# *Original Research Article*

# Meta-Heuristics Approach To Knapsack Problem In Memory Management

**Emmanuel Ofori Oppong[1], Stephen Opoku Oppong[2*], Dominic Asamoah[1], Nuku Atta Kordzo Abiew [2]**

[1] *Department of Computer Science, KNUST, Ghana*
[2] *Faculty of Computing and Information Systems, GTUC, Ghana*

**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 in optimizing computer memory. It is shown that Simulated Annealing performs better than the Genetic Algorithm for large number of processes.

## 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

44 capacity probably because the system is undersized, several options are available. This option
45 includes either upgrading the processor (if possible), reduce available functionality, or optimize.
46 A great deal of realistic problems where some predefined conditions are respected such that the sum
47 of the values of the selected entities is maximized can be represented by a set of entities, each
48 having an associated value, from which one or more subsets has to be selected. The most ordinary
49 situation is obtained by establishing that the sum of the entity sizes in each subset does not exceed
50 some prefixed bound by associating a weight/size to each entity.
51
52 The goal of this paper is to maximize the number of processes in a limited memory space.
53
54
55
## 56 2. LITERATURE REVIEW
57
58 Knapsack problems have been studied intensively in the past decade attracting both theorist and
59 practitioners. The theoretical interest arises mainly from their simple structure which both allows
60 exploitation of a number of combinational properties and permits more complex optimization problems
61 to be solved through a series of knapsack type. From a practical point of view, these problems can
62 model many industrial applications, the most classical applications being capital budgets, cargo
63 loading and cutting stock. In this section a review of literature on knapsack problems and applications
64 is presented.
65
66 The knapsack problem (KP) is a traditional combinatorial issue used to show numerous modern
67 circumstances. —Since Balas and Zemel a dozen years ago introduced the so-called core problem as
68 an efficient way of solving the Knapsack Problem, all the most successful algorithms have been
69 based on this idea. All knapsack Problems belong to the family of NP-hard problems, meaning that it
70 is very unlikely that polynomial algorithms for these problems can be devised [1].
71
72 The Knapsack problem has been concentrated on for over a century with prior work dating as far back
73 as 1897. —It is not known how the name Knapsack originated though the problem was referred to as
74 such in early work of mathematician Tobias Dantzig suggesting that the name could have existed in
75 folklore before mathematical problem has been fully defined [2].
76
77 Heuristic algorithms experienced in literature that can generally be named as population heuristics
78 include; —genetic algorithms, hybrid genetic algorithms, mimetic algorithms, scatter-search
79 algorithms and bionomic algorithms. Among these, Genetic Algorithms have risen as a dominant
80 latest search paradigm [3].
81
82 Genetic Algorithms (GA) are PC algorithms that hunt down fine solutions to a problem from among
83 countless solutions. They are versatile heuristic search algorithm in view of the evolutionary thoughts
84 of natural selection and hereditary qualities. "These computational paradigms were inspired by the
85 mechanics of natural evolution, including survival of the fittest, reproduction, and mutation. This
86 algorithm is an intelligent exploitation of random search used in optimisation problems" [4]
87
88 Bortfeldt and Gehring presented a hybrid genetic algorithm (GA) for the container packing problem
89 with boxes of unlike sizes and one container for stacking. Generated stowage plans include several
90 vertical layers each containing several boxes. Within the procedure, stowage plans were represented
91 by complex data structures closely related to the problem. To generate offspring, specific genetic
92 operators were used that are based on an integrated greedy heuristic [5]
93
94 GAs often calls for the creation and assessment of lots of dissimilar children. However, GAs are
95 capable of generating high-quality solutions to many problems within reasonable computation times.
96 [6], [7]. [8], [9]. Additionally, while performing search in large state-space or multi-modal state-space,
97 or n-dimensional surface, a genetic algorithm offers significant benefits over many other typical
98 search optimisation techniques like  linear programming, heuristic, depth-first, breath-first.
99
100 Proposed in [10], simulated annealing maintain a temperature variable to create heating process. The
101 temperature is earlier set high and after that allows to gradually "cool" as the algorithm runs. While
102 this temperature variable is high the algorithm will be permitted, with more recurrence, to accept
103 solutions that are more awful than the present solution. This gives the algorithm the capacity to hop

