Solving Subset Sum Problems using Binary Optimization with Applications in Auditing and Financial Data Analysis

David Biesner [1], Rafet Sifa [2], and Christian Bauckhage [2]

[1]Fraunhofer IAIS
[2]Affiliation not available

October 30, 2023

Abstract

Many applications in automated auditing and the analysis and consistency check of financial documents can be formulated in part as the subset sum problem: Given a set of numbers and a target sum, find the subset of numbers that sums up to the target. The problem is NP-hard and classical solving algorithms are therefore not practical to use in real applications. We tackle the problem as a QUBO (quadratic unconstrained binary optimization) problem and show how gradient descent on Hopfield Networks reliably finds solutions for both artificial and real data. We give an outlook for the application of specialized hardware and quantum algorithms.

Solving Subset Sum Problems using Binary Optimization with Applications in Auditing and Financial Data Analysis

1st David Biesner
Fraunhofer IAIS
Bonn, Germany
david.biesner@iais.fraunhofer.de

2nd Rafet Sifa
Fraunhofer IAIS
Bonn, Germany

3rd Christian Bauckhage
Fraunhofer IAIS
Bonn, Germany

4th Bernd Kliem
PWC GmbH WPG
Frankfurt am Main, Germany

***Abstract*—Many applications in automated auditing and the analysis and consistency check of financial documents can be formulated in part as the subset sum problem: Given a set of numbers and a target sum, find the subset of numbers that sums up to the target. The problem is NP-hard and classical solving algorithms are therefore not practical to use in real applications. We tackle the problem as a QUBO (quadratic unconstrained binary optimization) problem and show how gradient descent on Hopfield Networks reliably finds solutions for both artificial and real data. We give an outlook for the application of specialized hardware and quantum algorithms.**

***Index Terms*—qubo, binary optimization, hopfield networks, auditing, financial data analysis**

I. Introduction and Problem Statement

The financial auditing process involves the writing and proofreading of financial reports for the audited company. This process is still largely a manual one: auditors must read documents, compare document content with previous reports, check the completeness of the content to financial regulation checklists and check against both compliance and mathematical errors.

One aspect of mathematical correctness is the correctness of numerical tables, e.g. describing profit and loss for a given year or quarter. All values in these tables must of course correspond to the actual financial situation of the company, which includes the correctness of sums in the tables. For example, for a table depicting the revenue, expenses and income, the values for expenses and income must sum up to the revenue.

During the manual auditing process, one would apply knowledge about financial reports to evaluate which values correspond to which sums for each table and recheck the correctness of the calculations. This is of course highly time and labour intensive, and mistakes are easy to make when auditing a large number of tables.

Automating this process however proves difficult. While a machine has no problem evaluating the correctness of calculations for given table entries, evaluating which values must sum up to which other values is a complex task. Human auditors are either able to apply knowledge on financial reports or knowledge on structure of tables in general: While tables are formatted in a way that human readers can easily evaluate how sums are structured, e.g. by sum values being at the bottom of the tables, indicted by text in the row headers, split from other values by bold lines, teaching a machine to understand these intuitive rules is almost impossible. A strict rule-based approach would be highly dependent on the formatting of specific tables and not generalize well.

A different approach to this problem is therefore ignoring the table structure altogether. Treating single columns or the entire table as a (ordered) set of numbers, one can try to find the values which can be described as a sum of a subset of all other numbers. This problem can of course be solved exactly by deterministic algorithms. However, the size of tables and magnitude of their entries (for financial documents) make many algorithmic approaches impractical.

In this preprint, we evaluate how stochastic algorithms based on gradient descent on so-called Hopfield networks can solve the problem of finding sums in large (both in size and magnitude) tables:

- We first restate the problem of finding sums in tables as the subset sum problem and briefly discuss known deterministic solving algorithms.
- We then derive the general algorithm for gradient descent on Hopfield networks and how to restate the subset sum problem as a problem solvable by these networks (QUBOs).
- We evaluate our Hopfield algorithm on both artificial and real data and discuss applications and future work.

