ISSN: 2229-6956 (ONLINE)
DOI: 10.21917/ijsc.2025.0559

IMPROVED IMPERIALIST COMPETITIVE ALGORITHM FOR SELECTION PROBLEMS IN COMBINATORIAL OPTIMIZATION

Laxmikant[1], C. Vasantha Lakshmi[2] and C. Patvardhan[3]
[1,2]Department of Physics and Computer Science, Faculty of Science, Dayalbagh Educational Institute, India
[3]Department of Electrical Engineering, Faculty of Engineering, Dayalbagh Educational Institute, India

Abstract

Metaheuristics have been used to solve combinatorial optimization problems in recent decades. Metaheuristics inspired by various natural phenomena have been proposed due to their optimization characteristics. The Imperialist Competitive Algorithm (ICA) is one such metaheuristic inspired by the socio-political process of imperialism. ICA has become popular due to its extensive applications in various engineering domains. Originally, ICA was designed to solve continuous optimization problems. This paper presents a binary version of ICA, dubbed ICA with Binary-encoding (ICAwB), to solve selection problems. ICAwB works with binary encoding and utilizes new socio-politically inspired operators. Additional features are incorporated within ICAwB to develop an improved version dubbed IICAwB. ICAwB and IICAwB with other binary versions of ICA are compared. IICAwB shows much better performance than existing binary ICAs and ICAwB. The proposed IICAwB is quite generic, and its applicability to other combinatorial optimization problems can be attempted with advantage.

Keywords:

Metaheuristics, Knapsack problem, Discrete optimization, Imperialist Competitive Algorithm, Evolutionary Computation

1. INTRODUCTION

Over the last few decades, evolutionary computation metaheuristics have been extensively utilized to solve combinatorial optimization problems. These metaheuristics are generic population-based computational approaches that provide iterative improvement to a population of solution individuals directed by their fitness. Reproduction, mutation, recombination, selection, and migration are typical natural biological concepts used in evolutionary algorithms. The under applicability of these metaheuristics over exact algorithms is due to their simple concepts, easy implementation, and provision of satisfying results within reasonable computation time.

Evolution Strategies (ES) [1], Genetic algorithms (GA) [2], Particle Swarm Optimization (PSO) [3], Ant Colony Optimization (ACO) [4], and Imperialist Competitive Algorithm (ICA) [5] are some well-known metaheuristics. Evolution Strategies (ES) were introduced by [1] and further developed by [6]. ES uses an asexual reproduction mutation operator only. GA [2] offers a basic evolutionary computation model. It imitates the biological process of natural selection. PSO [3] is inspired by the swarm-intelligence behaviour of organisms in a bird flock or fish school. ACO [4] derives inspiration from real ants searching for food. It mimics the path-following behaviour of ants. ICA [5] is inspired by the socio-political process of imperialism and imperialistic competition. Recently, ICA has become popular due to its application in solving complex continuous problems. It has been used in various application areas, including industrial engineering, civil engineering, mechanical engineering, electrical engineering, and computer engineering [7].

ICA starts with an initial population of solution vectors termed Countries. These Countries are further classified into Imperialists and Colonies. The Imperialists acquire Colonies to form Empires. An Imperialist is a Country with the best cost/fitness in an Empire, while other Countries are its Colonies. Empires try to enhance their power by enhancing their Colonies and acquiring weaker Colonies from other Empires. Assimilation and revolution processes are used to improve Colonies by moving them toward their Imperialist. In assimilation, Colonies move some distance toward their Imperialists. However, in revolution, the movement of Colonies in the direction of their Imperialist deviates from some angle. If a Colony outperforms its Imperialist during these processes, their roles are interchanged. ICA starts an imperialistic competition between Empires aiming to acquire Colonies from weaker Empires. This competition leads to better Empires acquiring more Colonies, which increases exploration in ICA and eventually to a near-global optimum solution. Finally, only one Empire remains, which is the desired state to terminate the algorithm. The pseudo-code of an ICA is provided in Algorithm 1.

Algorithm 1: Pseudo-code of Imperialist Competitive Algorithm

1: Generate some random solutions (Countries)
2: Initialize the Empires
3: Repeat
4: Assimilate Empires: move the Colonies toward their Imperialist
5: Revolution: move the Colonies toward their Imperialist with some deviation
6: Interchange roles of Colony and Imperialist if a Colony outperforms Imperialist
7: Imperialistic competition: Empires fight to acquire more and more Colonies from weaker Empires
8: Eliminate Empires with no Colonies
9: Until only one Empire remains

As described in Algorithm.1, this basic ICA is designed for continuous optimization problems. However, many optimization problems are discrete – travelling salesman problems, subset selection problems, and vehicle routing problems, to name but a few. Subset selection problems involve selection rather than sequencing or ordering. If S is a set of objects, then a subset X of S(i.e., $X \subseteq S$) is to be found, such that for any other subset X' of S, $f(X) \geq f(X')$ for a maximization ($f(X) \leq f(X')$ for a minimization) problem. A subset selection problem means finding an optimal feasible subset of objects from a given set. Clique, satisfiability, and knapsack problems are some well-known subset selection problems. Knapsack problems (KPs) have multiple real-world optimization applications: budget control, cutting stock, airline cargo loading, and others. Several other optimization problems

can also be reduced or transformed to KPs easily, viz., clique and dense subgraph problems, to name a few. [8]. The basic KP is a 0-1 or binary knapsack problem. Furthermore, other extensions and variants of KP have also become standard problems independently.

