An Analysis of Branch-and-Bound

Rick Ramirez

Ruben Bramasco

California State Polytechnic University, Pomona

Abstract: The Branch-and-Bound paradigm is a widely used framework for solving combinatorial optimization problems. Here we describe the general algorithm and show how it can be broken into various sub-routines. For each sub-routine, we describe when it is appropriate to use it and analyze it's effect on performance. We follow this with two applications involving an integer constraint problem and the traveling salesman problem. Finally, we close with a discussion of recent improvements of the branch-and-bound framework in the wake of quantum computation and describe how a quantum algorithm could produce a quadratic acceleration of the classical time complexity.

Key Words: : Branch-and-Bound, Pruning rules, Branching strategy, Ising model, Quantum walk

1. Introduction

The branch-and-bound design paradigm has proven to be exceedingly useful in solving combinatorial optimization problems with applications in artificial intelligence, applied mathematics and theoretical computer science. The design can be applied to any problem where the goal is to find a minimum-cost solution in a setting where one has access to a bounding function that returns a lower bound on a given subset of solutions and a branching rule that can be applied to partition a subset if possible solutions cannot be ruled out. The technique has been used in solving a number of NP-hard problems and may also serve as the base of various heuristics.

The success of branch-and-bound has also sparked interest in applications involving quantum computation by encoding optimization problems to the ground state of classical spin systems. This has led to the development of quantum branch-and-bound algorithms which have been shown to increase performance significantly.

2. Basic Idea Behind BnB

The branch-and-bound framework is a set of techniques that encapsulates a family of algorithms that share a common solution procedure. The pseudo-code for a general branch-and-bound paradigm can be seen in Algorithm 1 where, for an optimization problem $\mathscr{P} = (X, f)$, X is the search space of valid solutions and $f : X \to \mathbb{R}$ is the objective function. The goal is to find an optimal solution $x^* \in argmin_{x \in X} f(x)$ by allowing the framework to incrementally move through a decision tree T. A incumbent solution $\hat{x} \in X$ is stored globally and at each iteration, the algorithm selects a new branch $S \subseteq X$ to explore from a list L of unexplored sub-problems. If a candidate incumbent $\hat{x}' \in S$ is found to have a better objective value than $\hat{x}$, i.e., $f(\hat{x}') < f(\hat{x})$, then the incumbent solution is updated and that solution branch is retained for further investigation. However, if it is found that $\forall x \in X, f(x) \geq f(\hat{x})$, then the sub-problem is pruned. Otherwise, child sub-problems are generated by partitioning S into a set of sub-problems $S_1, S_2 \ldots, S_r$ which are inserted into the tree. When the set of unexplored sub-problems is exhausted ($L = \emptyset$), the best incumbent is returned. Since sub-problems are discarded if they contain no solution better than $\hat{x}$, it

must be the case that $\hat{x} = x^*$ [1], [2], [3].

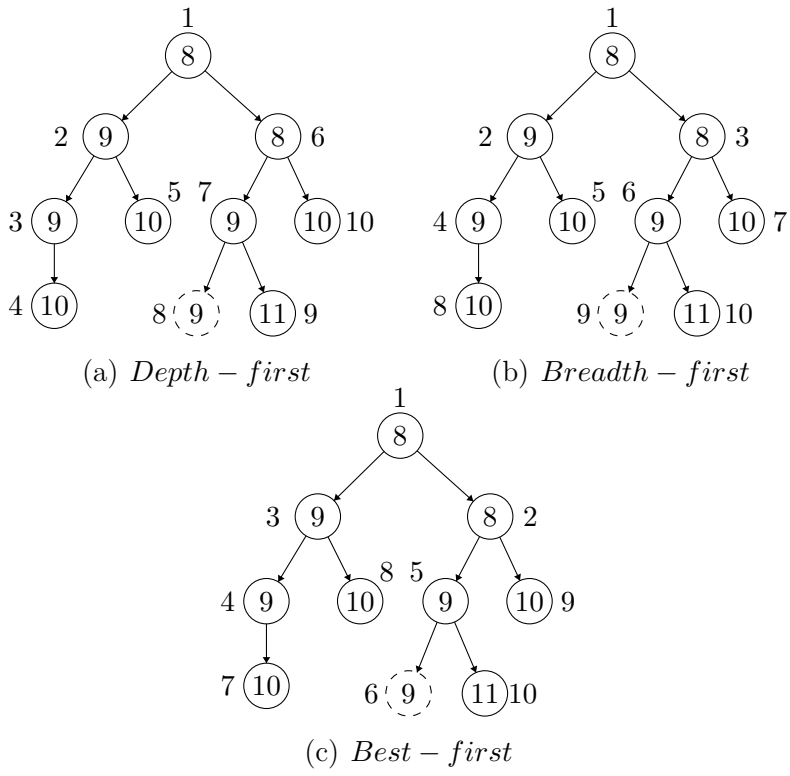
Algorithm 1 Branch-and-Bound(X,f)

```
Set L = {X} and initialize x̂
while L != ∅ do
    Select a sub-problem S from L to explore
    if a solution x̂' ∈ {x ∈ S | f(x) < f(x̂)} can be found then
        Set x̂ = x̂'
    end if
    if S cannot be pruned then
        Partition S into S_1, S_2 ..., S_r
        Insert S_1, S_2 ..., S_r into L
    end if
    Remove S from L
end while
return x̂
```

The user is free to choose which searching strategy, branching strategy, and pruning rules to utilize when employing branch-and-bound, but the nature of the problem at hand must be considered in order to obtain optimal performance. Here we explore the three sub-routines individually and examine cases where different variations are appropriate.