A. Subset Sum Formulation of the problem

We call a set of rules that describe the behaviour of sums in a document, e.g. rows 1-4 sum up to row 5, rows 5 and 6 sum up to row 8, as a *sum structure*. See Figure 1 for an example.

Many tables found in financial reports show the same sum structure in multiple columns, e.g. when comparing financial statements for several quarters and years. Having an efficient algorithm for discovering sums in columns could aid consistency checks by applying the algorithm on one column, extracting a sum structure and checking if the other columns also comply to the found sum structure. If the new column

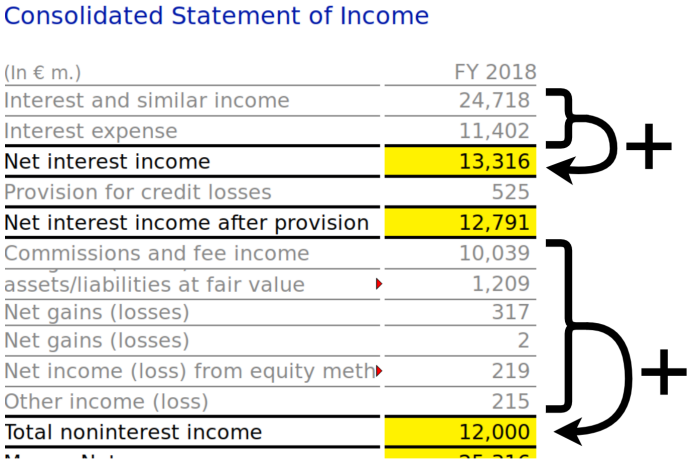

Fig. 1: Column of a table containing financial performance indicators with annotated sums. Numbers from [1].

does not comply to the same sum structure, it is an indication for some inconsistency happening in the table.

Finding sum structures in tables is closely related to a well known problem in algorithmic combinatorics, the subset sum problem. The subset sum problem is defined by a set of numbers $\mathcal{X} = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{N}$ and a target sum $T \in \mathbb{N}$. We aim to find a subset $\mathcal{Y} \subseteq \mathcal{X}$, such that the sum of the subset is equal to the target sum:

$$\sum_{x \in \mathcal{Y}} x = T. \tag{1}$$

In general, due to the fact that there are 2^n possible combinations of numbers for the subset, the problem is NP-hard.

In the framework of consistency checks and finding sums in tables, we can consider the entire table as a set of numbers and apply the problem to each entry: Taken the entry as a target sum, is it possible to find a subset of all other numbers that sums up to the target? Iterating a solving algorithm over each entry in the table yields a sum structure for the table.

B. Classical solving algorithms and algorithms for approximate solutions

There are several known algorithms for solving the subset sum problem.

The naive approach consists of cycling through each of the 2^n possible subsets, summing up all elements and comparing the sum to the target sum. This has a total complexity of $\mathcal{O}(2^n n)$. The algorithm can be improved by several heuristics (i.e. sorting the numbers and stopping iteration when the target sum is surpassed by the subset) but the exponential complexity remains.

Additionally there exist dynamic programming algorithms for solving the subset sum problem exactly in pseudo-polynomial time. That is $\mathcal{O}(n^2 C)$, where $C = B - A$ for A, B being the lower and upper bounds of the set of numbers S.

C. Rule-based algorithms for finding sums in financial tables

The problem of finding sum structures in tables does not have to be broken down to the subset sum problem. By ignoring the inherent structure and logic of the table, the complexity of the binary combination problem is increased. Applying rule-based logic and understanding of the general structure of tables can result in efficient algorithms to solve the problem of finding sum structures in tables.

These rules-based approaches can apply multiple heuristics to find sums: sums are generally more likely to be structured top-to-bottom, sums are likely to occur in adjacent rows to the corresponding subset, the last entries in columns are likely to be sums.

However, rule-based approaches require specialization to each type of table and are hard to generalize.

