Original Research Article

Meta-Heuristics Approach To Knapsack Problem In Memory Management

ABSTRACT

The Knapsack Problems are among the simplest integer programs which are NP-hard. Problems in this class are typically concerned with selecting from a set of given items, each with a specified weight and value, a subset of items whose weight sum does not exceed a prescribed capacity and whose value is maximum. The classical 0-1 Knapsack Problem arises when there is one knapsack and one item of each type. This paper considers the application of classical 0-1 knapsack problem with a single constraint to computer memory management. The goal is to achieve higher efficiency with memory management in computer systems.
This study focuses on using simulated annealing and genetic algorithm for the solution of knapsack problems. It is shown that Simulated Annealing performs better than the Genetic Algorithm for large number of processes.

Keywords: Knapsack, Memory Management, Genetic Algorithm, Simulated Annealing

1. INTRODUCTION

A great variety of practical problems can be represented by a set of entities, each having an associated value, from which one or more subsets has to be selected in such a way that the sum of the values of the selected entities is maximized, and some predefined conditions are respected. The most common condition is obtained by also associating a weight to each entity and establishing that the sum of the entity sizes in each subset does not exceed some prefixed bound. These problems are generally called knapsack problems, since they recall the situation of a traveler having to fill up his knapsack by selecting from among various possible objects those which will give him the maximum comfort. One such problem is in computer memory management.

Modern computer memory management is for some causes a crucial element of assembling current large applications. First, in large applications, space can be a problem and some technology are efficiently needed to return unused space to the program. Secondly, inexpert implementations can result in extremely unproductive programs since memory management takes a momentous portion of total program execution time and finally, memory errors become rampant, such that it is extremely difficult to find programs when accessing freed memory cells. It is much secured to build more unfailing memory management into design even though complicated tools exist for revealing a variety of memory faults. It is for this basis that efficient schemes are needed to manage allocating and freeing of memory by programs.

Optimizing current memory management strategies strength is performed by altering the space allocated to each task. To achieve high levels of multiprogramming while avoiding thrashing such policies vary the load (i.e., the number of active tasks). Additionally, in a system that runs out of capacity probably because the system is undersized, several options are available. This option includes either upgrading the processor (if possible), reduce available functionality, or optimize.
A great deal of realistic problems where some predefined conditions are respected such that the sum of the values of the selected entities is maximized can be represented by a set of entities, each having an associated value, from which one or more subsets has to be selected. The most ordinary situation is obtained by establishing that the sum of the entity sizes in each subset does not exceed some prefixed bound by associating a weight/size to each entity.

The goal of this paper is to maximize the number of processes in a limited memory space.

2. LITERATURE REVIEW

Knapsack problems have been studied intensively in the past decade attracting both theorist and practitioners. The theoretical interest arises mainly from their simple structure which both allows exploitation of a number of combinational properties and permits more complex optimization problems to be solved through a series of knapsack type. From a practical point of view, these problems can model many industrial applications, the most classical applications being capital budgets, cargo loading and cutting stock. In this section a review of literature on knapsack problems and applications is presented.

The knapsack problem (KP) is a traditional combinatorial issue used to show numerous modern circumstances. ―Since Balas and Zemel a dozen years ago introduced the so-called core problem as an efficient way of solving the Knapsack Problem, all the most successful algorithms have been based on this idea. All knapsack Problems belong to the family of NP-hard problems, meaning that it is very unlikely that polynomial algorithms for these problems can be devised [1].

The Knapsack problem has been concentrated on for over a century with prior work dating as far back as 1897. ―It is not known how the name Knapsack originated though the problem was referred to as such in early work of mathematician Tobias Dantzig suggesting that the name could have existed in folklore before mathematical problem has been fully defined [2].
Given a knapsack of limit, Z, and n dissimilar items, Caceres and Nishibe [3] algorithm resolved the single Knapsack problem using local computation time with communication rounds. With dynamic programming, their algorithm solved locally pieces of the Knapsack problem. The algorithm was implemented in Beowulf and the obtained time showed good speed-up and scalability [4].
Heuristic algorithms experienced in literature that can generally be named as population heuristics include; ―genetic algorithms, hybrid genetic algorithms, mimetic algorithms, scatter-search algorithms and bionomic algorithms‖. Among these, Genetic Algorithms have risen as a dominant latest search paradigm [5].

