The Sum-of-Subsets Problem

Backtracking

Oberon Ilano

CS 430

November 18, 2019

Abstract

The purpose of this research paper is to study the Sum-of-Subsets problem and the strategy that will solve this problem. This research paper will conclude the description of the problem, a description of the strategy, and a discussion of how such a strategy works for the problem. A comparison of strategies that will work efficiently to solve the problem will be discussed, including their algorithms, space complexities, and time complexities. Also, this research paper will conclude discussions of an example of a classic problem that can be used for more concrete problems. Finally, readers should have a better understanding of the Sum-of-Subsets problem and the use of Backtracking strategies.

Introduction

The “Sum-of-Subsets” or also known as “Subset Sum” is a classical optimization problem that is NP-complete or is a hard problem in NP (non-deterministic polynomial-time). In this research paper, the strategy of Backtracking is applied to the problem to study its algorithm, complexities and to be compared to the strategy of Dynamic Programming. The Sum-of-Subsets problem has been revisited several times by researchers, and many strategies have been applied to improve its complexity and efficiency. A strategy such as the Dynamic Programming method being the most popular used to solve the problem and studied for research purposes. The Sum-of-Subsets problem can be applied to a real-life situation and can be used for a more concrete problem such as cryptography.

Description of Sum-of-Subsets Problem

The Sum-of-Subsets problem is NP-complete that can be thought of as a special case of the Knapsack problem. The Sum-of-Subsets problem consists of positive integers, to that represents the weight values, and a positive integer that represents the given target value. Both and are given as distinct inputs. The goal of the problem is to determine if there exist subsets of the integers such that the total weight of all integers in a subset is equivalent to the target value . If there exists a sum of all numbers in that is equivalent to the target value , then output “True” or “Yes”. If the sum of all numbers in that is equivalent to the target value does not exists, then output “False” or “No”. It is possible to have multiple set solutions where the sums are equivalent to the target value . For example, given and target value . The sum of and is equivalent to 7, and the sum of and is also equivalent to 7. Therefore, a multiple sets solution is possible.

Description of Backtracking Strategy

Backtracking is the given strategy for this research paper. Backtracking is a modified depth-first search of a tree that is used for NP-complete problems and optimization problems. Typically, Backtracking involves only a tree search that is a preorder tree traversal. Each execution of divide and conquer returns one or more items of the solution through Backtracking. A depth-first search does not require that the children be visited in any order, but in most cases, the children are visited from left to right, and each subset is represented by a path from the root to a leaf which is a path that is followed until a dead end is reached. The Backtracking pruned the state space tree, and if a node is determined that it does not have a solution because there are no more weights that are left to add equivalent to the target value , then it is considered a non-promising. If the target value is smaller than the sum of all nodes in the path, then it is also considered a non-promising [1]. A state space tree is very helpful to use to solve the problem by checking the instance. The left branch of the tree includes the while the right branch of the tree does not include . In the worst case, a state space tree can have nodes. In a full binary of depth , the worst-case time complexity is , where the number of nodes is not smaller than the total number of recursive calls. If a non-promising node exists, then backtracks to its parent and finds another path to determine the solution. If the sum of the weights totals up to a node that is equivalent to the target value , then the path leads to a solution node, which means a promising node exists. The Backtracking algorithm is more efficient when the input values are in non-decreasing order. While Backtracking is a standard approach to solving the Sum-of-Subsets problem, another strategy can be also be used such as Dynamic Programming.

Comparison of Dynamic Programming and Backtracking

In a Dynamic Programming strategy, “instead of enumerating all possible subsets of , it enumerates all possible sums of subsets of , up to and including the target value . Since is a subset of {}, the sum of all its elements cannot be greater than [2].” The Dynamic Programming strategy operates in time where is the target value and is , attaining the complexity of , which is polynomial in and In the Dynamic Programming method, a two-dimensional array is used to solve the problem instead of using a tree. In some cases, Dynamic Programming can be worse than Backtracking when is much smaller than , even exponential in . If the worst-case time complexity for the Dynamic Programming , which is poorer than the worst-case time complexity of the Backtracking which is . The Dynamic Programming strategy or pseudo-polynomial is most efficient when the number of inputs is small, and its efficiency decreases as the number of inputs get larger. Both strategies have an exponential lower bound of which is closely related to the Knapsack problem when alone is used as the complexity parameter [2]. The Sum-of-Subsets problem is known to be sensitive to the density of the problem. The efficiency of the Backtracking is much better for sparse cases while Dynamic Programming is much better for dense cases.

Time and Space Efficiency Related Works

There has been plenty of work and research done relating to space complexity of Sum-of-Subsets problem over the last 62 years or more. The amount of workspace that is needed for an algorithm is measured either in the unit-cost RAM model, or the Turing machine model, where tapes are infinite on both sides. In 1957, both Danzig [4] and Bellman [5] used the unit-cost RAM model for measurement and the method of Dynamic Programming to run the Sum-of-Subsets problem in time and space . “This is pseudo-polynomial in that is if the value of t is bounded by a polynomial in , then it runs in polynomial time and space [3].” In 1975, Ibarra and Kim [8] were able to give a Fully Polynomial Time Approximation Scheme for the Sum-of-Subsets optimization problem with running time and space , where Gens and Levner [10] had another Fully Polynomial Time Approximation scheme algorithm that runs in time ) [3]. Both Fully Polynomial Time Approximation Scheme algorithms were done by using the method of Dynamic Programming and measured in the unit-cost RAM model. In 2010, Elberfeld, Jakoby, and Tantau [10] were able to be obtained logspace upper bounds by using unary encodings of the inputs in the Unary Subsets Sum problem. The Unary Subsets Sum problem is the same as the Sum-of-Subsets problem except where inputs are binary instead of decimal. Later that year, Kane was able to simply the logspace algorithm. In 2012, Elberfeld, Jakoby, and Tantau were able to improve the Unary Subsets Sum under the appropriate encodings. Elberfeld, Jakorby, Tantau, and Kane’s algorithms later became helpful for solving the optimization version of Sum-of-Subsets problem which runs in time polynomial in , , and space bound which is measured on the Turing machine model [3, 7].