II. Subset-Sum as QUBO

A quadratic unconstrained binary optimization problem (QUBO) is defined by a function $f : \{0,1\}^n \to \mathbb{R}$ which is a quadratic polynomial over its binary input variables,

$$f(\boldsymbol{z}) = \sum_{ij} p_{ij} z_i z_j = \sum_{i \neq j} p_{ij} z_i z_j + \sum_i p_{ii} z_i. \tag{2}$$

The QUBO problem consists of finding the optimal binary vector $\boldsymbol{z}^* \in \{0,1\}^n$ such that

$$\boldsymbol{z}^* = \underset{z \in \{0,1\}^n}{\arg\min}\, f(z). \tag{3}$$

The problem can be rewritten in matrix notation as

$$\boldsymbol{z}^* = \underset{\boldsymbol{z} \in \{0,1\}^n}{\arg\min}\; \boldsymbol{z}^\intercal \boldsymbol{P}\, \boldsymbol{z} - \boldsymbol{p}^\intercal \boldsymbol{z}$$

with a symmetric and hollow matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ and a vector $\boldsymbol{p} \in \mathbb{R}^n$.

To convert the subset sum problem into a QUBO, we recall the problem statement. Given set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and target value T, determine a subset $\mathcal{Y}^* \subseteq \mathcal{X}$ such that

$$\sum_{x \in \mathcal{Y}^*} x - T = 0. \tag{4}$$

The subset sum problem can therefore be stated as finding $\mathcal{Y}^*$ such that

$$\mathcal{Y}^* = \underset{\mathcal{Y} \subseteq \mathcal{X}}{\arg\min} \left(T - \sum_{y \in \mathcal{Y}} y \right)^2. \tag{5}$$

Collecting the numbers contained in set $\mathcal{X}$ in a vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^\intercal \in \mathbb{R}^n$ and introducing a binary indicator vector $\boldsymbol{z} \in \{0,1\}^n$ with entries

$$z_i = \begin{cases} 1 & \text{if } x_i \in \mathcal{Y} \\ 0 & \text{otherwise} \end{cases}$$

the subset sum problem can alternatively be written as

$$\boldsymbol{z}^* = \underset{\boldsymbol{z} \in \{0,1\}^n}{\operatorname{argmin}} \left(\boldsymbol{x}^\intercal \boldsymbol{z} - T \right)^2.$$

Expanding the equation we write

$$\begin{aligned} \boldsymbol{z}^* &= \underset{\boldsymbol{z}\in\{0,1\}^n}{\arg\min}\ \boldsymbol{z}^\intercal \boldsymbol{x}\,\boldsymbol{x}^\intercal \boldsymbol{z} - 2\,T\,\boldsymbol{x}^\intercal \boldsymbol{z} + const \\ &\equiv \underset{\boldsymbol{z}\in\{0,1\}^n}{\arg\min}\ \boldsymbol{z}^\intercal \boldsymbol{P}\,\boldsymbol{z} - \boldsymbol{p}^\intercal \boldsymbol{z} \\ &=: \underset{\boldsymbol{z}\in\{0,1\}^n}{\arg\min}\ E(\boldsymbol{z}) \end{aligned}$$

where we introduced the shorthands

$$\begin{aligned} \boldsymbol{P} &= \boldsymbol{x}\boldsymbol{x}^\intercal, \\ \boldsymbol{p} &= 2\,T\,\boldsymbol{x}. \end{aligned} \tag{6}$$

Closely related to QUBOs are Ising Models, where we optimize over $\boldsymbol{s} \in \{-1, 1\}$ instead of $\boldsymbol{z} \in \{0, 1\}$:

$$\begin{aligned} \boldsymbol{s}^* &= \underset{\boldsymbol{s}\in\{-1,+1\}^n}{\arg\min}\ \boldsymbol{s}^\intercal \boldsymbol{Q}\,\boldsymbol{s} + \boldsymbol{q}^\intercal \boldsymbol{s} \\ &=: \underset{\boldsymbol{s}\in\{-1,+1\}^n}{\arg\min}\ E(\boldsymbol{s}) \end{aligned}$$

Both problem statements are in fact equivalent, with conversion via $\boldsymbol{z} = \frac{1}{2}(\boldsymbol{s} + \boldsymbol{1})$ and $\boldsymbol{s} = 2\boldsymbol{z} - \boldsymbol{1}$.