The basic model of ICA applies only to continuous optimization problems, as its operators do not directly apply to binary-encoded solution vectors. However, several discrete versions of ICA have also been developed and reported in the literature, viz., Discrete Binary ICA (DB-ICA) [9], Discrete ICA (DICA) [10], Modified ICA (MICA) [11], and Binary ICA (BICA) [12] and others. DB-ICA uses a two-point crossover to simulate the binary assimilation process. DICA and MICA use crossovers for assimilation and are primarily utilized for ordering problems. BICA uses nine transfer functions for binary assimilation. It converts the distance between a Colony and its Imperialist, say d, into a probability value using a transfer function $F(d)$ (see Table.1). According to this probability value, the Colonies move towards their Imperialist by changing 0s and 1s. For transfer functions TF1 through TF4, Eq.(1) and TF5 through TF9, Eq.(2) are utilized to update the colony position; $x_i(t)$ represents i^{th} bit of a Country/Colony at time t. It is reported in [12] that BICA outperforms DB-ICA. Therefore, in the present article, the results of our proposed algorithm are only compared with those of BICA.

$$x_i(t+1) = \begin{cases} 0, & \text{if rand} < F(d_i(t+1)) \\ 1, & \text{if rand} \geq F(d_i(t+1)) \end{cases} \quad (1)$$

$$x_i(t+1) = \begin{cases} \text{Not}(x_i(t)), & \text{if rand} < F(d_i(t+1)) \\ x_i(t), & \text{if rand} \geq F(d_i(t+1)) \end{cases} \quad (2)$$

This paper presents a novel binary version of the Imperialist Competitive Algorithm (ICA) dubbed ICA with Binary-encoding (ICAwB). A new intuitive binary assimilation process is devised for this purpose. Furthermore, an improved version of ICAwB, dubbed Improved ICAwB (IICAwB), is proposed with several enhancements. The computational performance of these new algorithms is compared with the existing binary version of ICA (BICA). The proposed algorithms perform much better than the previously reported ICA algorithms in terms of solution quality and consistency of convergence. The proposed ICAwB and IICAwB are generic, can be employed with an advantage on other subset selection problems and can be adapted suitably for other combinatorial optimization problems. The performance of these algorithms is demonstrated on some standard Knapsack problems.

Table.1. Transfer Functions [12]

Name	Transfer function
TF1	$F(d) = \dfrac{1}{1+e^{-2d}}$
TF2	$F(d) = \dfrac{1}{1+e^{-d}}$
TF3	$F(d) = \dfrac{1}{1+e^{-d/2}}$
TF4	$F(d) = \dfrac{1}{1+e^{-d/3}}$
TF5	$F(d) = \left\lvert \text{erf}\left(\dfrac{\sqrt{\pi}}{2}d\right)\right\rvert = \left\lvert \dfrac{2}{\sqrt{\pi}}\int_0^{\frac{\sqrt{\pi}}{2}d} e^{-t^2}\,dt\right\rvert$
TF6	$F(d) = \tanh(d)$
TF7	$F(d) = \left\lvert \dfrac{d}{\sqrt{1+d^2}}\right\rvert$
TF8	$F(d) = \left\lvert \dfrac{2}{\pi}\arctan\left(\dfrac{\pi}{2}d\right)\right\rvert$
TF9	$F(d) = 2\times\left\lvert \dfrac{1}{1+e^{-d}} - 0.5\right\rvert$

The remainder of the paper is organized as follows. ICAwB and its improved version, IICAwB, are explained in section 2. Section 3 describes three problems considered, i.e., Difficult Knapsack problems (DKP), Quadratic Knapsack problems (QKP) and Quadratic Multiple Knapsack Problems (QMKP). The computational experiments and the selection of parameter values are described in section 4. The computational performance of BICA, ICAwB, and IICAwB on DKP, QKP and QMKP instances is reported and discussed in section 5. Conclusions are presented in section 6.

2. PROPOSED ALGORITHMS

2.1 PROPOSED IMPERIALIST COMPETITIVE ALGORITHM WITH BINARY-ENCODING (ICAwB)

ICAwB starts with a random population of solution individuals termed as Countries. A Country X in ICAwB is represented as a binary vector in N_{var} dimensional search space as in Eq.(3). In a selection problem, each binary value x_i in a Country X represents either inclusion (1) or exclusion (0) of objects in the solution.

$$X = [x_1, x_2, \ldots, x_{N_{\text{var}}}], \text{ where } x_i \in \{0,1\} \quad (3)$$

A random population of N_{pop} such Countries is initially generated, out of which N_{imp} Imperialists are designated. The selection of Imperialists is based on their profit $f(X_i)$ as given in Eq.(4), which is a counterpart of fitness in GA.

$$\text{profit } f(X_i) = f(x_{i1}, x_{i2}, \ldots, x_{iN_{\text{var}}}) \quad (4)$$

First, N_{imp} Countries with the best profit are selected as the Imperialists. These Imperialists create Empires by acquiring the remaining (N_{pop} - N_{imp}) Countries as their Colonies. The initial number of Colonies of the n^{th} Imperialist $\left(X_{\text{imp}_n}\right)$ is proportionate to its profit, as given in Eq.5. Thus, a better Imperialist occupies more Colonies and creates an enormous Empire. Colonies are occupied using the roulette-wheel selection process [13].

$$N.X_{\text{imp}_n} = \text{round}\left\lvert \frac{f(X_{\text{imp}_n})}{\sum_{i=1}^{N_{\text{imp}}} f(X_{\text{imp}_i})}\right\rvert \times (N_{\text{pop}} - N_{\text{imp}}) \quad (5)$$

Empires try to improve their Colonies with assimilation and revolution processes. In assimilation, Imperialists impose better characteristics on the acquired Colonies to increase their total power. Assimilation is implemented by moving the Colonies

toward their corresponding Imperialist. The Fig.1 depicts an example of the assimilation process; Colonies are represented as dots, and the Imperialist as a star. In ICAwB, the movement of Colonies is done based on the hamming distance. A Colony at hamming distance D from its Imperialist will move d bits (Eq.6) toward the Imperialist. These d bits are randomly selected from the hamming distance bit-vector. The pseudo-code given in Algorithm.2 describes the proposed binary assimilation process in detail.

$$d \sim U(0, D) \tag{6}$$

where, U is the integer uniform distribution function.

