IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 20, 2019, accepted March 24, 2019, date of publication April 1, 2019, date of current version April 15, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2908489

A Comparative Study of Meta-Heuristic Optimization Algorithms for 0 - 1 Knapsack Problem: Some Initial Results

ABSALOM E. EZUGWU[1], VEROSHA PILLAY[2], DIVYAN HIRASEN[2], KERSHEN SIVANARAIN[2], AND MELVIN GOVENDER[2]

[1]School of Computer Science, University of KwaZulu-Natal, Pietermaritzburg 3201, South Africa
[2]School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Durban 4000, South Africa

Corresponding author: Absalom E. Ezugwu (ezugwua@ukzn.ac.za)

ABSTRACT In this paper, we present some initial results of several meta-heuristic optimization algorithms, namely, genetic algorithms, simulated annealing, branch and bound, dynamic programming, greedy search algorithm, and a hybrid genetic algorithm-simulated annealing for solving the 0-1 knapsack problems. Each algorithm is designed in such a way that it penalizes infeasible solutions and optimizes the feasible solution. The experiments are carried out using both low-dimensional and high-dimensional knapsack problems. The numerical results of the hybrid algorithm are compared with the results achieved by the individual algorithms. The results revealed the superior performances of the branch and bound dynamic programming, and hybrid genetic algorithm with simulated annealing methods over all the compared algorithms. This performance was established by taking into account both the algorithm computational time and the solution quality. In addition, the obtained results also indicated that the hybrid algorithm can be applied as an alternative to solve small- and large-sized 0-1 knapsack problems.

INDEX TERMS Knapsack problem, genetic algorithms, simulated annealing, branch and bound, dynamic programming, greedy search algorithm, hybrid IGA-SA.

I. INTRODUCTION

Various combinatorial optimization problems are intrinsically NP-hard because their deterministic polynomial time algorithms are very unlikely to exist [1]. Heuristic approaches for these NP-hard problems have been the focus of the research community. The 0-1 Knapsack Problem (KP01) is popular and widely studied example of an NP-hard combinatorial optimization problem [1], [2], where we find the optimal solution of the given problem such that it satisfies the given constraint. The KP01s appear in real-world decision-making processes in a wide variety of fields [3]. Some examples of KP01s real-world applications includes capital budgeting allocation problem [4], real estate property maintenance problem [5], cargo loading problem [6], resource allocation problem [7], project selection problem [8], combinatorial auctions [9], available-to-promise problem [10], cutting stock problem [11], investment decision making [12]. In the KP01 problem, set of items are provided, each with a weight and a value. The idea is to determine the number of each item to include in a collection, so that the total weight is less than or equal to a maximum limit and the total value is as large as possible [13]–[15].

In this study, we use a variety of heuristic and metaheuristic algorithms to solve the KP01 problem where one has to maximize the benefit of objects in a knapsack without exceeding it is capacity. The algorithms studied in solving the KP01 problems includes: Greedy Search Algorithm (GSA), Dynamic Programming (DP), Branch and Bound (BB), Genetic Algorithm (GA) and Simulated Annealing (SA). Both SA and GA are nature-inspired algorithms with wide applications in solving real-world optimization problems [16]–[19]. SA is based on thermodynamics principle and GA is based on natural evolution [20], [21]. The major advantages of these nature-inspired algorithms are their broad applicability, flexibility, ease of implementation, and the potential of finding near-optimal solutions [22].

The first appearance of KP01 was in 1957 in two publications. The first was a research conducted by Dantzig [23], the founder of operations research and a developer of linear

The associate editor coordinating the review of this manuscript and approving it for publication was Mahammad Abdul Hannan.

programming. The second research is flawlessly maximized by selecting items by bang-for-buck [24], [25]. Furthermore, in [23], Dantzig proposed a greedy approximation algorithm to solve the unbounded knapsack problem [23], [26]. His version sorts the items in decreasing order of value per unit of weight v_i/w_i. A greedy algorithm is an algorithmic paradigm that follows the solving heuristic of making the locally optimal choice at each stage. The dynamic programming concept can be considered as both mathematical optimization and computer programming methods [27], [28]. The method was developed by Richard Bellman in the 1950s and has found applications in numerous field [29]. The Branch and Bound method was first proposed by Land and Doig [30] whilst carrying out research for discrete programming, and has become the most commonly used tool for solving NP-hard optimization problems such as the KP01 and travelling salesman problem [30]. GA is an algorithm that search for good solutions to a problem from among a number of possible solutions. The GA and its variants were pioneered and developed in the 1960s by John Holland, his students, and colleagues [31]. SA is a stochastic based general search tool that mimics the natural process of metals annealing [32]. The SA algorithm has the ability of escaping from local minima and has been widely applied in different domain [33]. The performance of the SA is dependent on the cooling schedule.

The main focus and contribution of this paper is to evaluate the capabilities of the selected meta-heuristics algorithms to solve the KP01 problem. More specifically, to investigate the effectiveness and efficiency of those well-known optimization algorithms namely, GA, SA, BB, DP, GSA and a Hybrid GA and SA (IGA-SA) that have notable track records in finding good quality solutions for the KP01 problems. Furthermore, to also carry out comparisons on the various levels of difficulties for the Knapsack datasets and to determine how well each of the aforementioned algorithms perform on different dimension of the KP01.

The remainder of the paper is organized as follows. In Section 2, the KP01 problem description and related work are discussed. In Section 3, the proposed heuristics and meta-heuristics algorithms are introduced in detail. In Section 4, the performance comparisons of the proposed GA, SA, BB, DP, GSA, and IGA-SA on different types of large-scale KP01 instances are conducted. Finally, conclusions are drawn in Section 5.

II. 0-1 KNAPSACK PROBLEM DESCRIPTION

The KP01 is an example of a combinatorial optimization problem. It searches for the best solution among many other solutions. The objective of this problem is to maximize the total value of items in the knapsack while the constraint ensures the sum of the weights is less than or equal to the knapsack capacity. There is only one item of each type and only two options for each item, that is included in the knapsack or not. Each item cannot be put into the knapsack more than once or be partially included in the knapsack. The KP01 restricts the number x_i copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n each with a weight w_i and a value v_i, along with a maximum weight capacity W. The KP01 can be formulated as follows (See Equation 1 and 2) [3]:

$$\begin{cases} maximize \sum_{i}^{n} v_i x_i \; x_i = \begin{cases} 1 & if \ item \ is \ in \ the \ bag \\ 0 & otherwise \end{cases} & (1) \\ subject \ to \sum_{i=1}^{n} v_i x_i \leq \ W \ and \ x_i \epsilon \{0, 1\} & (2) \end{cases}$$

Here, x_i represents the number of instances of item i to be included in the knapsack.

TABLE 1. Sample KP01 problem.

Item	Benefit	Volume
A	2	9
B	5	6
C	4	7

A. EXAMPLE OF 0-1 KNAPSACK PROBLEM

Suppose we have a knapsack that has a capacity of fifteen cubic inches and several items of different sizes and different benefits [34]. We want to include in the knapsack only these items that will have the greatest total benefit within the constraint of the knapsack's capacity. There are three potential items: A, B, and C. Their volumes and benefits are as shown in Table 1.

We want to maximize the total benefit:

$$\sum_{i=1}^{3} v_i x_i = 2x_1 + 5x_2 + 4x_3 \quad (3)$$

Subject to the following constraints:

$$\sum_{i=1}^{3} v_i x_i = 9x_1 + 6x_2 + 7x_3 \leq 15 \quad and \ x_i \epsilon \{0, 1\}, \quad for \ i = 1, 2..n \quad (4)$$

To find the best solution, we have to identify a subset that meets the constraint and has the maximum total benefit. For this problem, there are 2^3 possible subsets of items as shown in Table 2.

The highlighted row satisfies the constraint. Hence, the optimal benefit for the given constraint ($V = 15$) can only be obtained with one quantity of A, one quantity of B, and zero quantity of C, and it has a resulting benefit of 7.

B. RELATED WORK

Several algorithms have been proposed in the literature for solving the KP01 problem. Most of the algorithms deals with exact approaches. In this section, we present various approaches that have been used to solve the KP problem. Before further discussing each of the methods, we categorize the approaches into three groups i.e.: Exact, Metaheuristic and Hybrid Algorithms.

TABLE 2. Possible subsets of solution for KP01 Problem.

A	B	C	Benefit of Set	Volume of Set
0	0	0	0	0
0	0	1	4	7
0	1	0	5	6
1	0	0	2	9
0	1	1	9	13
1	0	1	-	16
1	**1**	**0**	**7**	**15**
1	1	1	-	22

1) EXACT ALGORITHMS

The BB algorithm was first proposed in [35] and is a general exact technique. The BB algorithm is essentially an enumeration strategy that prunes the non-promising regions of a search space [36]. One of the first BB algorithms proposed for the KP01 problem was presented in [37] and later on ameliorated version in [49], [50]. Bettinelli *et al.* [51] presented a BB based algorithm for Knapsack Problem with Conflict Graph (KPCG) which is an extension of the classic KP01 problem. The study employed the BB algorithm to obtain optimal solutions to the KPCG in short computational time. The reduction of a solution space and enumeration of a smaller number of nodes in BB based algorithms have been onerous in finding solutions for various knapsack problems. However, Tari [52] presented an algorithmic procedure based on the BB with three different selective branching mechanisms for the reduction of the solution space to derive an optimal solution of the Multi-dimensional Multi-choice Knapsack problem.

Another widely utilized exact approach is DP. The DP algorithm was introduced by Richard Bellman [27], [53] who first used the term DP in 1957. The proposed DP heuristic has an improved algorithmic complexity of (nW), where W is the total capacity of the knapsack problem. Later on, two new algorithms were presented in [41] that proved to outperform all previous exact methods that were implemented for the KP01 problem. Detailed summaries of the DP and BB approaches can be found in [25].

A review of the GSA, BB and DP algorithms are conducted by [54]. In their work, the authors compared and contrasted each of the three algorithms, concluding that the most efficient technique is the GSA. However, it is inappropriate under certain conditions as it does not result in the optimal solution. Furthermore, they noted that the DP algorithm proved to be immensely efficient in terms of the number of computations for KP01 problems with lesser capacities. However, as the capacity of the knapsack increases, the DP proved to be inefficient.

2) META-HEURISTIC ALGORITHMS

A variety of meta-heuristic algorithms with wide range of applications to several real-world problems exist in the literature [64]–[72], this paper mainly discussed the SA and the GA because both algorithms are fundamentals to other recently proposed meta-heuristic algorithms. Most importantly, the SA and GA have an excellent record of finding good quality solutions for the KP01 problems. The GA is a commonly used algorithm for solving the KP01 problem. The characteristics of the GA for the KP01 problem involves the encoding of solutions as an array of bits or character strings i.e. chromosomes, the manipulation of these strings by genetic operators and a selection based on their fitness to find the optimal solution to a given problem. Stripling *et al.* [55] reviewed how a simple GA can be applied to solve the knapsack problem. They outlined the similarities to the feature selection problem that frequently occur in the context of an analytical model. Yadav and Singh [57] used the GA to solve the KP01 problem by utilizing corrupted renewal and focal improvement operators which they apply to every recent generated solution. Furthermore, Yadav and Singh [56] conducted a review on the applications of GA for solving the KP01 problem.

The SA algorithm on the other hand was developed by Sonuc *et al.* [58] for solving economic activities problems in 1983. The purpose of the SA algorithm is to find a global maximum or minimum point of a function that has more than one local maximum or minimum point. Kirkpatrick and Vecchi [32] proposed an effective SA algorithm for the KP01 problem which runs on CPU and GPU. Their method runs in parallel on a GPU platform with multi-start technique to enhance the quality of solutions. They deduced that their method was capable of delivering good quality solution for both the low-dimensional and medium-dimensional instances within a short period of time.

3) HYBRID ALGORITHMS

Manaseer and Almogdady [59] hybridized DP and GSA to solve KP01 problem. It was found that the hybrid DP and GSA is better than the constituent GSA and other compared algorithms. However, the hybrid algorithm is found to be inferior in terms of time and space complexity, it achieved a time complexity of $O(n \log(n))$. Lin *et al.* [60] proposed a new stochastic approach for solving the well-known combinatorial optimization problems. The study integrates the GA into the SA algorithm to boost the performance of the SA. Lin *et al.* highlighted some of the major drawbacks of the SA and analysed the characteristics of the GA, noting that one of the various challenges faced by SA schedules is the concept of the quasi-equilibrium [61], [62]. To combat the previously proposed algorithms falling into local optimal solutions for solving the KP01 problem, Zhou *et al.* [63] proposed a binary version of the monkey algorithm where the GSA is employed to reinforce the local search ability. The algorithm used the GSA to correct the infeasible solutions (solutions whose sum of weights exceed the specified capacity). They concluded that the proposed algorithm has strong advantages in solving KP01 problems for testing fixed and random problems, small and large-scale problems. Rezoug *et al.* [19] presented

TABLE 3. Choice of algorithm selection for 0-1 Knapsack problem.

Algorithms	Choice of techniques based on implementation and performance advantages
GSA	• It is quite easy to come up with a greedy algorithm (or even multiple greedy algorithms) for a problem. • It requires less computing resources. • It is faster to execute.
DP	• It is well suited for a multi-stage, multi-point and sequential decision process. • It is suitable for linear or non-linear problems, discrete or continuous variables, and deterministic problems.
BB	• It can find an optimal solution (provided the problem is of limited size and enumeration can be computed in reasonable time). • Generally, it will inspect less sub-problems and thus save computational time.
GA	• GA's can work well on mixed discrete/continuous problems. • Support multi-objective optimization problems. • GA's search from a population of points, not a single point. • Makes use of an objective function and not derivatives.
SA	• It can deal with highly non-linear models, chaotic and noisy data. • It is flexible and is able to escape from a local optima.

a hybrid heuristic algorithm which is referred to as the Guided GA (GGA) for solving the Multidimensional Knapsack Problem (MKP). It is a two-step memetic algorithm composed of a data pre-analysis and a modified GA. They deduced that the GGA outperformed the basic GA on several instances. Similarly, the result showed that there is an improved performance achieved by the GA and accelerated speed of the convergence when guidance is provided to the algorithm.