2.1. Search Strategies

Searching strategies used in branch-and-bound algorithms determine the order in which sub-problems are explored. The three most common techniques are, depth-first search, breadth-first search, and best-first search. Each technique is well suited for a particular scenario and the choice of search strategy for a particular problem could have a significant impact on performance and/or memory requirements.



(a) $Depth - first$ (b) $Breadth - first$

(c) $Best - first$

Fig. 1: Sub-tree traversals for different searching techniques. The numbers on the side of each node correspond to the lower-bound at that step and the dotted node represents the optimal solution [1].

2.1.1. Depth-First

The depth-first search strategy follows a last-in, first-out order and is implemented by maintaining a list of unexplored nodes in a stack as shown in Figure 1 (a). The routine removes the top element from the stack to explore for the next sub-problem and the children generated are inserted on the top of the stack. The potential memory requirements needed to maintain a list of children nodes can become a problem, however we remedy this by maintaining a path of indices from the root to the current node instead of the entire list of unexplored sub-problems. When the current sub-problem is complete, the algorithm selects the next unexplored node for exploration and if one does not exist, the algorithm backtracks to the nearest ancestor node. The biggest issue with this technique is that it is blind to the structure of the problem and makes no use of the lower bounds encountered during a

traversal. This can cause the algorithm to spend large amounts of time exploring unpromising regions, especially when the search space is unbalanced. This results in a running time of $\Theta(V + E)$ [4], however DFS yields valuable information about the structure of a graph.

2.1.2. Breadth-First

The breadth-first search strategy, shown in Figure 1 (b), explores nodes based on a first-in, first-out order and is implemented by maintaining a queue data structure. This results in the algorithm exploring all nodes at the same level before moving on to deeper portions of the tree. The advantage of this technique is that if the optimal solution is close to the root, or the tree is unbalanced, breadth-first search will find the solution faster than depth-first. However, if the solution occurs at lower depths, breadth-first can require large amounts of memory because it does not take advantage of any pruning rules. The BFS technique runs on a running time of $O(V+E)$ [4] and is linear to the size of an adjacency list representation of the graph.

2.1.3. Best-First

The best-first search strategy is often used when memory considerations are of no concern. This technique stores the entire unexplored search space and applies a heuristic function to every unexplored node and selects the node that returns the minimum value. We say that the heuristic function is admissible if it never overestimates the best solution and is often referred to as the A^* method. The technique is implemented as a heap data structure and the order of a traversal can be seen in Figure 1 (c). The complexity of A^* depends on the heuristic used. In an unbounded search space, the worst case running time can reach $O(b^d)$ [5] where d is the shortest path to a solution and b is the branching factor.

2.2. Pruning Rules

The pruning rules of Algorithm 1 determine whether or not S can be fathomed. While the use of a heuristic allows us to approximate the utility of a state without performing a complete search, pruning rules allow us to neglect portions of the tree without affecting the final choice. Many of the pruning techniques are developed in the context of Artificial

Intelligence, however the basic function holds across all applications involving combinatorial optimization.

2.2.1. Lower-Bound

Producing a lower bound on the objective function at each node is the most common way to prune a search space [1]. The lower bound is used to discard sub-problems who's lower bound is no better than the incumbent solution. The value of the lower bound is calculated by relaxing aspects of the problem, and as such, some lower bound calculations may be easier to determine than others. The general technique is to attempt pruning with a lower bound that is easy to calculate before moving on to more complex (but possibly tighter) lower bounds.

2.2.2. Upper-Bound

Many problems have large solution spaces, and while a lower bound is helpful in narrowing down a solution space, an upper-bound can narrow the solution space further. An upper-bound represents the upper limit of the solution space, it means that any problem in the solution space can not be better than a given upper-bound. Knowing the upper-bound of a problem instance can help cut down extra computation, since if the bound is hit, we know we have an optimal solution, and we can stop searching. It is important to mention that having an upper bound does not mean that there exist a solution that equals the upper bound, it simply means there can not be one better.

2.2.3. Alpha-Beta Pruning

Alpha-beta pruning is often used in areas of artificial intelligence that employ a MinMax decision rule for minimizing the possible loss of a worst case scenario. The idea is to maximize the minimum gain, however, issues arise when the number of game states to examine is exponential in the depth of the tree. Alpha-beta pruning, while not eliminating the exponent, effectively cuts it in half by assigning a numerical score to each terminal node [5]. Each terminal node represents the outcome of the player with the next move. The technique is referred to as an adversarial search algorithm because the game tree can

represent many two-player zero-sum games. Two values α and β represent the minimum score that the maximizing player is assured of, and maximum score that the minimizing player is assured of respectively. When the value of a terminal node results in β being less than α, the maximizing player may ignore descendants of the node.

2.2.4. Decision Tree Pruning

For some problems, it is found that decision-tree learning algorithms can generate large trees with no inherent pattern. This can happen when too many irrelevant features are fed into the training model. For example, when measuring the outcome of a tossed coin, the color, time of day, the exact person doing the flipping, etc., have no effect on the outcome. However, if the model is provided these features as input, then the training could be subject to over-fitting. In general, over-fitting is more likely to occur when the hypothesis space and number of input attributes grows [5]. To combat this, we use decision-tree pruning. The general idea is that we start with a full tree and look at a test node whose descendants are only leaf nodes. If testing detects only noise in the data, then we eliminate the node and replace it with the leaf node.

2.3. Branching Strategies

The branching strategy affects the number of children generated and the way in which sub-problems of Algorithm 1 are partitioned. In general, an appropriate branching strategy can limit the number of decisions that are made. This can lead to a significant reduction in the search space. Two common types of branching strategies are binary branching and wide branching [1].

