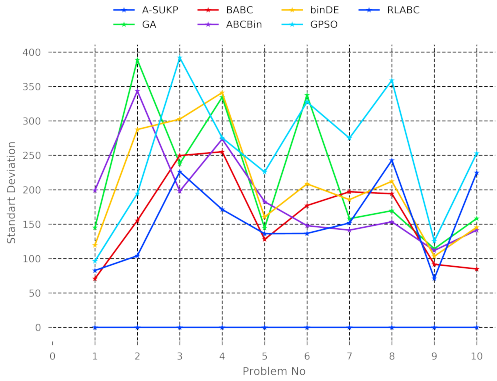
Figure 4: Comparative results for benchmarks in Set_2 by all algorithms



(a) *Best* values



(b) *Mean* values



(c) *St. Dev* values

Figure 5: Comparative results for benchmarks in Set_3 by all algorithms

confidence level for *Mean*, but GPSO remains winner with respect to *Best* measure, which does not support stability and robustness since it fails winning with respect to *Mean* measure.

Table 6: Friedman Test on algorithms with corresponding chi-square ($\tilde{\chi}^2$) and p-values. Proposed method is highlighted with bold.

	Set_1		Set_2		Set_3	
Algorithm	Best	Mean	Best	Mean	Best	Mean
A-SUKP	7.0	7.0	7.0	6.8	7.0	6.7
GA	4.1	4.6	4.6	4.5	3.9	4.4
BABC	3.6	3.0	3.1	3.1	3.3	3.3
ABCBin	5.4	5.9	5.9	6.1	5.6	5.9
binDE	3.8	4.2	3.6	4.4	4.6	4.6
GPSO	1.2	2.0	1.1	1.7	1.1	1.9
RLABC	1.5	**1.3**	1.8	**1.4**	1.5	**1.2**
p-value	1.19E-09	6.68E-10	9.91E-10	5.36E-10	1.45E-09	2.07E-09

5. Conclusion

In this paper, a novel adaptive operator selection scheme has been proposed based on reinforcement learning in which the problem states are mapped to the operators based on the success level per operation. The reinforcement learning algorithm has been developed based on Q-Learning merged with Hard-C-Means clustering algorithm, which is revised to facilitate the search agent to learn how to select the best operator online subject to given circumstances. The approach implies that each operator is represented with a cluster center and the dimensions of the centers are fine-tuned with gained rewards. The approach has been comparatively tested with state-of-art adaptive selection approaches in solving set union knapsack problems formalised in a binary representation. The results suggest that the proposed approach produces much better performance in comparison to the renown adaptive approaches.

The immediate future work of this study is that reinforcement learning algorithms require more investigations to achieve further generalisation to be used in solving more binary problems, especially more combinatorial problems. For longer term, it is needed to achieve transferability of learned experience via RL in problem solving while gained experience with some problem types and instances can be exploited in solving different problem types and instances. It is expected that search algorithms furnished with reinforcement learning facilities are to help develop approaches with which solving moderately difficult problems would be used as training stage and attempting to solve highly complex and larger size problems with adaptive selection schemes constructed via reinforcement learning.

References

[1] Y. Zhang, G.-G. Wang, K. Li, W.-C. Yeh, M. Jian, J. Dong, Enhancing moea/d with information feedback models for large-scale many-objective optimization, Information Sciences 522 (2020) 1–16.

[2] W. Sheng, X. Wang, Z. Wang, Q. Li, Y. Chen, Adaptive memetic differential evolution with niching competition and supporting archive strategies for multimodal optimization, Information Sciences 573 (2021) 316–331.

[3] Z. Cui, J. Zhang, D. Wu, X. Cai, H. Wang, W. Zhang, J. Chen, Hybrid many-objective particle swarm optimization algorithm for green coal production problem, Information Sciences 518 (2020) 256–271.

[4] G. Li, Q. Lin, W. Gao, Multifactorial optimization via explicit multipopulation evolutionary framework, Information Sciences 512 (2020) 1555–1570.

