Baltic J. Modern Computing, Vol. 12 (2024), No. 3, pp. 259–269
https://doi.org/10.22364/bjmc.2024.12.3.03

Quantum Algorithm for the Domatic Number Problem

Andris AMBAINIS, Iļja REPKO

Centre for Quantum Computer Science, Faculty of Sciences and Technology, University of Latvia
Raiņa bulvāris 19, Riga, LV–1586, Latvia

andris.ambainis@lu.lv, ilja.repko@gmail.com

ORCID 0000-0002-8716-001X, ORCID 0009-0003-3154-1803

Abstract. In this paper we design a quantum algorithm for the NP-complete problem of finding the domatic number. The DOMATIC NUMBER problem asks to determine the largest integer k, such that a given undirected graph with n vertices can be partitioned into k pairwise disjoint dominating sets. This problem finds significant applications, such as in wireless sensor networks, where the selection of multiple dominating sets balances energy consumption and extends network lifetime. Each node communicates exclusively with designated nodes, and dominant sets ensure network resilience by enabling seamless replacement in case of node cluster failure. The importance of this problem lies in its implications for network optimization, highlighting the advantages of quantum computing in addressing complex combinatorial challenges. We present a quantum algorithm that solves this problem in time $\mathcal{O}(2.4143^n)$, which we further improve to $\mathcal{O}(2.3845^n)$.

Keywords: domatic number, quantum algorithm, dominating set

1 Introduction

In this paper we have discovered two quantum algorithms to find the domatic number $d(G)$ for the graph G. The problem was discovered in (Chang, 1994).

These algorithms solve a critical problem in graph theory that affects diverse fields, including wireless sensor networks (Jiguo, Qingbo, Dongxiao, Congcong and Guanghui, 2014). Efficiently selecting and managing multiple dominating sets in these networks is crucial for saving energy and extending network lifespan. Quantum computing offers a promising approach to enhance network optimization and analyze data structures in new ways. This shows how quantum algorithms can tackle complex problems that traditional computers struggle with.

Previous results. Classically the problem can be solved in time $3^n \cdot n^{\mathcal{O}(1)}$ (Fomin, Kratsch, 2010) using an algorithm similar to Lawler's dynamic programming algorithm (Lawler, 1976). In the first half of 2005, Riege T. and Rothe J. "broke through" the 3^n barrier, solving the DOMATIC NUMBER problem for 3 dominating sets in time $\tilde{\mathcal{O}}(2.9416^n)$ (Riege, Rothe 2005). Later, authors combined the inclusion-exclusion approach with the Fomin algorithm (Fomin, Grandoni, Pyatkin, Stepanov, 2005) to solve the problem in time $\tilde{\mathcal{O}}(2.695^n)$ for 3 dominating sets (Riege, Rothe, Spakowski, Yamamoto, 2007).

Nevertheless the best known classical algorithm was presented by Johan M.M. van Rooij which solved the problem in time $\mathcal{O}(2.7139^n)$ (van Rooij, 2010).

Until recently, no quantum algorithm was known for this problem. Then, two algorithms were developed independently: the algorithm in this paper (which was developed and presented as a bachelor's thesis in June 2023 in University of Latvia (Repko, 2023)) and an algorithm (Gaspers and Li, 2023) which appeared in November 2023. The quantum algorithm of Gaspers and Li is faster, with the complexity of $\mathcal{O}((2-\epsilon)^n)$ but our algorithm is simpler.

Main contributions:

- Firstly, we designed an algorithm that solves the DOMATIC NUMBER problem in time $\mathcal{O}(2.4143^n)$. This technique combines Lawler's algorithm for finding the chromatic number (Lawler, 1976) with Grover's search (Durr and Høyer, 1996) and the naive computation of minimal dominating sets using dynamic programming in time $2^n \cdot n^{\mathcal{O}(1)}$.
- Secondly, we developed an algorithm that utilizes the precomputation of improved minimal dominating sets (Fomin, Grandoni, Pyatkin, and Stepanov, 2008), solving the DOMATIC NUMBER problem in time $\mathcal{O}(2.3845^n)$.
- Thirdly, we provided a data structure that precomputes all minimal dominating sets in time $\mathcal{O}^*(2^n)$ and allows each set to be obtained in time $\mathcal{O}(n^c)$.