2.3.1. Binary Branching

Binary branching focuses on dividing S from Algorithm 1 into two mutually exclusive sub-problems. This technique is commonly seen in problems like the knapsack problem which seeks a selection of items that maximize cost. In this case, we select some unassigned item and create two branches, one where the item is included in the set, and another where it is not. This still leaves freedom of the user to choose specific branching conditions. For

example, one might choose to branch dichotomously on the edges of the graph as in [6] or on the vertices as in [7].

2.3.2. Non-Binary Branching

Non-binary branching focuses on selecting a singe element from a set of options. This is often used in problems such as the maximum cliques where a set of unused vertices is maintained for each sub-problem and each vertex generates a child set. Non-binary branching allows for potentially large reductions in the search tree by bypassing long sequences of sub problems.

3. Integer Constraint Problems

One of the many types of problems that can be solved by branch-and-bound are Integer Constraint Problems. Integer Constraint Problems at their core are combinatorial optimization problems, thus can be solved by branch-and-bound. For this example we will use a system of linear equations problem, apply an integer constraint, and show how it can be solved using branch-and-bound. In the example below we focus on the case of two equations, but this can be expanded and solved by branch-and-bound.

System of Linear Equations Problem:

Instance:

$Z = f(x, y)$

$h(x, y) \leq c_1$

$g(x, y) \leq c_2$

Question: What is the best assignment of x and y such that Z is maximized?

3.0.1. Creating an Upper-Bound

First thing we must notice about Integer Constraint problems is that their solution space is smaller than their continuous counterparts and is in fact a subset. We can exploit this and create an upper-bound. To create the upper-bound we will relax the integer constraint and solve the relaxed problem. Solving the system of equations we get some optimal values: $x^{'}$, $y^{'}$, and $Z^{'}$. We notice that the integer constraint solution space is a subset of the relaxed

solution space, therefore the largest possible value of Z when applying the integer constraint is $Z = \lfloor Z^{'} \rfloor$. This will now be our upper-bound (UP), which can be used to find the optimal solution.

3.0.2. Combinations

Now we need to turn our problem into a combinatorial optimization problem. To do this, we need to notice that, graphically, the solution in the continuous space is at the intersection of the two lines.

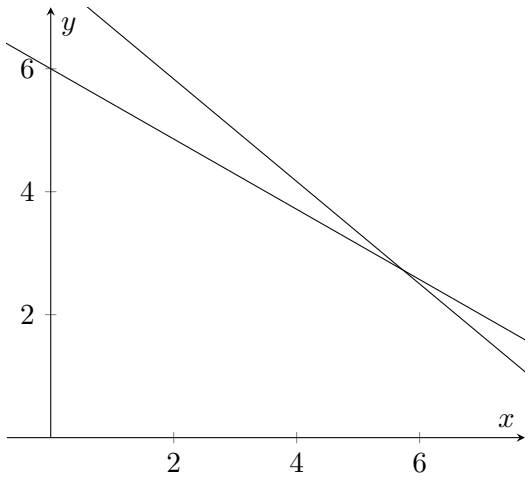


Fig. 2: An example of a system of linear equations.

Therefore, when applying the integer constraint our solution will be near the intersection. To generate all the solution combinations we will use two operations, the floor ($\lfloor \rfloor$) and ceiling ($\lceil \rceil$) operation. For example, take the solution pair in the continuous space $x^{'}$ and $y^{'}$, applying the floor and ceiling to each, we get the candidate integer constraint solutions: $(\lceil x^{'} \rceil, \lceil y^{'} \rceil)$, $(\lfloor x^{'} \rfloor, \lceil y^{'} \rceil)$ $(\lceil x^{'} \rceil, \lfloor y^{'} \rfloor)$, $(\lfloor x^{'} \rfloor, \lfloor y^{'} \rfloor)$. Each floor and ceil operation that we do will be a new constraint or equation on the problem, and can be represented in the following form $x \leq c_i$ or $x \geq c_i$. Each time a new branch is created every previous constraint on that path is considered in the calculation of its variables.

3.0.3. Algorithm

There are many ways to solve this problem using branch-and-bound. One way is to make an algorithm that utilizes a system of equations solver in the continuous space. Then at each branch apply a new constraint/equation, and have the solver compute the maximum value.

4. TSP Better Lower Bound

There are problems like the traveling salesman problem in this example that are easy to model using branch-and-bound but do not see too much improvements. The reason BnB does not work well with TSP is because BnB relies on pseudo parallelism. BnB will choose to expand best possible path each time, and with a problem like TSP, the best possible path will almost always be the last expanded path. This makes the algorithm almost useless since you would expand almost every combination of tour, and essentially brute-force the solution. To combat this researchers have used heuristics, in hopes of making a better choice for a lower-bound.

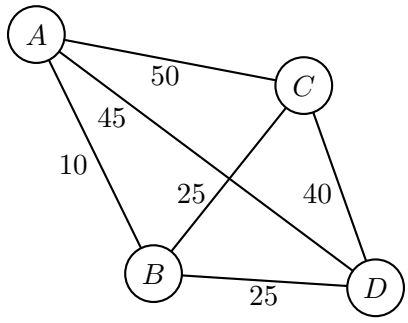


Fig. 3: An example of a TSP Instance

4.0.1. Creating a Better Lower Bound

Firstly, our lower-bound calculation before our heuristic is simply the sum of the tour edges. To improve this we will utilize an encoding scheme known as a graph adjacency matrix, and reduce it [8]. There are two reasons for the reduction, the first is that reduction keeps track of our current state of the tour, and the second is that the cost of reduction is an admissible heuristic in the TSP problem. The idea is that the cost of reduction is equal to

the shortest possible edges left to take on the graph [8]. Note, the shortest edges do not have to be constrained to the tour, they are just simply the shortest edges coming out of each node. To compute the reduction, take the shortest number in each row, then subtract it from each element, finally sum the shortest edges. The output matrix becomes the new state of the child branch and used to compute it's lower-bound.

