EasyChair Preprint

№ 5511

0-1 Knapsack Problem Using Genetic Algorithm

Aashutosh Bansal, Himanshu Gadia, Sanivarapu Dhanusha and Anil Pandey

May 13, 2021

Solving 0-1 Knapsack Problem using Genetic Algorithm

Aashutosh Bansal[1], Himanshu Gadia[1], S. Dhanusha[1], Anil Pandey[1]

[1]*Department of Information Technology, National Institute of Technology Raipur India*

***Abstract*- This paper alludes to taking care of 0-1 knapsack issue utilizing genetic algorithms (GA). Knapsack algorithm is a NP (Non-deterministic Polynomial) issue. By utilizing GA streamlining is performed. This paper contains three areas. In the principal area a short portrayal of GAs and a portion of its nuts and bolts. In the following segment execution of the knapsack issue is finished by utilizing genetic algorithms. The essential motivation behind this examination paper is to carry out 0-1 knapsack issues dependent on genetic algorithms. In the forthcoming piece of paper fitness function is utilized to decide the fitness and incentive for the succeeding populaces with the simple desire to track down the most right arrangement and the results were noticed.**

***Keywords*- Genetic Algorithm, knapsack problem, Selection operators, nondeterministic polynomial**

I. INTRODUCTION

A bunch of things is given with each having its own value and weight. This algorithm decides the quantity of everything we can incorporate with the goal that the absolute weight should be not exactly or equivalent to the limit of knapsack and augmentation of all out value. As this technique is 0-1 knapsack so dissimilar to partial knapsack no things will come in division. Either a thing will incorporate or not. That is the reason the name 0-1 knapsack.

This paper utilizes the genetic algorithm for addressing 0-1 knapsack algorithm to expand the absolute value and lessening the all-out weight so it should be not exactly the limit of knapsack. Anyway Knapsack issue is a contributor to NP issue which alludes to time complexity since more the absolute amount of things will be the time taken. As we probably are aware this issue can't be tackled in direct time anyway the arrangement of this issue can be confirmed in straight time. The ordinarily utilized methodologies utilized for tackling these issues are Dynamic Programming (DP), Greedy technique and so forth however as we probably are aware they are not productive. The Dynamic methodology has a complexity of O (n^3) while the Greedy strategy doesn't meet an ideal arrangement.

So by using genetic algorithms may have an edge over these above mentioned traditional topics and may lead to more efficient and optimum solutions.

II. LITERATURE REVIEW

. S. No	Title of the paper	Publisher	Methods used	Result
1	Arish Pitchai et al., 2015	IEEE	Quantum walk genetic algorithm(QWGA), Greedy genetic algorithm (GGA)	QWGA is better than GGA.
2	Ravneil Nand et al., 2019	IEEE	Firefly Algorithm(FA) + Genetic Algorithm(GA)	The FAGA model worked quite well on multidimensional knapsack problems.
3	Indresh Kumar Gupta, 2018	IEEE	Genetic algorithm(GA) , gravitational search algorithm(GSA)	Hybrid model of GSA-GA is used for solving multidimensional knapsack problems.
4	Mojtaba Montazeri et al., 2017	IEEE	GA + restart genetic algorithm(RGA)	Proposed GA has enhanced speed and performance.
5	Vikas Thada et al., 2014		Stochastic Uniform, Remainder, Roulette, Tournament, Uniform Selections	Roulette Selection has least performance.
6	Rattan Preet Singh et al., 2011	IEEE	Stochastic, Roulette wheel, Tournament, Selections	Solved knapsack problem in linear amount of time.
7	Tribikram Pradhan et al., 2014	IEEE	Genetic Algorithm, Rough Set Theory Hybrid Algorithm	The hybrid rough set theory has better performance than GA.
8	Abdellah Rezoug et al., 2017	IEEE	Genetic algorithm guided by Pretreatment information(GAGP)	GAGP is an Improved model for GA.
9	Frumen Olivas et al., 2020	IEEE	Fuzzy- based selection hyper-heuristic approach	Proposed method obtained better result than low-level and traditional selection hyper-heuristics.

III. THEORY

A. The Basic Structure of a Genetic Algorithm

1) **[Start]** Generate random population of n chromosomes (suitable solutions for the problem

2) **[Fitness]** Evaluate the fitness f(x) of each chromosome x in the population

3) **[Test]** If the end condition is satisfied, stop and return the population. If not, generate a new population.