104 out of any local optimums it discovers itself on early on in execution. As the temperature is decreased
105 so is the possibility of tolerating more awful solution, thus permitting the algorithm gradually focusing
106 on a zone of the search space in which ideally, a near ideal solution can be found.
107
108
109 Simoes and Costa [11] performed an empirical study and evaluated the exploits of the transposition
110 A-based Genetic Algorithm (GA) and the classical GA for solving the 0/1 knapsack problem. Obtained
111 results showed that, just like in the domain of the function optimization, transposition is always
112 superior to crossover.
113
114 Eager about making use of a easy heuristic scheme (simple flip) for answering the knapsack
115 problems, [12] offered a study work on the application of usual zero-1 knapsack trouble with a single
116 limitation to determination of television ads at significant time such as prime time news, news
117 adjacencies, breaking news and peak times.
118 Martello et al [13] presented a new algorithm for the optimal solution of the 0-1 Knapsack problem,
119 which is particularly effective for large-size problems. The algorithm is based on determination of an
120 appropriate small subset of items and the solution of the corresponding "core problem": from this they
121 derived a heuristic solution for the original problem which, with high probability, can be proved to be
122 optimal. The algorithm incorporated a new method of computation of upper bounds and efficient
123 implementations of reduction procedures.
124
125 Huttler and Mastrolilli [14] addressed the classical knapsack problem and a variant in which an upper
126 bound is imposed on the number of items that can be selected. It was shown that appropriate
127 combinations of rounding techniques yield novel and more powerful ways of rounding. Moreover, they
128 presented a linear-storage polynomial time approximation scheme (PTAS) and a fully polynomial time
129 approximation scheme (FPTAS) that compute an approximate solution, of any fixed accuracy, in
130 linear time. These linear complexity bounds give a substantial improvement of the best previously
131 known polynomial bounds.
132
133 Hanafi and Freville [15] described a new approach to tabu search (TS) based on strategic oscillation
134 and surrogate constraint information that provides a balance between intensification and
135 diversification strategies. New rules needed to control the oscillation process are given for the 0 /1
136 multidimensional knapsack (0/1 MKP). Based on a portfolio of test problems from the literature, our
137 method obtains solutions whose quality is at least as good as the best solutions obtained by previous
138 methods, especially with large scale instances. These encouraging results confirm the efficiency of
139 the tunneling concept coupled with surrogate information when resource constraints are present.
140 Rinnooy et al. [16] proposed a class of generalized greedy algorithms is for the solution of the multi-
141 knapsack problem. Items are selected according to decreasing ratios of their profit and a weighted
142 sum of their requirement coefficients. The solution obtained depended on the choice of the weights. A
143 geometrical representation of the method was given and the relation to the dual of the linear
144 programming relaxation of multi-knapsack is exploited. They investigated the complexity of computing
145 a set of weights that gives the maximum greedy solution value. Finally, the heuristics were subjected
146 to both a worst-case and a probabilistic performance analysis.
147
148 Balachandar and Kannan [17] presented a heuristic to solve the 0/1 multi-constrained knapsack
149 problem (0/1 MKP) which is NP-hard. In this heuristic the dominance property of the constraints is
150 exploited to reduce the search space to find near optimal solutions of 0/1 MKP. This heuristic was
151 tested for 10 benchmark problems of sizes up to 105 and for seven classical problems of sizes up to
152 500, taken from the literature and the results were compared with optimum solutions. Space and
153 computational complexity of solving 0/1 MKP using this approach were also presented. The
154 encouraging results especially for relatively large size test problems indicate that this heuristic can
155 successfully be used for finding good solutions for highly constrained NP-hard problems.
156 Elhedhli [18] considered a class of nonlinear knapsack problems with applications in service systems
157 design and facility location problems with congestion. They provided two linearizations and their
158 respective solution approaches. The first is solved directly using a commercial solver. The second is a
159 piecewise linearization that is solved by a cutting plane method.
160
161 Devyaterikova et al. [19] presented discrete production planning problem which may be formulated as
162 the multidimensional knapsack problem is considered, while resource quantities of the problem are
163 supposed to be given as intervals. An approach for solving this problem based on using its relaxation