Eager about making use of a easy heuristic scheme (simple flip) for answering the knapsack problems, Oppong [6] offered a study work on the application of usual zero-1 knapsack trouble with a single limitation to determination of television ads at significant time such as prime time news, news adjacencies, breaking news and peak times.
Martello et al [7] presented a new algorithm for the optimal solution of the 0-1 Knapsack problem, which is particularly effective for large-size problems. The algorithm is based on determination of an appropriate small subset of items and the solution of the corresponding "core problem": from this they derived a heuristic solution for the original problem which, with high probability, can be proved to be optimal. The algorithm incorporated a new method of computation of upper bounds and efficient implementations of reduction procedures.

Huttler and Mastrolilli [8] addressed the classical knapsack problem and a variant in which an upper bound is imposed on the number of items that can be selected. It was shown that appropriate combinations of rounding techniques yield novel and more powerful ways of rounding. Moreover, they presented a linear-storage polynomial time approximation scheme (PTAS) and a fully polynomial time approximation scheme (FPTAS) that compute an approximate solution, of any fixed accuracy, in linear time. These linear complexity bounds give a substantial improvement of the best previously known polynomial bounds.

Hanafi and Freville [9] described a new approach to tabu search (TS) based on strategic oscillation and surrogate constraint information that provides a balance between intensification and diversification strategies. New rules needed to control the oscillation process are given for the 0 /1 multidimensional knapsack (0/1 MKP). Based on a portfolio of test problems from the literature, our method obtains solutions whose quality is at least as good as the best solutions obtained by previous methods, especially with large scale instances. These encouraging results confirm the efficiency of the tunneling concept coupled with surrogate information when resource constraints are present.

Rinnooy et al. [10] proposed a class of generalized greedy algorithms is for the solution of the multi-knapsack problem. Items are selected according to decreasing ratios of their profit and a weighted sum of their requirement coefficients. The solution obtained depended on the choice of the weights. A geometrical representation of the method was given and the relation to the dual of the linear programming relaxation of multi-knapsack is exploited. They investigated the complexity of computing a set of weights that gives the maximum greedy solution value. Finally, the heuristics were subjected to both a worst-case and a probabilistic performance analysis.

Balachandar and Kannan [11] presented a heuristic to solve the 0/1 multi-constrained knapsack problem (0/1 MKP) which is NP-hard. In this heuristic the dominance property of the constraints is exploited to reduce the search space to find near optimal solutions of 0/1 MKP. This heuristic was tested for 10 benchmark problems of sizes up to 105 and for seven classical problems of sizes up to 500, taken from the literature and the results were compared with optimum solutions. Space and computational complexity of solving 0/1 MKP using this approach were also presented. The encouraging results especially for relatively large size test problems indicate that this heuristic can successfully be used for finding good solutions for highly constrained NP-hard problems.
Elhedhli [12] considered a class of nonlinear knapsack problems with applications in service systems design and facility location problems with congestion. They provided two linearizations and their respective solution approaches. The first is solved directly using a commercial solver. The second is a piecewise linearization that is solved by a cutting plane method.

Devyaterikova et al. [13] presented discrete production planning problem which may be formulated as the multidimensional knapsack problem is considered, while resource quantities of the problem are supposed to be given as intervals. An approach for solving this problem based on using its relaxation set is suggested. Some *L*-class enumeration algorithms for the problem are described. Results of computational experiments were presented.
Chen et al. [14] presented pipeline architectures for the dynamic programming algorithms for the knapsack problems. They enabled them to achieve an optimal speedup using processor arrays, queues, and memory modules. The processor arrays can be regarded as pipelines where the dynamic programming algorithms are implemented through pipelining.

