Scientific Reports

https://doi.org/10.1038/s41598-026-66284-9

Article in Press

The new exponential Fibonacci neighborhood as QUBO generated on a D-Wave quantum computing environment for the permutational scheduling problem

Received: 18 May 2026

Accepted: 6 August 2026

Published online: 17 August 2026

Cite this article as: Bożejko W., Uchroński M. & Wodecki M. The new exponential Fibonacci neighborhood as QUBO generated on a D-Wave quantum computing environment for the permutational scheduling problem. *Sci Rep* (2026). https://doi.org/10.1038/s41598-026-66284-9

Wojciech Bożejko, Mariusz Uchroński & Mieczysław Wodecki

The new exponential Fibonacci neighborhood as QUBO generated on a D-Wave quantum computing environment for the permutational scheduling problem

Wojciech Bożejko[1,*,+], Mariusz Uchroński[1,2,+] , and Mieczysław Wodecki[3,+]

[1]Department of Control and Quantum Computing, Wrocław University of Science and Technology, Janiszewskiego 11/17, 50-372 Wrocław, Poland
[2]Wroclaw Centre for Networking and Supercomputing, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
[3]Department of Telecommunications and Teleinformatics, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
[*]wojciech.bozejko@pwr.edu.pl
[+]these authors contributed equally to this work

ABSTRACT

Quantum computers open up new possibilities for faster and more efficient solutions to real-world, large-scale optimization problems that currently pose a challenge for classical computer systems. One such problem is searching very large neighborhoods, with an exponential number of elements, in local search algorithms for solving NP-hard problems. Currently, the only barrier to the effective use of quantum computers in optimization is the limited number of available qubits. Therefore, the main challenge today, given the limited number of qubits, is to formulate the problem to be solved in a form that minimizes the number of variables and, consequently, the number of necessary qubits. To meet these expectations, we propose a new solution representation used to generate neighborhoods with an exponential number of elements. We prove that for n-element permutations, the number of elements in this neighborhood is the n-th Fibonacci number. Permutations are commonly represented by binary matrices with n^2 elements. We propose a new, cost-effective method for representing solutions using $n-1$ element binary sequences. For an $n=50$ element permutation, the former requires 2500 qubits, while the latter requires only 49. Computational experiments for a certain NP-hard single-machine scheduling problem were performed on a D-Wave quantum computer. The obtained results clearly indicate the high effectiveness of the presented method. With a limited number of qubits, it is possible to search neighborhoods for much larger permutations. By examining the capabilities and limitations of current quantum annealers, we can more precisely identify them and then use them in the construction of new algorithms better suited to current and future quantum computers.

Introduction

Discrete optimization is currently one of the primary applications of quantum computing. Research in this field focuses on developing new algorithms that fully leverage the immense capabilities of quantum computers. These efforts target computationally intensive (NP-hard) problems that pose a significant challenge to modern classical computers.

These advances have a direct impact on operational efficiency in various sectors of the economy, including logistics, transportation, manufacturing, and finance. Currently, the main barrier preventing the effective utilization of quantum computing's theoretical potential is the limited number of qubits available in existing quantum hardware. Reports from quantum computer manufacturers indicate that the number of qubits will increase significantly in the near future, enabling solutions on a scale that exceeds the computational capabilities of classical computers. As of 2026, the quantum hardware landscape started to shift toward scaling and fault-tolerant engineering. IonQ[1] is actively enhancing its platform. Its roadmap shows an acceleration toward 2027, with goals of reaching $10,000$ physical qubits and 800 logical qubits, and ultimately delivering a 'fault-tolerant fleet' comprising $80,000$ logical qubits by 2030. Similarly, QuEra Computing[2] is progressing toward a system with more than $10,000$ physical qubits, aimed at providing 100 logical qubits as early as 2026, as announced in 2024. Major industry players such as IBM[3] are targeting the 2027–2033+ timeframe. In the realm of topological quantum computing, Microsoft[4] is progressing toward outstanding milestones, working to integrate programmable QPUs into multi-qubit architectures with the long-term goal of realizing a fault-tolerant system.

The methods and algorithms presented in this work should therefore be treated as a form of preparation for their full

implementation on quantum computers with a significantly higher number of qubits. Growing globalization and increasing competition are the main driving forces behind the search for effective control and management methods, and consequently, the resolution of increasingly complex, often multi-criteria, optimization problems. Despite the continuous growth of computer processing power, the implementations of currently used methods for constructing algorithms often fail to meet the expectations of practitioners. This is because the scale of practical issues and the number of essential factors that must be taken into account are growing rapidly. This directly affects the complexity of the models and leads to a rapid increase ('exponential explosion') in the size of the search space.

The vast majority of optimization issues important for practical applications belong to the class of problems *NP*-hard. The failure of many years of research to prove the inequality of the classes $P \neq NP$ is a strong inspiration, and a necessity, for seeking heuristic methods that can quickly determine solutions that differ little from the optimal ones. Today's classic metaheuristics, based on searching the solution space for a better result than the current one, began to be widely used in the last decade of the twentieth century. Despite their continuous improvement, they do not always meet today's expectations[5–7].

Hence, on one hand, we observe the rapid development of methods for constructing algorithms inspired by various phenomena occurring in nature, and on the other, an intensification of individualized research into the solution spaces of various problems in terms of new properties that improve the efficiency of metaheuristics.

The research presented in this paper is conducted with a view toward its application in the most promising methods for solving NP-hard problems based on local search, and covers the most critical computational stages: generating and searching neighborhoods. We define very large neighborhoods for problems where solutions are represented as permutations. These were applied in an algorithm to solve the Total Weighted Tardiness (TWT) problem. Computational experiments were performed on a D-Wave quantum computer, focusing on both the neighborhoods themselves and their application within the Tabu Search algorithm.

The proposed neighborhood is smaller than the dynasearch neighborhood because it only contains adjacent moves. This allowed us to propose a linear data structure for representing the moves (compounded adjacent swaps), which in turn allowed us to use $n-1$ instead of n^2 qubits, which would be necessary to explore the full dynasearch environment.

The argument for using a quantum computer at the environment exploration stage is as follows: using the classical representation of a move (permutation) as a matrix of size $n \times n$, we potentially generate 2^{n^2} states using the quantum representation, which in practice (shots – repeated running of the quantum circuit) does not even allow for the potential appearance of certain binary systems even for $n = 10$ (2^{10^2} as the number of iterations is many orders of magnitude beyond the capabilities of any current computer). In turn, our proposed representation in the form of a vector of length $n-1$ can only be used for adjacent moves, which we call the Fibonacci neighborhood, for the full dynasearch neighborhood is no longer the case.

Besides this, according to the authors, the proposed environment has another advantage: it is elegant and simple, and such solutions in practice tend to be the most effective, and certainly worth describing for use by other researchers.

Starting with the simplest methods for generating permutation neighborhoods (swapping adjacent pairs of elements – API move) with an exponential number of elements, we generally conduct research in two directions:

1. Generating increasingly larger neighborhoods by combining moves, we examine their impact on the quality of the algorithmically improved solutions (to what extent is it profitable to use larger neighborhoods?). Currently, we are examining the Fibonacci neighborhood, then planning a Motzkin neighborhood based on non-intersecting moves, followed by intersecting moves (involutions).

2. Where is the limit indicating which neighborhoods can be searched polynomially, and what is the effect of this, beyond which the problem of neighborhood search is NP-hard?

Examining the effectiveness of using different neighborhoods in the case of CPUs, there is a strong correlation between the size of the environment and its search time. In the case of QPUs, this correlation is much smaller, so much larger environments can be used. Since the current main limitation of quantum computers is the small number of qubits, we are researching new methods for representing solutions. This research results in the vector representation of elements of the Fibonacci environment presented in this paper.

The entire material in this paper is divided into eight sections. The first section provides a brief introduction to the topic of permutational optimization problems. The next two sections, the second and third, provide basic definitions and describe the classical local search method for solving discrete optimization problems, including neighborhoods generated by compositions of swap moves. The fourth section proves that the number of elements in the neighborhood generated by the composition of adjacent swap moves is equal to the Fibonacci number – we propose calling it the Fibonacci Neighborhood. The next two sections, the fifth and sixth, are devoted to the problem of searching this neighborhood on a quantum computer. The Quadratic Unconstrained Binary Optimization (QUBO) model is formulated for a matrix and a completely new vector representation of solutions. The seventh section presents the results of computational experiments performed on a D-Wave quantum computer – a quantum annealer. The final section provides a brief summary and directions for further research.

Permutational discrete optimization problems

Permutations are feasible solutions to many theoretically and practically important discrete optimization problems. These include, in particular, single- and multi-machine scheduling problems, the traveling salesman problem, vehicle routing, packaging, and, more generally, any problem in which elements must be sorted at some stage of the solution.

Most of these are classified as classical discrete optimization problems. Current research focuses primarily on the use of new algorithm construction methods and new computational environments based on quantum phenomena. Their main goal is to improve the efficiency of algorithms whose solutions on classical computers fail to meet practitioner expectations.

Let $\mathscr{J} = \{1,2,\ldots,n\}$ be some n-element set, and $\mathscr{P}$ be the set of all permutations of elements from $\mathscr{J}$. Where this does not lead to ambiguity, the n-element permutation $\pi = \begin{pmatrix} 1 & 2 & \ldots & n-1 & n \\ \pi(1) & \pi(2) & \ldots & \pi(n-1) & \pi(n), \end{pmatrix}$ will be written, for short, as the sequence $\pi = (\pi(1),\pi(2),\ldots,\pi(n-1),\pi(n))$.

In general, the Permutational Optimization Problem (PPO) involves determining the optimal permutation $\pi_{best} \in \mathscr{P}$ such that:

$$\mathscr{F}(\pi_{best}) = \min\{\mathscr{F}(\pi) : \pi \in \mathscr{P}\}, \tag{1}$$

where the objective (criterion) function $\mathscr{F}$: $\mathscr{P} \to \mathbb{R}^+$.

Case study: Single Machine Total Weighted Tardiness Scheduling Problem

One of the simplest problems in the NP-hard class is the single-machine task scheduling problem with minimizing the sum of tardiness costs, abbreviated as TWT.

TWT Problem. Given a set of n tasks $\mathscr{J} = \{1,2,\ldots,n\}$ that must be executed sequentially on a machine. At any given time, the machine can execute at most one task without interruption. Each task $i \in \mathscr{J}$ is associated with: p_i – *execution time*, d_i – *desired completion deadline*, and w_i – *tardiness penalty factor*. If, for a given order (permutation), $C_i = \sum_{j=1}^{i} p_j$ is the deadline for completing the task, then $T_i = \max\{0, C_i - d_i\}$ is the *delay*, and $w_i T_i$ is the *penalty* (or *cost*) of completing the task. We also need to determine the order of task execution – a permutation of elements of the set $\mathscr{J}$ – that minimizes the *sum of penalties*, i.e., $\sum_{i=1}^{n} w_i T_i$.

The TWT problem is denoted in the literature by $1||\sum w_i T_i$ and belongs to the class of NP-hard problems[8]. The first optimal algorithm based on the branch and bound method was published in 1975 by Rinnooy Kan. et al.[8]. Many properties of the problem have been proven, including the so-called 'block elimination properties,' which allow for the elimination of certain subsets of the solution set without the need for searching. These properties have been successfully applied, among others, in the branch and bound algorithm (Wodecki[9]) and approximate tabu search (Bożejko et al.[10]). Congram et al.[11] presented an optimal algorithm based on the dynamic programming method. A comprehensive review of methods and algorithms for solving the TWT problem is presented in the works of Adamu and Adewumi[12], Martinelli et al.[13], and Koulamas[14].

Despite the passage of over 50 years since the publication of the first algorithm, this problem is still the subject of numerous studies. It is often used to verify the effectiveness of new properties and methods of algorithm construction. In recent years, papers have been published presenting quantum algorithms, as well as classical/quantum hybrid algorithms. See Bożejko et al.[15–19], Wang and Cheng[20], and Grange et al.[21]. A comprehensive overview of quantum algorithms for solving scheduling problems, including single-machine ones, is provided in Werner and Ullinger[22].

In the following section, we present the neighborhood generated by the composition of independent swap moves. Due to the number of elements, it is called the Fibonacci neighborhood. It was implemented in a hybrid classical/quantum approximate algorithm based on the Local Search (LS) method for solving the TWT problem. The most time-consuming procedure of the algorithm, the neighborhood search, is performed on a D-Wave quantum computer.

Local Search method

Many problems arising in areas such as production scheduling and planning, location and distribution management, internet routing, and bioinformatics are discrete optimization problems. They are usually very difficult to solve, as most of them belong to the class of NP-hard problems. Consequently, computational time grows exponentially with the size of the data, making it virtually impossible to solve many large practical examples. Even the constantly increasing computational power of classical computers cannot significantly increase the size of examples solvable in an acceptable time. Therefore, approximate (suboptimal) methods are almost exclusively used to solve NP-hard discrete optimization problems. They do not guarantee that the resulting solution is optimal. Among the most effective are algorithms based on the LS method. A comprehensive overview of exact and approximate methods for solving discrete optimization problems is presented in the works of Taham and Fakhravar[23], and Fakhravar[24].

The Local Search method involves iteratively improving the current solution by searching neighborhoods – subsets of the solution set. It begins with a feasible initial solution. Then, the neighborhood is generated, and a certain element (usually the

best) is determined from it, which is taken as the initial solution in the next iteration. Thus, the solution space is traversed by 'moves', transformations (usually small) of the current solution, moving from one element to another. This process continues until a certain termination criterion is met. This creates a sequence, a trajectory of solutions, the best element of which is the result of the algorithm. Let f be the criterion to be minimized, x the starting solution, and $\mathscr{N}(x)$ its neighborhood.