C. CHOICE OF TECHNIQUES

After conducting extensive research (See Section II-B). Furthermore, the reasons for our choice of the representative algorithms are as highlighted in Table 3. It is noteworthy to emphasis here that the main focus of the current study is to present an initial result based on the results obtained from the comparative analyses of all the aforementioned algorithms: GSA, DP, BB, SA and GA. More so, we have also tried to establish some performance profile for the five representative algorithms which are supported by the detailed literature review.

III. HEURISTICS AND METAHEURISTICS ALGORITHMS FOR SOLVING 0-1 KNAPSACK PROBLEM

In this section, we discuss in detail the five heuristics algorithms chosen to solve the 0-1 KP and subsequently provide an in-depth comparative analysis of the algorithms with respect to their capability to provide feasible and near optimum solutions to the problem. In addition, the structural design, limitations and advantages for each of the algorithms are provided. Furthermore, algorithmic design and implementation steps are discussed.

A. GREEDY SEARCH ALGORITHM

A GSA is an algorithm that uses a heuristic technique for making locally optimal choices at each stage with the hope of finding a global optimum. Greedy algorithm often fails to find the globally optimum solution because it usually does not operate exhaustively on all the data. This algorithm can make commitments to certain choices too early which prevent it from finding the best overall solution later [38]. Assume that there is an objective function that needs to be optimized (either maximized or minimized) at a given point. A GSA makes greedy choices at each step to ensure that the objective function is optimized. The GSA has only one shot to compute the optimal solution so that it never goes back and reverses the decision. Some of the notable limitations of GSA are as follows:

- The GSA does not always reach the global optimum solution.
- The difficult part is that, a greedy algorithm requires to work harder to understand the correct issues. Even with the correct algorithm, it is hard to prove why it is correct.
- It may produce the unique worst possible solution.

As regards the GSA and with respect to the KP01 problem, there are several other greedy algorithms been proposed to solve the KP01 problem and its other variants as well. The most efficient technique follows the following procedures [38], [39]

- Compute the profit-weight ratio of the given items.
- Sort the array containing the ratio of the items in decreasing order.
- Place the item with the highest ratio into the Knapsack if it does not exceed the capacity of the Knapsack, else proceed to the next item.

Our implementation using the GSA discussed in this paper adopts the following set of steps:

- Sort the items in descending order according to their values.
- If an item can fit in the knapsack, then it is added.
- Termination occurs when no more items can occupy the knapsack.

B. DYNAMIC PROGRAMMING

The DP is used for problems requiring a sequence of interrelated decisions, that is, to take another decision, this would depend on the previous decision or solution. The DP method is a technique for solving problems whose solutions satisfies recurrence relations with overlapping sub-problems [40], [41]. In terms of mathematical optimization, the DP usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. Some of the major limitations of the DP includes:

- No general formation of DP is available; every problem has to be solved in its own way.
- Divide problem into sub-problems and store intermediate results that consumes memory.

The DP with respect to the KP01 problem considers an instance of the problem defined by the first i items, $1 \leq i \leq N$, with:

weights $w_1, w_2 \ldots, w_i$,

values $v_1, v_2, \ldots, v_i$,

and knapsack capacity j, $1 \leq j \leq Capacity$.

Let Table [i, j] be the optimal solution of this instance (that is, the value of the most valuable subsets of the first i items that fit into the knapsack capacity of j). The Table can be populated by the Algorithm 1. The last value of the array Table, that is, Table [N, Capacity] contains the optimal solution of the problem. The implementation details of using DP to solve the KP01 is described as follows:

- Algorithm 1 was used to find the optimal value.
- Since the algorithm 1 only returns the optimal value, we had to employ some form of backtracking. The backtracking technique used is shown in Algorithm 2.

C. BRANCH AND BOUND

The BB algorithm is used to find the optimal solution by keeping the best solution found so far. The BB constructs candidate solutions one component at a time and evaluates the partly constructed solutions. If a partial solution can not improve on the currently optimal solution, it is abandoned. This approach makes it possible to solve some large instances of difficult combinatorial problems. This algorithm is based on the construction of a state space tree. A state space tree is a rooted tree where each level represents a choice in the solution space that depends on the level above and any possible solution is represented by some path starting out at the root and ending at a leaf. A node's bound value is compared with the value of the best current solution obtained. If the bound value is not better than the best current solution, that is, not smaller for minimization and not larger for maximization, the node is non-promising and can be terminated in view of the fact that no solution obtained from it can yield a better solution than the one already available solution. This is the principle idea of the BB algorithm [42], [43]. Next we discuss the branching strategy.

According to Gupta and Ravindran [44], there are generally two ways of implementing the branching:

i. Branching on the node with the smallest bound: Search all the nodes and find the one with the smallest bound and set it as the next branching node.
   Advantage: Generally, it will inspect less sub-problems and thus saves computational time.
   Disadvantage: Normally, it will require more storage.
ii. Branching on the newly created node with the smallest bound: Search the newly created nodes and find the one with the smallest bound and set it as the next branching node.
   Advantage: Saves storage space.
   Disadvantage: Require more branching effort and therefore not computationally efficient.

In the context of BB with respect to solving the KP01 problem, if there are N possible items to choose from, then, the k^{th} level represents the state where it has been decided which of the first k items have or have not been included in the knapsack. In this case, there are 2 k nodes on the k^{th} level and the state space tree's leaves are all on level N.

As regards our implementation using the BB algorithm, we implemented the BB algorithm using a priority queue. The solution is built stepwise. The upper bound (ub) is first calculated. This is computed by adding the total profit of the items that are already selected, p, the product of the remaining capacity of the knapsack, $M - w$, and the best profit-weight ratio, which is $\frac{P_i+1}{W_i+1}$. Hence, the formula used (Equation 5) adopted from [46]:

$$ub = p + (M - w)\left(\frac{P_i + 1}{W_i + 1}\right) \tag{5}$$

i. Using a heuristic, find a solution x_s to the KP01 problem. Next store it's value, $Best = f(x_s)$, where $Best$ denotes the best found current solution.
ii. Initialize a queue to the partial solution with none of the variables.
iii. Loop until the queue is empty:
   a. Take a Node N off the queue
   b. If N represents a single candidate solution x and $f(x) < Best$, then x is the best current solution. Record it and set $Best \leftarrow f(x)$
   c. Else , branch on N to produce new nodes N_i.
   d. For each of these:
      - If $g(N_i) > Best$, do nothing since the lower bound on this node is greater than the upper bound, since it will never lead to an optimal solution and therefore can be disregarded
      - Else, store N_i on the queue

D. GENETIC ALGORITHM

The GA is a heuristic search and optimization algorithm inspired by natural evolution [45]. The GA begin with a set of candidate solutions (chromosomes) called population. A new population is created from solutions of an old population with the hope of getting a better population. Solutions which are chosen to form new solutions (offspring) are selected according to their fitness. The more suitable the solutions are

Algorithm 1 Pseudocode of the Implemented Dynamic Programming Method

```
for i = 0 to N do
    for j = 0 to Capacity
        if j < Weights[i] then
        Table[i, j] ← Table[i − 1, j]
        else
            Table[i, j] ← maximum {Table[i − 1, j]
                                   AND
                          Values[i] + Table[i − 1, j − Weights[i]]}
        end
    end
end
return Table[N, Capacity]
```

Algorithm 2 Pseudocode of the Backtracking Technique

```
n ← N
c ← Capacity
Start at position Table[n, c]
while remaining Capacity > 0 do
    if Table[n, c] = Table[n − 1, c] then
        Item n has not been included in the optimal solution
    else
        Item n has been included in the optimal solution
        Process item n
        Move one row up to n − 1
        Move to column c− weight (n)
    end
end
```

the bigger chances they have to reproduce. This process is repeated until some condition is satisfied. The main components of the GA are the chromosome encoding, the fitness function, selection, recombination and the evolution scheme [34].

1) CHROMOSOME ENCODING

The GA manipulates population of the chromosomes. The chromosomes are string representations of solutions to a particular problem [31]. For the KP, we use binary encoding, where every chromosome is a string of bits, 0 or 1 [34].

2) FITNESS FUNCTION

The fitness function is a computation that evaluates the quality of the chromosome as a solution to a particular problem [31]. The fitness function allocates a score to each chromosome in the population. This will help determine how well a particular solution solves a problem.

3) SELECTION PROCESS

In this process, chromosomes are selected for recombination on the basis of the fitness function. An evaluation of the current population members is done. A selection of the subset with the best fitness values to act as parents for the next generation is carried out. Those chromosomes with a higher fitness score have a significantly higher chance of selection than those with a lower fitness score. Hence, creating a basis for more highly fit solutions.

4) CROSSOVER

Crossover takes chromosome pairs that have been chosen from the selection process. These selected pairs are merged together to generate the new successor population. The idea is to simulate the mixing of genetic material that can occur when organisms reproduce [31]. Consider the following parents and a crossover point at position 3:

Parent 1: 1 0 0 | 0 1 1 1
Parent 2: 1 1 1 | 1 0 0 0
Offspring 1: 1 0 0 1 0 0 0
Offspring 2: 1 1 1 0 1 1 1

These offspring are created by exchanging the genes of parents until the crossover point is reached. Hence, offspring 1 receives the first 3 bits to the left of the parent 1's crossover point and the remaining bits from the right of parent 2's crossover point. Similarly, offspring 2 receives the first 3 bits to the right of the parent 1's crossover point and the remaining bits from the left of parent 2's crossover point.

5) MUTATION

In terms of biology, mutation may allow for the probability of a child to inherit a characteristic (feature) that was not

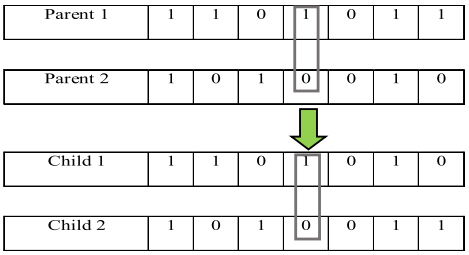

FIGURE 1. Single point crossover.

inherited from the parents. To maintain generic diversity from one generation of the population to the next, the mutation process flips bits from 0 to 1 or from 1 to 0.

6) LIMITATIONS OF THE GENETIC ALGORITHM

- It is really difficult for a researcher to come up with a good heuristic that actually reflects what we want the algorithm to do.
- GA's cannot always find the exact solution but they always find the best solution.
- GA's are computationally expensive, that is, they are time consuming.
- GA's requires less information about the problem but designing an objective function, getting the representation and operators right can be fairly difficult.

7) GENETIC ALGORITHM IMPLEMENTATION

The GA implementation steps centres around good formulation of the algorithm's solution representation, fitness function, genetic operators and termination condition.

Solution representation: A binary representation is used where a value of 1 indicates that an object is placed in the knapsack, whilst a value of 0 indicates that the object is left behind.

Fitness function: The GA fitness function denotes thesum of values represented by the chromosome.

8) GA OPERATORS

i. Single point crossover- in this crossover, a single crossover point on both parent chromosomes is selected by choosing a random number. Both the parent chromosomes are split at the crossover point chosen and all data beyond that point in either chromosome is swapped between the two parent chromosomes. A crossover rate of 0.1 is used. An example is depicted in Figure 1:

ii. *Bit-Flip Mutation*: Select one random bit and flip it. This is used for binary encoded GAs. A mutation rate of 0.3 is used. Figure 2 depicts an example of Bit-flip mutation.

iii. *Roulette wheel selection*: In this technique, all the chromosomes in the population are placed on the roulette wheel according to their fitness value. Each individual is assigned a segment of the roulette wheel whose size is proportional to the value of the fitness of the individual. The bigger the fitness value is, the larger the segment. Then, the virtual roulette wheel is span. The individual corresponding to the segment on which roulette wheel stops are then selected. The process is repeated until the desired number of individuals is selected.

iv. *Termination Criteria*: There are two termination criteria: Firstly, it occurs if 90% of the population fall within convergence of the fittest individual in the population. Secondly, it occurs if the fittest individual reflects no change for 700 generations. Either of these criterion will result in termination. A flowchart of the GA is depicted in Figure 3.

FIGURE 2. Bit-flip mutation.

E. SIMULATED ANNEALING ALGORITHM

The SA is a stochastic computational algorithm for finding global extremums to large optimization problems. The name and inspiration of the algorithm was originally inspired from the process of annealing in metal work. Annealing involves heating and cooling a material to alter its physical properties due to the changes in its internal structure. It is generally used when the search space is discrete. At each iteration of a SA algorithm applied to a discrete optimization problem, the objective function values for two solutions (the current solution and a newly generated neighbouring solution) are compared. Better solutions are always accepted, while a fraction of inferior solutions are accepted in the hope of escaping local optima in search of global optima. The key feature of the SA is that it is able to escape local optima by allowing worse moves (i.e. moves to a solution that corresponds to a worse objective function value) [17]. The basic ingredients for the SA process include solution space, neighbourhood structure, cost function and Annealing schedule.