[5] B. Bai, Z. Guo, C. Zhou, W. Zhang, J. Zhang, Application of adaptive reliability importance sampling-based extended domain pso on single mode failure in reliability engineering, Information Sciences 546 (2021) 42–59.

[6] E.-G. Talbi, Metaheuristics: from design to implementation, Vol. 74, John Wiley & Sons, 2009.

[7] K. M. Sallam, S. M. Elsayed, R. A. Sarker, D. L. Essam, Landscape-based adaptive operator selection mechanism for differential evolution, Information Sciences 418 (2017) 383–404.

[8] G. Wu, R. Mallipeddi, P. N. Suganthan, Ensemble strategies for population-based optimization algorithms–a survey, Swarm and evolutionary computation 44 (2019) 695–711.

[9] R. Durgut, M. E. Aydin, Adaptive binary artificial bee colony algorithm, Applied Soft Computing 101 (2021) 107054.

[10] R. Durgut, I. B. Yavuz, M. E. Aydin, Solving set union knapsack problems with adaptive binary artificial bee colony, Journal of Intelligent Systems: Theory and Applications (2021) 1–10.

[11] E.-G. Talbi, Machine learning into metaheuristics: A survey and taxonomy, ACM Computing Surveys (CSUR) 54 (6) (2021) 1–32.

[12] I.-S. Comşa, S. Zhang, M. E. Aydin, P. Kuonen, Y. Lu, R. Trestian, G. Ghinea, Towards 5g: A reinforcement learning-based scheduling solution for data traffic management, IEEE Transactions on Network and Service Management 15 (4) (2018) 1661–1675.

[13] M. E. Aydin, E. Öztemel, Dynamic job-shop scheduling using reinforcement learning agents, Robotics and Autonomous Systems 33 (2-3) (2000) 169–178.

[14] D. Kizilay, M. F. Tasgetiren, H. Oztop, L. Kandiller, P. N. Suganthan, A differential evolution algorithm with q-learning for solving engineering design problems, in: 2020 IEEE Congress on Evolutionary Computation (CEC), 2020, pp. 1–8.

[15] R. R. Gharieb, G. Gendy, H. Selim, A hard c-means clustering algorithm incorporating membership kl divergence and local data information for noisy image segmentation, International Journal of Pattern Recognition and Artificial Intelligence 32 (04) (2018) 1850012.

[16] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE computational intelligence magazine 1 (4) (2006) 28–39.

[17] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-international conference on neural networks, Vol. 4, IEEE, 1995, pp. 1942–1948.

[18] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm, Journal of global optimization 39 (3) (2007) 459–471.

[19] C. Ozturk, E. Hancer, D. Karaboga, A novel binary artificial bee colony algorithm based on genetic operators, Information Sciences 297 (2015) 154–170.

[20] J. Verheul, The influence of using adaptive operator selection in a multiobjective evolutionary algorithm based on decomposition, Master's thesis, Utrecht University (2020).

[21] K. Li, A. Fialho, S. Kwong, Q. Zhang, Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 18 (1) (2013) 114–130.

[22] L. Davis, Adapting operator probabilities in genetic algorithms, in: Proceedings of the third international conference on Genetic algorithms, 1989, pp. 61–69.

[23] J. M. Whitacre, T. Q. Pham, R. A. Sarker, Use of statistical outlier detection method in adaptive evolutionary algorithms, in: Proceedings of the 8th annual conference on Genetic and evolutionary computation, 2006, pp. 1345–1352.

[24] Á. Fialho, M. Schoenauer, M. Sebag, Toward comparison-based adaptive operator selection, in: Proceedings of the 12th annual conference on Genetic and evolutionary computation, 2010, pp. 767–774.

[25] F. G. Lobo, D. E. Goldberg, Decision making in a hybrid genetic algorithm, in: Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97), IEEE, 1997, pp. 121–125.

[26] A. Fialho, L. Da Costa, M. Schoenauer, M. Sebag, Analyzing bandit-based adaptive operator selection mechanisms, Annals of Mathematics and Artificial Intelligence 60 (1) (2010) 25–64.