Converting the QUBO derived from the subset sum problem above, we have

$$\begin{aligned} E(\boldsymbol{z}) &= \boldsymbol{z}^\intercal \boldsymbol{P}\,\boldsymbol{z} - \boldsymbol{p}^\intercal \boldsymbol{z} \\ &= \tfrac{1}{4}\,(\boldsymbol{s}+\boldsymbol{1})^\intercal \boldsymbol{P}\,(\boldsymbol{s}+\boldsymbol{1}) - \tfrac{1}{2}\,\boldsymbol{p}^\intercal(\boldsymbol{s}+\boldsymbol{1}) \\ &= \tfrac{1}{4}\,\boldsymbol{s}^\intercal \boldsymbol{P}\,\boldsymbol{s} + \tfrac{1}{4}\,\boldsymbol{s}^\intercal \boldsymbol{P}\,\boldsymbol{1} + \tfrac{1}{4}\,\boldsymbol{1}^\intercal \boldsymbol{P}\,\boldsymbol{s} - \tfrac{1}{2}\,\boldsymbol{p}^\intercal \boldsymbol{s} + const \\ &= E(\boldsymbol{s}) \end{aligned}$$

Since matrix $\boldsymbol{P} = \boldsymbol{x}\boldsymbol{x}^\intercal = \left(\boldsymbol{x}\boldsymbol{x}^\intercal\right)^\intercal = \boldsymbol{P}^\intercal$ is symmetric, we have $\boldsymbol{s}^\intercal \boldsymbol{P}\,\boldsymbol{1} = \boldsymbol{1}^\intercal \boldsymbol{P}\,\boldsymbol{s}$. Therefore

$$\begin{aligned} E(\boldsymbol{z}) &= \boldsymbol{z}^\intercal \boldsymbol{P}\,\boldsymbol{z} - \boldsymbol{p}^\intercal \boldsymbol{z} \\ &= \tfrac{1}{4}\,\boldsymbol{s}^\intercal \boldsymbol{P}\,\boldsymbol{s} + \tfrac{1}{2}\,\boldsymbol{1}^\intercal \boldsymbol{P}\,\boldsymbol{s} - \tfrac{1}{2}\,\boldsymbol{p}^\intercal \boldsymbol{s} + const \\ &= \tfrac{1}{4}\,\boldsymbol{s}^\intercal \boldsymbol{P}\,\boldsymbol{s} + \tfrac{1}{2}\left(\boldsymbol{P}\,\boldsymbol{1} - \boldsymbol{p}\right)^\intercal \boldsymbol{s} \\ &\equiv \boldsymbol{s}^\intercal \boldsymbol{Q}\,\boldsymbol{s} + \boldsymbol{q}^\intercal \boldsymbol{s} \\ &= E(\boldsymbol{s}) \end{aligned}$$

where we introduced the shorthands

$$\begin{aligned} \boldsymbol{Q} &= \tfrac{1}{4}\boldsymbol{P} \\ \boldsymbol{q} &= \tfrac{1}{2}\left(\boldsymbol{P}\,\boldsymbol{1} - \boldsymbol{p}\right). \end{aligned} \tag{7}$$

All in all, we can thus consider the subset sum problem as a minimzation problem over $\{-1, 1\}^n$ by

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{-1,+1\}^n}{\arg\min}\ \boldsymbol{s}^\intercal \boldsymbol{Q}\,\boldsymbol{s} + \boldsymbol{q}^\intercal \boldsymbol{s} \tag{8}$$

with Q and q defined in (6) and P and p defined in (7).