There have been very few existing implementations of the SA for the KP01 problem. Hence, this project attempts to explore this largely unexplored area. A common practice when using evolutionary algorithms is to disregard solutions that are infeasible (i.e. solutions where the weight exceeds the maximum capacity of the knapsack) and ''drop'' an item from the knapsack until it's feasible. However, due to the nature of SA, our implementation gives the infeasible solutions a chance to improve in future iterations. Empirically, this technique has been evaluated to be more optimum, hence it is used in our implementation. Below are two main limitations of the SA:

- There is a clear trade-off between the quality of the solution and the time required to produce the solution.
- The precision of numbers used can have a significant effect upon the quality of the outcome.

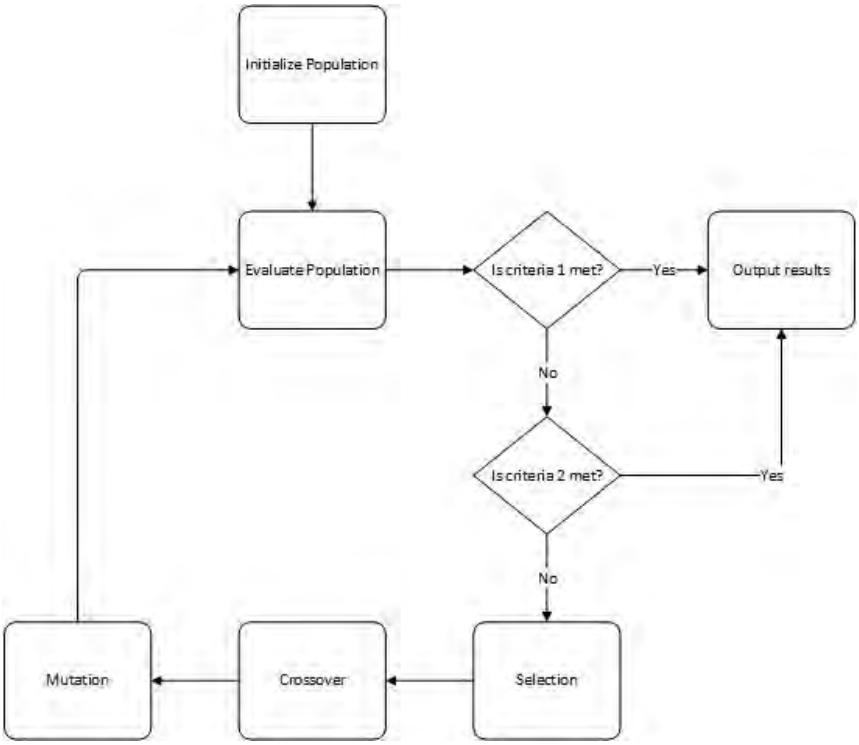

FIGURE 3. Flowchart for the genetic algorithm.

The implementation of the SA is described as follows:

Representation: A binary representation is used, where a value of 1 indicates that an object is placed in the knapsack, whilst a value of 0 indicates that the object is left behind.

Pseudocode: The pseudocode of our implementation is shown in Algorithm 3. It is inspired by the basic concept of the SA which is now extended to the knapsack problem. A finite-length Markov chain is generated at a certain temperature T_k and a temperature control parameter α, that slowly cools the system until a certain temperature at which the system is frozen.

State Generation: The state generation is performed by randomly selecting a bit of the current_state and flipping it to form the next_state. Figure 4 shows the state generation.

Termination Criteria: The algorithm terminates when the temperature reaches the minimum allowed temperature of the system, indicating that the system is frozen. Our implementation uses an initial temperature of 1000 and is slowly decreased to a frozen state of 0.0001. The temperature change is accomplished by multiplying the system temperature by $\alpha = 0.9999$.

F. HYBRID METAHEURISTIC ALGORITHMS FOR SOLVING 0-1 KNAPSACK PROBLEM

In an effort to improve the quality of solutions obtained by the basic meta-heuristic algorithms specifically the GA and SA algorithms, a hybrid approach was adopted. The IGA-SA [47] comprises of SA integrated in GA. The SA is expected to refine the population generated by the GA. This is

FIGURE 4. Simulated Annealing state generation.

accomplished by searching the neighbourhood of a candidate solution for better fit solutions. Majority of the hybrid approaches involving these two algorithms apply the SA algorithm to each individual in the population. However, this is computationally expensive and may ultimately prove to be useless as only half of the current population are selected to reproduce; and the other half is discarded. We adopt the approach in [47] where the SA algorithm is only applied to the fittest individual in each generation (see Figure 4). The GA and SA algorithms used in our hybrid IGA-SA implementation are of the simpler variations of each of the standard GA and SA algorithms, respectively. Therefore, we opted for simpler algorithms to be sure that the solutions generated by the hybrid approach are the direct result of combining the performance capabilities of the two algorithms and not narrowing down to the numerous improvements made on each algorithm by the research community.

The configuration of each algorithm is shown in Table 4, the hybrid algorithm adopts majority of its configuration values from the parameter settings of both the GA and SA shown in Table 4. Though, the SA executes for only 250 iterations in the hybrid implementation whereas it executes for 2000 iterations in the standalone implementation. This is due to the SA algorithm acting as a local search technique in the hybrid

Algorithm 3 Pseudocode of the Implemented Simulated Annealing Algorithm

```
Initialize parameters of the annealing schedule;
Generate an initial state as the current_state;
k = 1;
repeat
    repeat
        Generate the next_state;
        Δ = value of next_state – value of current_state;
        if Δ > 0 and solution is feasible then
            current_state = next_state
            if value of current_state >value of best_state then
                best_state = current_state;
            end
        else
            acceptance_function = exp(−Δ/T_k);
            if acceptance_function > random [0,1) then
                current_state = next_state;
            end
        end
    until system equilibrium at T_k
    T_{k+1} = T_k * α
until system has been frozen
print out the current_state as the final state.
```

implementation and only operating on the best individual in the current population, 2000 iterations essentially means the generation of 2000 neighbour solutions which is deemed too excessive.

The definition of the neighbourhood of a solution is pivotal in the performance of the SA algorithm. Since the SA algorithm acts as a local search technique here, it is beneficial that we generate neighbour solutions in such a way that they are of better fitness than the current solution. A neighbour solution is generated depending on the sum of its weights, if the sum of its weights exceeds the specified capacity of the knapsack then the neighbour solution is defined to be the solution with one item randomly removed. If the sum of its weights is below the specified capacity, then the neighbour solution is defined to be the solution with one item randomly added. In evaluating an individual, the same fitness function is applied to all three algorithms and is designed in such a way that it penalizes infeasible solutions.

GAs have shown to be well suited for high-quality solutions to larger NP problems and currently they are the most efficient methods for finding an approximately optimal solution for optimization problems. They do not involve extensive search techniques and do not try to find the best solution. The GA can reserve excellent individuals for the next generation in the genetic operation process and guarantee the diversity of population. The SA algorithm has strong local search ability and is capable of escaping from local optimal solutions. However, GA is liable to converge prematurely and be trapped in local optimal solutions. In addition, the SA is bottlenecked with high computational time. Therefore, by the combination of the two algorithms, an IGA-SA is selected for hybridization to escape from the limitations of the GA and SA pointed out to form a strong hybrid algorithm.

IV. COMPUTATIONAL EXPERIMENT

The experiments were carried out using a 2.80 GHz intel core i7 processor and 4GB memory. The entire algorithms were coded in Java with Eclipse integrated development environment (or IDE). The parameter settings for the two population based algorithms are presented in Table 4.

A. DATASETS

The datasets used for the 0-1 knapsack problem was obtained from David Pisinger's optimization codes [48]. There are four different categorical datasets. Low dimensional uncorrelated (LD-UC), high dimensional uncorrelated (HD-UC), high dimensional weakly correlated (HD-WC) and high dimensional strongly correlated (HD-SC). The variations within the correlation of instances in these datasets will provide an overall performance overview on each of the implemented algorithms. The details of the datasets together with their optimum values recorded are shown in Table 5, while the respective characteristics of the datasets used are shown in Table 6.

B. EXPERIMENT 1: RESULTS AND DISCUSSION

The five algorithms: BB, DP, GA, GSA and SA were all tested on the four different categorical datasets and a wide range of results were formulated from the experiments carried out. A summary of these results and key observations made are discussed in the next section. However, the discussion

TABLE 4. Genetic Algorithm and Simulated Annealing parameter configuration.

Genetic Algorithm		Simulated Annealing	
Population size	100	Max Iterations	2000
Crossover Rate	0.8	Maximum Temperature	2500
Mutation Rate	0.2	Temperature Change	0.90
Maximum Iterations	500		
Selection Operator	K-Tournament Selection		
K (for tournament selection)	20		
Crossover Operator	One-point crossover		
Mutation Operator	Two-point mutation		

TABLE 5. Datasets used with their optimum recorded values.

Low Dimensional – Uncorrelated Data Instances		High Dimensional – Uncorrelated Data Instances	
File: Dataset 1-10	Optimum	File: Dataset 11 - 17	Optimum
f1_l-d_kp_10_269	295	knapPI_1_100_1000_1	9147
f2_l-d_kp_20_878	1024	knapPI_1_200_1000_1	11238
f3_l-d_kp_4_20	35	knapPI_1_500_1000_1	28857
f4_l-d_kp_4_11	23	knapPI_1_1000_1000_1	54503
f5_l-d_kp_15_375	481,0694	knapPI_1_2000_1000_1	110625
f6_l-d_kp_10_60	52	knapPI_1_5000_1000_1	276457
f7_l-d_kp_7_50	107	knapPI_1_10000_1000_1	563647
f8_l-d_kp_23_10000	9767		
f9_l-d_kp_5_80	130		
f10_l-d_kp_20_879	1025		

High Dimensional – Weakly Correlated Data		High Dimensional – Strongly Correlated Data	
File: Dataset 18 - 24	Optimum	File: Dataset 25 - 31	Optimum
knapPI_2_100_1000_1	1514	knapPI_3_100_1000_1	2397
knapPI_2_200_1000_1	1634	knapPI_3_200_1000_1	2697
knapPI_2_500_1000_1	4566	knapPI_3_500_1000_1	7117
knapPI_2_1000_1000_1	9052	knapPI_3_1000_1000_1	14390
knapPI_2_2000_1000_1	18051	knapPI_3_2000_1000_1	28919
knapPI_2_5000_1000_1	44356	knapPI_3_5000_1000_1	72505
knapPI_2_10000_1000_1	90204	knapPI_3_10000_1000_1	146919

TABLE 6. Dataset characteristics.

Dataset 1-10	Dataset 11 - 17	Dataset 18 - 24	Dataset 25 - 31
Low-dimensional	High Dimensional	High-dimensional	High-dimensional
4 – 20 Items each	100 – 10 000 Items each	100 – 10 000 Items each	100 – 10 000 Items each
Uncorrelated data instances	Uncorrelated data instances	Weakly correlated data instances	Strongly correlated data instances

of results is presented according to the following dataset structure:

- LD-UC datasets
- HD-UC datasets
- HD-WC datasets
- HD-SC datasets
- SA and GA iterations
- Overall comparisons and summary of results

TABLE 7. Results obtained on LD-UC datasets.

Algorithm	Dataset	Number of Items	Best Value	Optimal Value	Shortfall	Time (Milli)
		f1_l-d_kp_10_269				
BB	f1_l-d_kp_10_269	10	280	295	15	16
DP	f1_l-d_kp_10_269	10	295	295	0	2
GA	f1_l-d_kp_10_269	10	295	295	0	133
GSA	f1_l-d_kp_10_269	10	288	295	7	0
SA	f1_l-d_kp_10_269	10	294	295	1	4265
		f2_l-d_kp_20_878				
BB	f2_l-d_kp_20_878	20	972	1024	52	0
DP	f2_l-d_kp_20_878	20	1024	1024	0	6
GA	f2_l-d_kp_20_878	20	1024	1024	0	146
GSA	f2_l-d_kp_20_878	20	1024	1024	0	0
BB	f2_l-d_kp_20_878	20	972	1024	52	0
		f3_l-d_kp_4_20				
BB	f3_l-d_kp_4_20	4	24	35	11	0
DP	f3_l-d_kp_4_20	4	35	35	0	1
GA	f3_l-d_kp_4_20	4	35	35	0	95
GSA	f3_l-d_kp_4_20	4	28	35	7	16
SA	f3_l-d_kp_4_20	4	35	35	0	2626
		f4_l-d_kp_4_11				
BB	f4_l-d_kp_4_11	4	22	23	1	0
DP	f4_l-d_kp_4_11	4	23	23	0	1
GA	f4_l-d_kp_4_11	4	23	23	0	82
GSA	f4_l-d_kp_4_11	4	19	23	4	0
SA	f4_l-d_kp_4_11	4	22	23	1	2563
		f5_l-d_kp_15_375				
BB	f5_l-d_kp_15_375	15	469,16	481,07	11,91	0
DP	f5_l-d_kp_15_375	15	477	481,07	4,07	3
GA	f5_l-d_kp_15_375	15	477	481,07	4,07	113
GSA	f5_l-d_kp_15_375	15	481,07	481,07	0	0
SA	f5_l-d_kp_15_375	15	481,07	481,07	0	4858
		f6_l-d_kp_10_60				
BB	f6_l-d_kp_10_60	10	49	52	3	0
DP	f6_l-d_kp_10_60	10	52	52	0	1
GA	f6_l-d_kp_10_60	10	52	52	0	120
GSA	f6_l-d_kp_10_60	10	43	52	9	15
SA	f6_l-d_kp_10_60	10	52	52	0	4141
		f7_l-d_kp_7_50				
BB	f7_l-d_kp_7_50	7	96	107	11	0
DP	f7_l-d_kp_7_50	7	107	107	0	2
GA	f7_l-d_kp_7_50	7	107	107	0	158
GSA	f7_l-d_kp_7_50	7	100	107	7	0
SA	f7_l-d_kp_7_50	7	102	107	5	3359

1) LOW-DIMENSIONAL UNCORRELATED DATASETS

Table 7 shows all the results obtained on LD-UC datasets. From Table 7 and Figure 5, it can be seen that on the initial low-dimensional datasets, all algorithms performed well and obtained results in the same range, except for the BB algorithm. The GSA had obtained near optimal results on this dataset. This could be a special case such that by selecting the items of highest value the weight requirements would increase accordingly and the results obtained becomes convincing. The GA, SA and DP all performed in the same range. Moreover, their execution time was similar, while DP had run in marginally less time as the rest. Even though the SA had obtained comparable results, it has the highest operational time. This is likely because of the gradual temperature decrease in the SA algorithm, which can take up larger amounts of computational resources.