[27] B. A. Julstrom, What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm, in: Proceedings of the 6th International Conference on Genetic Algorithms, 1995, pp. 81–87.

[28] J. Maturana, F. Lardeux, F. Saubion, Autonomous operator management for evolutionary algorithms, Journal of Heuristics 16 (6) (2010) 881–909.

[29] D. E. Goldberg, Probability matching, the magnitude of reinforcement, and classifier system bidding, Machine Learning 5 (4) (1990) 407–425.

[30] D. Thierens, An adaptive pursuit strategy for allocating operator probabilities, in: Proceedings of the 7th annual conference on Genetic and evolutionary computation, 2005, pp. 1539–1546.

[31] L. DaCosta, A. Fialho, M. Schoenauer, M. Sebag, Adaptive operator selection with dynamic multi-armed bandits, in: Proceedings of the 10th annual conference on Genetic and evolutionary computation, 2008, pp. 913–920.

[32] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine learning 47 (2) (2002) 235–256.

[33] F. V. Nepomuceno, A. P. Engelbrecht, A self-adaptive heterogeneous pso inspired by ants, in: International Conference on Swarm Intelligence, Springer, 2012, pp. 188–195.

[34] W.-x. Wang, K.-s. Li, X.-z. Tao, F.-h. Gu, An improved moea/d algorithm with an adaptive evolutionary strategy, Information Sciences 539 (2020) 1–15.

[35] R. S. Sutton, A. G. Barto, Reinforcement Learning, Second Edition: An Introduction: An Introduction, The MIT Press, 2018.

[36] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Processing Magazine 34 (6) (2017) 26–38.

[37] O. Goldschmidt, D. Nehme, G. Yu, Note: On the set-union knapsack problem, Naval Research Logistics (NRL) 41 (6) (1994) 833–842.

[38] C. Wu, Y. He, Solving the set-union knapsack problem by a novel hybrid jaya algorithm, Soft Computing 24 (2020) 1883–1902.

[39] Q. Lin, Z. Liu, Q. Yan, Z. Du, C. A. C. Coello, Z. Liang, W. Wang, J. Chen, Adaptive composite operator selection and parameter control for multiobjective evolutionary algorithm, Information Sciences 339 (2016) 332–352.

[40] B. Bischl, O. Mersmann, H. Trautmann, M. Preuß, Algorithm selection based on exploratory landscape analysis and cost-sensitive learning, in: Proceedings of the 14th annual conference on Genetic and evolutionary computation, 2012, pp. 313–320.

[41] P. A. Consoli, Y. Mei, L. L. Minku, X. Yao, Dynamic selection of evolutionary operators based on online learning and fitness landscape analysis, Soft Computing 20 (10) (2016) 3889–3914.

[42] C. J. C. H. Watkins, Learning from delayed rewards, Ph.D. thesis, Cambridge University (1989).

[43] T. J. Ross, Fuzzy Classification, John Wiley & Sons, Ltd, 2010, Ch. 10, pp. 332–368.

[44] M. S. Kiran, M. Gündüz, Xor-based artificial bee colony algorithm for binary optimization, Turkish Journal of Electrical Engineering & Computer Sciences 21 (Sup. 2) (2013) 2307–2328.

[45] R. Durgut, Improved binary artificial bee colony algorithm, Frontiers of Information Technology & Electronic Engineering 22 (8) (2021) 1080–1091.

[46] M. H. Kashan, N. Nahavandi, A. H. Kashan, Disabc: a new artificial bee colony algorithm for binary optimization, Applied Soft Computing 12 (1) (2012) 342–352.

[47] Y. He, H. Xie, T.-L. Wong, X. Wang, A novel binary artificial bee colony algorithm for the set-union knapsack problem, Future Generation Computer Systems 78 (2018) 77–86.

[48] F. B. Ozsoydan, A. Baykasoglu, A swarm intelligence-based algorithm for the set-union knapsack problem, Future Generation Computer Systems 93 (2019) 560–569.