164 set is suggested. Some *L*-class enumeration algorithms for the problem are described. Results of
165 computational experiments were presented.
166 Chen et al. [20] presented pipeline architectures for the dynamic programming algorithms for the
167 knapsack problems. They enabled them to achieve an optimal speedup using processor arrays,
168 queues, and memory modules. The processor arrays can be regarded as pipelines where the
169 dynamic programming algorithms are implemented through pipelining.
170
171
172 **3. METHODOLOGY**
173
174 Because of their wide range of applicability, knapsack problems have known a large number of
175 variations such as: single and multiple-constrained knapsacks, knapsacks with disjunctive constraints,
176 multidimensional knapsacks, multiple choice knapsacks, single and multiple objective knapsacks,
177 integer, linear, non-linear knapsacks, deterministic and stochastic knapsacks, knapsacks with convex
178 / concave objective functions, etc.
179
180 This is a 0-1 knapsack problem, pure integer programming with single constraint which forms a very
181 important class of integer programming**.**
182 The 0-1 Knapsack Problem (KP) can be mathematically formulated through the following integer
183 linear programming [21].
184

$$\text{Maximize} \sum_{j=1}^{n} P_j x_j \tag{1}$$

185
186

$$\text{Subject to} = \sum_{j=1}^{n} \left( w_j x_j \right) \leq c \tag{2}$$

187
188     $x_j = 0$ or $1$, $j = 1, \dots, n$
189
190 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
191 of item *j*, with respect to the knapsack and C refers to the capacity, or weight-constraint of the
192 knapsack. There exist $j = 1, \dots, n$ items, and there is only one knapsack.
193
194 The use of two major meta-heuristics approaches, Genetic algorithm and Simulated annealing which
195 have been used to solve large scale problems [22] will be considered in this paper.
196
197 **3.1 Simulated Annealing**
198 Simulated annealing (SA) is a local search algorithm capable of escaping from local optima. Its case
199 of implementation, convergence properties and its capability of escaping from local optima has made
200 it a popular algorithm over the past decades. Simulated annealing is so named because of its analogy
201 to the process of physical annealing with solids in which a crystalline solid is heated and then allowed
202 to cool very slowly until it achieves stable state. i.e. its minimum lattice energy state and thus is free of
203 crystal effects. Simulated annealing mimics this type of thermodynamic behavior in searching for
204 global optima for discrete optimization problems (DOP) [23].
205
206 To formally describe simulated annealing algorithm for KP, some definitions are needed. Let $\Omega$ be the
207 solution space: define $\eta(\omega)$ to be the neighborhood function for w $\in \Omega$. Simulated annealing starts with
208 an initial solution $\omega \in \Omega$. A neighborhood solution $\omega^1 \in \eta(\omega)$ is then generated randomly in most
209 cases. Simulated annealing is based on the Metropolis acceptance criterion, which models how a
210 thermodynamic system moves from its current solution $\omega \in \Omega$ to a candidate solution $\omega i \in \eta(\omega)$ in
211 which the energy content is being minimized. The candidate solution $\omega^1$ is accepted as the current
212 solution based on the acceptance probability.
213 In this survey, finite-time implementations of simulated annealing algorithm are considered, which can
214 no longer guarantee to find an optimal solution, but may result in faster executions without losing too
215 much on the solution quality. Simulated annealing algorithm with static cooling schedule [24] for KP is
216 outlined in pseudo-code.
217
218 1   Select an initial solution $\omega = (\varkappa_1, \dots, \varkappa_n) \in \Omega$; an initial temperature t = $t_0$;
219 2   control parameter value $\alpha$; final temperature e; a repetition schedule, M that defines the number of
220     iterations executed at each temperature;