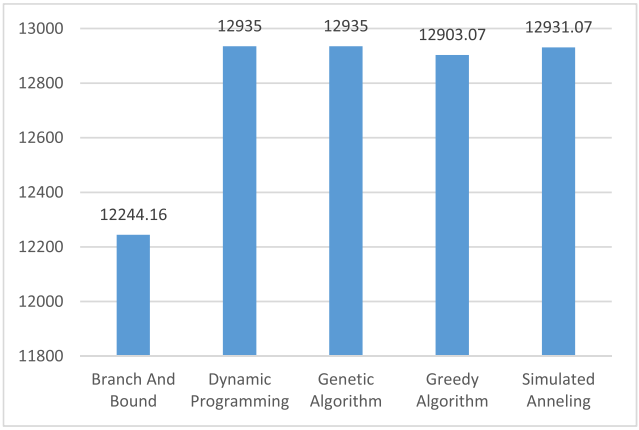

FIGURE 5. A representation of the performances on the LD-UC datasets in terms of the best values produced.

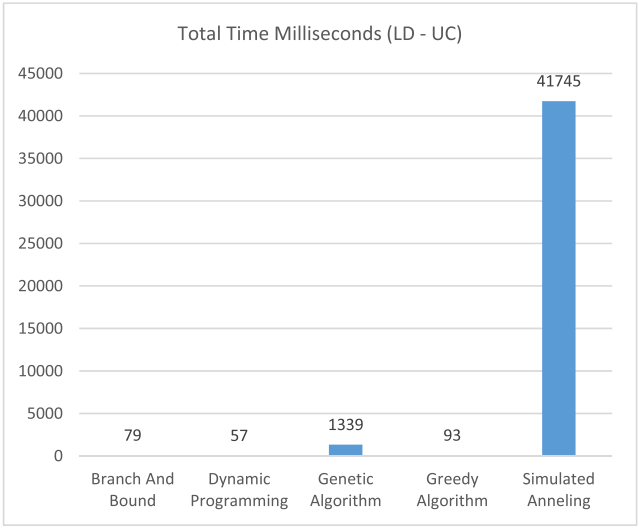

FIGURE 6. A representation of the performances on the LD-UC datasets in terms of the algorithm computational time.

The computational time for the five algorithms are shown in Figure 6.

2) HIGH-DIMENSIONAL UNCORRELATED DATASETS

Table 8 shows all the results obtained based on the HD-UC datasets. Figure 7 and 8 show performances of the algorithms on a HD-UC dataset. It should be noted that the numbers below the bar indicate the 'shortfall' (i.e. how much short the value is from the optimal solution recorded in the literature). For example, it can be seen from Figure 8 that in this instance, DP had a shortfall of zero (i.e. produced the same results as obtained in the literature) whereas BB had a shortfall of 1091 (i.e. compared results from literature had produced a solution with a value of 1091 greater than that produced by the BB).

An important observation is that the DP algorithm was the only algorithm to produce optimal results as compared to those found in the literature. Hence, the DP had achieved perfect outcomes on all datasets within this category. On the other hand, BB also performed consistently well throughout these datasets, followed closely behind DP as shown in Figure 9. Interesting observations can be picked up from the results produced by the GA. The performance seems to

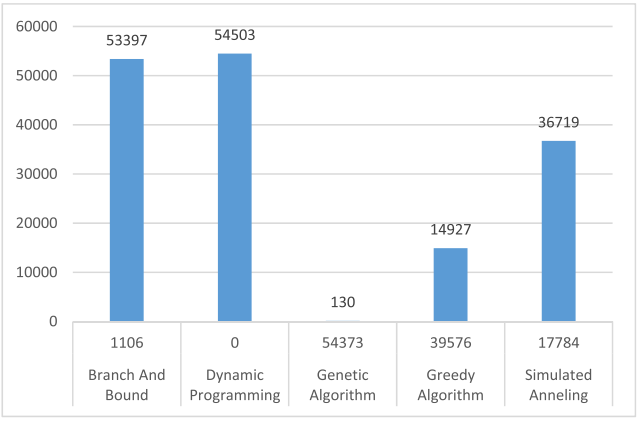

FIGURE 7. Performances of all the algorithms on a HD-UC dataset with 1000 items and a knapsack capacity of 1000.

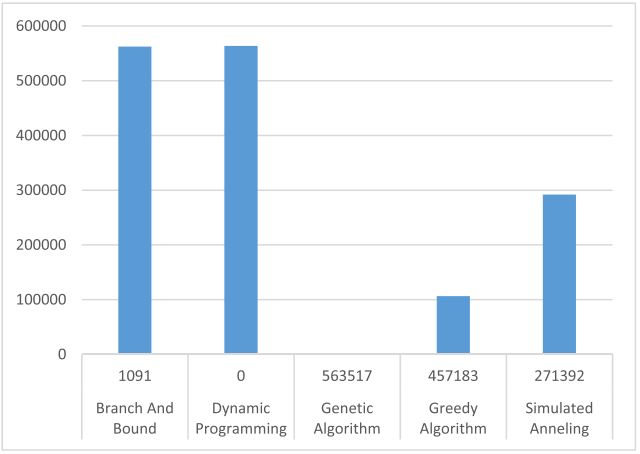

FIGURE 8. Performances of all the algorithms on a HD-UC dataset with 10,000 items and a knapsack capacity of 1000.

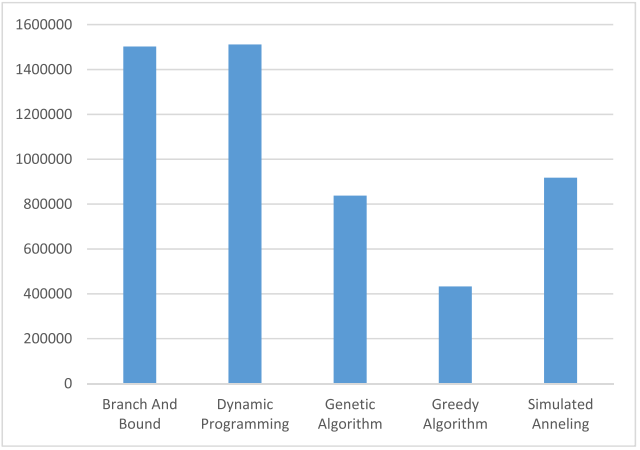

FIGURE 9. A representation of the performances of all the algorithms on HD-UC datasets in terms of the best values produced.

TABLE 8. Results obtained on HD-UC datasets.

Algorithm	Dataset	Number of Items	Best Value	Optimal Value	Shortfall	Time (Milli)
		knapPI_1_100_1000_1				
SA	knapPI_1_100_1000_1	100	9147	9147	0	8592
GA	knapPI_1_100_1000_1	100	9147	9147	0	282
GSA	knapPI_1_100_1000_1	100	2983	9147	6164	0
DP	knapPI_1_100_1000_1	100	9147	9147	0	2
BB	knapPI_1_100_1000_1	100	8026	9147	1121	0
		knapPI_1_200_1000_1				
SA	knapPI_1_200_1000_1	200	10163	11238	1075	11546
GA	knapPI_1_200_1000_1	200	102340	11238	-91102	6
GSA	knapPI_1_200_1000_1	200	4544	11238	6694	15
DP	knapPI_1_200_1000_1	200	11238	11238	0	3
BB	knapPI_1_200_1000_1	200	10436	11238	802	0
		knapPI_1_500_1000_1				
SA	knapPI_1_500_1000_1	500	21390	28857	7467	21620
GA	knapPI_1_500_1000_1	500	102340	28857	-73483	14
GSA	knapPI_1_500_1000_1	500	9865	28857	18992	15
DP	knapPI_1_500_1000_1	500	28857	28857	0	9
BB	knapPI_1_500_1000_1	500	28043	28857	814	15
		knapPI_1_1000_1000_1				
SA	knapPI_1_1000_1000_1	1000	36719	54503	17784	41084
GA	knapPI_1_1000_1000_1	1000	130	54503	54373	27
GSA	knapPI_1_1000_1000_1	1000	14927	54503	39576	0
DP	knapPI_1_1000_1000_1	1000	54503	54503	0	55
BB	knapPI_1_1000_1000_1	1000	53397	54503	1106	0
		knapPI_1_2000_1000_1				
SA	knapPI_1_2000_1000_1	2000	65793	110625	44832	91997
GA	knapPI_1_2000_1000_1	2000	102340	110625	8285	17546
GSA	knapPI_1_2000_1000_1	2000	25579	110625	85046	0
DP	knapPI_1_2000_1000_1	2000	110625	110625	0	108
BB	knapPI_1_2000_1000_1	2000	109679	110625	946	0
		knapPI_1_5000_1000_1				
SA	knapPI_1_5000_1000_1	5000	150731	276457	125726	275563
GA	knapPI_1_5000_1000_1	5000	102340	276457	174117	126
GSA	knapPI_1_5000_1000_1	5000	49306	276457	227151	16
DP	knapPI_1_5000_1000_1	5000	276457	276457	0	896
BB	knapPI_1_5000_1000_1	5000	275720	276457	737	32
		knapPI_1_10000_1000_1				
SA	knapPI_1_10000_1000_1	10000	563647	563647	0	3550
GA	knapPI_1_10000_1000_1	10000	562556	563647	1091	188
GSA	knapPI_1_10000_1000_1	10000	292255	563647	271392	603882
DP	knapPI_1_10000_1000_1	10000	106464	563647	457183	203
BB	knapPI_1_10000_1000_1	10000	130	563647	563517	294

fluctuate, often based on the size of the datasets. It can be seen from Figure 7 and Figure 8 that when used on datasets that are much larger in size, the GA performs extremely poor. This may be attributed to the fact that the GA can easily get stuck in a local optimum. Therefore, when much larger datasets are used, the chance of getting stuck in a local optimum is much higher. It can be seen from Figure 9 that the GSA had performed consistently poor whereas the SA showed relatively average performances on these datasets.

3) HIGH-DIMENSIONAL WEAKLY CORRELATED DATASETS

Table 9 shows all the results obtained on HD-WC datasets. The main observation from this dataset was that the shortfall in terms of literature recorded the best value of zero in total for the DP algorithm. Hence, DP had achieved perfect outcomes on all datasets within this category. Moreover, we can infer from Figure 10 that BB performed second best on these datasets. The computational time is shown in Figure 11. When comparing BB and DP in terms of computational time,

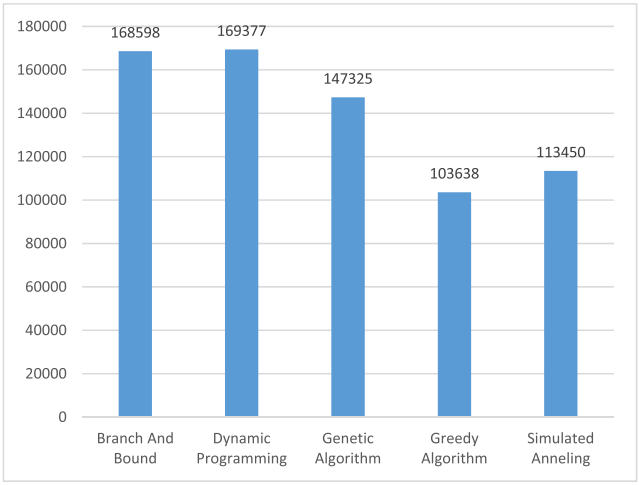

FIGURE 10. A representation of all the algorithms performances on the HD-WC datasets in terms of the best values produced.

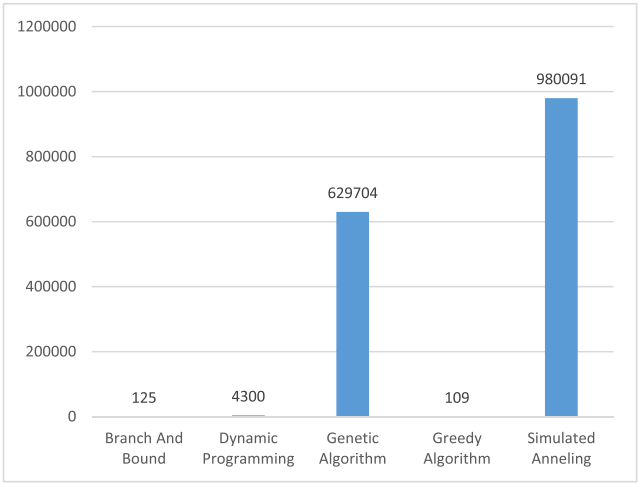

FIGURE 11. A representation of the performances of all the algorithms on the HD-WC datasets in terms of computational time .

