Quantum Algorithms for Backtracking Search

Sunandita Patra
patras@umd.edu

May 11, 2017

Abstract

This project studies how quantum algorithms can achieve speedup for backtracking search using the discrete quantum walk procedure. Backtracking is a technique which explores a search tree whose vertices are partial solutions in order to find a complete solution. It also shows how quantum backtracking algorithm can improve the running time of traveling salesman problem for bounded degree graphs upto degree 6. It identifies the limitations of quantum backtracking search and discusses some open problems.

1 Overview

Backtracking is a search procedure that can solve different types of problems, one of them being the class of NP-complete problems which have a certain structure, like the CSPs (Constraint Satisfaction Problems). Classical CSPs need to assign values to variables such that a given set of constraints is satisfied. One way to do this is through a backtracking search algorithm which explores a tree whose nodes represent partial solution to the CSP. Children of a node represents partial solutions build using the parent node's partial solution.

In this project, we explore how quantum walk can speed up the process of backtracking search. We discuss the classical backtracking technique in Section 2. Then we discuss quantum algorithms that use quantum walk and speed up backtracking search in Section 3. We study how quantum backtracking search can speed up the traveling salesman problem in Section 4. We identify the limitations of quantum backtracking in Section 5 and discuss some open problems in Section 6. We conclude in Section 7.

2 Classical Backtracking

Classical backtracking starts with no information about the solution and builds it step by step as it explores new nodes of the tree. Figure 2 shows how a backtracking tree might look like. Backtracking makes use of two functions, a predicate $P : D \rightarrow \{true, false, indeterminate\}$ and a heuristic $h : D \rightarrow \{1, ..., n\}$. P and h helps to check whether a particular node is a solution and enumerate the children of a node. The domain of P and h consist of all possible partial solutions where each of the n bits can have values between 0 and $d-1$, denoted by set $[d] = \{0, 1, ..., d-1\}$, or be unassigned, denoted by $*$. So, $D = ([d] \cup \{*\})^n$.

The pseudocode is given below in Algorithm 1. The initial call to the backtracking will look like Backtrack($*^n$) where each of the n bits are unassigned. It is a recursive algorithm which first checks $P(x)$. If P can say whether the partial assignment x is a solution($true$) or not($false$). If either of these is true, this branch of the backtracking algorithm ends and we return. Otherwise,

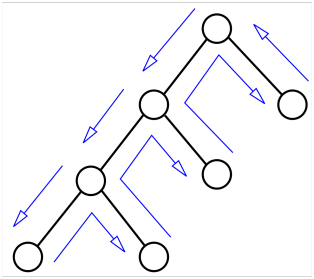

Figure 1: Classical Backtracking Search tree

if P returns *indeterminate*, we further explore this branch of the tree. We do this by generating d different children by assigning d different values (from $[d]$) to an unassigned position j as decided by the heristic h. Then, we recursively call Backtrack for each child y of x. The process continues until we have found a solution or we have looked at all possible assignments.

Backtrack(x):

1. If $P(x) = true$, output x and return
2. If $P(x) = false$ or x is a complete assignment, return
3. $j \leftarrow h(x)$
4. for each $w \in [d]$:
   (a) $y \leftarrow x$
   (b) $y[j] \leftarrow w$
   (c) Backtrack(y)

Algorithm 1: Classical backtracking algorithm

Some of the problems solved by classical backtracking technique include the travelling salesman problem, which we discuss in Section 4, the graph coloring problem as shown in Figure 2, the sliding-tile puzzle problem, the 8-queen problem and the constraint satisfaction problem.

3 Quantum Algorithms for Backtracking

When we are exploring a search tree, there are two problems which we can solve. One is checking whether there exists a solution or not, and another is finding a particular solution x_0. If we know how to detect a solution, we can use that as a subroutine to find the exact solution. Using quantum walk in the backtracking search tree ([1]) gives us the following results:

1. *Let T be an upper bound on the number of vertices in the tree explored by Alorithm 1. Then for any $0 < \delta < 1$ there is a quantum algorithm which, given T, evaluates P and h $O(\sqrt{Tn} \log 1/\delta)$ times each, outputs true if there exists x such that $P(x)$ is true, and outputs false otherwise. The algorithm uses poly(n) space, and fails with probability at most δ.*

2. *In order to find an exact solution x_0, we need $O(\sqrt{T} n^{3/2} \log n \log(1/\delta))$ evaluations of P and*

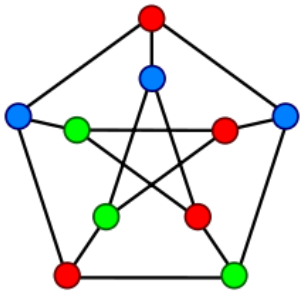

Figure 2: This figure shows an instance of the graph coloring problem where you need to color all the vertices of the graph with distinct colors such that no two adjacent vertices have the same color.

h with failure probability at most δ.