The article consists of four sections. The first section is the introduction, where the main problem and contributions are presented. The second section covers the preliminaries, theorems, and techniques important for implementing the algorithm on a quantum computer. The third section is divided into two parts: the first part describes the simple algorithm that solves the problem in time $\mathcal{O}(2.4143^n)$, and the second part presents an improved version of it that solved DOMATIC NUMBER in time $\mathcal{O}(2.3845^n)$. The fourth section presents the conclusions and discusses open problems that may be addressed in future research.

2 Preliminaries

In this section, the main theorems and the model necessary for the algorithm to work will be described.

Model. Our algorithms work in the commonly used QRAM (quantum random access memory) model of computation (Giovannetti, Lloyd and Maccone, 2008), which assumes quantum memory that can be accessed in a superposition.

Tools. We will use the following results in our algorithms:

Theorem 1. *[Quantum Minimum Finding (1996)] Let $a_1,\dots,a_n$ be integers, accessed by a procedure $\mathcal{P}$. There exists a quantum algorithm that finds $\min_{i=1}^{n}\{a_i\}$ with success probability at least 2/3 using $\mathcal{O}(\sqrt{n})$ applications of $\mathcal{P}$.*

Theorem 2. *[Minimal dominating sets listing (2008)] For any graph G on n vertices, all its minimal dominating sets in X can be listed in time $\mathcal{O}(1.7159^n)$.*

Theorem 3. *[Classical domatic number finding (2008)] There is a classical algorithm that solves* DOMATIC NUMBER *for any graph $G(V,E)$ on n vertices in time $\mathcal{O}(2.8718^n)$ and lists all minimal dominating sets contained in a given $X \subseteq V$ in time $\mathcal{O}(\lambda^{n+\alpha_4|X|})$ with the following values for the weights: $\lambda = 1.148698$ and $\alpha_4 = 2.924811$.*

3 The algorithm

In this section, two quantum algorithms that solve the problem will be described. The simple algorithm solves the problem in time $\mathcal{O}(2.4143^n)$, while the improved algorithm solves it in time $\mathcal{O}(2.3845^n)$.

3.1 Simple algorithm

Our strategy consists of two parts. Firstly, recursively finding all minimal dominating sets in the graph G. The naive deterministic algorithm for listing minimal dominating sets runs in time 2^n, up to polynomial factors. Secondly, we use *Quantum Maximum Finding* to find the largest partition into sets (Ahuja, Kapoor, 1999). This algorithm is based on quantum Grover's search algorithm (Grover, 1996). The analysis of the domatic number finding algorithm is based on (Ambainis, Balodis, Iraids, Kokainis, Prūsis and Vihrovs, 2018), Theorem 1 and is similar to Lawler's classical algorithm for computing the chromatic number (Lawler, 1976) (Fomin, Grandoni, 2005).

Theorem 4. *There is a bounded-error quantum algorithm that solves* DOMATIC NUMBER *in time $\mathcal{O}(2.4143^n)$.*

Proof. The algorithm calculates $d(G)$ for the graph $G(V,E)$ by iterating through all possible subsets $X \subseteq V$. The algorithm seeks to find the largest partition into disjoint dominating sets. The corresponding recursive formula is:

$$d(X) = \max\{d(X \setminus D) + 1 \mid D \subseteq X, D \text{ is a minimal dominating set in } G\} \quad (1)$$

Note that for $X \in \emptyset$, $d(X) = 0$.