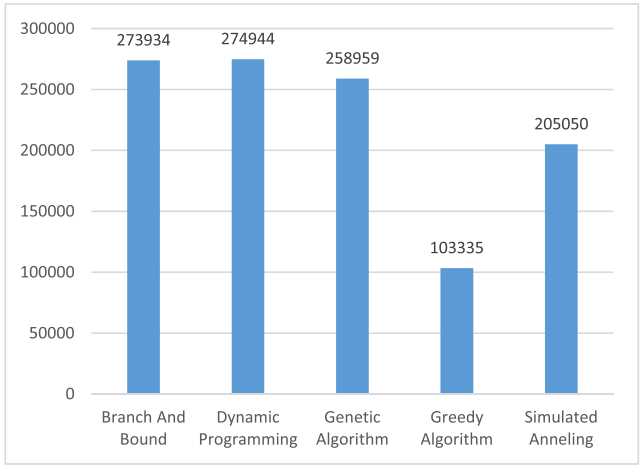

FIGURE 12. A representation of all the algorithms performances on the HD-SC datasets in terms of the best values produced.

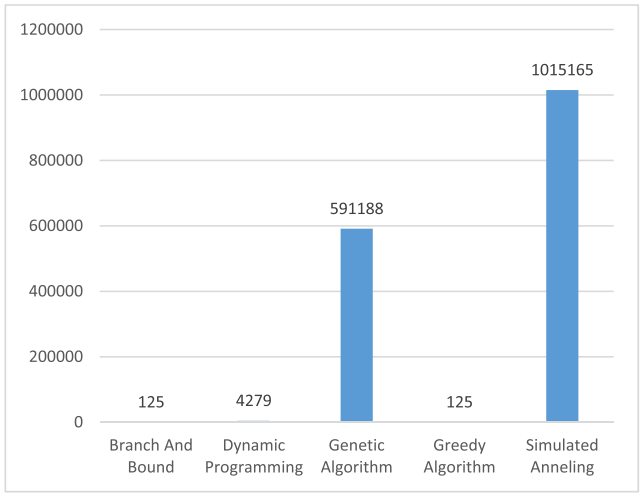

FIGURE 13. A representation of all the algorithms performances on the HD-WC datasets in terms of computational time.

it can be seen that the BB completed the dataset in 4.175s faster than the DP. Therefore, the BB algorithm did provide faster computational time for marginally lower results than the DP. While DP had taken a few seconds time longer to obtain marginally higher results. GA and SA had taken approximately 10min and 16min to complete its execution while still providing results lower than that of the BB or DP. This can be largely attributed to the time taken to create and evaluate generations in the GA algorithm and the iterative temperature changes implemented in the SA. These aspects of larger datasets increase the amount of time needed to complete execution. Amongst the five algorithms evaluated, GSA and SA performed the lowest.

4) HIGH-DIMENSIONAL STRONGLY CORRELATED DATASETS

Table 10 shows all the results obtained on HD-SC datasets. On this dataset, BB, DP and GA all performed amongst the best, while GSA and the SA performed with the lowest total values achieved. The results are illustrated in Figure 12 and the computational times are shown in Figure 13. The high performances of BB and DP along with their low computation time are trends seen inclusive of previous datasets. It can be seen that GSA had achieved the fastest convergence time on the dataset, but had produced the worst results. This is likely because of the algorithm accepting items without weight implications. This quickly leads to the GSA obtaining the best values items but not making optimal usage of it is weight, leading to an overall low value. On all datasets in this category, the DP algorithm had achieved most optimal results, followed by the BB algorithm. In this dataset category, we can still observe that BB and DP had performed in a shorter time compared to GA and SA.

5) COMPARISON BETWEEN GA AND SA

The SA and GA algorithms both have a degree of randomness associated with its performance. Therefore, these algorithms

TABLE 9. Results obtained on HD-WC datasets.

Algorithm	Dataset	Number of Items	Best Value	Optimal Value	Shortfall	Time (Milli)
		knapPI_2_100_1000_1				
SA	knapPI_2_100_1000_1	100	1486	1514	28	7703
GA	knapPI_2_100_1000_1	100	1158	1514	356	298
GSA	knapPI_2_100_1000_1	100	1041	1514	473	0
DP	knapPI_2_100_1000_1	100	1514	1514	0	1
BB	knapPI_2_100_1000_1	100	1440	1514	74	0
		knapPI_2_1000_1000_1				
SA	knapPI_2_1000_1000_1	1000	6831	9052	2221	36541
GA	knapPI_2_1000_1000_1	1000	7912	9052	1140	8835
GSA	knapPI_2_1000_1000_1	1000	5675	9052	3377	0
DP	knapPI_2_1000_1000_1	1000	9052	9052	0	41
BB	knapPI_2_1000_1000_1	1000	9006	9052	46	16
		knapPI_2_10000_1000_1				
SA	knapPI_2_10000_1000_1	10000	57852	90204	32352	570504
GA	knapPI_2_10000_1000_1	10000	79615	90204	10589	507752
GSA	knapPI_2_10000_1000_1	10000	54447	90204	35757	47
DP	knapPI_2_10000_1000_1	10000	90204	90204	0	3495
BB	knapPI_2_10000_1000_1	10000	90073	90204	131	47
		knapPI_2_200_1000_1				
SA	knapPI_2_200_1000_1	200	1537	1634	97	10699
GA	knapPI_2_200_1000_1	200	1306	1634	328	380
GSA	knapPI_2_200_1000_1	200	1073	1634	561	16
DP	knapPI_2_200_1000_1	200	1634	1634	0	2
BB	knapPI_2_200_1000_1	200	1603	1634	31	16
		knapPI_2_2000_1000_1				
SA	knapPI_2_2000_1000_1	2000	12780	18051	5271	79626
GA	knapPI_2_2000_1000_1	2000	15887	18051	2164	27151
GSA	knapPI_2_2000_1000_1	2000	11064	18051	6987	15
DP	knapPI_2_2000_1000_1	2000	18051	18051	0	107
BB	knapPI_2_2000_1000_1	2000	17794	18051	257	31
		knapPI_2_500_1000_1				
SA	knapPI_2_500_1000_1	500	3744	4566	822	19448
GA	knapPI_2_500_1000_1	500	3701	4566	865	2476
GSA	knapPI_2_500_1000_1	500	2951	4566	1615	16
DP	knapPI_2_500_1000_1	500	4566	4566	0	8
BB	knapPI_2_500_1000_1	500	4484	4566	82	0
		knapPI_2_5000_1000_1				
SA	knapPI_2_5000_1000_1	5000	29220	44356	15136	255570
GA	knapPI_2_5000_1000_1	5000	37746	44356	6610	82812
GSA	knapPI_2_5000_1000_1	5000	27387	44356	16969	15
DP	knapPI_2_5000_1000_1	5000	44356	44356	0	646
BB	knapPI_2_5000_1000_1	5000	44198	44356	158	15

were run with additional 10 times number of iterations on each of the 31 datasets. This allows us to further analyse the average performance of the algorithms and helps determine the individual algorithm performance consistency. Furthermore, from these 10 iterations we can observe the highest and lowest value obtained.

It can be seen from Figure 14 that for the LD-UC dataset, both the SA and GA performed the same on each of their iterations. This can be due to the smaller dataset dimensions making it easier to obtain the optimum values. It can also be noted that the GA had obtained a higher value than the SA.

For the larger and more complex datasets, such as the HD-UC we can observe the fluctuations between the two algorithms. This can be seen in Figure 15. It is apparent that the GA has two type of trends. It is either it converges to a

TABLE 10. Results obtained on HD-SC datasets.

Algorithm	Dataset	Number of Items	Best Value	Optimal Value	Shortfall	Time (Milli)
		knapPI_3_100_1000_1				
SA	knapPI_3_100_1000_1	100	2296	2397	101	8468
GA	knapPI_3_100_1000_1	100	2091	2397	306	247
GSA	knapPI_3_100_1000_1	100	1095	2397	1302	0
DP	knapPI_3_100_1000_1	100	2397	2397	0	2
BB	knapPI_3_100_1000_1	100	2268	2397	129	0
		knapPI_3_1000_1000_1				
SA	knapPI_3_1000_1000_1	1000	11789	14390	2601	35444
GA	knapPI_3_1000_1000_1	1000	13090	14390	1300	14300
GSA	knapPI_3_1000_1000_1	1000	5589	14390	8801	0
DP	knapPI_3_1000_1000_1	1000	14390	14390	0	28
BB	knapPI_3_1000_1000_1	1000	14271	14390	119	15
		knapPI_3_10000_1000_1				
SA	knapPI_3_10000_1000_1	10000	106114	146919	40805	590604
GA	knapPI_3_10000_1000_1	10000	124719	146919	22200	427978
GSA	knapPI_3_10000_1000_1	10000	54518	146919	92401	78
DP	knapPI_3_10000_1000_1	10000	146919	146919	0	3127
BB	knapPI_3_10000_1000_1	10000	146787	146919	132	47
		knapPI_3_200_1000_1				
SA	knapPI_3_200_1000_1	200	2594	2697	103	10552
GA	knapPI_3_200_1000_1	200	25319	2697	-22622	7
GSA	knapPI_3_200_1000_1	200	1095	2697	1602	0
DP	knapPI_3_200_1000_1	200	2697	2697	0	3
BB	knapPI_3_200_1000_1	200	2542	2697	155	16
		knapPI_3_2000_1000_1				
SA	knapPI_3_2000_1000_1	2000	22482	28919	6437	85298
GA	knapPI_3_2000_1000_1	2000	25319	28919	3600	28164
GSA	knapPI_3_2000_1000_1	2000	10818	28919	18101	16
DP	knapPI_3_2000_1000_1	2000	28919	28919	0	106
BB	knapPI_3_2000_1000_1	2000	28726	28919	193	16
		knapPI_3_500_1000_1				
SA	knapPI_3_500_1000_1	500	6103	7117	1014	21752
GA	knapPI_3_500_1000_1	500	6517	7117	600	2982
GSA	knapPI_3_500_1000_1	500	2916	7117	4201	0
DP	knapPI_3_500_1000_1	500	7117	7117	0	8
BB	knapPI_3_500_1000_1	500	6995	7117	122	0
		knapPI_3_5000_1000_1				
SA	knapPI_3_5000_1000_1	5000	53672	72505	18833	263047
GA	knapPI_3_5000_1000_1	5000	61904	72505	10601	117510
GSA	knapPI_3_5000_1000_1	5000	27304	72505	45201	31
DP	knapPI_3_5000_1000_1	5000	72505	72505	0	1005
BB	knapPI_3_5000_1000_1	5000	72345	72505	160	31

globally optimum value on the dataset indicated by the taller bars, or it converges to a local optimum and terminates, which are indicated by the shorter bars. This creates two very drastic changes in the results obtained on the same dataset. On the other hand, even though the SA did fluctuate between the 10 iterations, its results were more consistent with working towards higher values, while the bar graph shows no sign of repeated convergence towards some local value. Despite these trends in both the GA and SA, the GA still manages to obtain higher results if we compare the highest values achieved over the 10 iterations.

A similar trend can be seen in Figure 16 with the HD-SC datasets. The fluctuations on the GA graph point towards some form of global optimum and local convergences, while the fluctuations on the SA graph seem to be more randomised and doesn't indicate any form of local

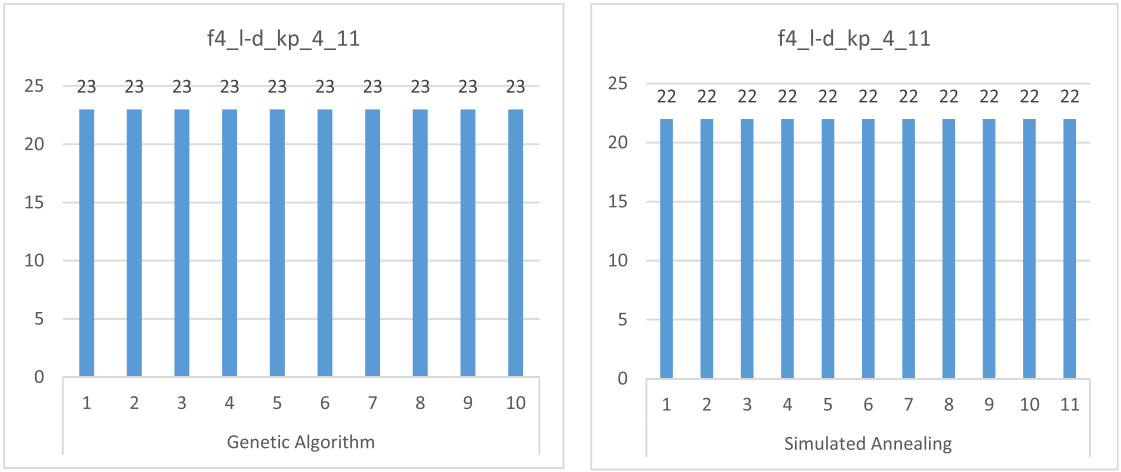

FIGURE 14. Multiple iterations of the GA and SA on a LD-UC dataset with 4 items.