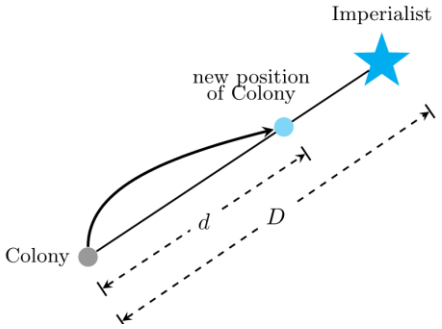

Fig.1. Assimilation process

A random disturbance in a Colony can prevent it from falling into a local optimum. A revolution process utilized for exploitation achieves this disturbance. In the revolution process, an arbitrary change in a Colony is made. In ICAwB, it is performed using a simple mutation operator.

Variation in Colonies through assimilation and revolution operators can provide better-performing solutions. If these solutions are better than their Imperialist, then the roles of the best-performing Colony and its Imperialist are interchanged. This mechanism maintains the status of the best local solution (Imperialist) as an inspirational body, and the assimilation operator will subsequently use this updated Imperialist to improve the Colonies.

Algorithm 2: Pseudo-code of the proposed binary assimilation process

1: Calculate the distance D between the Colony and its Imperialist $\left(N.X_{\text{imp}_n}\right)$
2: Calculate the number of bits d to move toward Imperialist using Eq.(6)
3: Select Colony's d bits out of D bits randomly and change according to $\left(N.X_{\text{imp}_n}\right)$
4: If new_colony outperforms the Colony, then
5: Keep the new_colony, discard the Colony
6: Else
7: Discard the new_colony
8: End If

As in the case of ICA, Empires try to improve their power or at least maintain it. ICAwB starts a competition between Empires to increase their power and get more and more Colonies. For this, better Empires try to capture the weakest Colony (in terms of their profit) from the worst Empire (Fig.2). The discrimination of Empires is done based on their power (PE). The power PE_n of an Empire E_n is calculated as an aggregated sum of its Imperialist and a fraction of its Colonies' mean power as in Eq.7. The value of ξ represents the weightage of the Colonies in an Empire's power. A lower value of ξ (≈0.25) is recommended to prioritize the Imperialist's profit while calculating an Empire's power.

$$P.E_n = \text{profit}(X_{\text{imp}_n}) + \xi \cdot \frac{\text{profit(Colonies of } E_n)}{N.X_{\text{imp}_n}} \tag{7}$$

In this imperialistic competition process, weaker Empires become weaker and collapse into stronger Empires. By acquiring more Colonies, the better Empires can improve their exploration, resulting in better solutions. Due to imperialistic competition, only one Empire remains at last, which is a desirable condition to terminate the algorithm. However, a maximum number of generations is also a better termination condition and is utilized in this article.

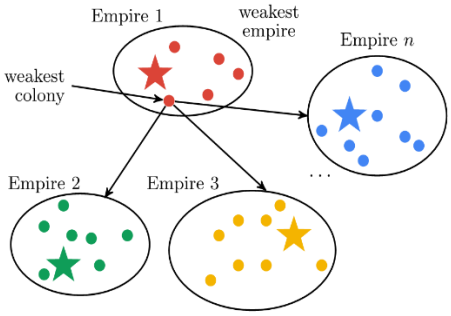

Fig.2. Imperialistic competition

2.2 IMPROVED ICAWB (IICAWB)

The ICAwB is further improved for constrained combinatorial optimization problems. Various enhancements (summarized in Table.2) are proposed in the ICAwB framework to improve the search process.

2.2.1 Better Initialization:

Problem-specific greedy approaches based on the efficiencies of the objects can be utilized to generate better initial solutions in the initial run [14]. IICAwB incorporates seeding the initial population with a solution generated with some problem domain-specific heuristic. This seeded solution provides a better start-off and improves the convergence speed.

2.2.2 Assimilation between Imperialists:

Fast convergence due to seeding better solutions sometimes invites the problem of falling in local optima, as enunciated in the No Free Lunch theorem [15] [16]. Thus, better exploration is required to be incorporated with heuristics in initialization. For this purpose, in IICAwB, assimilation is also performed between the Imperialists, i.e., Imperialists move toward the best Imperialist.

2.2.3 Repair Operator:

The problem-specific heuristics utilized for better initialization can also be used for repairing infeasible solutions. The sequence or sequences generated using heuristics provide an ordering mechanism considered while selecting an object before others in selection problems [17]. The reverse of the sequence is utilized to remove objects from an infeasible solution vector.

2.2.4 Local Improvement:

A problem domain-specific local improvement mechanism can improve a partial or complete solution. In a selection problem, objects in a solution are replaced repetitively with those not in the solution if this swapping process improves the solution quality. Moreover, if the given constraint is not violated, one or more objects already in the solution are selected randomly, and excluded objects are selected for inclusion following a greedy approach.

Table.2. Algorithmic level comparison between ICAwB and IICAwB

Step	ICAwB	IICAwB	Implementation details
Initializing the best solution using a heuristic	No	Yes	The best solution generated using heuristic-based greedy order is included in the initial population. This solution provides the initial best solution for the first run.
Assimilation between Imperialists	No	Yes	Imperialists also moved toward the best Imperialist to maintain exploration.
Repairing using heuristic	No	Yes	Infeasible solutions are repaired using a greedy order. Objects are included in decreasing order of their profitability, defined by their value-weight ratio. However, their exclusion is based on their increasing order of value-weight ratio.
Local Improvement	No	Yes	Random objects are excluded from the solution and included using a random or greedy order.

