Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

3-1-2023 12:30 PM

A Modified Hopfield Network for the K-Median Problem

Cody Rossiter, *The University of Western Ontario*

Supervisor: Solis-Oba, Roberto, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

© Cody Rossiter 2023

Follow this and additional works at: https://ir.lib.uwo.ca/etd

 Part of the Theory and Algorithms Commons

Recommended Citation

Rossiter, Cody, "A Modified Hopfield Network for the K-Median Problem" (2023). *Electronic Thesis and Dissertation Repository*. 9152.
https://ir.lib.uwo.ca/etd/9152

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

Abstract

The k-median problem is a clustering problem where given n locations one wants to select k locations such that the total distance between every non-selected location and its nearest selected location is minimized. The problem has applications in several fields, including network design, resource allocation, and data mining.

There is currently limited research on applying neural networks to combinatorial optimization problems and we contribute by presenting a modified Hopfield network for the k-median problem. Hopfield networks are a type of neural network that can be applied to combinatorial optimization problems but often run slowly and produce poor solutions.

Our modifications address these speed and solution quality issues. We test our approach by comparing it with Haralampiev's competition-based neural network and the local search algorithm of Arya, Garg, Khandekar, Meyerson, Munagala, and Pandit. Our network produces competitive results while running significantly faster on larger ($n \geq 300$) problems. Furthermore, our network scales well and produces solutions for problems where $n = 13509$.

Keywords: Hopfield Networks, K-Median Problem, Neural Networks, Combinatorial Optimization

Summary for Lay Audience

Combinatorial optimization problems are an important field of computer science. Here we are given a group of objects and we want to select the best subgroup of objects in order to solve a problem. We focus on the k-median problem where we are given several locations and we want to select k of them. We call these selected locations facilities and our goal is to select facilities such that the total distance from all locations to their nearest facility is minimized. A practical application of the k-median problem is placing schools so that student travel time is minimized.

Unfortunately this problem is difficult to solve and it is not known if there is a fast way to always produce the best solution. Instead, we can develop an approach that will run quickly and produce a good solution instead of the best solution.

We use a type of artificial neural network called a Hopfield network to produce solutions for the k-median problem. Artificial neural networks solve problems by emulating how neurons in the brain function. We do this by creating simple artificial neurons and connecting them together so that their output can be used to solve problems. In Hopfield networks, each neuron has a value that can be 0 or 1 and changes over time based on the other values in the network. Once the neuron values stop changing, we use the neuron values to determine which facilities to select.

However, Hopfield networks often run slowly and produce poor solutions. To address this we make several modifications to the network. We demonstrate the effectiveness of our modified Hopfield network by using it to solve several k-median problems. We also compare the performance of our modified Hopfield network with two other approaches and we find that our network produces competitive solutions. Furthermore, our network scales well as we were able to solve problems that were too large for the other two approaches solve.

Contents

Abstract ii

Summary for Lay Audience iii

List of Figures vii

List of Tables viii

List of Appendices ix

1 Introduction 1
- 1.1 Our Contributions . . . 3
- 1.2 Outline of the Thesis . . . 3

2 K-Median Problem 4
- 2.1 Problem Overview . . . 4
- 2.2 Integer Linear Program Formulation . . . 5
- 2.3 Algorithms for the K-Median Problem . . . 6
  - 2.3.1 Exact Algorithms . . . 6
  - 2.3.2 Approximation Algorithms . . . 6
  - 2.3.3 Parameterized Algorithms . . . 8
  - 2.3.4 Metaheuristics and Machine Learning . . . 8

3 Hopfield Networks 10
- 3.1 Neural Networks . . . 10
- 3.2 Hopfield Networks . . . 11
- 3.3 Hopfield Networks for Content Addressable Memory . . . 13
- 3.4 Applications in Combinatorial Optimization . . . 15
- 3.5 Travelling Salesman Problem . . . 17
- 3.6 Issues . . . 19
- 3.7 Applications of and Modifications to Hopfield Networks . . . 20

4 K-Median Hopfield Network Formulation 22

5 The Modified Hopfield Network 27
- 5.1 Modified Hopfield Network Update Operations . . . 30
- 5.2 Modified Hopfield Network Algorithm . . . 31

5.3 Extensions to the Algorithm . . . 34
5.3.1 Reinforcement Learning Facility Selection . . . 34
5.3.2 State Space Exploration . . . 35

6 Experimental Results **37**
6.1 Haralampiev's Competition-Based Neural Network . . . 37
6.1.1 Overview . . . 37
6.1.2 Algorithm . . . 41
6.2 Local Search Algorithm . . . 44
6.3 Datasets . . . 46
6.3.1 Random Point Datasets . . . 46
6.3.2 Cities USCA312 Dataset . . . 46
6.3.3 OR-Library P-Median Dataset . . . 46
6.3.4 TSPLib Dataset . . . 46
6.4 Testing Environment . . . 47
6.5 Results . . . 47
6.6 Analysis . . . 47

7 Conclusions **51**
7.1 Future Work . . . 52

Bibliography **53**

A K-Median Hopfield Network Energy Equation **57**
A.1 Constraint Term 1 . . . 57
A.2 Constraint Term 2 . . . 58
A.3 Constraint Term 3 . . . 58
A.4 Constraint Term 4 . . . 60
A.5 Constraint Term 5 . . . 61
A.6 Connection and Input Bias Values . . . 62

B Full Dataset Test Results **65**
B.1 Random-Small Dataset . . . 66
B.1.1 Hopfield Network and Haralampiev Network Results . . . 66
B.1.2 Hopfield Network and Local Search Results . . . 70
B.2 Random-Large Dataset . . . 74
B.2.1 Hopfield Network and Haralampiev Network Results . . . 74
B.2.2 Hopfield Network and Local Search Results . . . 76
B.3 Cities USCA312 Dataset Results . . . 78
B.3.1 Hopfield Network and Haralampiev Network Results . . . 78
B.3.2 Hopfield Network and Local Search Results . . . 88
B.4 OR-Library P-Median Dataset Results . . . 98
B.4.1 Hopfield Network and Haralampiev Network Results . . . 98
B.4.2 Hopfield Network and Local Search Results . . . 101
B.5 TSPLib Dataset Results . . . 104

B.5.1 Hopfield Network and Local Search Results 104
B.5.2 Hopfield Network usa13509 Results 107

Curriculum Vitae **108**

List of Figures

2.1 K-Median Problem Example 5

3.1 Feedforward Neural Network Diagram 11
3.2 Example Hopfield Network for Content Addressable Memory 14

5.1 Hopfield Network Connection Diagram for the K-Median Problem 28
5.2 Example Search Tree for our Modified Hopfield Network 36

6.1 Haralampiev Network Neuron Layout and Example Solution 38
6.2 Haralampiev Network Connection Diagram 40

List of Tables

2.1 Example Distance Matrix for the k-median Problem 4

3.1 Example Values for a Hopfield Network Connection Matrix T 14
3.2 Example Hopfield Network Calculations for Content Addressable Memory . . . 14
3.3 Visualization of a Travelling Salesman Tour 18

4.1 Hopfield Network Neuron Representations for a Three Location k-median Problem 23

5.1 Example Hopfield Network Distance Matrices D and D' 29

6.1 Example Haralampiev Network Probability Values 43
6.2 Modified Hopfield Network and Haralampiev Network Test Results 48
6.3 Modified Hopfield Network and Local Search Algorithm Test Results 48
6.4 Modified Hopfield Network Results for the TSPLib usa13509 Dataset 49

List of Appendices

Appendix A K-Median Hopfield Network Energy Equation 57
Appendix B Full Dataset Test Results 65

Chapter 1

Introduction

Combinatorial optimization is an important field of computer science. In a combinatorial optimization problem one is given a set of objects and problem constraints and the goal is to select the best subset of objects that satisfy the constraints. Each subset of objects is called a solution and a cost function is defined to give each solution a value. The best solution is called the optimal solution and depending on the problem it represents either the minimum or maximum of the cost function.

Examples include the travelling salesman problem where given a set of locations one wants to find the shortest tour that visits all locations once, and the knapsack problem where given a set of items one wants to fill a knapsack with the most valuable items without exceeding the knapsacks maximum weight limit.

In this thesis we focus on the k-median problem where given a set of locations one wants to select k locations such that the total distance between non-selected locations and their nearest selected location is minimized. Practical applications of the k-median problem include placing schools so that student travel time is minimized [34], grouping similar machine parts so that they may be manufactured more efficiently [45], and determining what variations of a product should be produced in order to best satisfy customer demands [31].

A well studied variation is the metric k-median problem. In this instance the distance between two locations is symmetrical and satisfies the triangle inequality. Formally, if $D_{i,j}$ represents the distance from location i to location j then $D_{i,j} = D_{j,i}$ and for any three locations i, j, and k the following equation holds $D_{i,k} \leq D_{i,j} + D_{j,k}$. We will focus on the metric k-median problem in this thesis.

Unfortunately, this problem is NP-Hard [25] and there is no known polynomial-time algorithm for finding an optimal solution. As a result computing the optimal solution can require a significant amount of time for small problems and an infeasible amount of time for large problems as each problem contains $\frac{n!}{(n-k)!k!}$ potential solutions. One way to address this is to create polynomial-time algorithms that produce satisfactory or near optimal solutions. While these solutions are not optimal, they can perform well in practice [32].

A type of polynomial-time approach to deal with NP-Hard problems is heuristics. Heuristics can produce solutions within a reasonable amount of time but we do not know how close these solutions are to optimal ones. The reason why we usually cannot determine how close solutions computed by heuristics are to optimal ones is because heuristics are very complicated. Heuristics are constructed specifically for a given problem and rely on using techniques

or methods that have been shown to be effective through empirical testing.

An example heuristic is simulated annealing [26]. Here each solution to a combinatorial optimization problem is called a state and an energy function gives each state an energy value such that states representing good or optimal solutions have a low energy value. The heuristic will transition from state to state until an acceptable solution is found. At each step the heuristic will select a state to transition to and generate a probability to accept that transition. This probability will be based on the difference in energy between the two states with decreases in energy resulting in higher probabilities and a temperature variable that when high increases the chance of accepting any state and when low only allows state changes that decrease energy. The heuristic then explores the state space while the temperature is high before settling into a low energy state when the temperature is low.

For some heuristics we can show that the produced solutions will be of a certain quality. These are called approximation algorithms and they are constructed to produce an approximate solution for a problem that is guaranteed to be within some factor α of an optimal solution. This factor α is called the approximation ratio and the current best approximation ratio for the metric k-median problem is $2.611 + \epsilon$ for any $\epsilon > 0$ [8]. Local search algorithms are a type of approximation algorithm which compute an initial solution and repeatedly swap objects of the solution with objects not in the solution in order to improve the solution value. A solution is returned when no further improvements can be made through the swaps. For the metric k-median problem the best known local search algorithm for this problem has an approximation ratio of $3 + 2/p$ where p is the number of objects that may be swapped at one time [2].

Neural networks are another approach for solving combinatorial optimization problems [20]. These networks attempt to emulate the natural computing power of the brain by creating simple artificial neurons and linking them together such that their output can be used to solve a problem. One of the more popular network types is the feedforward neural network which takes in training data, uses it to set neuron connections in a process called training, and then applies the trained neurons to some problem. A neural network could be trained to solve a combinatorial optimization problem by providing it with numerous solved examples. After the neurons have been trained the network should be able to solve new problem instances.

Our work uses neural networks to solve the k-median problem. The neural network we use is the Hopfield network [19]. This network functions differently compared to the aforementioned feedforward neural network; instead of outputting a solution, each neuron contains a state value and a solution is derived from these state values. The state values can be between 0 and 1 and a solution to a general combinatorial optimization problem can be derived as follows: map each neuron to an object, for each neuron that has the state value 1 select the corresponding object, and return the selected objects as a solution.

Each neuron in a Hopfield network will repeatedly evaluate its state value and send output to its connected neurons. When evaluating the state value a neuron will sum the input it receives and if a certain threshold is reached, it will increase the state value otherwise it will decrease the state value. This continues until the neurons stop changing their state values and the network is said to have stabilized. It is from this stabilized state that we derive a solution from the neuron state values.

1.1 Our Contributions

Our main contribution is a new Hopfield network for solving the k-median problem. We modify the Hopfield network operations in order to address the stability and solution quality issues present in standard Hopfield networks. Additionally, we structure our modified Hopfield network operations as a series of matrix operations which allows us to leverage the computing power of GPUs. We expand on the previously limited research of applying Hopfield networks to combinatorial optimization problems and demonstrate how a few modifications can produce a competitive solver from an underperforming Hopfield network. We empirically tested our approach and compared the results with both Haralampiev's competition-based neural network [16, 17, 18] and Arya et al.'s k-median local search algorithm [2]. Our approach is significantly faster than Haralampiev's network for large problem sizes while providing competitive results. For large problems sizes such as $n = 800$ our network stabilizes up to 145 times faster than Haralampiev's network. When compared with Arya et al.'s local search algorithm our network returns a better solution in 76% of tests when $n \geq 300$ and 86% of tests when both $n \geq 300$ and $k \geq 20$. We also demonstrate that our network scales well to larger problem sizes by providing solutions to problems where $n = 13509$. This is significant as the two compared approaches struggled to produce solutions to problems where $n \geq 1000$.

1.2 Outline of the Thesis

In Chapter 2 we further explain the k-median problem and provide an integer linear program formulation for it. We then review approaches that have been proposed in the literature for solving the problem, including approximation algorithms, parameterized algorithms, metaheuristics, and machine learning.

In Chapter 3 we provide a brief overview of neural networks before discussing Hopfield networks in detail. We describe their construction, use in problem solving, and the issues associated with them. We then go over modifications that have been applied to Hopfield networks in order to improve performance on several combinatorial optimization problems.

In Chapter 4 we construct a Hopfield network for the k-median problem and briefly discuss its performance and issues.

In Chapter 5 we present a new network based on the previous Hopfield network. We demonstrate how to modify the network in order to improve solution quality and algorithm runtime.

In Chapter 6 we present our experimental results. We compare our modified Hopfield network with Haralampiev's network and Arya et al.'s local search algorithm using five datasets. Three of the datasets contain test data created for this thesis: datasets one and two consist of randomly selected points on a 2D plane and dataset three consists of North American city coordinates from John Burkardt's USCA312 dataset [6]. We use an integer linear program solver to find optimal solutions to the generated problems. The other two datasets are the OR-Library p-median tests [5] and a subset of the TSPLib dataset [37] which has been adapted for the k-median problem by García et al. [15].

In Chapter 7 we summarize our work and present future research directions.

Chapter 2

K-Median Problem

2.1 Problem Overview

The k-median problem is a classical clustering problem: Given a set of n locations we want to select a subset of k of them such that the sum of distances between every non-selected location and their closest selected location is minimized. Non-selected locations are customarily called clients and selected locations are called facilities. We say that a client is served by its closest facility, breaking ties arbitrarily so each client is served by a unique facility.

Figure 2.1 shows an instance of the metric k-median problem and Table 2.1 contains the distances between the locations. In this example there are five potential locations on a 2D plane for the facilities and the goal is to optimally place facilities at two locations. Each location can be identified by an (x, y) coordinate pair and the Euclidean distance between locations i and j is calculated with $D_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

In this case the problem size is small enough that we can find an optimal solution by hand. Visually we see two potential clusters consisting of locations $\{1, 2\}$ and $\{3, 4, 5\}$. We use the distance matrix in Table 2.1 to evaluate the potential facility locations and we select locations 1 and 4 as the optimal solution with a total distance of $D_{1,2} + D_{4,5} + D_{4,3} = 0.64$. Note that a second optimal solution exists that uses the locations 2 and 4 as facilities.

Some related problems are the facility location problem and its variations. In the uncapacitated facility location problem there is a cost to open a facility and no limit on how many facilities can be opened. In this problem the goal is to minimize the total distance from clients to their nearest facility plus the total cost of the opened facilities. In the capacitated facility

	1	2	3	4	5
1	0	0.14	0.63	0.64	0.67
2	0.14	0	0.58	0.67	0.73
3	0.63	0.58	0	0.36	0.50
4	0.64	0.67	0.36	0	0.14
5	0.67	0.73	0.50	0.14	0

Table 2.1: The distance matrix for the locations in Figure 2.1.

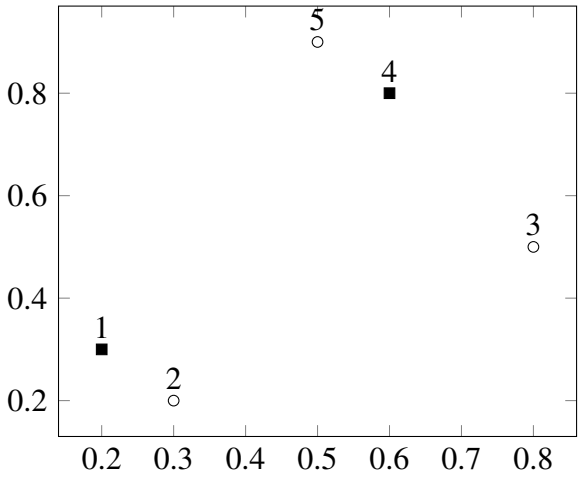


Figure 2.1: An example of a solved k-median problem for $n = 5$ and $k = 2$. Squares represent facilities and circles represent clients.

location problem there is a constraint limiting the number of clients that an individual facility can serve and there might be any number of clients at each location. In the connected facility location problem there is an extra cost for opening a facility based on that facility's distance to other opened facilities. In the k-center problem the goal is to minimize the maximum distance between a facility and a client instead of the sum of distances between clients and their facilities.

2.2 Integer Linear Program Formulation

We can formalize the k-median problem as an integer linear program as follows: Let X be a set of n data points and F be the set of facilities. The distance between two data points X_i and X_j will be represented by $D_{i,j}$. We use binary variables $y_{i,j}$ that take the value 1 to indicate that client X_i is being served by facility X_j. We also use the binary variable F_i that takes the value 1 if data point X_i is selected as a facility.

The k-median problem can be defined with the following integer linear program:

$$\text{Minimize:} \quad \sum_{X_i \in X} \sum_{X_j \in X} y_{i,j} D_{i,j} \tag{2.1}$$

$$\text{Subject To:} \quad \sum_{X_j \in X} y_{i,j} = 1 \qquad \text{for each } X_i \in X \tag{2.2}$$

$$\sum_{X_i \in X} F_i \leq k \tag{2.3}$$

$$y_{i,j} \leq F_j \qquad \text{for each } X_i, X_j \in X \tag{2.4}$$

$$F_i, y_{i,j} \in \{0, 1\} \tag{2.5}$$

Equation (2.1) represents the goal of minimizing the total distance from the clients to their nearest facility: Every X_i and X_j pair is evaluated and if X_j serves X_i then the distance between the two locations is added to the total distance.

Several constraint terms are needed to ensure that a solution is valid. The first constraint, Equation (2.2), enforces that every client X_i is served by exactly one facility X_j. This is satisfied for a client X_i when there is exactly one client-facility variable $y_{i,j}$ with the value 1. Equation (2.3) ensures that at most k facilities are selected and is satisfied when the total number of facility variables with the value 1 is less than or equal to k. The third constraint is Equation (2.4) which ensures that a location must be a facility if it is serving a client. There are three ways to satisfy this equation for a pair of locations X_i and X_j: the client X_i is being served by a facility X_j which gives us $1 \leq 1$, the client or facility X_i is not being served by a facility X_j which gives us $0 \leq 1$, or the client or facility X_i is not being served by a client X_j which gives us $0 \leq 0$. The final constraint is Equation (2.5) which restricts the facility variables F_i and the client-facility variables $y_{i,j}$ to the values of 0 or 1.

2.3 Algorithms for the K-Median Problem

2.3.1 Exact Algorithms

A popular approach for solving the k-median problem is the branch and bound algorithm [14, 23, 33]. This algorithm functions by representing the search space of potential solutions as a tree with each node representing a partial or complete candidate solution. At each step the algorithm will determine which nodes to explore further by calculating if the value of the corresponding candidate solutions are within some estimated bound of the optimal answer. If a node passes this check it is expanded by adding child nodes whose corresponding candidate solutions are derived from the parent node. This expansion process is repeated until the optimal solution is found.

Another exact approach is provided by Senne et al. [40] who apply the branch and price algorithm to the k-median problem. The branch and price algorithm is used to solve integer linear programs with many variables and, like the branch and bound algorithm, it creates a search tree of solutions that it expands over time. It starts by generating a linear program relaxation of the original problem which will provide good bounds but also contain many variables. This relaxation is called the Master Problem and the algorithm will focus on solving a modified version called the Restricted Master Problem. The Restricted Master Problem will use a subset of the variables and at each step the algorithm will solve this smaller problem. Afterwards, the algorithm will try to find variables to add back that will improve the value of the objective function. This process repeats until no more variables can be introduced at which point the algorithm can return a solution or branch and try different variables.

2.3.2 Approximation Algorithms

With the k-median problem being NP-Hard and therefore computationally difficult to solve, several approximation algorithms have been developed. The simplest algorithm is the forward greedy algorithm [11]. The algorithm starts with no facilities selected and at each step the

algorithm selects the facility that will result in the shortest increase of the overall distance. It continues until k facilities are selected. This algorithm results in an approximation ratio of $\Omega(n)$.

The reverse greedy algorithm [11] starts with all locations selected as facilities. From this state the algorithm removes a facility based on which removal decreases the overall distance the most. It continues until k facilities remain and results in an approximation ratio between $\Omega(\log n / \log \log n)$ and $O(\log n)$ for the metric k-median problem.

An early non-greedy attempt at approximating the k-median problem was a linear program rounding approach by Lin and Vitter [29]. Their approach uses a three phase method to solve an integer linear program: the first phase solves a linear program built by relaxing the integrality constraints of the integer linear program and produces a fractional solution, the second phase uses that fractional solution to create an integer linear program that violates constraint 2.3, and the final phase solves this latter integer linear program through the use of a random sampling technique. This provides an approximation ratio of $1 + \epsilon$, for any $\epsilon > 0$, however the solution is infeasible as it opens up to $(1 + 1/\epsilon)(\ln n + 1)k$ facilities.

Another approach that provides a feasible solution is Bartal's randomized approximation algorithm [3, 4]. This algorithm uses a probabilistic method to simplify the metric space of the k-median problem into a k-hierarchical well separated tree. From here the simplified problem can be solved with tree-based algorithms and the approach provides an $O(\log k \log \log k)$ approximation for the metric k-median problem.

Charikar et al. [9] then build on Lin and Vitter's approach and provided the first constant factor approximation algorithm for the metric k-median problem. Their algorithm follows a similar three phase method of solving a linear program and using the fractional solution to produce and solve an integer linear program. By making improvements to the second phase they provide an algorithm with an approximation ratio of $6\frac{2}{3}$.

Jain and Vazirani provided a primal-dual approximation method for the k-median problem and the uncapacitated facility location problem [22]. Their algorithm runs in two distinct phases: the first phase focuses on finding an integral primal solution satisfying the primal complementary slackness conditions; the second phase ensures that all complementary slackness conditions are satisfied, but the primal solution may be infeasible which requires that the primal conditions be relaxed by a factor of 3. This algorithm provides an approximation ratio of 6 for the metric k-median problem and an approximation ratio of 3 for the metric uncapacitated facility location problem.

A local search approach is provided by Arya et al. [2] for the metric k-median problem. This algorithm operates by examining a current solution and checking if swapping a facility with a client will improve the solution. The swapping process repeats until the algorithm can no longer improve the solution. If the algorithm is constrained to swapping one facility at a time it has an approximation ratio of 5 but if one allows simultaneous swaps of p facilities with p clients the approximation ratio is $3 + 2/p$.

The current best approximation algorithm for the metric k-median problem is provided by Byrka et al. [8] with an approximation ratio of $2.611 + \epsilon$ for any $\epsilon > 0$. They built on the previous best approach of Li and Svensson [27] whose algorithm provided an approximation ratio of $1 + \sqrt{3} + \epsilon$. The algorithm first provides an approximate solution to an instance of the problem with $k + 1$ facilities and then uses that to construct a solution with k facilities.

2.3.3 Parameterized Algorithms

An approach for solving the k-median problem that combines approximation algorithms and fixed-parameter tractable (FPT) algorithms is provided by Cohen-Addad et al. [12]. FPT algorithms relax the requirement that they must run in polynomial time. Instead a parameter k is chosen such that the runtime of the algorithm is of the form $f(k)n^{O(1)}$, where $f(k)$ is an arbitrary function and n is the size of the input. FPT and approximation algorithms can be combined so that an FPT algorithm finds an approximate solution instead of an optimal solution. Cohen-Addad et al. take this combined approach and present a $1+2/e+\epsilon$ FPT approximation algorithm for the k-median problem where e is the Euler number and $\epsilon > 0$. Their algorithm has three main steps: the first step reduces the size of the client set by consolidating nearby clients, then the algorithm guesses which clients would be the closest to an optimal facility and searches around these clients to find the best set of k facilities. The running time of the algorithm is $O(\epsilon^{-2}k \log k)^k \cdot n^{O(1)}$.

Byrka et al. also study the use of FPT algorithms for approximating the k-median problem by approximating a closely related problem [7]. Instead of focusing on how many facilities to open with the parameter k they look at how many facilities to close and define the dual parameter $\ell = |F| - k$ where F is a set of facilities. They call this new problem the CO-ℓ-Median problem and provide an FPT approximation algorithm with an approximation ratio of $(1 + \epsilon)$ for any $\epsilon > 0$ and a running time of $2^{\ell log(\ell/\epsilon)} \cdot n^{0(1)}$.

2.3.4 Metaheuristics and Machine Learning

Research on the k-median problem extends past approximation algorithms and includes approaches such as metaheuristics and machine learning. The following is a small sample of other approaches to solving and approximating the k-median problem.

Evolutionary Algorithms (EA) are a metaheuristic approach that attempts to mimic the process of natural selection in order to solve a problem. This can be applied to approximating combinatorial optimization problems with the following steps: produce an initial set of potential solutions, apply a fitness function to the set in order to determine the best potential solutions, and use these selected solutions to produce the next batch of potential solutions.

Silva et al. [41] present an EA that expresses a potential solution to the k-median problem as a string of numbers where each number is a facility location index. After fitness evaluation new solutions are created in two ways: a one-point crossover where two strings randomly exchange index values from a randomly chosen section of the string, and a mutation operation that randomly changes one index value of a string. They tested their algorithm with the OR-library and report that it found the optimal solution in the majority of the test cases.

Huang et al. [21] use a specific EA called (1 + 1)EA which formulates a solution to the k-median problem as a string of bits where a 0 or 1 value indicates if a location is a client or facility, respectively. These bit strings are evaluated for fitness as described above and new bit strings are produced by randomly flipping the bits of the current best strings. This approach provides a 5 approximation in the case where all distance values are integers. They also propose an improved (1 + 1)EA that provides a $5 + \epsilon$ approximation for any $\epsilon > 0$ in the case where the distance values are real numbers.

Wilder et al. [43] create a machine learning system to solve the closely related k-center

problem. Their system runs in three main steps: a graph representation of a problem is first reduced into a simpler graph, a differentiable solver takes the simplified graph and solves the problem, and the simpler problem solution is used to produce an approximate solution to the original problem. They report that by incorporating both the reduction process and the solver into one machine learning system their system outperforms similar methods that separate the two. Their system also generalizes well to graphs outside of their training set.

Another machine learning approach for the k-median problem and other combinatorial optimization problems is presented by Tayebi et al. [42]. They note that in practice the datasets used in combinatorial optimization problems do not change significantly over time. For example, a logistics company planning vehicle routes would use the same dataset of roads with only the delivery locations changing. Their approach exploits this by training a machine learning model to identify what areas of the solution space have contributed to past optimal solutions. The intuition is that focusing on this smaller solution space will reduce algorithm runtimes without affecting solution quality. Their method first solves several integer linear programs in order to gather training data. This data is then used to train a model that will give each location a numeric value representing how confident the model is that the location could be part of an optimal solution. The search space of the problem is reduced to only include locations with a high confidence value and an integer linear program solver solves the problem using this reduced search space. They report that this reduces runtimes with only a minor degradation in solution quality.

Chapter 3

Hopfield Networks

3.1 Neural Networks

Neural Networks are a type of computing network designed to emulate the natural computing power of the brain. A neuron is a relatively simple processing unit that can be reduced to two main functions: it connects to other neurons in order to give and receive electrical impulses and it outputs an impulse based on the amount of input it receives. However, the aggregate of hundreds or thousands of neurons can solve complex tasks in organisms such as processing vision or reacting to stimulus. The goal is to design a computer neural network that can emulate these biological systems to solve computational problems.