4) **[New population]** Create a new population by repeating following steps until the new population is complete:

i. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

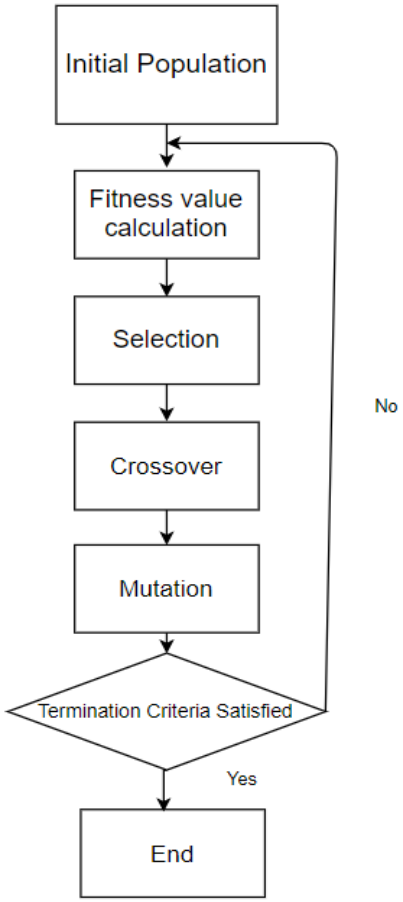
ii. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

iii. **[Mutation]** With a mutation probability mutates new offspring at each locus (position in chromosome).

iv. **[Accepting]** Place new offspring in a new population

5) **[Replace]** Use new generated population for a further run of algorithm

6) **[Loop]** Go to the Fitness step



B. Chromosomes

Chromosomes express an expected testament through an encoding like double, values, or trees. For the 0-1 knapsack issue, we will utilize a double exhibit to communicate which things of the knapsack will be added (1's) and which will be excluded (0's).

1) Start

We produce an underlying population by making irregular varieties of 1's and 0's. The population size is determined by the client. A bigger population size will hinder the algorithm however take into account more genetic variety. Bigger populations, nonetheless, are typically quicker at joining to an answer and less helpless to neighborhood maximums. We picked a population size of 100 chromosomes after some experimentation [1]. As you will find later, there is no basically "right" arrangement of variables, for instance, population size, mutation rate, or most prominent generations, just certain blends that achieve better results.

2) Fitness

This movement evaluates how "fit" each chromosome is tantamount to the specific issue. For the 0-1 knapsack, the chromosomes are each situated by the total weight of the knapsack. If the weight of a specific chromosome is higher than the most limited weight, a discretionary 1-value is changed and the chromosome is rethought until it doesn't outperform the best. The fitness of each chromosome in the population is taken care of in a show. An intriguing note is that the fitness function abuses the NP-complete-ness of the knapsack issue, that is, a confirmation is checkable in polynomial time [1].

3) New Population

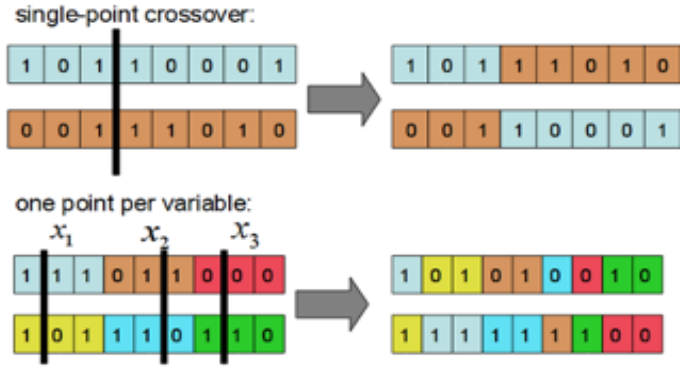
Another population (in like manner called a generation) is made by "mating" the fittest chromosomes of the population.

i. Selection

The chromosomes are picked such that those with a higher fitness will undoubtedly be picked as mates. Such a selection that we did is called roulette-wheel selection, anyway various techniques for selection like position selection, pack selection, and predictable state selection exist. Roulette-wheel selection picks a self-assertive number among 0 and the measure of the finesses. By then, underscoring through the fitness show, we deduct each individual fitness from our sporadic number until the number is more critical than or identical to nothing. Whatever chromosome we stop on is picked to mate. Not at all like nature, we can use a system called elitism, and copy the fittest chromosome thus from the population to the new generation. Elitism is a crucial practice to ensure that we join in on an answer.

ii. Crossover

The crossover is analogous to biological “mating” of species. Our virtual chromosomes, however, are not double helices but a single strand of binary. The crossover function takes two “parents” and creates a new chromosome by splicing parts of one with parts of another. These are two examples of crossovers.