221  3  Incumbent solution ← f(ω);
222  4  Repeat;
223  5  Set repetition counter m = 0;
224  6  Repeat;
225  7  Select an integer i from the set {1,2,…,n} randomly:
226  8  If $x_i = 0$, pick up item i, i.e. set $x_i = 1$, obtain new solution ω1 then
227  9  while solution ω1 is infeasible, do
228  10 drop another item from ω randomly; denote the new solution as ω1
229  11 let $\Delta = f(ω1) − f(ω)$
230  12 while $\Delta \geq 0$ or Random (0,1) < $e^{\Delta/t}$ do ω ← ω1
231  13 Else
232  14 drop item i and pick another item randomly, get new solution ω1
233  15 let $\Delta = f(ω1) − f(ω)$
234  16 while $\Delta \geq 0$ or Random (0,1) < $e^{\Delta/t}$ do ω ← ω1
235  17 End If
236  18 If incumbent solution < $f(ω)$, Incumbent solution ← f(ω)
237  19 m = m + 1;
238  20 Until m = M
239  21 set t = a ∗ t;
240  22 Until t < $e$
241
242  A set of parameters needs to be specified that govern the convergence of the algorithm, i.e. initial
243  temperature $to$, temperature control parameter $\alpha$, final temperature $e$, and Markov chain length M, in
244  order to study the finite-time performance of simulated annealing algorithm. Here $t_o$ should be the
245  maximal difference in cost between any two neighboring solutions [24].
246
247  The parameters used for the Simulated Annealing are:
248          Cooling factor: 0.98
249          Termination Temperature: 0.2
250          Initial Temperature: 100
251          Neighbor Sampling Size: 350
252
253  **3.2 Genetic Algorithm**
254  A genetic algorithm (GA) can be described as an "intelligent" probabilistic search algorithm and is
255  based on the evolutionary process of biological organisms in nature. During the course of evolution,
256  natural populations evolve according to the principles of nature selection and "survival of the fittest."
257  Individuals who are most successful in adapting to their environment will have a better chance of
258  surviving and reproducing, while individuals who are less fit will be eliminated. This means that the
259  genes from highly fit individuals will spread to an increasing number of individuals in each successive
260  generation. The combination of good characteristics from highly adapted parents may produce even
261  more fit offspring. In this way, species evolve to become increasingly better adapted to the
262  environment [25].
263
264  A GA simulates these processes by taking an initial population of individuals and applying genetic
265  operators in each reproduction. In optimization terms, each individual in the population is encoded
266  into a string or chromosome that represents a possible solution to a given problem. The fitness of an
267  individual is evaluated with respect to a given objective function. Highly fit individuals or solutions are
268  given opportunities to reproduce by exchanging pieces of their genetic information in a crossover
269  procedure with other highly fit individuals. This produces new "offspring" solutions (i.e. children) who
270  share some characteristics taken from both parents. Mutation is often applied after crossover by
271  altering some genes in the strings. The offspring can either replace the whole population
272  (generational approach) or replace fewer fit individuals (steady-state approach). This evaluation-
273  selection-reproduction cycle is repeated until a satisfactory solution is found.
274
275  The basic steps of a simple GA are shown below [26]
276          Step 1: Generate an initial population
277
278          Step 2: Evaluate fitness of individuals in the population
279  The objective function value $(\sum_{j=1}^{n} pjXj)$ equates to how good a solution is, that is, its fitness.

280         In general, an initial population is randomly generated in some way.
281
282         Step 3: repeat
283                 a.  Select individuals from the population to be parents
284                     In the GA world for the KP, parents will be chosen by binary tournament selection.
285                     In binary tournament selection, two individuals are randomly selected from the
286                     population. From these two, the individual with the best fitness is selected to be the
287                     first parent
288                 b.  Recombine (mate) parents to produce children
289                     In the GA world for the KP, a single child will be obtained from two parents by
290                     uniform crossover. In uniform crossover each bit in the child solution is created by:
291                     repeat for each bit in turn
292                         choose one of the two parents at random
293                         set the child bit equal to the bit in the chosen parent
294                     In one-point crossover, a pint between two adjacent bits is randomly selected, "cut"
295                     the parents into two segments and create two children by rejoining the segments.
296                 c.  Mutate the children Evaluate fitness of the children
297                     Mutation corresponds to small changes that are stochastically applied to the
298                     children
299                     Mutation can be applied with a constant probability or with an adaptive probability
300                     that changes over the course of the algorithm (perhaps in response to the number
301                     of iterations that have passed or in response to population characteristics).
302                 d.  Replace some or all of the population by the children
303             until
304
305         Step 4:  you decide to stop whereupon report the best solution encountered
306
307     The parameters used for the Genetic Algorithm are:
308             Population Size: 500
309             Recombination Rate:0.7
310             Mutation Rate: 0.005
311             Number of Crossover Points: 3
312