[49] A. P. Engelbrecht, G. Pampara, Binary differential evolution strategies, in: 2007 IEEE congress on evolutionary computation, IEEE, 2007, pp. 1942–1947.

[50] M. S. Kiran, The continuous artificial bee colony algorithm for binary optimization, Applied Soft Computing 33 (2015) 15–23.

Appendix A. Comparison tables for all benchmark instances

Table A.7: Comparison with state-of-art methods for Set_1 problem instances

Benchmarks		A-SUKP	GA	BABC	ABCBin	binDE	GPSO	RLABC
PI_1_1	Best	12459	13044	13251	13044	13044	13283	13283
	Mean	12459	12956	13029	12819	12991	13051	**13066.58**
	Std	0	130.66	92.63	153.06	75.95	37.41	55.69079
PI_1_2	Best	11119	12066	12238	12238	12274	12274	12274
	Mean	11119	11546	12155	12049	12124	12085	**12143.75**
	St Dev	0	214.94	53.29	96.11	67.61	95.38	74.65017
PI_1_3	Best	11292	13064	13241	12946	13241	13405	13502
	Mean	11292	12493	13064	11862	12941	**13287**	13275.5
	St Dev	0	320.03	99.57	324.65	205.7	93.18	105.1587
PI_1_4	Best	12262	13671	13829	13671	13671	14044	14215
	Mean	12262	12803	13359	12537	13110	13493	**13660.93**
	St Dev	0	291.66	234.99	289.53	269.69	328.72	259.8194
PI_1_5	Best	8941	10553	10428	9751	10420	11335	11319
	Mean	8941	9981	9995	9339	9899	10670	**10678.22**
	St Dev	0	142.97	154.03	158.15	153.18	227.85	191.8163
PI_1_6	Best	9432	11016	12012	10913	11661	12245	12245
	Mean	9432	10350	10903	9958	10499	11607	**11742.11**
	St Dev	0	215.13	449.45	276.9	403.95	477.8	298.4585
PI_1_7	Best	9076	10083	10766	9674	10576	11484	11294
	Mean	9076	9642	10065	9188	9681	**10916**	10757.61
	St Dev	0	168.94	241.45	167.08	275.05	367.75	219.1303
PI_1_8	Best	8514	9831	9649	8978	9649	10710	10267
	Mean	8514	9327	9136	8540	9021	9865	**10070.94**
	St Dev	0	192.2	151.9	161.83	150.99	315.38	127.9108
PI_1_9	Best	9864	11031	10784	10340	10586	11722	11495
	Mean	9864	10568	10452	9910	10364	11185	**11229.93**
	St Dev	0	123.15	114.35	120.82	93.39	322.98	146.9566
PI_1_10	Best	8299	9472	9090	8789	9191	10022	9707
	Mean	8299	8693	8858	8364	8784	9300	**9325.42**
	St Dev	0	180.12	94.55	114.1	131.05	277.62	161.3318