3. METHODOLOGY

Because of their wide range of applicability, knapsack problems have known a large number of variations such as: single and multiple-constrained knapsacks, knapsacks with disjunctive constraints, multidimensional knapsacks, multiple choice knapsacks, single and multiple objective knapsacks, integer, linear, non-linear knapsacks, deterministic and stochastic knapsacks, knapsacks with convex / concave objective functions, etc.

This is a 0-1 knapsack problem, pure integer programming with single constraint which forms a very important class of integer programming**.**
The 0-1 Knapsack Problem (KP) can be mathematically formulated through the following integer linear programming.

$$\text{Maximize} \sum_{j=1}^{n} P_j x_j \qquad [1]$$

$$\text{Subject to } = \sum_{j=1}^{n} (w_j x_j) \leq c \qquad [2]$$

$x_j = 0 \text{ or } 1,\ j = 1, \ldots, n$

Where, P_j refers to the value, or worth of item j, x_j refers to the item j, w_j refers to the relative-weight of item *j*, with respect to the knapsack and C refers to the capacity, or weight-constraint of the knapsack. There exist *j* = 1…*n* items, and there is only one knapsack.

The use of two major meta-heuristics approaches, Genetic algorithm and Simulated annealing which have been used to solve large scale problems [15] will be considered in this paper.

3.1 Simulated Annealing

Simulated annealing (SA) is a local search algorithm capable of escaping from local optima. Its case of implementation, convergence properties and its capability of escaping from local optima has made it a popular algorithm over the past decades. Simulated annealing is so named because of its analogy to the process of physical annealing with solids in which a crystalline solid is heated and then allowed to cool very slowly until it achieves stable state. i.e. its minimum lattice energy state and thus is free of crystal effects. Simulated annealing mimics this type of thermodynamic behavior in searching for global optima for discrete optimization problems (DOP).

At each iteration of simulated annealing, algorithm applied to a DOP, the objective function values for two solutions (the current solution and a newly generated neighboring solution) are compared. Better solutions are always accepted, while a fraction of inferior solutions is accepted in the hope of escaping local optima in search of global optima. The probability of accepting non-improving solutions depends on a temperature parameter, which is non-increasing with each iteration of the algorithm.
The key algorithm feature of simulated annealing is that provides a means to escape local optima by allowing worse moves (i.e. moves to a solution that corresponds to a worse objective value function). As the temperature is decreased to zero, worse moves occur less frequently and the solution distribution associated with the inhomogeneous Markov chain that models the behavior of the algorithm converges to a distribution in which all the probability is concentrated on the set of globally optimal solutions which means that the algorithm is asymptotically convergent.

To formally describe simulated annealing algorithm for KP, some definitions are needed. Let Ω be the solution space: define η(ω) to be the neighborhood function for w ∈ Ω. Simulated annealing starts with an initial solution ω ∈ Ω. A neighborhood solution $\omega^{1} \in \eta(\omega)$ is then generated randomly in most cases. Simulated annealing is based on the Metropolis acceptance criterion, which models how a thermodynamic system moves from its current solution ω ∈ Ω to a candidate solution $\omega i \in \eta(\omega)$ in which the energy content is being minimized. The candidate solution ω^{1} is accepted as the current solution based on the acceptance probability.
In this survey, finite-time implementations of simulated annealing algorithm are considered, which can no longer guarantee to find an optimal solution, but may result in faster executions without losing too much on the solution quality. Simulated annealing algorithm with static cooling schedule [16] for KP is outlined in pseudo-code.

```
1  Select an initial solution $\omega = (x_1, \ldots, x_n) \in \Omega$; an initial temperature $t = t_0$;
2  control parameter value $\alpha$; final temperature $e$; a repetition schedule, M that defines the number of
   iterations executed at each temperature;
3  Incumbent solution $\leftarrow f(\omega)$;
4  Repeat;
5  Set repetition counter $m = 0$;
6  Repeat;
7  Select an integer i from the set $\{1,2,\ldots,n\}$ randomly:
8  If $x_i = 0$, pick up item i, i.e. set $x_i = 1$, obtain new solution $\omega 1$ then
9  while solution $\omega 1$ is infeasible, do
10 drop another item from $\omega$ randomly; denote the new solution as $\omega 1$
11 let $\Delta = f(\omega 1) - f(\omega)$
12 while $\Delta \geq 0$ or Random (0,1) < $e^{\Delta/t}$ do $\omega \leftarrow \omega 1$
13 Else
14 drop item i and pick another item randomly, get new solution $\omega 1$
15 let $\Delta = f(\omega 1) - f(\omega)$
16 while $\Delta \geq 0$ or Random (0,1) $< e^{\Delta/t}$ do $\omega \leftarrow \omega 1$
17 End If
18 If incumbent solution $< f(\omega)$, Incumbent solution $\leftarrow f(\omega)$
19 $m = m + 1$;
20 Until $m = M$
21 set $t = a * t$;
22 Until $t < e$
```