Algorithm 1: Local search method

Input : permutation x_0 – initial solution;
Output : permutation x_{best} – the best solution;

1 $x_{best} \leftarrow x_0$; $x \leftarrow x_0$;
2 **while** *not stop criterion* **do**
3 determine the neighborhood $\mathscr{N}(x)$;
4 determine the element $y \in \mathscr{N}(x)$ such that $f(y) = \min\{f(z) : z \in \mathscr{N}(x)\}$;
5 **if** $f(y) < f(x_{best})$ **then** $x_{best} := y$;
6 $x \leftarrow y$;
7 **return** x_{best}

The procedure of generating and searching the neighborhood (lines 3 and 4 of the algorithm) has a decisive impact on the computation time and the value of the determined solution. Considering modern quantum computers with a small number of qubits, the most efficient are hybrid silicon/quantum algorithms, in which only small, time-consuming calculations (e.g., searching the neighborhood) are performed on the quantum computer.

Matrix representation of a permutation

Solving the TWT problem involves determining the optimal permutation π_{best} in the set of all permutations $\mathscr{P}$. Any permutation π of $\mathscr{P}$ can be represented by a square binary matrix $\mathbf{x} = [x_{ij}]_{n\times n}$ $(x_{ij} \in \{0,1\})$ such that:

$$x_{ij} = \begin{cases} 1, & \text{if } \pi(i) = j, \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \ \sum_{j=1}^{n} x_{ij} = 1, \ \ i,j = 1,2,\ldots,n, \tag{3}$$

Example 1 Example of a permutation and its corresponding matrix.

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 5 & 3 & 1 & 2 & 6 \end{pmatrix} \qquad \mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Let $\mathscr{X}$ denote the set of all binary matrices satisfying the constraints (2) and (3). Each matrix from $\mathscr{X}$ corresponds to some permutation of $\mathscr{P}$, and vice versa. In the remainder of this paper, we will use both representations of solutions to permutation problems interchangeably.

Representing permutations as binary matrices is commonly used in the construction of optimization algorithms executed on quantum computers, see for example Cattelan and Yarkoni[25]. In the following, we will show that this representation is an extremely inefficient way of representing solutions due to the large number of variables (often the decision variable x has not two, but even three indices!). This is particularly disadvantageous for computations on a quantum computer, as it requires a large number of qubits. We propose a method for representing solutions much more efficiently.

Moves and neighborhoods in the local search method

A fundamental element found in approximate algorithms for solving NP-hard discrete optimization problems based on the local search method is the *neighborhood* – a mapping $\mathscr{N} : \mathscr{P} \mapsto 2^{\mathscr{P}}$, assigning to each element $\pi \in \mathscr{P}$ a subset (neighborhood) $\mathscr{N}(\pi)$ of the set of feasible solutions $\mathscr{P}$, $\mathscr{N}(\pi) \subseteq \mathscr{P}$. Neighborhoods with a polynomial number of elements are most often used in these algorithms.

The number of neighborhood elements, the method of determining and browsing them, has a decisive impact on the efficiency (computation time and the value of the objective function of the resulting solution) of an algorithm based on the local search method. The literature presents numerous procedures for generating and searching neighborhoods used in LS algorithms, see Ahuja et al.[26], Windras et al.[27].

Moves and compositions of moves

A *move* m is a function $m : \mathscr{P} \mapsto \mathscr{P}$, so it transforms any permutation $\pi \in \mathscr{P}$ into another permutation $m(\pi) \in \mathscr{P}$ (actually it's just a composition of permutations since the move represented by m itself is also a permutation). Of the many types of moves which generate neighborhoods considered in the literature, three are particularly important and most frequently used: *swap*, *insert*, and *twist*. In the following, we will consider neighborhoods generated by swap moves. Let π be an n-element permutation.

The swap move s_l^k $(k \neq l,\ k,l = 1,2,\ldots n)$ consists of swapping the positions of two elements $\pi(k)$ with $\pi(l)$, located at positions k and l w π, respectively. All possible such moves are $n(n-1)/2$. Of course, $s_l^k = s_k^l$. Therefore, it is sufficient to consider only one of these moves. To simplify notation, in the remainder of this paper, we will consider only moves s_l^k for $l > k$. The computational complexity of executing swap move is $O(1)$.

Let β be the permutation generated from π by executing move s_l^k (i.e., $\beta = s_l^k(\pi)$, then

$$\beta(i) = \begin{cases} \pi(i), & \text{if } i \neq k \wedge i \neq l, \\ \pi(k), & \text{if } i = l, \\ \pi(l), & \text{if } i = k. \end{cases} \tag{4}$$

Consider the matrix $\mathbf{x}$ of permutations π and the move s_l^k generating the permutation β of π. Let $t = \pi(k)$ and $z = \pi(l)$. Then the matrix $\mathbf{y}$ of permutations β is of the form:

$$y_{ij} = \begin{cases} x_{ij}, & \text{if } (i,j) \notin \{(k,t),(k,z),(l,t),(l,z)\},\ i,j = 1,2,\ldots,n, \\ 1, & \text{if } (i,j) = (k,z) \vee (i,j) = (l,t), \\ 0, & \text{if } (i,j) = (k,t) \vee (i,j) = (l,z). \end{cases} \tag{5}$$

Example 2 Generating permutations (definition in eq. (4)).

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 4 & 5 & 2 \end{pmatrix} \text{ move } s_4^2,\ \beta = s_4^2(\pi) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 1 & 2 \end{pmatrix}.$$

Example 3 Generating a matrix (definition in eq. (5)) for the permutation from the Example 2.

$$\mathbf{x} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{move } s_4^2, \quad \mathbf{y} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Composition of moves. Let $\mathscr{M}$ be the set of moves performed on permutations from the set $\mathscr{P}$. We consider two moves $m, v \in \mathscr{M}$. Let δ be the permutation generated by performing move m on permutation π, i.e., $\beta = m(\pi)$. Then, performing move v on permutation β generates permutation γ, i.e., $\gamma = v(\beta)$. Therefore, γ is generated from π by combining moves (multimoves) v and m. We then write that $\gamma = v(m(\pi))$ or $\gamma = vm(\pi)$. The composition operation can, of course, be extended to a larger number (sequence) of moves. It should be remembered that, in general, the composition of moves may not be commutative. *Multimoves* will be used to generate 'very large' neighborhoods – with an exponential number of elements, also called 'huge'.

Multimoves were used in LS algorithms to disperse computations. In the works of Grabowski and Wodecki[28,29] they were used to direct computational trajectories to distant (due to the number of moves) regions, which effectively caused the current local minimum to be left behind.

Neighborhoods

For many years, research, mainly experimental, has been conducted on the use of increasingly larger neighborhoods in LS algorithms, including those with an exponential number of elements. The size of the neighborhood, the method of generating and browsing it, has a decisive influence on convergence, the values of the solutions, and the algorithm's running time. Larger neighborhoods enable searching large areas of the solution space. However, they increase the computation time of a single iteration of the algorithm and, consequently, the number of iterations within a given time. This does not always lead to improved

solutions. Currently, for a given problem, there are no methods for selecting the best neighborhood from many possible ones. This selection is usually made experimentally. Unfortunately, the results are often ambiguous. Therefore, neighborhoods, i.e., the procedures for generating and browsing them, should be considered in the context of a specific optimization problem and its solution method. In the following sections, we consider neighborhoods generated by combining independent changes to neighboring elements.

For simplicity, when considering neighborhoods, we will assume that $\pi = (1,2,\ldots,n)$ is a *natural permutation* (identity permutation). The elements of any permutation can be renumbered to make it a natural permutation. A natural permutation corresponds to the identity matrix (the main diagonal contains ones, and the off-diagonal contains zeros). Furthermore, we will denote the ceiling function by $\lceil\ \rceil$, and the set of natural numbers with zero by $\mathbb{N}^+$, i.e., $\mathbb{N}^+ = \mathbb{N} \cup \{0\}$.

Independent adjacent swaps – Fibonacci neighborhood

A special case of a swap movement is *Adjacent Pair Interchange*, or API for short, i.e., a movement of the form s_k^{k+1}, $k = 1,2,\ldots,n-1$. In an n-element permutation, there are $n-1$ such movements.

Two API moves s_k^{k+1} and s_l^{l+1} $(1 \le k,l \le n-1)$ are called *independent* if

$$(l > k+1) \vee (k > l+1) \tag{6}$$

the condition $|k-l| \ge 2$. is equivalent to generating neighborhoods. When generating neighborhoods, we will compose API moves. Below we present an example of compositing two API moves.

Example 4 Composition of API moves (permutation).

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} \text{ composition of } s_1^2 \text{ and } s_4^5 \ \ \beta = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathbf{2} & \mathbf{1} & 3 & \mathbf{5} & \mathbf{4} & 6 \end{pmatrix}.$$

Example 5 Composition of API moves (matrix).

$$\mathbf{x} = \begin{bmatrix} 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{bmatrix} \quad \text{composition of } s_1^2 \text{ and } s_4^5 \quad \mathbf{y} = \begin{bmatrix} 0&\mathbf{1}&0&0&0&0 \\ \mathbf{1}&0&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&0&\mathbf{1}&0 \\ 0&0&0&\mathbf{1}&0&0 \\ 0&0&0&0&0&1 \end{bmatrix}$$

Let $\overline{API}(\pi)$ be a neighborhood of the permutation π, whose elements are generated by sequences of pairwise independent API moves (we also include the permutation π in this set). We will show that the number of elements in this neighborhood is a Fibonacci number. Therefore, we will call it *Fibonacci neighborhood*.

The Fibonacci sequence is defined by the recursive formula:

$$F_0 = 1,\ F_1 = 1, \quad F_n = F_{n-1} + F_{n-2},\ n = 2,3,\ldots. \tag{7}$$

The exact value of F_n is given by

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1},$$

approximately $F_n \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} = O(1.618^n)$.

Theorem 1 *If π_n is an n-element natural permutation, then the number of elements in the neighborhood $\overline{API}(\pi_n)$ generated by a sequence of independent API moves is*

$$|\overline{API}(\pi_n)| = F_n,$$

The theorem states that the number of compositions of independent adjacent interchanges equals a Fibonacci number, which is equivalent to the classical identity that the number of binary strings of length $n-1$ without two consecutive ones equals a Fibonacci number. This is a well-known combinatorial result[30]; however, we will present a proof illustrating the dynamic programming approach for generating such moves.

Proof. Proof by induction on n – the size of the natural permutation π_n.

For $n = 1$, $|\overline{API}(\pi_1)| = 1 = F_1$.

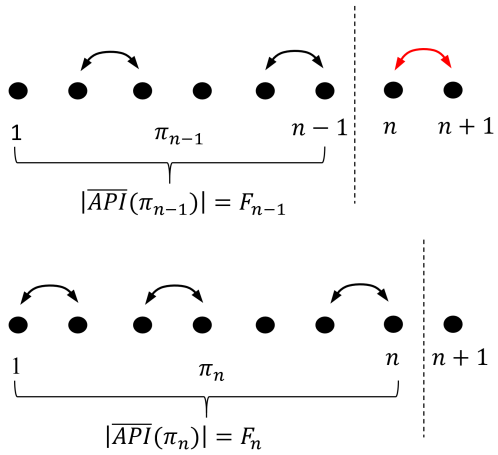

Figure 1. Example: Adding element $n+1$ to permutation π_n

In turn, when $n = 2$, two permutations $(1,2)$ and $(2,1)$ can be generated. Therefore, $|\overline{API}(\pi_2)| = 2 = F_2$.

Inductive assumption. We assume that the neighborhood $\overline{API}(\pi_k)$ of any k-element permutation π_k $(1 \leq k \leq n)$ has F_k elements.
Thesis. We prove that the neighborhood $|\overline{API}(\pi_{n+1})|$ of permutations π_{n+1} has F_{n+1} elements.

Let π_{n+1} be an $n+1$ natural permutation. This permutation is obtained by adding element $n+1$ to the end of permutation π_n. When generating elements of the environment $|\overline{API}(\pi_{n+1})|$, we consider two cases (Figure 1):

1. We swap the positions of element $n+1$ with the n-th, performing move s_n^{n+1} (top Figure 1). In this case, the neighborhood $\overline{API}(\pi_{n+1})$ contains all elements of the neighborhood $n-1$ of the element permutation π_{n-1}, i.e. the elements of the set $\overline{API}(\pi_{n-1})$, which is assumed to be F_{n-1}.

2. The element $n+1$ remains in its position (it is not moved, bottom Figure 1). In this way, we obtain all elements of the neighborhood of permutation π_n (elements of the set $\overline{API}(\pi_n)$), of which is by definition F_n. To summarize,

$$|\overline{API}(\pi_{n+1})| = |\overline{API}(\pi_{n-1})| + |\overline{API}(\pi_n)| = F_{n-1} + F_n = F_{n+1}.$$

■

In the following sections, we present a method for searching the $\overline{API}$ neighborhood for the task scheduling problem on a single TWT machine, formulated in Section 2. We present the mathematical model and formulate it as a Constrained Quadratic Model (CQM) and then a Binary Quadratic Model (BQM). Based on the latter, we provide a version of QUBO that can be directly implemented on a D-Wave quantum computer.