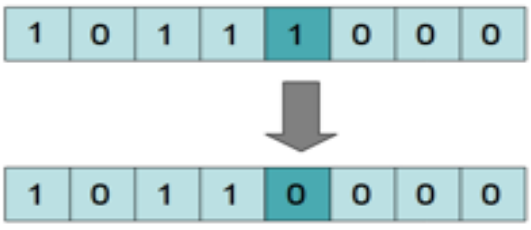


We utilized single point crossover for our execution. For each set of guardians an arbitrary rotate point is picked and the pieces up to that turn point are traded, making two new kid chromosomes. We additionally decided to do 100% crossover for straightforwardness (beside the world class chromosome). Some genetic algorithms with 85% crossover, for example, will copy 15% of the old population to the new population [1].

iii. **Mutation**

The mutation function, like self-assertive mutations in nature, keeps our populations genetically various. In our algorithm, mutations keep our movement of generations away from meeting at a neighborhood most noteworthy. The mutation function goes through the entirety of the new chromosomes and learns whether to flip the piece subject to a little probability.

We picked a 1% probability for mutation reliant upon experimentation. Having too immense a mutation probability will cause the generations to be genetically unstable, and it will take more effort to join on an answer. (Recall that 100% mutation is essentially making an unpredictable chromosome.) Too low of a mutation rate will keep the population got in the genetic possibilities of the first [2].

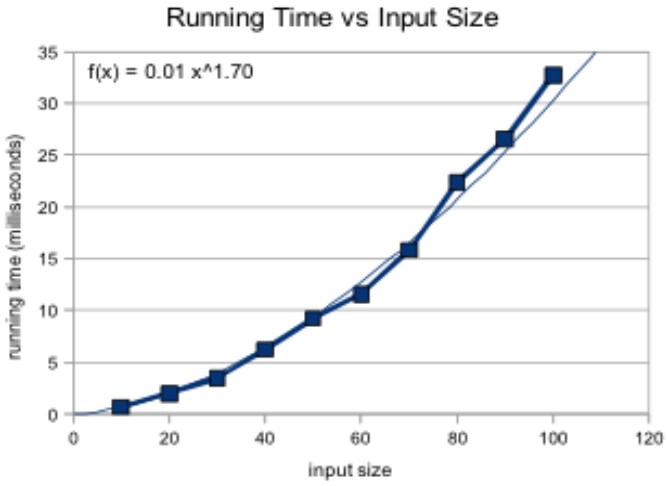


4) **Test & Loop**

In the wake of being created, the new population replaces the old population. The fitness of the population is determined at that point and tries to check whether an end condition is fulfilled. In our algorithm we tried for a combination factor of 90% or a most extreme number of generations came to. We found that extending the most extreme number of generations relating to different occasions the data size helped scale the algorithm to greater information sources [1]. If the end condition isn't reached, the algorithm continues surrounding, creating another population and testing again.

IV. COMPLEXITY

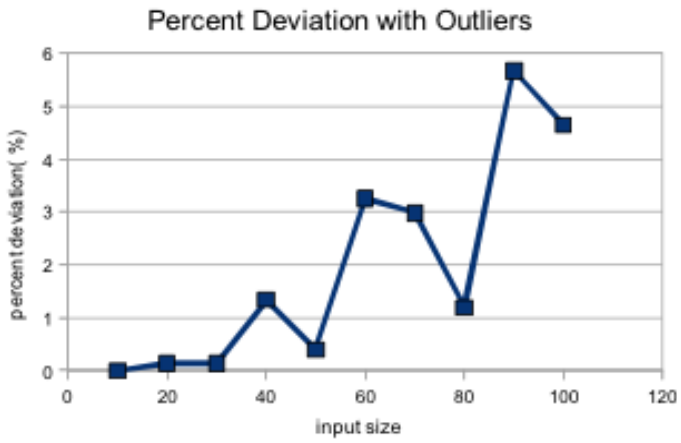
Since we decided the best number of generations, the complexity of our program has a polynomial O(n) upper bound. Making the fundamental population takes O(n). Both the crossover and mutation functions underline over each chromosome once, taking O(n) time copied by the consistency of the population size. In this way the circle takes O(n) time. This circle will run a limit of 10*n occasions, making the all-out complexity of our algorithm $O(n^2)$. This lines up with our experimentation on the running time. We ran 10 preliminaries, with n = 10, 20, …,100. Every preliminary comprised 10 arrangements of haphazardly created contributions of size n, and information was then tried multiple times. A normal of the running time was assuming control over the 10 reiterations of the 10 sets for every preliminary. The normal deviation and percent deviation of the arrangements was additionally determined for each set.