A. QUBO-Solving with Hopfield Networks

A Hopfield Network is a recurrent neural net of n interconnected neurons. The state of the network is described by a bipolar vector $\boldsymbol{s} \in \{-1, 1\}^n$. Each neuron is connected to every other neuron, with connection weights given by a matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$. Each neuron s_i is a bipolar threshold unit with threshold θ_i, such that

$$s_i = \operatorname{sign}\left(\boldsymbol{w}_i^\intercal \boldsymbol{s} - \theta_i\right). \tag{9}$$

A Hopfield network architecture is therefore fully described by a matrix $\boldsymbol{W}$, a vector $\boldsymbol{\theta}$ and a current state $\boldsymbol{s}$. An update of the network is done via (9), either for all neurons at once or only a subset of neurons.

We define the energy of the Hopfield network by

$$E(\boldsymbol{s}) \coloneqq -\frac{1}{2}\boldsymbol{s}^\intercal \boldsymbol{W}\,\boldsymbol{s} + \boldsymbol{\theta}^\intercal \boldsymbol{s}. \tag{10}$$

We find that if the weight matrix $\boldsymbol{W}$ is symmetric and is hollow (i.e. has diagonal of all zeros), then the Hopfield energy can never increase when updating one neuron by (9). Since

$$\nabla E(\boldsymbol{s}) = -\boldsymbol{W}\,\boldsymbol{s} + \theta \tag{11}$$

the updates in (9) amount to $s_i = \operatorname{sign}(-\nabla E(\boldsymbol{s})$ and each update performs gradient descent on $E(\boldsymbol{s})$. Since there are only 2^n possible states of the network, successive updates of single neurons will reach a local or global minimum after finitely many updates.

This behaviour can be leveraged to solve problems stated as QUBOs. Encoding the problem in weight and bias parameters $\boldsymbol{W}$ and $\boldsymbol{\theta}$, such that minimum energy states

$$\boldsymbol{s}^* = \underset{\boldsymbol{s}\in\{-1,+1\}^n}{\arg\min}\ -\frac{1}{2}\boldsymbol{s}^\intercal \boldsymbol{W}\,\boldsymbol{s} + \boldsymbol{\theta}^\intercal \boldsymbol{s} \tag{12}$$

solve the underlying QUBO problem, the network may find solutions to the QUBO by the described gradient descent updates of single neurons.

To apply Hopfield networks to the subset sum problem, recall the problem statement as minimization problem over $\boldsymbol{s} \in \{-1, 1\}^n$ in (8). Defining

$$\begin{aligned} \boldsymbol{W} &\coloneqq -2\boldsymbol{Q} = -\frac{1}{2}\boldsymbol{P} \\ \boldsymbol{\theta} &= \boldsymbol{q} = \frac{1}{2}(\boldsymbol{P}\boldsymbol{1} - \boldsymbol{p}) \end{aligned} \tag{13}$$

we find suitable weights and biases such that a the state of a Hopfield network optimized to a global minimum encodes a solution to the subset sum problem.

Note that a Hopfield network with a random initialization does not necessarily converge to a global optimum and local optima are not solutions to the subset sum problem. However, initializing the network multiple times with random states and running until convergence increases the chances of finding a global optimum.

See Algorithm 1 for a description of the full solving algorithm.

Parallel optimization of multiple independent Hopfield networks can efficiently be done on GPUs.