While there are many approaches and variations for constructing artificial neural networks, we can mention a few common elements. The first is the artificial neuron which can be viewed as a function. A neuron will take numerical inputs, it will aggregate them, and an activation function will take this aggregation and produce a final output value. This output will be sent to other neurons for future processing.

The second element is the connections between the neurons. As stated above it is the combination of neurons working together that provides the computing power to solve problems. We connect the neurons by providing connections from the output of one neuron to the input of one or many other neurons. These connections will send numerical values between the neurons. Each connection can be given a weight such that the output placed on a connection can be weakened or strengthened depending on the value of the weight. These weights are adjusted during the so-called training stage to improve the quality of the output of the network. The connections generally send data in one direction from the source neuron to the destination neuron. For bidirectional data flow, two directed connections are usually used.

With these two elements one can construct the actual network structure. A common base structure is the feedforward network. Here the neurons are separated into ordered groups called layers where each layer takes input from the preceding layer and sends output to the subsequent layer. The two exceptions are the first layer which receives input from outside of the network and the last layer which outputs data from the network. The first and last layer are given the names input layer and output layer, respectively and the middle layers are known as hidden layers. Information in feedforward networks move in one direction only: from the input layer, through each hidden layer in sequence, and finally to the output layer from which the output is

read. This flow of information is where the feedforward title gets its name. Figure 3.1 depicts a simple three layer network. There are several variations of neural networks where small cycles or backward connections are allowed such as in recurrent neural networks.

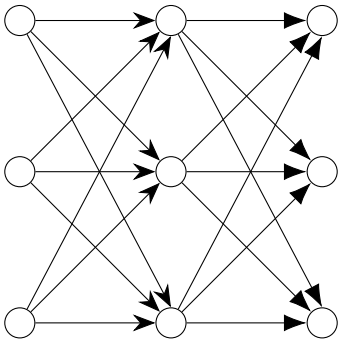


Figure 3.1: An example of a feedforward neural network containing three layers.

With a neural network structure in place one now has to set the weights of the connections and the neuron values. As mentioned, this is known as training the network and two common training methods are supervised learning and unsupervised learning.

Supervised learning utilizes a common training format for tasks that range from labelling images to analyzing sentences. The neural network is given training data which consists of pairs of input data and expected output values. For each piece of training data the neural network processes it and produces an output. This output is compared to the expected output and the difference is recorded as the error. The specific function that determines the error is the *loss function* and the goal is to minimize the loss function over the training data. A process called backpropogation is then performed where working backwards from the output layer the connection weights are adjusted so that future runs on the training data will have a smaller error. This is repeated several times until the overall error rate reaches an acceptably low value.

In unsupervised learning, the neural network trains using only the input data. This training method is used for tasks such as clustering or aggregating elements of a dataset. The neural network will attempt to find patterns in the data and use those patterns to complete the clustering or aggregation tasks. These neural networks can also use backpropagation to update their connection weights by using an error metric specific to their task. For example, a neural network performing distance based clustering can use the distances from datapoints to their nearest cluster center as an error value that backpropogation will attempt to reduce.

3.2 Hopfield Networks

An alternative to the above feedforward network is the Hopfield neural network. While they both share the goal of emulating a biological system with artificial neurons and connections, Hopfield networks differ from other neural networks in their construction and operation.

Instead of building a system of calculations and reading the final values as output, Hopfield networks consist of a network of neurons in which the values of the neurons can be interpreted as a solution to a problem: Each neuron contains an internal state variable with a value between

0 and 1. When the internal state variable is 0, the neuron is considered off and does not send output to other neurons. When the internal state variable is 1, the neuron is considered on and it will send output to other neurons. The set of all these internal state variables is the state of the network and each state can be seen as a potential solution.

For example, the state of the network could be used as a solution for the knapsack problem. Here there are n items each with their own weight and value and the goal is to pack a knapsack with a subset of the items such that the total value is maximized while the total weight is less than or equal to the weight limit of the knapsack. A Hopfield network representation for this problem could use n neurons with each neuron representing one item. The state of the network can be used as a potential solution by selecting all items whose corresponding neuron is on.

The Hopfield network operates by moving between states with the final state being presented as the solution. This movement is driven by an energy function which gives each state a numeric value called energy. The energy function will be unique to each problem and be defined such that the states with low energy will correspond to good or optimal solutions. The Hopfield network will attempt to move between states in order to reduce the current energy of the system. When the Hopfield network has reached a global or local energy minimum it stops moving and the network is said to have stabilized. This stabilized state is the aforementioned final state and is presented as a solution.

Unlike feedforward networks where neurons can be connected only to those neurons in the preceding and succeeding layers, Hopfield neurons can connect to any other neuron in the network. This generally creates a well connected and cyclical network. The weights of these connections are derived from terms in the energy function and will be further explained later. A connection can have a positive or negative weight with positive weights indicating a excitatory connection, which encourages other neurons to turn on, and negative weights indicating an inhibitory connection, which encourages other neurons to turn off.

In terms of operation, the Hopfield Network runs completely asynchronously. Each neuron will periodically aggregate its input and evaluate that aggregate with its activation function. The result of this activation function determines if the neuron will modify its state and what value, if any, the neuron will output. Each neuron will perform this and then wait some arbitrarily selected time interval before repeating the process. This waiting period ensures that other neurons have a chance to modify their state before the current neuron repeats its calculations. As the individual neurons change state, so does the network as a whole and as described earlier the network will gradually move to a stable low energy state. Once the Hopfield network enters a stable state, the individual neurons will still evaluate their state but will not change it.

Finally, there is no learning process in a Hopfield network. As discussed in the previous section, feedforward networks adjust their connection weights in the training phase in order to minimize a loss function. While the Hopfield energy function may appear to be analogous to the loss function, it has a different role in the operation of the network. The energy function is used to derive connection weights that result in the network transitioning to some low energy state. There is no backpropagation and the weights do not change unless the energy function itself changes. Once the energy function is created and the connection weights are set the network is ready to operate.

3.3 Hopfield Networks for Content Addressable Memory

The first application to demonstrate the computational power of Hopfield networks was using a Hopfield network to implement Content Addressable Memory (CAM). Here one wants to be able to store a piece of information in a neural network, and at a later time be able to retrieve it by querying the network with some fragment of the stored information. For example, if one wants to store a contact with the name "John Smith" then the queries "John", "Smith", and "J Smith" should return the contact. Ideally, the network would also be able to handle minor errors in the query such as "Simth".

For a physical system to be able to operate as CAM a few requirements need to be met. The first is that the system must be able to navigate between several states; this allows the system to effectively "search" the state space for the requested information. Second, there must be stable states where the system stops changing and it stabilizes. These will be the states that contain the desired information and one can consider a query finished once the system reaches any stable state. Finally, starting near a stable state should result with the system entering that stable state. To use the above example, "John Smith" should be represented by a stable state and the system should be able to transition from the neighbouring state "Smith" to that stable state.

To begin building the Hopfield neural network, we first define its components and their function. Each neuron i will contain an individual state value V_i such that the value is 0 when not firing (or off) and 1 when firing (or on). The state of the system will then be a vector V containing all V_i values and can be viewed as a binary word.

A symmetrical matrix of real numbers T will represent the connections between neurons with $T_{i,j}$ representing the connection between neuron i and neuron j. A non-zero value for $T_{i,j}$ will indicate that the two neurons are connected. The value that a neuron places on this connection will be multiplied by the connection weight in order to increase or decrease the output value.

The network will function asynchronously with every neuron calculating if it should change or maintain its state. This is determined by a threshold value U_i: Each neuron i will calculate the sum of all its input and will activate if the threshold is met or exceeded, otherwise it will deactivate. The neurons will repeatedly attempt to update their state and once every neuron stabilizes to a state value, the system will have reached a stable state and information can be retrieved.

The above system can be used as CAM. The neurons can take on 0 or 1 values so they can be used to store binary words. The set S will contain all binary words that one wants to store in the system and a specific state vector V^s will correspond to a specific binary word s in S. The individual neurons of the state V^s are denoted by V_i^s. For example, if V^s corresponds to "10" in a two neuron system and the system has entered the state corresponding to that word, then V_1^s will be 1 and V_2^s will be 0.

The system will have connection values set such that each stored state corresponding to a vector V^s is stable and low energy. Hopfield proposes the following formula for defining the connections:

$$T_{i,j} = \begin{cases} \sum_{s \in S} (2V_i^s - 1)(2V_j^s - 1) & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{3.1}$$

	1	2	3
1	0	1	-1
2	1	0	-1
3	-1	-1	0

Table 3.1: Example connection values for a matrix T in a Hopfield network storing the binary word ”110”.

Neuron	Input	Change	Resulting System State
V_1	$(0 \times 1) + (0 \times -1) = 0$	No Change	1, 0, 0
V_2	$(1 \times 1) + (0 \times -1) = 1$	Turn On	1, 1, 0
V_3	$(1 \times -1) + (1 \times -1) = -2$	No Change	1, 1, 0
V_1	$(1 \times 1) + (0 \times -1) = 1$	No Change	1, 1, 0
V_2	$(1 \times 1) + (0 \times -1) = 1$	No Change	1, 1, 0
V_3	$(1 \times -1) + (1 \times -1) = -2$	No Change	1, 1, 0

Table 3.2: The calculations performed by a Hopfield network starting from the state ”100” and stabilizing at the state ”110”.

As an example, suppose we have a three neuron Hopfield network with the binary word ”110” stored. Equation (3.1) is used to generate the connection values and the resulting matrix is shown in Table 3.1. The layout of the network is illustrated in Figure 3.2.

The system can be queried by setting the neuron state values to another state such as ”100” and allowing the network to run. Table 3.2 shows the calculations the system will perform in the case where each neuron is evaluated sequentially and in ascending numerical order. The neurons will use the threshold value $U_i = 0$ as that is the value Hopfield suggests as a default. The first row displays neuron V_1 aggregating the input from the other nodes and the result is 0. As this meets the threshold and the neuron is already on, no change occurs to the neuron’s state. The second row shows neuron V_2 perform its calculations and since the result of 1 is greater than the threshold it turns on. Then V_3 performs the same calculation and stays off as its value of -2 does not meet or exceed the threshold. The neurons then repeat their calculations to confirm that no further changes will occur and therefore the system is in a stable state.

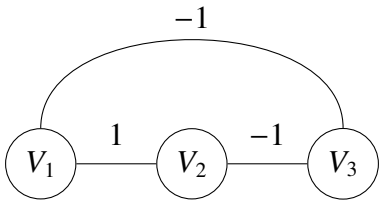


Figure 3.2: A three neuron Hopfield network for content addressable memory with the state ”110” stored.

3.4 Applications in Combinatorial Optimization

Building on the principles of neural networks for CAM, Hopfield and Tank [20] extended the network to solve combinatorial optimization problems. Here, the network is constructed such that the stable states represent possible solutions to a problem. The neurons will function as described above but the actual network construction will be handled differently.

As stated in the overview, the first step is to establish a set of neurons such that their state values can be interpreted as a solution to a problem. A simple example would be having n neurons for the knapsack problem. Here, each neuron would represent an item and its state would indicate if it was selected. Once the system has stabilized, one can select the items that are assigned to the on neurons and present that as a solution.

More complex cases could be structured by larger sets of neurons. For example, we could construct a Hopfield network to solve the graph colouring problem. In this problem one is given a graph and wants to label each node with a colour such that no two adjacent nodes have the same colour. If there are k colours and n nodes, then one could have k groups of n neurons where each state $V_{k,i}$ represents if the node i has the colour k. Like the knapsack solution, the on neurons will represent which colour is chosen for a node.

The next step is to construct the energy function and for combinatorial optimization problems Hopfield and Tank propose the following general equation for a n neuron Hopfield network where neurons are numbered from 1 to n:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}V_iV_j - \sum_{i=1}^{n} V_iI_i \tag{3.2}$$

The neuron state variables V_i and the connection matrix T are defined as they were in the previous section. The variable I_i is the input bias and will add a constant amount of input to a neuron i. Hopfield and Tank claim that the local minima of the equation will correspond to the stable states of the system.

A potential issue with the above structure and energy function is that there is no guarantee that the solution provided by a stable state will be a valid solution i.e., it satisfies the constraints of the problem. In the graph colouring example, the network could return an invalid solution where the node i is labelled with zero colours or more than one colour. The goal in this case would be to compute a valid solution where each node is labelled with exactly one colour and no two adjacent nodes have the same colour.

To address this we introduce terms to the energy function similar to how constraints are added to integer linear programs. These terms are built so that they add little or no energy to the total energy when the state's corresponding solution satisfies the constraints of the combinatorial optimization problem. However, these terms should add a significant amount of energy when the corresponding solution does not satisfy the constraints. In the literature these terms are referred to as *constraint terms* due to their role in ensuring that the produced solutions are valid.

A constant value can also be multiplied to these terms in order to prioritize a constraint or loosen it. The selection of these constant values is crucial as setting them too low leads to invalid solutions but setting them too high leads to valid but potentially poor solutions. For

example, in the knapsack problem there could be a constraint term to ensure that the total weight of the items is at most the weight limit of the knapsack. If this constraint is set too low, the proposed set of items could be far heavier than the knapsack allows. If set too high, only the lightest items may be selected regardless of their value or the knapsack may only be partially filled.

As these constant values are selected externally and passed to the network, they can be seen as tuning parameters. A simple way to select these parameters is to run the network with arbitrarily selected values and if the returned solution is invalid then the values should be increased. If the returned solution is poor then the values should be decreased. This process is then repeated until the produced solutions are both good and valid.

Once the constraint terms are defined they can be used to produce the full energy function as well as the connection and input bias values. Continuing the knapsack problem example, each item i has a weight w_i and the knapsack has a maximum weight limit of W. The constraint that the total weight of the selected items is at most W can be represented with the following constraint term:

$$C_1 \left(\left(\sum_{i=1}^{n} V_i w_i \right) - W \right)^2 \tag{3.3}$$

This term is zero when the total weight of the selected items is the same as the weight limit of the knapsack and increases when the total weight is larger than W. However, if the total weight is less than W then negative energy is introduced and this is addressed by squaring the term to remove negative values. This also encourages a full knapsack as a partially filled knapsack will also increase the total energy. Finally, the constant C_1 is multiplied to the term in order to prioritize or loosen the constraint.

In order to derive the connection values and input biases the constraint terms must be rewritten into the form of Equation (3.2). The following steps are used to rewrite the constraint term:

Expand the squared term:

$$E = C_1 \left(W^2 - 2W \sum_{i=1}^{n} V_i w_i + \left(\sum_{i=1}^{n} V_i w_i \right)^2 \right) \tag{3.4}$$

Replace the squared summation:

$$E = C_1 \left(W^2 - 2W \sum_{i=1}^{n} V_i w_i + \sum_{i=1}^{n} \sum_{j=1}^{n} V_i V_j w_i w_j \right) \tag{3.5}$$

Multiply in the constant:

$$E = C_1 W^2 - 2C_1 W \sum_{i=1}^{n} V_i w_i + C_1 \sum_{i=1}^{n} \sum_{j=1}^{n} V_i V_j w_i w_j \tag{3.6}$$

Multiply by $\frac{-2}{-2}$ in order to get a leading $-\frac{1}{2}$ in the first term; reduce the other $\frac{-2}{-2}$ instances to 1:

$$E = -\frac{1}{2} C_1 \sum_{i=1}^{n} \sum_{j=1}^{n} (-2 V_i V_j w_i w_j) - 2C_1 W \sum_{i=1}^{n} V_i w_i + C_1 W^2 \tag{3.7}$$

Rearrange the variables in order to correspond to the position of the $T_{i,j}$ and I_i variables in Equation (3.2):

$$E = -\frac{1}{2}\sum_{i}^{n}\sum_{j}^{n}(-2C_1 w_i w_j V_i V_j) - \sum_{i}^{n} V_i w_i 2C_1 W + C_1 W^2 \tag{3.8}$$

With the energy function now in the form of Equation (3.2) the values can be derived. The connection values will be defined as:

$$T_{ij} = -2C_1 w_i w_j \tag{3.9}$$

The input bias will be defined as:

$$I_i = 2C_1 W w_i \tag{3.10}$$

The constant term $C_1 W^2$ will be discarded as it adds a constant amount of energy to the equation. Note also that this energy function only ensures that the knapsack is full. A second term would be needed to ensure that the value of the items in the knapsack is maximized.

3.5 Travelling Salesman Problem

To demonstrate the effectiveness of Hopfield networks for combinatorial optimization, Hopfield and Tank created a solver for the Travelling Salesman Problem (TSP) [20]. The problem was chosen as it is a well researched NP-hard combinatorial optimization problem. Here, one is given a set of n locations and the goal is to find the shortest tour that visits all locations once before returning to the starting location.

To begin, several TSP variables are defined. There will be n cities to consider and each city will have a unique label $A, B, C, ...$ The distance between cities A and B will be represented by d_{AB} and the total distance of the tour will have the form $d = d_{AB} + d_{BC} + \cdots + d_{KA}$.

The next step is to define how the neurons can be constructed so that a solution can be decoded from their states. Each city will be represented by n neurons that indicate where the city is on the tour. Of these n neurons, only one should be on and that on neuron indicates when the city is visited. For example, if city A is visited second, and there are three cities then the neurons representing A should have the following state values: $\{0, 1, 0\}$. When constructing the energy function a subscript will be used to indicate which specific city neuron state is being evaluated. Using the information from the above example the neuron states for A would be $V_{A_1} = 0$, $V_{A_2} = 1$, and $V_{A_3} = 0$.

This means that n^2 neurons are used to represent the entire problem and a solution can be visualized in tabular format where the rows are cities and the columns are the order in which they are visited. Table 3.3 contains an example tour.

With the network constructed, the next step is to define an energy function such that the state with the lowest energy value represents the best solution. This will require four constraint terms.

	1	2	3
A	0	1	0
B	1	0	0
C	0	0	1

Table 3.3: An example TSP tour for three cities. Here the tour is B → A → C → B.

The first constraint is that every city should be visited once. Each city satisfies this constraint when one of their neurons has the value 1 and all other neurons have the value 0. In the tabular format this constraint ensures that each row has at most one active neuron. Each city X will determine if this is being satisfied with the summation $\sum_i \sum_{j \neq i} V_{X_i} V_{X_j}$: When one state value is equal to 1 and all others are 0, then this summation will have the value 0. If there are multiple state values equal to 1 then the energy will increase. Note that this summation will also be 0 if all neurons have the value 0. This is addressed by the third constraint that ensures that n neurons are active. Hopfield and Tank also use the constant term C_1 which amplifies how much extra energy is added when the constraint is violated. The full constraint term is then:

$$C_1 \sum_X \sum_i \sum_{j \neq i} V_{X_i} V_{X_j} \tag{3.11}$$

The second constraint is that each tour index should be used once. This is satisfied when only one city neuron X has a value of 1 for a tour index i and all other city neurons Y have the value 0. This constraint can be viewed in the tabular format as a way to ensure each column has at most one active neuron. Like the first constraint, no extra energy will be added if there are no active neurons and the third constraint will address this issue. The summation used will be $\sum_i \sum_X \sum_{X \neq Y} V_{X_i} V_{Y_i}$. This term will also have a constant C_2 and the full term becomes:

$$C_2 \sum_i \sum_X \sum_{X \neq Y} V_{X_i} V_{Y_i} \tag{3.12}$$

The third constraint is that n cities are selected and this is satisfied when there are n neurons with a 1 value and all other neurons have a 0 value. This term is built by summing all state values and then subtracting n. However, this term could introduce negative energy if the sum of the state values is less than n. To address this the term is squared which removes any possible negative numbers. Finally, the constant C_3 is multiplied to the term and the full term is:

$$C_3 \left(\left(\sum_X \sum_i V_{X_i} \right) - n \right)^2 \tag{3.13}$$

The fourth and final constraint is that the total distance between cities on the tour should be minimized. This term is built by considering a city X at index i and all other cities Y at the indices $i - 1$ and $i + 1$. If the city neuron at X_i has the value 1 and so does either Y_{i-1} or Y_{i+1}, then the distance between the two cities is added to the total energy. The summation used is $\sum_X \sum_{Y \neq X} \sum_i d_{XY} V_{X_i} (V_{Y_{i+1}} + V_{Y_{i-1}})$. Like the previous terms, a constant C_4 is used and the full term is:

$$C_4 \sum_X \sum_{Y \neq X} \sum_i d_{XY} V_{X_i} (V_{Y_{i+1}} + V_{Y_{i-1}}) \tag{3.14}$$

Combining these constraint terms gives the full energy function E:

$$\begin{aligned} E = & C_1 \sum_X \sum_i \sum_{j \neq i} V_{X_i} V_{X_j} \\ & + C_2 \sum_i \sum_X \sum_{X \neq Y} V_{X_i} V_{Y_i} \\ & + C_3 \left(\left(\sum_X \sum_i V_{X_i} \right) - n \right)^2 \\ & + C_4 \sum_X \sum_{Y \neq X} \sum_i d_{XY} V_{X_i} (V_{Y_{i+1}} + V_{Y_{i-1}}) \end{aligned} \tag{3.15}$$

With E defined, the connection values and input biases can be derived by rewriting the equation into the general form of Equation (3.2). The variable δ will be used to determine if two indices are the same and will be defined as $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise.

The connection values are defined by:

$$T_{X_i,Y_j} = -C_1 \delta_{XY} (1 - \delta_{ij}) - C_2 \delta_{ij} (1 - \delta_{XY}) - C_3 - C_4 d_{XY} (\delta_{j,i+1} + \delta_{j,i-1}) \tag{3.16}$$

The input bias is:

$$I_{X_i} = C_3 n \tag{3.17}$$

Hopfield and Tank applied the network to several test cases with 10 to 30 cities. They reported that the shortest path was selected 50% of the time in the 10 city tests and that solutions were within 2 times of the optimal answer in the 30 city tests.

3.6 Issues

Despite the promising results reported by Hopfield and Tank, attempts to recreate their experiments ran into issues. Wilson and Pawley [44] evaluated the model and found that even for the small 10 city test case they could not recreate the above results. They note that the network primarily returned invalid solutions and that the few valid solutions were not much better than random.

Other researchers continued to look into the faults reported by Wilson and Pawley and attempted to recreate the successes of Hopfield and Tank. Aiyer et al. [1] examined the state space set up by the network and provided constraint values that always return valid solutions for tests of up to 50 cities. However they do not comment on the quality of solutions in terms of approximation ratios.

Kamgar-Parsi and Kamgar-Parsi [24] also tackled the discrepancy between the papers. In their testing they note that while it is difficult to find valid solutions, the solutions presented are often good or optimal. They do note that the network does not scale well and succeeds less in finding valid solutions as n increases. They posit that while technically functional for TSP, Hopfield networks are better suited for clustering problems. With this in mind they tested the Hopfield network on the k-means problem which performed well for n values up to 128 [24].

3.7 Applications of and Modifications to Hopfield Networks

Several researchers have attempted to apply Hopfield networks to combinatorial optimization problems. In order to address the issues discussed in the previous section they often modify the network to improve the runtime, solution feasibility, and solution quality. Liang [28] extends Hopfield networks by adding "Adjusting Neurons" to the network. In a standard Hopfield network the neurons can be placed into two categories: hypothesis neurons that are used to derive a solution and slack neurons that enforce problem constraints. The new adjusting neurons exist outside of these two categories and are not directly related to the problem being solved. Instead they are connected to existing neurons in order to modify the energy of the system and help the network find better solutions. For example, an adjusting neuron could be connected to the set of neurons for a single problem constraint and add energy to ensure that the constraint is satisfied. Liang tests this new network on several quadratic assignment problems and reports that it provides better solutions than a standard Hopfield network [28].

Chen et al. [10] apply a Hopfield network variation to the vertex cover problem. They note that Hopfield networks will attempt to enter low energy states but these states may not always provide a good solution. To continue exploring the state space and to provide a better solution they introduce a second step after the network has stabilized. This step changes the energy equation such that the energy in the current stable state is raised which causes the network to begin moving to a new and potentially lower energy state. They empirically test this method and report that it provides better solutions than an unmodified Hopfield network and a 2-approximation vertex cover algorithm [10].

Salcedo-Sanz and Yao [38] present a hybrid algorithm that combines Hopfield networks with genetic algorithms for the terminal assignment problem. This problem is similar to the k-median problem except that each client has a capacity value that must be fulfilled by a facility and each facility has a maximum capacity limit. As described in Section 2.3.4, genetic algorithms create a population of potential solutions, evaluate those solutions, and then keep, modify, and propagate the best performing solutions. The process is repeated until a satisfactory solution is found.

The hybrid algorithm uses Hopfield networks to improve the solutions before they are evaluated. The population consists of binary strings where each value indicates if a client is assigned to a facility. These strings can be directly mapped to a Hopfield network where each neuron indicates an assignment from a client to a facility. For each solution X in the population, the binary string values will be used to initialize the state values of a Hopfield network. Once the Hopfield network has stablized the resulting state values will be converted to a solution string X' that replaces X for the next steps of the genetic algorithm. Salcedo-Sanz and Yao report good results on test problems and attribute this to the Hopfield network keeping the potential solutions feasible while the genetic algorithm searches for high quality solutions [38].

Manjunath et al. [30] provide a system for solving NP-complete problems using Hopfield networks. Their system solves a NP-complete problem through two main steps: it generates a set of related NP-complete problems through reduction and then it uses Hopfield networks to solve these new problems. The solutions are applied to the original problem and the best solution is returned. They tested their system on three problems: the travelling salesmen problem, finding the minimum vertex cover of a graph, and finding the maximum cut of a graph. They report that their system performs well for small problem instances and is a promising start for

using reductions and Hopfield networks to solve NP-complete problems [30].

Haralampiev [16, 17, 18] has provided a neural network approach for the k-median problem that derives from both Hopfield networks and Boltzmann machines. In machine learning, Boltzmann machines function similarly to Hopfield networks: they contain neurons that change state based on their connections, they use the energy function in Equation (3.2) and attempt to minimize the total energy, and a solution to a combinatorial optimization problem can be retrieved from the stabilized neuron state values. The main difference is their update method that includes the concept of a temperature T which determines the probability of accepting a neuron update and decreases over time. The update operation for a neuron begins by calculating the difference in energy D if the neuron's state value changes. If $D < 0$ then the neuron changes its state value, otherwise it changes the state value with probablity $(1 + \exp(D/T))^{-1}$.

Haralampiev structures the neural network using $2 \cdot n \cdot k$ neurons and splits them into two $n \cdot k$ groups: Each neuron in the first group represents a client i connecting to a facility j and each neuron in the second group represents a facility j being placed at a location s. These are further divided into subgroups of competing neurons. For instance one subgroup will contain the n neurons that represent client 1 connecting to a facility. These subgroups are used to enforce the problem constraints as the update function will allow only one neuron in a subgroup to be active when the system stabilizes. Neurons in this network have a cost value based on their incoming connections and will activate if their cost value is lower than their competitors. For example, a neuron representing a client connected to a nearby facility will have a lower cost value and activate whereas a neuron with a more distant connection will deactivate. This cost value comparison is used instead of the energy difference for D in the Boltzmann update function.

The full update process starts with a neuron checking if there are any other active neurons in its subgroup and if there are none, it activates. Otherwise, it compares its cost value with the active neurons in the group. If the cost value is lower than the current best value it activates, and if it is higher than the best value it deactivates. Finally, the Boltzmann update described earlier is applied to the neuron which may cause it to change its state. Haralampiev reports that this network performs well and returned the optimal solution in 88% of the performed tests [16].

Chapter 4

K-Median Hopfield Network Formulation

We now describe our implementation of a Hopfield network for computing approximate solutions for the k-median problem. We follow the steps presented in Section 3 and start by creating the set of neurons whose state values will represent a solution. There will be two groups of neurons in our network: the first group will represent if a client i is being served by facility j and the second group will represent if a location j is selected as a facility. These two groups can be viewed as the client-facility variables $y_{i,j}$ and the facility variables F_j from the k-median integer linear program in Section 2.2. Note however that the neurons can take on state values between 0 and 1.

This results in $n^2 + n$ neurons and we visualize these in a tabular format as an $n \times (n + 1)$ neuron matrix. The first column contains neurons that map to the facility variables and the remaining columns contain neurons that map to the the client-facility variables. Table 4.1 contains four figures for a three location k-median problem demonstrating how the neuron matrix corresponds to integer linear program variables, how the neuron state values are arranged, and how to derive a solution from the neuron state values.

Matrix 4.1 (a) shows how the neuron matrix is structured such that each row begins with a facility neuron and how the following neurons in the row represent the potential client-facility connections for that facility.

Matrix 4.1 (b) demonstrates how the neuron state variables V_i are arranged. The neuron state variables V_1 to V_n correspond to the F_j facility variables and the remaining neuron state variables V_{n+1} to V_{n^2+n} correspond to the $y_{i,j}$ client-facility variables.