### 3.3 Chi-Square

314     To ascertain whether the time taken and memory sued to obtain a solution is dependent or not on the
315     number of processes, the chi-square test is used. The chi-square test of independence is a statistical
316     test to determine if two or more classifications of the samples are independent or not.
317     The chi-square test is computed with the following equation [27]

$$\chi^2 = \sum_i^k \frac{(O_i - E_i)^2}{E_i} \qquad (3)$$

319     Where:
320         $O_i$ is the cell frequencies actually observed in  category i
321         $E_i$ is the cell frequencies that would be expected in category i, if the two tables were
322         statistically independent
323         k is the total number of cells or categories
324
325     Hypothesis testing is a statistical method that is used in making statistical decisions using
326     experimental data. Hypothesis tests are used to test the validity of a claim that is made about a
327     population (in this context all data under consideration). This claim made, in essence, is called the null
328     hypothesis ($H_0$) and the alternative hypothesis ($H_1$) is the one considered if the null hypothesis is
329     concluded to be untrue. For this paper, the hypothesis is to ascertain whether the time taken or the
330     memory used to obtain a solution is dependent on the number of processes. Therefore, the null
331     hypothesis ($H_0$) states that no association exists between the two variables i.e. the variables are
332     statistically independent and the alternate hypothesis ($H_1$) states that the two variables are related. In
333     performing a hypothesis test in statistics, a p-value helps determine the significance of the results.
334     The p-value can be estimated using the chi-square distribution table or using a statistical package.
335     Before the hypothesis test is performed, a threshold value is chosen, called the significance level of
336     the test and is denoted by $a$.
337

338 The p-value is a number always between 0 and 1 and interpreted in the following way:
339 • A small p-value (≤ 0.05) indicates strong evidence against the null hypothesis, so you reject
340 the null hypothesis if the significance level $a(= 0.05)$.
341 • A large p-value (> 0.05) indicates weak evidence against the null hypothesis, so you fail to
342 reject the null hypothesis if the significance level $a(= 0.05)$.
343 R (a statistical package) is used to calculate the chi-square value and p-value using this pseudocode
344     x<=matrix(data)
345     View(x)
346     Chisq.test(x)
347
348 **4. ANALYSIS AND RESULTS**
349
350 Category A: The computer system with a total of 10 created processes, all with their system
351 information in figures. The computer memory can accommodate capacity of 50mb but the total
352 memory of the process is 56 with a combined process activity (number of times process is accessed
353 of 123
354
355

**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 |

356
357 From Table 1, it could be seen that all three algorithms provide the same output in terms of all the
358 parameters under consideration. This means that both DP, GA and SA
359
360 Category B: The table below shows a computer system with a total of 50 created processes, all with
361 their system information in figures. The computer memory can accommodate capacity of 100mb. but
362 the total memory of the process is 281 with a combined process activity (number of times process is
363 accessed of 483
364
365

**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 |

366
367 From Table 2, GA provided a slight advantage of in terms of the number of process used. Apart from
368 that all three algorithms provided fairly the same result
369
370 Category C: The table below shows a computer system with a total of 100 created processes, all with
371 their system information in figures. The computer memory can accommodate capacity of 300mb. but
372 the total memory of the process is 574 with a combined process activity (number of times process is
373 accessed of 1011
374
375

**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 |

376
377 Table 3 shows that DP provides a better result than the rest. All memory needed was utilized showing
378 efficient use of memory available.