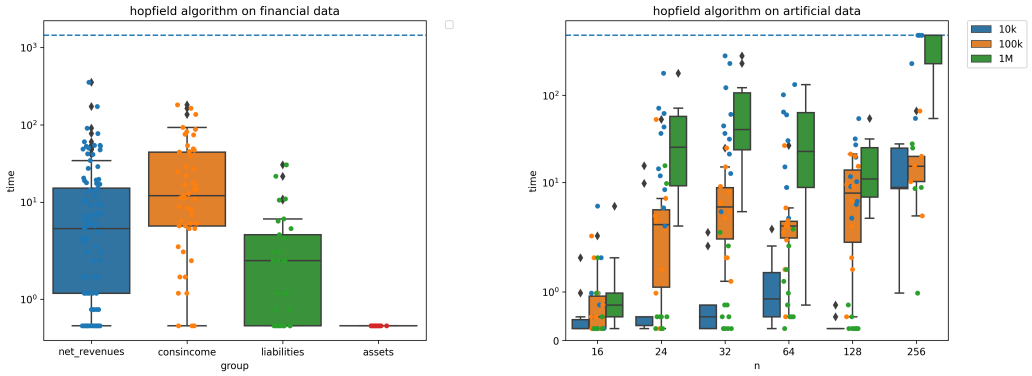

Fig. 2: Benchmarking the Hopfield algorithm on the parsed financial documents (left) and artificial data (right).

Algorithm 1 Algorithm for solving the subset sum problem with Hopfield networks.

Given a vector of numbers $\boldsymbol{x} \in \mathbb{N}^n$ and target sum $T \in \mathbb{N}$, construct Hopfield network weights $\boldsymbol{W}$ and biases $\boldsymbol{\theta}$ by $\boldsymbol{W} \leftarrow -\frac{1}{2}\boldsymbol{x}\boldsymbol{x}^\intercal, \quad \boldsymbol{\theta} \leftarrow \frac{1}{2}(\boldsymbol{x}\boldsymbol{x}^\intercal \mathbf{1} - 2T\boldsymbol{x})$.
Define maximum number of steps $s_{\max}$, $s \leftarrow 0$.
while No solution found and $s < s_{\max}$ **do**
  Initialize network states, sample $\boldsymbol{s} \in \{-1, 1\}^n$
  while Hopfield energy has not converged **do**
    Calculate gradient:
    $\nabla E \leftarrow -\boldsymbol{W}\boldsymbol{s} + \boldsymbol{\theta}$
    Select index of maximum change:
    $i \leftarrow \arg\min_i \nabla E_i$
    Update $\boldsymbol{s}$ at index i:
    $\boldsymbol{s}_i \leftarrow \text{sign}(W_i^\intercal \boldsymbol{s} - \boldsymbol{\theta}_i)$
  if Hopfield energy is minimal: $E(\boldsymbol{s}) = 0$ **then**
    End algorithm.
    return $\{\boldsymbol{x}_i | \boldsymbol{s}_i = 1\}$
  $s \leftarrow s + 1$
Algorithm has not found a solution in $s_{\max}$ steps.
End algorithm.
return None.

III. Experiments

A. Data

We conduct experiments with both artificial data and real data.

To create artificial data we uniformly sample n integers between $X_{\min}$ and $X_{\max}$, select k of the sampled integers at random and calculate the target sum T as the sum of the selected integers.

Note that the selection of n, $X_{\min}$ and $X_{\max}$ has a defines the number of possible solutions to the problem and therefore influences the difficulty of finding a solution. Given a set $\mathcal{X}$ of n integers between $X_{\min} < 0$ and $X_{\max} > 0$, the sum of any subset of $\mathcal{X}$ must be in the interval

$$\mathcal{T} = [nX_{\min}, nX_{\max}] \tag{14}$$

name	n	k	$X_{\max}$	R
n_016_10k	16	4	1e+4	2.0e−01
n_016_100k	16	4	1e+5	2.0e−02
n_016_1M	16	4	1e+6	2.0e−03
n_032_10k	32	8	1e+4	6.7e+03
n_032_100k	32	8	1e+5	6.7e+02
n_032_1M	32	8	1e+6	6.7e+01
n_064_10k	64	8	1e+4	1.4e+13
n_064_100k	64	8	1e+5	1.4e+14
n_064_1M	64	8	1e+6	1.4e+15
n_128_10k	128	8	1e+4	1.3e+32
n_128_100k	128	8	1e+5	1.3e+31
n_128_1M	128	8	1e+6	1.3e+30
n_256_10k	256	8	1e+4	2.6e+70
n_256_100k	256	8	1e+5	1.3e+69
n_256_1M	256	8	1e+6	1.3e+68

TABLE I: Configurations of artificial data. Numbers sampled in the interval $[-X_{\max}, X_{\max}]$. Ratio $R = 2^n/2nX_{\max}$ describes the expected number solutions when sampling a set of numbers and a random target solution.

for a total of $\#\mathcal{T} = n(X_{\max} - X_{\min})$. However, there are 2^n possible subsets of $\mathcal{X}$. For many combinations of n, $X_{\min}$ and $X_{\max}$ we have $2^n >> n(X_{\max} - X_{\min})$ and therefore some target values must have multiple solutions. Finding solutions for a problem with many distinct solutions is of course easier than finding one correct solution in 2^n possible combinations.