3. *If we have the promise that x_0 is unique, we need $O(\sqrt{Tn}\log^3 n \log(1/\delta))$ evaluations of P and h, with failure probability at most δ.*

We describe here how point 1. above is true using discrete quantum walk on the classical backtracking search tree having T nodes. The procedure is described in Algorithm 2 below. It is developed based on a beautiful connection between quantum walks and electric circuit due to Belovs ([5]). For backtracking, the quantum walk needs to take place on a tree, and hence the analysis is simpler and more intuitive.

DoesMarkedVertexExist(R_A, R_B, δ, n, T):
$\beta, \gamma \leftarrow$ universal constants to be determined

1. $AcceptCount \leftarrow 0; RejectCount \leftarrow 0$
2. loop $K = \lceil \gamma \log(1/\delta) \rceil$
   (a) Apply phase estimation to the operator $R_B R_A$ with precision $\beta/\sqrt{Tn}$
   (b) If the eigenvalue is 1
   - $AcceptCount \leftarrow AcceptCount + 1$

   else
   - $RejectCount \leftarrow RejectCount + 1$
3. If $AcceptCount \geq 3K/8$, return "marked vertex exists"
   else return "no marked vertex"

Algorithm 2: Detecting a marked vertex

Let's say the nodes of the backtracking tree be labeled as r, 1, 2, ..., $T-1$, with node r being the root. The maximum level of the backtracking tree, which is also the maximum distance of a node from the root is n. Let

- A be the set of nodes at even distance from the root including the root itself

- B be the set of nodes at odd distance from the root

For any node x, let

- $l(x)$ be the level of x which is also the length of the path from r to x
- $x \to y$ denote that y is a child of x in the backtracking tree
- d_x is the degree of x if we consider the backtracking tree as an undirected graph. Thus, for all $x \neq r$, $d_x = |\{y : x \to y\}| + 1$ and $d_r = |\{y : r \to y\}|$

The quantum walk operates on the Hilbert space $\mathcal{H}$ spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, 2, ..., T-1\}\}$ and starts in the state $|r\rangle$. The walk is based on a set of diffusion operators D_x. D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$. It does not use the coin operator which the other discrete quantum walk procedures usually use. The diffusion operators are defined as follows:

- If x is marked, D_x is the identity
- If x is not marked and $x \neq r$

$$D_x = I - 2|\psi_x\rangle\langle\psi_x|, \text{ where}$$

$$|\psi_x\rangle = \frac{1}{\sqrt{d_x}}(|x\rangle + \sum_{y, x \to y} |y\rangle),$$

- If $x = r$, we assume that root is never marked and

$$D_r = I - 2|\psi_r\rangle\langle\psi_r|, \text{ where}$$

$$|\psi_r\rangle = \frac{1}{\sqrt{1 + nd_r}}(|r\rangle + \sqrt{n} \sum_{y, r \to y} |y\rangle)$$

We observe that D_x only depends on the local neighborhood and can be implemented using local knowledge about a node. A step of the quantum walk consists of applying the operator $R_B R_A$, where

$$R_A = \bigoplus_{x \in A} D_x$$

$$R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$$

We will explain how to implement R_A and R_B using queries to P and h later.

Another way of viewing this quantum walk is as a quantum walk on a graph where the vertices represent edges of the backtracking search tree and each vertex in the modified graph is identified with the edge from its parent and an additional "input" edge to the root. The algorithm is proven to be correct through the following Lemma.

Lemma 1 Algorithm 2 uses the operators R_A and R_B $O(\sqrt{Tn} \log 1/\delta)$ times. There exist universal constants β and γ such that it fails with probability at most δ.