By replacing classical maximal value search with the quantum maximum finding algorithm in $d(G)$, we obtain a quantum speedup for this problem. Quantum maximum search achieves a quadratic speedup over classical exhaustive search. For each $X \subseteq V$, the quantum algorithm will find $\max_{i=1}^{|X|}\{D_i\}$ in time $\mathcal{O}^*(\sqrt{|X|})$ [1] (Durr and Høyer,

[1] The $\mathcal{O}^*(f(n))$ notation hides a polynomial factor in n

1996). Until we have to iterate through all subsets in G, the running time of the algorithm is bounded by:

$$\sum_{i=0}^{n} \binom{n}{i} i^{\mathcal{O}(1)} \sqrt{2^i} \le n^{\mathcal{O}(1)} \sum_{i=0}^{n} \binom{n}{i} \sqrt{2^i} = n^{\mathcal{O}(1)} (1+\sqrt{2})^n \in \mathcal{O}(2.4143^n) \square$$

3.2 Improved algorithm

The main idea of the improved algorithm is to precompute all minimal dominating sets (MDS) in $G(V, E)$, where $|V| = n$. The analysis of this algorithm is based on (Ambainis, Balodis, Iraids, Kokainis, Prūsis and Vihrovs, 2018), Theorems 1, 3, and has similarities with Lawler's classical algorithm for computing the chromatic number (Lawler, 1976).

Theorem 5. *There is a bounded-error quantum algorithm that solves* DOMATIC NUMBER *in time* $\mathcal{O}(2.3845^n)$.

Proof. We use the same recurrence as in the proof for Theorem 4 to find the domatic number $d(G)$ for graph $G(V, E)$:

$$d(X) = \max\{d(X \setminus D) + 1 \mid D \subseteq X, D \text{ is a minimal dominating set in } G\}$$

Note that for $X \in \emptyset$, $d(X) = 0$. We preprocess the data to enable quick access to all minimal dominating sets. Namely, we create a data structure that can answer two types of queries:

- Given a subset of vertices $X \subseteq V$, what is the number of minimal dominating sets contained in X?
- Given X and i, what is the i^{th} minimal dominating set contained in X (in some fixed ordering)?

Lemma 1. *There is a data structure that can be created in time* $\mathcal{O}^*(2^n)$ *and, given this data structure, the queries of the two types can be answered in time* $\mathcal{O}(n^c)$.

If this data structure has been created, we can find the value of $d(X)$ from equation (1) in time $\mathcal{O}^*(\sqrt{D(X)})$ (where $D(X)$ denotes the number of minimal dominating sets contained in X) using quantum maximum finding from Theorem 1.

We then use this to compute $d(X)$ for all X, in the order of increasing $|X|$. Since $D(X) = \mathcal{O}(\lambda^{n+\alpha_4 i})$, the running time for performing this is of the order at most

$$n^{\mathcal{O}(1)} \sum_{i=0}^{n} \binom{n}{i} \sqrt{\lambda^{n+\alpha_4 i}} = n^{\mathcal{O}(1)} \sum_{i=0}^{n} \binom{n}{i} \lambda^{\frac{n}{2}+\frac{\alpha_4 i}{2}} = \lambda^{\frac{n}{2}} (1+\lambda^{\frac{\alpha_4}{2}})^n n^{\mathcal{O}(1)}$$

$$1.148698^{\frac{n}{2}} (1 + 1.148698^{\frac{2.924811}{2}})^n n^{\mathcal{O}(1)} \in \mathcal{O}(2.3845^n)$$

It remains to prove Lemma 1. To store all the sets in memory, we will utilise two Hasse diagrams denoted as h_1 and h_2. (A Hasse diagram is a structure with entries for subsets $X \in V$ in which the entry for X has pointers to the entries for $X - \{u\}, u \in X$.) Each element in h_1 will have the data type 'Node', while each element in h_2 will have the data type 'DNode'.

```
struct Node
 Set: Set of integer
 Subsets: Set of Node
 Mds: bool
 MinDomSets: DNode
 InDNode: List of ($A$: set, $B$: set, $Ref$: DNode)
```

For an element v, v.*Set* contains the set X. v.*Subsets* contains links to 'Node' structures for all subsets of X whose cardinality is one less than the one of X. v.*Mds* is **true** if X is a minimal dominating set and **false** otherwise. v.*MinDomSets* and v.*InDNode* provide links to the entries of Hasse diagram h_2 and are described later, after we describe h_2.