We construct artificial data in the configurations described in Table I. For each configuration, we sample $M = 5$ different subset sum problems.

We evaluate our algorithm on a set of real data problems. We parse a financial report[1] containing multiple sheets with financial reports for the quarters from Q1 2019 to Q4 2020 for a total of 190 independent columns. Each column contains numbers describing amounts up to multiple billion €, exact to one cent, for a total of 14 significant figures. See Table II for details on the dataset.

B. Experiments and Results

We run the Hopfield algorithm on artificial data for up to 1e+8 initializations in batches of 1e+4 on one NVIDIA A100

[1]Deutsche Bank Financial Data Supplement Q4 2020, https://investor-relations.db.com/files/documents/quarterly-results/4Q20_FDS.xlsx

name	M	n	X_{min}	X_{max}	R
assets	15	17	1.7e+07	1.5e+14	5.1e−11
consincome	49	31	−2.5e+12	2.5e+12	1.4e−05
liabilities	29	20	2.5e+10	1.5e+14	3.5e−10
net revenues	97	25	−5.3e+10	2.5e+12	5.3e−07

TABLE II: Configurations of parsed financial data. Each group describes one table in the financial report, with M individual subset sum problems (i.e. sums described in the table), a column length of $n+1$ which corresponds to n values in the subset sum problem, and values between X_{min} and X_{max}. Again, R describes the expected number solutions when sampling a set of numbers and a random target solution defined by n, X_{min} and X_{max}. We see that each $R << 1$ and in most columns there is only one unique solution to the subset sum problem.

GPU, for a maximum computation time of around 8 minutes. For almost all configurations the algorithm reliably finds a correct solution for all samples. Only for the configuration of $n = 256$ and $X_{\text{max}} = 1\text{e}{+6}$ not all samples are solved in the specified maximum number of runs (2 of 5 found). See Figure 2 (right) for comparison of computation time against n and X_{max}. We see that the number of values n has a smaller impact on the computation time until solution than the magnitude of the numbers X_{max}.

We run the Hopfield algorithm with the same configuration of the financial data. The algorithm finds a correct solution to the problem under the maximum number of iterations in all cases. Note that unlike for most of the artificial data, the combination of n and X_{max} lead to a situation where each problem likely only contains one correct solution, which is found by our algorithm. See Figure 2 (left) for a comparison of computation time for different tables. We see that average computation time clearly increases with the size of the problem, i.e. amount of values in the table.

See Tables III and III for statistics on the runs. In total, we conclude that optimization of binary vectors with Hopfield networks is a reliable algorithm for solving subset sum problems and can be applied to real-world examples.

IV. Conclusion and Outlook

In this work we investigated how the subset sum problem plays a vital part in the automation of the auditing process, how the subset sum problem can be restated as a well known problem architecture which can be solved by the application of Hopfield networks. We found that the proposed algorithm reliably finds correct sum structures for artificial and real data.

Future work will include an investigation of the application of special purpose hardware (FPGAs) for QUBO-solving and implications of this algorithm architecture for quantum computing applications.