$$A = \begin{vmatrix} \infty & 10 & 50 & 45 \\ 10 & \infty & 25 & 25 \\ 50 & 25 & \infty & 40 \\ 45 & 25 & 40 & \infty \end{vmatrix} \qquad A' = \begin{vmatrix} \infty & 0 & 40 & 35 \\ 0 & \infty & 15 & 15 \\ 25 & 0 & \infty & 15 \\ 20 & 0 & 15 & \infty \end{vmatrix}$$

Matrix for TSP instance in Figure 5. Row reduced matrix of matrix A

Fig. 4: First reduction matrix of TSP instance in Figure 5. The cost of reduction $R(A) = 10 + 10 + 25 + 25 = 70$

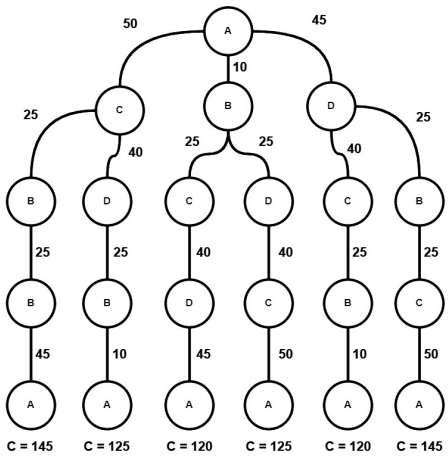
After computing the reduction cost we can now make our lower bound calculation.

$$LB = LB_p + w(p, n) + R(M_n) \tag{1}$$

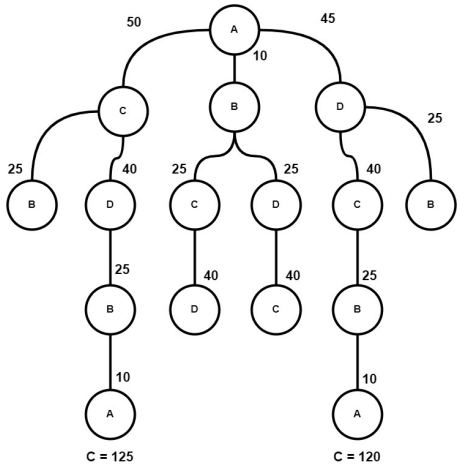
Where LB_p is the lower-bound of the parent, $w(p, n)$ is the weight of the edge between the parent, and $R(M_n)$ is the cost of reduction.

4.0.2. Preformance Gain

It is important to realize that branch and bound is not one size fits all algorithm, so although there may be a problem that can be encoded with an adjacency matrix, it does not mean there exists an admissible heuristic for it. Creating a better lower bound by using a heuristic can provide performance gains, but again it is problem dependent.



Tree generated without heuristic.



Tree generated with row reduction heuristic.

Fig. 7: An example of the trees generated with and without a heuristic for the TSP instance in Figure 5.

5. Advancing Branch And Bound

The three main components of Algorithm 1 (searching, pruning and branching) are generally left unspecified and interchanged by the user. Each of these areas has a significant impact on performance and substantial research has been done to develop them individually even though the branch-and-bound framework has remained largely unchanged. Three viable research directions that can improve performance further are, the formulation of new search strategies, an analysis of how branching affects search and verification phases, and the development of a unified pruning rules. Most work has been focused on the development of pruning rules which has improved the verification phase significantly, however branching and searching strategies remain open for refinement [1]. In the following sections we examine modern techniques in quantum computing and explore how they can be applied to the branch-and-bound framework to increase the search time of NP-complete problems.

6. Traveling Salesman Problem To Ising Model

Recent interest in solving NP-complete problems via adiabatic quantum optimization has led to convenient transformations of Ising formulations. The idea is that if one problem has

a quantum Hamiltonian H_p whose ground state energy encodes the solution to a problem of interest, and another Hamiltonian H_0, whose ground state is "easy" to find and prepare, then we can prepare the system to be in the ground state of H_0 before adiabatically changing it in terms of H_p for a time T [9]. So long as T is large enough and H_0 and H_p do not commute, the system should remain in the resulting ground state for all time. Therefore, measuring the ground state of the system at time T will return a solution to the problem of interest. Though there has been debate on the practicality of such a method, there has been extensive effort in finding formulations of various NPC and NPH problems in terms of Ising models. Here we present one such transformation and show how it can be used to improve the computation time of a generic Branch-and-Bound technique.

6.1. TSP to Hamiltonian Circuit

Consider an instance of the Traveling Salesman problem for the graph $G = (V, E)$ as shown in Figure 8. Each edge uv has a weight W_{uv} associated with it. The set of solutions to this

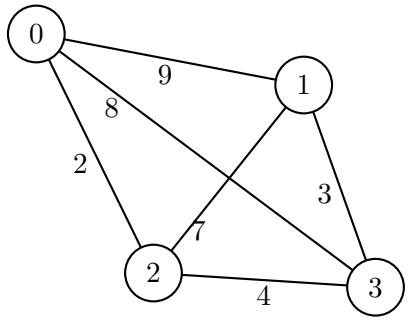


Fig. 8: A graph G representing an instance of the Traveling Salesman Problem. Each node V is connected via a weighted edge $E = W_{uv}$.