3. PROBLEM DESCRIPTIONS AND DATASETS

The effectiveness of these proposed ICAwB and IICAwB for solving real-life NP-hard combinatorial optimization problems is demonstrated on three subset selection problems – Difficult 0-1 knapsack problems, Quadratic knapsack problems and Quadratic multiple knapsack problems. These problems are described as follows.

3.1 DIFFICULT 0-1 KNAPSACK PROBLEMS

In a 0-1 KP, a set of n items and a knapsack with a positive capacity C are given. Each item i in general KP has some associated positive weight w_i and value v_i. Several subsets of items can be generated from this set. The problem is to find a subset of such items that maximizes the overall sum of values while the sum of their weight does not exceed the knapsack capacity C.

$$\text{maximize} \sum_{i=1}^{n} v_i x_i \tag{8}$$

$$\text{subject to:} \sum_{i=1}^{n} w_i x_i \le C, \tag{9}$$

$$x_i \in \{0,1\}, \text{where } i = 1,2,\ldots,n \tag{10}$$

0-1 KPs are considered one of the easier NP-hard problems. These can be solved in pseudo-polynomial time using dynamic programming [18]. However, benchmark knapsack instances are difficult in practice [19], [20]. Most exact algorithms perform poorly on them. These instances are termed Difficult Knapsack Problems (DKPs). [21] reported nine such instance categories: uncorrelated data instances, weakly correlated instances, strongly correlated instances, inverse strongly correlated instances, almost strongly correlated instances, subset-sum, uncorrelated instances with similar weights, even-odd subset, and even-odd knapsack. These instances are categorized according to the correlation between the weights and values of items. Among these, uncorrelated instances are sometimes easy to solve. However, other instances are of varying difficulty.

For this study, three hard knapsack category instances are picked: uncorrelated, weakly correlated, and strongly correlated. In these instances, weights (w_i) of the objects were generated using uniform distribution in the data range R=1000, while values (v_i) were generated as a weights function. In uncorrelated data instances, values and weights of objects are chosen randomly in [1,R]. Hence, there is no visible correlation between w_{is} and v_{is}. In weakly correlated data instances, the weights are chosen randomly in [1,R] and the non-zero values in [w_i-R/10,w_i+R/10]. All instances' values are chosen as v_i=w_i+R/10 in strongly correlated data instances.

A good greedy solution to these problems is the selection of elements in non-increasing order of their corresponding value-weight ratios (v_i/w_i) until the knapsack constraint is not violated. However, the problem instances with poor greedy solutions tend to be harder to tackle [14]. This greedy order provides an improved repairing mechanism for infeasible solution vectors; see Algorithm.3.

Algorithm 3: Pseudo-code for *RepairSolutionGreedy(X)*

1: Knapsack_overfilled ← false

2: $w \leftarrow \sum_{i=1}^{n} w_i x_i$

3: If w > capacity, then

4: Knapsack_overfilled ← true

5: End If

6: While Knapsack_overfilled Do

7: Select i^{th} object with the smallest value-weight ratio from the knapsack

8: x_i←0 (i.e., remove i^{th} object from the knapsack)

9: w←w-w_i

10: If $w \le$ capacity, then

11: Knapsack_overfilled ← false

12: End If

13: End While

14: For each i^{th} object not in knapsack considered in decreasing order of value-weight ratio, do

15: If $w + w_i \le$ *capacity* then

16: $x_i \leftarrow 1$ (i.e., include i^{th} object into knapsack)
17: $w \leftarrow w + w_i$
18: End If
19: End For

3.2 . QUADRATIC KNAPSACK PROBLEMS

QKP is a quadratic counterpart of the canonical or 0-1 knapsack problem. It is reported as one of the most challenging combinatorial optimization problems in the NP-hard class [22], [23]. In a QKP, each pair of items: i and j has an associated non-negative integer value v_{ij}, added to the knapsack profit when both the items are selected in the solution. These value pairs form a $n \times n$ non-negative integer matrix $Q=(v_{ij})$. Diagonal values v_{ii} in this matrix represent the selection values of the i^{th} item, while other values v_{ij} represent additional profit when both items i and j are selected. The weight and capacity constraints are unchanged from 0-1 KP.

$$\max \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij}x_i x_j \tag{11}$$

$$\text{subject to:} \sum_{i=1}^{n} w_i x_i \le C, \tag{12}$$

$$x_i, x_j \in \{0,1\}, \; i=1,2,\ldots,n \tag{13}$$

QKP reverts to KP when all the quadratic values v_{ij} (i.e., profit, when item $i \neq j$) become zero. It has been applied in the fields of compiler design [24], VLSI design [25], clique problem [26], [27] and others.

Julstrom [28] provided a greedy Genetic Algorithm (GGA) with the use of three popular heuristics for QKP – Absolute Value Density (AVD), Relative Value Density (RVD), and Dual Heuristic. The Absolute Value Density (AVD) heuristic of an object i is given as the ratio of the total linear as well as quadratic values associated with that object to its weight, see Eq.(14). However, the Relative Value Density (RVD) heuristic utilizes only objects already in the knapsack. The RVD value of any excluded object i concerning a (partially filled) knapsack K is calculated as in Eq.(15).

$$AVD_i = \frac{v_i + \sum_{j \neq i} v_{ij}}{w_i} \tag{14}$$

$$RVD_i = \frac{v_i + \sum_{j \in K} v_{ij}}{w_i} \tag{15}$$

The pseudo-code given in Algorithm.4 is used to generate an RVD heuristic-based greedy order. This greedy order is utilized to repair an overfilled or partially filled knapsack. Pseudo-code given in Algorithm.5 presents this greedy repairing process.

