WARSAW UNIVERSITY OF TECHNOLOGY
Faculty of Mathematics
and Information Science

Artificial Intelligence Fundamentals
Project of an Evolutionary Algorithm for Solving the Subset Sum Problem

Mikołaj Jędrzejewski
Mateusz Małkiewicz
Piotr Syrokomski

Computer Science and Information Systems
06.03.2024

History of Changes			
Version	Date	Who	Description
01	2024-03-06	Piotr	Description of the problem
02	2024-03-20	Mikołaj, Piotr	An analysis of the problem
03	2024-04-27	Mateusz	Description of the preferred solution
04	2024-05-12	Mikołaj	Add positive tests
05	2024-05-23	Mikołaj	Application implemented
06	2024-05-23	Mikołaj	AI implementation
07	2024-05-30	Mateusz	Tests
08	2024-05-30	Mikołaj	Description of AI implementation
09	2024-05-30	Mikołaj	Description of chapter 6
10	2024-05-30	Mateusz	Chapter 7 description
11	2024-05-30	Mateusz	Presentation

Responsibilities (planned)	
Name	Task
Mikołaj	Chapter 2 description, research of existing solutions, GUI, Chapter 6 description
Mateusz	research of existing solutions, presentation, generation of tests, description of Chapter 4
Piotr	first chapter description, research of existing solutions and partial third chapter description, AI implementation, Chapter 7 description

Table of Contents

1.Description of the problem

In the subset sum problem (SSP), we are given:

- a multiset S of integers
- a target sum T

The task is to decide, whether there exists a subset $S' \in S$, such that the sum of all the elements in S' is equal to T. A calculation of such a subset for verification purposes may be useful.

The problem is NP-hard. Moreover, there are some variations which are also NP-Complete, such as:

- when all the numbers in S are positive
- $T = 0$
- the partition problem: is it possible to split the set S into two sets with the same sum

To better illustrate this, let us analyse the following example:
$S = \{-17, -9, 3, 7, 10, 15, 18, 20, 40\}$
$T = 22$
Then, there a multiple subsets of S whose sum of all elements is equal to T,
$S'_1 = \{15, 7\}$
$S'_2 = \{-17, 18, -9, 10, 20\}$
$S'_3 = \{-9, 10, 3, 18\}$
The above example is represented graphically in Figure 1.

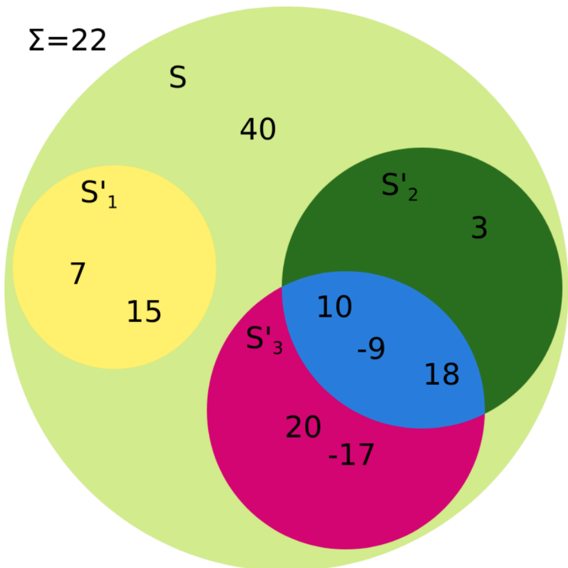

Figure 1. An example of graphical representation of SSP with solutions of subsets whose elements sum up to T

2.An analysis of the problem

The input of an algorithm is a multiset S of integers and a target sum T. Our program should output one bit of information - 1 in case there exists a subset of S that sums up to T, 0 otherwise, it is illustrated by Figure 2.