graph can be seen in Table I located in the appendix and the shortest Hamiltonian cycle that can be taken has a total distance of 18. One possible route is $2 \to 3 \to 1 \to 0$. This route can be represented as a matrix as shown if Figure 9, where i and p correspond to the city and order of traversal respectively. The solution matrix of Figure 9 can be unpacked into the form of a vector

$$\vec{x} = \langle x_1, x_2, \ldots, x_{N^2} \rangle \tag{2}$$

i \ p	0	1	2	3
0	0	0	0	1
1	0	0	1	0
2	1	0	0	0
3	0	1	0	0

Fig. 9: Solution to TSP in Figure 8 represented in the form of a matrix. The column i represents each node, and the row p corresponds to the nodes position in the traversal. A decision variable value of 1 corresponds to the path being taken.

such that $x_j = x_{N \cdot i+p}$ where N is the number of nodes in the graph. For example, the value at $x_{2,1}$ in the matrix will map to x_9 in the vector. Therefore, the vector representation of the solution to TSP in Figure 8 is $\vec{x} =< 0,0,0,1,0,0,1,0,1,0,0,0,0,1,0,0 >$. Applying the cost function to the eigenvector will result in an eigenvalue (the total distance traveled) of $C(\vec{x}) = 18$.

For nodes forming a traversal such that $x_{i,p}$ and $x_{i,p+1}$ are both equal to one but not connected in G, i.e., $(i,j) \notin E$, then an energy penalty should be imposed in the form $\sum_{i,j \notin E} \sum_p x_{i,p} x_{i,p+1} > 0$ [9]. However, since we are only considering fully connected graphs, this term may be omitted and the net cost of a tour can be calculated from the cost function of Equation 3.

$$C(X) = \sum_{i,j} w_{i,j} \sum_p x_{i,p} x_{j,p+1} \tag{3}$$

Where $w_{i,j}$ is the weight between nodes i and j, and $x_{i,p}x_{j,p+1} = 1$ if the route is actually taken. Inspecting the features of Figure 9, we find that every vertex can only appear once in a cycle and each position must be occupied by a node. This amounts to the two constraints given in Equation 4.

$$\sum_p x_{i,p} = 1 \quad \forall\, i \in \mathit{cities} \quad \text{and} \quad \sum_i x_{i,p} = 1 \quad \forall\, p \in \mathit{route} \tag{4}$$

By imposing these constraints to Equation 3, we find that the objective function to be

minimized takes the form

$$C(X) = \sum_{i,j} w_{i,j} \sum_{p} x_{i,p} x_{j,p+1} + A \sum_{p} (1 - \sum_{i} x_{i,p})^2 + A \sum_{i} (1 - \sum_{p} x_{i,p})^2 \qquad (5)$$

where $A > W_{max}$ is a positive constant set to be much larger than maximum weight encountered during the traversal. Note that the constraints are squared so as to prevent the algorithm from diverging to negative infinity.

To complete the preparation of TSP, we need to convert the cost function of Equation 3 into a quantum Hamiltonian by changing variables from x_i to the Pauli operator σ_i^z (a $2x2$ matrix whose eigenvectors $|1, 0\rangle$ and $|0, -1\rangle$ have eigenvalues (+1,-1)).

$$x_i \rightarrow \frac{1 - \sigma_{i,p}^z}{2} \qquad \Rightarrow \qquad C(X) \rightarrow C(\sigma_{i,p}^z) \equiv H_p \qquad (6)$$

6.2. The Ising Model

In ideal paramagnetism (magnetism where materials are weakly attracted by an external magnetic field) microscopic magnetic dipole moments respond only to an external field. However, in the real world, neighboring atomic dipoles are influenced by each other. When neighboring dipoles align parallel, even in the absence of an external field, we call the material a ferromagnet. The Ising model is a mathematical model used to describe ferromagnetism in terms of statistical mechanics. The model consists of discrete variables that represent the magnetic dipole moments of atomic spins and can have a value of ± 1. The variables are used to describe preference for neighboring dipoles to align parallel or anti-parallel to each other [10].

For example, consider a lattice of particles as shown in Figure 10. Each particle has an associated spin state that can take the orientations of spin-up $|+\rangle$ or spin-down $|-\rangle$ with values $\sigma_k = +1$ and $\sigma_k = -1$ respectively. The energy associated with the particle at lattice site k is found to be

$$E_k = -J \sum_{<i,j>} \sigma_k^i \sigma_k^j \quad [11] \qquad (7)$$

where J is a coupling interaction, σ_k is a discrete variable, and $< i, j >$ means to sum over the nearest neighbors. Note that for parallel spins, $J > 0$, for anti-parallel spins $J < 0$, and

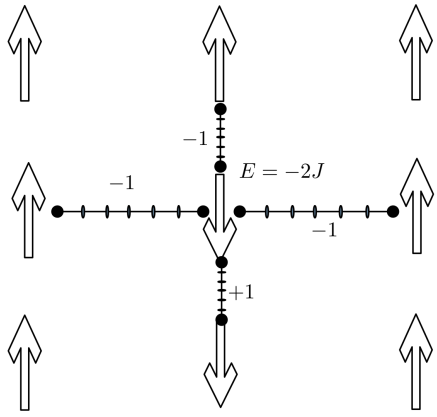


Fig. 10: Atomic spin states arranged into a 2-dimensional lattice structure under periodic boundary conditions. The energy of the particle at lattice site k has a value $E = -2J$ and the total energy of the system is found by summing over the entire structure.

for no interaction $J = 0$. In the presence of a magnetic field $\vec{B}$, the kinetic and potential energy of the system can be represented by the Hamiltonian

$$H_0 = \frac{g_s}{2}\mu_B B \sum_{i=1}^{N} \sigma_z^i - J \sum_{<i,j>} \sigma_z^i \sigma_z^j \tag{8}$$

where the spin g-factor for an electron g_s and the Bohr magneton μ_B are constants, and the discrete variable σ_k has been replaced with the Pauli z-operator σ_z as in Equation 6. There are many techniques for solving generalized Ising models including transfer matrix methods[12], graphical/combinatorial methods[13], and Monte Carlo simulations[10]. However, with the advent of quantum computation, the time independent Schrodinger equation is especially convenient.