Dynamic Programming for Fibonacci neighborhood

We define a partial sequence to be in state (j,π), for $j = 1,2, \ldots, n$, if it can be obtained from the partial sequence $\pi(1),\pi(2), \ldots pi(j)$ by applying a series of independent adjacent swaps. Let π_j be a partial sequence with minimum total weighted tardiness for jobs $\pi(1),\pi(2),\ldots,\pi(j)$ among partial sequences in state (j,π). Further, let $F(\pi_j)$ be the total weighted tardiness for jobs $\pi(1),\pi(2),\ldots,\pi(j)$ in π_j i.e.

$$F(\pi_j) = min\{F(\beta): \quad \beta \in (j,\pi\}.$$

This partial sequence must be obtained from a partial sequence π_i that has the minimum objective value from all partial sequences in first previous state (i,π), where $0 \leq i < j$, by appending job $\pi(j)$ if $i = j-1$, or by first appending job $\pi(j)$ and then interchanging jobs $\pi(i+1)$ and $\pi(i)$. These two possibilities are considered in detail bellow.

A. $i = j-1$. In this case, job $\pi(j)$ is not involved in any $\pi(j)$ swap, and $\pi(j)$ is simply append to a partial sequence π_{j-1}; hence $\pi_j = (\pi_{j-1},\pi_j)$. Accordingly,

$$F(\pi_j) = F(\pi_{j-1}) + w_{\pi(j)}(C_{\pi(j)} - d_{\pi(j)})^+, \tag{8}$$

where, for any real x, $(x)^+ = \max\{0,\ x\}$.

B. $i = j-2$. Here, jobs $\pi(j)$ and $\pi(j-1)$ are swapped, and the total weighted tardiness $F(\pi_j)$ is readily computed as

$$F(\pi_j) = F(\pi_{j-2}) + w_{\pi(j)}(C_{\pi(j-2)} + p_{\pi(j)} - d_{\pi(j)})^+ + w_{\pi(j-1)}(C_{\pi(j-2)} + p_{\pi(j)} + p_{\pi(j-1)} - d_{\pi(j-1)})^+. \tag{9}$$

These values can be determined recursively using dynamic programming.

For any $j = 2,\ 3\ ,\ \ldots\ ,\ n$

$$F(\pi_j) = \min \begin{cases} F(\pi_{j-1}) + w_{\pi(j)}(C_{\pi(j)} - d_{\pi(j)})^+, \\ \\ F(\pi_{j-2}) + w_{\pi(j)}(C_{\pi(j-2)} + p_{\pi(j)} - d_{\pi(j)})^+ + w_{\pi(j-1)}(C_{\pi(j-2)} + p_{\pi(j)} + p_{\pi(j-1)} - d_{\pi(j-1)})^+. \end{cases} \tag{10}$$

where, the initialization is $F(\pi_0) = 0$ and $F(\pi_1) = w_{\pi(1)}(p_{\pi(1)} - d_{\pi(1)})^+$.

The optimal solution is $F(\pi_n)$, and this dynamic programming algorithm runs in $O(n)$ time.

Searching the Fibonacci neighborhood on a quantum computer – matrix representation

To simplify the notation, we assume that $\pi \in \mathscr{P}$ is an n element-long natural permutation, and the matrix $\mathbf{x} \in \mathscr{X}$ is its representation.

If we make a move s_i^{i+1} (i.e., replacing $\pi(i)$ with $\pi(i+1)$, $i = 1,2,\ldots,n-1$), then the following substitutions must be made in the permutation matrix $\mathbf{x}$: $x_{ii} = x_{i+1,i+1} = 0$, $x_{i+1,i} = x_{i,i+1} = 1$. The remaining elements remain unchanged in the same positions. Similar modifications must be made repeatedly if we make a series of independent API moves. Therefore, the equalities:

$$x_{i+1,i} = x_{i,i+1} \quad i = 1,2,\ldots,n-1, \tag{11}$$

are a necessary and sufficient condition for the matrix $\mathbf{x}$ to be an element of the Fibonacci neighborhood, i.e., $\mathbf{x} \in \overline{API}$.

Mathematical model of the single machine total weighted tardiness scheduling problem

Let $\pi \in \mathscr{P}$ be an n-element natural permutation and let the matrix $\mathbf{x} \in \mathscr{X}$ be its representation. In this case, the desired deadline for task i, $d_i = \sum_{j=1}^{n} d_j x_{ij}$, and its completion time $C_i = \sum_{k=1}^{i} \sum_{j=1}^{n} x_{kj} p_k$.

The problem of determining the optimal element from the Fibonacci neighborhood (searching the set $\overline{API}$), as a mathematical programming task, can be represented as follows: determine

$$\min_{\mathbf{x} \in \mathscr{X}} \sum_{i=1}^{n} \sum_{j=1}^{n} w_j x_{ij} T_i, \tag{12}$$

subject to

$$x_{i+1,i} = x_{i,i+1}, \quad i = 1,2,\ldots,n-1, \tag{13}$$

$$x_{i-1,i} + x_{i,i} + x_{i+1,i} = 1, \quad i = 1,2\ldots,n, \tag{14}$$

$$x_{i,i-1} + x_{i,i} + x_{i,i+1} = 1, \quad i = 1,2,\ldots,n, \tag{15}$$

$$x_{0,1} = x_{1,0} = x_{n,n+1} = x_{n+1,n} = 0, \tag{16}$$

$$T_i = \max\{0, C_i - d_i\} = \max\left\{0, \sum_{k=1}^{i} \sum_{j=1}^{n} x_{kj} p_k - \sum_{j=1}^{n} x_{ij} d_j\right\}. \tag{17}$$

The constraint (13) expresses the symmetry of the matrix elements lying on the diagonals adjacent to the main diagonal, below and above it. The constraints (13), (14) and (15) guarantee that the matrix $\mathbf{x}$ was generated from the identity matrix by executing a series of independent API moves. The additional constants (16) are technical in nature and allow us to write recursive formulas (14) and (15) referring to row and column elements with indices 0 and $n+1$. We assign them the constant value 0. Recall that $\mathscr{X}$ is a set of n^2-element binary matrices satisfying the constraints (3), i.e., feasible solutions to the TWT problem.

Example 6 The neighborhood matrix of $\overline{API}$.

Let $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \mathbf{2} & \mathbf{1} & 3 & \mathbf{5} & \mathbf{4} & 6 & \mathbf{8} & \mathbf{7} \end{pmatrix}$. The permutation matrix $\mathbf{x}$ presented below was generated from a natural permutation (the identity matrix) by performing independent moves s_1^2, s_4^5 and s_7^8. It was augmented with additional 'artificial' columns, and rows numbered 0 and $n+1$. In the mathematical model (12) – (17) of the TWT problem, these elements are given the value 0 (they are highlighted in bold in the matrix)

$$\mathbf{x} = \begin{bmatrix}
- & \mathbf{0} & - & - & - & - & - & - & - & - \\
\mathbf{0} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & - \\
- & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & - \\
- & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & - \\
- & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & - \\
- & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & - \\
- & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & - \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & - \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \mathbf{0} \\
- & - & - & - & - & - & - & - & \mathbf{0} & -
\end{bmatrix}.$$

In the matrix $\mathbf{x}$, three-element horizontal and vertical rectangles are marked. The horizontal rectangles represent constraints (14), and the vertical rectangles represent constraints (15). Each rectangle contains exactly one 1; the remaining elements are 0.

CQM model with inequalities

The next step leading to the QUBO model is to formulate the Fibonacci neighborhood search problem as a Constrained Quadratic Model (CQM). We replace the maximum in the equality (17) with two inequalities:

$$T_i \geq 0, \tag{18}$$

$$T_i \geq \sum_{k=1}^{i} \sum_{j=1}^{n} x_{kj} p_k - \sum_{j=1}^{n} x_{ij} d_j, \; i = 1,2,\ldots,n. \tag{19}$$

They guarantee the admissibility of the thus determined tardiness T_k, and furthermore, the determined solution to the problem (12) is optimal. Next, we reduce these inequalities to equality by introducing additional non-negative variables s_i ($s_i \in \mathbb{N}^+$, $s_i \leq T_i$). Then, the tardiness

$$T_i = s_i + C_i - \sum_{j=1}^{n} x_{ij} d_j, \; i = 1,2,\ldots,n. \tag{20}$$

Therefore, the problem of searching the neighborhood $\overline{API}$ reduces to the following linear programming problem:

$$\min_{\substack{x_{ij} \in \{0,1\},\, s_i \in \mathbb{N}^+ \\ 1 \leq i,j \leq n}} \; \sum_{i=1}^{n} \sum_{j=1}^{n} w_j x_{ij} T_i \tag{21}$$

subject to:

$$x_{i+1,i} = x_{i,i+1}, \;\; i = 1,2,\ldots,n-1, \tag{22}$$

$$x_{i-1,i} + x_{i,i} + x_{i+1,i} = 1, \;\; i = 1,2\ldots,n, \tag{23}$$

$$x_{i,i-1} + x_{i,i} + x_{i,i+1} = 1, \;\; i = 1,2,\ldots,n, \tag{24}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \; j = 1,2,\ldots,n, \;\; \sum_{j=1}^{n} x_{ij} = 1, \; i = 1,2,\ldots,n, \tag{25}$$

$$x_{0,1} = x_{1,0} = x_{n,n+1} = x_{n+1,n} = 0, \tag{26}$$

$$T_i = s_i + \sum_{i=k}^{i} \sum_{j=1}^{n} x_{kj} p_k - \sum_{j=1}^{n} x_{ij} d_j. \tag{27}$$

BQM model with equalities

The next step is to formulate the neighborhood search problem ((21)-(27)) as a Binary Quadratic Model (BQM).

This boils down to replacing the variables (taking integer values) with binary variables. By $\hat{T}$ we denote the upper estimate of the maximum length of the binary expansion of the delays T_k (we assumed $\hat{T} = \sum_{i=1}^{n} p_i - \min_i\{d_i\}$). Then

$$T_i = \sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l z_{il} \text{ and } s_i = \sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l \cdot y_{il}, \tag{28}$$

where $z_{il}, y_{il} \in \{0,1\}$, $i \in \{1,2,\ldots,n\}$, $l \in \{1,2,\ldots,\lceil \log \hat{T} \rceil\}$.

We will present the solution to the neighborhood search problem as a binary quadratic programming problem:

$$\min_{\substack{x_{ij}, y_{il}, z_{il} \in \{0,1\} \\ 1 \le l \le \lceil \log_2 \hat{T} \rceil \\ 1 \le i,j \le n}} \sum_{i=1}^{n} \sum_{j=1}^{n} w_j x_{ij} \left(\sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l z_{il} \right) \tag{29}$$

subject to

$$x_{i+1,i} = x_{i,i+1}, \ \ i = 1,2,\ldots,n-1, \tag{30}$$

$$\sum_{i=j-1}^{j+1} x_{ij} = 1, \ j = 1,2,\ldots,n, \ \sum_{j=i-1}^{i+1} x_{ij} = 1, \ i = 1,2,\ldots,n, \tag{31}$$

$$x_{0,1} = x_{1,0} = x_{n,n+1} = x_{n+1,n} = 0, \tag{32}$$

$$\sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l z_{kl} = \sum_{j=0}^{\lceil \log_2 \hat{T} \rceil} 2^j \cdot y_{kj} + \sum_{i=1}^{k} \sum_{j=1}^{n} p_j x_{ij} - \sum_{j=1}^{n} x_{kj} d_j, \ \ k = 1,2,\ldots,n. \tag{33}$$

Determining the minimum $\hat{T}$ for the given data is correlated with solving the whole problem of minimizing the weighted number of lags, hence to minimize this value – and thus the number of bits dedicated to its storage – we have assumed a certain upper bound. Setting $\hat{T} = \sum p_j - \min d_j$ guarantees that for each permutation all delays $T_i \le \hat{T}$.

QUBO model

Finally, the QUBO (unconstrained binary quadratic programming) model for the Fibonacci neighborhood search problem with matrix solution representation takes the form:

$$\min_{\substack{x_{ij}, y_{il}, z_{il} \in \{0,1\} \\ 1 \le l \le \lceil \log_2 \hat{T} \rceil \\ 1 \le i,j \le n}} \left[\sum_{i=1}^{n} \sum_{j=1}^{n} w_j x_{ij} \left(\sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l z_{il} \right) + \hat{K} \sum_{k=1}^{n} \left(\sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l z_{kl} - \sum_{j=0}^{\lceil \log_2 \hat{T} \rceil} 2^j \cdot y_{kj} - \sum_{i=1}^{k} \sum_{j=1}^{n} p_j x_{ij} + \sum_{j=1}^{n} x_{kj} d_j \right)^2 + \right.$$

$$+\hat{K}\sum_{j=1}^{n}\left(\sum_{i=j-1}^{j+1}x_{ij}-1\right)^2+\hat{K}\sum_{i=1}^{n}\left(\sum_{j=i-1}^{i+1}x_{ij}-1\right)^2+\hat{K}\sum_{i=1}^{n-1}(x_{i+1,i}-x_{i,i+1})^2\Bigg], \tag{34}$$

where $x_{0,1}=x_{1,0}=x_{n,n+1}=x_{n+1,n}=0$, and constants $\hat{T}=\sum_{j=1}^{n}p_j-\min_i\{d_i\}$, $\hat{K}=\sum_{j=1}^{n}w_j\hat{T}$. The values of the constant $\hat{T}$ have a direct impact on the number of qubits needed to solve a particular example.

Although representing permutations (solutions) as binary matrices is commonly used in the construction of optimization algorithms executed on quantum computers (see, for example, Cattelan and Yarkoni,[25]), we will later show that this representation is extremely inefficient. We will demonstrate a method for efficiently storing solutions in the form of a one-dimensional $n-1$-element vector.

Binary vectors as a representation of solutions – compounded adjacent swaps

In this section, we present a very efficient binary sequence (vector) representation of such a permutation, which is a composition of adjacent swaps (compounded adjacent swaps). It will be used in the algorithm for generating and searching the Fibonacci neighborhood $\overline{API}$.