Proof of Lemma 1 is as follows.
To calculate the query complexity of the Algorithm 2, we use a phase estimation theorem([6]) which says that to estimate a phase of a vector with precision $O(\epsilon)$, we need $O(1/\epsilon)$ queries. Therefore,

line 2(a) of Algorithm 2 needs $O(\sqrt{Tn}/\beta) = O(\sqrt{Tn})$ queries, because β is a constant. We do the phase estimation $K = O(\lceil \gamma \log(1/\delta) \rceil) = O(\log(1/\delta))$ times. Therefore, the total query complexity is $O(\sqrt{Tn}) \times O(\log(1/\delta)) = O(\sqrt{Tn}\log(1/\delta))$.

We proof the correctness of the algorithm by proving that if there is a marked vertex, the accept count will be atleast $3K/8$. We start by showing that if there is a marked vertex, then $|r\rangle$ is quite close to an eigenvector $|\phi\rangle$ of $R_B R_A$ with eigenvalue 1. Let x_0 be a marked vertex and set

$$|\phi\rangle = \sqrt{n}|r\rangle + \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{l(x)}|x\rangle$$

$x \rightsquigarrow x_0$ denotes the nodes x which are on the unique path from the root to x_0, including x_0 itself. To see that $|\phi\rangle$ is invariant under $R_B R_A$, we first observe that $|\phi\rangle$ is orthogonal to all states $|\psi_x\rangle$ where $x \neq r$ and x is not marked. This is because any such $|\psi_x\rangle$ has equal support from two consequtive nodes v in the path from r to x_0 or is not supported by any node in this path. Also, $|\phi\rangle$ is orthogonal to $|\psi_r\rangle$ by the following calculation.

$$\begin{aligned}
\langle\phi|\psi_r\rangle &= \left(\sqrt{n}\langle r| + \sum_{x \neq r, x \rightsquigarrow x_0} (-1)^{l(x)}\langle x|\right)\left(\frac{1}{\sqrt{1+nd_r}}(|r\rangle + \sqrt{n}\sum_{y, r \to y}|y\rangle)\right) \\
&= \sqrt{\frac{n}{1+nd_r}}\langle r|r\rangle + 0 + 0 + (-1)^1\sqrt{\frac{n}{1+nd_r}}\langle x'|x'\rangle \\
&\quad \text{where } x' \text{ is the child of } r \text{ in the path from } r \text{ to } x_0 \\
&= 0
\end{aligned}$$

Now, we have

$$\||\phi\rangle\|^2 = n + l(x_0) \leq 2n$$

Thus,

$$\frac{\langle r|\phi\rangle}{\||\phi\rangle\|} \geq \frac{\sqrt{n}}{\sqrt{2n}} \geq \frac{1}{\sqrt{2}}$$

Therefore, phase estimation returns 1 with probability atleast 1/2. On the other hand, if there are no marked vertices, we consider the vector

$$|\eta\rangle = |r\rangle + \sqrt{n}\sum_{x \neq r}|x\rangle$$

Let Π_A and Π_B be projectors onto the invariant subspaces of R_A and R_B. Π_A and Π_B are spanned by vectors of the form $|\psi_x^\perp\rangle$ for $x \in A \cup B$, where $|\psi_x^\perp\rangle$ is orthogonal to $|\psi_x\rangle$ and has support only on $|x\rangle$ and children of $|x\rangle$; in addition to $|r\rangle$ in case of R_B. On each subspace $\mathcal{H}_x$, $x \in A$, η is proportional to $|\psi_x\rangle$, so $\Pi_A|\eta\rangle = 0$. Similarly, $\Pi_B|\eta\rangle = |r\rangle$. By the effective spectral gap lemma (from [7]),

$$\|P_\chi|r\rangle\| = \|P_\chi \Pi_B|\eta\rangle\| \leq \chi\||\eta\rangle\| \leq \chi\sqrt{Tn}$$

where P_χ be the projector onto the span of the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ such that $|\theta| \leq \chi$. For small enough $\chi = \Omega(1/\sqrt{Tn})$, this is upper bounded by 1/2. By the phase estimation theorem (from [6]), there exists β such that applying phase estimation to $R_B R_A$ with precision $\beta/\sqrt{Tn}$ returns eigenvalue 1 with probability at most 1/4.