6.3. Time Independent Schrodinger Equation

In quantum mechanics, the Schrodinger equation is a partial differential equation that governs the dynamics of the wave function $\Psi(x, t)$ of a particle in a system. In essence, it is an accounting of energy where the kinetic and potential energy are encapsulated in a single operator called a Hamiltonian, and the total energy of the system is set to be the time derivative acting on the wave function. Since the Ising model focuses on the spin

configuration of a particle and neglects the space and time components, we can utilize a time independent version of the Schrodinger equation and treat it as an eigenvalue problem as shown in Equation 9.

$$H|\Psi\rangle = E|\Psi\rangle \tag{9}$$

As the Hamiltonian operator acts on the wave function, a scalar value is produced along with the same vector. In terms of linear algebra, the wave function $|\Psi\rangle$ can be treated as an eigenvector with a corresponding eigenvalue E. Note that each eigenvalue can correspond to a set of eigenvectors and that minimizing Equation 9 is to solve for the ground state energy of the system.

Relating this back to the Traveling Salesman problem, recall that in Equation 6 we transformed the cost function into the same form as the Ising Hamiltonial of Equation 8. We now need to map the solution vector of Equation 2 into qubits to be fed into a quantum computer.

$$\vec{x} = \langle x_1, x_2, \ldots, x_{N^2}\rangle \rightarrow |\Psi\rangle = |x_1\rangle \otimes |x_2\rangle \cdots \otimes |x_{N^2}\rangle \tag{10}$$

Note that we do not require a transformation between the energy of the Schrodinger equation and the distance of the Hamiltonian Circuit because both quantities are scalars. This completes the transformation from the Traveling Salesman problem into the Ising model. The solution for the ground state energy of the Ising Hamiltonian corresponds directly to the minimum tour distance in TSP, however, in this form we are able to exploit the features of quantum mechanics and obtain a solution in far less time.

7. Quantum Improvements

With the recent advancements in quantum computing, access to hardware has become readily available to the general public and quantum algorithms have been shown to perform exponentially better than some of their classical counterparts for select problems. For example, Grover's algorithm can sort through and find an element within an unstructured set using $O(\sqrt{N})$ operations as oppose to the $O(N)$ performance of an exhaustive search. However, for many problems, it's is more beneficial to utilize techniques found in classical

algorithms that take advantage of the structure of a problem. One such technique is Branch-and-bound. By applying the Branch-and-Bound paradigm in a quantum setting, it is possible to achieve a near quadratic improvement beyond that of classical pruning techniques [14].

Though there are several different implementations of quantum computing systems, the most popular is the quantum circuit. The model is based on the concept of a qubit which is analogous to the bit in classical computing. The qubit acts as the basic unit of quantum information and is physically realized as the spin state of a fermion (a particle with a half integer spin; typically 1/2). Since the qubit is measured as the spin state of a particle, it's properties are governed by the wave function of that particle $|\Psi\rangle$. For example, since the quantum spin number of a qubit is 1/2, it can exist in one of two states (spin up $|0\rangle$ or spin down $|1\rangle$) when measured. However, before the measurement, the particle can be in a superposition of both states

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{11}$$

where α and β are probability amplitudes such that $|\alpha|^2 + |\beta|^2 = 1$. It is only when the wave-function is measured that it actually collapse in a definite value of $|0\rangle$ or $|1\rangle$. This means that unlike a classical bit that can discretely take on a value of 0 or 1, the qubit can exist in a state of 0 and 1 simultaneously. We can exploit this feature of quantum mechanics to increase the speed of classical algorithms by placing the qubit into a superposition of states and letting the system evolve in time before taking a measurement.

7.1. Classical vs Quantum Walk

Consider the case of a classical random walk. In this example we search for a marked node within a binary tree of size n nodes. If we have no indication of where in the tree the target might be, then a sure proof way of finding it is to visit each node, one step at a time, with 50% probability of taking a step to the left and 50% probability of taking a step to the right. Naturally, for every step away from the origin, there is an equal probability of taking a step back toward the origin, and over time we will obtain a normal Gaussian distribution of steps as shown in Figure 11. The classical random walk distribution exhibits a variance

of

$$\sigma^2 \propto n \tag{12}$$

which is to say that there is a very low probability of hitting the target if it is far away from the origin. Since each step is based on a set of definite states, the randomness of the traversal is caused by a stochastic transition between states. Now consider a random walk in a quantum setting. In this case, the randomness comes from the quantum superposition imposed by Equation 11, a reversible unitary evolution (typically in the form of a Hadamard coin), and a collapse of the wave-function when a measurement is actually taken. Another difference between the quantum and classical random walk is the time at which a measurement is taken. In the classical version, it makes no difference to take measurements at each iteration, however in the quantum version, if a measurement of the wave-function is taken at each iteration, then the distribution converges to the classical Gaussian form. However, if the wave-function is left to interfere with itself before a measurement is taken, then the probability distribution is much more intricate, as can be seen in Figure 11. It can be shown the after n steps, the quantum walk obtains a variance of

$$\sigma^2 \propto n^2 \quad [15] \tag{13}$$

Therefore, the quantum walk propagates quadratically faster than the classical walk, and this speed increase can be applied to the branch-and-bound framework. From Figure 11, if the wave-function is placed in a superposition, then the distribution exhibits destructive interference about the origin. However, if the wave-function begins in a pure spin-up or spin-down configuration, then it is possible to create a bias toward any particular direction as shown in Figure 12 of the appendix. Therefore, we can still utilize the heuristic methods found in the classical branch-and-bound technique.