Let $\pi=(1,2,\ldots\ldots n)$ be a natural permutation. We will represent it by an $n-1$ element sequence (vector) containing all zeros. If the permutation β was generated from π by performing a sequence of independent moves API

$$(s_{i_1}^{i_1+1},s_{i_2}^{i_2+1},\ldots,s_{k_k}^{i_k+1}),$$

then it has a corresponding binary sequence $\mathbf{x}=(x_1,x_2,\ldots,x_{n-1})$ such that

$$x_i=\begin{cases}1, & \text{if } i\in\{i_1,i_2,\ldots,i_k\},\\ 0, & \text{if } i\notin\{i_1,i_2,\ldots,i_k\}.\end{cases} \tag{35}$$

If $x_i=1$, this means that when generating the permutation β from π, the API move s_i^{i+1}, was performed, i.e., the element $\pi(i)$ from $\pi(i+1)$ was swapped with π.

Example 7 Binary sequence representing an element of the environment $\overline{API}$.

From the natural permutation π, the moves s_1^2 and s_4^5 were used to generate the permutation β. The swapped elements are marked in bold.

$$\pi=\begin{pmatrix}1&2&3&4&5&6\\1&2&3&4&5&6\end{pmatrix}\text{ composition of } s_1^2 \text{ and } s_4^5\ \ \beta=\begin{pmatrix}1&2&3&4&5&6\\\mathbf{2}&\mathbf{1}&3&\mathbf{5}&\mathbf{4}&6\end{pmatrix}.$$

The permutation β is represented by the $n-1$ element binary sequence $\mathbf{x}$=(1,0,0,1,0). Since $x_1=1$, when generating the β permutation, the positions of the elements $\pi(1)$ and $\pi(2)$ were swapped (movement s_1^2). Similarly, $x_4=1$ so the positions of $\pi(4)$ and $\pi(5)$ were swapped (movement s_4^5).

Mathematical model of the Fibonacci neighborhood search problem

Elements of the Fibonacci neighborhood of the natural permutation π (of the set $\overline{API}$) are generated by sequences of independent API moves. From the definition of (35), it follows that the $n-1$ element binary sequence $\mathbf{x}=(x_1,x_2,\ldots,x_{n-1})$ generates an element of the Fibonacci neighborhood of the permutation π. This is admissible if a zero appears before and after the one. Assuming two additional constant values $x_0=0$ and $x_n=0$, the above constraints can be written as follows:

$$x_i+x_{i+1}\le 1,\ i=1,2,\ldots,n-1. \tag{36}$$

Let $\mathscr{Y}$ be the set of all $n-1$ element binary sequences satisfying the constraint (36). It is easy to prove the one-to-one correspondence between permutations of the set $\overline{API}$ and binary sequences from the set $\mathscr{Y}$. As we have already proven (Theorem 1), for n-element permutations, the number of elements in the neighborhood $|\overline{API}|=F_n$, where F_n is the n-th Fibonacci number.

Suppose that the permutation $\pi'\in\mathscr{Y}$ (an element of the $\overline{API}$) represented by the binary sequence $\mathbf{x}=(x_1,x_2,\ldots,x_{n-1})$ was generated from the natural permutation π. If $x_i=1$ $(i=1,2,\ldots,n-1)$, then when generating π', the elements $\pi(i)$ and $\pi(i+1)$ were swapped. Therefore, for the TWT problem considered in this paper, the completion time of the task located at position i in the π' permutation

$$C_i'=\sum_{j=1}^{i}p_j+x_i(p_{i+1}-p_i),\ i=1,2\ldots,n-1, \tag{37}$$

$$C'_n = \sum_{j=1}^{n} p_j, \tag{38}$$

Additionally assuming constants $x_n = 0$ and $p_{n+1} = 0$, we can express both the equalities (37) and (38) as follows:

$$C'_i = \sum_{j=1}^{i} p_j + x_i(p_{i+1} - p_i),\ i = 1,2\ldots,n. \tag{39}$$

Similarly, if $x_i = 1$, then we need to determine new values for the desired completion dates of tasks d'_i. To do this, we perform the following substitutions:

$$d'_i = d_{i+1} \Leftrightarrow d'_i = x_i d_{i+1} \text{ and } d'_{i+1} = d_i \Leftrightarrow d'_{i+1} = (1 - x_{i+1})x_i d_i,$$

(the next $x_{i+1} = 0$ because x is allowed, i.e., there are no two 1s next to each other). In turn, when $x_{i-1} = 0$ and $x_i = 0$ (the element $\pi(i)$ was not 'moved'), then

$$d'_i = d_i \ \Leftrightarrow\ d'_i = (1 - x_i)(1 - x_{i-1})d_i.$$

Using the above relations, we can determine the new desired task completion deadlines for the permutation π', as follows:

$$d'_i = x_i d_{i+1} + (1 - x_i)x_{i-1}d_{i-1} + (1 - x_i)(1 - x_{i-1})d_i,\ i = 2,3\ldots,n-1. \tag{40}$$

Additionally, for the deadline of the first task (there is no previous one)

$$d'_1 = x_1 d_2 + (1 - x_1)d_1, \tag{41}$$

and for the last task (there is no next one)

$$d'_n = x_{n-1}d_{n-1} + (1 - x_{n-1})d_n. \tag{42}$$

Assuming the constants $x_0 = 0$ and $d_0 = d_{n+1} = 0$, the equality (40)–(42) can be represented as:

$$d'_i = x_i d_{i+1} + (1 - x_i)x_{i-1}d_{i-1} + (1 - x_i)(1 - x_{i-1}),\ \ i = 1,2\ldots,n. \tag{43}$$

Since $x_i x_{i-1} = 0$ (two 1's cannot appear next to each other in the sequence), then

$$d'_i = x_i d_{i+1} + x_{i-1}d_{i-1} - \underbrace{x_i x_{i-1}}_{=0} d_{i-1} + d_i - x_i d_i - x_{i-1}d_i + \underbrace{x_i x_{i-1}}_{=0} d_i =$$

$$x_i d_{i+1} + x_{i-1}d_{i-1} + d_i - x_i d_i - x_{i-1}d_i,\ \ i = 1,2\ldots,n, \tag{44}$$

Similarly to determining the dependence on d'_i (formula (43)), we can determine new values (i.e., in the π' permutation) of the penalty coefficients

$$w'_i = x_i w_{i+1} + (1 - x_i)x_{i-1}w_{i-1} + (1 - x_i)(1 - x_{i-1})w_i,\ \ i = 1.2\ldots,n. \tag{45}$$

Additionally, assuming constant values $w_0 = w_{n+1} = 0$. Finally,

$$w'_i = x_i w_{i+1} + x_{i-1}w_{i-1} + w_i - x_i w_i - x_{i-1}w_i,\ \ i = 1.2\ldots,n. \tag{46}$$

The problem of searching the Fibonacci neighborhood of a natural permutation, generated by binary sequences (compositions of independent API moves), comes down to determining

$$\min_{\mathbf{x} \in \{0,1\}^{n-1}} \sum_{i=1}^{n} w'_i T'_i \tag{47}$$

subject to:

$$x_i + x_{i+1} \leq 1,\ i = 1,2\ldots,n-2, \tag{48}$$

where:

$$T_i' = \max\{0, C_i' - d_i'\},\ i = 1,2\ldots,n, \tag{49}$$

$$C_i' = \sum_{j=1}^{i} p_j + x_i(p_{i+1} - p_i),\ i = 1,2\ldots,n, \tag{50}$$

$$d_i' = x_i d_{i+1} + x_{i-1} d_{i-1} + d_i - x_i d_i - x_{i-1} d_i,\ \ i = 1,2\ldots,n, \tag{51}$$

$$w_i' = x_i w_{i+1} + x_{i-1} w_{i-1} + w_i - x_i w_i - x_{i-1} w_i,\ \ i = 1,2\ldots,n, \tag{52}$$

assuming $x_0 = x_n = 0$, $d_0 = d_{n+1} = 0$ and $w_0 = w_{n+1} = 0$.

CQM model

Similarly to Section 7, the definition of T_i from formula (49) takes the form:

$$T_i' \geq C_i' - d_i' = \sum_{j=1}^{i} p_j + x_i(p_{i+1} - p_i) - (x_i d_{i+1} + x_{i-1} d_{i-1} + d_i - x_i d_i - x_{i-1} d_i),\ i = 1,2\ldots,n. \tag{53}$$

Therefore, the problem of searching the Fibonacci neighborhood, as a CQM problem, takes the form:

$$\min_{\substack{\mathbf{x} \in \{0,1\}^{n-1} \\ T_i', w_i' \in \mathbb{N}^+ \\ 1 \leq i \leq n}} \sum_{i=1}^{n} w_i' T_i' = \min_{\substack{\mathbf{x} \in \{0,1\}^{n-1} \\ T_i' \in \mathbb{N}^+ \\ 1 \leq i \leq n}} \sum_{i=1}^{n} (x_i w_{i+1} + x_{i-1} w_{i-1} + w_i - x_i w_i - x_{i-1} w_i)\, T_i' \tag{54}$$

subject to:

$$x_i + x_{i+1} \leq 1,\ i = 1,2\ldots,n-1, \tag{55}$$

$$T_i' \geq \sum_{j=1}^{i} p_j + x_i(p_{i+1} - p_i) - (x_i d_{i+1} + x_{i-1} d_{i-1} + d_i - x_i d_i - x_{i-1} d_i),\ i = 1,2\ldots,n, \tag{56}$$

$$T_i' \geq 0,\ i = 1,2\ldots,n, \tag{57}$$

$$x_0 = x_n = d_0 = d_{n+1} = w_0 = w_{n+1} = 0. \tag{58}$$

This notation allows us to formulate the neighborhood search problem in the QUBO form.

QUBO model

Similarly to Section 7, we convert the integer variables T_i' and s_i to binary variables (note: we need to know their range to determine how many bits we need). Finally, we obtain the QUBO model:

$$\min_{\substack{x_i, v_i, \sigma_{il}, z_{il} \in \{0,1\} \\ 1 \leq l \leq \lceil \log_2 \hat{T} \rceil \\ 0 \leq i \leq n+1}} \left[\sum_{i=1}^{n} (x_i w_{i+1} + x_{i-1} w_{i-1} + w_i - x_i w_i - x_{i-1} w_i) \left(\sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l z_{il} \right) + \hat{K} \sum_{i=1}^{n} \left(\sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l z_{il} - \sum_{l=0}^{\lceil \log_2 \hat{T} \rceil} 2^l \sigma_{il} - \right. \right.$$

$$\left. \left. - \sum_{j=1}^{i} p_j - x_i(p_{i+1} - p_i) + x_i d_{i+1} + x_{i-1} d_{i-1} + d_i - x_i d_i - x_{i-1} d_i \right)^2 + \hat{K}(x_0 + x_n) + \hat{K} \sum_{i=1}^{n-1} (x_i + x_{i+1} + v_i - 1)^2 \right], \tag{59}$$

where constants $\hat{T} = \sum_{j=1}^{n} p_j - \min_{1 \leq i \leq n}\{d_i\}$, $\hat{K} = \sum_{j=1}^{n} w_j \hat{T}$.

Benefits of replacing a matrix with a vector

In this work, we considered Fibonacci neighborhoods ($\overline{API}$) generated by compositions of independent API moves. The elements of these neighborhoods (n-element permutations) were represented by

1. n^2-element binary matrices, whose set has 2^{n^2} elements,
2. $(n-1)$-element binary sequences, whose set has 2^{n-1} elements.

We examined the effect of the solution representation method on the number of elements in the set of all solutions and the proportion of this set that the Fibonacci neighborhood constitutes. Comparative results are presented in Table 1. For the matrix and vector representations, the individual matrix columns denote: n – permutation sizes, F_n – Fibonacci numbers (size of the $\overline{API}$ neighborhood), the potential number of all elements (columns 3 and 5) of a given representation, and the fraction sizes ($frac_M$ for the matrix representation and $frac_V$ for the vector representation) in percentages (columns 4 and 6). The fraction size is the ratio of the number of elements of the Fibonacci neighborhood to the number of all solutions for a given representation. For the estimations, we assumed the asymptotic approximation $F_n \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} = \Theta(1.618^n)$.

As expected, for the matrix representation, as the permutation size (column n) increases, the ratio of the number of elements in the Fibonacci neighborhood (column F_n) to all solutions (column 2^{n^2}) decreases rapidly. Approximately, this ratio is given by:

$$frac_M = \Theta\left(\frac{1.618^n}{2^{n^2}}\right) = \Theta\left(\frac{0.809^n}{2^{n(n-1)}}\right).$$

In turn, for the vector representation,

$$frac_V = \Theta\left(\frac{1.618^n}{2^{n-1}}\right) = \Theta(0.809^n).$$

Hence,

$$frac_M = \Theta\left(\frac{1}{2^{n(n-1)}} frac_V\right).$$

From the above, it follows that the fraction of feasible solutions (in the sense of belonging to the Fibonacci neighborhood) for the matrix representation rapidly decreases relative to the fraction of feasible solutions for the vector representation. This quotient is asymptotically $1/(2^{n(n-1)})$.