The Hasse diagram h_2 has elements corresponding to pairs of sets A, B with $A, B \subseteq V$ and $\max(i \in A) < \min(j \in B)$. (That is, every element of A must be smaller than every element of B. The corresponding entry describes the number of minimal dominating sets X with $A \subseteq X \subseteq A \cup B$ and provides a way to index them. The elements of h_2 will have the data type 'DNode'.

```
struct DNode
 A: Set of integer
 B: Set of integer
 Count: integer
 Mds: bool
 Subsets: Set of DNode
```

We say that for each element v_1, v_2 in h_2: $v_2 \prec v_1$ iff

$$\exists b \in v_1.B : (v_2.A = v_1.A \cup b) \wedge (v_2.B = v_1.B \setminus \{x \in v_1.B | x \leq b\})$$

In h_2, the fields have the following content. A, B represent vertex sets in v_1, while *Subsets* contains links to all v_2 with $v_2 \prec v_1$. We mark *Mds* as True iff A is minimal dominating set. *Count* contains the number of X with $v_1.A \subseteq X \subseteq v1.A \cup v1.B$.

We note that if X is such that $v_1.A \subset X \subseteq v1.A \cup v1.B$, then X satisfies $v_2.A \subseteq X \subseteq v2.A \cup v2.B$ for exactly one v_2 with $v_2 \prec v_1$. Namely, this will be the node v_2 with $v_2.A = v_1.A \cup \{c\}$ where c is the smallest element of $v_1.B$ and $v_2.B = \{x | x \in v_1.B \wedge x > c\}$. Thus, the set of X with $v_1.A \subset X \subseteq v1.A \cup v1.B$ is a disjoint union of the sets of X with $v_2.A \subseteq X \subseteq v2.A \cup v2.B$ for all $v_2 \prec v_1$. All those v_2 are enumerated by the *Subsets* field.

Lastly, we describe the references from h_1 to h_2. There are two types of them.

The first type, denoted as *MinDomSets*, finds the element of type **DNode** that refers to i-th MDS within h_2. **Node**.*MinDomSet* = **DNode** iff **Node**.*Set* = **DNode**.*B* and **DNode**.$A = \emptyset$.

The second type of reference, called *Ref*, is stored in the *IsDNode* list. For every N of **Node** objects, the *IsDNode* list maintains pairs of sets denoted as $(\mathcal{A}, \mathcal{B})$. These pairs satisfy the condition that $\mathcal{A} \cup \mathcal{B} = N.Set$. Furthermore, it is required that an object of type **DNode** has been previously instantiated within h_2. This prevents the occurrence of duplicates in h_2.

Next we describe Algorithm 1 that find i-th minimal dominating set in polynomial time. The algorithm 1 consists of two parts.

In the first part, we identify the subset S in the Hasse diagram h_1 in time $\mathcal{O}(n^2)$ using function FINDSUBSET.

In the second part, the algorithm refers to **DNode** object in h_2 using **Node** *MinDomSet*. After this, the search for the i-th subset in h_2 starts. We check all *Subsets* of the current **DNode** element until the sum of all previous *Count* values is less than i. When this sum equals or exceeds i, the algorithm descends one level lower. This process continues until **DNode** *Mds* is not *True*. When navigating h_2, this search is confined to a maximum of n *Subsets*, at most n times, for each element in B. Therefore the second part operates in time $\mathcal{O}(n^2)$.

Algorithm 1 Finding the i-th minimal dominating set among subsets of S

Input: S - subset of V, i - index for the minimal dominating set

```
 1: function FINDSUBSETMINDOMSETBYINDEX($S$: set, $i$: integer)
 2:     $\mathcal{S} \leftarrow$ FINDSUBSET($S$).MinDomSets                ▷ $\mathcal{S}$: DNode
 3:     while $\mathcal{S}$.Mds = False do                              ▷ Search in $h_2$
 4:         $c \leftarrow 0$
 5:         for $subset$ in $\mathcal{S}$.Subsets do                     ▷ $subset$: DNode
 6:             if $c$+subset.Count $\geq i$ then
 7:                 $\mathcal{S} \leftarrow subset$
 8:                 $i$ -= $c$
 9:                 $c \leftarrow 0$
10:                 break
11:             else
12:                 $c \leftarrow c +$ Count
13:     return $\mathcal{S}$.A
```