Using a Chernoff bound, by repeating the subroutine $O(\log 1/\delta)$ times and returning "marked vertex exists" if the number of acceptances is greater than $3K/8$, and "no marked vertex" otherwise, gives a failure probability of at most δ.

$R_A(|l\rangle|(i_1, v_1)\rangle...|(i_l, v_l)\rangle, \mathcal{H}_{anc})$

1. If $P(x) = true$, return
2. If l is odd,
   - subtract $h((i_1, v_1), ..., (i_{l-1}, v_{l-1}))$ from i_l
   - swap a with v_l
3. If $a \neq *$
   - $l \leftarrow l - 1$ (Now l is even and $(i_{l+1}, v_{l+1}) = (0, *)$)
4. Add $h((i_1, v_1), ..., (i_l, v_l))$ to j.
5. For each $w \in [d]$:
   - If $P((i_1, v_1), ..., (i_l, v_l), (j, w)) \neq false$, $S \leftarrow S \cup \{w\}$
6. If $l = 0$:
   - Perform $I - 2|\phi_{n,S}\rangle\langle\phi_{n,S}|$ on H_{anc}

   else
   - Perform $I - 2|\phi_{1,S}\rangle\langle\phi_{1,S}|$ on H_{anc}
7. Uncompute S and j by reversing steps 5 and 6.
8. If $a \neq *$, $l \leftarrow l + 1$
   If l is now odd,
   - add $h((i_1, v_1), ..., (i_{l-1}, v_{l-1}))$ to i_l
   - swap v_l with a

Algorihm 3: An implementation of the operator R_A. The implementation of R_B is similar.

Now, we discuss how to implement the operators R_A and R_B. The input to R_A, $|l\rangle|(i_1, v_1)\rangle...|(i_l, v_l)\rangle$ corresponds to a partial assignment $x_{i_1} = v_1, ..., x_{i_l} = v_l$ and some ancilla registers $\mathcal{H}_{anc}$ storing a tuple (a, j, S), where $a \in \{*\} \cup [d], j \in \{0, 1, ..., n\}, S \subseteq [d]$ initialized to $a = *, j = 0, S = \emptyset$.

Let $U_{\alpha,S}$ for $S \subseteq [d]$ and $\alpha \in R$ be a unitary whose action is as follows.

$$U_{\alpha,S}|*\rangle = |\phi_{\alpha,S}\rangle$$

where

$$|\phi_{\alpha,S}\rangle = \frac{1}{\sqrt{\alpha|S| + 1}} \left(|*\rangle + \sqrt{\alpha} \sum_{i \in S} |i\rangle \right)$$

Algorithm 3 describes how we can implement R_A using $|\phi_{\alpha,S}\rangle$. The implementation of R_B is similar to Algorithm 3. The differences are that we return in line 1 if $l = 0$ because R_B acts as identity for the root. In step 2, We check that l is even, instead of checking that l is odd. Same

is true for step 8. Also, we remove the check $l = 0$ in step 6 because l will always be odd at this point.

The queries to P and h made by the algorithm described above from [1] depends on the total number of nodes in the backtracking tree, which is T. But classical backtracking might do some efficient pruning and explore a much less number of nodes than T, say T'. In problem instances where T' is much less than T, we might not achieve speed up. A new quantum algorithm for backtracking search by Ambainis([3]) solves this problem by removing the dependency on T, and having it on T' instead. The complexity of the algorithm is $O(\sqrt{T'}n^{3/2})$, T' being the actual number of nodes visited by classical backtracking search.

4 Application of quantum backtracking to the Traveling Salesman Problem (TSP)

The traveling salesman problem is one of the most well known problems in computer science. It aims at finding the most optimal path (a path with the lowest cost), which covers all locations from a set of n locations and visits each location exactly once. If we represent the n locations by a graph G, where the vertices represent the locations and an edge (v_1, v_2) represents that we can go directly from one location to another. Each edge is associated a cost, $c(v_1, v_2)$. A feasible solution of the traveling salesman problem will correspond to a Hamiltonian cycle in G. Figure 3 shows a solution for the travelling salesman covering 49 states of USA. Figure 4 shows a Hamiltonian cycle in a graph.