Table 1. Comparison of the fractions of feasible solutions for matrix and vector representations

	Matrix representation			Vector representation	
n	F_n	2^{n^2}	$frac_M[\%]$	2^{n-1}	$frac_V[\%]$
1	1	2.00	$5.00 \cdot 10^{-1}$	1	1.00
2	2	$1.60 \cdot 10^{1}$	$1.25 \cdot 10^{-1}$	2	1.00
3	3	$5.12 \cdot 10^{2}$	$5.86 \cdot 10^{-3}$	4	$7.50 \cdot 10^{-1}$
4	5	$6.55 \cdot 10^{4}$	$7.63 \cdot 10^{-5}$	8	$6.25 \cdot 10^{-1}$
5	8	$3.36 \cdot 10^{7}$	$2.38 \cdot 10^{-7}$	16	$5.00 \cdot 10^{-1}$
6	13	$6.87 \cdot 10^{10}$	$1.89 \cdot 10^{-10}$	32	$4.06 \cdot 10^{-1}$
7	21	$5.63 \cdot 10^{14}$	$3.73 \cdot 10^{-14}$	64	$3.28 \cdot 10^{-1}$
8	34	$1.84 \cdot 10^{19}$	$1.84 \cdot 10^{-18}$	128	$2.66 \cdot 10^{-1}$
9	55	$2.42 \cdot 10^{24}$	$2.27 \cdot 10^{-23}$	256	$2.15 \cdot 10^{-1}$
10	89	$1.27 \cdot 10^{30}$	$7.02 \cdot 10^{-29}$	512	$1.74 \cdot 10^{-1}$
11	144	$2.66 \cdot 10^{36}$	$5.41 \cdot 10^{-35}$	1024	$1.41 \cdot 10^{-1}$
12	233	$2.23 \cdot 10^{43}$	$1.04 \cdot 10^{-41}$	2048	$1.14 \cdot 10^{-1}$
13	377	$7.48 \cdot 10^{50}$	$5.04 \cdot 10^{-49}$	4096	$9.20 \cdot 10^{-2}$
14	610	$1.00 \cdot 10^{59}$	$6.07 \cdot 10^{-57}$	8192	$7.45 \cdot 10^{-2}$
15	987	$5.39 \cdot 10^{67}$	$1.83 \cdot 10^{-65}$	16384	$6.02 \cdot 10^{-2}$
16	1597	$1.16 \cdot 10^{77}$	$1.38 \cdot 10^{-74}$	32768	$4.87 \cdot 10^{-2}$

The exact values of both fraction coefficients are presented in columns 3 and 5 of Table 1. For $n = 5$, the number of all elements for the matrix representation is $33,600,000$, and the Fibonacci neighborhood elements constitute 0.00000023% of this set. For the vector representation, these are 16 and 0.5%, respectively. In turn, for $n = 10$, the number of all elements for the matrix representation is $1.27 \cdot 10^{30}$, and the Fibonacci neighborhood elements constitute only $7.02 \cdot 10^{-29}\%$ of this set. For the vector representation, these are 512 and 0.174%, respectively.

We strive to use exponential neighborhoods in local search algorithms, which will simultaneously constitute a significant portion of the entire solution space. Table 1 clearly shows that the proposed representation of permutations by binary vectors fully satisfies both conditions (postulates) when used to generate Fibonacci neighborhoods.

Computational experiments on quantum annealer

Computational experiments were performed on a D-Wave quantum computer – a quantum annealer, using both the hybrid solver `LeapHybridCQMSampler`, which uses a CPU and QPU, and the native `Advantage_system6.4` (Pegasus topology, 5612 qubits) and `Advantage2_system1` (Zephyr topology, 4578 qubits) that perform pure quantum annealing.

Generation of experimental data

The method described in this paper for solving the TWT task scheduling problem on a quantum computer was tested on a representative sample of randomly generated data. The data preparation method was presented in detail in the paper by Potts and Van Wassenhove[31]. Each example consists of n triples (w_i, p_i, d_i), where n is the number of tasks, and w_i, p_i, and d_i are, respectively: the weight of the cost function, the execution time, and the desired completion date for the i-th task. The weights of the cost function and the execution times are realizations of uniformly distributed random variables on the intervals [1,10] and [1,100], respectively. The values of the desired task completion deadlines depend on two parameters: *RDD* (relative of due date) and TF (average tardiness factor), which take values from the set $\{0.2; 0.4; 0.6; 0.8; 1.0\}$. These deadlines are realizations of a random variable with a uniform distribution on the interval $[P(1-TF-RDD/2), P(1-TF+RDD/2)]$, where $P = \sum_{i=1}^{n} p_i$. The data were drawn for various values of the *RDD* and TF parameters. For small values of these parameters, the examples are considered easy, while for large values, close to 1.0, they are considered difficult.

Computational results

Table 2 shows the theoretical number of qubits required to represent elements of the Fibonacci neighborhood in the QUBO model using matrices (columns 2–5) and vectors (columns 6–9), respectively, for different numbers of tasks. The subsequent columns denote the size of the task n (column 1), the number of binary variables in the representation (column 2), the number of remaining variables (i.e., t and s, column 3), the number of bits needed to represent these integer variables (column 4), and the total number of variables (column 5) – all for the matrix representation. The next four columns, 6–9, correspond to columns 2–5 for the vector representation. In both representations, the number of integer variables (i.e., the auxiliary variables s and t,, columns 3 and 7) is, of course, the same. Therefore, the number of binary variables representing them is the same, columns 5 and 8. However, a huge difference occurs in the case of decision variables (columns 2 and 7). For example, for $n = 10$, the matrix representation requires 100 binary variables (qubits), while the vector representation requires only 9 (over 11 times fewer). In contrast, for $n = 300$, this ratio is significantly larger, reaching over 300 times more. In general, for a vector, the logarithm growth is linear, while for a matrix, it is quadratic.

In turn, Table 3 contains, only for the vector representation, theoretical and real numbers of qubits in the QUBO models (the matrix representation was not possible for embedding in practice) on the D-Wave quantum annealer, the `Advantage_system6.4` (Pegasus) and `Advantage2_system1` (Zephyr) models with a maximum degree of 15 and 20, respectively. The first column in Table 3 contains the number of permutation elements, the second column contains the number of binary decision variables, and the third column contains the number of integer auxiliary variables. The fourth column shows the number of qubits per integer variable (q/var, depending on the range of $\hat{T}$ variables). The next, fifth column shows the number of binary variables (qubits) representing the additional variables from column 4. Column 6 shows the total number of binary variables. Column 7 shows the actual number of binary variables used to represent the model, taking into account the actual range of variables t and s, but excluding the additional variables needed within the chains in the individual computer topologies. The next four columns show, for the Pegas and Zephyr topologies, the number of additional qubits used within the chains and the total actual number of qubits used in the actual computation after embedding.

The actual number of qubits used by the quantum annealer (columns 9 and 11) is significantly larger than the theoretical number (column 6), which results from the topology of the individual processors (Pegasus, Zephyr), which do not have any one-to-one qubit connections. Consequently, additional qubits must be used to enforce the connections required by the Hamiltonian using specific 'chains' of working qubits, see Boothby et al.[32]. Furthermore, the Zephyr topology has a higher degree of vertices in the connection graph, which translates into a smaller number of qubits required (by several percent). Figure 2 shows the embedding of the QUBO model of a Fibonacci neighborhood for the Pegasus topology (maximum vertex degree $deg = 15$) and Zephyr topologies ($deg = 20$). White lines represent qubit entanglement (physical couplers) for $n = 3$. It can be seen that the more modern Zephyr topology requires fewer qubits compared to the earlier Pegasus topology, as the connection graph has a higher degree of vertices, which requires the use of more additional qubits to create intermediary chains.

Tables 4 and 5 show the exact values of the following parameters for native quantum annealing, for $n = 3, 4$, and 5 (the names are self-descriptive): num_feasible – number of feasible solutions, num_infeasible – number of infeasible solutions, qpu_anneal_time_per_sample – in microseconds $[\mu s]$, chain_strength, topology, num_reads, lagrange_multiplier, bqm_linear_min – minimum value of the QUBO coefficients before scaling for the vector representation, bqm_linear_max – maximum value of the QUBO coefficients before scaling for the vector representation, bqm_quadratic_min – minimum value of the QUBO coefficients before scaling for the matrix representation, bqm_quadratic_max – maximum value

of the QUBO coefficients before scaling for the matrix representation, scaled_bqm_linear_min, scaled_bqm_linear_max, scaled_bqm_quadratic_mins, scaled_bqm_quadratic_max (last four after scaling to the hardware qubit range).

Such solver performance analysis allows us to precisely trace the mechanism of operation of the proposed methodology for various topologies of quantum processors (QPUs). We report the solution quality obtained from the native QPU (objective value, number of feasible and infeasible solutions, and optimality gap) together with the annealing parameters (num reads, annealing time, and chain strength). Average constraint satisfaction rate is 4.012%, but it varies greatly (depending on the instance) ranging from 0.025% to 18.737%.

The computational results of the Fibonacci neighborhood search algorithms for both solution representations are presented in Tables 6 (matrix representation) and 7 (vector representation). The D-Wave hybrid model `LeapHybridCQMSampler` was used to generate the neighborhood elements. The individual columns of both tables represent the name of the test instance, the determined solution, the determined objective function value f, the objective function value of the initial solution (natural permutation) f_{nat}, the optimal neighborhood' solution f_{opt}, QPU computation time, D-Wave charging time, and actual runtime. As expected, noticeable differences in the results for both solution representations appear only for larger example sizes, for $n = 30$, 40, and 50. According to theoretical analysis and experiments, the vector representation not only requires significantly fewer qubits compared to the matrix representation (thus enabling the solution of much larger examples – embedding on modern quantum machines is possible), but also yields better solutions for this representation. The relative improvement of the solution (compared to the matrix representation) is: 1.98% for $n = 30$, 1.07% for $n = 40$ and 0.83% for $n = 50$.

Table 2. Summary of the number of variables and qubits for matrix and vector representations. Assuming $\lceil(\log_2 \sum_{i=1}^{n} p_i)\rceil$ bits per constant $\hat{T}$.

	matrix representation				vector representation			
n	#binary variables (x)	#other integer variables (t,s)	t,s integers as binary	total	#binary variables (x)	#other integer variables (t,s)	t,s integers as binary	total
2	4	4	32	36	1	4	32	33
3	9	6	54	63	2	6	54	56
4	16	8	72	88	3	8	72	75
5	25	10	90	115	4	10	90	94
6	36	12	120	156	5	12	120	125
7	49	14	140	189	6	14	140	146
8	64	16	160	224	7	16	160	167
9	81	18	180	261	8	18	180	188
10	100	20	200	300	9	20	200	209
20	400	40	440	840	19	40	440	459
30	900	60	720	1620	29	60	720	749
40	1600	80	960	2560	39	80	960	999
50	2500	100	1300	3800	49	100	1300	1349
60	3600	120	1560	5160	59	120	1560	1619
70	4900	140	1820	6720	69	140	1820	1889
80	6400	160	2080	8480	79	160	2080	2159
90	8100	180	2520	10620	89	180	2520	2609
100	10000	200	2800	12800	99	200	2800	2899
200	40000	400	6000	46000	199	400	6000	6199
300	90000	600	9000	99000	299	600	9000	9299

Table 3. Summary of the number of variables and qubits for real embedding in the D-Wave Pegasus (graph degree $deg = 15$, each qubit is coupled to 15 different qubits) and Zephyr ($deg = 20$) topologies, for vector representation. Native quantum annealing (`DWaveSampler`) was used.

	theoretical qubits					real binary	Pegasus		Zephyr	
n	x with sentinels x_0, x_n	t,s	q/var	t,s qubits	total	variables	add. qubits	total qubits	add. qubits	total qubits
3*	4	6	9	54	58	56	102	158	94	150
4	5	8	9	72	77	67	103	170	85	152
5	6	10	9	90	96	68	70	138	58	126
6	7	12	10	120	127	95	150	245	118	213
7	8	14	10	140	148	106	141	247	109	215

* visualized in Figure 2

Table 4. Parameters of native quantum annealing for small n (part 1)

n	π_{min}	f_{min}	num_ feasible	num_ infeasible	qpu_anneal_ time_per_sample	chain_ strength	topology	num_ reads	lagrange_ multiplier
3	[1, 3, 2]	46	136	864	20	37635339.851600	pegasus	1000	5056534800
3	[1, 3, 2]	46	121	1879	20	37635339.851600	pegasus	2000	5056534800
3	[1, 3, 2]	46	70	2930	20	37635339.851600	pegasus	3000	5056534800
3	[1, 3, 2]	46	470	3530	20	37635339.851600	pegasus	4000	5056534800
3	[1, 3, 2]	46	15	985	20	37635339.851600	zephyr	1000	5056534800
3	[1, 3, 2]	46	4	1996	20	37635339.851600	zephyr	2000	5056534800
3	[1, 3, 2]	46	129	2871	20	37635339.851600	zephyr	3000	5056534800
3	[1, 3, 2]	46	290	3710	20	37635339.851600	zephyr	4000	5056534800
3	[1, 3, 2]	46	789	4211	20	37635339.851600	zephyr	5000	5056534800
4	[2, 1, 4, 3]	84	4	996	20	80858528.707551	pegasus	1000	11527126800
4	[1, 2, 4, 3]	84	42	1958	20	80858528.707551	pegasus	2000	11527126800
4	[2, 1, 4, 3]	84	26	2974	20	80858528.707551	pegasus	3000	11527126800
4	[1, 2, 4, 3]	84	123	3877	20	80858528.707551	pegasus	4000	11527126800
4	[1, 2, 4, 3]	84	9	991	20	80858528.707551	zephyr	1000	11527126800
4	[2, 1, 4, 3]	84	6	1994	20	80858528.707551	zephyr	2000	11527126800
4	[2, 1, 4, 3]	84	95	2905	20	80858528.707551	zephyr	3000	11527126800
4	[2, 1, 4, 3]	84	1	3999	20	80858528.707551	zephyr	4000	11527126800
4	[2, 1, 4, 3]	84	29	4971	20	80858528.707551	zephyr	5000	11527126800
5	[1, 2, 3, 5, 4]	44	45	955	20	43440207.709942	pegasus	1000	5689958400
5	[1, 3, 2, 5, 4]	44	24	1976	20	43440207.709942	pegasus	2000	5689958400
5	[1, 3, 2, 5, 4]	44	142	2858	20	43440207.709942	pegasus	3000	5689958400
5	[2, 1, 3, 5, 4]	44	18	3982	20	43440207.709942	pegasus	4000	5689958400
5	[1, 2, 3, 5, 4]	44	1	999	20	43440207.709942	zephyr	1000	5689958400
5	[1, 3, 2, 5, 4]	44	82	1918	20	43440207.709942	zephyr	2000	5689958400
5	[1, 3, 2, 5, 4]	44	55	2945	20	43440207.709942	zephyr	3000	5689958400
5	[2, 1, 3, 5, 4]	44	109	3891	20	43440207.709942	zephyr	4000	5689958400
5	[2, 1, 3, 5, 4]	44	58	4942	20	43440207.709942	zephyr	5000	5689958400