For the purpose of precomputation all minimal dominating sets for G, we execute Algorithm 2. This precomputation algorithm is designed to store all subsets of V within the h_1. The naive algorithm for finding the i-th minimal dominating set of S in h_1 requires exponential time to visit all subsets of S. Consequently, we introduce h_2 and the procedure D to calculate the count of minimal dominating sets that include the elements of set A and potentially some elements from set B:

$$\mathrm{D}(A,B) = \begin{cases} \sum_{i=0}^{k<n} \mathrm{D}(A \cup b_i, \{b_{i+1}, \dots, b_k\}) & \text{, if } A \text{ is not MDS } \textbf{and } B \neq \emptyset \\ 1 & \text{, if } A \text{ is MDS} \\ 0 & \text{, otherwise.} \end{cases}$$

$$B = \{b_0, \dots, b_k\} = \{b \mid \forall i, j \geq 0 (i < j \to b_i < b_j)\}$$

To obtain all MDS for subset S we run D($\emptyset, S$).

Algorithm 2 Precomputation of minimal dominating sets

Input: $G(V, E)$ - input graph.
Output: h_1, h_2 with precomputed MDS.

```
1. Generate all subsets of $V$ and place them within the $h_1$.
2. Execute the algorithm for generating minimal dominating sets from Theorem 2. When a
   certain set $D$ is returned:
   2.1. Find $D$ in $h_1$, using Function FINDSUBSET.
   2.2. Mark Mds as True in found $D$ node in $h_1$.
3. Execute Procedure PRECOMPUTEMINDOMSETS.
```

The procedure PRECOMPUTEMINDOMSETS is implemented in Algorithm 3, which creates h_2 using h_1. For each subset $S \subseteq V$, the algorithm executes D($\emptyset, S$) and stores each term of D($\emptyset, S$) in h_2. To find all subsets of V we run Function GETSUBSETS. While $\forall a(a \in A \to \forall b(b \in B \to a < b))$ holds true, we state that $A \subseteq \{0, \dots, m-1\}$ and $B \subseteq \{m, \dots, n-1\}$. Consequently, there are 2^m ways to choose A and 2^{n-m+1} ways to choose B, with n ways to choose the value of m. Therefore, Algorithm 3 runs in time $\mathcal{O}^*(n \cdot 2^m 2^{n-m+1}) = \mathcal{O}^*(2^n)$. Now, we just need to demonstrate that the auxiliary functions for the Algorithm 3 will operate in polynomial time.

Note that the Algorithm 3 requires exponential space $\mathcal{O}^*(2^n)$.

Algorithm 3 Implementation of the PRECOMPUTEMINDOMSETS Procedure

Input: P: Node - pointer on h_1 head element

```
1: procedure PRECOMPUTEMINDOMSETS                                ▷ Creates $h_2$
2:     for $subset$ in GETSUBSETS($P.Set$) do                     ▷ Subsets in $h_1$
3:         $A \leftarrow \emptyset$
4:         $B \leftarrow subset.Set = \{x \mid \forall i, j \geq 0(i < j \to x_i < x_j)\}$
5:         $N \leftarrow$ new DNode(
6:                     $A, B, Count : 0, Mds :$ ISMDS($subset$), $Subsets : \emptyset$)
7:         $subset$.MinDomSets $\leftarrow N$
8:         INSERTDNODEREF($A, B$)
9:         D($A, B$)
```