Matrix 4.1 (c) contains example neuron state values for a stabilized Hopfield network. In this example, the active facility neuron state values are V_1 and V_3 which correspond to F_1 and F_3 in Matrix 4.1 (a). The active client-facility neuron state values are V_4, V_7, and V_{12} which correspond to $y_{1,1}$, $y_{2,1}$, and $y_{3,3}$ in Matrix 4.1 (a).

Table 4.1 (d) demonstrates the solution derived from the values in Matrix 4.1 (c). Facilities 1 and 3 have been selected and serve themselves; client 2 will be served by facility 1.

We now construct the energy function and for the k-median problem we will need five constraint terms. The energy function will use the neuron state variables V_i and to simplify our notation we define a function $V(i, j)$ that maps a client-facility variable $y_{i,j}$ to its corresponding neuron state variable V_h. We define this function as $V(i, j) = V_{n*i+j}$ where the i and j values come from the $y_{i,j}$ variable and n is the number of locations. For example, if we use the variable $y_{2,3}$ and $n = 3$ from Table 4.1 (a) then we get $V(2, 3) = V_{(3*2)+3} = V_9$ which is the corresponding

$$\begin{vmatrix} F_1 & y_{1,1} & y_{2,1} & y_{3,1} \\ F_2 & y_{1,2} & y_{2,2} & y_{3,2} \\ F_3 & y_{1,3} & y_{2,3} & y_{3,3} \end{vmatrix}$$

(a)

$$\begin{vmatrix} V_1 & V_4 & V_7 & V_{10} \\ V_2 & V_5 & V_8 & V_{11} \\ V_3 & V_6 & V_9 & V_{12} \end{vmatrix}$$

(b)

$$\begin{vmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{vmatrix}$$

(c)

Selected Facility	Clients Served
1	1, 2
3	3

(d)

Table 4.1: Matrix representation of the neurons in a three location k-median problem as well as how to derive a solution from the neuron state values. Matrix (a) displays the neurons as the integer linear program variables that they represent, Matrix (b) displays the neuron state values, Matrix (c) contains example values for the neuron state values of Matrix (b), and Table (d) presents the solution derived from Matrix (c).

neuron state variable in Table 4.1 (b). The indices for facility variables F_j range from 1 to n so we define the mapping $F_j \rightarrow V_j$ e.g., $F_2 \rightarrow V_2$.

The first constraint term for the energy function is intended to minimize the total distance between clients and the facilities that serve them. This is represented by evaluating every client-facility neuron and for each active neuron we add the distance between the corresponding client and facility to the total distance. The other constraint terms will restrict the number of activated client-facility neurons which means that this constraint term will provide the smallest amount of energy when client-facility neurons corresponding to shorter distances are activated. As described in Section 3.4, we introduce a constant value C_1 that will increase or decrease the amount of energy the first constraint term produces; each other constraint term will also have its own constant value as explained in Section 3.4. The variable $D_{i,j}$ represents the distance between the client i and the facility j of the client-facility neuron $y_{i,j}$. The first constraint term is:

$$C_1 \sum_{i=1}^{n} \sum_{j=1}^{n} V(i, j) D_{i,j} \tag{4.1}$$

The second constraint term tries to ensure that only k facilities are selected; to represent this we sum the facility neuron state values and subtract k from that sum. Note that if fewer than k facilities are active then this constraint term would add negative energy. We address this by squaring the term so the second constraint term is:

$$C_2 \left(\left(\sum_{i=1}^{n} V_i \right) - k \right)^2 \tag{4.2}$$

The third constraint term is designed to enforce that a client can be served by only one facility. This corresponds to having one active client-facility variable per column of the neuron matrix. For each client its neuron state variables are summed and 1 is subtracted from the sum. Like the previous constraint term, if no client-facility neuron is active then negative energy

would be introduced and we handle this by squaring the term. For a client i this constraint term will be $\left(\left(\sum_{j=1}^{n} V(i,j)\right) - 1\right)^2$. As there are n clients, we add a second summation to the constraint term which becomes:

$$C_3 \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{n} V(i,j)\right) - 1\right)^2 \tag{4.3}$$

The fourth constraint term tries to ensure that exactly n client-facility neurons are active. Note that this constraint term does not guarantee that each client would be connected to an active facility; that is handled by the fifth constraint term (4.5). The neuron state variables for the client-facility neurons are summed and n is subtracted from the sum. If there are fewer than n active neurons negative energy would be introduced so we square the term to handle this:

$$C_4 \left(\left(\sum_{i=1}^{n} \sum_{j=1}^{n} V(i,j)\right) - n\right)^2 \tag{4.4}$$

The final constraint term is intended to enforce that a client-facility neuron is active only if the corresponding facility neuron is active. In the neuron matrix this means that a row can only have active neurons if the first entry of the row corresponds to a facility neuron that is active. This term only adds energy in the case where a facility neuron is inactive and its client-facility neurons are active. We determine this by subtracting the facility neuron state value from 1 and multiplying the result with the client-facility neuron state values. If the facility neuron is active then the result is 0 and therefore no energy will result from the multiplication. If the facility neuron is inactive then the result is 1 and energy will be added only if some of its client-facility neurons are active. We check every facility and client-facility pair and the constraint term becomes:

$$C_5 \sum_{i=1}^{n} \sum_{j=1}^{n} (1 - V_i) V(j,i) \tag{4.5}$$

Combining the above constraint terms gives the full energy equation:

$$
\begin{aligned}
E = {} & C_1 \sum_{i=1}^{n} \sum_{j=1}^{n} V(i,j) D_{i,j} \\
& + C_2 \left(\left(\sum_{i=1}^{n} V_i \right) - k \right)^2 \\
& + C_3 \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{n} V(i,j) \right) - 1 \right)^2 \\
& + C_4 \left(\left(\sum_{i=1}^{n} \sum_{j=1}^{n} V(i,j) \right) - n \right)^2 \\
& + C_5 \sum_{i=1}^{n} \sum_{j=1}^{n} (1 - V_i) V(j,i)
\end{aligned} \tag{4.6}
$$

We then rearrange the energy equation into the form of Equation (3.2). The steps can be found in Appendix A. From this rearranged form we can derive the connection values T and input bias values I. Note that unlike the examples in Section 3, the constraint terms are defined for subsets of the neurons. For example the k facility constraint term (4.2) applies only to the facility neurons numbered 1 to n and the n client-facility constraint term (4.4) applies only to the client-facility neurons numbered $n+1$ to n^2+n. This means that the connection matrix T will be defined by a piecewise function that uses the indices of neurons i and j to determine their connection. As described above a neuron i is a facility neuron if $1 \leq i \leq n$ or a client-facility neuron if $(n+1) \leq i \leq n^2 + n$.

The connection values are defined as:

$$
T_{i,j} = \begin{cases}
-2C_2 & \text{if } i \text{ and } j \text{ are facility neurons} \\
2C_5 & \text{if } i \text{ is a facility neuron and } j \text{ is a client-facility neuron served by } i \\
-2C_3 - 2C_4 & \text{if } i \text{ and } j \text{ are client-facility neurons in the same column} \\
-2C_4 & \text{if } i \text{ and } j \text{ are client-facility neurons in different columns} \\
0 & \text{otherwise}
\end{cases} \tag{4.7}
$$

The input bias values are defined as:

$$
I_i = \begin{cases}
2C_2 k & \text{if } i \text{ is a facility neuron} \\
-C_1 D(i) + 2C_3 + 2C_4 n - C_5 & \text{if } i \text{ is a client-facility neuron}
\end{cases} \tag{4.8}
$$

Where $D(i)$ is a distance function that takes a client-facility neuron i and returns the distance between the represented client and facility. We need this function because our previous distance variable $D_{j,k}$ requires two locations j and k whereas the input bias function operates using a neuron i. This function allows us to map a neuron i to a distance $D_{j,k}$ and is defined as $D(i) = D_{\lfloor (i-1)/n \rfloor, ((i-1) \bmod n)+1}$. For example, neuron 9 in Table 4.1 (a) corresponds to the client-facility variable $y_{2,3}$ in Table 4.1 (b). Using $D(i)$ with $i = 9$ and $n = 3$ produces $D(9) = D_{\lfloor (9-1)/3 \rfloor, ((9-1) \bmod 3)+1} = D_{2,3}$.

The performance of the above network is underwhelming and we experience the same issues as described in Section 3.6: The network takes a significant amount of time to stabilize and the solutions returned are often poor or invalid. We speculate that constraint value selection plays a large role in these issues as it is a known problem in the literature and our network requires selecting five values. We do not report test results for this network due to the poor performance.

Chapter 5

The Modified Hopfield Network

We now describe how to create a modified Hopfield network that addresses the issues of the Hopfield network described in the preceding section. Our new modified Hopfield network will reuse parts of the previous Hopfield network's structure but differ in construction and operation.

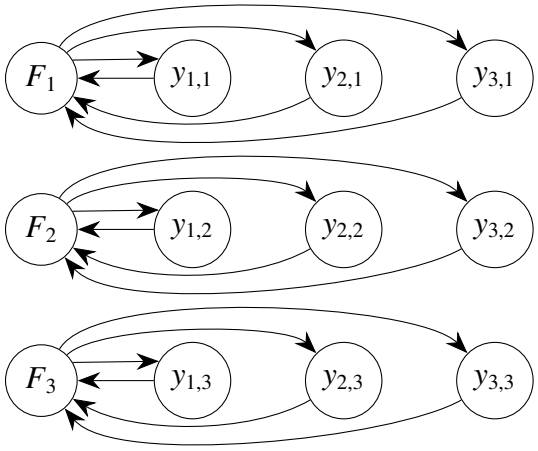
We use the same neuron groups as in Section 4 and we arrange and visualize the neurons as in Table 4.1. Each neuron will also have a state variable which can take a value between 0 and 1. This state variable will determine if the neuron is on or off and going forward we will refer to it as the neuron's activation value.

Our first modification will be adding a second variable to each neuron that stores the sum of all positive input into the neuron. As described in Section 3.2 a positive connection will encourage a neuron to activate and is referred to as an excitatory connection. The excitatory connections in our network will be determined by the distance between locations, with smaller distances producing larger excitatory connection values. As a result the sum of the positive input will be referred to as the neuron's inner value and will determine how valuable a neuron is to a potential solution. Ideally, the active neurons will have a large inner value as that would mean they are connected to neurons that represent nearby locations.

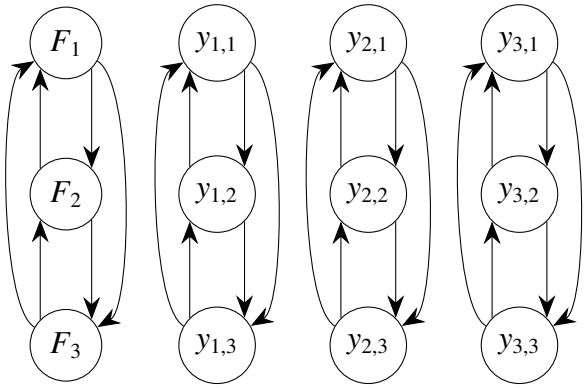
The next step for a Hopfield network would be to generate an energy function from the constraint terms. Our implementation does not do this and we instead incorporate the constraint terms (4.1) - (4.5) into the neuron update operations. For example, instead of a facility neuron activating because its input crosses a threshold, it activates only if its inner value is one of the k largest facility inner values. Moving the constraint terms into the network itself makes energy an implicit part of the system; the network will still move between states before stabilizing onto what would be a stable state in a traditional Hopfield network. This removes the difficult problem of selecting values for the constraint term constants that result in both valid and good solutions. The update operations will be explained further in Section 5.1.

The connection values are no longer derived from the energy function. Instead we create a connection scheme specifically for our representation of the k-median problem. Each facility neuron will have an excitatory connection to all the client-facility neurons that it can potentially serve and an inhibitory connection to all other facility neurons. Likewise, each client-facility neuron will have an excitatory connection to the potential serving facility neuron and an inhibitory connection to all competing client-facility neurons.

Figure 5.1 demonstrates the connections for a three location k-median problem where Figure 5.1 (a) contains the excitatory connections and Figure 5.1 (b) contains the inhibitory con-



(a) Excitatory Connections



(b) Inhibitory Connections

Figure 5.1: The neuron connections for a three location k-median problem. Figure (a) displays the excitatory connections and Figure (b) displays the inhibitory connections.

nections. Using neuron F_1 as a facility neuron example, we see it has an excitatory connection to the neurons $y_{1,1}$, $y_{2,1}$, and $y_{3,1}$ in Figure 5.1 (a) and an inhibitory connection to all other facility neurons in 5.1 (b). Likewise, the neuron $y_{2,1}$ demonstrates the client-facility neuron connections where it has a single excitatory connection to the potential serving facility neuron F_1 and inhibitory connections to the competing client-facility neurons $y_{2,2}$ and $y_{2,3}$.

The excitatory connections will be based on the distances between locations: As discussed earlier each neuron's inner value will be the sum of all its excitatory input so we will modify the distances such that short distances produce large excitatory values and long distances produce low excitatory values. We use a $n \times n$ matrix D to store the $D_{i,j}$ values and perform two steps to get a modified matrix D'. The first step is to use min-max normalization in order to scale the distances to the range [0, 1]: Given the maximum value, $max(D)$, and minimum value, $min(D)$, of D we apply the following formula to a value $D_{i,j}$ in order to get the normalized form:

$$normalized(D_{i,j}) = \frac{D_{i,j} - min(D)}{max(D) - min(D)}$$

	1	2	3	4	5
1	0	0.14	0.63	0.64	0.67
2	0.14	0	0.58	0.67	0.73
3	0.63	0.58	0	0.36	0.50
4	0.64	0.67	0.36	0	0.14
5	0.67	0.73	0.50	0.14	0

(a) D

	1	2	3	4	5
1	0	0.19	0.86	0.88	0.92
2	0.19	0	0.79	0.92	1
3	0.86	0.79	0	0.49	0.68
4	0.88	0.92	0.49	0	0.19
5	0.92	1	0.68	0.19	0

(b) $normalized(D)$

	1	2	3	4	5
1	1	0.81	0.14	0.12	0.08
2	0.81	1	0.21	0.08	0
3	0.14	0.21	1	0.51	0.32
4	0.12	0.08	0.51	1	0.81
5	0.08	0	0.32	0.81	1

(c) D'

Table 5.1: An example of converting a distance matrix D into the modified distance matrix D'. Matrix (a) contains the distance data from Table 2.1, Matrix (b) is the result of normalizing all values to the range [0, 1], and Matrix (c) shows the final result of subtracting each value from 1.

The second step is to take the normalized value and subtract it from 1 i.e., $D'_{i,j} = 1 - normalized(D_{i,j})$. These two operations are applied to every element of D to produce the modified matrix D'. Table 5.1 contains an example of modifying the distances using data from Table 2.1.

The neurons will use the values in D' for their excitatory connections. In our Hopfield network the only excitatory connections will be between a facility neuron and a client-facility neuron that can potentially be served by that facility: Formally, the connection from a facility neuron F_j and a client-facility neuron $y_{i,j}$ will have the value $D'_{i,j}$. Note that we are not using bidirectional connections; there will be a connection from F_j to $y_{i,j}$ and a second connection from $y_{i,j}$ to F_j. The excitatory output that a neuron produces is the connection value multiplied by the neuron's activation value. For example, if facility neuron F_j is partially activated with an activation value of 0.7 it will output $0.7 \times D'_{i,j}$ to the connected client-facility neuron $y_{i,j}$.

This scheme of using the modified distance matrix D' for excitatory connections has several benefits. The first is that nearby neurons will encourage each other to activate due to nearby locations sending large excitatory output to each other. This can cause a positive feedback loop as a facility near many potential clients will encourage those neurons to activate which will then further encourage the facility neuron to activate. For client-facility neurons, their inner value will be based entirely on the distance to their serving facility which means that it is simple to determine for a client i which of the client-facility neurons $y_{i,j}$ should activate further by selecting the neuron with the largest inner value. Having the client-facility neuron with the largest inner value activate will also result in the client-facility neurons being assigned to the nearest facility when the network stabilizes.

The inhibitory connections will be used to discourage competing neurons from activating.

Our connection scheme will assign a value of 1 to every inhibitory connection and an inhibitory output is produced by multiplying a neuron's inner value with the inhibitory connection value. For example, if a facility neuron F_j has the inner value 2.5 it will output 2.5 to all other facility neurons.

At a high level this is how neurons can compare themselves to their competition. As described earlier, all facility neurons will have an inhibitory connection to all other facility neurons and for each client, the corresponding client-facility neurons will have an inhibitory connection to each other. The neurons will all attempt to activate and giving a neuron access to the inner value of its competitors allows us a simple way to determine which neuron has the largest inner value. For example, the client-facility neurons for a client 2 in a three location k-median problem may have the following inner values: $y_{2,1} = 0.5$, $y_{2,2} = 1$, and $y_{2,3} = 0.3$. In this case all three neurons can see that $y_{2,2}$ has the best inner value which means $y_{2,2}$ should be activated further and the other neurons should be deactivated.

To be consistent with the previous Hopfield network notation we can summarize the connection values between neurons i and j into a connection matrix T as:

$$T_{i,j} = \begin{cases} D'_{i,j} & \text{if } i \text{ to } j \text{ is an excitatory connection} \\ 1 & \text{if } i \text{ to } j \text{ is an inhibitory connection} \\ 0 & \text{if } i \text{ is not connected to } j \end{cases} \tag{5.1}$$

Note also that we do not add an input bias I to our neurons.

5.1 Modified Hopfield Network Update Operations

As discussed previously we modify the Hopfield network update operation to include the problem constraints. Our implementation will use two update operations: one for facility neurons and a second for client-facility neurons.

The facility update operation checks if the current facility's inner value is one of the k largest facility inner values. If so the facility neuron is activated by setting its activation value to 1 and if not the neuron is deactivated by setting its activation value to 0. This operation ensures that only k facilities are selected and it contributes to the overall goal of reducing the total distance by preferring facilities that are serving nearby clients. Algorithm 1 contains the simplified steps of this operation while Algorithm 4 contains the pseudocode our implementation uses.

Algorithm 1 Simple Facility Update Operation

Input: Facility f
  Sort the facility inner values in decreasing order
  if f has a larger inner value than any of the first k facilities **then**
    Set the activation value of f to 1
  else
    Set the activation value of f to 0
  end if

The client-facility update operation also modifies the neurons based on their inner values. The main difference is that the client-facility update operation updates the neurons as a batch instead of individually. For each client location, the client-facility neuron with the maximum inner value will be selected and its activation value will set to 1 while the other neurons will have their activation value set to 0. Grouping the client-neuron updates together increases performance by reducing the number of individual update operations and it allows us to use efficient GPU matrix operations in our implementation. There also does not appear to be a decrease in solution quality when performing batch updates instead of individual updates. Algorithm 2 contains the simplified steps of the operation while Algorithm 5 contains the pseudocode our implementation uses.

Algorithm 2 Simple Client Update Operation

```
for Each client location i do
    Get the client-facility neuron y_{i,j} with the largest inner value
    Set the activation value of y_{i,j} to 1
    for Each facility location h that is not j do
        Set the activation value of y_{i,h} to 0
    end for
end for
```

The client-facility update operation ensures that several problem constraints are satisfied: each client will have exactly one active client-facility neuron so each client will be served by only one facility, the client-facility neuron with the largest inner value will be activated which means that each client will be connected to the nearest active facility, and activating client-facility neurons based on how close they are to an active facility also reduces the total distance between facilities and their clients.

5.2 Modified Hopfield Network Algorithm

Our implementation uses a sequential control flow that alternates facility neuron and client-facility neuron updates. We start by initializing matrices that contain the activation values, inner values, and modified distance values. We define both the activation value matrix and the inner value matrix as $n \times (n+1)$ matrices whose elements correspond to the variables in matrix 4.1 (a) i.e., the first column contains the facility neurons and the remaining columns contain client-facility neurons. Our modified distance matrix D' is a $n \times n$ matrix where each element (i, j) corresponds to the modified distance between locations i and j. In our pseudocode we refer to these matrices as $activationValues$, $innerValues$, and $distances$, respectively.

Our algorithm then enters the *runFunction* which will call the update functions *updateFacility* and *updateClient* until the network stabilizes and a solution is returned. This section will explain these three functions in depth.

The main program flow is handled by the *runFunction*. This function coordinates the update operations and proceeds until the neurons have stabilized. It consists of a while loop that continues until a stabilization variable is set to true. In each iteration the function will

operate differently depending on if it is updating the facility neurons (hereon called facilities) or the client-facility neurons (hereon called clients).

A facility update will start by arbitrarily selecting a facility and updating its activation value based on its competitors. If there is no change (the facility stayed on or off) then the next update will also be a facility update as there has been no change to the client inner values. If there was a change the network is checked for stabilization, the client inner values are updated, and the next update will be a client update.

A client update will start by increasing or decreasing the activation values of all clients. If there was a change to any client value then the facility inner values are updated. The next update will always be a facility update. This reduces unneeded computations as clients will stabilize much sooner than facilities as well as giving facilities more opportunities to change their activation values before the network begins stabilizing.

Algorithm 3 contains the pseudocode for the *runFunction*. The variables *stabilized*, $updateType$, and *hasChanged* will direct the flow of operations. *stabilized* controls the while loop and initializes to false. It is set to true when k facilities and n clients are active and in a valid configuration i.e., all clients are connected to their nearest active facility. $updateType$ determines the update operation and is set to one of two constant values: $FACILITY$ or $CLIENT$. Finally, *hasChanged* is the return value of the update operations and specifies if an activation value has been changed.

The updateFacility function operates as described in Algorithm 1 but contains extra steps for the actual implementation. It starts by caching the current activation value of the facility from the activation values matrix. This will be used in the return value to determine if it has changed. We then sort the inner values matrix to get the top k facility inner values and the indices for the corresponding facilities. The if statement is based on either the facility having a better value than any of the top k facilities or the facility already being in the top k facility indices. The activation value is set to 1 if the statement is true and set to 0 otherwise. The update operation then returns a boolean that indicates if the activation value has changed. Algorithm 4 contains the pseudocode.

Note that we will use python slice notation for selecting sections of the matrices. The character ":" indicates that all elements should be selected. If a value is placed before it such as "5:" then we select all elements starting from element 5 onwards. If a value is placed after it such as ":7" then we select all elements from the first element up to but not including element 7. For matrices we can apply the above operations as follows on some arbitrary $x \times x$ matrix A; $A[:, :]$ returns all rows and columns, $A[: 4, :]$ returns the first four rows and all columns, $A[:, 7 :]$ returns all rows and columns 7 through x, $A[: 3, 9]$ returns the first three rows and column 9.

Like the $updateFacility$ function, the $updateClient$ function contains a simple description in Algorithm 2 and includes extra steps in the implementation. This function starts by caching the current activation values of all clients in order to test for any changes at the end of the function. Then for each client column (columns 1 through $n + 1$) the function gets the index of the client with the largest inner value. The client with the largest inner value will have its activation value set to 1 and all other clients in the same column will have their activation values set to 0. Finally a boolean is returned that indicates if the activation values have changed. Algorithm 5 contains the pseudocode.

Algorithm 3 runFunction

```
stabilized ← false
updateType ← FACILITY
while stabilized is false do
    if updateType is FACILITY then
        # selectFacility is an arbitrary facility selection function
        facility ← selectFacility()
        hasChanged ← updateFacility(facility)
        if hasChanged is true then
            # Update the client inner values
            updateClientValues()
            stabilized ← isStabilized()
            updateType ← CLIENT
        else
            updateType ← FACILITY
        end if
    else
        hasChanged ← updateClients()
        if hasChanged is true then
            # Update the facility inner values
            updateFacilityValues()
        end if
        updateType ← FACILITY
    end if
end while
```

Algorithm 4 updateFacility

```
Input: facility, activationValues, innerValues
# The first column contains the facilities
oldValue ← activationValues[facility, 0]
facilityValue ← innerValues[facility, 0]

# sort will return the largest k inner values and the corresponding indices
values, indices ← sort(innerValues[:, 0], k)

if facilityValue > values or facility in indices then
    facilityValue ← 1
else
    facilityValue ← 0
end if

return facilityValue ≠ OldValue
```

Algorithm 5 updateClient

Input: $activationValues$, $innerValues$
  # Clients are in columns 1 to $(n + 1)$
  $oldValues \leftarrow activationValues[:, 1:]$

  # $argmax()$ returns the maximum index for each column
  $indices \leftarrow argmax(innerValues[:, 1:])$

  $activationValues[indices, 1:] \leftarrow 1$
  # We use $\neg indices$ as a shorthand to indicate all non-max value indices
  $activationValues[\neg indices, 1:] \leftarrow 0$

  return $activationValues[:, 1:] \neq oldValues$

5.3 Extensions to the Algorithm

Our modified Hopfield network will also contain two reinforcement learning based extensions to improve performance and solution quality. The first extension will reduce the number of facility update calls by selecting facilities based on whether they have stabilized or not. The second extension focuses on solution quality by restricting the state space to areas that have shown potential to have good solution values.

5.3.1 Reinforcement Learning Facility Selection

Our first optimization uses a variation of the reinforcement learning bandit algorithm for our *selectFacility* function in Algorithm 3. The bandit algorithm is given several possible actions with some reward distribution for taking an action and its goal is to maximize the total reward over time. The name comes from the slot machine slang term "one armed bandit" and a simple example is given n slot machines, which machine or machines should be played the most in order to maximize the total winnings over time? This type of algorithm balances exploitation (repeatedly performing known high reward actions) with exploration (trying new and possibly suboptimal actions to find better actions) in order to maximize the reward.

We use this type of algorithm in our facility selection function to avoid repeatedly evaluating facility neurons that have stabilized. The action our bandit algorithm performs is selecting a facility to update. Internally, it contains two sets: an active set of facilities with the potential to change their activation values and an inactive set containing facilities who have been updated but did not change. The bandit algorithm will randomly select a facility from the active set and is given a reward of 1 if it changes and 0 if it does not. Facilities with a reward of 1 stay in the active set otherwise they are moved to the inactive set.

To encourage exploration and to accommodate facility neurons that may change after reaching an activation value of 0 or 1, we allow the algorithm to occasionally select a facility from the inactive set. To do this we introduce a probability variable ϵ where $0 \leq \epsilon \leq 1$ and have the algorithm select a random number between 0 and 1 when selecting a facility; if the random number is less than ϵ, then the algorithm will select from the inactive set, and if it is equal to

or larger than ϵ, it selects from the active set. If a facility changes after being selected from the inactive set it will be moved to the active set.

The use of a bandit algorithm reduces the number of redundant facility updates when compared to simpler methods of sequentially iterating over all facilities or randomly selecting facilities. However, it does introduce another hyperparameter ϵ which requires more testing in order to find which values of ϵ best reduce the number of facility updates. We use $\epsilon = 0.05$ as we find in practice that stabilized facility neurons tend to stay stabilized. Other reinforcement learning improvements could also be used such as starting with a high ϵ to encourage exploration and reducing it over time to focus on reward exploitation.

5.3.2 State Space Exploration

The second optimization is adding functionality to our modified Hopfield network to search the state space for better solutions. Our current implementation evaluates all neurons and produces a solution when the network stabilizes. Due to the randomness in activation value initialization and facility update selection we can attempt to get a better solution by reinitializing the network values and allowing the network to stabilize again. We will refer to each iteration of initialization and stabilization as a run of the network and we can try to find a better solution by performing several runs and returning the best solution. Building on this we attempt to improve the solution quality by performing multiple runs where the network only evaluates a subset of facilities.

We call this variation the searching Hopfield network. At a high level the searching Hopfield Network keeps track of previously found solutions and uses that data to narrow down the search space over time. Given a set of k facilities from a previous solution the network will only consider those facilities and nearby locations in the next run. If the solution improves then the new solution's facility set and a smaller set of nearby locations is used for the next run. Otherwise, the network will try a previous solution with a different set of nearby locations.

This search process can be visualized as a search tree where each node determines what area of the state space should be considered during a run. When a node is selected the network uses its state space values and produces a new solution. This new solution will be used to create a node and it will be attached as a child to the original node. A node can be evaluated several times and have multiple children. The search aspect comes from selecting a node with a promising solution and using it to direct future runs. This process will continue until a certain number of runs have been completed and the best solution found will be returned.

The nodes of the search tree will contain three values: a set of k facilities, a search radius value r, and a node score. The facility set will be the solution that was used to create the node and comes from a run that used the parent node's information. These facilities will be used alongside the search radius to reduce the search space.

The search radius r is an integer value that determines how many nearby locations should be included alongside the facility set. For each facility we find the r closest locations and include them in our search. The value of r will decrease the further down the tree the node is in order to narrow down the search space. We use a simple formula for this where given the distance between a node i and the root node $dist(i, root)$ and the total number of locations n, $r = (n/k)/(dist(i, root) + 1)$.