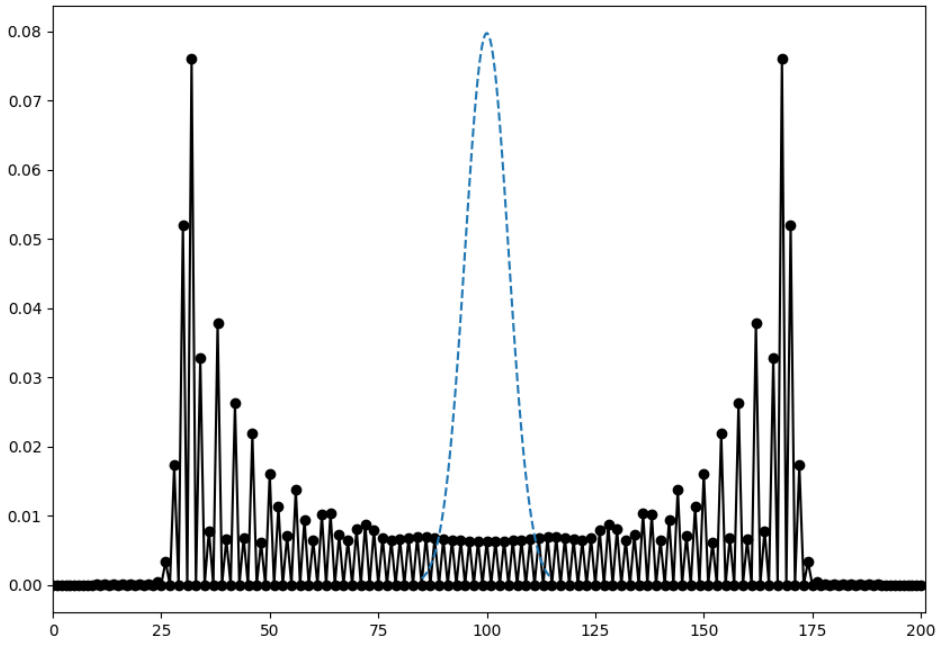


Fig. 11: Difference between quantum and classical random walk. The blue dotted line represents the Gaussian distribution created by the classical walk and has a variance $\sigma^2 \propto n$. The black distribution is the result of the quantum random walk where the initial wave-function placed into a superposition $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and acted on by a balanced Hadamard coin. The quantum distribution exhibits a variance of $\sigma^2 \propto n^2$ with de-constructive interference about the origin and constructive interference at the edges of propagation.

To summarize the results, consider three individual cases of solving combinatorial problems with Hamiltonians similar to that of Equation 8. One case is solved using the classical branch-and-bound technique, one is solved using a quantum walk, and one is solved using branch-and-bound in a quantum setting. If the classical branch-and-bound techniques are applied to solve for the largest known ground states of the Beransconi model, the runtime is roughly $O(2^{0.79n})$ [16]. If we apply a quantum walk to classical algorithms that solve the Sherrington-Kirkpatrick model (an Ising model with long range anti-ferrmagnetic couplings) then it has been shown that we can increase runtime to approximately $O(2^{0.41n})$ [17]. Finally, by applying quantum branch-and-bound algorithms to the same model, it has been shown

that the runtime can be increased further to $O(2^{0.226n})$ [14].

8. Conclusion

The branch-and-bound framework is an old technique that has proven to be extremely useful even when applied to new technology. The technique acts as a recipe for handling issues encountered in combinatorial optimization problems by helping the user make informed decisions when applying matrix reductions or producing lower-bounds with which to branch on, etc. However, though the general algorithm can be used in a wide range of problems, it is still up to the user to determine which sub-routine is most beneficial for the problem at hand. This relies on experience and trial-and-error, but when used properly, branch-and-bound can produce exceptional results.

References

[1] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," Discrete Optimization, vol. 19, pp. 79–102, 2016.

[2] L. Huang, X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai, and L. Shi, "Branch and bound in mixed integer linear programming problems: A survey of techniques and trends," arXiv preprint arXiv:2111.06257, 2021.

[3] N. Virdi, "Development of a connected and autonomous vehicle modelling framework, with implementation in evaluating transport network impacts and safety," Ph.D. dissertation, University of New South Wales, 2020.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. MIT press, 2009.

[5] S. Russell and P. Norvig, "Artificial intelligence: A modern approach. third edit," Prentice Hall. doi, vol. 10, pp. B978–012 161 964, 2010.

[6] F. Rendl, G. Rinaldi, and A. Wiegele, "Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations," Mathematical Programming, vol. 121, no. 2, pp. 307–335, 2010.

[7] F. Baccari, C. Gogolin, P. Wittek, and A. Acín, "Verifying the output of quantum optimizers with ground-state energy lower bounds," Physical Review Research, vol. 2, no. 4, p. 043163, 2020.

[8] A. Rastogi, A. K. Shrivastava, N. Payal, R. Singh, and A. Sharma, "A proposed solution to travelling

salesman problem using branch and bound," International Journal of Computer Applications, vol. 65, no. 5, 2013.

[9] A. Lucas, "Ising formulations of many np problems," Frontiers in physics, vol. 2, p. 5, 2014.

[10] D. V. Schroeder, "Thermal physics," 2011.

[11] E. Ising, "Beitrag zur theorie des ferromagnetismus," Zeitschrift für Physik, vol. 31, no. 1, pp. 253–258, 1925.