A set of parameters needs to be specified that govern the convergence of the algorithm, i.e. initial temperature to, temperature control parameter α, final temperature e, and Markov chain length M, in order to study the finite-time performance of simulated annealing algorithm. Here t_o should be the maximal difference in cost between any two neighboring solutions [6].

3.2 Genetic Algorithm

A genetic algorithm (GA) can be described as an "intelligent" probabilistic search algorithm and is based on the evolutionary process of biological organisms in nature. During the course of evolution, natural populations evolve according to the principles of nature selection and "survival of the fittest." Individuals who are most successful in adapting to their environment will have a better chance of surviving and reproducing, while individuals who are less fit will be eliminated. This means that the genes from highly fit individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from highly adapted parents may produce even more fit offspring. In this way, species evolve to become increasingly better adapted to the environment.

A GA simulates these processes by taking an initial population of individuals and applying genetic operators in each reproduction. In optimization terms, each individual in the population is encoded into a string or chromosome that represents a possible solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or solutions are given opportunities to reproduce by exchanging pieces of their genetic information in a crossover procedure with other highly fit individuals. This produces new "offspring" solutions (i.e. children) who share some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the strings. The offspring can either replace the whole population (generational approach) or replace fewer fit individuals (steady-state approach). This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found.

The basic steps of a simple GA are shown below
Step 1: Generate an initial population

Step 2: Evaluate fitness of individuals in the population
The objective function value ($\sum_{j=1}^{n} pjXj$) equates to how good a solution is, that is, its fitness.
In general, an initial population is randomly generated in some way.

Step 3: repeat

a. Select individuals from the population to be parents
   In the GA world for the KP, parents will be chosen by binary tournament selection. In binary tournament selection, two individuals are randomly selected from the population. From these two, the individual with the best fitness is selected to be the first parent
b. Recombine (mate) parents to produce children
   In the GA world for the KP, a single child will be obtained from two parents by uniform crossover. In uniform crossover each bit in the child solution is created by:
   repeat for each bit in turn
      choose one of the two parents at random
      set the child bit equal to the bit in the chosen parent
   In one-point crossover, a pint between two adjacent bits is randomly selected, "cut" the parents into two segments and create two children by rejoining the segments.
c. Mutate the children Evaluate fitness of the children
   Mutation corresponds to small changes that are stochastically applied to the children
   Mutation can be applied with a constant probability or with an adaptive probability that changes over the course of the algorithm (perhaps in response to the number of iterations that have passed or in response to population characteristics).
d. Replace some or all of the population by the children

until

Step 4: you decide to stop whereupon report the best solution encountered

To perform the simulations, theses are the parameters used for the above methods described.
The parameters used for the Simulated Annealing are:

Cooling factor: 0.98
Termination Temperature: 0.2
Initial Temperature: 100
Neighbor Sampling Size: 350

The parameters used for the Genetic Algorithm are:
Population Size: 500
Recombination Rate:0.7
Mutation Rate: 0.005
Number of Crossover Points: 3

4. ANALYSIS AND RESULTS

Category A: The computer system with a total of 10 created processes, all with their system information in figures. The computer memory can accommodate capacity of 50mb but the total memory of the process is 56 with a combined process activity (number of times process is accessed of 123