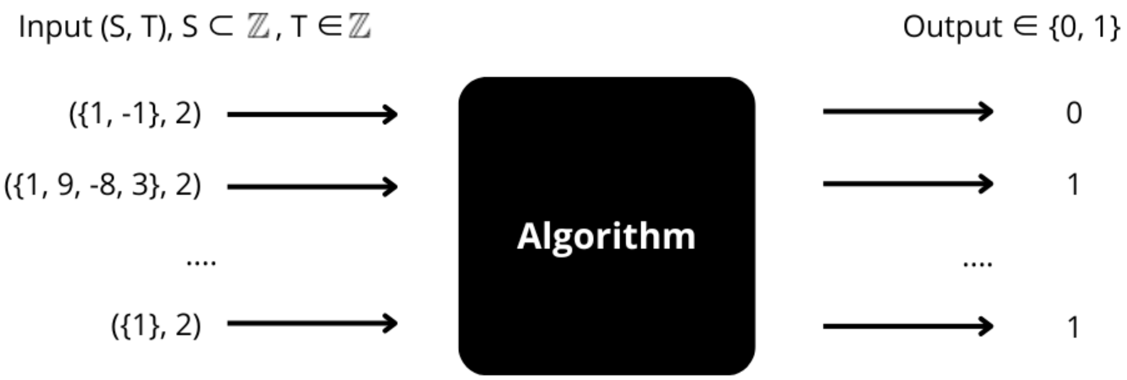

Figure 2. Specification of inputs and outputs used in an algorithm solving SSP

State of the art algorithm for the Subset Sum Problem runs in $O(2^{0.291N})$ time complexity. Because of this, for large input sets, solving the problem is infeasible. Instead, we will try to develop an algorithm that sacrifices correctness of the solution for the run time. As can be seen in Figure 2, the last example does not provide a correct answer, however, it is enough that the algorithm works a good fraction of all the runs. Thus, we should consider how our algorithm can be assessed and then compared with other approaches.

Our problem concerns a binary classification, thus we can distinguish 4 possible classes of outputs, namely: True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN).

Let us observe that for a given set S, there can be only a finite number of values T can take such that there exists a subset of S whose elements sum up to T. However, T can be arbitrarily large which makes negative cases more likely. In general, we should incorporate that idea into scoring to eliminate approaches that would take advantage of this imbalance. We propose to assess an algorithm based on the score metric, given by the formula:

$$score_n = \frac{1}{max(1, time)} \cdot \sum_{i \in \{TP, FP, TN, FN\}} \alpha_i \cdot \#i \qquad (1)$$

We will run an algorithm for some constant number of cases z and the input set of cardinality |S| = n and compute the score based on the number of True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN) and their corresponding alpha coefficients, values of which can be found in Table 1. Time is given by the cumulative time elapsed during execution of all z test cases.

	Negative	Positive
True	2	1
False	0	0

Table 1. Values of coefficients α_i used in determining the score

Now we are able to analyse when our algorithm performs better than other approaches as n changes.

It is worth mentioning that the output of an algorithm solving SSP might be a binary sequence $\{a_i\}$ which describes which elements of S should be used in the sum. If the elements from the multiset S are ordered then we can denote i-th element by w_i. Then, the sum of $w_i \cdot a_i$ over all elements can be compared to T. If it is equal to T then it is the TP case and we have certainty that our algorithm found the correct solution. Otherwise, when the sum is not equal to T then we would need to decide if to output 0 or 1, this is the case of TN, FN or FP. Additionally, in case of the output being a binary sequence we could modify the scoring metric to reward sums which use less elements.

To test our solutions we will generate test cases on our own. In the case of positive cases the approach is straightforward, since once we have randomised the input set the target sum can be computed as a sum of randomly selected subset of the input set. Generating random negative cases for large cardinalities of the input set NP-hard on its own as it requires solving SSP. However, this problem might be overcome with the use of properties of T and elements of S, for example T is odd but S contains only even numbers.

3.Existing solutions

1. Genetic algorithm
   A Genetic Algorithm (GA) is an optimization method inspired by natural selection. It starts by randomly initialising a population of potential solutions. Each solution, represented as an individual or chromosome, is evaluated using a fitness function. Selection operators choose individuals based on their fitness to form the next generation. Through crossover, genetic material is exchanged between selected individuals to create offspring, introducing new genetic combinations.

2. Dynamic programming - pseudo-polynomial time solution

   It works by breaking down the main problem into smaller subproblems, solving each subproblem only once, and reusing the solutions to these subproblems to solve larger ones. It does not take the target sum T as a parameter, which could be useful or a waste of time. It does not work well when the absolute values of elements are large, because of the requirement for large amounts of memory

	s\t	0	1	2	3	4	5	6	7	8	9	10	11	12
S[0]	2	1	0	1	0	0	0	0	0	0	0	0	0	0
S[1]	3	1	0	1	1	0	1	0	0	0	0	0	0	0
S[2]	5	1	0	1	1	0	2	0	1	1	0	0	0	0
S[3]	7	1	0	1	1	0	2	0	2	1	1	2	0	2
S[4]	9	1	0	1	1	0	2	0	2	1	2	2	1	3

   Figure 3. An example table used in the dynamic programming approach.