[12] L. Onsager, "Crystal statistics. i. a two-dimensional model with an order-disorder transition," Physical Review, vol. 65, no. 3-4, p. 117, 1944.

[13] R. Feynman, "Statistical mechanics, a set of lectures, california, institute of technology," 1972.

[14] A. Montanaro, "Quantum speedup of branch-and-bound algorithms," Physical Review Research, vol. 2, no. 1, p. 013056, 2020.

[15] J. Kempe, "Quantum random walks: an introductory overview," Contemporary Physics, vol. 44, no. 4, pp. 307–327, 2003.

[16] T. Packebusch and S. Mertens, "Low autocorrelation binary sequences," Journal of Physics A: Mathematical and Theoretical, vol. 49, no. 16, p. 165001, 2016.

[17] A. Callison, N. Chancellor, F. Mintert, and V. Kendon, "Finding spin glass ground states using quantum walks," New Journal of Physics, vol. 21, no. 12, p. 123022, 2019.

Appendix

$0 \to 1 \to 2 \to 3 \to 0$	$\text{dist} = 28$	$1 \to 0 \to 2 \to 3 \to 1$	$\text{dist} = 18$
$0 \to 1 \to 3 \to 2 \to 0$	$\text{dist} = 18$	$1 \to 0 \to 3 \to 2 \to 1$	$\text{dist} = 28$
$0 \to 2 \to 1 \to 3 \to 0$	$\text{dist} = 20$	$1 \to 2 \to 0 \to 3 \to 1$	$\text{dist} = 20$
$0 \to 2 \to 3 \to 1 \to 0$	$\text{dist} = 18$	$1 \to 2 \to 3 \to 0 \to 1$	$\text{dist} = 28$
$0 \to 3 \to 1 \to 2 \to 0$	$\text{dist} = 20$	$1 \to 3 \to 0 \to 2 \to 1$	$\text{dist} = 20$
$0 \to 3 \to 2 \to 1 \to 0$	$\text{dist} = 28$	$1 \to 3 \to 2 \to 0 \to 1$	$\text{dist} = 18$
$2 \to 0 \to 1 \to 3 \to 2$	$\text{dist} = 18$	$3 \to 0 \to 1 \to 2 \to 3$	$\text{dist} = 28$
$2 \to 0 \to 3 \to 1 \to 2$	$\text{dist} = 20$	$3 \to 0 \to 2 \to 1 \to 3$	$\text{dist} = 20$
$2 \to 1 \to 0 \to 3 \to 2$	$\text{dist} = 28$	$3 \to 1 \to 0 \to 2 \to 3$	$\text{dist} = 18$
$2 \to 1 \to 3 \to 0 \to 2$	$\text{dist} = 20$	$3 \to 1 \to 2 \to 0 \to 3$	$\text{dist} = 20$
$2 \to 3 \to 0 \to 1 \to 2$	$\text{dist} = 28$	$3 \to 2 \to 0 \to 1 \to 3$	$\text{dist} = 18$
$2 \to 3 \to 1 \to 0 \to 2$	$\text{dist} = 18$	$3 \to 2 \to 1 \to 0 \to 3$	$\text{dist} = 28$

TABLE I: Solutions to TSP in Figure 8. Each eigenvector forms a tour about the graph with an associated eigenvalue corresponding to the total distance traveled. The optimal ground state eigenvalue 18 corresponds to a set of possible eigenvectors.

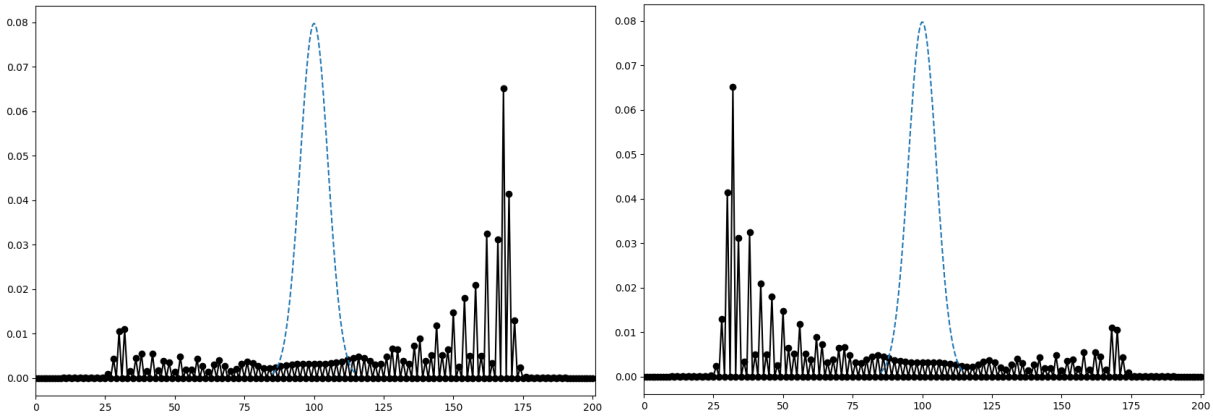


Fig. 12: Plot of the probability distributions of a classical random walk and a quantum random walk. The blue dotted line in each plot is the result of a classical Gaussian distribution. In the left plot, the black distribution is the result of preparing the quantum walk with a particle in a pure spin-up configuration $|\Psi\rangle = |0\rangle$. In the right plot, the particle is prepared in a spin-down configuration $|\Psi\rangle = |1\rangle$. In both cases, the quantum distribution exhibits de-constructive interference about the origin, however it is possible to impose a bias toward a particular direction simply by initializing the system with different weights in α and β from Equation 11.