Algorithm.4. Pseudo-code for RVD-based *GreedyOrder*

1: $PS \leftarrow \emptyset, j \leftarrow 1$
2: Let $i \leftarrow \max(v_{ii}/w_i)$ (i.e., item with best diagonal value-weight ratio)
3: $GO[j] \leftarrow$ i
4: $PS \leftarrow PS \cup i$
5: For $j \in [2,n]$ Do
6: let $i \leftarrow \max_{i \notin PS}\left(RVD_i^{GO}\right)$
7: $GO[j] \leftarrow i$
8: $PS \leftarrow PS \cup i$
9: End For
10: Return (GO)

Algorithm 5: Pseudo-code for *RepairSolutionGreedyOrder(X)*

1: Knapsack_overfilled ← false
2: $w \leftarrow \sum_{i=1}^{n} w_i x_i$
3: If w>capacity then
4: Knapsack_overfilled ← true
5: End If
6: While Knapsack_overfilled Do
7: For i from $GO[n]$ to $GO[1]$ Do
8: If $x_i = 1$ then
9: $x_i \leftarrow 0$
10: $w \leftarrow w - w_i$
11: End If
12: End For
13: If $w \le$capacity, then
14: Knapsack_overfilled ← false
15: End If
16: End While
17: For i from $GO[1]$ to $GO[0]$ Do
18: If $x_i=0$ and $w + w_i \le$capacity, then
19: $x_i \leftarrow 1$
20: $w \leftarrow w+w_i$
21: End If
22: End For

The dual heuristic utilizes both AVD and RVD heuristics. It starts with a filled knapsack. Objects from this knapsack are iteratively removed with a greedy approach if the knapsack capacity is violated. Pseudo-code for generating such a greedy solution is presented in Algorithm.6. The solution thus generated is further improved using local improvement (see Algorithm.7).

Algorithm 6: Pseudo-code for Dual heuristic-based Greedy*Solution*

1: $GS \leftarrow \{1,2,...,n\}$
2: $w \leftarrow \sum_{i=1}^{n} w_i$
3: While true Do
4: let $i | \min_{i \in GS}\left(RVD_i^{GS}\right)$
5: If w>capacity then
6: $GS \leftarrow GS/i$
7: $w \leftarrow w-w_i$
8: Else
9: ImproveLocal(GS)

10: return (*GS*)
11: End If
12: End While

Algorithm.7. Pseudo-code for *ImproveLocal(X)*

1: While true Do
2: $bg \leftarrow 0$
3: $m_i = -1$
4: $m_j = -1$
5: For $\forall i \notin X$ Do
6: If $w_i \le \text{capacity} - \sum_{k \in X} w_k$ then
7: $g = v_{ii} + \sum_{k \in X} v_i v_k$
8: If $g > b_g$ then
9: $b_g \leftarrow g$
10: $m_i = i$
11: End If
12: Else
13: For $\forall j \in X$ Do
14: $g = p_{ii} - p_{jj} + \sum_{k \in X/j} \left(v_i v_k - v_j v_k \right)$
15: If $g > b_g$ then
16: $b_g \leftarrow g$
17: $m_i = i$
18: $m_j = j$
19: End If
20: End For
21: End If
22: End For
23: If $b_g = 0$ then
24: exit
25: End If
26: $X \leftarrow X \cup m_i$
27: If $m_j \neq -1$ then
28: $X \leftarrow X / m_j$
29: End If
30: End While

For QKP instances, in this study, a benchmark dataset of 100 instances with 100, 200, and 300 objects is taken from [29]. These instances are classified into 4 groups according to their quadratic value density distributions: 0.25, 0.5, 0.75 and 1.0. This quadratic value density of objects represents the proportion of object pairs that provide additional benefit when selected together. A lower-density instance has more independent objects in quadratic terms.

3.3 QUADRATIC MULTIPLE KNAPSACK PROBLEMS

A multiple knapsack problem (MKP) places objects into (K, where $K \ge 2$) knapsacks. The capacity of such multiple knapsacks could be identical. The solution to such a problem is to search for the best way to put the objects without repetition into different available knapsacks to maximize their total value without exceeding corresponding capacity constraints. The mixture of multiple and quadratic knapsack concepts yields the quadratic multiple knapsack problem (QMKP).

The total value of a particular knapsack (k) in QMKP is calculated by totalling the values (v_i) of the included objects ($\forall i \in k$) along with their quadratic values (v_{ij}: $i \neq j, \forall j \in k$). QMKP maximizes the total value of all the knapsacks at once. A greedy heuristic for such a problem can be prepared from already defined AVD, RVD, or dual heuristic concepts.

For QMKP, instances from the QKP benchmark dataset are taken. Ten 25% density QKP instances are the first five instances of 100 and 200 objects. These instances are solved for three knapsack configurations, 3, 5, and 10, making 30 QKP instances with 25% density. Similarly, ten instances with 75% density are chosen; this study considers 60 QMKP instances. For each of these instances, capacity constraints are recalculated. An 80% of the total objects' weight is used as the complete capacity, divided by the number of knapsacks, to calculate each knapsack's capacity. Hence, each knapsack's capacity constraints are identical and calculated as in Eq.(16) for these instances.

$$C_k = \frac{0.8 \times \sum_{i=1}^{n} w_i}{K}, \quad \forall k \in K \tag{16}$$

4. COMPUTATIONAL EXPERIMENTS

In a metaheuristic algorithm, exploration and exploitation are the fundamental operations to search for the optimum solution. A proper balance in the computational effort devoted to these two operations is critical to the success of a metaheuristic algorithm. Exploration is used to explore the feasible solution space to find better solutions. Exploitation searches for the solution space around the solution in hand to find more promising solutions nearby. These exploration and exploitation activities are controlled using various algorithmic parameters. The performance of a population-based metaheuristic algorithm varies with the change in its underlying parameter values. In conformance with the general principle in population-based search and optimization algorithms, the initial phase has more exploration, and the later phase is devoted to exploitation. Hence, proper parameter tuning can provide better results.