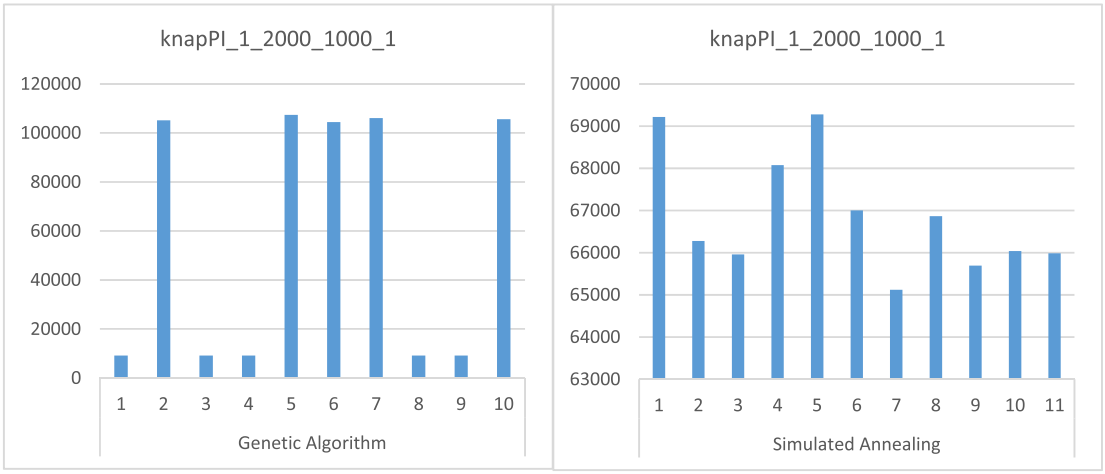

FIGURE 15. Multiple iterations of the GA and SA on a high-dimensional uncorrelated dataset with 2000 items.

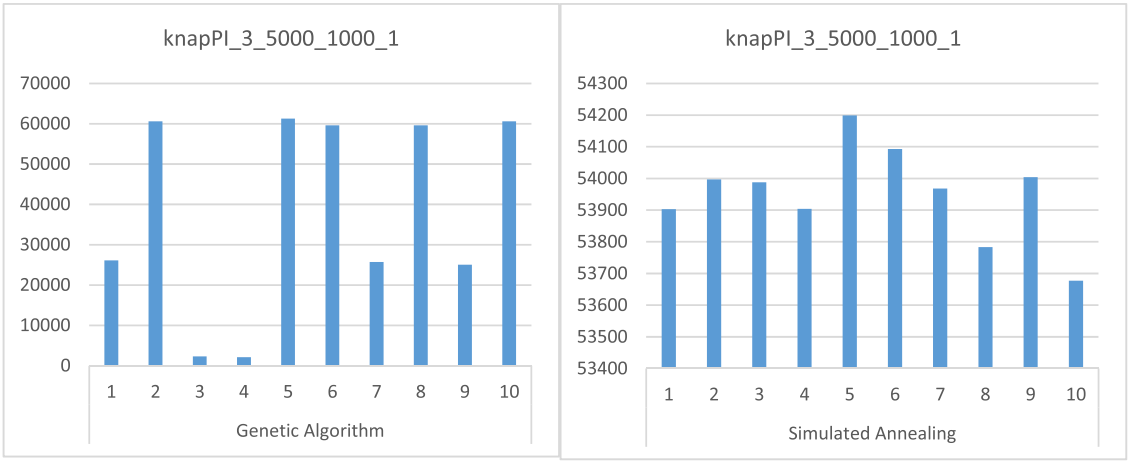

FIGURE 16. Multiple iterations of the GA and SA on a HD-SC dataset with 5000 items.

convergence. Furthermore, we can still see that the GA had obtained higher results than the SA on its iterations for all 4 types of datasets.

6) OVERALL EVALUATION STUDY FOR THE ALGORITHMS

The GA, GSA, SA, DP and BB algorithms were all implemented to evaluate their effectiveness on solving the

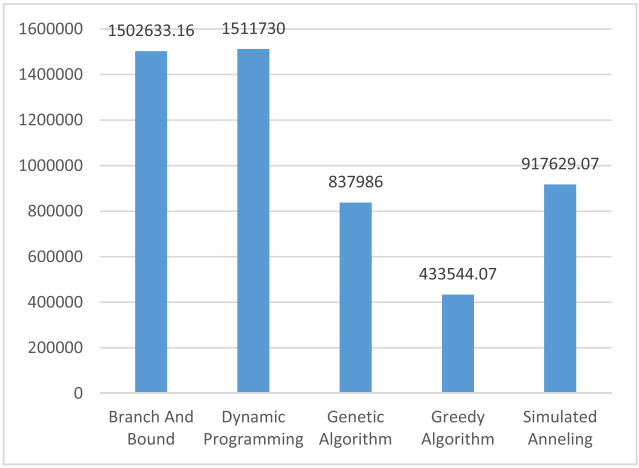

FIGURE 17. A representation of the performances on all the datasets in terms of the best values produced.

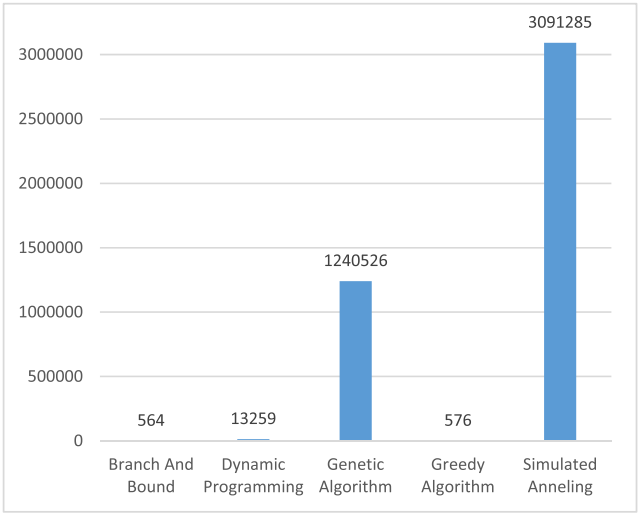

FIGURE 18. A representation of the performances on all the datasets in terms of the algorithm computational time.

KP01 problem. Figures 17 and 18 show the performances of these algorithms on all datasets in terms of the best values produced and computational time respectively. Furthermore, it was observed that the GA did produce exceptional good quality solution on the LD-UC dataset but fell short compared to the other algorithms on the high dimensional datasets. This can be due to the fact that the algorithm can easily get trapped in local optimum states. Moreover, the difficulty of obtaining effective heuristics can alter the outcomes of specific results. On the high dimensional datasets, the GA had taken up the second position most time compared to all the algorithms. This is likely due to the generation of populations for each iteration. In some high dimensional datasets this process becomes lengthy. Moreover, when the average result of each dataset was observed over 10 iterations we could distinctly observe a pattern in the results. The GA had either converged to a global optimum or a local optimum. Overall the GA has the potential to find high values solutions but does take extra computational time. The GSA had in all cases performed the fastest. This is simply due to it is nature of selecting the highest valued items without considering future implications of the weight obtained. Through all the datasets, the GSA had performed below average and with the fastest amount of time. The GSA is not an optimal algorithm in terms of solving the KP01 problem. In terms of the SA, we can observe a similar pattern to that of the GA. We had obtained comparable results in all datasets. The only exception was that the SA had taken the longest time to execute on all datasets and in some cases not achieving the most optimal result. This can also be constrained to the difficulty of fine turning all the parameter settings which could be different for each dataset. In terms of consistency, it showed improvements compared to the GA when we observe the results over 10 iterations on each dataset. Unlike the obvious convergence to local minimums experienced by the GA, there are no indications of local convergence from the SA algorithm. The DP and BB both performed very similar in terms of both the optimal value being obtained and the execution time required. On all the dataset dimensionalities both algorithms produced the most competitive results. Furthermore, we can observe that the execution time of these algorithms were far lower than the GA and SA, with the exception of GSA. On a marginal note, we can deduce that the DP had obtained an overall higher value than the BB. The important fact to note is that the DP algorithm had taken an overall time, approximately 23 times larger than the BB.

C. EXPERIMENT 2: RESULTS AND DISCUSSION

The second experiment considers an improvement strategy that would enhance the performance of the standard GA and SA. Therefore, the results of a hybrid implementation of the IGA-SA is presented in this section. The experimentation in this case comprises of three algorithms namely, GA, SA, and IGA-SA. More so, all the three algorithms were tested using the parameter configurations described in Table 4 and under the same experimental conditions, that is, each algorithm was tested on the exact same machine using the same dataset. The machine was also left idle in an effort to speed up execution and prevent the execution time of an algorithm being influenced by external factors. Due to the stochastic nature of metaheuristic algorithms, several iterations are required to gain a near optimum solution, we executed the three algorithms for 10 iterations and the best solution from those 10 iterations was chosen.

In the results shown in Table 11, the GA and IGA-SA (denoted as hybrid in the figures) algorithms produced near-optimum and even optimum solutions on datasets of low dimensionality but performance worsens as the size of the dimensionality of the problem increases, this is likely due to the complexity of the problem increasing. Overall, the simulated annealing algorithm performed the worst – it even produced infeasible solutions on the low dimensionality datasets, therefore, its results was omitted in Figures 19, 20, 21, and 22.

TABLE 11. Overall results.

Dataset	Number of Items	Optimal Value	SA	GA	Hybrid IGA-SA
f4_l-d_kp_4_11	4	23	29	23	23
f3_l-d_kp_4_20	4	35	37	35	35
f9_l-d_kp_5_80	5	130	142	130	130
f7_l-d_kp_7_50	7	107	161	107	107
f6_l-d_kp_10_60	10	52	68	52	52
f1_l-d_kp_10_269	10	295	334	295	295
f2_l-d_kp_20_878	20	1024	991	1024	1009
f10_l-d_kp_20_879	20	1025	973	1025	1025
f8_l-d_kp_23_10000	23	9767	10704	9767	9763
knapPI_2_100_1000_1	100	2397	1347	2390	1217
knapPI_3_100_1000_1	100	1514	1603	1481	1690
knapPI_1_100_1000_1	100	9147	1729	18083	8575
knapPI_2_200_1000_1	200	1634	1426	1504	1347
knapPI_3_200_1000_1	200	2697	1677	3900	1495
knapPI_1_200_1000_1	200	11238	1246	31905	8576
knapPI_2_500_1000_1	500	7117	3071	14613	3038
knapPI_3_500_1000_1	500	4566	3897	6832	3412
knapPI_1_500_1000_1	500	28857	3672	91515	12072
knapPI_2_1000_1000_1	1000	9052	111674	43991	5435
knapPI_3_1000_1000_1	1000	14390	141174	67175	6290
knapPI_1_1000_1000_1	1000	54503	110734	179560	14563
knapPI_2_2000_1000_1	2000	28919	363989	265509	10938
knapPI_3_2000_1000_1	2000	18051	432138	204527	12312
knapPI_1_2000_1000_1	2000	110625	355951	433143	27645
knapPI_2_5000_1000_1	5000	72505	1125629	1062418	25448
knapPI_3_5000_1000_1	5000	44356	1368562	889459	30302
knapPI_1_5000_1000_1	5000	276457	1127599	1168959	39677
knapPI_2_10000_1000_1	10000	90204	2432161	2087706	50965
knapPI_3_10000_1000_1	10000	563647	2878499	2417809	59716
knapPI_1_10000_1000_1	10000	146919	2385662	2500225	73749

The GA outperforms the hybrid algorithm on a few datasets but overall the hybrid algorithm outperforms the GA, as the dimensionality of the dataset increases, the GA struggles to produce feasible solutions whilst the hybrid algorithm always produces a feasible solution albeit that solution isn't optimal.

The poor performance of the GA on an uncorrelated dataset with a relatively large dimensionality can be seen in Figure 19, all 20 solutions produced are infeasible whereas all 20 solutions produced by the hybrid algorithm are feasible with one near-optimum solution (optimum value: 11238). The GA outperform the hybrid algorithm on weakly correlated datasets (Figure 20). However, the hybrid algorithm still produces feasible solutions of sufficient quality. The optimum value for the dataset used in Figure 21 was 1514.

Interestingly, the results were mixed for the experiments conducted on the strongly correlated datasets. On the dataset knapPI_3_100_1000_1 (see Table 11), the GA produced a near-optimum solution whereas the hybrid algorithm produced a solution well below the optimum solution but on the dataset knapPI_3_200_1000_1, we see the usual trend, the GA produces infeasible solution whereas the hybrid algorithm still produces feasible solutions (see Figure 22). It is worth noting that the second dataset has a higher dimensionality which might be the reason for the GA to perform poorly on the second dataset despite producing a near-optimum solution on the first dataset.

From the above graphs (Figures 19 to 22) we can gain an overall idea of the performance of the IGA-SA when compared to the SA and GA. It has been noted that the GA had produced infeasible solutions. The comparison in Figure 23 shows the comparison between the best values obtained by each of the three algorithms on the entire dataset. From this it can be assumed that the GA had greatly outperformed the SA and even the hybrid IGA-SA. Now we

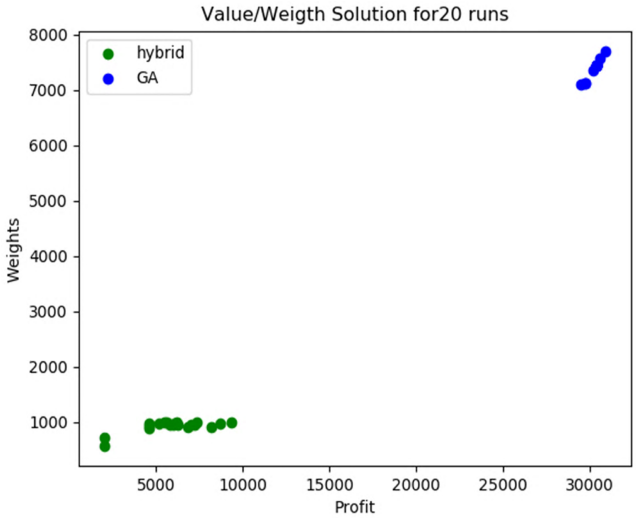

FIGURE 19. Scatter plot of solutions produced by the hybrid and GA on the uncorrelated dataset knapPI_1_200_1000_1 (Max Weight: 1008).