Prior to adding an element to the h_2, Algorithm 3 performs a verification step by invoking the FINDDNODE function. This function checks for the presence of the current element within h_2. Unless each $a \in \mathcal{A}$ and $b \in \mathcal{B}$, $a < b$, the size of the *IsDNode* list will be at most $\mathcal{O}(n)$. Therefore FINDDNODE operates in polynomial time. If the element is found, the *Count* attribute of the corresponding **DNode** instance is incremented by the value from the current subset. In cases where the element is not found, the algorithm adds reference *Ref* from h_1 to h_2 using INSERTDNODEREF function. After that it computes the *Count* value via a recursive formula for the D function.

```
procedure D($A, B = \{x \mid \forall i, j \geq 0(i < j \to x_i < x_j)\}$)
    if IsMds($A$) then
        return 1

    for $k = 0; k < |B|; k{+}{+}$ do
        $\mathcal{A} \leftarrow A \cup \{x_k\}$
        $\mathcal{B} \leftarrow \{x_{k+1}, x_{k+2}, \ldots, x_n\}$
        $R \leftarrow$ FindDNode($\mathcal{A}, \mathcal{B}$)                              ▷ Type: DNode

        if $R = \emptyset$ then
            $N.Subsets$ add new DNode(
                    $\mathcal{A}, \mathcal{B}, Count : 0, Mds : False, Subsets : \emptyset$)

            InsertDNodeRef($\mathcal{A}, \mathcal{B}$)
            $N.Count$ += D($\mathcal{A}, \mathcal{B}$)
        else                                                         ▷ Computed early
            $N.Subsets$ add $R$
            $N.Count$ += $R.Count$
    return 0

function GetSubsets($node$: Node)
    if $node = \emptyset$ then
        return
    yield $node$                                                     ▷ Type: Node
    for $subset$ in $node.Subsets$ do
        GetSubsets($subset$)

function InsertDNodeRef($A$: Set, $B$: Set)
    $\mathcal{S} \leftarrow$ FindSubset($A \cup B$)                                 ▷ Type: Node
    $\mathcal{S}$.InDNode add ($A, B$)

function IsMds($S$: Set)
    $\mathcal{S} \leftarrow$ FindSubset($S$)                                      ▷ Type: Node
    if $\mathcal{S} = \emptyset$ then                                   ▷ No such set in diagram
        return False
    return $\mathcal{S}$.Mds

function FindDNode($A$: set, $B$: set)
    $\mathcal{S} \leftarrow$ FindSubset($A \cup B$)                                 ▷ Type: Node
    for ($\mathcal{A}, \mathcal{B}, Ref$) in $\mathcal{S}$.InDNode do
        if $A = \mathcal{A}$ and $B = \mathcal{B}$ then
            return $Ref$                                             ▷ Type: DNode
    return $\emptyset$
```

Before calculating each term of the recursive function D for $(A, B) = (\emptyset, S)$, the Algorithm 3 checks whether the set A is MDS using the IsMDS function. If the current set A is MDS, then the algorithm will return 1. Otherwise, it will call the function D until $B \neq \emptyset$.

Next we describe how to implement FINDSUBSET, using Algorithm 4. Note that each set contains no more than n edges from current node to it subsets. When navigating the Hasse diagram h_1, our search is confined to a maximum of n sets. Therefore Algorithm 4 runs in time $\mathcal{O}(n^2)$.

Algorithm 4 Finding subsets in Hasse diagram h_1

```
Input: $D$ - subset, $G$ - pointer on $h_1$ head element
1: function FINDSUBSET($D$: set)
2:     $S \leftarrow G$
3:     while $S.Set \neq D$ do
4:         if $S.Subsets = \emptyset$ then          ▷ No such set in diagram
5:             return $\emptyset$
6:         for $s \in S.Subsets$ do
7:             if $D \subseteq s$ then
8:                 $S \leftarrow s$
9:                 break
10:    return $S$
```

4 Conclusion and open problems

The DOMATIC NUMBER problem holds significant importance in graph theory and finds practical applications in network optimization. The quantum algorithm for this problem recursively searches for minimal dominating sets within a graph. At each recursion level, sets are excluded from the current vertex set until all vertices are covered. The algorithm determines the maximum recursion level where each excluded set remains dominating.

Our findings revel that the one of the best-known classical algorithm solves the problem with a time complexity of $\mathcal{O}(2.7139^n)$ (van Rooij, 2010) using a dynamic programming approach. Quantum maximum search provides a quadratic speedup for enumerating all minimal dominating sets, thereby achieving a faster solution to the DOMATIC NUMBER problem.