The initial number of population individuals to be generated is one of the basic parameters in any metaheuristic algorithm. A larger population size provides a high exploration rate. However, it increases the computational time, while a smaller population size utilized to decrease the computational effort can show slower convergence speed or premature convergence due to less exploration. Therefore, a balanced number of individuals in the initial population is essential.

During population initialization, ICA has an extra parameter in addition to population size (N_{pop}), the number of Imperialists (N_{imp}). It is typically considered 10-13% of the population [7]. The remaining Countries (N_{pop}-N_{imp}) become Colonies; hence, no separate parameter is required for the number of Colonies. A population size of 100 Countries and 10 Imperialists is used in all the experiments reported in this paper.

Metaheuristic algorithms with too many parameters require extra experimental work to tune the best parameters for a given problem. One of the disadvantages of ICA is its increased number of parameters. In ICAwB, one ICA parameter: assimilation rate (β) is omitted. In the case of binary encoding, it is impossible to move Colonies toward their Imperialist with certain bits more than their difference. Thus, the new assimilation process devised in this paper does not require an assimilation rate (β) parameter.

Revolution introduces random changes in the solution individuals. It prevents ICAwB from falling in local optima and is implemented using a mutation operator. A low revolution rate (γ) is enough to provide good exploitation. However, a more significant revolution rate provides more exploration than exploitation [30]. The revolution (or mutation) rate (γ) is 0.05 for good exploitation in our implementations carried out by empirical experimentations.

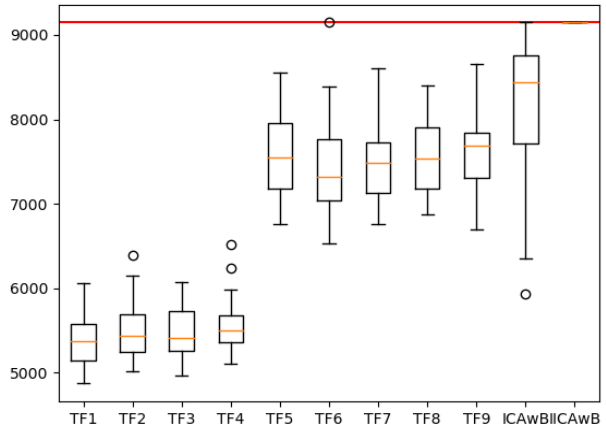

Fig.3(a). Comparison of solution quality for a DKP instance

In ICA, while calculating the overall power of an Empire ($P.E_n$), Imperialists and a proportion (ξ) of Colonies' mean power are aggregated. The power of an Empire represents its possessiveness over other weaker Colonies. The value of ξ determines the Colonies' role while calculating an Empire's power. A high value of ξ may lead to the possession of most of the weaker Colonies to a particular Empire having the highest number of Colonies. However, a small value may emphasize the possessiveness of an Empire depending only upon its Imperialist giving less preference to its possessed Colonies. ξ=0.3 is taken in this study based on computational experiments.

5. RESULTS AND DISCUSSIONS

All the algorithms are implemented in C++ using JetBrains Clion. The experiments were done on Intel® Core™ i7-7700 CPU (3.60GHz) with 8M cache and 8GB RAM. The machine uses Windows 10 as the operating system.

First, the computational experiments have been performed on the 21 DKP instances mentioned for the data range R=1000 and the number of items ranging from 100 to 10000. For BICA, optimal parameters - assimilation rate (β)=1.5, revolution rate (γ)=0.8 and value of ξ=0.77 as suggested in [12] are utilized. ICAwB and IICAwB are demonstrated on these DKP instances with parameter values γ=0.05 and ξ=0.3. These parameter values are summarized in Table.3. In all experiments, the number of initial populations and Imperialists are taken as 100 and 10, respectively, and the number of maximum decades/iterations is taken as 1000. The Table.4 reports computational results of BICA on these benchmark DKP instances. BS and WS are the best and worst solutions found in a run, and MBS is the average of best solutions in 30 runs.

Table.3. Parameter values used in BICA, ICAwB and IICAwB

Algorithm	Parameter Settings
BICA	$\beta = 1.5$, $\gamma = 0.8$, $\xi = 0.77$
Proposed ICAwB	$\gamma = 0.05$, $\xi = 0.3$
Proposed IICAwB	$\gamma = 0.05$, $\xi = 0.3$

It is clear from Table.4 that BICA with TF9 outperforms other versions of BICA in most instances. A comparison between the best results demonstrated by any version of BICA, ICAwB, and IICAwB on DKP instances is given in Table.5. ICAwB performed better than the best BICA in 18 out of 21 instances. IICAwB performed even better and provided optimal solutions for 16 out of 21 instances. In the remaining 5 instances, IICAwB is near the optimal. The Fig.3(a) illustrates the comparison of solution quality for a DKP instance; the upper horizontal line represents the optimal solution.

In addition, the performance of these algorithms is demonstrated and compared on QKP instances of 100, 200 and 300 objects. The Table.6, Table.7 and Table.8 provide the results of these algorithms on different object sizes. It is clear from Table.6 that ICAwB provided near-optimal solutions and performed better than BICA in all instances. IICAwB provided near-optimal solutions with a maximum gap of 0.01%.

The Table.7 shows the results of 40 QKP instances of size 200. Again, for object size 200, ICAwB performed better than BICA for all the instances. However, IICAwB provided the best results for most instances, with a maximum gap of 0.34%.