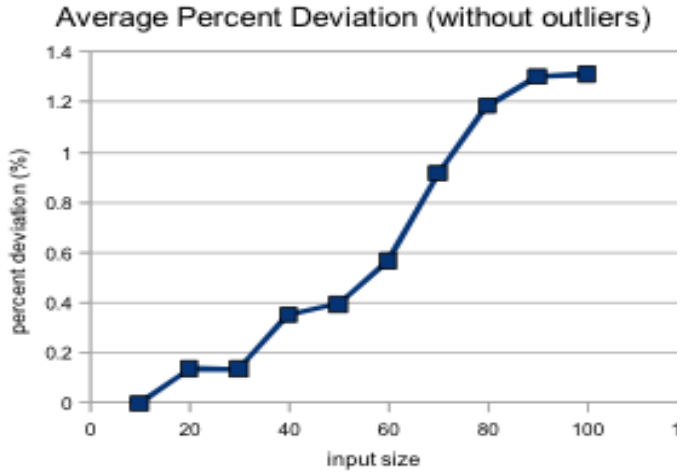
As should be obvious, the running time shows a force bend. The best fit force bend ascertains a rough real running season of $n^{1.7}$ which is inside our $O(n^2)$ bound.

Degree of Approximation

Since our genetic 0-1 knapsack algorithm is an assessment, something vital for note is the ordinary deviation of the made plans at different data sizes.



We found that at whatever point the greatest weight for the knapsack was minuscule (with an answer of generally 0's) the normal deviation could be up to 42%. This is probably because of the manner in which all chromosomes are adjusted so they fit under the most extreme weight before the new population is produced. One possible approach to handle this later on is to consider chromosomes with above-most extreme weights to exist in the population, however with simply a higher fitness punishment for surpassing the breaking point. In the wake of eliminating the conspicuous anomalies, our percent deviation is as per the following:



As should be obvious, the percent deviation of the appropriate responses increments with the information size. A potential method to battle this is to raise the greatest number of generations, albeit this will expand the running season of the algorithm.

V. CONCLUSIONS

Sometimes we needn't mess with the best game plan yet a near best-course of action. Various NP-complete issues, similar to 0-1 knapsack issues, can be approximated by imitating the computational power of advancement in the normal world. Genetic algorithms themselves are really adaptable to the prerequisites of the customer, with various elements, for instance, population size, association factor, most limited generations, mutation rate, and crossover extent that can be adjusted for better results. There are in like manner different strategies for executing the essential development of the algorithms, for instance, the chromosome encoding or selection factor. Further examinations could attempt to explore the different mixes of such choices on the 0-1 knapsack or other NP-complete issues.

REFERENCES

1. "Solving the 0-1 Knapsack Problem with Genetic Algorithms" by Maya Hristakev and Dipti Shresth (http://www.sc.ehu.es/ccwbayes/docencia/kzmm/files/AG-knapsack.pdf)

2. A. Pitchai, A. V. Reddy and N. Savarimuthu, "Quantum Walk based genetic algorithm for 0–1 quadratic knapsack problem," 2015 International Conference on Computing and Network Communications (CoCoNet), Trivandrum, India, 2015, pp. 283-287, doi: 10.1109/CoCoNet.2015.7411199.

3. R. Nand and P. Sharma, "Iteration split with Firefly Algorithm and Genetic Algorithm to Solve Multidimensional Knapsack Problems," *2019 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, Melbourne, VIC, Australia, 2019, pp. 1-7, doi: 10.1109/CSDE48274.2019.9162422.

4. I. K. Gupta, "A hybrid GA-GSA algorithm to solve multidimensional knapsack problem," *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, Dhanbad, India, 2018, pp. 1-6, doi: 10.1109/RAIT.2018.8389069.

5. Lu, Fan & Liu, Dong & Liu, Yang & Li, Zhenwei & Jiang, Qilong & Chen, Yuanlong. (2020). A Study on Low-Temperature Model Parameter Identification of LTO Battery by Cuckoo Search. 490-494. 10.1109/ICIEA48937.2020.9248168.

6. V Thada, S Dhaka - International Journal of Computer Applications 2014 (https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.679.466&rep=rep1&type=pdf)

7. R. P. Singh, "Solving 0–1 Knapsack problem using Genetic Algorithms," *2011 IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, China, 2011, pp. 591-595, doi: 10.1109/ICCSN.2011.6013975.

8. T. Pradhan, A. Israni and M. Sharma, "Solving the 0–1 Knapsack problem using Genetic Algorithm and Rough Set Theory," *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, Ramanathapuram, India, 2014, pp. 1120-1125, doi: 10.1109/ICACCCT.2014.7019272.

9. A. Rezoug, M. Bader-El-Den and D. Boughaci, "Knowledge-based Genetic Algorithm for the 0–1 Multidimensional Knapsack Problem," *2017 IEEE Congress on Evolutionary Computation (CEC)*, Donostia, Spain, 2017, pp. 2030-2037, doi: 10.1109/CEC.2017.7969550.

10. Olivas, Frumen & Amaya, Ivan & Ortiz-Bayliss, José Carlos & Conant-Pablos, Santiago & Terashima-Marín, Hugo. (2021). Enhancing Hyper-heuristics for the Knapsack Problem through Fuzzy Logic. Computational Intelligence and Neuroscience. 2021. 1-17. 10.1155/2021/8834324.