In the near future, the algorithm will be ready to deploy on existing smart auditing software to directly benefit auditors in their daily work.

n	X_{max}	mean time	mean runs
16	10k	0.4	2.0e+04
	100k	0.7	3.2e+04
	1M	1.1	4.9e+04
32	10k	0.7	3.3e+04
	100k	6.9	3.1e+05
	1M	77.2	3.4e+06
64	10k	1.0	4.5e+04
	100k	4.9	2.1e+05
	1M	40.1	1.7e+06
128	10k	0.3	1.3e+04
	100k	9.0	3.6e+05
	1M	18.4	7.3e+05

TABLE III: Results for the Hopfield algorithm on artificial data configurations. The algorithm found a solution for every given sample. For each data configuration we decribe the mean time to find a solution in seconds and the mean number of individual runs until a solution is found.

name	n	M	M_{found}	n	mean time	mean runs
assets	17	15	15	17	0.4	1.0e+04
consincome	31	49	49	31	30.4	8.0e+05
liabilities	20	29	29	20	4.1	1.1e+05
net revenues	25	97	97	25	17.6	4.6e+05

TABLE IV: Results for the Hopfield algorithm on artificial data configurations. For each data configuration we decribe the mean time to find a solution in seconds and the mean number of individual runs until a solution is found. Of M subset sum problems on each table we solved M_{found}.

References

[1] "Deutsche Bank Financial Data Supplement Q4 2020." investor-relations.db.com/files/documents/quarterly-results/4Q20_FDS.xlsx. Accessed: 2021-12-10.

[2] R. Rojas, *Neural Networks: A Systematic Introduction*. Springer, 1996.

[3] C. Bauckhage, R. Ramamurthy, and R. Sifa, "Hopfield networks for vector quantization," in *Proc. ICANN*, Springer, 2020.

[4] R. Sifa, A. Ladi, M. Pielka, R. Ramamurthy, L. Hillebrand, B. Kirsch, D. Biesner, R. Stenzel, T. Bell, M. Lübbering, U. Nütten, C. Bauckhage, U. Warning, B. Fürst, T. D. Khameneh, D. Thom, I. Huseynov, R. Kahlert, J. Schlums, H. Ismail, B. Kliem, and R. Loitz, "Towards Automated Auditing with Machine Learning," in *Proc. DocEng*, ACM, 2019.

[5] K. Koiliaris and C. Xu, "A Faster Pseudopolynomial Time Algorithm for Subset Sum," *arXiv:1507.02318 [cs.DS]*, 2016.

[6] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[7] "Deutsche Bank Jahres- und Konzernabschluss zum Geschäftsjahr vom 01.01.2020 bis zum 31.12.2020." www.bundesanzeiger.de. Accessed: 2021-12-22.

[8] E. L. Schreiber, R. E. Korf, and M. D. Moffitt, "Optimal Multi-Way Number Partitioning," *J. ACM*, vol. 65, no. 4, 2018.

[9] R. Korf, "An Improved Algorithm for Optimal Bin Packing," in *Proc. IJCAI*, 2003.

[10] H. Fujiwara, H. Watari, and H. Yamamoto, "Dynamic Programming for the Subset Sum Problem," *Formalized Mathematics*, vol. 28, no. 1, 2020.

[11] V. Curtis and C. Sanches, "A low-space algorithm for the subset-sum problem on GPU," *Computers Operations Research*, vol. 83, no. C, 2017.

[12] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, 2014.

[13] C. Bauckhage, R. J. Sánchez, and R. Sifa, "Problem Solving with Hopfield Networks and Adiabatic Quantum Computing," in *Proc. IJCNN*, IEE, 2020.

[14] S. Mücke, N. Piatkowski, and K. Morik, "Hardware Acceleration of Machine Learning Beyond Linear Algebra," in *Proc. ECML/PKDD*, Springer, 2019.
[15] S. Mücke, N. Piatkowski, and K. Morik, "Learning Bit by Bit: Extracting the Essence of Machine Learning," in *Proc. KDML-LWDA*, 2019.
[16] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, "Perspectives of Quantum Annealing: Methods and Implementations," *Reports on Progress in Physics*, vol. 83, no. 5, 2020.
[17] "D-Wave System Documentation - What is Quantum Annealing?." docs.dwavesys.com/docs/latest/c_gs_2.html. Accessed: 2021-12-21.