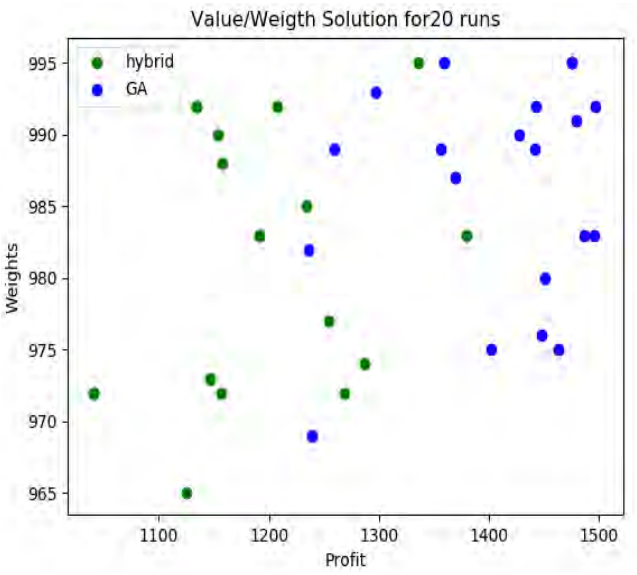

FIGURE 20. Scatter plot of solutions produced by the hybrid and GA on the weakly correlated dataset knapPI_2_100_1000_1 (Max Weight: 995).

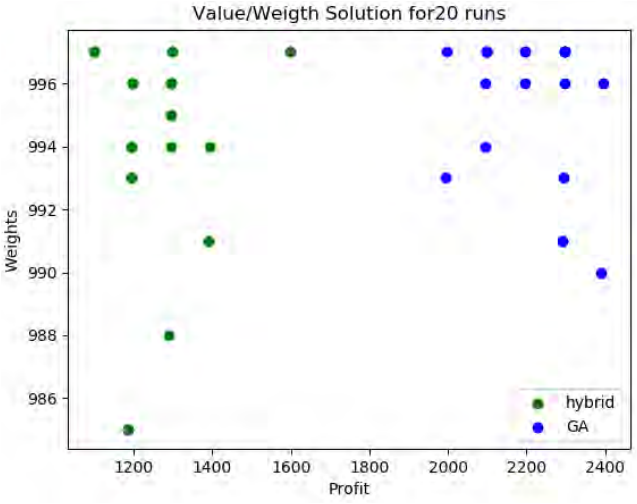

FIGURE 21. Scatter plot of solutions produced by the hybrid and GA on the strongly correlated dataset knapPI_3_100_1000_1 (Max Weight: 997).

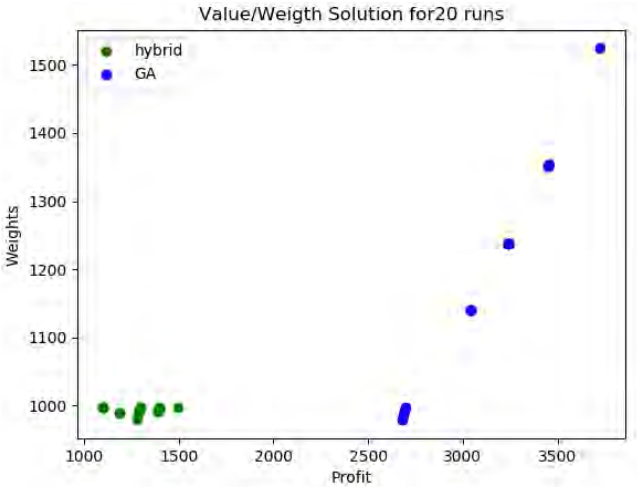

FIGURE 22. Scatter plot of solutions produced by the hybrid and GA on the strongly correlated dataset knapPI_3_200_1000_1 (Max Weight: 997).

consider the removal of invalid solutions. Figure 24 represents the best values achieved by the algorithms once the invalid solutions are removed. We can now compare the actual performance of the GA. It can be observed that the GA had produced inferior results compared to the SA and IGA-SA. It can be deduced that most likely the hybridisation of GA with SA allowed the properties of SA, specifically the property of escaping local convergence, to aid the GA in avoiding premature convergence. Therefore, this property combined with the GA's ability to explore large amounts of solution spaces allowed for an improved IGA-SA that only created feasible solutions.

Lastly, we consider the operational and computational time of the hybrid IG-SA algorithm. It can be seen in Figure 25 that the operation of the IGA-SA is much larger than the individual GA and SA algorithms. This is due to the fact that the SA refines the population generated by the GA by searching the neighbourhood of candidate solutions for more fit individuals. Hence, the operational cost is much higher. Figure 26 shows the time expenditure which directly correlates with the operational cost.

We can observe independently that GA algorithm seemed to perform well but had incorporated infeasible solutions. When combined with the SA we observe that only feasible solutions were obtained. Moreover, the SA had been able to generate populations of better quality for the GA by refining the population generated and searching the neighbourhood of a candidate solution for better fit solutions. The property of the SA which allows it to prevent accepting solutions which aren't global had further improved the GA's performance and worked in tangent with the mutation and

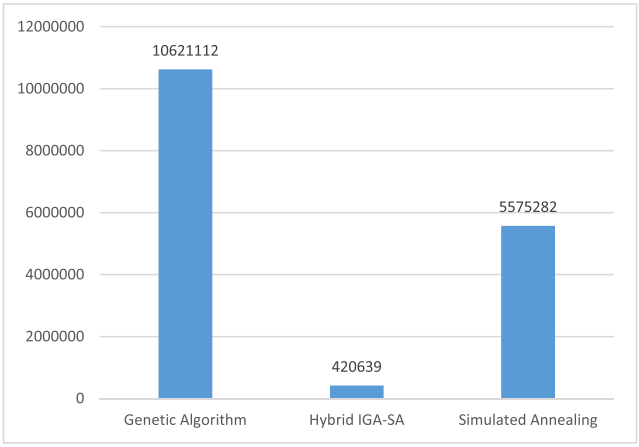

FIGURE 23. Bar graph representing best values achieved inclusive of infeasible solutions.

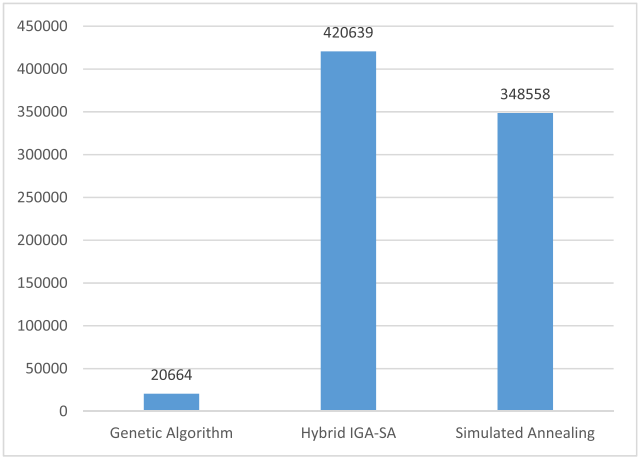

FIGURE 24. Bar graph representing best values achieved exclusive of infeasible solutions.

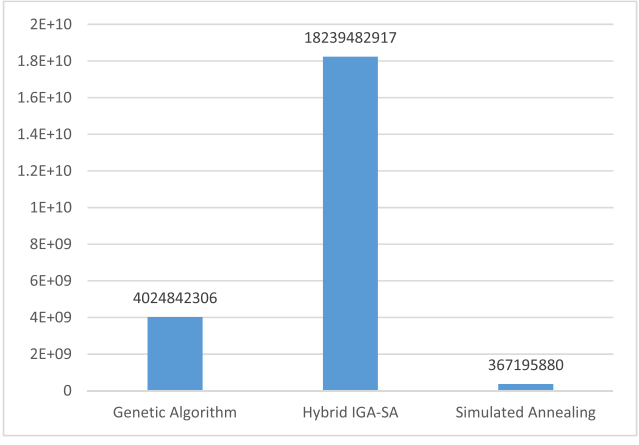

FIGURE 25. Bar graph representing total operations done by each of the constituent algorithm.

crossover properties of GA. In summary, it is clear that the hybrid IGA-SA algorithm is a suitable algorithm in solving the KP01 problem. Furthermore, it outperforms the GA and SA algorithm on majority of datasets and produces feasible

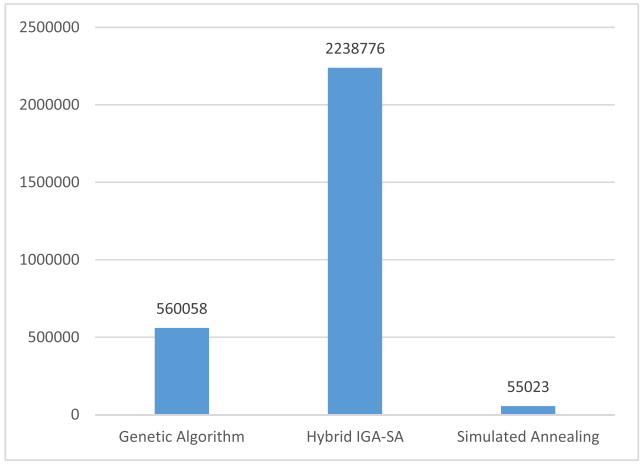

FIGURE 26. Bar graph representing total time taken by each the constituent algorithm.

solutions to the large-scale datasets whereas the GA and SA algorithms produce infeasible solutions to these datasets. However, the hybrid IGA-SA is significantly more computationally expensive since it does combine the logical components of both the GA and SA.

V. CONCLUSIONS

This paper presents some initial results of DP, BB, GSA, SA, GA and hybrid IGA-SA algorithms for solving the KPO1 problem. In the comparison of these algorithms, we had observed multiple trends and trade-offs. Some trends included the observation of the GA over 10 iterations either getting stuck in a common local optimum or achieving the global optimum. Moreover, a notable trend was seen with the BB and DP algorithms achieving global optimum results in low times. It was also observed that the population based algorithms namely GA, IGA-SA, and SA resulted in the highest computational time, simply based on the iterative nature of these algorithms. The GSA showed low time complexity, however performed poorly in terms of generating optimum values. Overall, we can deduce that both the DP and BB are the most effective approaches which were evaluated to solve the KP01 problem. DP had produced slightly more optimal values than BB, whereas BB showed a slightly lower computational time than the DP. These two algorithms performed exceptionally well on both low and high dimensional datasets with different levels of correlation. This performance was established taking into account both the algorithm time complexity as well as the optimum values found. More consideration was also given to the GA and SA due to both algorithms being nature-inspired algorithms with seemingly complementary properties. The SA has strong local search ability and is capable of escaping from local optimal solutions. However, GAs are liable to converge prematurely and be trapped in local optimal solutions. In addition, the SA is bottlenecked with high computational time. Taking into account the properties mentioned above, a hybrid of the two algorithms was implemented and the result proved to be

more superior than that of the GA and SA respectively. In the future, we intend to carry out more complex performance study on multiple greedy problems.