Our newly presented quantum algorithms demonstrate the power of quantum computing to accelerate classical algorithms and achieve improved time complexity with relatively straightforward implementation. Combining dynamic programming with quantum algorithms improves the evaluation of the DOMATIC NUMBER algorithm.

A key achievement in this work includes Theorem 4 and 5. Employing the Quantum Maximum Finding algorithm, we have demonstrated algorithms that solve the DOMATIC NUMBER problem in time $\mathcal{O}(2.4143^n)$, which we have further improved to

$\mathcal{O}(2.3845^n)$. Gaspers and Li (2023) have independently developed a quantum algorithm with the complexity of $\mathcal{O}((2-\epsilon)^n)$.

It would be interesting to explore whether there exists an algorithm that can further improve this complexity. However, limitations include the requirement for exponential space, which may not be practical for large-scale graphs. This presents a crucial area for further research.

Acknowledgment. AA was supported by QuantERA ERANET Cofund project QOPT (Quantum algorithms for optimization).

References

Ahuja, A., Kapoor, S. (1999) A Quantum Algorithm for finding the Maximum *Quantum Physics* at `arxiv.org/abs/quant-ph/9911082`

Ambainis, A., Balodis, K., Iraids, J., Kokainis, M., Prūsis, K., Vihrovs, J. (2018). Quantum Speedups for Exponential-Time Dynamic Programming Algorithms, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1783-1793. doi:10.1137/1.9781611975482.107

Chang., G.J. (1994) The domatic number problem, *Elsevier*, 125, issues 1-3. doi: 10.1016/0012-365X(94)90151-1

Durr, C., Høyer, P. (1996). A Quantum Algorithm for Finding the Minimum, preprint, available at `arXiv:quant-ph/9607014`

Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A. (2005) Bounding the number of minimal dominating sets: A measure and conquer approach. *Algorithms and Computation* 3827. doi:10.1007/11602613_58

Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A. (2008). Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms* **5**(1), 9:1-9:17. doi:10.1145/1435375.1435384

Fomin, F.V., Kratsch, D. (2010). Exact Exponential Algorithms. *Springer* p.36. doi:10.1145/2428556.2428575

Gaspers, S., Li, J.Z. (2023). Quantum Algorithms for Graph Coloring and other Partitioning, Covering, and Packing Problems. preprint, available at `arXiv:2311.08042`

Giovannetti, V., Lloyd, S., Maccone, L. (2008). Quantum Random Access Memory. *Phys. Rev. Lett.* **100**, p. 160501. doi: 10.1103/PhysRevLett.100.160501

Grover, L. (1996). A Fast Quantum Mechanical Algorithm for Database Search. *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing.* p.2112-229. doi: 10.1145/237814.237866

Jiguo, Y., Qingbo, Z., Dongxiao, Y., Congcong. C., Guanghui, W. (2014). Domatic partition in homogeneous wireless sensor networks. *Elsevier* doi: 10.1016/j.jnca.2013.02.025

Lawler, E.L. (1976). A note on the complexity of the chromatic number problem. *Information Processing Lett.* **5**, 66–67, doi: 10.1016/0020-0190(76)90065-X

Repko, I. (2023). Quantum Algorithms for Domatic Number Finding Problem. preprint, available at `https://dspace.lu.lv/dspace/handle/7/63279`, last viewed <14.02.2024>

Riege, T., Rothe, J. (2005) An Exact 2.9416ˆn Algorithm for the Three Domatic Number Problem. *Mathematical Foundations of Computer Science* 3618. doi: 10.1007/11549345_63

Riege, T., Rothe, J., Spakowski, H., Yamamoto, M. (2007) An improved exact algorithm for the domatic number problem *Information Processing Letters* 101, p.101-106. doi: 10.1016/j.ipl.2006.08.010

Van Rooij, J.M.M. (2010). Polynomial Space Algorithms for Counting Dominating Sets and the Domatic Number. *Algorithms and Complexity, Lecture Notes in Computer Science* **6078**, pp. 73-84. doi: 10.1007/978-3-642-13073-1_8

Received February 14, 2024 , revised June 16, 2024, accepted August 16,2024