For our test results in Section 6 we make two modifications to our search radius values. The first is adding a randomly selected small integer between 1 and 5 to r. This slightly varies the locations that will be evaluated by the Hopfield network. The second is adding a probability to exclude the original facility set from the search. In this case, the Hopfield network will construct a set of potential facility locations using the node's facility set but will not include the facility set in its search. This prevents the network from continually returning the same solution and forces it to search different areas of the state space. We set the probability of excluding the starting facilities to 0.5.

The node score represents how good of a search candidate a node is. The score is initialized to the solution value of the node's facility set S i.e., the sum of distances between all clients and their nearest facility in S. The score is a rolling average of the solution value of S and all solution values derived directly from this node: Every time a node is evaluated a facility set is produced and the value of this solution is added to the rolling average. The intent is that if on average a node produces a good solution then the area of the state space it evaluates should be further explored.

The root node of the search tree is a special case in that it always evaluates all n locations. The node score will be initialized to the distance value of the first returned solution and will be updated in the same way as the other nodes. Figure 5.2 contains an illustration of a four node search tree.

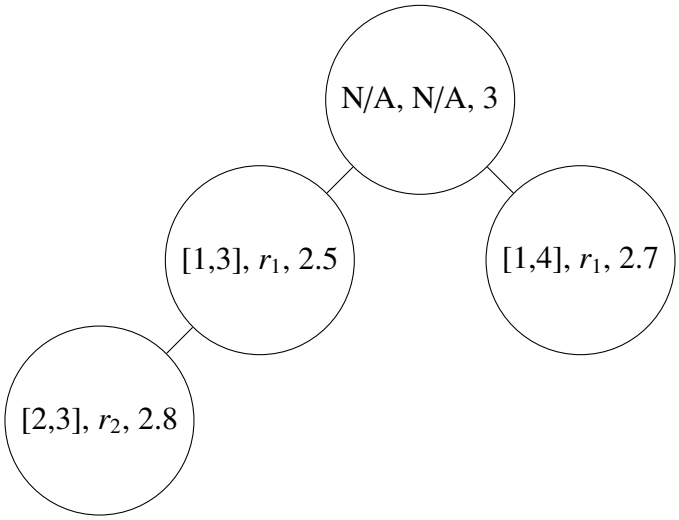


Figure 5.2: A searching Hopfield Network search tree. Each node stores a tuple containing the k facility solution set, a search radius value r determining how many nearby locations should be evaluated, and the node score. Note that the root node will always evaluate the entire state space.

Chapter 6

Experimental Results

We evaluated our modified Hopfield network using five datasets. Each dataset contains test problems for various n and k values. Three of the datasets have been created by us and we used an integer linear program solver to find the optimal solutions. The remaining two datasets are publicly available and come with precalculated optimal solution values. The datasets are further explained in Section 6.3.

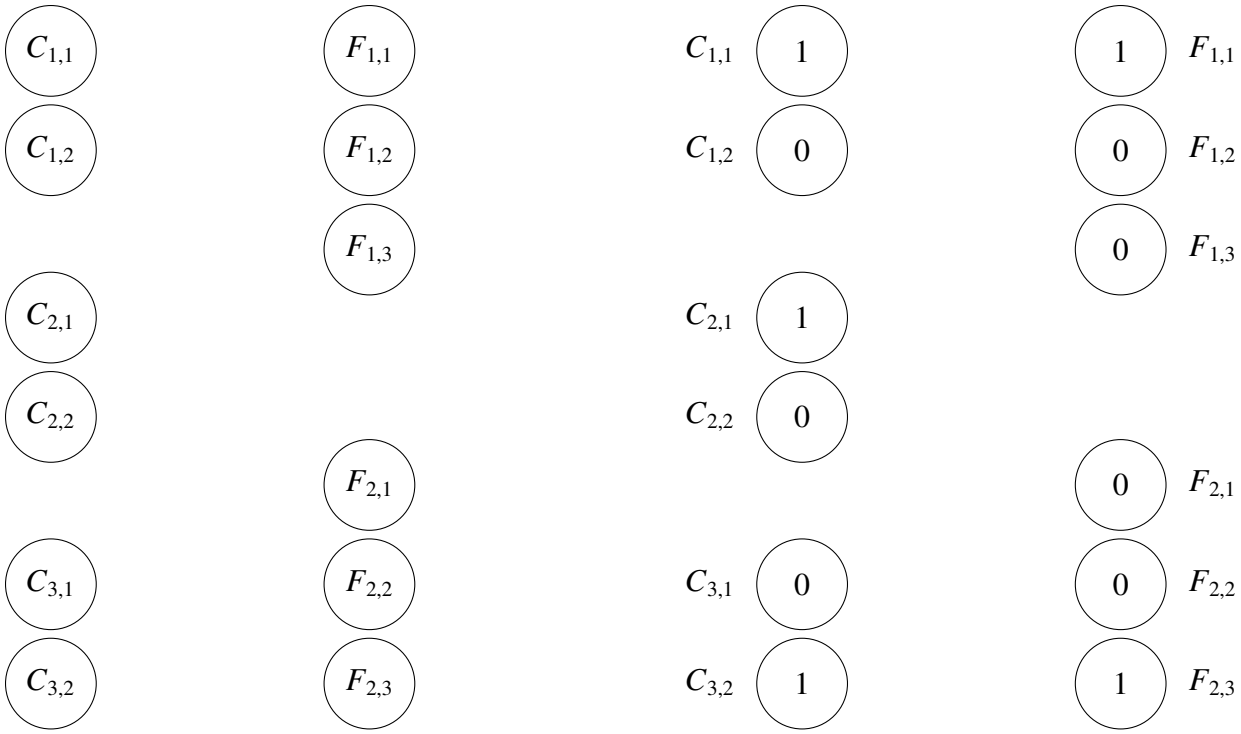
We focus on two statistics for our modified Hopfield network: the approximation ratio of the returned solution and the total time taken to create the solution. Through these statistics we show that our approach can produce competitive solutions in a relatively short amount of time. To demonstrate that our approach is competitive we report the same statistics for Haralampiev's competition-based neural network [16, 17, 18] and Arya et al.'s local search algorithm [2]. Haralampiev's approach was chosen as it is based on Hopfield networks and is used for solving the k-median problem. The local search algorithm was chosen as it is considered one of the simplest and most effective algorithms for the k-median problem [13, 35].

6.1 Haralampiev's Competition-Based Neural Network

6.1.1 Overview

We introduced Haralampiev's competition-based neural network [16, 17, 18] in Section 3.7. As in Hopfield networks the neurons contain state values and are arranged such that a solution can be derived from these state values. Haralampiev's network contains $2 \cdot n \cdot k$ neurons split into two groups of size $n \cdot k$. The first group C determines the assignment of clients to facilities where each neuron indicates if a client in location i is being served by a facility j. The second group F is responsible for placing facilities in locations and each neuron indicates if a facility j is placed in a location s. The individual neurons in C will be referenced by their location and serving facility e.g., the neuron representing a client in location 1 being served by facility 3 is $C_{1,3}$. Likewise, the individual neurons in F will be referenced by their facility number and location e.g., the neuron for facility 2 being placed in location 4 is $F_{2,4}$.

C and F will be further split into subgroups of competing neurons. These subgroups will be used to ensure that the solution produced is valid as the neuron update process will result in one neuron per subgroup activating. C will be split into n subgroups of size k where each



(a) Neuron Subgroups (b) Example Neuron State Values

Figure 6.1: The neuron layout for Haralampiev's network where $n = 3$ and $k = 2$. Figure (a) displays the subgroups of the neurons and Figure (b) displays an example solution using neuron state values.

subgroup is responsible for the assignment of one client to a facility. For example, if $n = 3$ and $k = 2$ then the client in location 1 can be served by facility 1 or facility 2 which produces the subgroup $\{C_{1,1}, C_{1,2}\}$. F will be split into k subgroups of size n where each subgroup determines where the facility is placed. Using $n = 3$ and $k = 2$, facility 2 can be placed at locations 1, 2, or 3. The subgroup for this facility is $\{F_{2,1}, F_{2,2}, F_{2,3}\}$.

Figure 6.1 contains all subgroups for a network where $n = 3$ and $k = 2$ as well as an example of a valid solution. In Figure 6.1a the client neurons are in groups of two. Only one of the neurons in each group will be active and that will indicate if the client is served by facility 1 or facility 2. The facility neurons are in groups of three and indicate if a facility is placed at location 1, 2, or 3. Figure 6.1b contains the neuron state values for an example solution. Looking at the client groups we see that client 1 is being served by facility 1, client 2 by facility 1, and client 3 by facility 2. For the facilities we see that facility 1 is placed at location 1 and facility 2 is placed at location 3. The returned solution is then $\{1, 3\}$.

However, when running our tests we encountered issues with the above facility subgroup definition producing invalid solutions. While the facility subgroups ensure that one facility per subgroup is active, they do not prevent facilities in different subgroups from being placed in the same location. Using the $n = 3$ and $k = 2$ example, the network could have activated both $F_{1,1}$ and $F_{2,1}$ which would return $\{1\}$ as the solution. This violates the constraint that k facilities are selected.

To address this we modify the subgroup for each facility neuron and to explain how this fixes the problem we briefly go over the neuron state value update process. Again, as in Hopfield networks each neuron may connect to several other neurons and when a neuron evaluates its state value it begins by summing the input received from other neurons connected to it.

This is called the neuron's *cost value* and this cost value will be compared with the cost values of the neurons in its subgroup. If a neuron has the smallest cost value in its subgroup it changes its state value to 1, otherwise it changes its state value to 0.

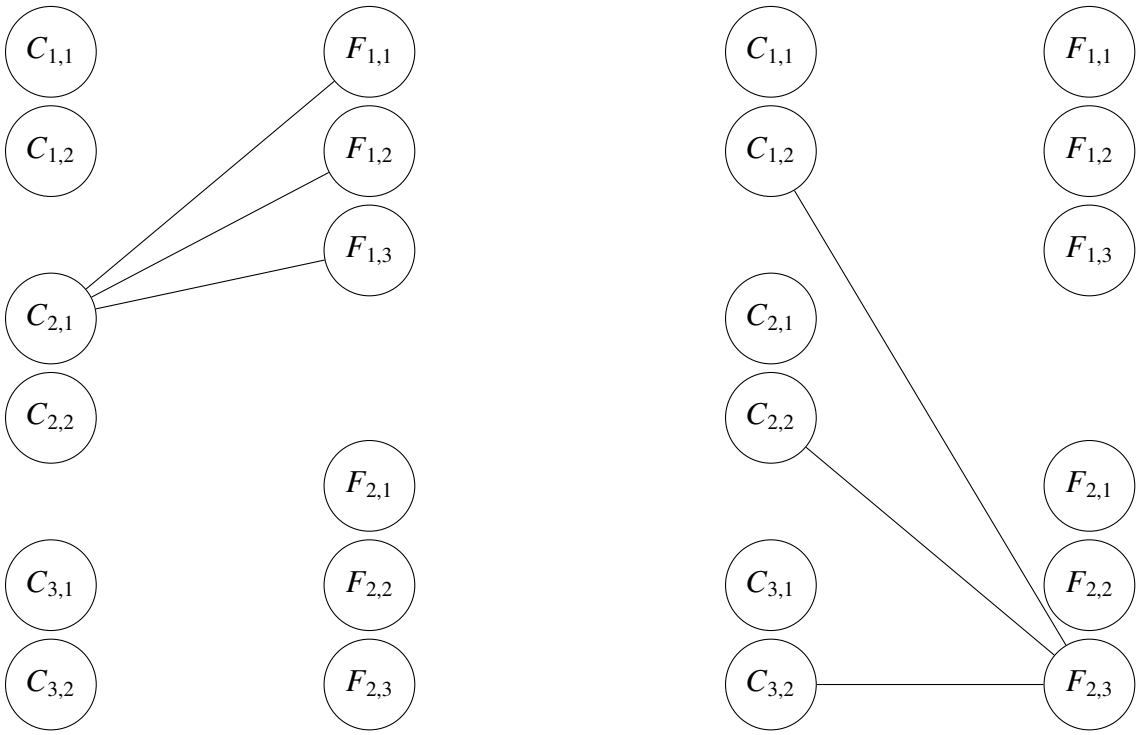
When a facility neuron is selected for an update it will compare its cost value with those of the facility neurons in its facility subgroup. For example, the neuron $F_{1,1}$ will compare its cost value with the cost values of $F_{1,2}$ and $F_{1,3}$. As stated above this process of only comparing neurons in a facility subgroup can result in invalid solutions. Our fix involves having each facility neuron compare itself with the members of its facility subgroup as well as any other neurons that represent a facility being placed in the same location. Since the neuron $F_{1,1}$ represents placing facility 1 on location 1 it will also compare its cost value with that of $F_{2,1}$ which represents placing facility 2 on location 1. Thus the full group that $F_{1,1}$ competes with is $F_{1,2}$, $F_{1,3}$, and $F_{2,1}$. This ensures that only one facility is placed on a location as a facility neuron will only activate if it has both a better cost value than the facility neurons in its subgroup as well as any other neuron that may place a facility in the same location.

Continuing with the description of the network structure we now define the neuron connections. The connections will be between client neurons in C and facility neurons in F. The neurons will connect based on their facility index i.e., a client neuron $C_{i,j}$ will have a connection to a facility neuron $F_{j,s}$. The connection values will be the distance between the client location and the facility location. For the above example, the connection between $C_{i,j}$ and $F_{j,s}$ will have the value $D_{i,s}$ where D is the distance matrix used to store the distances between locations. The value that one neuron outputs to another will be the sending neuron's state value multiplied by the connection value. Continuing the previous example, the output that $C_{i,j}$ sends to $F_{j,s}$ will be $stateValue(C_{i,j}) \cdot D_{i,s}$ where $stateValue()$ is a function that returns a neuron's state value. Figure 6.2 contains two connection visualizations: Figure 6.2 (a) shows the neurons that a client neuron is connected to and Figure 6.2 (b) shows the neurons that a facility neuron is connected to.

We now expand on the neuron state value update process mentioned earlier. The neurons will repeatedly evaluate and update their state values until no further changes occur at which point the network is said to have stabilized and a solution can be derived from the neuron state values. The network will attempt to activate neurons that have smaller connection values than their competitors which corresponds to searching for good or optimal solutions as the connection values are based on the distances between clients and facilities. While there is no explicit energy function to minimize, this process functions similarly as a stabilized state containing small connection values corresponds to a low energy state in a Hopfield network.

The main difference is that the neuron update function now contains an extra step in regards to the changed state value. When a neuron is selected for an update it starts by summing its incoming connection values to produce a cost value. It then compares its cost value with the minimum cost value of the neurons in its competing subgroup. If no other competing neuron is active or the neuron's cost value is lower than the groups minimum value the neuron activates, otherwise it deactivates.

The state value is then passed to an accept function that determines if the state value will be



(a) Connections for $C_{2,1}$

(b) Connections for $F_{2,3}$

Figure 6.2: Example neuron connections for Haralampiev's network where $n = 3$ and $k = 2$. Figure (a) displays the connections for $C_{2,1}$ and Figure (b) displays the connections for $F_{2,3}$.

flipped to the opposite value. The accept function uses a cost difference Δ and a temperature value T to generate the probability to flip the neuron's state value. The cost difference Δ is the difference between the neuron's cost value and the best competing cost value i.e., $\Delta = |neuronCost - BestCost|$. A high Δ will decrease the probability of flipping the state value whereas a low Δ will increase this probability.

The temperature value T modifies the probability in the opposite way: A high T value increases the probability of flipping the neuron state value whereas a low T value decreases the probability of flipping it. T is used for exploration as it will start at a high value that results in most state values being flipped before being lowered so that only state value changes that reduce the cost of the network will be accepted. Haralampiev defines the probability of flipping a neuron state value using Δ and T as $1/(1 + exp(\Delta/T))$.

When using Haralampiev's network we will initialize the neuron state values and wait for the network to stabilize. Like our modified Hopfield network, we refer to each iteration of this initialization and stabilization process as a run of the network and we try to improve the solution by running the network multiple times. Haralampiev's suggestion for neuron state value initialization is to randomly set every neuron state value to 0 or 1.

We differ by using a previous solution to set the initial neuron state values. When we run the network we generate a random solution to the current k-median problem and use the selected facilities in the solution to initialize the neuron state values. The neuron state values are set such that the network is in a stable state from which the input facilities can be derived. For example, given two facilities $\{1, 3\}$ the network will activate $F_{1,1}$ and $F_{2,3}$ and then activate the client neurons such that each client is connected to the nearest facility. We use this approach as it results in faster stabilization compared to a completely random start and it still encourages exploration as a high temperature will immediately move the network out of the starting stable state.

When we run the network multiple times we only generate a random solution for the first run. The subsequent runs are initialized with the best solution found in the previous runs. In testing we found this to produce better solutions than selecting a random solution for each run.

6.1.2 Algorithm

We present pseudocode for three of the Haralampiev network functions: *run*, *update*, and *accept*. The *run* function is the main function of the algorithm and it selects which neuron to update and modifies the temperature value over time. The *update* function takes in a neuron and determines the neuron's state value. The *accept* function uses neuron cost values and the temperature to determine if a neuron state value will be flipped to the opposite value.

Algorithm 6 contains pseudocode for the *run* function. The function will update every neuron in a random order before decreasing the temperature. This differs from Haralampiev's suggestion of using a specified number of updates between temperature changes and selecting any neuron at random for an update. We use our approach as we found the network stabilized faster without any noticeable difference in solution quality.

For the temperature we want the initial value to start high and be reduced to a near 0 value over time. To do this we select a value α that will be used to decrease the temperature. We select α externally using the following steps: select an initial *temperature* value, decide on the number of temperature updates w that will reduce *temperature* to almost 0, and solve the

equation $temperature \cdot \alpha^w = 0.001$ for α. For example, if $temperature = 1$ and $w = 3$ we have $1 \cdot \alpha^3 = 0.001$ which produces $\alpha = 0.1$.

Algorithm 6 run (*temperature*, α)

```
# α is selected to reduce temperature to a near 0 value after a desired number
# of while loop iterations.
Input: temperature, α
hasStabilized ← false
# Note that the update process ensures that a stabilized network contains a valid solution.
while not hasStabilized do
    for Each neuron x in a random order do
        update(x, temperature)
    end for
    temperature ← temperature · α
    # Call a helper function to see if the network has stabilized
    # i.e., each competing subgroup has exactly one active neuron.
    hasStabilized ← checkNetworkStabilization()
end while
```

Algorithm 7 contains pseudocode for the *update* function which determines the potential state change of a neuron. The pseudocode follows the steps listed previously for updating neurons and we make no changes to Haralampiev's approach here.

Algorithm 8 contains pseudocode for the *accept* function which determines if a neuron's state value remains the same or is flipped to the opposite value. It does this by generating a probability to flip the neuron's state value based on a cost difference Δ and the temperature. The cost difference Δ is the difference between the neuron's cost value and the smallest cost value of its competitors.

Δ increases the probability of flipping the neuron's state value when it is low and decreases the probability when it is high. This is to encourage retaining good changes such as keeping a neuron active when it has a much smaller cost value than its competitors. The temperature functions differently as it increases the probability to flip the state value when high and decreases it when low. This promotes exploration as this value will start high and will be decreased over time.

The interaction between Δ and temperature can be seen in Table 6.1. In this table we see that temperature has a greater influence on the resulting probability value than Δ. When the temperature is equal to 5 we see that there is a non-zero probability for all Δ values. This table also shows how a low temperature value allows the network to stabilize as the probabilities become 0 when the temperature is equal to 0.1.

We make one modification to Haralampiev's *accept* function by adding a check to see if Δ is 0 as this results in the probability becoming 0.5. This was an issue in our testing as without the check the network could randomly activate neurons when the system is close to stabilization which greatly prolonged the algorithm runtime.

Algorithm 7 update (*neuron*, *temperature*)

Input: *neuron*, *temperature*

```
# getOnCompetitors() returns the active neurons in the competing subgroup.
onCompetitors ← getOnCompetitors(neuron)
# If no competitors are active.
if onCompetitors is empty then
    # V is the array containing the neuron state values.
    V[neuron] ← 1
else
    # costValue() returns the sum of the input connection values for a neuron.
    neuronValue ← costValue(neuron)
    # calculateCostValues() returns a list of values produced by calling
    # costValue() on each neuron in onCompetitors.
    onValues ← calculateCostValues(onCompetitors)
    bestValue ← min(onValues)
    # Δ is the cost difference between the neuron's cost value and its best competitor.
    Δ ← |neuronValue − bestValue|

    if neuronValue < bestValue then
        V[neuron] = 1
        if accept(Δ, temperature) then
            V[neuron] = 0
        end if
    else
        V[neuron] = 0
        if accept(Δ, temperature) then
            V[neuron] = 1
        end if
    end if
end if
```

		Δ			
		1	2	4	8
Temperature	0.1	0.00	0.00	0.00	0.00
	1	0.27	0.12	0.02	0.00
	5	0.45	0.40	0.31	0.17

Table 6.1: Sample probability values for different cost difference Δ and temperature values. The formula used to generate each value is $1/(1 + exp(\Delta/T))$.

Algorithm 8 accept (Δ, *temperature*)

```
  # Δ is the difference between the current neuron's cost value
  # and that of the smallest competing cost value.
Input: Δ, temperature
  # If Δ is 0 then the probability becomes 0.5.
  # In this case we immediately return false.
  if Δ == 0 then
      return False
  else
      probability ← (1/(1 + exp(Δ/Temperature)))
      # RandomValue(0, 1) returns a random real number between 0 and 1.
      return RandomValue(0, 1) < probability
  end if
```

6.2 Local Search Algorithm

As discussed in Section 2.3.2 Arya et al.'s local search algorithm operates by repeatedly attempting to swap facility locations with non-facility locations in order to reduce the cost of the solution. If the algorithm swaps p locations in one operation then it has an approximation ratio of $3+2/p$ [2]. Our implementation of the algorithm will swap only one non-facility and facility location per operation giving a worst-case approximation ratio of 5 [2]. However, in practice we often end up with better approximation results. Algorithm 9 contains the pseudocode for our local search algorithm.

We add two early exit conditions to the algorithm in order to improve the runtime. The first exit condition is a simple timeout check that bounds the maximum runtime. When that bound is exceeded the algorithm stops and returns the current solution. The second exit condition is a minimum improvement requirement. As the algorithm runs, the later swaps tend to contribute smaller improvements to the total cost of the solution. Instead of spending a large amount of time getting minuscule improvements, we stop making swaps when they cause an improvement below a certain threshold. We use the following threshold equation presented by Cohen-Addad and Mathieu [13] and only swap when it holds:

$$distance(F') \leq (1 - 1/n)distance(F)$$

where F is the original set of selected facilities and F' is the set created by swapping a member of F with another location. The $distance()$ function returns the total distance from clients to their nearest facilities and n is the number of locations.

Our local search implementation uses the GPU for distance calculations to allow a fair comparison with our GPU-bound Hopfield network. We implemented the distance calculation as a series of matrix operations and this greatly improved the algorithm runtime.

Algorithm 9 Local Search Algorithm (*maxTime*, *initialFacilities*)

Input: *maxTime*, *initialFacilities*

```
# calculateDistance returns the total distance from all clients to their nearest facility
bestDistance ← calculateDistance(initialFacilities)
facilitySet ← initialFacilities
while true do
    # allLocations is a set containing every location. By removing the facilities
    # from this set we get the set of client locations.
    clientSet ← allLocations − facilitySet
    # For every client location we check if swapping it with a facility location will
    # improve the solution.
    for client in clientSet do
        # swapCandidate will contain the facility that will be swapped with the
        # current client if there exists a swap that reduces the total cost of the solution.
        swapCandidate ← none
        for facility in facilitySet do
            if Maximum time has been reached then
                return facilitySet
            end if
            # For each client-facility pair we check if the corresponding swap results in a
            # better solution. swapFacility() creates a new facility set where the
            # inputted facility and client have been swapped.
            newFacilities ← swapFacility(facilitySet, facility, client)
            newDistance ← calculateDistance(newFacilities)
            # A swap is only considered if it satisfies our distance reduction requirement.
            if newDistance < (1 − (1/n)) ∗ bestDistance then
                BestDistance ← newDistance
                swapCandidate ← facility
            end if
        end for
        # If a satisfying swap has been found then swap the facility and client
        # and return to the start of the while loop.
        if swapCandidate is not none then
            facilitySet ← swapFacility(facilitySet, swapCandidate, client)
            break
        end if
    end for
    # If there is no better swap or no potential swap satisfies the minimum improvement
    # requirement then return the current solution and exit.
    if No swap has occurred then
        return facilitySet
    end if
end while
```

6.3 Datasets

6.3.1 Random Point Datasets

Our first two datasets are constructed by randomly selecting points on a 2D plane. Each point i is a potential location and is assigned a randomly generated coordinate pair (x_i, y_i) where x_i and y_i can take any value between 0 and 1.

Each individual test was constructed using a n and k pair. Each test generated n points and solved the k-median problem for k. We formulated the problems as integer linear programs and used the Coin-OR solver [39] to produce optimal solutions.

The first dataset is called the random-small dataset. The n value for a test pair can be one of $[20, 30, 40, 50, 60, 70, 80, 90, 100]$ and the k value can be one of $[2, 3, 4, 5, 6, 7, 8, 9, 10]$. Each n and k pair has 100 tests.

The second dataset is the random-large dataset. Its n value can be one of $[500, 600, 700, 800]$ and its k value can be one of $[20, 30, 40, 50]$. Each n and k pair has 3 tests.

6.3.2 Cities USCA312 Dataset

The third dataset uses cities located across North America as the potential facility locations. We use John Burkardt's USCA312 dataset [6] which provides a list of 312 cities and their location information. The dataset converts a city's longitude and latitude into an equivalent x and y coordinate pair which we used for our calculations.

We used this data to construct several tests. For each n and k pair we randomly selected n locations and used the Coin-OR solver to produce a solution. The n value can be any multiple of 10 between 30 and 300 and the k value is any integer between 2 and 10. Each n and k pair has 100 tests.

6.3.3 OR-Library P-Median Dataset

The fourth dataset comes from the OR-Library [5] and contains 40 tests labelled pmed1 through pmed40. The dataset locations are nodes in a graph and the dataset provides a list of edge values that correspond to the distance between nodes. We used this list of edge values to construct a distance matrix for the nodes. Additionally, each test is assigned a k value and contains the value of an optimal solution for each k-median problem. The n values range from 10 to 900 and the k values range from 5 to 200. Each test file contains one test using a single n and k pair.

6.3.4 TSPLib Dataset

The final dataset is the TSPLib dataset [37] which contains large problems for the travelling salesman problem. Each problem contains over 1000 locations and each location is specified by an x and y coordinate. This dataset has been adapted for the k-median problem by García et al. [15] and they provide solutions for several different k values. There are several tests per problem with k values ranging from 5 to 5000.

6.4 Testing Environment

All tests were performed on a desktop computer running Ubuntu 21.10. The machine has a 6 core AMD Ryzen 5 1600X CPU, 16 GB of RAM, and a Nividia GeForce GTX 1060 GPU with 6 GB of memory.

The code is written in python and makes extensive use of the PyTorch framework [36]. PyTorch allows us to express matrices as tensors and perform calculations on a CUDA supported Nividia GPU. This greatly improves performance as we structured most of the calculations as matrix operations which can be executed in parallel on the GPU.

6.5 Results

We present test results for our modified Hopfield network, Haralampiev's competition-based neural network [16, 17, 18], and Arya et al.'s local search algorithm [2]. For each test we report two statistics: the approximation ratio and the total runtime in seconds. The approximation ratio is calculated as S/S^* where S is the value of the solution computed by an algorithm and S^* is the value of the optimal solution. The total runtime of a test is calculated by starting a timer when an algorithm begins initialization and ending the timer when a solution is returned. When there are multiple tests for a single n and k pair we return the average approximation ratio and average runtime of each algorithm for the n and k pair.

In the tests we use our modified Hopfield network from Section 5 and include the extensions described in Section 5.3. The network was allotted 1, 4, and 10 runs for each test and the best solution found was returned.

Haralampiev's network functions as described in Section 6.1. We present results for 4 runs of Haralampiev's network and directly compare the results with 4 runs of our modified Hopfield network. The network will use $temperature = 1$ and $\alpha = 0.1$ for each test as we found that these values produced good solutions while also stabilizing quickly. Note that we do not include TSPLib results as the Haralampiev network tests did not complete despite being given several days of computation time. We also ran Haralampiev's network on the CPU due to the update process not being optimized for GPU calculations. When running Haralampiev's network on the GPU we found that the run times were significantly longer.