20 QKP instances of 300 objects are solved with these algorithms in Table.8. In these instances, IICAwB provided the best results with a maximum gap of 0.5% from the optimal. A graphical comparison of these algorithms on a QKP instance is presented in Fig.3b.

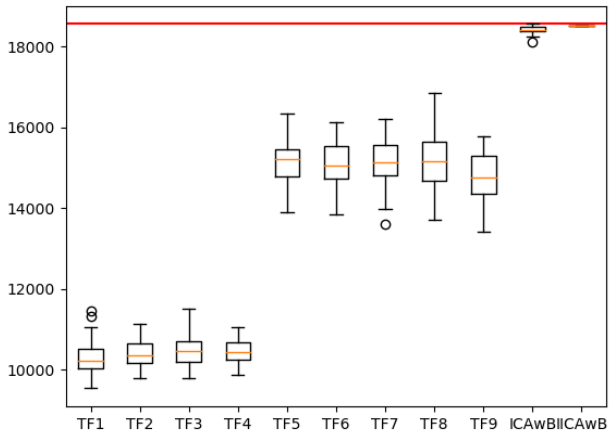

Fig.3(b). Comparison of solution quality for a QKP instance

The Table.9 and Table.10 provide the results of the best BICA, ICAwB, and IICAwB on QMKP instances with three knapsack configurations. The results provided by ICAwB and IICAwB

easily outperform the best BICA. Our IICAwB provides the best solutions to the maximum gap of 0.22% on 25% density instances and 0.1% on 75% density instances.

Further, to clarify the significant differences between the algorithms, two statistical non-parametric tests (Wilcoxon signed rank test and Friedman test) are performed and summarized in Table.11, Table.12, and Table.13. The Table.11 shows how ICAwB and IICAwB outperformed BICA on all the problems considered in this article. The Table.12 compares ICAwB and IICAwB using the Wilcoxon signed rank test. ICAwB performs comparably to IICAwB on QKP with 100 objects, while on other problems, IICAwB easily overperforms ICAwB.

Table.11. Performance comparison summary of BICA with ICAwB and IICAwB using Wilcoxon signed rank test

Problem	BICA vs	R^+	R^-	*p*	+	≈	-	Dec.
DKP	ICAwB	9	222	0.00	3	0	18	-
	IICAwB	0	231	0.00	0	0	21	-
QKP (100)	ICAwB	0	820	0.00	0	0	40	-
	IICAwB	0	820	0.00	0	0	40	-
QKP (200)	ICAwB	0	820	0.00	0	0	40	-
	IICAwB	0	820	0.00	0	0	40	-
QKP (300)	ICAwB	0	210	0.00	0	0	20	-
	IICAwB	0	210	0.00	0	0	20	-
QMKP (25%)	ICAwB	0	465	0.00	0	0	30	-
	IICAwB	0	465	0.00	0	0	30	-
QMKP (75%)	ICAwB	0	465	0.00	0	0	30	-
	IICAwB	0	465	0.00	0	0	30	-

Table.12. Comparing ICAwB with IICAwB using Wilcoxon signed rank test

Problem	R^+	R^-	*p*	+	≈	-	Dec.
DKP	0	230	0.00	0	1	20	-
QKP (100)	353	297	0.74	12	18	10	≈
QKP (200)	45	774	0.00	5	1	34	-
QKP (300)	0	210	0.00	0	0	20	-
QMKP (25%)	7	458	0.00	2	0	28	-
QMKP (75%)	0	465	0.00	0	0	30	-

Additionally, Table.13 shows the algorithms' ranks on different problems calculated using the Friedman test. Again, only on QKP with 100 objects IICAwB ranked second; otherwise, it was first. BICA always performed the last, while ICAwB ranked second on average.

Table.13. Ranking of all algorithms on each problem by Friedman test

Problem	BICA	ICAwB	IICAwB
DKP	2.8571	2.1190	1.0238
QKP (100)	3.0000	1.4750	1.5250
QKP (200)	3.0000	1.8625	1.1375
QKP (300)	3.0000	2.0000	1.0000
QMKP (25%)	3.0000	1.9333	1.0667
QMKP (75%)	3.0000	2.0000	1.0000

6. CONCLUSIONS AND FUTURE WORKS

This paper proposes a binary version of the Imperialist Competitive algorithm dubbed ICAwB. A novel binary assimilation process is devised for this purpose. Furthermore, an improved version of our proposed ICAwB, dubbed IICAwB, is also proposed. For better exploration, IICAwB also utilizes assimilation between Imperialists. The proposed algorithms are first demonstrated on 21 DKP benchmark problem instances, and then the results obtained are compared with a discrete version of ICA, namely the Binary Imperialist Competitive Algorithm (BICA). Results show that our algorithm works better than BICA for all DKP instances.

Furthermore, to establish the superiority of our proposed algorithm, BICA, ICAwB, and IICAwB are demonstrated and compared on benchmark QKP and QMKP instances. ICAwB outperformed BICA in finding the best and average best solutions. However, IICAwB provided the best results for most of the instances.

A recent development in the ICA field is mainly devoted to combining ICA with well-known state-of-the-art algorithms. In future works, ICAwB can also be hybridized with these state-of-the-art algorithms. Parallelization of these algorithms for faster computational results for large-scale problems can also be attempted with advantage.

REFERENCES

[1] Ingo Rechenberg, "Optimierung Technischer Systeme Nach Prinzipien Der biologischen Evolution", Available at https://gwern.net/doc/reinforcement-learning/exploration/1973-rechenberg.pdf, Accessed in 1970.

[2] H. John Holland, "*Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*", MIT Press, 1992.

[3] J. Kennedy and R. Eberhart, "Particle Swarm Optimization", *Proceedings of International Conference on Neural Networks*, Vol. 4, pp. 1942-1948, 1995.