3. Fully-polynomial time approximation schemes.
   An approximation algorithm to SSP aims to find a subset of S with a sum between rT and T, where r is a number in (0,1) called the approximation ratio. The fully polynomial time approximation scheme, for any $\epsilon > 0$ attains the approximation ratio of $1 - \epsilon$. It's both polynomial in n and $\frac{1}{\epsilon}$.

4. Naive approach.
   We can iterate over all the subsets in the power set of S and calculate the sum of their elements. If N is the number of elements in S, 2^N is the number of all its subsets, since each element of S can either be or be not included in any subset. While this method is extremely slow, its one strength is the simplicity with which it can be implemented.

4. My preferred solution

State of the art algorithms for the Subset Sum Problem run with $O(2^{0.291N})$ time complexity. However, as inputs grow in size, solution of the problem becomes increasingly impractical to compute. In such scenarios, the focus should shift towards seeking "approximate" solutions, which offer faster computation times. This is a field where artificial intelligence, particularly Genetic Programming (GP), emerges as a potential helper.

A similar approach was already explored in [1] where a modified genetic algorithm was used to solve the Subset Sum Problem. It appears that it was capable of finding an optimal solution and got 71% accuracy. Moreover, as the author says, this approach is problem independent and can be used to solve other combinatorial optimization problems.

Additionally, we would like to compare this approach with a naive one and a fully-polynomial time approximation scheme (FPTAS). This FPTAS solution provides a feasible alternative that allows for rapid computation while maintaining an acceptable level of accuracy. Evaluating the effectiveness of our obtained solution involves comprehensive comparison with existing algorithms on the basis of such factors as correctness, as well as space and time complexities. Additionally, the flexibility of AI enables the incorporation of custom constraints, such as incentivizing shorter solutions, further tailoring the algorithm to specific needs.

After obtaining the approximate solution, it's crucial to conduct thorough analysis. This step not only validates the solution's effectiveness but also holds the potential to deepen our understanding of the problem. Essentially, this analysis may pave the way for groundbreaking advancements in our understanding and approach to solving the Subset Sum Problem.

Our algorithm will be implemented using the Python programming language. The main advantage of this language is that it is one of most preferred tools for Genetic Programming, thanks to its support for various libraries for GP, such as Pyvolution, DEAP, pySTEP, LEAP and PyGAD. For this application, we're going to use the DEAP library due to it being one of the lead libraries used for Genetic Programming.

Another advantage of Python is that it enables creating GUI, thanks to such libraries as PyQT, Tkinter and wxPython. Another thing that influenced our choice was our experience with the language. Our knowledge of Python is deep enough to create our application efficiently.

Additionally, as Python is one of the easiest programming languages to learn, we can quickly expand our knowledge even more, in case we come across a concept that is unknown to us.

5. Implementation of the AI part

The implementation of the Genetic Algorithm (GA) for the Subset Sum Problem (SSP) utilises the DEAP library, a powerful tool for evolutionary computation.

DISTRIBUTED EVOLUTIONARY ALGORITHMS IN PYTHON

Figure 3. DEAP library

We begin with the initialization of a population, where each individual is represented as a binary sequence indicating whether the corresponding element from the set S should be included in the subset sum to achieve the target sum t. The fitness function is then defined, evaluating each individual's fitness as the absolute difference between the sum of its selected elements and the target sum t. The objective is to minimise this difference, guiding the algorithm towards finding the subset that best satisfies the target sum condition. To enable the evolution of solutions, generic genetic operators are implemented. These include crossover (mate), where genetic material from selected individuals is exchanged to create offspring with new combinations of elements. Mutation (mutate) introduces diversity by randomly flipping bits in individual chromosomes, allowing for exploration of alternative solutions. Additionally, selection (select) operators are defined to determine which individuals proceed to the next generation based on their fitness. In this implementation, a tournament selection strategy is employed, selecting a fixed number of individuals (3 in this case) from each tournament. Moreover, the mutation probability is set to 0.05, indicating the likelihood of a random bit flip occurring in an individual during the mutation process. These parameters and operators collectively drive the evolutionary process, iteratively improving the population over multiple generations to converge towards an optimal or near-optimal solution for the Subset Sum Problem. Through the iterative application of these steps, the Genetic Algorithm efficiently explores the solution space, progressively refining candidate solutions until satisfactory results are achieved.