REFERENCES

[1] R. Merkle and M. Hellman, "Hiding information and signatures in trapdoor knapsacks," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 525–530, Sep. 1978.
[2] J. Cao, B. Yin, X. Lu, Y. Kang, and X. Chen, "A modified artificial bee colony approach for the 0-1 Knapsack problem," *Appl. Intell.*, vol 48, no. 6 pp 1582–1595, 2017.
[3] Y. Feng, G. G. Wang, S. Deb, M. Lu, and X. J. Zhao, "Solving 0-1 Knapsack problem by a novel binary monarch butterfly optimization," *Natural Comput. Appl.*, vol. 28, no. 7, pp. 1619–1634, 2017.
[4] E. Bas, "A capital budgeting problem for preventing workplace mobbing by using analytic hierarchy process and fuzzy 0-1 bidimensional Knapsack model," *Expert Syst. Appl.*, vol. 38, no. 10, pp. 12415–12422, 2011.
[5] F. Taillandier, C. Fernandez, and A. Ndiaye, "Real estate property maintenance optimization based on multiobjective multidimensional Knapsack problem," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 32, no. 3, pp. 227–251, 2017.
[6] D. Pisinger, "Heuristics for the container loading problem," *Eur. J. Oper. Res.*, vol. 141, no. 2, pp. 382–392, 2002.
[7] G. L. Reniers and K. Sörensen, "An approach for optimal allocation of safety resources: Using the Knapsack problem to take aggregated cost-efficient preventive measures," *Risk Anal.*, vol. 33, no. 11, pp. 2056–2067, 2013.
[8] P. T. Chang and J. H. Lee, "A fuzzy DEA and Knapsack formulation integrated model for project selection," *Comput. Oper. Res.*, vol. 39, no. 1, pp. 112–125, 2012.
[9] J. Pfeiffer and F. Rothlauf, "Analysis of greedy heuristics and weight-coded eas for multidimensional Knapsack problems and multi-unit combinatorial auctions," in *Proc. Conf. 9th Annu. Genetic Evol. Comput.*, Jul. 2007, p. 1529.
[10] H. C. Lau and M. K. Lim, "Multi-period multi-dimensional Knapsack problem and its application to available-to-promise," in *Proc. Int. Symp. Scheduling (ISS)*, 2004, pp. 94–99.
[11] I. Muter and Z. Sezer, "Algorithms for the one-dimensional two-stage cutting stock problem," *Eur. J. Oper. Res.*, vol. 271, no. 16, pp. 20–32, 2018.
[12] S. Peeta, F. S. Salman, D. Gunnec, and K. Viswanath, "Pre-disaster investment decisions for strengthening a highway network," *Comput. Oper. Res.*, vol. 37, no. 10, pp. 1708–1719, 2010.
[13] Y. Feng, G. G. Wang, W. Li, and N. Li, "Multi-strategy monarch butterfly optimization algorithm for discounted 0-1 Knapsack problem," *Neural Comput. Appl.*, vol. 30, no. 10, pp. 3019–3036, 2018.
[14] H. Wu, Y. Zhou, and Q. Luo, "Hybrid symbiotic organisms search algorithm for solving 0-1 Knapsack problem," *Int. J. Bio-Inspired Comput.*, vol. 12, no. 1, pp. 23–53, 2018.
[15] Z. Li, L. Ma, and H. Z. Zhang, "Genetic mutation bat algorithm for 0-1 Knapsack problem," *Comput. Eng. Appl.*, vol. 48, no. 4, pp. 50–53, 2012.
[16] B. A. Dantas and E. N. Cáceres, "An experimental evaluation of a parallel simulated annealing approach for the 0-1 multidimensional Knapsack problem," *J. Parallel Distrib. Comput.*, vol. 120, pp. 211–221, Oct. 2018.
[17] S. H. Zhan, Z. J. Zhang, L. J. Wang, and Y. W. Zhong, "List-based simulated annealing algorithm with hybrid greedy repair and optimization operator for 0-1 Knapsack problem," *IEEE Access*, vol. 6, pp. 54447–54458, 2018.
[18] L. Soukaina, N. Mohamed, E. A. Hassan, and A. Boujemâa, "A hybrid genetic algorithm for solving 0/1 Knapsack problem," in *Proc. Int. Conf. Learn. Optim. Algorithms Theory Appl.*, May 2018, p. 51.
[19] A. Rezoug, M. Bader-El-Den, and D. Boughaci, "Guided genetic algorithm for the multidimensional Knapsack problem," *Memetic Comput.*, vol. 10, no. 1, pp. 29–42, 2018.
[20] A. Kerr and K. Mullen. (2018). "A comparison of genetic algorithms and simulated annealing in maximizing the thermal conductivity of discrete massive chains," [Online]. Available: https://arxiv.org/abs/1801.09328
[21] T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Evolutionary Computation 1: Basic Algorithms and Operators*. Boca Raton, FL, USA: CRC Press, 2018.
[22] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Frome, U.K.: Luniver Press, 2010.
[23] G. B. Dantzig, "Discrete-variable extremum problems," *Oper. Res.*, vol. 5, no. 2, pp. 266–288, 1957.
[24] J. J. Bartholdi, "The Knapsack problem," in *Building Intuition*. Boston, MA, USA: Springer, 2008, pp. 19–31.
[25] A. Shaheen and A. Sleit, "Comparing between different approaches to solve the 0/1 Knapsack problem," *Int. J. Comput. Sci. Netw. Secur.*, vol. 16, no. 7, p. 1. 2016.
[26] R. Andonov, V. Poirriez, and S. Rajopadhye, "Unbounded Knapsack problem: Dynamic programming revisited," *Eur. J. Oper. Res.*, vol. 123, no. 2, pp. 394–407, Jun. 2000.
[27] R. Bellman, "Dynamic programming," *Science*, vol. 153, nos. 37–31, pp. 34–37, 1966.
[28] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*. Princeton, NJ, USA; Princeton Univ. Press. 2015.
[29] S. Dreyfus, "Richard Bellman on the birth of dynamic programming," *Oper. Res.*, vol. 50, no. 1, pp. 48–51, Jan. 2005.
[30] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.
[31] J. McCall, "Genetic algorithms for modelling and optimisation," *J. Comput. Appl. Math.*, vol. 184, no. 1, pp. 205–222, 2005.
[32] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, nos. 45–98, pp. 671–680, 1983.
[33] W. L. Goffe, G. D. Ferrier, and J. Rogers, "Global optimization of statistical functions with simulated annealing," *J. Econometrics*, vol 60, pp. 65–99, Jan. 1994.
[34] M. Hristakeva and D. Shrestha, "Solving the 0-1 Knapsack problem with genetic algorithms," in *Proc. Midwest Instruct. Comput. Symp.*, Apr. 2004, pp. 8–10.
[35] A. H. Land and A. G. Doig, "An automatic method for solving discreteprogramming problems," *Econometrica*, vol. 28, pp. 497–520, Jul. 1960.
[36] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Oper. Res.*, vol. 4, no. 4, pp. 669–719, 1966.
[37] P. J. Kolesar, "A branch and bound algorithm for the Knapsack problem," *Manage. Sci.*, vol. 13, no. 9, pp. 723–735, 1967.
[38] J. Paulus and A. Klapuri, "Music structure analysis using a probabilistic fitness measure and a greedy search algorithm," *IEEE Trans. Audio, Speech, Language Process.*, vol. 17, no. 6, pp. 1159–1170, Aug. 2009.
[39] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *J. Global Optim.*, vol. 6, no. 2, pp. 109–133, 1995.
[40] T. K. Truong, K. Li, and Y. Xu, "Chemical reaction optimization with greedy strategy for the 0-1 Knapsack problem," *Appl. Soft Comput.*, vol. 13, no. 4, pp. 1774–1780, 2013.
[41] S. Martello, D. Pisinger, and P. Toth, "Dynamic programming and strong bounds for the 0-1 Knapsack problem," *Manage. Sci.*, vol. 45, no. 3, pp. 414–424, 1999.
[42] S. Martello, D. Pisinger, and P. Toth, "New trends in exact algorithms for the 0-1 Knapsack problem," *Eur. J. Oper. Res.*, vol. 123, no. 2, pp. 325–332, 2000.
[43] G. T. Ross and R. M. Soland, "A branch and bound algorithm for the generalized assignment problem," *Math. Program.*, vol. 8, no. 1, pp. 91–103, Dec. 1975.
[44] O. K. Gupta and A. Ravindran, "Branch and bound experiments in convex nonlinear integer programming," *Manage. Sci.*, vol. 31, no. 12, pp. 1533–1546, 1985.
[45] C. Changdar, G. S. Mahapatra, and R. K. Pal, "An improved genetic algorithm based approach to solve constrained Knapsack problem in fuzzy environment," *Expert Syst. Appl.*, vol. 42, no. 4, pp. 2276–2286, 2015.
[46] M. Hristakeva and D. Shrestha, "Different approaches to solve the 0/1 Knapsack problem," in *Proc. Midwest Instruct. Comput. Symp.*, Apr. 2005, pp. 1–15.
[47] X. G. Li and X. Wei, "An improved genetic algorithm-simulated annealing hybrid algorithm for the optimization of multiple reservoirs," *Water Resour. Manage.*, vol. 22, no. 8, pp. 1031–1049, 2008.
[48] *Instances of 0/1 Knapsack Problem*. Accessed: Nov. 21, 2018. [Online]. Available: http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/
[49] S. Martello and P. Toth, "An upper bound for the zero-one Knapsack problem and a branch and bound algorithm," *Eur. J. Oper. Res.*, vol. 1, no. 3, pp. 169–175, 1977.
[50] R. M. Nauss, "An efficient algorithm for the 0-1 Knapsack problem," *Manage. Sci.*, vol. 23, no. 1, pp. 27–31, 1976.

[51] A. Bettinelli, V. Cacchiani, and E. Malaguti, "A branch-and-bound algorithm for the Knapsack problem with conflict graph," *Informs J. Comput.*, vol. 29, no. 3, pp. 457–473, 2017.

[52] F. Tari, *Exact Solution Algorithms for Multi-Dimensional Multiple-Choice Knapsack Problems*. West Bengal, India: Current Journal of Applied Science and Technology, 2018. doi: 10.9734/CJAST/2018/40420.

[53] A. J. Kleywegt and D. J. Papastavrou, "The dynamic and stochastic Knapsack problem," *Oper. Res.*, vol. 46. no. 1, pp. 17–35, 1998.

[54] S. K. Pushpa, T. V. Mrunal, and C. Suhas, "A study of performance analysis on Knapsack problem," *Int. J. Comput. Appl.*, to be published.

[55] *Solving the Knapsack Problem with a Simple Genetic Algorithm*. Accessed: Nov. 9, 2018. [Online]. Available: https://www.dataminingapps.com/2017/03/solving-the-Knapsack-problem-with-a-simple-genetic-algorithm/

[56] V. Yadav and S. Singh, "A review paper on solving 0-1 Knapsack problem with genetic algorithms," *Int. J. Comput. Sci. Inf. Technol.*, vol. 7, no. 2, pp. 830–832, 2016.

[57] V. Yadav and S. Singh, "Genetic algorithms based approach to solve 0-1 Knapsack problem optimization problem," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 4, no. 5, pp. 8595–8602, May 2016.

[58] E. Sonuc, B. Sen, and S. Bayir, "Solving bin packing problem using simulated annealing," in *Proc. 65th ISERD Int. Conf.*, Mecca, Saudi Arabia, Jan. 2017, p. 3.

[59] S. Manaseer and H. Almogdady, "New hybrid approach to solve the 0/1," *Bounded Knapsack*, vol. 3, p. 125, Mar. 2017.

[60] F.-T. Lin, C.-Y. Kao, and C.-C. Hsu, "Applying the genetic algorithm approach tosimulated annealing in solving some NP-Hard problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 6, pp. 1752–1767, Jul. 1993. doi: 10.1109/21.257766.

[61] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht, The Netherlands: Holland, 1987.

[62] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines-A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York, NY, USA: Wiley, 1989.

[63] Y. Zhou, X. Chen, and G. Zhou, "An improved monkey algorithm for a 0-1 Knapsack problem," *Appl. Soft Comput.*, vol. 38, pp. 817–830, Jan. 2016.

[64] B. S. Yildiz and A. R. Yildiz, "Comparison of grey wolf, whale, water cycle, ant lion and sine-cosine algorithms for the optimization of a vehicle engine connecting rod," *Mater. Testing*, vol. 60, no. 3, pp. 311–315, 2018. doi: 10.3139/120.111153.

[65] B. S. Yildiz and A. R. Yildiz, "Moth-flame optimization algorithm to determine optimal machining parameters in manufacturing processes," *Mater. Test.*, vol. 59, no. 5, pp. 425–429, 2017. doi: 10.3139/120.1110242017.

[66] F. Hamza, H. Abderazek, S. Lakhdar, D. Ferhat, and A. R. Yildiz, "Optimum design of cam-roller follower mechanism using a new evolutionary algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 99, nos. 5–8, pp. 1267–1282, 2018.

[67] A. R. Yildiz and K. N. Solanki, "Multi-objective optimization of vehicle crashworthiness using a new particle swarm based approach," *Int. J. Adv. Manuf. Technol.*, vol. 59, nos. 1–4, pp. 367–376, 2012.

[68] A. R. Yildiz, E. Kurtulus, E. Demirci, B. S. Yildiz, and S. Karagöz, "Optimization of thin-wall structures using hybrid gravitational search and Nelder-Mead algorithm," *Mater. Testing*, vol. 58, no. 1, pp. 75–78, 2016.

[69] A. R. Yildiz, "A new hybrid bee colony optimization approach for robust optimal design and manufacturing," *Appl. Soft Comput.*, vol. 13, no. 5, pp. 2906–2912, 2013.

[70] B. S. Yildiz, "A comparative investigation of eight recent population-based optimisation algorithms for mechanical and structural design problems," *Int. J. Vehicle Des.*, vol. 73, nos. 1–3, pp. 208–218, 2017. doi: 10.1504/IJVD.2017.10003412.

[71] S. Karagöz and A. R. Yildiz, "A comparison of recent metaheuristic algorithms for crashworthiness optimisation of vehicle thin-walled tubes considering sheet metal forming effects," *Int. J. Vehicle Des.*, vol. 73, nos. 1–3, pp. 179–188, 2017.

[72] N. Pholdee, S. Bureerat, and A. R. Yildiz, "Hybrid real-code population-based incremental learning and differential evolution for many-objective optimisation of an automotive floor-frame," *Int. J. Vehicle Des.*, vol. 73, nos. 1–3, pp. 20–53, 2017.

ABSALOM E. EZUGWU received the B.Sc. degree in mathematics with computer science and the M.Sc. and Ph.D. degrees in computer science from Ahmadu Bello University, Zaria, Nigeria. He is currently a Senior Lecturer with the School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Durban, South Africa. He has published articles relevant to his research interest in internationally referred journals and edited books, conference proceedings, and local journals. His main research interests include parallel algorithms design in cloud and grid computing environments, artificial intelligence with specific interest to computational intelligence, and metaheuristics solutions to real-world global optimization problems. He is a member of IAENG and ORSSA.

VEROSHA PILLAY received the B.Sc. degree in computer science and information technology and the Honours degree *(summa cum laude)* in computer science from the University of KwaZulu-Natal, in 2018. Her honours research work was focused in the area of computer vision and machine learning. She is currently pursuing the master's degree in computer science with an interest in computer vision and deep learning.

DIVYAN HIRASEN received the B.Sc. degree in computer science and information technology from the University of KwaZulu-Natal, in 2017, and the B.Sc. degree (Hons.) *(cum laude)* in computer science, in 2018, with a keen interest in the machine learning and computer vision research domain. He is currently pursuing the master's degree in computer science and exploring this research domain further.

KERSHEN SIVANARAIN received the B.Sc. degree *(cum laude)* in computer science and information technology from the University of KwaZulu-Natal (UKZN), in 2017, and the B.Sc. degree (Hons.) *(summa cum laude)* in computer science, UKZN, in 2018, with his research area in computer vision and machine learning. He intends on pursuing his master's in this area and is currently working as a Software Developer for an online gaming company.

MELVIN GOVENDER received the B.Sc. degree in computer science from the University of KwaZulu-Natal, in 2017, and the B.Sc. degree (Hons.) in computer science, in 2018. He pursued his research in artificial intelligence with the integration of music. His research work focused on the ensemble approach for composing melody and harmony.