Table 5. Parameters of native quantum annealing for small n (part 2)

n	bqm_linear_ min	bqm_linear_ max	bqm_quadratic_ min	bqm_quadratic_ max	scaled_bqm_ linear_min	scaled_bqm_ linear_max	scaled_bqm_ quadratic_mins	scaled_bqm_ quadratic_max
3	−319768470	505653480	−210240000	122780160	−0.837366	1.324137	−0.550548	0.321520
3	−319768470	505653480	−210240000	122780160	−1.037878	1.641208	−0.682380	0.398510
3	−319768470	505653480	−210240000	122780160	−0.669811	1.059180	−0.440384	0.257184
3	−319768470	505653480	−210240000	122780160	−0.862750	1.364276	−0.567237	0.331267
3	−319768470	505653480	−210240000	122780160	−0.773115	1.222536	−0.508304	0.296850
3	−319768470	505653480	−210240000	122780160	−1.007012	1.592400	−0.662086	0.386658
3	−319768470	505653480	−210240000	122780160	−0.809255	1.279684	−0.532066	0.310726
3	−319768470	505653480	−210240000	122780160	−0.883743	1.397472	−0.581039	0.339327
3	−319768470	505653480	−210240000	122780160	−0.840706	1.329418	−0.552744	0.322802
4	−578374440	1152712680	−385323428	295536640	−0.636503	1.268565	−0.424050	0.325239
4	−578374440	1152712680	−385323428	295536640	−0.754322	1.503379	−0.502543	0.385442
4	−578374440	1152712680	−385323428	295536640	−0.713459	1.421939	−0.475319	0.364562
4	−578374440	1152712680	−385323428	295536640	−0.766566	1.527783	−0.510700	0.391699
4	−578374440	1152712680	−385323428	295536640	−0.806465	1.607303	−0.537282	0.412086
4	−578374440	1152712680	−385323428	295536640	−0.742762	1.480341	−0.494842	0.379535
4	−578374440	1152712680	−385323428	295536640	−0.729287	1.453486	−0.485864	0.372650
4	−578374440	1152712680	−385323428	295536640	−0.626960	1.249545	−0.417692	0.320363
4	−578374440	1152712680	−385323428	295536640	−0.790629	1.575740	−0.526731	0.403994
5	−325140480	568995840	−242851840	196689920	−0.651822	1.140689	−0.486855	0.394312
5	−325140480	568995840	−242851840	196689920	−0.701884	1.228296	−0.524246	0.424596
5	−325140480	568995840	−242851840	196689920	−0.640339	1.120593	−0.478278	0.387365
5	−325140480	568995840	−242851840	196689920	−0.693222	1.213138	−0.517777	0.419356
5	−325140480	568995840	−242851840	196689920	−0.698463	1.222311	−0.521692	0.422527
5	−325140480	568995840	−242851840	196689920	−0.668509	1.169890	−0.499318	0.404406
5	−325140480	568995840	−242851840	196689920	−0.794914	1.391100	−0.593732	0.480874

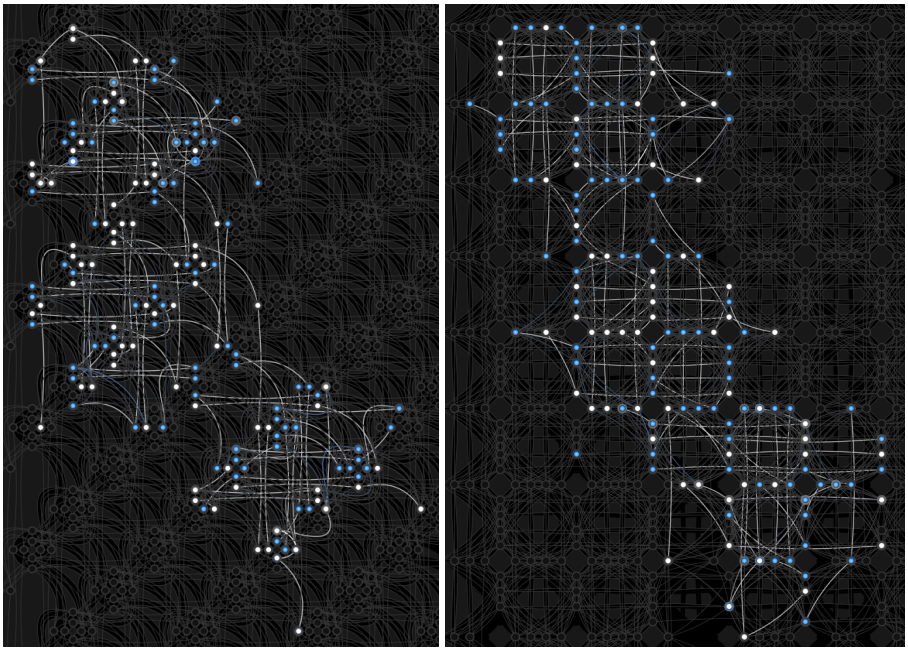

Figure 2. Comparison of qubits embedding of Fibonacci neighborhood generation for Pegasus (left figure) and Zephyr (right one) D-Wave QPUs architectures for $n = 3$. White dots represent spin -1, and blue dots $+1$.

Table 5. Parameters of native quantum annealing for small n (part 2)

n	bqm_linear_ min	bqm_linear_ max	bqm_quadratic_ min	bqm_quadratic_ max	scaled_bqm_ linear_min	scaled_bqm_ linear_max	scaled_bqm_ quadratic_mins	scaled_bqm_ quadratic_max
5	−325140480	568995840	−242851840	196689920	−0.620064	1.085113	−0.463135	0.375101
5	−325140480	568995840	−242851840	196689920	−0.683649	1.196386	−0.510627	0.413565

Table 6. Results for matrix representation for `LeapHybridCQMSampler`

Instance	Permutation	f	f_{nat}	f_{opt}	QPU time [μs]	Charge time [μs]	Run time [μs]
wt5_001	[2,1,3,5,4]	44	130	44	34741	5000000	5209665
wt6_001	[1, 3, 2, 4, 5, 6]	198	198	198	34751	5000000	5186681
wt7_001	[2, 1, 3, 4, 5, 6, 7]	693	693	693	<1	5000000	5272544
wt8_001	[2, 1, 3, 4, 5, 7, 6, 8]	435	948	435	34749	5000000	5249845
wt9_001	[1, 3, 2, 4, 5, 6, 8, 7, 9]	368	400	368	34759	5000000	5229812
wt10_001	[1, 3, 2, 5, 4, 7, 6, 8, 10, 9]	1683	1865	1683	34756	5000000	5308891
wt20_001	[1, 3, 2, 5, 4, 6, 8, 7, 9, 10, 11, 12, 14, 13, 15, 16, 18, 17, 19, 20]	2258	2625	2258	34752	5000000	5272628
wt30_001	[1, 2, 3, 5, 4, 6, 7, 9, 8, 10, 12, 11, 14, 13, 15, 16, 18, 17, 20, 19, 21, 23, 22, 24, 25, 26, 27, 29, 28, 30]	2993	3966	2935	34762	5000000	5122579
wt40_001	[2, 1, 3, 4, 5, 6, 8, 7, 9, 10, 11, 12, 13, 14, 15, 17, 16, 18, 20, 19, 22, 21, 24, 23, 25, 26, 27, 28, 29, 31, 30, 32, 34, 33, 35, 36, 38, 37, 40, 39]	15589	16672	15424	34764	5000000	5003015
wt50_001	[1, 2, 4, 3, 5, 6, 7, 9, 8, 10, 11, 13, 12, 15, 14, 16, 18, 17, 19, 20, 22, 21, 23, 24, 25, 27, 26, 29, 28, 30, 31, 33, 32, 34, 35, 36, 38, 37, 39, 40, 41, 43, 42, 44, 46, 45, 47, 49, 48, 50]	21861	22931	21679	69419	4803305	4803305

Theoretical results presented in Table 2, as well as the experimental results presented in Tables 3–7, clearly indicate that the vector representation requires significantly fewer qubits to generate neighborhoods than the classical matrix representation of solutions. Of course, both formulas (matrix- and vector-based) are mathematically equivalent, defining the same set of tractable solutions—the Fibonacci neighborhood. The fact that the vector representation yields better solutions stems from the (experimental) difference in solver performance.

Table 10 presents the results of a sensitivity analysis of the coefficient $\hat{K}$ for the number of reads of 4000 and $n = 5$, for the Zephyr and Pegasus topologies. The coefficient $\varepsilon \cdot \hat{K}$ was scaled for $\varepsilon = 0.5, 0.75, \ldots, 1.75$. The subsequent columns in Table 10 denote: topology type, solution π_{min}, number of feasible solutions num_feas, number of infeasible solutions num_infeas, and the coefficient ε. The best results for the Pegasus topology were obtained for $\varepsilon = 1.00$, and for the Zephyr topology for

Table 7. Vector representation results for `LeapHybridCQMSampler`

Instance	Permutation	f	f_{nat}	f_{opt}	QPU time [μs]	Charge time [μs]	Run time [μs]
wt5_001	[1, 3, 2, 5, 4]	44	130	44	34706	5000000	5169348
wt6_001	[1, 3, 2, 5, 4, 6]	198	198	198	34722	5000000	5174075
wt7_001	[1, 3, 2, 5, 4, 6, 7]	693	693	693	34706	5000000	5157446
wt8_001	[2, 1, 4, 3, 5, 7, 6, 8]	435	948	435	34723	5000000	5235347
wt9_001	[2, 1, 4, 3, 6, 5, 8, 7, 9]	368	400	368	34706	5000000	5161768
wt10_001	[1, 2, 3, 5, 4, 7, 6, 8, 10, 9]	1683	1865	1683	34721	5000000	5170861
wt20_001	[1, 2, 4, 3, 5, 7, 6, 9, 8, 10, 12, 11, 13, 14, 15, 16, 18, 17, 19, 20]	2258	2625	2258	34744	5000000	5200120
wt30_001	[0, 1, 3, 2, 4, 6, 5, 7, 9, 8, 11, 10, 13, 12, 15, 14, 16, 17, 18, 19, 20, 21, 23, 22, 24, 26, 25, 27, 29, 28, 30]	**2935**	3966	2935	34720	5000000	5187132
wt40_001	[2, 1, 4, 3, 5, 6, 8, 7, 10, 9, 12, 11, 13, 15, 14, 16, 17, 19, 18, 21, 20, 22, 23, 25, 24, 26, 27, 28, 29, 31, 30, 33, 32, 34, 35, 36, 38, 37, 40, 39]	**15424**	16672	15424	34765	5000000	5235415
wt50_001	[1, 2, 3, 5, 4, 7, 6, 9, 8, 11, 10, 12, 13, 14, 16, 15, 18, 17, 20, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 32, 34, 35, 37, 36, 38, 39, 41, 40, 43, 42, 44, 46, 45, 47, 49, 48, 50]	**21679**	22931	21679	69510	5000000	5192429

Table 8. Comparison of the efficiency of Fibonacci neighborhoods and QUBO-modeled adjacent swaps (FAPI) built into local search on D-Wave LeapHybrid

	Fibonacci neighborhood (vector representation)				FAPI neighborhood			
instance	wall clock time [s]	f_{start}	f_{end}	PRD [%]	wall clock time [s]	f_{start}	f_{end}	PRD [%]
wt10_001	19.746	1865	1439	22.84	46.44	1865	1439	22.84
wt20_001	28.901	2625	1856	29.30	38.04	2625	1856	29.30
wt30_001	48.805	3966	1624	59.05	67.522	3966	1624	59.05
wt40_001	118.740	16672	7002	58.00	336.838	16672	7002	58.00
wt50_001	191.877	22931	7791	60.79	526.136	22931	8526	62.82
average	**81.614**			**46.00**	**202.995**			**46.40**

$\varepsilon = 1.25$. This indicates that the coefficient value was chosen correctly.

From an implementation perspective, for native runs using `DWaveSampler`, the problem was defined as `ConstrainedQuadraticModel` and then converted to `BinaryQuadraticModel` using the `cqm_to_bqm` function from the `dimod` package. The default value of `lagrange_multiplier` was used. According to the documentation, the penalty strength used when converting constraints into penalty models is default equal to 10x the largest bias in the objective. The values of the penalty factor, scaling factor, and ranges of QUBO coefficients before and after scaling were added to Tables 11 and 12. During native calculations, the admissibility of solutions was checked using the `check_feasible` function defined within `ConstrainedQuadraticModel` after inverting the BQM problem to CQM using the `invert` function from the `dimod` library. Tables 11 and 12 do not contain results for $n = 6$ and $n = 7$, because it was not possible to perform the aforementioned inversion due to an error related to out-of-bounds variables. BQM^{l}_{min}, BQM^{l}_{max}, $BQM^{l_s}_{min}$ (after scalling) and $BQM^{l_s}_{max}$ (after scalling) refers to vector (linear) representation QUBO linear coefficients before and after scalling, and BQM^{q}_{min}, BQM^{q}_{max}, $BQM^{q_s}_{min}$ (after scalling)and $BQM^{q_s}_{max}$ (after scalling) refers to matrix representation QUBO quadratic coefficents before and after scalling.