6.Simulation results from the application

The application allows for testing and comparing up to three solutions solving the SSP. We successfully performed simulations for multiple datasets, which included only positive cases, negative or mixed. The app updates in real time the progress with graphs of accuracy, execution time and score from formula (1). Results are presented on Figure 4 and 5.

Figure 4. Graph of execution times for all 3 approaches (naive, FPTAS, GP)

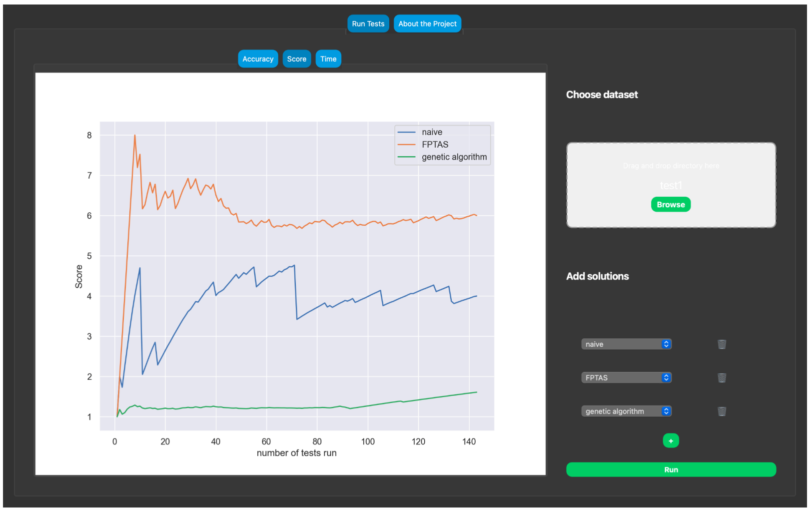

Figure 5. Graph of the score for all 3 approaches (navie, FPTAS, GP)

6. Conclusions

We have successfully implemented a project which consists of an GUI application that allows for comparing solutions for solving the SSP. From the test that we have conducted based on generated test cases we observed that the genetic algorithm performed well with 90%> accuracy. It is a higher value than the one obtained in [1]. However, more tests would be preferable, including one just greater sizes of the input set but also ensuring that there are few or only one subset that sums up to the target sum. In the latter, algorithms would have less luck and would give more inaccurate answers and in case of the naive approach run much longer. From the very limited test we can already observe the naive approach execution times rise quickly. FTPAS excels in terms of accuracy and speed. Thus any additional test should only include GP and FTPAS. Moreover, we should acknowledge the amazing tool ChatGPT is. It could be used to explore new techniques for solving the SSP and developing new insights. Furthermore, we could split the problem into smaller ones, maybe inquire ChatGPT about some subsets which we might find hard to analyse.

Literature

[1] Rong Long Wang, A genetic algorithm for subset sum problem, Neurocomputing, Volume 57, 2004, Pages 463-468, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2003.12.003.
[2] Saketh, K.H., Jeyakumar, G. (2020). Comparison of Dynamic Programming and Genetic Algorithm Approaches for Solving Subset Sum Problems. In: Smys, S., Tavares, J., Balas, V., Iliyasu, A. (eds) Computational Vision and Bio-Inspired Computing. ICCVBIC 2019. Advances in Intelligent Systems and Computing, vol 1108. Springer, Cham. https://doi.org/10.1007/978-3-030-37218-7_53
[3] Shenshen Gu, Rui Cui, An efficient algorithm for the subset sum problem based on finite-time convergent recurrent neural network, Neurocomputing, Volume 149, Part A, 2015, Pages 13-21, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2013.12.063.
[4] DEAP library, https://deap.readthedocs.io/en/master/
[5] https://math.mit.edu/~goemans/18434S06/knapsack-katherine.pdf
[6] Russel S., Norvig P.: Artificial Intelligence: A Modern Approach. Prentice Hall, 2009. Third Edition, ISBN 978-0-13-604259-4
[7] Y. Shoham, K. Leyton-Brown: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge 2008, ISBN 978-0521899437