379
380 Category D: The table below shows a computer system with a total of 500 created processes, all with
381 their system information in figures. The computer memory can accommodate capacity of 1000mb. but
382 the total memory of the process is 2661 with a combined process activity (number of times process is
383 accessed of  5287
384
385

**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 |

386
387 Category E: The table below shows a computer system with a total of 1000 created processes, all
388 with their system information in figures. The computer memory can accommodate capacity of
389 5000mb. but the total memory of the process is 5626 with a combined process activity (number of
390 times process is accessed of 10480).
391
392

**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 |

393
394 GA and SA provide fairly the same results in Table 4 and 5.
395
396 The main criteria in evaluating the efficiency of an algorithm is time and space. Even though in terms
397 of results the three algorithms provided similar results, their efficiency will be determined based on the
398 time it took to produce the results and the amount of memory resource it took on the computer.
399
400

**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 |

401
402

**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

Using the chi-square test on Table 6, the null and alternate hypothesis are defined as follows
$H_0$: Time taken to obtain a solution is independent of number of processes.
$H_1$: Time taken to obtain a solution is not independent of number of processes.

The chi-square statistic ($\chi^2$)= 18.7547.
The p-value is 0.000878.

Since the p-value of 0.000878 is less than the significance level of 0.05, the null hypothesis($H_0$) which stated that the time taken to obtain a solution is independent of number of processes is rejected. This implies that number of processes is dependent on the time taken to obtain a solution

**Table 7: Results for based on Memory Taken**

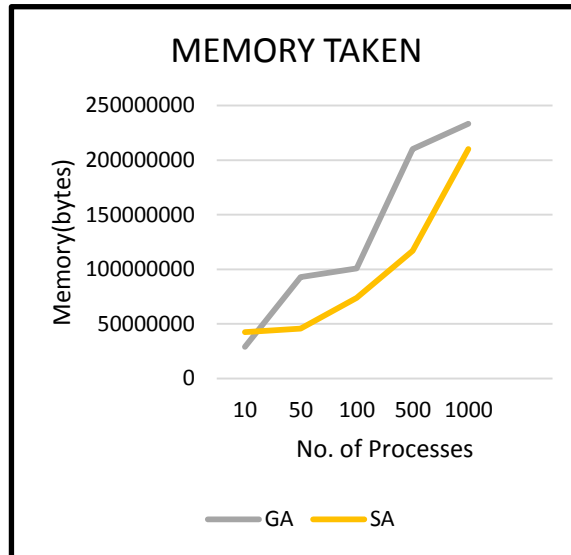| MEMORY (byte) | | |
|---|---|---|
| **No. of Process** | **GA** | **SA** |
| 10 | 28880312 | 42511800 |
| 50 | 92815928 | 45555312 |
| 100 | 100774992 | 73927720 |
| 500 | 210273904 | 117057112 |
| 1000 | 233449048 | 210256440 |

**Figure 2: Results for based on Memory Taken**

Using the chi-square test on Table 7, the null and alternate hypothesis are also defined as follows

$H_0$: Memory used in obtaining a solution is independent of number of processes.

$H_1$: Memory used in obtaining a solution is not independent of number of processes.

The chi-square statistic $(\chi^2)$= 22.8798

<mark>The p-value is 0.000134.</mark>

Since the p-value of 0.000134 is less than the significance level of 0.05, the null hypothesis is also rejected here. This implies that memory used to obtain a solution is dependent on the number of processes.

## 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. From Figure 2, it can be seen that with increase in number of processes, experiments with simulated annealing gives better result than the Genetic Algorithm in terms of both time-taken to obtain a solution and memory taken. 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.

Notwithstanding it extensive use, both SA and GA have their limitations. For SA, If the starting temperature is very high, the search will be a random local search for a period of time i.e. accepting all neighbors during the initial phase of the algorithm. Also, In the SA algorithm, the temperature is decreased gradually. If the temperature is decreased slowly, better solutions are obtained but with a more significant computation time. For GA, if reproduction fails to produce good chromosomes then convergence in the right direction is not possible.

467
## REFERENCES
469

470    [1] Pisinger, D. Core problems in knapsack algorithms. Operations Research. 1999;47(4):570-575.

471    [2] Kellerer H, Pferschy U, Pisinger D. Knapsack Problems. Springer, Berlin Heidelberg. 2004.

472    [3] Chu PC, Beasley JE. A genetic algorithm for multidimensional knapsack problem. Journal of
473        Heuristics. 1998;4(4):63-68.