The local search algorithm operates as described above in Section 6.2. We allowed a maximum runtime of 1, 5, or 15 seconds and for a fair comparison we attempt to match the local search runtime with the Hopfield network runtime.

Tables 6.2, 6.3, and 6.4 contain a selection of the test results. Refer to Appendix B for the complete set of test results.

6.6 Analysis

From these test results we can view a few general trends when comparing the algorithms. For the Haralampiev network comparison we see that both networks produce good approximations with Haralampiev's often returning the better solution. The caveat is that Haralampiev's network does not appear to scale as well as our modifed Hopfield Network. For the random-small

			Our Network (4 runs)		Haralampiev's Network (4 runs)	
Test	n	k	Ratio	Time (s)	Ratio	Time (s)
random-small	20	5	1.09	0.04	1.05	0.10
random-small	50	5	1.08	0.11	1.02	0.41
random-small	100	5	1.08	0.21	1.01	1.13
random-large	500	50	1.12	1.32	1.07	91.57
random-large	600	50	1.09	1.67	1.06	126.27
random-large	700	50	1.10	1.85	1.05	167.77
random-large	800	50	1.09	2.78	1.05	235.13
USCA312	100	5	1.08	0.20	1.05	0.42
USCA312	200	5	1.11	0.43	1.06	1.95
USCA312	300	5	1.06	0.64	1.01	8.42
pmed25	500	167	1.34	1.21	1.38	47.05
pmed30	600	200	1.31	1.75	1.37	76.52
pmed35	800	5	1.04	2.06	1.01	3.45
pmed40	900	90	1.10	2.72	1.18	58.17

Table 6.2: Selection of test results comparing our modified Hopfield network with Haralampiev's network.

			Our Network (10 Runs)		Local Search	
Test	n	k	Ratio	Time (s)	Ratio	Time (s)
random-small	20	5	1.06	0.11	1.07	0.01
random-small	50	5	1.06	0.25	1.04	0.03
random-small	100	5	1.06	0.47	1.02	0.09
random-large	500	50	1.11	3.25	1.12	5
random-large	600	50	1.08	4.10	1.14	5
random-large	700	50	1.08	4.95	1.16	5
random-large	800	50	1.08	6.41	1.18	5
USCA312	100	5	1.07	0.43	1.02	0.08
USCA312	200	5	1.10	0.88	1.01	0.24
USCA312	300	5	1.05	1.35	1.01	0.48
pmed25	500	167	1.38	3.36	1.42	5
pmed30	600	200	1.31	4.10	1.51	5
pmed35	800	5	1.04	4.58	1.00	5
pmed40	900	90	1.10	5.78	1.31	5
rl1304	1304	50	1.36	14.20	1.61	15
fl1400	1400	50	1.97	6.32	2.44	5
u1432	1432	50	1.34	16.58	1.62	15
vm1748	1748	50	1.29	22.27	1.57	15

Table 6.3: Selection of test results comparing our modified Hopfield network with the local search algorithm.

			Our Network (1 Run)		Our Network (10 runs)	
Test	n	k	Ratio	Time (s)	Ratio	Time (s)
usa13509	13509	300	1.50	838	1.47	6583
usa13509	13509	400	1.51	831	1.48	6521
usa13509	13509	500	1.51	830	1.49	3291
usa13509	13509	800	1.53	825	1.52	5408
usa13509	13509	1000	1.54	812	1.52	5448
usa13509	13509	2000	1.55	785	1.55	6001
usa13509	13509	3000	1.54	771	1.54	6614
usa13509	13509	4000	1.63	750	1.63	5921
usa13509	13509	5000	1.72	774	1.70	6261

Table 6.4: The performance of our modified Hopfield network on the TSPLib dataset usa13509. Note that the dataset does not provide an optimal solution for k values of 600, 700, and 900.

and USCA312 datasets our modified Hopfield network runtime stays under one second whereas Haralampiev's network takes up to 8 seconds in the larger tests. The pmed and random-large tests can take several minutes with the TSPLib dataset being excluded as the Haralampiev network tests did not complete even with several days of runtime.

This makes Haralampiev's network a good choice for smaller problems, especially as the network size of $2 \cdot n \cdot k$ neurons will be significantly smaller than our $n^2 + n$ neuron network. However, as the problems get larger the runtime becomes infeasible and our network may be preferred. For the random-large $n = 800$ and $k = 50$ example we could run our network many times in the 235 seconds it takes to run Haralampiev's network four times as our approach takes approximately 0.7 seconds per run.

Furthermore, if we compare the runtimes for the random-large dataset results in Appendix B we see that on average our network stabilizes 67 times faster than Haralampiev's network. This speedup can increase dramatically as n grows as we see individual tests complete up to 145 times faster when $n = 800$. When performing multiple runs of the networks, our approach produces a solution up to 285 times faster for four runs when $n = 800$. This multiple run speedup comes from the state space exploration functionality described in Section 5.3.2. Our network will narrow down the search space over time in order to focus on promising areas for facility locations. This decreases the runtime as only locations in those promising areas will be evaluated.

This runtime difference can also be attributed to how the Haralampiev network selects and updates neurons. In each iteration of the main loop every neuron is updated before the network is checked for stabilization. This can result in wasted updates as the neurons stop changing their state values, particularly when the network is close to a stable state. Our network addresses this by choosing neurons that are likely to change, and keeping track of neurons that have not changed. Additionally, our network groups client neuron updates which greatly decreases runtime in comparison to individually updating every neuron.

For the local search comparison we see that when the n and k values are relatively small ($n \leq 300$ and $k = 5$) the local search algorithm outperforms our network. This is unsurprising because even though there are an exponential number of possible solutions, the problem sizes

are still small enough that the local search algorithm can quickly evaluate a significant number of solutions. While our network does return competitive solutions, it cannot compete with a method that can exhaustively search the state space in a short amount of time.

We also note that modern computing power makes solving these smaller problems trivial. Although not presented, both brute force methods and integer linear program solvers can provide the optimal solution in a time frame of seconds to minutes. Additionally, our early testing of the local search algorithm without GPU functionality gave similar optimal and near optimal solutions for these small problems.

When the values of n and k start to grow ($n > 300$ and $k > 5$) we see our network outperforming the local search algorithm. Here the local search algorithm suffers due to the exponential number of potential solutions. In contrast, our network appears to scale well as it returns a better solution in all cases except for the pmed35 test where the small value of k greatly reduces the number of potential solutions.

We attribute this scaling to how our modified Hopfield network performs updates: the n^2 client-facility neurons are updated in a single operation no matter the size of n and while the facility neurons need to be updated individually, there are at most n neurons and we have introduced measures to reduce unnecessary updates. This scaling allows our network to process relatively large problems such as the TSPLib usa13509 dataset which contains 13509 locations. This problem size is significant as both the local search algorithm and Haralampiev's network would require a huge amount of processing time to produce a solution. Table 6.4 shows the performance of our network on this large dataset. We believe that our network will scale to larger values due to our resource usage growing quadratically, but our testing was restricted by the testing machine's 6 gigabytes of GPU memory.

Chapter 7

Conclusions

In this thesis we provide a neural network approach for solving the k-median problem. Here we are given n locations and want to select k locations such that the total distance between each unselected location and its nearest selected location is minimized.

Our approach is based on modifying a Hopfield network for the k-median problem. We acknowledge that when Hopfield networks are applied to combinatorial optimization problems they may take an unfeasibly large time to produce solutions as well as provide solutions that are of poor quality or invalid. We address these issues by rewriting the neuron update functions in order to efficiently produce valid solutions.

We empirically show that our modified Hopfield network quickly produces approximate solutions that are often within 2 times of the optimal solution. Furthermore, we demonstrate that our approach scales well by producing solutions for problems where $n = 13509$. To demonstrate the effectiveness of our approach we compare our modified Hopfield network with Haralampiev's competition-based neural network [16, 17, 18] and Arya et al.'s local search algorithm [2]. Our network stabilizes significantly faster than Haralampiev's in the majority of the tests while also providing competitive solutions. Our network stabilizes up to 145 times faster than Haralampiev's when $n = 800$.

For the local search algorithm our network outperforms it once $n \geq 300$. We compare the two approaches by taking the runtime required for our network to produce a solution and using resulting runtime as the maximum runtime for the local search algorithm. I.e., if our network takes X seconds to produce a solution, we compare it with the solution provided from running the local search algorithm for X seconds. For these large problems the solution returned by our approach is better in 76% of tests when $n \geq 300$ and 86% of tests when both $n \geq 300$ and $k \geq 20$.

Additionally, our restructuring of the update operations allows us to efficiently run our network on a GPU which greatly improved runtimes. We also extend the neuron selection algorithm to focus on neurons that have a higher probability to change their status which further improves the network runtime. Finally, we believe that our network will scale well to larger problems given sufficient GPU memory for storing the matrices used by our neural network.

7.1 Future Work

A starting point for improving our approach is enhancing how our modified Hopfield network searches the state space. Our implementation uses a relatively simple method of evaluating increasingly smaller sets of locations near previously selected facilities. While this does provide better solutions our testing shows that the improvements plateau relatively quickly. Investigating alternate or more complex search heuristics could greatly improve our results. For example, we could combine our modified Hopfield network with another algorithm such as the local search algorithm. In this case the network would produce a solution which would then be improved by the second algorithm. The resulting improved solution could be returned as is or used as the basis for a new search.

Another area of improvement is optimizing or rewriting the Hopfield network update operations. We have shown that converting the update operations of a Hopfield network into a series of matrix calculations allows us to utilize GPUs to quickly produce solutions. We believe that improvements can be found by further optimizing our update operations or using alternate update operations. For example, would grouping together facility updates improve the total runtime without significantly reducing the solution quality? Also, could we apply reinforcement learning to our client updates in order to only update the sections of the client matrix that may change?

Additionally, we can move away from the k-median problem and use the ideas presented here to make modified Hopfield networks for other combinatorial optimization problems. A starting point would be clustering problems such as the facility location problem or the k-means problem. These would be good candidates as their similarity to the k-median problem could allow us to reuse parts of the network structure and update operations.

A more difficult problem would be applying these ideas to create a modified Hopfield network for the travelling salesman problem. This is the original problem presented by Hopfield and Tank [20] and subsequent literature discusses the viability of using Hopfield networks as a solver. Creating a modified Hopfield network for this problem would allow direct comparisons of our approach with the results in those papers.

Furthermore, we could evaluate larger Hopfield network sizes. As mentioned in Section 3.1 artificial neural networks are an attempt at emulating the natural computing power of neurons in the brain. These relatively simple biological neurons can perform complex tasks when working together in large groups. However, the number of artificial neurons in our Hopfield network is significantly smaller than the number of biological neurons found in the brain. Feedforward neural networks have found success by increasing the number of neuron layers and these larger networks are referred to as deep neural networks. The question then is can we get better results by adding more neurons to a Hopfield network in order to create a ”deep” Hopfield network and what would this new network look like? The issue here would be that increasing the network size would likely exacerbate the runtime and stability problems already present in Hopfield networks. Ideally our method of incorporating constraints into the update operations as well as moving the calculations to the GPU would provide a start to alleviating these issues on large networks.

Bibliography

[1] S. Aiyer, M. Niranjan, and F. Fallside. A theoretical investigation into the performance of the Hopfield model. *IEEE Transactions on Neural Networks*, 1(2):204–215, 1990.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local Search Heuristics for k -Median and Facility Location Problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[3] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE Comput. Soc. Press, 1996.

[4] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*, pages 161–168. ACM Press, 1998.

[5] J. E. Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[6] J. Burkardt. Cities-City Distance Datasets. `https://people.math.sc.edu/Burkardt/datasets/cities/cities.html`, 2011.

[7] J. Byrka, S. Dudycz, P. Manurangsi, J. Marcinkowski, and M. Włodarczyk. To Close Is Easier Than To Open: Dual Parameterization To k-Median. In *Approximation and Online Algorithms*, volume 12806, pages 113–126. Springer International Publishing, 2021.

[8] J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An Improved Approximation for k -median, and Positive Correlation in Budgeted Optimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 737–756. Society for Industrial and Applied Mathematics, 2015.

[9] M. Charikar, S. Guha, É. Tardos, and D. Shmoys. A Constant-Factor Approximation Algorithm for the k-Median Problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

[10] X. Chen, Z. Tang, X. Xu, S. Li, G. Xia, Z. Zong, and J. Wang. An Hopfield network learning for minimum vertex cover problem. In *SICE 2004 Annual Conference*, volume 2, pages 1150–1155, 2004.

[11] M. Chrobak, C. Kenyon, and N. Young. The reverse greedy algorithm for the metric k-median problem. *Information Processing Letters*, 97(2):68–72, 2006.

[12] V. Cohen-Addad, A. Gupta, A. Kumar, E. Lee, and J. Li. Tight FPT Approximations for k-Median and k-Means. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, pages 42:1–42:14, 2019.

[13] V. Cohen-Addad and C. Mathieu. Effectiveness of Local Search for Geometric Optimization. *31st International Symposium on Computational Geometry (SoCG 2015)*, 34:329–343, 2015.

[14] R. Galvão. A Dual-Bounded Algorithm for the *p* -Median Problem. *Operations Research*, 28(5):1112–1121, 1980.

[15] S. García, M. Labbé, and A. Marín. Solving Large *p*-Median Problems with a Radius Formulation. *INFORMS Journal on Computing*, 23(4):546–556, 2011.

[16] V. Haralampiev. Neural network approaches for a facility location problem. *Mathematical Modeling*, 4(1):3–6, 2020.

[17] V. Haralampiev. Theoretical Justification of a Neural Network Approach to Combinatorial Optimization. In *Proceedings of the 21st International Conference on Computer Systems and Technologies '20*, pages 74–77, 2020.

[18] V. Haralampiev. *Neural Networks for Facility Location Problems*. PhD thesis, Sofia University, "St. Kliment Ohridski", 2021.

[19] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[20] J. Hopfield and D. Tank. "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, 1985.

[21] Z. Huang, Y. Zhou, X. Xia, and X. Lai. An improved (1+1) evolutionary algorithm for k - median clustering problem with performance guarantee. *Physica A: Statistical Mechanics and its Applications*, 539, 2020.

[22] K. Jain and V. Vazirani. Approximation algorithms for metric facility location and *k* - Median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[23] P. Järvinen, J. Rajala, and H. Sinervo. A Branch-and-Bound Algorithm for Seeking the *P*-Median. *Operations Research*, 20(1):173–178, 1972.

[24] B. Kamgar-Parsi and B. Kamgar-Parsi. On problem solving with Hopfield neural networks. *Biological Cybernetics*, 62(5):415–423, 1990.

[25] O. Kariv and S. Hakimi. An Algorithmic Approach to Network Location Problems. II: The p-Medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[27] S. Li and O. Svensson. Approximating k-Median via Pseudo-Approximation. *SIAM Journal on Computing*, 45(2):530–547, 2016.

[28] Y. Liang. Combinatorial optimization by Hopfield networks using adjusting neurons. *Information Sciences*, 94(1-4):261–276, 1996.

[29] J. Lin and J. Vitter. ϵ-approximations with minimum packing constraint violation (extended abstract). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing - STOC '92*, pages 771–782. ACM Press, 1992.

[30] T. D. Manjunath, S. Samarth, N. Prafulla, and J. Nayak. Hopfield Network Based Approximation Engine for NP Complete Problems. In *Innovative Data Communication Technologies and Application*, volume 46, pages 319–331. Springer International Publishing, 2020.

[31] A. Masone, C. Sterle, I. Vasilyev, and A. Ushakov. A three-stage p -median based exact method for the optimal diversity management problem. *Networks*, 74(2):174–189, 2019.

[32] C. Nagarajan and D. Williamson. An experimental evaluation of incremental and hierarchical k-median algorithms. In *Experimental Algorithms*, pages 169–180, 2011.

[33] S. Narula, U. Ogbu, and H. Samuelsson. An Algorithm for the p -Median Problem. *Operations Research*, 25(4):709–713, 1977.

[34] F. Ndiaye, B. Ndiaye, and I. Ly. Application of the p-Median Problem in School Allocation. *American Journal of Operations Research*, 02(02):253–259, 2012.

[35] R. Pan and D. Zhu. An Efficient Local Search Algorithm for k-Median Problem. *Proceedings of the 6th WSEAS international conference on Applied computer science*, pages 19–24, 2006.

[36] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, 2019.

[37] G. Reinelt. TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[38] S. Salcedo-Sanz and X. Yao. A Hybrid Hopfield Network-Genetic Algorithm Approach for the Terminal Assignment Problem. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 34(6):2343–2353, 2004.

[39] M. Saltzman. Coin-Or: An Open-Source Library for Optimization. In *Programming Languages and Systems in Computational Economics and Finance*, volume 18, pages 3–32. 2002.

[40] E. Senne, L. Lorena, and M. Pereira. A branch-and-price approach to p-median location problems. *Computers & Operations Research*, 32(6):1655–1664, 2005.

[41] S. Silva, M. Costa, and C. Filho. In *Customized Genetic Algorithm for Facility Allocation using p-median*, pages 165–169, 2019.

[42] D. Tayebi, S. Ray, and D. Ajwani. Learning to Prune Instances of k-median and Related Problems. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 184–194. Society for Industrial and Applied Mathematics, 2022.

[43] B. Wilder, E. Ewing, B. Dilkina, and M. Tambe. End to end learning and optimization on graphs. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[44] G. Wilson and G. Pawley. On the stability of the Travelling Salesman Problem algorithm of Hopfield and Tank. *Biological Cybernetics*, 58(1):63–70, 1988.

[45] Y. Won and K. Currie. Efficient p -median mathematical programming approaches to machine-part grouping in group technology manufacturing. *Engineering Optimization*, 36(5):555–573, 2004.

Appendix A

K-Median Hopfield Network Energy Equation

A.1 Constraint Term 1

Steps for rearranging Equation (4.1):

$$C_1 \sum_{i=1}^{n} \sum_{j=1}^{n} V(i, j) D_{i,j}$$

We start by rewriting the equation in order to remove the function $V(i, j)$ and replacing it with the neurons that it maps to. As a reminder, $V(i, j) = V_{n*i+j}$ where the i and j values come from the client-facility $y_{i,j}$ variable and n is the number of locations.

$$C_1 \sum_{i=1}^{n} \sum_{j=1}^{n} V_{n*i+j} D_{i,j} \tag{A.1}$$

For this constraint term we are evaluating only the client-facility neurons and we can simplify further by rewriting the two summations into one summation that evaluates neurons $n + 1$ to $n^2 + n$. For convenience we define $N = n^2 + n$. Reducing to one summation means that we lose the j variable and therefore cannot use the $D_{i,j}$ variable for distance. We define a new distance function $D(i) = D_{\lfloor (i-1)/n \rfloor, ((i-1) \bmod n)+1}$ that takes a client-facility neuron i and returns the distance between the represented client and facility.

$$C_1 \sum_{i=n+1}^{N} V_i D(i) \tag{A.2}$$

Multiply by $\frac{-1}{-1}$ in order to make the term negative and rearrange to follow the form of Equation (3.2):

$$- \sum_{i=n+1}^{N} V_i (-C_1 D(i)) \tag{A.3}$$

A.2 Constraint Term 2

Steps for rearranging Equation (4.2):

$$C_2\left(\left(\sum_{i=1}^{n} V_i\right) - k\right)^2$$

Expand the squared term:

$$C_2\left(k^2 - 2k\sum_{i=1}^{n} V_i + \left(\sum_{i=1}^{n} V_i\right)^2\right) \tag{A.4}$$

Expand the squared summation:

$$C_2\left(k^2 - 2k\sum_{i=1}^{n} V_i + \sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j\right) \tag{A.5}$$

Multiply in the constant:

$$C_2k^2 - 2C_2k\sum_{i=1}^{n} V_i + C_2\sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j \tag{A.6}$$

Multiply by $\frac{-2}{-2}$ in order to get a leading $-\frac{1}{2}$ in the first term:

$$-\frac{1}{2}C_2\sum_{i=1}^{n}\sum_{j=1}^{n} -2V_iV_j - 2C_2k\sum_{i=1}^{n} V_i + C_2k^2 \tag{A.7}$$

Rearrange the variables in order to correspond to the position of the $T_{i,j}$ and I_i variables in Equation (3.2). Drop the constant term C_2k^2 as it adds a constant amount of energy to the equation:

$$-\frac{1}{2}\sum_{i}^{n}\sum_{j}^{n}(-2C_2)V_iV_j - \sum_{i}^{n} V_i(2C_2k) \tag{A.8}$$

A.3 Constraint Term 3

Steps for rearranging Equation (4.3):

$$C_3\sum_{i=1}^{n}\left(\left(\sum_{j=1}^{n} V(i,j)\right) - 1\right)^2$$

Expand the first summation:

$$C_3\left(\left(\sum_{j=1}^{n} V(1,j)\right) - 1\right)^2 + C_3\left(\left(\sum_{j=1}^{n} V(2,j)\right) - 1\right)^2 + \cdots + C_3\left(\left(\sum_{j=1}^{n} V(n,j)\right) - 1\right)^2 \tag{A.9}$$

Each term in Equation (A.9) will evaluate a separate group of n client-facility neurons. Each of these n groups will correspond to one client location e.g., the first term will evaluate all client-facility neurons where the client is in location 1. The summations can then be rewritten to replace the $V(i, j)$ function with the actual neurons:

$$C_3\left(\left(\sum_{i=n+1}^{2n} V_i\right)-1\right)^2 + C_3\left(\left(\sum_{i=2n+1}^{3n} V_i\right)-1\right)^2 + \cdots + C_3\left(\left(\sum_{i=n^2+1}^{N} V_i\right)-1\right)^2 \tag{A.10}$$

Each individual term will then be rearranged. We show the steps for a single term as they will be the same for all terms. For a given term let a and b be the indices for the first and last neuron in that term, respectively:

$$C_3\left(\left(\sum_{i=a}^{b} V_i\right)-1\right)^2 \tag{A.11}$$

Expand the squared term:

$$C_3\left(1-2\sum_{i=a}^{b} V_i + \left(\sum_{i=a}^{b} V_i\right)^2\right) \tag{A.12}$$

Expand the squared summation:

$$C_3\left(1-2\sum_{i=a}^{b} V_i + \sum_{i=a}^{b}\sum_{j=a}^{b} V_iV_j\right) \tag{A.13}$$

Multiply in the constant:

$$C_3\sum_{i=a}^{b}\sum_{j=a}^{b} V_iV_j - 2C_3\sum_{i=a}^{b} V_i + C_3 \tag{A.14}$$

Multiply by $\frac{-2}{-2}$ in order to get a leading $-\frac{1}{2}$ in the first term:

$$-\frac{1}{2}C_3\sum_{i=a}^{b}\sum_{j=a}^{b} -2V_iV_j - 2C_3\sum_{i=a}^{b} V_i + C_3 \tag{A.15}$$

Rearrange the variables in order to correspond to the position of the $T_{i,j}$ and I_i variables in Equation (3.2). Drop the constant term C_3 as it adds a constant amount of energy to the equation:

$$-\frac{1}{2}\sum_{i=a}^{b}\sum_{j=a}^{b}(-2C_3)V_iV_j - \sum_{i=a}^{b} V_i(2C_3) \tag{A.16}$$

With the term rearranged we can replace the indices a and b with the indices from (A.10) and produce the full constraint term:

$$
\begin{aligned}
&-\frac{1}{2}\sum_{i=n+1}^{2n}\sum_{j=n+1}^{2n}(-2C_3)V_iV_j-\sum_{i=n+1}^{2n}V_i(2C_3)\\
&-\frac{1}{2}\sum_{i=2n+1}^{3n}\sum_{j=2n+1}^{3n}(-2C_3)V_iV_j-\sum_{i=2n+1}^{3n}V_i(2C_3)\\
&-\ \cdots\\
&-\frac{1}{2}\sum_{i=n^2+1}^{N}\sum_{j=n^2+1}^{N}(-2C_3)V_iV_j-\sum_{i=n^2+1}^{N}V_i(2C_3)
\end{aligned}
\tag{A.17}
$$

A.4 Constraint Term 4

Steps for rearranging Equation (4.4):

$$C_4\left(\left(\sum_{i=1}^{n}\sum_{j=1}^{n}V(i,j)\right)-n\right)^2$$

Expand the squared term:

$$C_4\left(n^2-2n\sum_{i=1}^{n}\sum_{j=1}^{n}V(i,j)+\left(\sum_{i=1}^{n}\sum_{j=1}^{n}V(i,j)\right)^2\right) \tag{A.18}$$

Using $N = n^2 + n$ we rewrite the summations to use the actual neuron indices:

$$C_4\left(n^2-2n\sum_{i=n+1}^{N}V_i+\left(\sum_{i=n+1}^{N}V_i\right)^2\right) \tag{A.19}$$

Expand the squared summation:

$$C_4\left(n^2-2n\sum_{i=n+1}^{N}V_i+\sum_{i=n+1}^{N}\sum_{j=n+1}^{N}V_iV_j\right) \tag{A.20}$$

Multiply in the constant:

$$C_4\sum_{i=n+1}^{N}\sum_{j=n+1}^{N}V_iV_j-2C_4n\sum_{i=n+1}^{N}V_i+C_4n^2 \tag{A.21}$$

Multiply by $\frac{-2}{-2}$ in order to get a leading $-\frac{1}{2}$ in the first term:

$$-\frac{1}{2}C_4\sum_{i=n+1}^{N}\sum_{j=n+1}^{N}-2V_iV_j-2C_4n\sum_{i=n+1}^{N}V_i+C_4n^2 \tag{A.22}$$

Rearrange the variables in order to correspond to the position of the $T_{i,j}$ and I_i variables in Equation (3.2). Drop the constant term C_4n^2 as it adds a constant amount of energy to the equation:

$$-\frac{1}{2}\sum_{i=n+1}^{N}\sum_{j=n+1}^{N}(-2C_4)V_iV_j - \sum_{i=n+1}^{N}V_i(2C_4n) \tag{A.23}$$

A.5 Constraint Term 5

$$C_5\sum_{i=1}^{n}\sum_{j=1}^{n}(1-V_i)V(j,i)$$

Expand the term:

$$C_5\left(\sum_{i=1}^{n}\sum_{j=1}^{n}(V(j,i)-V_iV(j,i))\right) \tag{A.24}$$

Split the summations:

$$C_5\left(\sum_{i=1}^{n}\sum_{j=1}^{n}V(j,i)-\sum_{i=1}^{n}\sum_{j=1}^{n}V_iV(j,i)\right) \tag{A.25}$$

Use $N = n^2 + n$ to replace the first summation with one that uses the actual neuron indices; Replace the V(j,i) function in the second summation with the neuron indices:

$$C_5\left(\sum_{i=n+1}^{N}V_i-\sum_{i=1}^{n}\sum_{j=1}^{n}V_iV_{nj+i}\right) \tag{A.26}$$

Multiply in the constant:

$$-C_5\sum_{i=1}^{n}\sum_{j=1}^{n}V_iV_{nj+i}+C_5\sum_{i=n+1}^{N}V_i \tag{A.27}$$

Multiply by $\frac{2}{2}$ in order to get a leading $-\frac{1}{2}$ in the first term:

$$-\frac{1}{2}C_5\sum_{i=1}^{n}\sum_{j=1}^{n}2V_iV_{nj+i}+C_5\sum_{i=n+1}^{N}V_i \tag{A.28}$$

Multiply by $\frac{-1}{-1}$ in order to make the second term negative:

$$-\frac{1}{2}C_5\sum_{i=1}^{n}\sum_{j=1}^{n}2V_iV_{nj+i}-C_5\sum_{i=n+1}^{N}-V_i \tag{A.29}$$

Rearrange the variables in order to correspond to the position of the $T_{i,j}$ and I_i variables in Equation (3.2):

$$-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(2C_5)V_iV_{nj+i} - \sum_{i=n+1}^{N}V_i(-C_5) \tag{A.30}$$

A.6 Connection and Input Bias Values

With the constraint terms in the form of Equation (3.2) we can now collect these terms and determine the $T_{i,j}$ and I_i values. Note that unlike the examples in Section 3, the constraint terms are defined for subsets of neurons. For example the k facility constraint term (4.2) applies only to the facility neurons numbered 1 to n and the n client-facility constraint term (4.4) applies only to the client-facility neurons numbered $(n+1)$ to (n^2+n). This means that our connection matrix T and input biases I will be defined by piecewise functions that use the indices of neurons i and j to determine their values. A neuron i is a facility neuron if $1 \leq i \leq n$ or a client-facility neuron if $(n+1) \leq i \leq (n^2+n)$.

For reference, we include below the general energy Equation (3.2). Note that the original equation uses n to denote the total number of neurons in the network. For clarity we maintain that n refers to the number of locations in the k-median problem and that our network contains $N = (n^2+n)$ neurons.