-

Table A.8: Comparison with state-of-art methods for Set_2 problem instances

Benchmarks		A-SUKP	GA	BABC	ABCBin	binDE	GPSO	RLABC
PI_2_1	Best	10231	11454	11664	11206	11352	12045	12045
	Mean	10231	11093	11183	10880	11075	11487	**11650.44**
	St Dev	0	171.22	183.57	163.62	119.42	137.52	191.2869
PI_2_2	Best	10483	12124	12369	12006	12369	12369	12369
	Mean	10483	11326	12082	11485	11876	11994	**12172.78**
	St Dev	0	417	193.79	248.33	336.94	436.81	214.5339
PI_2_3	Best	11508	12841	13047	12308	13024	13696	13614
	Mean	11508	12237	12523	11668	12278	13204	**13315.14**
	St Dev	0	198.18	201.35	177.14	234.24	366.56	127.9523
PI_2_4	Best	8621	10920	10602	10376	10547	11298	11298
	Mean	8621	10352	10151	9684	10085	10801	**10849.9**
	St Dev	0	208.08	152.91	184.84	160.6	205.76	145.3334
PI_2_5	Best	9961	10994	11158	10269	11152	11568	11538
	Mean	9961	10640	10776	9957	10661	**11318**	11246.91
	St Dev	0	126.84	116.8	141.48	149.84	182.82	183.8626
PI_2_6	Best	9618	11093	10528	10051	10528	11517	11502
	Mean	9618	10190	9898	9424	9832	10899	**11024.78**
	St Dev	0	249.76	186.53	197.14	232.72	300.36	232.0975
PI_2_7	Best	8672	9799	10085	9235	9883	10483	10180
	Mean	8672	9433	9538	8905	9315	**10013**	9941.43
	St Dev	0	163.84	184.62	111.85	191.59	202.4	76.98003
PI_2_8	Best	8064	9173	9456	8932	9352	10338	9960
	Mean	8064	8704	9090	8407	8847	**9525**	9462.28
	St Dev	0	154.15	156.69	148.52	210.91	286.16	165.6945
PI_2_9	Best	9559	10311	10823	10537	10728	11094	10846
	Mean	9559	9993	10483	9615	10159	**10688**	10640.58
	St Dev	0	117.73	228.34	151.41	198.49	168.06	108.1116
PI_2_10	Best	8157	9329	9333	8799	9218	10104	9964
	Mean	8157	8849	9086	8348	8920	9383	**9434.52**
	St Dev	0	141.84	115.62	122.65	168.9	241.01	155.9758

Table A.9: Comparison with state-of-art methods for Set_3 problem instances

Benchmarks		A-SUKP	GA	BABC	ABCBin	binDE	GPSO	RLABC
PI_3_1	Best	13634	14044	13860	13860	13814	14044	14044
	Mean	13634	13806	13735	13547	13676	13855	**13935.99**
	St Dev	0	144.91	70.76	199.11	119.53	96.23	82.66734
PI_3_2	Best	11325	13145	13508	13498	13407	13508	13508
	Mean	11325	12235	13352	13103	13212	13347	**13419.77**
	St Dev	0	388.66	155.14	343.46	287.45	194.34	104.0195
PI_3_3	Best	10328	11656	11846	11191	11535	12522	12374
	Mean	10328	10889	11194	10424	10969	11899	**11941.37**
	St Dev	0	237.85	249.58	197.88	302.52	391.83	226.143
PI_3_4	Best	9784	11792	11521	11287	11469	12317	12063
	Mean	9784	10828	10945	10346	10717	11585	**11688.66**
	St Dev	0	334.43	255.14	273.47	341.08	275.32	171.2813
PI_3_5	Best	10208	12055	12186	11494	12304	12695	12677
	Mean	10208	11755	11946	10922	11865	12411	**12574.97**
	St Dev	0	144.45	127.8	182.63	160.42	225.8	136.0644
PI_3_6	Best	9183	10666	10382	9633	10382	11425	11425
	Mean	9183	10099	9860	9187	9710	10568	**10683.27**
	St Dev	0	337.42	177.02	147.78	208.48	327.48	136.5139
PI_3_7	Best	9751	10570	10626	10160	10462	11531	11249
	Mean	9751	10112	10101	9549	9976	**10959**	10910.82
	St Dev	0	157.89	196.99	141.27	185.57	274.9	151.6247
PI_3_8	Best	8497	9235	9541	9033	9388	10927	10500
	Mean	8497	8794	9033	8366	8768	9845	**9881.76**
	St Dev	0	169.52	194.18	153.4	212.24	358.91	242.734
PI_3_9	Best	9615	10460	10755	10071	10546	10888	10902
	Mean	9615	10185	10328	9738	10228	**10681**	10669.59
	St Dev	0	114.19	91.61	111.64	103.32	125.36	71.08264
PI_3_10	Best	7883	9496	9318	9262	9312	10194	10194
	Mean	7883	8883	9181	9618	9096	9704	**9857.12**
	St Dev	0	158.21	84.91	141.32	145.45	252.84	224.8679