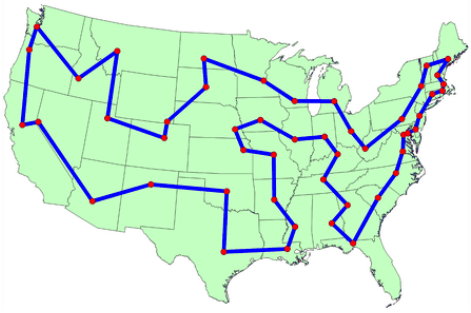

Figure 3: A solution for the traveling salesman problem visiting 49 states of USA

There are several ways we can solve the traveling salesman problem, like dynamic programming, heurtistic search algorithms, backtracking algorithms for bounded degree graphs, each having their own advantages and disadvantages. We study how quantum algorithm of [1] can used to solve the travelling salesman problem for graphs of maximum degree 6. The algorithm we discuss is from [2]. Xiao and Nagamochi ([4]) is a classical algorithm for degree 3 TSPs which finds the Hamiltonian graph with the lowest cost. It is a backtracking algorithm where the backtracking step is forcing an edge to be on the Hamiltonian cycle, or discarding it from the cycle. When all the edges have been forced or discarded, we reach the last level of the backtracking tree. It proves that for degree 3 TSPs, this procedure correctly find a Hamiltonian cycle by a series of reduction of graphs which guarantees an one-to-one correspondence between the Hamiltonian cycles of the original graph and the Hamiltonian cycle of the reduced graphs.

We now discuss how the quantum backtracking algorithm discussed in Section 3, achieves a quadratic speedup for the algorithm of Xiao and Nagamochi and also can be used for speedup

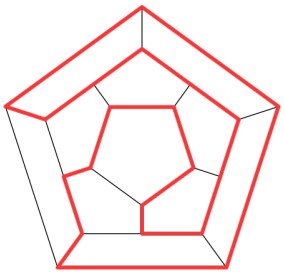

Figure 4: Hamiltonian cycle in a graph. The Hamiltonian cycle is shown in red visit each vertex exactly once and covering all the vertices of the graph. The final solution in a TSP is a Hamiltonian cycle.

for graphs with maximum degree ≤ 6. The quantum algorithm will return a feasible Hamiltonian cycle, not necessarily the shortest. Running the quantum algorithm multiple times and pruning the solutions with higher cost at every iteration until we don't find any more solutions will allow us to find the lowest cost solution for the TSP.

The first step towards applying the quantum algorithm is to define the set of partial assignments. A partial assignment will say which edges we have forced, which ones we have discarded and which ones we haven't decided about yet. Let the number of edges in the graph be m. Since the maximum degree of a vertex is 3, therefore, $m \leq 3n/2$. An assignment is denoted as $A \in (\{1, ..., m\}, \{force, remove\})^j$ with $j \leq n$. The predicate P and heuristic h makes use of a reduction function described in [4]. P takes as input a partial assignment $A = ((e_1, A_1), ..., (e_j, A_j))$ and returns *true*, $false$ or *indeterminate*. It applies the reduction function on the partial assignment A and get a modified graph G', and a new set of forced edges F'. If G' can never have a Hamiltonian cycle, it returns $false$. If G' is a graph, where finding the Hamiltonian cycle is trivial, it returns *true*. In all other cases, it returns *indeterminate*.

h also takes as input an partial assignment A and applies a reduction function to get a modified graph G'. It looks for candidate edges in G' which can be forced or removed and randomly returns one of them in case there are multiple candidates.

The complexity of the classical backtracking algorithm is $O^*(2^{3n/10})$, where the notation O^* means that we ignore the polynomial terms from the complexity. The complexity of the quantum algorithm reduced to $O^*(2^{3n/20} \log(1/\delta))$, where δ is the maximum failure probability.

5 Limitations of quantum backtracking

The quantum algorithms achieve quadratic speed up in the running time of the backtracking search algorithm. But they have some limitations which we should discuss. The quantum backtracking algorithm assumes that the nodes of the backtracking search tree are independent of each other. In other words, the decisions made in one part of the tree does not depend on any other part. This is not true for some classical backtracking algorithms, like the constraints recording algorithms ([8]).

The quantum backtracking algorithm is not very fit for finding the minimum or maximal solution. It just finds a feasible solution and needs to be called multiple times to find the best possible solution. This makes some of the work redundant.