Table 1: Results for Category A

	GA	SA
No. of Processes Used	9	9
Memory Used	46	46
Number of Times Process Is Accessed	119	119

From Table 1, it could be seen that all three algorithms provide the same output in terms of all the parameters under consideration. This means that both DP, GA and SA

Category B: The table below shows a computer system with a total of 50 created processes, all with their system information in figures. The computer memory can accommodate capacity of 100mb. but the total memory of the process is 281 with a combined process activity (number of times process is accessed of 483

Table 2: Results for Category B

	GA	SA
No. of Processes Used	25	23
Memory Used	100	100
Number of Times Process Is Accessed	327	328

From Table 2, GA provided a slight advantage of in terms of the number of process used. Apart from that all three algorithms provided fairly the same result

Category C: The table below shows a computer system with a total of 100 created processes, all with their system information in figures. The computer memory can accommodate capacity of 300mb. but the total memory of the process is 574 with a combined process activity (number of times process is accessed of 1011

Table 3: Results for Category C

	GA	SA
No. of Processes Used	61	62
Memory Used	300	300
Number of Times Process Is Accessed	815	803

Table 3 shows that DP provides a better result than the rest. All memory needed was utilized showing efficient use of memory available.

Category D: The table below shows a computer system with a total of 500 created processes, all with their system information in figures. The computer memory can accommodate capacity of 1000mb. but the total memory of the process is 2661 with a combined process activity (number of times process is accessed of 5287

Table 4: Results for Category D

	GA	SA
No. of Processes Used	258	252
Memory Used	1000	1000
Number of Times Process Is Accessed	3551	3431

Category E: The table below shows a computer system with a total of 1000 created processes, all with their system information in figures. The computer memory can accommodate capacity of 5000mb. but the total memory of the process is 5626 with a combined process activity (number of times process is accessed of 10480).

Table 5: Results for Category E

	GA	SA
No. of Processes Used	915	916
Memory Used	5000	5000
Number of Times Process Is Accessed	10299	10307

GA and Sa provide fairly the same results in Table 4 and 5.

The main criteria in evaluating the efficiency of an algorithm is time and space. Even though in terms of results the three algorithms provided similar results, their efficiency will be determined based on the time it took to produce the results and the amount of memory resource it took on the computer.

Table 6: Results for based on Time Taken

TIME (ms)		
No. of Process	**GA**	**SA**
10	436	60
50	323	52
100	385	87
500	1374	300
1000	2338	554

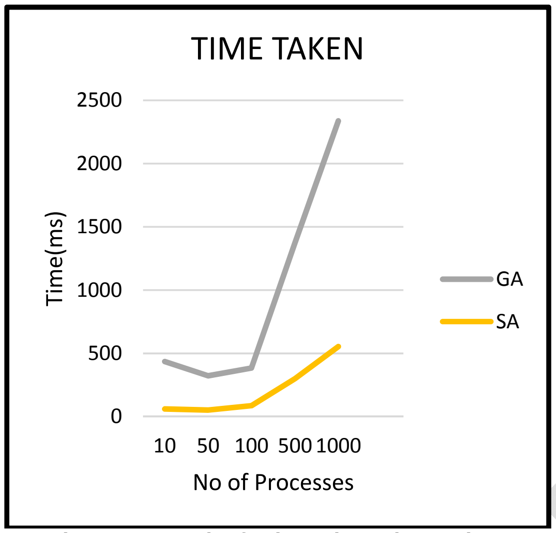

Figure 1: Results for based on Time Taken

From Table 6 and Figure 1, It is seen that GA took more time in giving an optimum out than SA for larger number of processes. As the number of processes increases, time taken increases exponentially for GA as compared to SA.
Also the GA also used more memory utilization for than SA from Table 7 and Figure 2. The GA outperformed the Sa only when the number of processes