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}T_{ij}V_iV_j - \sum_{i=1}^{N}V_iI_i$$

For each constraint term, we will demonstrate how to derive the T and I values for our piecewise functions by matching the values in our constraint terms to the $T_{i,j}$ and I_i variables in Equation (3.2). For clarity, we will refer to $-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}T_{ij}V_iV_j$ as the *connection term* and $-\sum_{i=1}^{N}V_iI_i$ as the *bias term*. The piecewise functions will be constructed incrementally by evaluating each constraint term individually and adding the new values to the piecewise functions.

We begin with the first constraint term (A.3):

$$-\sum_{i=n+1}^{N}V_i(-C_1D(i))$$

This constraint term contains no value for T as there is no *connection term*. The *bias term* only evaluates client-facility neurons and the input bias I_i corresponds to $-C_1D(i)$. This gives us the first values for the piecewise functions:

$$T_{i,j} = 0$$

$$I_i = \begin{cases} 0 & \text{if } i \text{ is a facility neuron} \\ -C_1D(i) & \text{if } i \text{ is a client-facility neuron} \end{cases}$$

Next is the second constraint term (A.8):

$$-\frac{1}{2}\sum_{i}^{n}\sum_{j}^{n}(-2C_2)V_iV_j - \sum_{i}^{n}V_i(2C_2k)$$

This constraint term evaluates the facility neurons and we see that $T_{i,j}$ corresponds to $-2C_2$ and I_i corresponds to $2C_2K$. We add these to the functions to produce:

$$T_{i,j} = \begin{cases} -2C_2 & \text{if } i \text{ and } j \text{ are facility neurons} \\ 0 & \text{otherwise} \end{cases}$$

$$I_i = \begin{cases} 2C_2k & \text{if } i \text{ is a facility neuron} \\ -C_1D(i) & \text{if } i \text{ is a client-facility neuron} \end{cases}$$

The third constraint term (A.17) is:

$$\begin{aligned} &-\frac{1}{2}\sum_{i=n+1}^{2n}\sum_{j=n+1}^{2n}(-2C_3)V_iV_j - \sum_{i=n+1}^{2n}V_i(2C_3) \\ &-\frac{1}{2}\sum_{i=2n+1}^{3n}\sum_{j=2n+1}^{3n}(-2C_3)V_iV_j - \sum_{i=2n+1}^{3n}V_i(2C_3) \\ &- \quad \cdots \\ &-\frac{1}{2}\sum_{i=n^2+1}^{N}\sum_{j=n^2+1}^{N}(-2C_3)V_iV_j - \sum_{i=n^2+1}^{N}V_i(2C_3) \end{aligned}$$

This constraint has been expanded to several sub-constraint terms where each sub-constraint term refers to one group of client-facility neurons. Each group will correspond to a column of client-facility neurons in the matrix representation. The *connection term* for each sub-constraint term will evaluate only the neurons in a client-facility group and therefore each client-facility neuron will have a connection value of $-2C_3$ if they are in the same group. The *bias term* in each sub-constraint term also evaluates the neurons in a client-facility group. However, we can combine the individual *bias terms* and as a result each client-facility neuron has $2C_3$ added to its input bias. These terms are then added to the piecewise functions:

$$T_{i,j} = \begin{cases} -2C_2 & \text{if } i \text{ and } j \text{ are facility neurons} \\ -2C_3 & \text{if } i \text{ and } j \text{ are client-facility neurons in the same column} \\ 0 & \text{otherwise} \end{cases}$$

$$I_i = \begin{cases} 2C_2k & \text{if } i \text{ is a facility neuron} \\ -C_1D(i) + 2C_3 & \text{if } i \text{ is a client-facility neuron} \end{cases}$$

The fourth constraint term (A.23) is:

$$-\frac{1}{2}\sum_{i=n+1}^{N}\sum_{j=n+1}^{N}(-2C_4)V_iV_j - \sum_{i=n+1}^{N}V_i(2C_4n)$$

The *connection term* evaluates all client-facility neurons. This means that all connections between client-facility neurons have the value $-2C_4$. For the client-facility neurons in the same column, this value is subtracted from the previous $-2C_3$ value. Like the third constraint term, the *bias term* adds $2C_4n$ to all client-facility neurons and this is added to the previous values in our function. The piecewise functions are now:

$$T_{i,j} = \begin{cases} -2C_2 & \text{if } i \text{ and } j \text{ are facility neurons} \\ -2C_3 - 2C_4 & \text{if } i \text{ and } j \text{ are client-facility neurons in the same column} \\ -2C_4 & \text{if } i \text{ and } j \text{ are client-facility neurons in different columns} \\ 0 & \text{otherwise} \end{cases}$$

$$I_i = \begin{cases} 2C_2k & \text{if } i \text{ is a facility neuron} \\ -C_1D(i) + 2C_3 + 2C_4n & \text{if } i \text{ is a client-facility neuron} \end{cases}$$

The fifth constraint term (A.30) is:

$$-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(2C_5)V_iV_{nj+i} - \sum_{i=n+1}^{N}V_i(-C_5)$$

The *connection term* only evaluates neuron pairs where one neuron is a facility neuron and the second neuron is a client-facility neuron that can potentially be served by that facility neuron. These connections will have a value of $2C_5$. The *bias term* evaluates only the client-facility neurons and $-C_5$ will be added to the input bias.

The final connection values are defined as:

$$T_{i,j} = \begin{cases} -2C_2 & \text{if } i \text{ and } j \text{ are facility neurons} \\ 2C_5 & \text{if } i \text{ is a facility neuron and } j \text{ is a client-facility neuron served by } i \\ -2C_3 - 2C_4 & \text{if } i \text{ and } j \text{ are client-facility neurons in the same column} \\ -2C_4 & \text{if } i \text{ and } j \text{ are client-facility neurons in different columns} \\ 0 & \text{otherwise} \end{cases} \tag{A.31}$$

The final input bias values are defined as:

$$I_i = \begin{cases} 2C_2k & \text{if } i \text{ is a facility neuron} \\ -C_1D(i) + 2C_3 + 2C_4n - C_5 & \text{if } i \text{ is a client-facility neuron} \end{cases} \tag{A.32}$$

Appendix B

Full Dataset Test Results

The following sections contain test results for our modified Hopfield network, Haralampiev's competition-based neural network, and Arya et al.'s local search algorithm. We used our modified Hopfield network from Section 5 and included the extensions described in Section 5.3. We report data for 1, 4, and 10 runs of our modified Hopfield network: The 1 run results are to demonstrate how well our network can produce an initial approximation, the 4 run results are for comparing with Haralampiev's network which was also allotted 4 runs, and the 10 run results are to show how well our approach can improve on an initial approximation.

Haralampiev's network functions as described in Section 6.1. We report data for 1 and 4 runs of the network in order to show the initial approximation and how multiple runs can improve the solution. We use 4 runs as it is Haralampiev's suggestion for getting a good solution from the network. The network will use $temperature = 1$ and $\alpha = 0.1$ for each test as we found these values to produce good solutions while stabilizing quickly. We do not present test results for the TSPLib dataset as Haralampiev's network did not finish despite being given several days of computation time.

The local search algorithm works as described in Section 6.2. We allowed a maximum runtime of 1, 5, or 15 seconds. For a fair comparison we attempt to match the local search runtime with our modified Hopfield network runtime and report the matched times. For example, the local search algorithm finishes all random-small tests in less then a second so we do not report the 5 and 15 second results. Conversely, the difficult TSPLib tests result in the local search algorithm using the full 15 seconds and for those tests we report all three times.

We report several statistics for the algorithms. The first is the approximation ratio which we calculate using S/S^* where S is the value of the proposed solution and S^* is the value of the optimal solution. The second statistic is the total runtime of the algorithm in seconds where the timer starts when the algorithm begins initialization and ends when a solution is returned.

When there are multiple tests for a single n and k pair we return the average approximation ratio and average runtime values for the n and k pair. In this case we also calculate the standard deviation for the approximation ratios in order to determine how consistent an algorithms approximation ratio is.

B.1 Random-Small Dataset

The random-small dataset was constructed by randomly selecting points on a 2D plane. Each point i is a potential location and was assigned a randomly generated coordinate pair (x_i, y_i) where x_i and y_i can take any value between 0 and 1.

Each individual test was constructed using a n and k pair. Each test generates n points and solves the k-median problem for k. The n value for a test pair can be one of $[20, 30, 40, 50, 60, 70, 80, 90, 100]$ and the k value can be one of $[2, 3, 4, 5, 6, 7, 8, 9, 10]$. Each n and k pair has 100 tests.

For each n and k pair we report the average approximation ratio of the 100 tests, the standard deviation of the approximation ratios, and the average runtime in seconds. These values are denoted in the results as A. Ratio, σ, and A. Time, respectively.

B.1.1 Hopfield Network and Haralampiev Network Results

N	K	Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
20	2	1.091	0.067	0.011	1.048	0.047	0.042	1.04	0.069	0.009	1.007	0.021	0.034
	3	1.129	0.105	0.012	1.064	0.059	0.042	1.065	0.075	0.016	1.014	0.024	0.059
	4	1.125	0.092	0.012	1.077	0.073	0.043	1.087	0.099	0.022	1.02	0.035	0.081
	5	1.146	0.094	0.012	1.086	0.07	0.043	1.137	0.137	0.028	1.052	0.059	0.103
	6	1.174	0.116	0.011	1.113	0.087	0.043	1.171	0.112	0.032	1.062	0.058	0.127
	7	1.168	0.123	0.011	1.124	0.099	0.044	1.202	0.134	0.037	1.093	0.082	0.148
	8	1.189	0.109	0.011	1.131	0.079	0.044	1.217	0.156	0.041	1.116	0.082	0.165
	9	1.261	0.139	0.011	1.186	0.124	0.044	1.299	0.18	0.047	1.158	0.101	0.187
	10	1.347	0.184	0.011	1.242	0.167	0.043	1.336	0.173	0.052	1.19	0.107	0.204
30	2	1.099	0.078	0.018	1.054	0.044	0.062	1.034	0.051	0.015	1.006	0.016	0.059
	3	1.117	0.087	0.018	1.071	0.055	0.064	1.051	0.063	0.027	1.015	0.03	0.103
	4	1.129	0.102	0.018	1.072	0.061	0.064	1.062	0.081	0.04	1.021	0.029	0.151
	5	1.122	0.072	0.018	1.078	0.051	0.067	1.079	0.078	0.052	1.031	0.036	0.19
	6	1.135	0.073	0.018	1.085	0.052	0.065	1.11	0.096	0.06	1.034	0.032	0.232
	7	1.141	0.085	0.018	1.103	0.063	0.066	1.14	0.094	0.07	1.057	0.051	0.27
	8	1.153	0.091	0.018	1.111	0.067	0.065	1.165	0.122	0.073	1.073	0.05	0.303
	9	1.179	0.099	0.017	1.127	0.073	0.064	1.182	0.122	0.086	1.086	0.06	0.345
	10	1.168	0.086	0.018	1.115	0.06	0.065	1.187	0.096	0.098	1.099	0.059	0.38
40	2	1.09	0.067	0.024	1.05	0.041	0.08	1.028	0.04	0.025	1.006	0.013	0.089
	3	1.131	0.074	0.023	1.061	0.04	0.083	1.043	0.055	0.043	1.011	0.024	0.161
	4	1.103	0.066	0.024	1.069	0.041	0.086	1.049	0.059	0.062	1.015	0.024	0.223
	5	1.128	0.066	0.023	1.078	0.05	0.088	1.061	0.065	0.083	1.022	0.03	0.292
	6	1.126	0.067	0.023	1.089	0.048	0.085	1.107	0.074	0.095	1.036	0.039	0.358
	7	1.124	0.074	0.023	1.091	0.049	0.085	1.109	0.082	0.107	1.035	0.032	0.426
	8	1.142	0.063	0.023	1.091	0.05	0.085	1.111	0.077	0.126	1.056	0.034	0.482
	9	1.14	0.073	0.023	1.098	0.049	0.086	1.137	0.088	0.136	1.062	0.037	0.526
	10	1.142	0.065	0.023	1.103	0.053	0.087	1.158	0.122	0.151	1.072	0.042	0.584
50	2	1.103	0.067	0.028	1.051	0.038	0.097	1.023	0.04	0.036	1.006	0.014	0.125

		Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	3	1.116	0.073	0.029	1.069	0.044	0.101	1.033	0.043	0.062	1.01	0.017	0.226
	4	1.104	0.063	0.03	1.068	0.041	0.105	1.055	0.058	0.092	1.014	0.022	0.318
	5	1.121	0.064	0.03	1.077	0.043	0.107	1.055	0.045	0.111	1.022	0.026	0.41
	6	1.119	0.064	0.029	1.075	0.034	0.107	1.075	0.065	0.131	1.027	0.031	0.503
	7	1.134	0.067	0.029	1.091	0.044	0.108	1.089	0.059	0.153	1.038	0.038	0.592
	8	1.13	0.053	0.029	1.089	0.04	0.11	1.107	0.073	0.169	1.04	0.035	0.666
	9	1.13	0.06	0.03	1.091	0.046	0.107	1.122	0.085	0.189	1.053	0.032	0.742
	10	1.134	0.061	0.029	1.103	0.049	0.107	1.135	0.071	0.202	1.061	0.035	0.81
60	2	1.094	0.061	0.034	1.05	0.035	0.117	1.02	0.034	0.05	1.007	0.021	0.163
	3	1.126	0.08	0.035	1.072	0.045	0.12	1.028	0.039	0.084	1.013	0.029	0.297
	4	1.124	0.071	0.034	1.078	0.044	0.122	1.044	0.047	0.108	1.009	0.017	0.394
	5	1.112	0.057	0.035	1.075	0.041	0.124	1.063	0.054	0.144	1.019	0.025	0.569
	6	1.118	0.053	0.036	1.081	0.037	0.128	1.065	0.053	0.165	1.022	0.022	0.639
	7	1.118	0.058	0.036	1.083	0.044	0.127	1.083	0.07	0.198	1.032	0.026	0.77
	8	1.121	0.057	0.034	1.083	0.034	0.127	1.082	0.059	0.224	1.037	0.032	0.874
	9	1.129	0.066	0.035	1.092	0.045	0.131	1.105	0.061	0.254	1.047	0.034	0.976
	10	1.132	0.058	0.035	1.091	0.039	0.131	1.108	0.07	0.278	1.057	0.032	1.073
70	2	1.101	0.069	0.039	1.05	0.037	0.132	1.017	0.022	0.067	1.006	0.014	0.226
	3	1.109	0.064	0.041	1.066	0.039	0.137	1.027	0.04	0.113	1.008	0.019	0.382
	4	1.121	0.059	0.041	1.075	0.043	0.139	1.045	0.038	0.174	1.015	0.023	0.542
	5	1.118	0.059	0.04	1.07	0.032	0.145	1.054	0.047	0.184	1.018	0.023	0.686
	6	1.114	0.055	0.041	1.081	0.04	0.146	1.058	0.05	0.235	1.021	0.024	0.822
	7	1.108	0.048	0.042	1.079	0.033	0.152	1.065	0.047	0.251	1.025	0.022	0.977
	8	1.111	0.051	0.043	1.078	0.031	0.151	1.073	0.046	0.291	1.038	0.024	1.046
	9	1.117	0.044	0.042	1.089	0.038	0.149	1.084	0.062	0.323	1.034	0.026	1.247
	10	1.114	0.053	0.041	1.088	0.035	0.15	1.096	0.06	0.375	1.043	0.024	1.322
80	2	1.096	0.07	0.045	1.05	0.031	0.154	1.024	0.039	0.078	1.007	0.02	0.293
	3	1.107	0.063	0.046	1.066	0.034	0.156	1.03	0.05	0.134	1.009	0.015	0.534

N	K	Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	4	1.127	0.068	0.047	1.081	0.045	0.158	1.035	0.04	0.185	1.015	0.022	0.692
	5	1.111	0.06	0.047	1.073	0.034	0.166	1.046	0.042	0.239	1.013	0.02	0.836
	6	1.11	0.047	0.048	1.079	0.032	0.166	1.056	0.044	0.294	1.017	0.019	1.005
	7	1.11	0.051	0.048	1.082	0.031	0.168	1.062	0.05	0.311	1.025	0.024	1.167
	8	1.125	0.054	0.049	1.079	0.033	0.172	1.071	0.051	0.339	1.032	0.026	1.319
	9	1.111	0.048	0.046	1.094	0.035	0.169	1.083	0.046	0.38	1.036	0.025	1.473
	10	1.116	0.052	0.048	1.089	0.037	0.175	1.093	0.051	0.423	1.04	0.024	1.573
90	2	1.096	0.061	0.049	1.049	0.032	0.169	1.021	0.032	0.105	1.006	0.014	0.348
	3	1.108	0.068	0.052	1.068	0.036	0.176	1.024	0.035	0.175	1.005	0.01	0.631
	4	1.108	0.062	0.053	1.071	0.039	0.18	1.04	0.042	0.232	1.011	0.016	0.837
	5	1.108	0.055	0.053	1.078	0.04	0.184	1.036	0.037	0.312	1.015	0.021	0.973
	6	1.104	0.051	0.053	1.08	0.034	0.187	1.048	0.041	0.371	1.016	0.018	1.233
	7	1.113	0.045	0.055	1.075	0.029	0.188	1.059	0.039	0.389	1.021	0.017	1.411
	8	1.108	0.046	0.053	1.08	0.032	0.19	1.062	0.045	0.444	1.026	0.019	1.55
	9	1.12	0.047	0.055	1.088	0.035	0.192	1.072	0.047	0.477	1.035	0.023	1.787
	10	1.117	0.045	0.054	1.09	0.031	0.194	1.082	0.047	0.509	1.04	0.023	1.922
100	2	1.09	0.055	0.057	1.045	0.03	0.189	1.022	0.028	0.124	1.005	0.011	0.45
	3	1.11	0.062	0.058	1.071	0.039	0.193	1.021	0.03	0.225	1.005	0.01	0.727
	4	1.121	0.054	0.058	1.075	0.035	0.198	1.029	0.03	0.271	1.009	0.016	1.014
	5	1.113	0.056	0.058	1.078	0.036	0.209	1.037	0.034	0.345	1.013	0.014	1.125
	6	1.116	0.046	0.06	1.08	0.036	0.206	1.044	0.041	0.44	1.014	0.018	1.451
	7	1.116	0.048	0.06	1.08	0.03	0.209	1.048	0.037	0.452	1.016	0.017	1.682
	8	1.115	0.045	0.059	1.08	0.029	0.211	1.066	0.049	0.513	1.023	0.016	1.867
	9	1.124	0.046	0.061	1.084	0.03	0.211	1.069	0.045	0.538	1.03	0.02	2.035
	10	1.11	0.04	0.061	1.087	0.028	0.217	1.077	0.046	0.584	1.034	0.019	2.262

B.1.2 Hopfield Network and Local Search Results

N	K	Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
20	2	1.091	0.067	0.011	1.027	0.039	0.099	1.036	0.033	0.004
	3	1.129	0.105	0.012	1.041	0.043	0.102	1.048	0.041	0.005
	4	1.125	0.092	0.012	1.051	0.047	0.105	1.065	0.045	0.007
	5	1.146	0.094	0.012	1.063	0.056	0.107	1.066	0.046	0.009
	6	1.174	0.116	0.011	1.083	0.064	0.106	1.074	0.053	0.01
	7	1.168	0.123	0.011	1.099	0.08	0.107	1.066	0.047	0.011
	8	1.189	0.109	0.011	1.111	0.076	0.108	1.056	0.042	0.013
	9	1.261	0.139	0.011	1.15	0.1	0.108	1.055	0.044	0.014
	10	1.347	0.184	0.011	1.214	0.131	0.109	1.055	0.044	0.013
30	2	1.099	0.078	0.018	1.029	0.033	0.137	1.024	0.024	0.006
	3	1.117	0.087	0.018	1.042	0.042	0.151	1.038	0.037	0.009
	4	1.129	0.102	0.018	1.053	0.043	0.151	1.047	0.036	0.012
	5	1.122	0.072	0.018	1.056	0.042	0.16	1.053	0.034	0.015
	6	1.135	0.073	0.018	1.069	0.048	0.158	1.058	0.035	0.018
	7	1.141	0.085	0.018	1.08	0.058	0.162	1.06	0.036	0.022
	8	1.153	0.091	0.018	1.092	0.06	0.158	1.064	0.036	0.025
	9	1.179	0.099	0.017	1.104	0.058	0.159	1.065	0.036	0.028
	10	1.168	0.086	0.018	1.092	0.056	0.161	1.067	0.041	0.03
40	2	1.09	0.067	0.024	1.032	0.03	0.179	1.021	0.025	0.008
	3	1.131	0.074	0.023	1.048	0.03	0.185	1.033	0.027	0.013
	4	1.103	0.066	0.024	1.05	0.031	0.206	1.036	0.031	0.018
	5	1.128	0.066	0.023	1.058	0.037	0.208	1.043	0.026	0.023
	6	1.126	0.067	0.023	1.073	0.047	0.211	1.048	0.03	0.028
	7	1.124	0.074	0.023	1.077	0.045	0.206	1.051	0.028	0.033
	8	1.142	0.063	0.023	1.072	0.043	0.21	1.053	0.028	0.04
	9	1.14	0.073	0.023	1.081	0.038	0.217	1.061	0.032	0.044
	10	1.142	0.065	0.023	1.082	0.045	0.219	1.064	0.029	0.05
50	2	1.103	0.067	0.028	1.03	0.026	0.222	1.018	0.019	0.011

N	K	Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	3	1.116	0.073	0.029	1.046	0.034	0.236	1.024	0.022	0.017
	4	1.104	0.063	0.03	1.05	0.028	0.245	1.029	0.02	0.022
	5	1.121	0.064	0.03	1.059	0.031	0.252	1.039	0.027	0.031
	6	1.119	0.064	0.029	1.067	0.039	0.25	1.038	0.023	0.039
	7	1.134	0.067	0.029	1.074	0.038	0.255	1.052	0.026	0.047
	8	1.13	0.053	0.029	1.073	0.031	0.265	1.049	0.024	0.057
	9	1.13	0.06	0.03	1.076	0.041	0.257	1.049	0.025	0.063
	10	1.134	0.061	0.029	1.084	0.04	0.265	1.057	0.031	0.071
60	2	1.094	0.061	0.034	1.036	0.026	0.259	1.02	0.025	0.014
	3	1.126	0.08	0.035	1.045	0.03	0.279	1.019	0.021	0.022
	4	1.124	0.071	0.034	1.06	0.039	0.281	1.033	0.031	0.032
	5	1.112	0.057	0.035	1.054	0.033	0.296	1.032	0.023	0.042
	6	1.118	0.053	0.036	1.062	0.03	0.306	1.032	0.019	0.051
	7	1.118	0.058	0.036	1.068	0.035	0.303	1.044	0.022	0.061
	8	1.121	0.057	0.034	1.069	0.025	0.302	1.046	0.025	0.074
	9	1.129	0.066	0.035	1.074	0.035	0.309	1.047	0.023	0.086
	10	1.132	0.058	0.035	1.085	0.035	0.322	1.047	0.022	0.097
70	2	1.101	0.069	0.039	1.033	0.022	0.286	1.015	0.021	0.018
	3	1.109	0.064	0.041	1.05	0.033	0.318	1.02	0.019	0.027
	4	1.121	0.059	0.041	1.057	0.03	0.319	1.026	0.024	0.037
	5	1.118	0.059	0.04	1.058	0.028	0.331	1.027	0.017	0.05
	6	1.114	0.055	0.041	1.064	0.03	0.335	1.035	0.025	0.062
	7	1.108	0.048	0.042	1.067	0.028	0.36	1.035	0.019	0.077
	8	1.111	0.051	0.043	1.066	0.027	0.357	1.042	0.021	0.087
	9	1.117	0.044	0.042	1.075	0.031	0.354	1.046	0.024	0.109
	10	1.114	0.053	0.041	1.074	0.032	0.365	1.048	0.022	0.115
80	2	1.096	0.07	0.045	1.034	0.026	0.339	1.012	0.016	0.019
	3	1.107	0.063	0.046	1.044	0.026	0.353	1.017	0.016	0.032

N	K	Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	4	1.127	0.068	0.047	1.055	0.03	0.371	1.023	0.018	0.047
	5	1.111	0.06	0.047	1.054	0.022	0.384	1.028	0.019	0.061
	6	1.11	0.047	0.048	1.062	0.026	0.388	1.03	0.02	0.076
	7	1.11	0.051	0.048	1.063	0.026	0.388	1.034	0.02	0.093
	8	1.125	0.054	0.049	1.07	0.026	0.416	1.038	0.02	0.114
	9	1.111	0.048	0.046	1.073	0.029	0.403	1.041	0.017	0.129
	10	1.116	0.052	0.048	1.074	0.03	0.42	1.042	0.018	0.146
90	2	1.096	0.061	0.049	1.034	0.024	0.378	1.012	0.016	0.024
	3	1.108	0.068	0.052	1.049	0.032	0.401	1.013	0.014	0.041
	4	1.108	0.062	0.053	1.052	0.027	0.401	1.02	0.017	0.056
	5	1.108	0.055	0.053	1.058	0.028	0.424	1.024	0.017	0.074
	6	1.104	0.051	0.053	1.058	0.025	0.425	1.032	0.022	0.091
	7	1.113	0.045	0.055	1.065	0.026	0.44	1.034	0.02	0.106
	8	1.108	0.046	0.053	1.065	0.023	0.446	1.034	0.019	0.137
	9	1.12	0.047	0.055	1.07	0.028	0.464	1.037	0.021	0.159
	10	1.117	0.045	0.054	1.076	0.025	0.461	1.04	0.017	0.17
100	2	1.09	0.055	0.057	1.034	0.024	0.421	1.013	0.014	0.028
	3	1.11	0.062	0.058	1.05	0.03	0.442	1.012	0.015	0.049
	4	1.121	0.054	0.058	1.056	0.026	0.453	1.017	0.014	0.071
	5	1.113	0.056	0.058	1.057	0.026	0.474	1.021	0.016	0.089
	6	1.116	0.046	0.06	1.059	0.024	0.47	1.025	0.017	0.114
	7	1.116	0.048	0.06	1.061	0.026	0.486	1.029	0.017	0.137
	8	1.115	0.045	0.059	1.067	0.025	0.5	1.034	0.021	0.16
	9	1.124	0.046	0.061	1.078	0.029	0.505	1.035	0.019	0.181
	10	1.11	0.04	0.061	1.069	0.026	0.515	1.041	0.018	0.204

B.2 Random-Large Dataset

The random-large dataset was constructed by randomly selecting points on a 2D plane. Each point i is a potential location and was assigned a randomly generated coordinate pair (x_i, y_i) where x_i and y_i can take any value between 0 and 1.

Each individual test was constructed using a n and k pair. Each test generates n points and solves the k-median problem for k. The n value can be one of $[500, 600, 700, 800]$ and the k value can be one of $[20, 30, 40, 50]$. Each n and k pair has 3 tests.

For each n and k pair we report the average approximation ratio of the 3 tests, the standard deviation of the approximation ratios, and the average runtime in seconds. These values are denoted in the results as A. Ratio, σ, and A. Time, respectively.