Another limitation is that quantum backtracking does not give any guarantees to speed up

classical backtracking algorithms that can do backjumping. Backjumping allows a node to skip a number of nodes and go an ancestor directly. This can sometimes prune quite a number of nodes which correspond to the unexplored children of the nodes in the path to the ancestor. An instance of backjumping is shown in Figure 5.

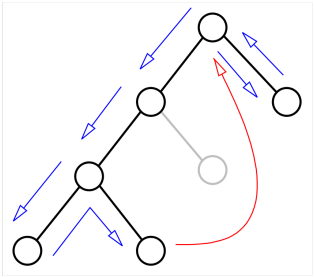

Figure 5: Backjumping in classical backtracking

In Section 4, we saw how quantum backtracking can speed up the traveling salesman problem for degree 3 graphs. Graphs upto degree 6 can be reduced into degree 3 instances and solved using this technique. The limitation is for upto degree 6, because for higher degrees, we need to transform the graph into equivalent graphs of lower degrees. This transformation needs to choose between a lot of possible choices. The number of possible choices increase with increase in the maximum degree of the TSP and this leads to an increased running time. For maximum degree ≥ 7, the number of choices dominate over the quantum speedup and thus the quantum algorithm does not achieve any speedup.

6 Open problems

In this section, we discuss how the quantum backtracking search can be improved and be made more flexible so that it can be applied to solve a larger class of problems and achieve a higher speed up.

Classical backtracking has several variants, one of them being backjumping, as shown in Figure 5. It will be interesting to see if the quantum algorithm can adapt to make use of these techniques and achieve a higher speed up. It might not improve the worst case complexity but can help compete with classical backtracking where these techniques perform well.

Montanaro ([1]) talks about how the quantum speedup might depend on the particular structure of the backtracking search tree. Quantum backtracking achieves highest speed up for long thin trees. A complete theoretical analysis can be done to develop more relations betweeen the structure and the speed ups achieved.

Dynamic programming is one way to solve any general n degree TSP. A quantum algorithm for dynamic programming ([9]), which is yet to be published, achieves a speed up for n degree TSP. There is no proof that quantum backtracking cannot speed up any general n-degree TSP. So, coming up with an algorithm or a proof is an open problem.

Also, quantum backtracking can be applied to solve problems where classical algorithm performs well, like puzzles, combinatorial optimization, logic programming, dynamic or weighted CSPs.

7 Conclusion

In this project, we have seen how discrete quantum walk procedure can be used to achieve quadratic speed-up for backtracking search. This is done by performing a quantum walk using the diffusion operators on the classical backtracking search tree. We saw how the existence of a solution is linked to an eigenvector with eigenvalue 1 of the operator $(R_B R_A)$ that represents one step of the quantum walk. We then see how the quantum backtracking algorithm can speed up the traveling salesman problem upto degree 6 by modifying the classical backtracking algorithm of Xiao and Nagamochi ([4]) for degree 3 TSPs. We also discussed some of the limitations of quantum backtracking search and identified some open problems.

References

[1] Montanaro, Ashley. *Quantum walk speedup of backtracking algorithms.* arXiv preprint arXiv:1509.02374 (2015).

[2] Moylett, Dominic J., Noah Linden, and Ashley Montanaro. *Quantum speedup of the Travelling Salesman Problem for bounded-degree graphs.* arXiv preprint arXiv:1612.06203 (2016).

[3] Ambainis, Andris, and Martins Kokainis. *Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games.* arXiv preprint arXiv:1704.06774 (2017).

[4] Xiao, Mingyu, and Hiroshi Nagamochi. *An exact algorithm for TSP in degree-3 graphs via circuit procedure and amortization on connectivity structure.* Algorithmica 74.2 (2016): 713-741.

[5] Belovs, Aleksandrs. *Quantum walks and electric networks.* arXiv preprint arXiv:1302.3143 (2013).

[6] Cleve, Richard, et al. *Quantum algorithms revisited.* Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences. Vol. 454. No. 1969. The Royal Society, 1998.

[7] Lee, Troy, et al. *Quantum query complexity of state conversion.* Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on. IEEE, 2011.

[8] Dechter, Rina. *Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition.* Artificial Intelligence 41.3 (1990): 273-312.

[9] Ambainis, Andris, and Martins Kokainis. (unpublished)

[10] Web for images used throughout the project