[4] Marco Dorigo and Gianni Di Caro, "Ant Colony Optimization: A New Meta-Heuristic", *Proceedings of International Conference on Evolutionary Computation*, Vol. 2, pp. 1470-1477, 1999.

[5] E. Atashpaz-Gargari and C. Lucas, "Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition", *Proceedings of International Conference on Evolutionary Computation*, pp. 4661-4667, 2007.

[6] Hans-Paul Paul Schwefel, "*Evolution and Optimum Seeking: The Sixth Generation*", John Wiley and Sons, 1993.

[7] Seyedmohsen Hosseini and Abdullah Al Khaled, "A Survey on the Imperialist Competitive Algorithm Metaheuristic: Implementation in Engineering Domain and Directions for

Future Research", *Applied Soft Computing*, Vol. 24, pp. 1078-1094, 2014.
[8] Hans Kellerer, Ulrich Pferschy and David Pisinger, "*Knapsack Problems*", Springer Berlin, 2004.
[9] Shirin Nozarian, Hodais Soltanpoora and Majid Vafaei Jahanb, "A Binary Model on the Basis of Imperialist Competitive Algorithm in Order to Solve the Problem of Knapsack 1-0", *Proceedings of International Conference on System Engineering and Modeling*, pp. 130-135, 2012.
[10] Hojjat Emami and Shahriar Lotfi, "Graph Colouring Problem based on Discrete Imperialist Competitive Algorithm", *International Journal in Foundations of Computer Science and Technology*, Vol. 3, No. 4, pp. 1-12, 2013.
[11] S.J. Mousavirad and H. Ebrahimpour-Komleh, "Feature Selection using Modified Imperialist Competitive Algorithm", *Proceedings of International Conference on Computer and Knowledge Engineering*, pp. 400-405, 2013.
[12] Mina Mirhosseini and Hossein Nezamabadi-Pour, "BICA: A Binary Imperialist Competitive Algorithm and its Application in CBIR Systems", *International Journal of Machine Learning and Cybernetics*, Vol. 9, No. 12, pp. 2043-2057, 2018.
[13] E. David Goldberg, "*Genetic Algorithms in Search, Optimization and Machine Learning*", Addison-Wesley Longman Publishing House, 1989.
[14] Chellapilla Patvardhan, Sulabh Bansal and Anand Srivastav, "Towards the Right Amount of Randomness in Quantum-Inspired Evolutionary Algorithms", *Soft Computing*, Vol. 21, No. 7, pp. 1765-1784, 2017.
[15] H. David Wolpert and G. William Macready, "No Free Lunch Theorems for Search", *Technical Report*, pp. 1-38, 1995.
[16] H. David Wolpert and G. William Macready, "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 67-82, 1997.
[17] Zhongyu Zhao, Xiyuan Peng, Yu Peng and Enzhe Yu, "An Effective Repair Procedure based on Quantum-Inspired Evolutionary Algorithm for 0/1 Knapsack Problems", *Proceedings of International Conference on Instrumentation, Measurement, Circuits and Systems*, pp. 16-18, 2006.
[18] H. Christos Papadimitriou, "On the Complexity of Integer Programming", *Journal of the ACM*, Vol. 28, No. 4, pp. 765-768, 1981.
[19] Silvano Martello and Paolo Toth, "Upper Bounds and Algorithms for Hard 0-1 Knapsack Problems", *Operations Research*, Vol. 45, No. 5, pp. 768-778, 1997.
[20] David Pisinger, "Where are the Hard Knapsack Problems?", *Computers and Operations Research*, Vol. 32, No. 9, pp. 2271-2284, 2005.
[21] Silvano Martello, David Pisinger and Paolo Toth, "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem", *Management Science*, Vol. 45, No. 3, pp. 414-424, 1999.
[22] Alberto Caprara, David Pisinger and Paolo Toth, "Exact Solution of the Quadratic Knapsack Problem", *Informs Journal on Computing*, Vol. 11, No. 2, pp. 125-137, 1999.
[23] David Pisinger, "The Quadratic Knapsack Problem-A Survey", *Discrete Applied Mathematics*, Vol. 155, No. 5, pp. 623-648, 2007.
[24] C. Helmberg, F. Rendl and R. Weismantel, "Quadratic Knapsack Relaxations using Cutting Planes and Semidefinite Programming", *Integer programming and Combinatorial Optimization*, pp. 175-189, 1996.
[25] E. Carlos Ferreira, Alexander Martin, C. Carvalho de Souza, Robert Weismantel and A. Laurence Wolsey, "Formulations and Valid Inequalities for the Node Capacitated Graph Partitioning Problem", *Mathematical Programming*, Vol. 74, No. 3, pp. 247-266, 1996.
[26] G. Dijkhuizen and U. Faigle, "A Cutting-Plane Approach to the Edge-Weighted Maximal Clique Problem", *European Journal of Operational Research*, Vol. 69, No. 1, pp. 121-130, 1993.
[27] Kyungchul Park, Kyungsik Lee and Sungsoo Park, "An Extended Formulation Approach to the Edge-Weighted Maximal Clique Problem", *European Journal of Operational Research*, Vol. 95, No. 3, pp. 671-682, 1996.
[28] A. Bryant Julstrom, "Greedy, Genetic and Greedy Genetic Algorithms for the Quadratic Knapsack Problem", *Proceedings of the International Conference on Genetic and Evolutionary Computation*, pp. 607-614, 2005.
[29] J.E. Beasley, "OR-Library", Available at http://people.brunel.ac.uk/ ~mastjjb/jeb/info.html, Accessed in 2018.
[30] E. Agoston Eiben and A. Cornelis Schippers, "On Evolutionary Exploration and Exploitation", *Fundamenta Informaticae*, Vol. 35, No. 1, pp. 35-50, 1998.