474    [4] Sinapova L. An Introduction to Algorithms. Simpson College, Department of Computer Science
475        701 North C Street, Indianola IA 50125. 2014.

476    [5] Bortfeldt A, Gehring H. A hybrid genetic algorithm for the container loading problem [J].
477        European Journal of Operational Research. 2001;131(1):143-161.

478    [6] Beasley JE, and Chu PC. A genetic Algorithm for the set covering problem. European Journal of
479        Operations Research.1996; 94(2):392-404

480    [7] Chu PC, Beasley JE. A genetic algorithm for generalized assignment problem. Computer and
481        Operations Research 1997; 24(1):17-23

482    [8] Chu PC, Beasley JE. A genetic algorithm for multidimensional knapsack problem. Journal of
483        Heuristics. 1998;4(1):63-86

484    [9] Chang JT, Meade N, Beasley JE, Sharaiha YM. Heuristics for cardinality constrained portfolio
485        optimization. Computer and Operations Research. 1999; 27(2000):1271-1302

486    [10] Kirkpatrick S, Gelett CD, Vecchi MP. Optimization by simulated annealing. Science. 1983; 621-
487        630.

488    [11] Simoes A, Costa E. Using Genetic Algorithm with Asexual Transposition. Proceedings of the
489        genetic and evolutional computation conference 2001; 323-330.

490    [12] Amponsah SK., Oppong EO, Agyeman E, Optimal Television Adverts Selection, Case Study:
491        Ghana Television (GTV). Research Journal of Information Technology, 2011; 3(1):49-54.

492    [13] Martello S, Pisinger D, Toth P. New trends in exact algorithms for the 0–1 knapsack problem.
493        European Journal of Operations Research. 2000;123:325-332

494    [14] Mastrolilli M, Huttler M. Hybrid rounding techniques for knapsack problems.Journal of discrete
495        applied mathematics. 2006;154(4):640-649

496    [15] Hanafi S, Freville A. An efficient tabu search approach for the 0–1 multidimensional knapsack
497        problem. European Journal of Operations Research. 1998;106(2-3):659-675.

498    [16] Rinnooy K, A.H. G. L., Stougie, C. Vercellis  A class of generalized greedy algorithms for the
499        multi-knapsack problem.European Journal of Operation Research.1993.

500    [17] Balachandar R, Kannan K. A new polynomial time algorithm for 0–1 multiple knapsack problem
501        based on dominant principles. www.sciencedirect.com 2008.

502    [18] Elhedhli S, Exact solution of a class of nonlinear knapsack problems. Operations Research
503        Letters. 2005; 33(6): 615-624.

504    [19] Devyaterikova MV, Kolokolov AA, Kolosov AP,  L-class enumeration algorithms for a discrete
505        production planning problem with interval resource quantities. Computers & Operations
506        Research. 2009; 36(2):316-324

507    [20] Chen G., Maw-Sheng Chern, Jin-Hwang Jang. Pipeline architectures for dynamic programming
508        algorithms. www.sciencedirect.com   1990.

509    [21] Hisatoshi S. A Generalized Knapsack Problem with Variable Coefficients, Mathematical
510        Programming. North-Holland Publishing Company. 1975;162-176.

511    [22] Asamoah D, Baidoo E, Oppong S, Optimizing Memory using Knapsack Algorithm, International
512        Journal of Modern Education and Computer Science (IJMECS). 2017;9(5):34-42.

513    [23] Fubin Q, Rui D. Simulated Annealing for the 0/1 Multidimensional Knapsack Problem, Numerical
514        Mathematics. A Journal of Chinese Universities. 2007;16(4):320-327

515    [24] Oppong OE. Optimal resource Allocation Using Knapsack Problems: A case Study of Television
516        Advertisements at GTV. Master's Thesis, KNUST. 2009.

517    [25] Djannaty F, Doostdar S. A Hybrid Genetic Algorithm for the Multidimensional Knapsack Problem,
518        International Journal of Contemp. Math. Sciences. 2008;3(9):443 – 456

519    [26] Carr J. An Introduction to Genetic Algorithms. 2014.

520    [27] McHugh ML. The chi-square test of independence. Biochemia medica. 2013; 23(2):143-149