Table 7: Results for based on Memory Taken

MEMORY (byte)		
No. of Process	**GA**	**SA**
10	28880312	42511800
50	92815928	45555312
100	100774992	73927720
500	210273904	117057112
1000	233449048	210256440

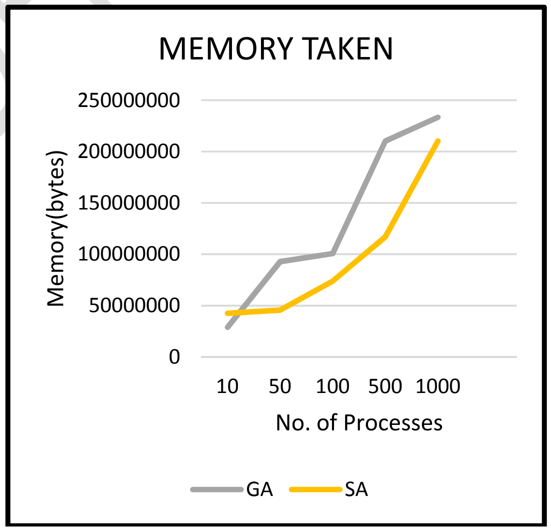

Figure 2: Results for based on Memory Taken

5. CONCLUSION AND RECOMMENDATIONS

This paper showed that memory optimization as well as knapsack problem can be successfully solved using heuristic algorithms
In this paper, meta-heuristic algorithms i.e. simulated annealing and genetic algorithm were testes compared for their efficiency in optimizing memory.
Experiments with simulated annealing showed that increase in number of processes gives better result than the Genetic Algorithm. From the analysis, it can be seen that for smaller number of processes the GA and SA performance are identical but as the number of processes increases, SA performs better than GA.
Therefore, it is concluded that, the most efficient algorithm in knapsack optimizing among the two for large number of processes is Simulated Annealing.

REFERENCES

[1] Pisinger, D. (1994). Core problems in knapsack algorithms. Operations Research 47, 570-575.

[2] Kellerer, H., Pferschy, U., Pisinger, D. (2004). Knapsack Problems. Springer, Berlin Heidelberg.

[3] Cáceres E. N., and Nishibe, C. (2005). 0-1 Knapsack Problem: BSP/CGM Algorithm and Implementation. IASTED PDCS: 331-335.

[4] Robert M, & Thompson, K (1978). Password Security: A Case History. Bell Laboratories, K8.

[5] Chu P.C and Beasley J. E. (1998), A genetic algorithm for multidimensional knapsack problem. Journal Heuristics. 4:63-68.

[6] Oppong, O. E. (2009). Optimal resource Allocation Using Knapsack Problems: A case Study of Television Advertisements at GTV. Master's degree paper, KNUST.

[7] Martello S., Pisinger D., Toth P., (2000). New trends in exact algorithms for the 0–1 knapsack problem

[8] Mastrolilli M., Huttler M., (2006). Hybrid rounding techniques for knapsack problems. www.sciencedirect.com

[9] Hanafi S., Freville A., (1998). An efficient tabu search approach for the 0–1 multidimensional knapsack problem. www.sciencedirect.com

[10] Rinnooy K, A.H. G. L., Stougie, C. Vercellis (1993). A class of generalized greedy algorithms for the multi-knapsack problem. www.sciencedirect.com

[11] Balachandar R., Kannan K., (2008). A new polynomial time algorithm for 0–1 multiple knapsack problem based on dominant principles. www.sciencedirect.com

[12] Elhedhli S., (2005). Exact solution of a class of nonlinear knapsack problems.

[13] Devyaterikova, M.V., A.A. Kolokolov, A.P. Kolosov (2009). L-class enumeration algorithms for a discrete production planning problem with interval resource quantities

[14] Chen G., Maw-Sheng Chern, Jin-Hwang Jang (1990). Pipeline architectures for dynamic programming algorithms. www.sciencedirect.com

[15] Asamoah D., Baidoo E., Oppong S., Optimizing Memory using Knapsack Algorithm", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.5, May 2017, Pages 34-42.

[16] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. (1983). Optimization by simulated annealing.