We also examined the effectiveness of using the Fibonacci neighborhood in the local search algorithm. Tables 8 and 9 present the results of a comparison of the effectiveness of two types of neighborhoods: the new Fibonacci neighborhood proposed in this paper versus the classical API neighborhood based on a single swap of adjacent elements of a permutation.

To ensure the comparisons were conducted in the same computational environments, the method for generating the classic API neighborhood was adapted to the QUBO requirements by adding the constraint $\sum_{i=1}^{n-1} x_i = 1$, instead of (55) to the Fibonacci neighborhood model defined by the formulas (54)–(58), making this neighborhood (abbreviated as FAPI) identical to the API neighborhood (generating a single element of the FAPI neighborhood involved executing just one API call). Similarly, the Fibonacci neighborhood for CPU computations was determined using dynamic programming, as described earlier in the section 'Dynamic Programming for Fibonacci Neighborhoods'. The neighborhoods were incorporated into the classic simple *descent search* algorithm. This algorithm iteratively searches the neighborhood and moves to the best found neighborhood element until no further improvement is possible (i.e., it reaches a local extreme). The natural permutation was assumed as the starting solution.

Both tables 8 and 9 present the following columns: computing wall clock time performed to reach a local extreme (columns

Table 9. Comparison of the efficiency of Fibonacci neighborhoods and classic adjacent swaps (API) built into local search on CPU

	Fibonacci neighborhood (vector representation)				classic API neighborhood			
instance	wall clock time [s]	f_{start}	f_{end}	PRD [%]	wall clock time [s]	f_{start}	f_{end}	PRD [%]
wt10_001	0.021	1865	1439	22.84	0.001	1865	1439	22.84
wt20_001	0.033	2625	1856	29.30	0.001	2625	1856	29.30
wt30_001	0.070	3966	1624	59.05	0.001	3966	1624	59.05
wt40_001	0.190	16672	7002	58.00	0.013	16672	7002	58.00
wt50_001	0.292	22931	8685	62.12	0.021	22931	8526	62.82
average	**0.121**			**46.26**	**0.007**			**46.40**

Table 10. Experimental results for different $\hat{K}$ coefficients for Pegasus and Zephyr topologies

Topology	π_{min}	num_feas	num_infeas	ε
Pegasus	[1, 2, 3, 5, 4]	91	3909	0.50
	[1, 3, 2, 5, 4]	66	3934	0.75
	[1, 2, 3, 5, 4]	164	3836	1.00
	[2, 1, 3, 5, 4]	157	3843	1.25
	[2, 1, 3, 5, 4]	30	3970	1.50
	[2, 1, 3, 5, 4]	12	3988	1.75
Zephyr	[1, 2, 3, 5, 4]	3	3997	0.50
	[1, 2, 3, 5, 4]	90	3910	0.75
	[1, 3, 2, 5, 4]	109	3891	1.00
	[1, 3, 2, 5, 4]	186	3814	1.25
	[1, 3, 2, 5, 4]	34	3966	1.50
	[2, 1, 3, 5, 4]	22	3978	1.75

Table 11. QUBO linear coefficients before and after scaling – vector (linear) representation

n	topology	scalar	BQM^{l}_{min}	BQM^{l}_{max}	BQM^{ls}_{min}	BQM^{ls}_{max}
3	pegasus	3.0433009402787616e-09	-319768470.0	505653480.0	-0.973151685422501	1.538855711139228
4	pegasus	1.0302116964487435e-09	-578374440.0	1152712680.0	-0.595848113014992	1.1875380855807776
5	pegasus	2.5318368423950 75e-09	-325140480.0	568995840.0	-0.8232026462180191	1.4406046308815335
3	zephyr	2.340999628177 16e-09	-319768470.0	505653480.0	-0.7485778693727794	1.183734608666487
4	zephyr	8.944659265452236e-10	-578374440.0	1152712680.0	-0.5173362293646748	1.0310622153566278
5	zephyr	2.344678257646577e-09	-325140480.0	568995840.0	-0.7623498141367718	1.3341121747393505

Table 12. QUBO quadratic coefficients before and after scaling – matrix representation

n	topology	scalar	BQM^{q}_{min}	BQM^{q}_{max}	BQM^{qs}_{min}	BQM^{qs}_{max}
3	pegasus	3.0433009402787616e-09	-210240000.0	122780160.0	-0.6398235896842068	0.3736569763755768
4	pegasus	1.0302116964487435e-09	-385323428.0	295536640.0	-0.39696470244132526	0.3044653032571616
5	pegasus	2.531836842395075e-09	-242851840.0	196689920.0	-0.614861235755434	0.49798678598373997
3	zephyr	2.340999628177 16e-09	-210240000.0	122780160.0	-0.4921717618279662	0.28742830890753224
4	zephyr	8.944659265452236e-10	-385323428.0	295536640.0	-0.34465867704560177	0.26434745452566216
5	zephyr	2.344678257646577e-09	-242851840.0	196689920.0	-0.5694094290774654	0.46117457892224467

2 and 6, for the Fibonacci neighborhood and the API neighborhood, respectively), the starting and final values of the objective function f_{start} and f_{end}, for the Fibonacci neighborhood (columns 3 and 4) and the API neighborhood (columns 7 and 8), respectively. For both types of analyzed neighborhoods, the percent relative deviation of the starting and final solutions was determined: $PRD = (f_{start} - f_{end})/f_{start} \cdot 100\%$.

As can be seen in Table 8, with practically the same level of improvement of (the same) starting solutions, the Fibonacci neighborhood required a significantly smaller time if it is executed in D-Wave LeapHybrid quantum computing environment: on average 81.614 seconds for the Fibonacci neighborhood versus on average 202.995 seconds for the FAPI neighborhood. This confirms the effectiveness of the approach of using vector representation and multimoves in local search metaheuristics in quantum computing environment. The mean errors are marginally better for the Fibonacci neighborhood (46% vs. 46.40%).

In turn, Table 9 presents the results obtained in a classical CPU computing environment. Here, the Fibonacci neighborhood

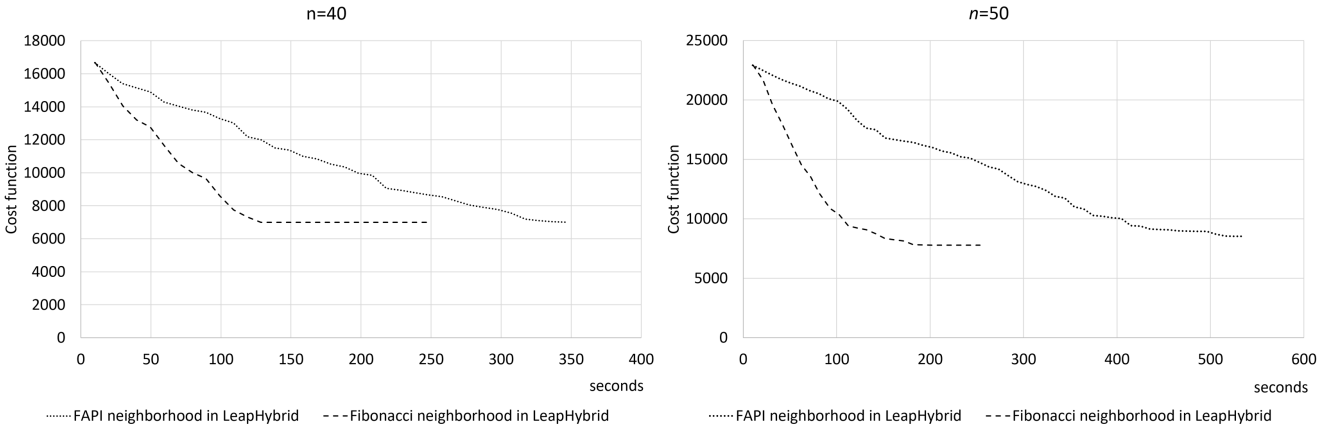

Figure 3. Convergence of the local search algorithm under the same `LeapHybridCQMSampler` neighborhood generation environment for $n = 40, 50$ cases.

does not achieve spectacular efficiency results (it has a significantly longer average computation time: 0.121 seconds vs. 0.007 seconds, and minimally lower average solution quality). This indicates that the methodology we propose is dedicated to calculations in quantum computing environments.

The obtained results are shown in Figure 3. The left graph presents results for the number of tasks $n = 40$, and the right graph is for $n = 50$. The dashed line in the graphs represents the FAPI environment, and the dotted line represents the API.) The calculations were performed using the LeapHybrid environment. Analysis of both graphs shows that the FAPI algorithm converges to the best solution much faster. This is a result of using a vector representation. This not only reduces the number of qubits but also allows us to obtain better solutions more quickly.

Table 13. Comparison of optimization results for different RDD and TF settings for matrix and vector representations in D-Wave Leap quantum computing environment.

			Best Energy		**QPU Access Time**		**Total Samples**		**Feasible Samples**	
Instance	**RDD**	**TF**	matrix	vector	matrix	vector	matrix	vector	matrix	vector
wt5_001.txt	0.2	0.2	44.0	44.0	34744	34722	125	122	85	90
wt5_011.txt	0.2	0.6	3656.0	3656.0	34743	34743	125	118	72	73
wt5_021.txt	0.2	1.0	1434.0	1434.0	34743	34707	123	113	75	90
wt5_026.txt	0.4	0.2	36.0	36.0	34743	34707	125	126	81	113
wt5_036.txt	0.4	0.6	372.0	372.0	34744	34722	125	122	78	98
wt5_046.txt	0.4	0.8	3783.0	3783.0	34744	34744	125	118	82	79
Partial Mean	-	-	1554.17	1554.17	34743.50	34724.17	124.67	119.83	78.83	90.50
Std. Deviation	-	-	1754.01	1754.01	0.55	16.41	0.82	4.49	4.79	14.15
wt6_001.txt	0.2	0.2	198.0	198.0	34751	34721	125	117	85	83
wt6_011.txt	0.2	0.6	1279.0	1279.0	34755	34721	125	118	75	93
wt6_021.txt	0.2	1.0	4985.0	4985.0	34751	34720	125	125	68	98
wt6_026.txt	0.4	0.2	111.0	111.0	34744	34724	125	114	88	76
wt6_036.txt	0.4	0.6	1664.0	1664.0	34742	34707	124	126	82	102
wt6_046.txt	0.4	0.8	4834.0	4834.0	34753	34722	125	124	77	99
Partial Mean	-	-	2178.50	2178.50	34749.33	34719.17	124.83	120.67	79.17	91.83
Std. Deviation	-	-	2199.95	2199.95	5.16	6.11	0.41	4.97	7.31	10.23
wt7_001.txt	0.2	0.2	693.0	693.0	34720	34707	125	126	74	100
wt7_011.txt	0.2	0.6	1060.0	1060.0	34748	34744	125	109	79	75
wt7_021.txt	0.2	1.0	5879.0	5879.0	34743	34722	125	118	87	87
wt7_026.txt	0.4	0.2	755.0	755.0	34745	34740	125	122	81	96
wt7_036.txt	0.4	0.6	4299.0	4299.0	34750	34723	124	116	81	78
wt7_046.txt	0.4	0.8	3596.0	3596.0	0	34753	124	115	76	76
Partial Mean	-	-	2713.67	2713.67	28951.00	34731.50	124.67	120.33	79.67	85.33
Std. Deviation	-	-	2076.79	2076.79	14183.04	17.05	0.52	6.09	4.55	10.76
wt8_001.txt	0.2	0.2	435.0	435.0	34745	34721	124	114	80	75
wt8_011.txt	0.2	0.6	3639.0	3639.0	34705	34741	123	123	81	99
wt8_021.txt	0.2	1.0	11231.0	11231.0	34760	34720	125	113	85	78
wt8_026.txt	0.4	0.2	114.0	114.0	34756	34722	124	114	83	80
wt8_036.txt	0.4	0.6	2244.0	2244.0	34755	34721	125	125	79	100