Application of Sum-of-Subsets in a Concrete Problem

Cryptography is an example of the use of the Sum-of-Subsets problem in real-life situations. Merkle and Hellman [12] were the first to consider the use of Sum-of-Subsets problem in a public key protocol [11]. The Sum-of-Subsets problem based public key cryptosystem, a set of of size is calculated by utilizing a private key. The private key is distributed as a public key along with some other certain scheme parameters. The sender selects a subset in distinctively related to a plaintext through a binary encoding, then calculates the sum-of-subset and sends as the ciphertext. The receiver with the private key translates the sum-of-subset instance to an efficiently solvable problem and recovers the plaintext by solving it. Any sum-of-subset or ciphertext should not have two different subsets related to it, as in that case, distinct decryption would be impossible.

Conclusion

The Sum-of-Subsets or the Subsets Sum is a classical optimization problem that is closely related to the Knapsack problem. Many strategies that have been implemented to this classic problem including the Backtracking, but the most popular method used is Dynamic Programming. The Dynamic Programming is much better for dense cases while the efficiency of the Backtracking is much better for sparse cases. Researchers have done little improvement with the time and space complexity of the Sum-of-Subsets Problem by using the Turing machine model and unit-cost in RAM model to measure the space of algorithms. Some have used unary or binary encoding as the inputs instead of decimal numbers to further out the research. The Sum-of-Subsets can also be used for secure communications such as encryption and decryption. Such a problem can be helpful among computer systems and cryptography. This research paper has helped me to further my understanding of Sum-of-Subsets and Backtracking.

Appendix

Procedure SumOfSubsets (, weight, total)

// is an integer type that represents number of nodes//

//weight is an integer type that represents the sum of node included in partial solution node//

//total is an integer type represents the remaining items to n (for a node at depth )//

integer , weight, total, w (1 : ) //w is an array sorted in nondecreasing order//

boolean include (1 : n) //include is a boolean type of array//

if is promising then //may lead to solution//

if weight = W then

print (include (1 : ))

else //expand the node when weight is less than W//

include ( + 1) = “true”

SumOfSubsets ( + 1, weight + w ( + 1), total – w ( + 1))

include () = “false”

SumOfSubsets ( + 1, weight, total – w ( + 1))

end SumOfSubsets

Function promising ()

integer weight, total, w (1 : ), W

// represents number of nodes//

//weight represents the sum of node included in partial solution node//

//total represents the remaining items to n (for a node at depth )//

//w is an array sorted in nondecreasing order//

// W represents the target value

if weight + total >= W AND (weight = W OR weight + w () <= W) then

return (true)

else

return (false)

end promising

Bibliography

[1] Richard Neapolitan. 2015. Foundations of algorithms (5th ed.). Jones and Bartlett Publishers, Burlington, MA.

[2] Thomas O’Neil. 2013. An Empirical Study of Algorithms for the Subset Sum Problem. Retrieved November 16, 2019 from http://micsymposium.org/mics\_2013\_Proceedings/submissions/mics20130\_submission\_13.pdf

[3] Anna Gal, Jing-Tang Jang, Nutan Limaye, Meena Mahajan and Karteek Sreenivasaiah. 2016. Space-Efficient Approximations for Subset Sum. Journal of the ACM DOI:https://dx.doi.org/10.1145/2894843

[4] Richard Bellman. 1957. Dynamic Programming. Princeton University Press, Princeton, NJ.

[5] George Dantzig. 1957. Discrete-variable extremum problems. Operations Research 5

[6] Michael Elberfeld, Andreas Jakoby, and Till Tantau. 2010. Logspace versions of the theorems of Bodlaender and Courcelle. In Proceedings of the 2010 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS’10).

[7] Michael Elberfeld, Andreas Jakoby, and Till Tantau. 2012. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In Proceedings of the Symposium on Theoretical Aspects of ComputerScience (STACS’12).

[8] Oscar H. Ibarra and Chul E. Kim. 1975. Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM DOI:https://dx.doi.org/10.1145/321906.321909

[9] Daniel M. Kane. 2010. Unary Subset-Sum is in logspace. arXiv:1012.1336. Retrieved from https://arxiv.org/abs/1012.1336

[10] George Gens and Eugene Levner. 1979. Computational complexity of approximation algorithms for combinatorial problems. In Proceedings of the 8th Symposium of the Mathematical Foundations of Computer Science. DOI:https://dx.doi.org/10.1145/1008861.1008867

[11] Russell Impagliazzo and Moni Naor. 1995. Efficient Cryptographic Schemes Provably as Secure as Subset Sum. Retrieved November 16, 2019 from https://web.stevens.edu/algebraic/Files/SubsetSum/impagliazzo96efficient.pdf

[12] R. C. Merkle and M. Hellman. 1978. Hiding information and Signature in Trapdoor Knapsack, IEEE Transaction on Information Theory. Vol 24.