Continued on next page

Table 13 – Continued from previous page

Instance	RDD	TF	Best Energy		QPU Access Time		Total Samples		Feasible Samples	
			Matrix	Vector	Matrix	Vector	Matrix	Vector	Matrix	Vector
wt8_046.txt	0.4	0.8	5266.0	5266.0	34749	34741	125	118	84	81
Partial Mean	-	-	3821.50	3821.50	34745.00	34727.67	124.33	117.83	82.00	85.50
Std. Deviation	-	-	4116.38	4116.38	20.31	10.35	0.82	5.23	2.37	11.04
wt9_001.txt	0.2	0.2	368.0	368.0	34750	34722	125	126	78	103
wt9_011.txt	0.2	0.6	2908.0	2908.0	34755	34722	125	114	83	80
wt9_021.txt	0.2	1.0	10477.0	10477.0	34751	34721	125	111	83	66
wt9_026.txt	0.4	0.2	530.0	530.0	34754	34722	124	114	82	83
wt9_036.txt	0.4	0.6	2979.0	2979.0	34754	34746	111	114	62	72
wt9_046.txt	0.4	0.8	8268.0	8255.0	34757	34743	125	123	86	89
Partial Mean	-	-	4255.00	4252.83	34753.50	34729.33	122.50	117.00	79.00	82.17
Std. Deviation	-	-	4177.19	4174.69	2.59	11.79	5.65	6.26	8.72	13.04
wt10_001.txt	0.2	0.2	1683.0	1683.0	34765	34746	114	115	61	81
wt10_011.txt	0.2	0.6	4715.0	4715.0	34758	34754	124	111	77	58
wt10_021.txt	0.2	1.0	5328.0	5328.0	34762	34741	123	117	75	81
wt10_026.txt	0.4	0.2	396.0	396.0	34759	34740	125	114	84	77
wt10_036.txt	0.4	0.6	7178.0	7178.0	34749	34745	124	126	76	89
wt10_046.txt	0.4	0.8	8072.0	8072.0	34755	34751	110	123	56	82
Partial Mean	-	-	4562.00	4562.00	34758.00	34746.17	120.00	117.67	71.50	78.00
Std. Deviation	-	-	3013.90	3013.90	5.59	5.49	6.32	5.85	10.67	10.55
wt20_001.txt	0.2	0.2	2258.0	2258.0	34744	34744	110	114	62	78
wt20_011.txt	0.2	0.6	20714.0	20582.0	34748	34765	117	124	73	95
wt20_021.txt	0.2	1.0	46036.0	45959.0	34758	34742	117	114	72	75
wt20_026.txt	0.4	0.2	2940.0	2940.0	34761	34721	114	118	66	84
wt20_036.txt	0.4	0.6	15170.0	15102.0	34750	34764	114	123	69	90
wt20_046.txt	0.4	0.8	50736.0	50736.0	34756	34755	124	114	80	76
Partial Mean	-	-	22975.67	22929.50	34752.83	34748.50	116.00	117.83	70.33	83.00
Std. Deviation	-	-	20971.93	20962.97	6.52	16.57	4.69	4.96	6.22	8.15
wt30_001.txt	0.2	0.2	2935.0	2935.0	34723	34743	114	123	63	86
wt30_011.txt	0.2	0.6	47518.0	47114.0	34722	34744	117	110	71	56
wt30_021.txt	0.2	1.0	148758.0	148225.0	34753	34755	117	123	73	83
wt30_026.txt	0.4	0.2	11582.0	11531.0	34721	34753	114	122	63	81
wt30_036.txt	0.4	0.6	46790.0	46593.0	34756	34755	117	122	68	80
wt30_046.txt	0.4	0.8	77897.0	77792.0	34742	34747	118	111	70	57
Partial Mean	-	-	55913.33	55698.33	34736.17	34749.50	116.17	118.50	68.00	73.83
Std. Deviation	-	-	52969.99	52802.72	16.22	5.50	1.72	6.09	4.20	13.59
wt40_001.txt	0.2	0.2	15424.0	15424.0	34762	34757	117	124	73	85
wt40_011.txt	0.2	0.6	76315.0	75795.0	34762	34765	110	117	61	77
wt40_021.txt	0.2	1.0	139375.0	138135.0	34764	34753	109	117	59	77
wt40_026.txt	0.4	0.2	11075.0	11077.0	34722	34764	118	110	75	60
wt40_036.txt	0.4	0.6	83431.0	83304.0	34739	0	94	123	49	81
wt40_046.txt	0.4	0.8	140709.0	140198.0	34756	0	111	123	64	81
Partial Mean	-	-	77721.50	77322.17	34750.83	23173.17	109.83	119.00	63.50	76.83
Std. Deviation	-	-	56790.65	56408.32	16.86	17949.86	8.52	5.40	9.59	8.77
wt50_001.txt	0.2	0.2	21679.0	21679.0	34762	34760	100	124	57	76
wt50_011.txt	0.2	0.6	133275.0	132768.0	34750	34751	106	108	49	51
wt50_021.txt	0.2	1.0	392032.0	390946.0	34722	34757	95	124	42	80
wt50_026.txt	0.4	0.2	7475.0	7390.0	34722	34751	107	100	64	57
wt50_036.txt	0.4	0.6	110607.0	110911.0	34765	34751	107	122	60	83
wt50_046.txt	0.4	0.8	335072.0	332976.0	34746	34755	106	124	58	82
Partial Mean	-	-	166690.00	166111.67	34744.50	34754.17	103.50	117.00	55.00	71.50
Std. Deviation	-	-	161103.06	160378.59	18.82	3.82	4.64	10.41	8.05	13.90
Mean	0.300	0.567	34238.53	34114.43	34168.47	33580.33	117.85	121.10	72.70	81.85
Std. Deviation	-	-	73341.54	73035.21	4485.91	6288.36	9.14	7.55	10.59	12.49

For smaller instances (`wt5` through `wt10`), both the matrix and vector representations produce identical or nearly identical energy scores (cost function value), indicating that both formulations perform exceptionally well on lower-dimensional problems. As the problem size grows (`wt20` to `wt50`), a minor divergence emerges where the vector formulation occasionally reports a slightly lower (better) energy mean (166111.67 vs. 166690.00 in the final block). The final mean also slightly favors vector (34114.43 vs. 34238.53), showing that the vector formulation provides a marginal advantage in optimization quality for larger instances, without taking into account the significant reduction in the required number of qubits.

Constraints satisfaction rate is also better for vector representation (67.5%: 81.85 feasible for 121.10 total samples for vector representation, versus 61.7%: 72.70 feasible for 117.85 total samples for matrix representation).

Summary

Considering the dynamic development of quantum computers and based on the presented results, it can be concluded that within the next few years, quantum computers will have significantly greater potential for solving discrete optimization problems than currently used silicon computers. Currently, the main barrier to the effective use of quantum computers (to be more efficient than contemporary silicon computers) is, on the one hand, the insufficient number of available qubits and, on the other, their excessively large (in terms of solution representation) use by existing methods.

The proposals for a new, more efficient solution (compared to the existing matrix-based solution) representation presented in this work inspire continued research on new dedicated methods for binary problem representation designed to generate specific subsets, e.g., neighborhoods in local search algorithms, for various representations of solutions to discrete optimization problems.

This work is a prelude to further research on methods for constructing algorithms that effectively utilize the capabilities offered by quantum computers in the context of searching exponential sets. In this work, we focused on the simplest case, the Fibonacci neighborhood, and we plan to extend the research to more complex sets of this type – based on compositions of arbitrary exchanges, including overlapping ones, which leads to sets of size corresponding to the Motzkin numbers.

Acknowledgements

The calculations were made at the Wroclaw Centre for Networking and Supercomputing (http://wcss.pl).

Author contributions statement

W.B.: writing – review & editing, writing – original draft, methodology, investigation, conceptualization. M.U.: visualization, validation, software, methodology, investigation, data curation. M.W. writing – review & editing, writing – original draft, validation, methodology, investigation, formal analysis, conceptualization. All authors reviewed the manuscript.

Data availability statement

Data were deposited into the repository database and are available at the following URLs: http://zasobynauki.pl/zasoby/74584 and http://zasobynauki.pl/zasoby/51561.

Funding Declaration

Funding: No Funding

References

1. Ionq industry-leading roadmap. https://www.ionq.com/roadmap. Accessed: 2026-04-30.
2. QuEra. Quera computing releases a groundbreaking roadmap for advanced error-corrected quantum computers, pioneering the next frontier in quantum innovation. https://www.quera.com/press-releases/quera-computing-releases-a-groundbreaking-roadmap-for-advanced-error-corrected-quantum-computers-pioneering-the-,next-frontier-in-quantum-innovation-0 (2024). Access: 2024-05-23.
3. IBM. Ibm quantum roadmap. https://www.ibm.com/roadmaps/quantum.pdf.
4. Azure, M. Microsoft quantum roadmap. https://quantum.microsoft.com/en-us/vision/quantum-roadmap.
5. Han, Z.-Q.-Z. *et al.* Preparation and characterisation of high-dimensional frequency-orbital angular momentum entangled states. *Sci. China Physics, Mech. & Astron.* **69**, 254211, DOI: 10.1007/s11433-025-2908-0 (2026).
6. Zhang, G.-Q., Quezada, L. F. & Dong, S.-H. Reentrant topological phases and entanglement scalings in moiré-modulated extended su-schrieffer-heeger model. *Sci. China Physics, Mech. & Astron.* **69**, 230314, DOI: 10.1007/s11433-025-2855-1 (2026).
7. Jing, R.-H. *et al.* Deterministic bidirectional hierarchical teleportation of an arbitrary high-dimensional multi-particle state with a partially entangled quantum channel. *The Eur. Phys. J. Plus* **139**, 894, DOI: 10.1140/epjp/s13360-024-05702-1 (2024).
8. Rinnooy Kan, A. H. G., Lageweg, B. J. & Lenstra, J. K. Minimizing total costs in one-machine scheduling. *Oper. Res.* **25**, 908–927 (1975).

9. Wodecki, M. A branch-and-bound parallel algorithm for single-machine total weighted tardiness problem. *Int. J. Adv. Manuf. Technol.* **37**, 996–1004, DOI: 10.1007/s00170-007-1023-y (2008).

10. Bożejko, W., Grabowski, J. & Wodecki, M. Block approach—tabu search algorithm for single machine total weighted tardiness problem. *Comput. & Ind. Eng.* **50**, 1–14, DOI: 10.1016/j.cie.2005.12.001 (2006).

11. Congram, R. K., Potts, C. N. & Van de Velde, S. L. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS J. on Comput.* **14**, 52–67 (2002).

12. Adamu, M. O. & Adewumi, A. O. A survey of single machine scheduling to minimize weighted number of tardy jobs. *J. Ind. Manag. Optim.* **10**, 219–241 (2013).

13. Martinelli, R., Mariano, F. & Martins, C. Single machine scheduling in make to order environments: A systematic review. *Comput. & Ind. Eng.* **169**, 108190, DOI: 10.1016/j.cie.2022.108190 (2022).

14. Koulamas, C. The single-machine total tardiness scheduling problem: Review and extensions. *Eur. J. Oper. Res.* **202**, 1–7 (2010).

15. Bożejko, W., Burduk, A., Pempera, J. & Wodecki, M. Optimal solving of a binary knapsack problem on a d-wave quantum machine and its implementation in production systems. *Annals Oper. Res.* DOI: 10.1007/s10479-024-06025-1 (2024).

16. Bożejko, W. *et al.* Optimal solving of a scheduling problem using quantum annealing metaheuristics on the D-wave quantum solver. *IEEE Transactions on Syst. Man, Cybern. Syst.* **55**, 196–208, DOI: 10.1109/TSMC.2024.3458873 (2025).

17. Bożejko, W., Pempera, J., Uchroński, M. & Wodecki, M. Distributed quantum annealing on D-wave for the single machine total weighted tardiness scheduling problem. In Groen, D. *et al.* (eds.) *Computational Science – ICCS 2022*, vol. 13353 of *LNCS*, 171–178 (Springer, 2022).

18. Bożejko, W., Pempera, J., Uchroński, M. & Wodecki, M. Quantum annealing-driven branch and bound for the single machine total weighted number of tardy jobs scheduling problem. *Futur. Gener. Comput. Syst.* **155**, 245–255, DOI: 10.1016/j.future.2024.02.016 (2024).

19. Bożejko, W., Uchroński, M. & Wodecki, M. Quantum annealing-driven branch and bound for the total weighted tardiness traveling salesman problem on the d-wave quantum computer. *Int. Transactions Oper. Res.* DOI: https://doi.org/10.1111/itor.70161. https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.70161.

20. Wang, Y. S. & Cheng, J. A quantum optimization algorithm for single machine total weighted tardiness minimization. In *2022 IEEE Integrated STEM Education Conference (ISEC)*, 255–255, DOI: 10.1109/ISEC54952.2022.10025146 (Princeton, NJ, USA, 2022).

21. Grange, C., Poss, M., Bourreau, E., T'kindt, V. & Ploton, O. Moderate exponential-time quantum dynamic programming across the subsets for scheduling problems. *Eur. J. Oper. Res.* **320**, 516–526, DOI: 10.1016/j.ejor.2024.09.005 (2025).

22. Werner, T. & Ullinger, F. Quantum algorithms for scheduling problems: A survey (2025). DOI: 10.13140/RG.2.2.13588.82560. Preprint.

23. Tahami, H. & Fakhravar, H. A literature review on combining heuristics and exact algorithms in combinatorial optimization. *Eur. J. Inf. Technol. Comput. Sci.* **2**, 6–12, DOI: 10.24018/ejcompute.2022.2.2.50 (2022).

24. Fakhravar, H. Combining heuristics and exact algorithms: A review (2022). ArXiv preprint arXiv:2202.02799.

25. Cattelan, M. & Yarkoni, S. Modeling routing problems in QUBO with application to ride-hailing. *Sci. Reports* **14**, DOI: 10.1038/s41598-024-70649-3 (2024).

26. Ahuja, R. K., Ergun, Ö., Orlin, J. B. & Punnen, A. P. A survey of very large-scale neighborhood search techniques. *Discret. Appl. Math.* **123**, 75–102 (2002).

27. Windras Mara, S. T., Norcahyo, R., Jodiawan, P., Lusiantoro, L. & Rifai, A. P. A survey of adaptive large neighborhood search algorithms and applications. *Comput. Oper. Res.* **146**, 105903 (2022).

28. Grabowski, J. & Wodecki, M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Comput. Oper. Res.* **31**, 1891–1909 (2004).

29. Grabowski, J. & Wodecki, M. A very fast tabu search algorithm for the job shop problem. In Rego, C. & Alidaee, B. (eds.) *Adaptive memory and evolution; tabu search and scatter search*, 117–144 (Kluwer Academic Publishers, Dordrecht, 2005).

30. Naranjo, B. J. & Ramírez, J. L. Generalized fibonacci numbers: Compositions and graphs. *Resonance* **31**, 503–519, DOI: 10.1007/s12045-026-1973-8 (2026).

31. Potts, C. N. & Van Wassenhove, L. N. A branch and bound algorithm for the total weighted tardiness problem. *Oper. Res.* **33**, 363–377, DOI: 10.1287/opre.33.2.363 (1985).

32. Boothby, K., Bunyk, P., Raymond, J. & Roy, A. Next-generation topology of d-wave quantum processors. *arXiv preprint arXiv:2003.00133* (2020).