B.2.1 Hopfield Network and Haralampiev Network Results

		Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
500	20	1.109	0.021	0.407	1.075	0.014	1.269	1.037	0.006	17.702	1.023	0.008	73.033
	30	1.082	0.005	0.402	1.089	0.013	1.223	1.065	0.036	17.465	1.038	0.005	73.473
	40	1.09	0.012	0.383	1.083	0.007	1.312	1.077	0.018	20.468	1.068	0.017	86.171
	50	1.116	0.004	0.385	1.118	0.008	1.319	1.082	0.013	24.804	1.066	0.012	91.565
600	20	1.091	0.014	0.477	1.087	0.006	1.661	1.03	0.01	28.908	1.018	0.006	129.879
	30	1.106	0.01	0.509	1.083	0.006	1.599	1.054	0.018	28.039	1.039	0.003	105.746
	40	1.097	0.012	0.491	1.083	0.003	1.717	1.067	0.014	34.081	1.05	0.014	112.783
	50	1.104	0.008	0.518	1.088	0.021	1.668	1.073	0.011	41.416	1.06	0.001	126.269
700	20	1.103	0.009	0.563	1.086	0.008	1.862	1.031	0.023	81.832	1.02	0.012	152.162
	30	1.099	0.005	0.625	1.09	0.011	1.799	1.041	0.012	44.993	1.038	0.01	129.504
	40	1.102	0.025	0.619	1.096	0.014	1.826	1.054	0.009	40.647	1.039	0.007	187.047
	50	1.107	0.003	0.712	1.102	0.005	1.85	1.1	0.01	47.741	1.047	0.008	167.765
800	20	1.092	0.016	0.79	1.077	0.003	2.31	1.025	0.003	50.343	1.012	0.008	660.428
	30	1.103	0.011	0.769	1.077	0.021	2.381	1.043	0.017	34.805	1.024	0.003	224.722
	40	1.112	0.006	0.711	1.085	0.018	2.727	1.073	0.021	50.203	1.039	0.011	241.097
	50	1.11	0.006	0.78	1.087	0.005	2.776	1.055	0.016	57.658	1.047	0.009	235.132

B.2.2 Hopfield Network and Local Search Results

		Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1s)			Local Search (5s)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
500	20	1.109	0.021	0.407	1.075	0.012	2.937	1.087	0.008	1	1.044	0.003	5
	30	1.082	0.005	0.402	1.083	0.008	3.231	1.162	0.014	1	1.075	0.011	5
	40	1.09	0.012	0.383	1.088	0.002	3.124	1.25	0.084	1	1.12	0.029	5
	50	1.116	0.004	0.385	1.105	0.008	3.251	1.207	0.012	1	1.121	0.006	5
600	20	1.091	0.014	0.477	1.074	0.01	3.97	1.101	0.009	1.001	1.038	0.008	5
	30	1.106	0.01	0.509	1.094	0.006	3.978	1.141	0.027	1	1.077	0.005	5
	40	1.097	0.012	0.491	1.082	0.003	4.072	1.181	0.01	1	1.102	0.007	5
	50	1.104	0.008	0.518	1.082	0.007	4.103	1.243	0.021	1	1.139	0.003	5
700	20	1.103	0.009	0.563	1.08	0.001	4.735	1.113	0.023	1	1.062	0.006	5
	30	1.099	0.005	0.625	1.076	0.014	4.866	1.159	0.025	1	1.079	0.022	5.001
	40	1.102	0.025	0.619	1.08	0.003	4.776	1.252	0.028	1.001	1.149	0.028	5.001
	50	1.107	0.003	0.712	1.081	0.009	4.947	1.266	0.09	1.001	1.155	0.047	5.001
800	20	1.092	0.016	0.79	1.066	0.005	5.594	1.11	0.021	1.001	1.053	0.013	5.001
	30	1.103	0.011	0.769	1.08	0.008	5.619	1.181	0.01	1.001	1.103	0.015	5.001
	40	1.112	0.006	0.711	1.082	0.006	5.925	1.257	0.008	1.001	1.148	0.008	5.001
	50	1.11	0.006	0.78	1.08	0.007	6.407	1.234	0.043	1.001	1.177	0.035	5.001

B.3 Cities USCA312 Dataset Results

The Cities USCA312 Dataset uses cities located across North America as the potential facility locations. We used John Burkardt's USCA312 dataset [6] which provides a list of 312 cities and their location information. The dataset converts a city's longitude and latitude into an equivalent x and y coordinate pair which we used for our calculations.

We used this data to construct several tests. For each n and k pair we randomly selected n locations and solved the resulting k-median problem. The n value can be any increment of 10 between 30 and 300 and our k value can be between 2 and 10. Each n and k pair has 100 tests.

For each n and k pair we report the average approximation ratio of the 100 tests, the standard deviation of the approximation ratios, and the average runtime in seconds. These values are denoted in the results as A. Ratio, σ, and A. Time, respectively.

B.3.1 Hopfield Network and Haralampiev Network Results

N	K	Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
30	2	1.152	0.151	0.018	1.081	0.079	0.06	1.042	0.095	0.012	1.032	0.095	0.023
	3	1.232	0.196	0.018	1.125	0.095	0.063	1.079	0.088	0.023	1.056	0.069	0.037
	4	1.217	0.144	0.018	1.167	0.133	0.063	1.144	0.169	0.028	1.116	0.113	0.049
	5	1.297	0.193	0.018	1.231	0.181	0.065	1.123	0.11	0.034	1.122	0.136	0.057
	6	1.394	0.24	0.018	1.291	0.154	0.063	1.236	0.182	0.037	1.214	0.176	0.06
	7	1.511	0.261	0.017	1.447	0.232	0.065	1.416	0.278	0.042	1.388	0.27	0.068
	8	1.356	0.21	0.017	1.289	0.182	0.063	1.312	0.206	0.038	1.272	0.186	0.073
	9	1.651	0.275	0.017	1.57	0.266	0.064	1.471	0.258	0.044	1.448	0.266	0.076
	10	1.833	0.449	0.017	1.756	0.441	0.063	1.569	0.391	0.044	1.566	0.354	0.082
40	2	1.137	0.14	0.023	1.082	0.099	0.082	1.034	0.109	0.018	1.027	0.102	0.03
	3	1.117	0.067	0.023	1.073	0.041	0.081	1.047	0.046	0.043	1.051	0.048	0.058
	4	1.13	0.105	0.023	1.079	0.058	0.086	1.073	0.066	0.047	1.056	0.058	0.078
	5	1.338	0.201	0.024	1.226	0.146	0.084	1.161	0.13	0.058	1.183	0.188	0.087
	6	1.293	0.121	0.024	1.247	0.114	0.086	1.182	0.13	0.061	1.158	0.119	0.105
	7	1.412	0.098	0.024	1.373	0.097	0.084	1.262	0.148	0.076	1.267	0.128	0.107
	8	1.23	0.098	0.023	1.188	0.085	0.087	1.183	0.11	0.073	1.181	0.124	0.117
	9	1.55	0.163	0.022	1.497	0.139	0.085	1.411	0.163	0.088	1.406	0.194	0.132
	10	1.587	0.267	0.023	1.523	0.225	0.086	1.424	0.229	0.077	1.389	0.213	0.144
50	2	1.149	0.113	0.029	1.081	0.058	0.096	1.025	0.046	0.029	1.025	0.05	0.05
	3	1.199	0.134	0.028	1.134	0.098	0.099	1.062	0.09	0.058	1.058	0.082	0.092
	4	1.216	0.129	0.029	1.141	0.108	0.099	1.071	0.082	0.072	1.068	0.091	0.11
	5	1.209	0.097	0.029	1.15	0.061	0.105	1.094	0.074	0.074	1.074	0.06	0.112
	6	1.183	0.093	0.029	1.113	0.051	0.106	1.081	0.052	0.118	1.085	0.075	0.15
	7	1.319	0.13	0.029	1.257	0.113	0.105	1.23	0.16	0.107	1.207	0.137	0.155
	8	1.231	0.084	0.03	1.175	0.067	0.107	1.15	0.092	0.105	1.155	0.087	0.173
	9	1.462	0.335	0.029	1.328	0.235	0.106	1.182	0.173	0.141	1.168	0.146	0.199
	10	1.408	0.114	0.029	1.362	0.114	0.104	1.328	0.15	0.128	1.309	0.155	0.206
60	2	1.17	0.131	0.034	1.098	0.097	0.111	1.033	0.07	0.04	1.022	0.05	0.07

		Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	3	1.217	0.175	0.035	1.123	0.112	0.118	1.033	0.043	0.081	1.027	0.039	0.115
	4	1.198	0.168	0.035	1.1	0.097	0.121	1.061	0.095	0.094	1.041	0.074	0.126
	5	1.171	0.123	0.036	1.093	0.051	0.124	1.066	0.074	0.132	1.057	0.069	0.184
	6	1.214	0.106	0.035	1.136	0.05	0.126	1.098	0.064	0.153	1.076	0.045	0.212
	7	1.277	0.083	0.035	1.228	0.061	0.126	1.153	0.082	0.172	1.15	0.088	0.223
	8	1.249	0.069	0.034	1.203	0.056	0.125	1.16	0.068	0.157	1.148	0.077	0.236
	9	1.32	0.075	0.033	1.268	0.065	0.124	1.218	0.091	0.176	1.204	0.091	0.272
	10	1.459	0.115	0.034	1.415	0.088	0.124	1.28	0.138	0.196	1.301	0.146	0.264
70	2	1.23	0.118	0.04	1.153	0.075	0.126	1.053	0.066	0.055	1.05	0.068	0.074
	3	1.182	0.164	0.04	1.102	0.108	0.134	1.017	0.046	0.093	1.014	0.028	0.132
	4	1.186	0.062	0.041	1.152	0.044	0.138	1.108	0.049	0.175	1.097	0.05	0.238
	5	1.227	0.113	0.042	1.161	0.097	0.143	1.077	0.051	0.162	1.081	0.073	0.215
	6	1.21	0.084	0.04	1.173	0.059	0.14	1.103	0.063	0.199	1.102	0.066	0.269
	7	1.23	0.071	0.04	1.194	0.063	0.141	1.155	0.086	0.269	1.15	0.092	0.282
	8	1.32	0.103	0.041	1.283	0.092	0.148	1.196	0.112	0.252	1.225	0.113	0.313
	9	1.243	0.107	0.041	1.206	0.082	0.143	1.146	0.081	0.248	1.13	0.081	0.34
	10	1.29	0.064	0.04	1.255	0.054	0.147	1.207	0.075	0.274	1.214	0.075	0.384
80	2	1.189	0.098	0.045	1.095	0.077	0.144	1.044	0.054	0.076	1.042	0.046	0.103
	3	1.265	0.087	0.045	1.195	0.065	0.15	1.057	0.072	0.124	1.045	0.062	0.191
	4	1.084	0.07	0.048	1.051	0.041	0.161	1.04	0.059	0.155	1.044	0.08	0.219
	5	1.095	0.051	0.045	1.067	0.04	0.165	1.06	0.052	0.214	1.056	0.047	0.279
	6	1.155	0.046	0.046	1.123	0.041	0.163	1.073	0.04	0.329	1.076	0.04	0.373
	7	1.147	0.073	0.046	1.107	0.043	0.165	1.068	0.057	0.249	1.064	0.061	0.353
	8	1.329	0.076	0.046	1.295	0.071	0.164	1.217	0.102	0.311	1.207	0.097	0.399
	9	1.327	0.119	0.048	1.298	0.118	0.167	1.214	0.128	0.312	1.196	0.129	0.41
	10	1.463	0.154	0.046	1.416	0.155	0.168	1.257	0.178	0.34	1.231	0.179	0.458
90	2	1.223	0.082	0.05	1.145	0.075	0.16	1.027	0.044	0.093	1.019	0.039	0.128
	3	1.283	0.199	0.051	1.15	0.134	0.171	1.049	0.093	0.127	1.024	0.068	0.183

N	K	Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	4	1.122	0.063	0.052	1.065	0.035	0.178	1.035	0.038	0.25	1.025	0.036	0.346
	5	1.113	0.048	0.052	1.09	0.039	0.186	1.061	0.049	0.314	1.069	0.053	0.36
	6	1.182	0.073	0.052	1.15	0.058	0.182	1.102	0.067	0.282	1.091	0.063	0.394
	7	1.185	0.068	0.054	1.148	0.053	0.185	1.08	0.053	0.33	1.087	0.057	0.384
	8	1.186	0.058	0.053	1.157	0.048	0.184	1.112	0.068	0.396	1.114	0.07	0.499
	9	1.269	0.095	0.052	1.22	0.081	0.189	1.146	0.085	0.425	1.142	0.085	0.528
	10	1.334	0.052	0.053	1.294	0.043	0.186	1.215	0.071	0.431	1.201	0.082	0.567
100	2	1.106	0.057	0.055	1.061	0.03	0.181	1.022	0.017	0.125	1.017	0.017	0.159
	3	1.254	0.121	0.057	1.138	0.075	0.185	1.024	0.035	0.182	1.026	0.049	0.266
	4	1.083	0.074	0.056	1.039	0.026	0.2	1.017	0.027	0.266	1.016	0.033	0.314
	5	1.142	0.117	0.058	1.082	0.058	0.199	1.049	0.059	0.315	1.048	0.065	0.416
	6	1.168	0.044	0.057	1.13	0.035	0.198	1.077	0.041	0.453	1.08	0.048	0.585
	7	1.188	0.065	0.057	1.151	0.046	0.207	1.107	0.059	0.432	1.102	0.051	0.539
	8	1.194	0.064	0.058	1.156	0.042	0.205	1.108	0.051	0.44	1.111	0.061	0.555
	9	1.213	0.055	0.061	1.18	0.043	0.206	1.151	0.066	0.499	1.142	0.068	0.682
	10	1.29	0.059	0.058	1.243	0.048	0.205	1.17	0.063	0.561	1.15	0.052	0.71
110	2	1.201	0.076	0.062	1.131	0.063	0.193	1.023	0.041	0.159	1.017	0.034	0.198
	3	1.204	0.105	0.063	1.134	0.078	0.202	1.021	0.042	0.275	1.014	0.028	0.343
	4	1.142	0.102	0.064	1.079	0.056	0.217	1.038	0.065	0.275	1.031	0.055	0.332
	5	1.115	0.059	0.064	1.079	0.033	0.225	1.049	0.051	0.334	1.04	0.033	0.519
	6	1.174	0.061	0.065	1.143	0.05	0.22	1.089	0.054	0.569	1.097	0.054	0.745
	7	1.162	0.053	0.064	1.132	0.044	0.225	1.09	0.065	0.569	1.088	0.068	0.732
	8	1.211	0.055	0.065	1.19	0.048	0.219	1.133	0.062	0.711	1.128	0.073	0.894
	9	1.192	0.069	0.066	1.158	0.045	0.226	1.12	0.067	0.655	1.121	0.062	0.836
	10	1.224	0.073	0.067	1.194	0.067	0.228	1.158	0.089	0.575	1.155	0.081	0.715
120	2	1.173	0.085	0.068	1.098	0.058	0.216	1.036	0.045	0.137	1.029	0.034	0.18
	3	1.157	0.117	0.069	1.09	0.06	0.225	1.012	0.015	0.258	1.006	0.01	0.335
	4	1.192	0.102	0.069	1.114	0.083	0.225	1.029	0.039	0.433	1.026	0.035	0.445

		Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	5	1.123	0.074	0.074	1.081	0.041	0.236	1.03	0.029	0.471	1.027	0.031	0.608
	6	1.17	0.049	0.072	1.134	0.038	0.24	1.069	0.033	0.729	1.067	0.032	0.795
	7	1.206	0.083	0.07	1.148	0.062	0.237	1.08	0.063	0.552	1.078	0.058	0.751
	8	1.308	0.048	0.072	1.265	0.036	0.243	1.155	0.105	0.551	1.15	0.101	0.716
	9	1.212	0.067	0.07	1.179	0.056	0.246	1.119	0.062	0.719	1.117	0.073	0.825
	10	1.282	0.118	0.071	1.239	0.107	0.25	1.167	0.087	0.704	1.158	0.085	0.967
130	2	1.199	0.085	0.078	1.137	0.077	0.241	1.021	0.032	0.18	1.012	0.025	0.233
	3	1.179	0.149	0.078	1.108	0.094	0.252	1.01	0.023	0.27	1.013	0.029	0.326
	4	1.108	0.068	0.078	1.071	0.038	0.265	1.027	0.018	0.384	1.03	0.034	0.541
	5	1.119	0.047	0.08	1.084	0.027	0.275	1.045	0.022	0.771	1.048	0.029	0.901
	6	1.15	0.052	0.081	1.114	0.044	0.277	1.055	0.035	0.506	1.048	0.032	0.759
	7	1.216	0.073	0.079	1.18	0.055	0.268	1.113	0.055	0.846	1.104	0.05	1.013
	8	1.203	0.054	0.08	1.165	0.036	0.271	1.1	0.056	0.836	1.091	0.048	1.018
	9	1.186	0.054	0.083	1.148	0.033	0.27	1.087	0.047	0.825	1.093	0.051	0.988
	10	1.287	0.068	0.08	1.249	0.058	0.277	1.162	0.081	1.052	1.158	0.069	1.089
140	2	1.233	0.149	0.084	1.144	0.102	0.258	1.011	0.023	0.147	1.005	0.014	0.199
	3	1.238	0.113	0.084	1.142	0.087	0.264	1.008	0.011	0.328	1.007	0.015	0.462
	4	1.135	0.101	0.084	1.064	0.035	0.282	1.023	0.026	0.426	1.022	0.022	0.582
	5	1.142	0.063	0.086	1.105	0.044	0.292	1.052	0.039	0.595	1.049	0.034	0.737
	6	1.136	0.054	0.087	1.109	0.04	0.285	1.056	0.039	0.846	1.051	0.035	1.19
	7	1.157	0.045	0.089	1.129	0.026	0.297	1.072	0.032	0.838	1.078	0.035	1.052
	8	1.136	0.048	0.087	1.106	0.034	0.29	1.066	0.032	1.128	1.068	0.038	1.317
	9	1.156	0.053	0.087	1.127	0.044	0.297	1.059	0.03	0.935	1.062	0.039	1.148
	10	1.318	0.066	0.088	1.281	0.06	0.285	1.16	0.081	0.978	1.144	0.079	1.328
150	2	1.165	0.087	0.088	1.097	0.054	0.275	1.017	0.022	0.281	1.01	0.017	0.309
	3	1.234	0.094	0.09	1.173	0.057	0.285	1.046	0.055	0.449	1.036	0.051	0.555
	4	1.165	0.147	0.091	1.074	0.057	0.299	1.022	0.033	0.915	1.019	0.028	1.097
	5	1.119	0.055	0.092	1.081	0.024	0.319	1.025	0.026	0.668	1.03	0.03	0.735

N	K	Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	6	1.143	0.037	0.094	1.116	0.032	0.315	1.059	0.04	0.82	1.054	0.037	1.002
	7	1.166	0.061	0.092	1.132	0.041	0.306	1.072	0.05	0.978	1.08	0.048	1.212
	8	1.213	0.052	0.095	1.178	0.037	0.311	1.114	0.049	1.015	1.101	0.042	1.307
	9	1.172	0.042	0.096	1.141	0.032	0.322	1.094	0.041	1.303	1.092	0.048	1.556
	10	1.221	0.046	0.095	1.197	0.036	0.32	1.132	0.053	0.922	1.115	0.061	1.214
160	2	1.158	0.081	0.098	1.101	0.058	0.288	1.01	0.009	0.24	1.006	0.008	0.285
	3	1.24	0.137	0.097	1.123	0.083	0.305	1.009	0.023	0.334	1.004	0.006	0.44
	4	1.102	0.078	0.096	1.059	0.037	0.332	1.024	0.03	0.572	1.02	0.027	0.746
	5	1.125	0.081	0.099	1.081	0.045	0.334	1.026	0.023	0.878	1.021	0.024	1.219
	6	1.174	0.059	0.099	1.141	0.043	0.328	1.073	0.037	1.149	1.074	0.04	1.386
	7	1.178	0.07	0.102	1.149	0.057	0.332	1.097	0.047	1.314	1.092	0.049	1.58
	8	1.29	0.047	0.099	1.255	0.032	0.334	1.166	0.064	1.255	1.156	0.06	1.435
	9	1.219	0.049	0.101	1.184	0.035	0.331	1.125	0.062	1.287	1.122	0.05	1.67
	10	1.199	0.051	0.099	1.162	0.044	0.343	1.108	0.059	1.305	1.11	0.058	1.558
170	2	1.229	0.099	0.099	1.108	0.073	0.312	1.01	0.019	0.207	1.005	0.012	0.29
	3	1.197	0.127	0.104	1.099	0.069	0.32	1.007	0.009	0.515	1.008	0.018	0.599
	4	1.153	0.131	0.103	1.079	0.071	0.336	1.026	0.031	1.103	1.019	0.021	1.642
	5	1.101	0.056	0.106	1.061	0.027	0.356	1.023	0.03	0.963	1.015	0.021	1.215
	6	1.131	0.048	0.106	1.095	0.025	0.348	1.036	0.026	1.027	1.037	0.024	1.465
	7	1.168	0.058	0.105	1.135	0.038	0.352	1.084	0.039	1.303	1.082	0.039	1.546
	8	1.185	0.052	0.108	1.146	0.035	0.351	1.088	0.042	1.473	1.089	0.046	1.669
	9	1.233	0.052	0.109	1.207	0.031	0.351	1.122	0.043	1.608	1.123	0.049	1.856
	10	1.26	0.058	0.11	1.227	0.045	0.359	1.157	0.056	1.555	1.134	0.057	2.268
180	2	1.257	0.105	0.108	1.17	0.096	0.333	1.006	0.036	0.206	1.002	0.003	0.274
	3	1.246	0.081	0.109	1.165	0.074	0.338	1.005	0.015	0.564	1.008	0.026	0.71
	4	1.095	0.054	0.111	1.062	0.024	0.363	1.014	0.024	0.743	1.022	0.036	1.085
	5	1.112	0.072	0.112	1.072	0.029	0.374	1.022	0.016	1.28	1.021	0.019	1.405
	6	1.124	0.044	0.111	1.097	0.028	0.375	1.042	0.024	1.263	1.045	0.025	1.61

		Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	7	1.189	0.04	0.113	1.161	0.023	0.383	1.092	0.034	1.363	1.092	0.031	1.619
	8	1.252	0.046	0.113	1.213	0.027	0.38	1.15	0.043	1.698	1.141	0.045	2.266
	9	1.217	0.072	0.115	1.193	0.061	0.392	1.119	0.063	1.508	1.121	0.064	1.946
	10	1.203	0.052	0.115	1.168	0.045	0.378	1.106	0.052	1.842	1.094	0.051	2.587
190	2	1.188	0.089	0.111	1.102	0.062	0.346	1.008	0.011	0.284	1.004	0.008	0.374
	3	1.294	0.114	0.117	1.169	0.095	0.366	1.007	0.008	0.531	1.007	0.02	0.614
	4	1.089	0.072	0.115	1.054	0.027	0.383	1.014	0.014	0.742	1.013	0.013	0.95
	5	1.112	0.065	0.12	1.062	0.032	0.396	1.019	0.018	1.317	1.019	0.024	1.781
	6	1.172	0.063	0.119	1.123	0.03	0.4	1.064	0.03	1.759	1.067	0.022	2.261
	7	1.148	0.048	0.118	1.119	0.037	0.401	1.067	0.034	1.686	1.067	0.034	1.966
	8	1.16	0.038	0.118	1.127	0.027	0.394	1.064	0.031	2.069	1.066	0.037	3.123
	9	1.203	0.039	0.122	1.177	0.031	0.398	1.115	0.054	1.925	1.109	0.053	2.045
	10	1.306	0.056	0.121	1.269	0.047	0.386	1.175	0.068	2.004	1.166	0.06	2.646
200	2	1.181	0.075	0.118	1.116	0.058	0.372	1.014	0.023	0.347	1.013	0.02	0.479
	3	1.245	0.11	0.122	1.138	0.075	0.383	1.006	0.008	0.572	1.004	0.007	0.67
	4	1.124	0.088	0.121	1.07	0.035	0.396	1.016	0.023	1.487	1.014	0.022	1.886
	5	1.143	0.047	0.125	1.109	0.023	0.425	1.057	0.025	1.394	1.059	0.02	1.949
	6	1.159	0.055	0.127	1.117	0.035	0.411	1.066	0.033	1.816	1.063	0.035	2.595
	7	1.159	0.042	0.126	1.125	0.033	0.422	1.067	0.031	3.267	1.071	0.032	3.836
	8	1.213	0.042	0.127	1.18	0.026	0.42	1.112	0.059	2.079	1.102	0.065	2.491
	9	1.207	0.043	0.128	1.179	0.031	0.432	1.104	0.053	2.899	1.113	0.051	3.538
	10	1.204	0.04	0.131	1.179	0.03	0.428	1.108	0.042	2.235	1.113	0.048	2.634
210	2	1.18	0.097	0.125	1.09	0.062	0.396	1.002	0.003	0.312	1.005	0.034	0.413
	3	1.183	0.101	0.13	1.111	0.059	0.404	1.009	0.017	1.325	1.008	0.017	1.92
	4	1.207	0.137	0.134	1.107	0.09	0.413	1.021	0.019	1.717	1.02	0.017	1.925
	5	1.115	0.047	0.134	1.086	0.025	0.444	1.039	0.019	1.829	1.037	0.017	2.391
	6	1.127	0.039	0.132	1.101	0.028	0.439	1.042	0.023	1.865	1.044	0.022	2.257
	7	1.172	0.052	0.136	1.138	0.029	0.441	1.079	0.031	2.996	1.079	0.031	4.307

N	K	Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	8	1.132	0.047	0.136	1.114	0.037	0.451	1.07	0.042	2.485	1.073	0.044	2.517
	9	1.227	0.043	0.135	1.192	0.025	0.448	1.124	0.041	2.684	1.117	0.049	3.786
	10	1.23	0.04	0.138	1.203	0.028	0.443	1.11	0.054	2.865	1.112	0.048	3.3
220	2	1.235	0.088	0.136	1.124	0.078	0.41	1.005	0.011	0.389	1.002	0.003	0.49
	3	1.264	0.12	0.137	1.183	0.098	0.43	1.009	0.009	0.829	1.011	0.023	1.058
	4	1.117	0.104	0.14	1.062	0.039	0.448	1.013	0.017	1.926	1.014	0.018	2.592
	5	1.106	0.05	0.142	1.069	0.02	0.468	1.025	0.026	2.476	1.022	0.023	3.266
	6	1.125	0.044	0.144	1.099	0.033	0.464	1.044	0.028	2.263	1.054	0.029	2.661
	7	1.158	0.045	0.146	1.137	0.04	0.467	1.076	0.035	3.177	1.073	0.041	3.379
	8	1.165	0.05	0.147	1.131	0.029	0.473	1.063	0.031	2.533	1.063	0.036	2.854
	9	1.203	0.041	0.143	1.176	0.029	0.465	1.101	0.057	2.282	1.09	0.049	2.967
	10	1.222	0.04	0.143	1.192	0.028	0.478	1.092	0.063	2.495	1.103	0.071	3.046
230	2	1.194	0.089	0.142	1.108	0.071	0.438	1.007	0.008	0.502	1.008	0.035	0.817
	3	1.207	0.108	0.145	1.113	0.077	0.454	1.005	0.005	0.691	1.007	0.017	0.94
	4	1.135	0.106	0.146	1.086	0.052	0.474	1.021	0.026	1.534	1.014	0.025	2.337
	5	1.11	0.048	0.149	1.072	0.025	0.484	1.016	0.017	2.391	1.017	0.017	2.33
	6	1.13	0.046	0.151	1.1	0.029	0.498	1.036	0.026	2.592	1.035	0.017	2.872
	7	1.17	0.051	0.155	1.128	0.03	0.479	1.067	0.037	3.159	1.072	0.035	3.485
	8	1.137	0.051	0.153	1.106	0.027	0.495	1.056	0.036	3.253	1.052	0.037	3.881
	9	1.185	0.045	0.155	1.152	0.025	0.501	1.077	0.04	2.967	1.087	0.044	3.621
	10	1.229	0.031	0.154	1.208	0.026	0.495	1.122	0.051	3.616	1.107	0.045	4.362
240	2	1.174	0.079	0.147	1.09	0.054	0.47	1.012	0.01	0.394	1.011	0.009	0.52
	3	1.204	0.144	0.151	1.105	0.087	0.475	1.01	0.017	0.881	1.006	0.014	1.161
	4	1.125	0.097	0.153	1.092	0.062	0.5	1.017	0.023	2.192	1.014	0.019	2.536
	5	1.113	0.058	0.152	1.071	0.029	0.512	1.028	0.023	2.542	1.025	0.018	3.05
	6	1.13	0.049	0.159	1.097	0.027	0.517	1.032	0.021	2.444	1.035	0.028	2.747
	7	1.189	0.036	0.157	1.159	0.029	0.518	1.082	0.036	3.523	1.078	0.034	3.744
	8	1.201	0.05	0.157	1.162	0.034	0.513	1.105	0.047	3.278	1.092	0.041	4.15

		Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	9	1.18	0.043	0.159	1.15	0.022	0.514	1.097	0.048	3.888	1.082	0.042	4.463
	10	1.215	0.044	0.159	1.183	0.035	0.518	1.076	0.043	3.437	1.076	0.053	3.806
250	2	1.217	0.095	0.155	1.127	0.068	0.463	1.01	0.017	0.575	1.008	0.017	0.789
	3	1.246	0.128	0.159	1.138	0.093	0.484	1.006	0.007	0.876	1.005	0.007	1.112
	4	1.095	0.058	0.159	1.061	0.028	0.529	1.018	0.02	2.418	1.014	0.019	5.035
	5	1.102	0.061	0.16	1.069	0.025	0.531	1.025	0.014	2.592	1.02	0.012	3.485
	6	1.154	0.045	0.166	1.13	0.021	0.531	1.06	0.025	4.761	1.057	0.024	5.286
	7	1.157	0.044	0.166	1.128	0.034	0.523	1.064	0.029	3.689	1.064	0.029	5.5
	8	1.187	0.042	0.165	1.156	0.023	0.538	1.085	0.04	4.858	1.088	0.044	5.133
	9	1.174	0.048	0.166	1.147	0.034	0.54	1.085	0.04	2.713	1.082	0.04	3.304
	10	1.228	0.048	0.164	1.194	0.031	0.542	1.087	0.049	4.378	1.092	0.054	4.408
260	2	1.199	0.087	0.152	1.109	0.054	0.478	1.006	0.01	0.579	1.003	0.005	0.81
	3	1.248	0.121	0.158	1.108	0.076	0.512	1.008	0.016	1	1.005	0.006	1.065
	4	1.122	0.096	0.161	1.071	0.035	0.533	1.013	0.028	2.997	1.009	0.025	4.385
	5	1.103	0.043	0.162	1.068	0.024	0.544	1.015	0.015	3.682	1.013	0.013	4.257
	6	1.142	0.044	0.163	1.109	0.034	0.536	1.049	0.03	3.629	1.049	0.028	4.653
	7	1.165	0.044	0.163	1.131	0.034	0.541	1.076	0.032	4.452	1.074	0.037	5.126
	8	1.17	0.05	0.167	1.127	0.026	0.543	1.074	0.036	4.353	1.073	0.035	4.734
	9	1.183	0.033	0.168	1.156	0.022	0.536	1.09	0.04	5.105	1.086	0.041	5.721
	10	1.2	0.041	0.173	1.171	0.03	0.553	1.088	0.041	4.868	1.09	0.043	5.253
270	2	1.208	0.1	0.166	1.106	0.069	0.489	1.005	0.004	0.499	1.004	0.005	0.685
	3	1.227	0.147	0.163	1.116	0.081	0.514	1.004	0.003	0.905	1.003	0.003	1.273
	4	1.12	0.105	0.163	1.07	0.038	0.54	1.018	0.032	7.937	1.013	0.028	16.369
	5	1.104	0.051	0.168	1.068	0.024	0.572	1.019	0.014	4.553	1.021	0.022	6.589
	6	1.129	0.035	0.172	1.105	0.025	0.562	1.05	0.021	4.646	1.047	0.019	5.512
	7	1.16	0.05	0.173	1.122	0.031	0.558	1.07	0.031	5.827	1.07	0.032	6.094
	8	1.154	0.035	0.175	1.133	0.024	0.57	1.076	0.035	5.169	1.075	0.033	5.591
	9	1.189	0.034	0.174	1.163	0.028	0.569	1.092	0.048	3.832	1.086	0.046	4.424

N	K	Hopfield (1 Run)			Hopfield (4 Runs)			Haralampiev (1 Run)			Haralampiev (4 Runs)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	10	1.192	0.036	0.176	1.173	0.027	0.579	1.097	0.038	4.573	1.099	0.032	5.898
280	2	1.187	0.088	0.171	1.11	0.06	0.519	1.007	0.005	0.633	1.006	0.004	0.741
	3	1.214	0.121	0.17	1.122	0.076	0.547	1.011	0.023	0.944	1.008	0.006	1.161
	4	1.128	0.092	0.174	1.065	0.032	0.565	1.013	0.022	4.496	1.007	0.016	5.319
	5	1.093	0.037	0.174	1.07	0.023	0.588	1.022	0.018	5.862	1.017	0.014	7.399
	6	1.124	0.035	0.176	1.097	0.02	0.589	1.037	0.017	6.064	1.036	0.018	6.494
	7	1.155	0.046	0.178	1.12	0.031	0.594	1.059	0.03	7.614	1.054	0.027	10.402
	8	1.189	0.047	0.185	1.148	0.027	0.593	1.071	0.038	4.284	1.072	0.029	5.449
	9	1.194	0.043	0.184	1.171	0.029	0.587	1.084	0.03	6.006	1.088	0.035	7.101
	10	1.229	0.031	0.181	1.202	0.026	0.592	1.112	0.053	4.58	1.118	0.061	5.422
290	2	1.208	0.09	0.173	1.124	0.073	0.54	1.006	0.006	0.7	1.004	0.004	1.001
	3	1.208	0.115	0.182	1.117	0.084	0.564	1.007	0.016	1.179	1.005	0.005	1.358
	4	1.16	0.121	0.18	1.08	0.05	0.568	1.009	0.021	5.715	1.01	0.025	8.204
	5	1.096	0.05	0.18	1.069	0.023	0.609	1.016	0.012	5.977	1.016	0.015	7.357
	6	1.116	0.038	0.187	1.091	0.02	0.613	1.036	0.016	6.891	1.035	0.017	8.203
	7	1.139	0.046	0.184	1.105	0.033	0.603	1.048	0.024	5.963	1.046	0.022	7.3
	8	1.161	0.033	0.187	1.135	0.028	0.608	1.071	0.034	4.679	1.072	0.036	5.199
	9	1.188	0.033	0.183	1.166	0.022	0.615	1.088	0.031	6.711	1.096	0.038	8.13
	10	1.211	0.029	0.19	1.19	0.024	0.629	1.106	0.041	5.919	1.101	0.037	6.579
300	2	1.192	0.083	0.182	1.113	0.064	0.573	1.006	0.006	0.633	1.004	0.005	0.926
	3	1.236	0.124	0.183	1.108	0.074	0.59	1.005	0.004	1.094	1.005	0.004	1.366
	4	1.134	0.103	0.19	1.069	0.04	0.612	1.014	0.024	5.174	1.008	0.015	8.291
	5	1.099	0.041	0.192	1.064	0.024	0.639	1.013	0.013	6.428	1.013	0.011	8.426
	6	1.126	0.036	0.195	1.092	0.023	0.641	1.036	0.017	6.926	1.035	0.018	8.014
	7	1.156	0.047	0.191	1.116	0.031	0.632	1.055	0.023	6.763	1.055	0.025	7.341
	8	1.16	0.045	0.196	1.134	0.031	0.63	1.07	0.035	5.826	1.066	0.029	6.745
	9	1.18	0.033	0.187	1.156	0.019	0.636	1.092	0.044	5.433	1.087	0.03	6.525
	10	1.211	0.034	0.199	1.183	0.024	0.663	1.105	0.049	5.384	1.105	0.04	5.918

B.3.2 Hopfield Network and Local Search Results

		Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
30	2	1.152	0.151	0.018	1.048	0.058	0.139	1.021	0.019	0.006	1.021	0.019	0.006
	3	1.232	0.196	0.018	1.095	0.075	0.146	1.032	0.029	0.01	1.032	0.029	0.009
	4	1.217	0.144	0.018	1.129	0.123	0.148	1.043	0.033	0.013	1.043	0.033	0.012
	5	1.297	0.193	0.018	1.185	0.155	0.158	1.046	0.032	0.016	1.046	0.032	0.015
	6	1.394	0.24	0.018	1.284	0.163	0.148	1.046	0.028	0.018	1.046	0.028	0.017
	7	1.511	0.261	0.017	1.396	0.22	0.155	1.045	0.036	0.029	1.045	0.036	0.027
	8	1.356	0.21	0.017	1.274	0.17	0.152	1.06	0.037	0.025	1.06	0.037	0.023
	9	1.651	0.275	0.017	1.545	0.241	0.156	1.056	0.036	0.035	1.056	0.036	0.033
	10	1.833	0.449	0.017	1.659	0.4	0.158	1.05	0.031	0.045	1.05	0.031	0.042
40	2	1.137	0.14	0.023	1.045	0.07	0.185	1.014	0.012	0.008	1.014	0.012	0.007
	3	1.117	0.067	0.023	1.055	0.032	0.188	1.03	0.024	0.012	1.03	0.024	0.012
	4	1.13	0.105	0.023	1.069	0.052	0.199	1.038	0.022	0.019	1.038	0.022	0.017
	5	1.338	0.201	0.024	1.21	0.159	0.191	1.034	0.022	0.022	1.034	0.022	0.02
	6	1.293	0.121	0.024	1.217	0.094	0.203	1.046	0.035	0.033	1.046	0.035	0.031
	7	1.412	0.098	0.024	1.353	0.097	0.197	1.041	0.026	0.034	1.041	0.026	0.031
	8	1.23	0.098	0.023	1.178	0.082	0.206	1.056	0.032	0.045	1.056	0.032	0.042
	9	1.55	0.163	0.022	1.473	0.142	0.203	1.054	0.028	0.056	1.054	0.028	0.052
	10	1.587	0.267	0.023	1.491	0.239	0.211	1.053	0.031	0.065	1.053	0.031	0.06
50	2	1.149	0.113	0.029	1.051	0.055	0.201	1.017	0.023	0.011	1.017	0.023	0.01
	3	1.199	0.134	0.028	1.083	0.077	0.225	1.018	0.016	0.017	1.018	0.016	0.016
	4	1.216	0.129	0.029	1.111	0.089	0.223	1.026	0.024	0.03	1.026	0.024	0.028
	5	1.209	0.097	0.029	1.116	0.061	0.242	1.035	0.024	0.031	1.035	0.024	0.029
	6	1.183	0.093	0.029	1.093	0.031	0.242	1.037	0.023	0.045	1.037	0.023	0.042
	7	1.319	0.13	0.029	1.236	0.112	0.245	1.037	0.02	0.061	1.037	0.02	0.056
	8	1.231	0.084	0.03	1.151	0.068	0.26	1.054	0.026	0.056	1.054	0.026	0.052
	9	1.462	0.335	0.029	1.305	0.226	0.243	1.059	0.025	0.068	1.059	0.025	0.063
	10	1.408	0.114	0.029	1.329	0.106	0.252	1.047	0.024	0.087	1.047	0.024	0.082
60	2	1.17	0.131	0.034	1.059	0.081	0.23	1.019	0.025	0.017	1.019	0.025	0.015

		Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	3	1.217	0.175	0.035	1.078	0.071	0.244	1.023	0.019	0.021	1.023	0.019	0.02
	4	1.198	0.168	0.035	1.076	0.083	0.265	1.026	0.021	0.033	1.026	0.021	0.031
	5	1.171	0.123	0.036	1.077	0.053	0.282	1.031	0.024	0.045	1.031	0.024	0.042
	6	1.214	0.106	0.035	1.118	0.041	0.293	1.03	0.018	0.053	1.03	0.018	0.05
	7	1.277	0.083	0.035	1.205	0.059	0.281	1.041	0.021	0.065	1.041	0.021	0.061
	8	1.249	0.069	0.034	1.183	0.046	0.285	1.044	0.02	0.066	1.044	0.02	0.061
	9	1.32	0.075	0.033	1.251	0.059	0.294	1.037	0.021	0.12	1.037	0.021	0.111
	10	1.459	0.115	0.034	1.394	0.088	0.306	1.037	0.02	0.098	1.037	0.02	0.091
70	2	1.23	0.118	0.04	1.112	0.06	0.255	1.009	0.015	0.014	1.009	0.015	0.013
	3	1.182	0.164	0.04	1.05	0.063	0.278	1.026	0.029	0.029	1.026	0.029	0.027
	4	1.186	0.062	0.041	1.139	0.038	0.311	1.018	0.017	0.038	1.018	0.017	0.035
	5	1.227	0.113	0.042	1.129	0.071	0.307	1.023	0.016	0.05	1.023	0.016	0.047
	6	1.21	0.084	0.04	1.157	0.055	0.312	1.03	0.018	0.057	1.03	0.018	0.053
	7	1.23	0.071	0.04	1.183	0.063	0.334	1.029	0.019	0.072	1.029	0.019	0.068
	8	1.32	0.103	0.041	1.261	0.086	0.335	1.034	0.02	0.09	1.034	0.02	0.085
	9	1.243	0.107	0.041	1.184	0.075	0.325	1.04	0.02	0.125	1.04	0.02	0.118
	10	1.29	0.064	0.04	1.237	0.047	0.35	1.034	0.016	0.178	1.034	0.016	0.168
80	2	1.189	0.098	0.045	1.058	0.048	0.295	1.007	0.007	0.018	1.007	0.007	0.017
	3	1.265	0.087	0.045	1.136	0.066	0.307	1.011	0.008	0.03	1.011	0.008	0.028
	4	1.084	0.07	0.048	1.038	0.025	0.37	1.018	0.014	0.039	1.018	0.014	0.036
	5	1.095	0.051	0.045	1.059	0.039	0.392	1.019	0.019	0.062	1.019	0.019	0.057
	6	1.155	0.046	0.046	1.105	0.035	0.353	1.03	0.016	0.093	1.03	0.016	0.085
	7	1.147	0.073	0.046	1.085	0.034	0.384	1.028	0.018	0.089	1.028	0.018	0.084
	8	1.329	0.076	0.046	1.282	0.06	0.384	1.038	0.017	0.119	1.038	0.017	0.11
	9	1.327	0.119	0.048	1.282	0.109	0.386	1.036	0.015	0.144	1.036	0.015	0.134
	10	1.463	0.154	0.046	1.397	0.154	0.377	1.042	0.018	0.168	1.042	0.018	0.156
90	2	1.223	0.082	0.05	1.084	0.067	0.325	1.004	0.005	0.022	1.004	0.005	0.02
	3	1.283	0.199	0.051	1.11	0.11	0.354	1.012	0.009	0.047	1.012	0.009	0.044

N	K	Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	4	1.122	0.063	0.052	1.05	0.025	0.384	1.015	0.018	0.054	1.015	0.018	0.05
	5	1.113	0.048	0.052	1.083	0.036	0.444	1.016	0.015	0.069	1.016	0.015	0.065
	6	1.182	0.073	0.052	1.136	0.059	0.401	1.024	0.015	0.112	1.024	0.015	0.105
	7	1.185	0.068	0.054	1.12	0.045	0.409	1.035	0.018	0.12	1.035	0.018	0.113
	8	1.186	0.058	0.053	1.142	0.044	0.415	1.033	0.018	0.115	1.033	0.018	0.106
	9	1.269	0.095	0.052	1.206	0.077	0.44	1.033	0.017	0.143	1.033	0.017	0.132
	10	1.334	0.052	0.053	1.278	0.034	0.422	1.03	0.015	0.186	1.03	0.015	0.169
100	2	1.106	0.057	0.055	1.051	0.026	0.363	1.013	0.013	0.026	1.013	0.013	0.024
	3	1.254	0.121	0.057	1.083	0.048	0.386	1.018	0.014	0.045	1.018	0.014	0.042
	4	1.083	0.074	0.056	1.023	0.015	0.463	1.016	0.008	0.062	1.016	0.008	0.057
	5	1.142	0.117	0.058	1.069	0.054	0.433	1.017	0.018	0.083	1.017	0.018	0.077
	6	1.168	0.044	0.057	1.107	0.025	0.439	1.021	0.013	0.096	1.021	0.013	0.09
	7	1.188	0.065	0.057	1.139	0.047	0.465	1.028	0.016	0.136	1.028	0.016	0.125
	8	1.194	0.064	0.058	1.138	0.038	0.468	1.034	0.018	0.181	1.034	0.018	0.168
	9	1.213	0.055	0.061	1.166	0.038	0.479	1.024	0.018	0.198	1.024	0.018	0.188
	10	1.29	0.059	0.058	1.231	0.047	0.458	1.029	0.017	0.235	1.029	0.017	0.215
110	2	1.201	0.076	0.062	1.1	0.07	0.376	1.004	0.004	0.034	1.004	0.004	0.032
	3	1.204	0.105	0.063	1.077	0.051	0.405	1.008	0.007	0.048	1.008	0.007	0.045
	4	1.142	0.102	0.064	1.058	0.039	0.466	1.009	0.012	0.086	1.009	0.012	0.08
	5	1.115	0.059	0.064	1.064	0.03	0.51	1.024	0.02	0.107	1.024	0.02	0.102
	6	1.174	0.061	0.065	1.127	0.053	0.491	1.018	0.014	0.141	1.018	0.014	0.132
	7	1.162	0.053	0.064	1.117	0.043	0.509	1.03	0.019	0.149	1.03	0.019	0.138
	8	1.211	0.055	0.065	1.179	0.045	0.514	1.024	0.013	0.193	1.024	0.013	0.176
	9	1.192	0.069	0.066	1.142	0.047	0.507	1.031	0.017	0.204	1.031	0.017	0.188
	10	1.224	0.073	0.067	1.178	0.064	0.521	1.03	0.015	0.256	1.03	0.015	0.238
120	2	1.173	0.085	0.068	1.067	0.049	0.403	1.004	0.009	0.037	1.004	0.009	0.034
	3	1.157	0.117	0.069	1.062	0.032	0.456	1.013	0.012	0.078	1.013	0.012	0.074
	4	1.192	0.102	0.069	1.08	0.051	0.464	1.012	0.009	0.085	1.012	0.009	0.081

		Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	5	1.123	0.074	0.074	1.054	0.025	0.531	1.02	0.014	0.119	1.02	0.014	0.112
	6	1.17	0.049	0.072	1.117	0.033	0.53	1.023	0.015	0.188	1.023	0.015	0.172
	7	1.206	0.083	0.07	1.113	0.053	0.499	1.024	0.018	0.158	1.024	0.018	0.147
	8	1.308	0.048	0.072	1.251	0.03	0.554	1.033	0.016	0.23	1.033	0.016	0.221
	9	1.212	0.067	0.07	1.164	0.055	0.538	1.02	0.012	0.232	1.02	0.012	0.217
	10	1.282	0.118	0.071	1.229	0.103	0.574	1.028	0.016	0.325	1.028	0.016	0.31
130	2	1.199	0.085	0.078	1.087	0.063	0.461	1.003	0.003	0.035	1.003	0.003	0.033
	3	1.179	0.149	0.078	1.064	0.048	0.515	1.006	0.006	0.074	1.006	0.006	0.07
	4	1.108	0.068	0.078	1.057	0.027	0.563	1.021	0.016	0.128	1.021	0.016	0.12
	5	1.119	0.047	0.08	1.075	0.021	0.613	1.017	0.015	0.117	1.017	0.015	0.11
	6	1.15	0.052	0.081	1.098	0.035	0.563	1.018	0.012	0.19	1.018	0.012	0.182
	7	1.216	0.073	0.079	1.164	0.049	0.581	1.024	0.016	0.252	1.024	0.016	0.237
	8	1.203	0.054	0.08	1.151	0.032	0.585	1.03	0.014	0.229	1.03	0.014	0.221
	9	1.186	0.054	0.083	1.128	0.029	0.59	1.025	0.013	0.26	1.025	0.013	0.244
	10	1.287	0.068	0.08	1.23	0.045	0.605	1.025	0.013	0.34	1.025	0.013	0.319
140	2	1.233	0.149	0.084	1.067	0.06	0.521	1.004	0.003	0.041	1.004	0.003	0.039
	3	1.238	0.113	0.084	1.083	0.056	0.503	1.007	0.007	0.087	1.007	0.007	0.082
	4	1.135	0.101	0.084	1.046	0.023	0.583	1.017	0.015	0.103	1.017	0.015	0.096
	5	1.142	0.063	0.086	1.086	0.038	0.63	1.02	0.016	0.151	1.02	0.016	0.142
	6	1.136	0.054	0.087	1.092	0.033	0.624	1.017	0.015	0.218	1.017	0.015	0.203
	7	1.157	0.045	0.089	1.116	0.031	0.66	1.022	0.014	0.186	1.022	0.014	0.175
	8	1.136	0.048	0.087	1.092	0.028	0.648	1.028	0.015	0.277	1.028	0.015	0.269
	9	1.156	0.053	0.087	1.109	0.034	0.627	1.029	0.015	0.274	1.029	0.015	0.248
	10	1.318	0.066	0.088	1.247	0.051	0.58	1.023	0.01	0.29	1.023	0.01	0.274
150	2	1.165	0.087	0.088	1.069	0.036	0.534	1.005	0.003	0.056	1.005	0.003	0.053
	3	1.234	0.094	0.09	1.12	0.042	0.532	1.008	0.01	0.096	1.008	0.01	0.092
	4	1.165	0.147	0.091	1.052	0.049	0.588	1.015	0.014	0.161	1.015	0.014	0.155
	5	1.119	0.055	0.092	1.059	0.02	0.685	1.018	0.014	0.192	1.018	0.014	0.184

N	K	Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	6	1.143	0.037	0.094	1.095	0.033	0.695	1.022	0.016	0.192	1.022	0.016	0.178
	7	1.166	0.061	0.092	1.114	0.038	0.639	1.024	0.011	0.271	1.024	0.011	0.262
	8	1.213	0.052	0.095	1.165	0.034	0.678	1.017	0.01	0.281	1.017	0.01	0.271
	9	1.172	0.042	0.096	1.131	0.033	0.71	1.018	0.013	0.343	1.018	0.013	0.343
	10	1.221	0.046	0.095	1.183	0.035	0.711	1.029	0.015	0.34	1.029	0.015	0.331
160	2	1.158	0.081	0.098	1.067	0.045	0.548	1.001	0.003	0.05	1.001	0.003	0.048
	3	1.24	0.137	0.097	1.074	0.07	0.588	1.007	0.005	0.095	1.007	0.005	0.091
	4	1.102	0.078	0.096	1.041	0.023	0.716	1.007	0.009	0.153	1.007	0.009	0.149
	5	1.125	0.081	0.099	1.064	0.024	0.71	1.01	0.009	0.201	1.01	0.009	0.189
	6	1.174	0.059	0.099	1.122	0.041	0.677	1.036	0.027	0.23	1.036	0.027	0.217
	7	1.178	0.07	0.102	1.129	0.053	0.708	1.018	0.011	0.278	1.018	0.011	0.261
	8	1.29	0.047	0.099	1.244	0.028	0.73	1.021	0.012	0.338	1.021	0.012	0.326
	9	1.219	0.049	0.101	1.172	0.028	0.697	1.02	0.011	0.417	1.02	0.011	0.403
	10	1.199	0.051	0.099	1.142	0.038	0.754	1.022	0.01	0.318	1.022	0.01	0.307
170	2	1.229	0.099	0.099	1.067	0.055	0.594	1.002	0.002	0.078	1.002	0.002	0.076
	3	1.197	0.127	0.104	1.065	0.036	0.635	1.009	0.008	0.127	1.009	0.008	0.126
	4	1.153	0.131	0.103	1.051	0.043	0.699	1.016	0.013	0.158	1.016	0.013	0.152
	5	1.101	0.056	0.106	1.045	0.02	0.791	1.015	0.01	0.165	1.015	0.01	0.166
	6	1.131	0.048	0.106	1.077	0.019	0.741	1.015	0.009	0.197	1.015	0.009	0.181
	7	1.168	0.058	0.105	1.12	0.035	0.772	1.016	0.013	0.276	1.016	0.013	0.277
	8	1.185	0.052	0.108	1.131	0.036	0.732	1.025	0.013	0.313	1.025	0.013	0.3
	9	1.233	0.052	0.109	1.186	0.03	0.779	1.022	0.013	0.422	1.022	0.013	0.419
	10	1.26	0.058	0.11	1.211	0.047	0.714	1.03	0.015	0.439	1.03	0.015	0.418
180	2	1.257	0.105	0.108	1.074	0.066	0.611	1.003	0.002	0.059	1.003	0.002	0.057
	3	1.246	0.081	0.109	1.104	0.061	0.659	1.007	0.006	0.136	1.007	0.006	0.13
	4	1.095	0.054	0.111	1.046	0.02	0.784	1.006	0.005	0.192	1.006	0.005	0.184
	5	1.112	0.072	0.112	1.055	0.023	0.833	1.018	0.012	0.237	1.018	0.012	0.233
	6	1.124	0.044	0.111	1.084	0.022	0.819	1.016	0.008	0.241	1.016	0.008	0.238

		Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	7	1.189	0.04	0.113	1.148	0.019	0.841	1.017	0.011	0.325	1.017	0.011	0.313
	8	1.252	0.046	0.113	1.2	0.024	0.787	1.017	0.008	0.529	1.017	0.008	0.5
	9	1.217	0.072	0.115	1.176	0.061	0.845	1.021	0.011	0.488	1.021	0.011	0.453
	10	1.203	0.052	0.115	1.157	0.04	0.807	1.021	0.01	0.451	1.021	0.01	0.428
190	2	1.188	0.089	0.111	1.054	0.037	0.662	1.003	0.002	0.066	1.003	0.002	0.067
	3	1.294	0.114	0.117	1.107	0.07	0.663	1.015	0.009	0.09	1.015	0.009	0.091
	4	1.089	0.072	0.115	1.037	0.017	0.791	1.012	0.011	0.174	1.012	0.011	0.175
	5	1.112	0.065	0.12	1.046	0.022	0.826	1.015	0.009	0.173	1.015	0.009	0.174
	6	1.172	0.063	0.119	1.113	0.022	0.877	1.016	0.012	0.316	1.016	0.012	0.318
	7	1.148	0.048	0.118	1.1	0.029	0.843	1.012	0.009	0.305	1.012	0.009	0.306
	8	1.16	0.038	0.118	1.116	0.021	0.822	1.015	0.007	0.341	1.015	0.007	0.341
	9	1.203	0.039	0.122	1.168	0.029	0.878	1.019	0.009	0.45	1.019	0.009	0.446
	10	1.306	0.056	0.121	1.249	0.046	0.755	1.024	0.013	0.532	1.024	0.013	0.539
200	2	1.181	0.075	0.118	1.069	0.049	0.745	1.001	0.001	0.059	1.001	0.001	0.058
	3	1.245	0.11	0.122	1.093	0.056	0.749	1.008	0.008	0.148	1.008	0.008	0.143
	4	1.124	0.088	0.121	1.048	0.022	0.793	1.012	0.013	0.253	1.012	0.013	0.245
	5	1.143	0.047	0.125	1.098	0.015	0.881	1.007	0.008	0.237	1.007	0.008	0.235
	6	1.159	0.055	0.127	1.102	0.028	0.835	1.018	0.01	0.235	1.018	0.01	0.232
	7	1.159	0.042	0.126	1.11	0.025	0.912	1.011	0.01	0.404	1.011	0.01	0.402
	8	1.213	0.042	0.127	1.165	0.022	0.924	1.017	0.009	0.364	1.017	0.009	0.367
	9	1.207	0.043	0.128	1.162	0.028	0.934	1.018	0.01	0.416	1.018	0.01	0.411
	10	1.204	0.04	0.131	1.163	0.028	0.958	1.026	0.011	0.476	1.026	0.011	0.478
210	2	1.18	0.097	0.125	1.053	0.035	0.733	1.003	0.002	0.088	1.003	0.002	0.088
	3	1.183	0.101	0.13	1.069	0.048	0.788	1.004	0.004	0.187	1.004	0.004	0.186
	4	1.207	0.137	0.134	1.054	0.038	0.764	1.015	0.01	0.23	1.015	0.01	0.23
	5	1.115	0.047	0.134	1.072	0.016	0.956	1.011	0.012	0.236	1.011	0.012	0.236
	6	1.127	0.039	0.132	1.084	0.022	0.939	1.014	0.014	0.269	1.014	0.014	0.265
	7	1.172	0.052	0.136	1.123	0.023	0.98	1.025	0.01	0.414	1.025	0.01	0.409

N	K	Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	8	1.132	0.047	0.136	1.097	0.028	0.994	1.024	0.011	0.435	1.024	0.011	0.433
	9	1.227	0.043	0.135	1.178	0.022	0.919	1.017	0.008	0.605	1.017	0.008	0.592
	10	1.23	0.04	0.138	1.186	0.03	0.892	1.021	0.01	0.626	1.021	0.011	0.621
220	2	1.235	0.088	0.136	1.081	0.062	0.737	1.002	0.001	0.068	1.002	0.001	0.067
	3	1.264	0.12	0.137	1.097	0.06	0.771	1.003	0.005	0.159	1.003	0.005	0.16
	4	1.117	0.104	0.14	1.047	0.039	0.957	1.012	0.011	0.249	1.012	0.011	0.248
	5	1.106	0.05	0.142	1.053	0.016	1.017	1.015	0.007	0.284	1.015	0.007	0.285
	6	1.125	0.044	0.144	1.083	0.024	0.93	1.014	0.011	0.352	1.014	0.011	0.342
	7	1.158	0.045	0.146	1.118	0.031	0.949	1.02	0.01	0.394	1.02	0.01	0.404
	8	1.165	0.05	0.147	1.117	0.027	1.007	1.019	0.008	0.467	1.019	0.008	0.46
	9	1.203	0.041	0.143	1.155	0.02	1.003	1.015	0.009	0.581	1.015	0.009	0.569
	10	1.222	0.04	0.143	1.173	0.028	1.047	1.021	0.01	0.581	1.021	0.01	0.569
230	2	1.194	0.089	0.142	1.067	0.054	0.807	1.002	0.001	0.081	1.002	0.001	0.08
	3	1.207	0.108	0.145	1.054	0.036	0.838	1.006	0.005	0.202	1.006	0.005	0.203
	4	1.135	0.106	0.146	1.052	0.03	0.926	1.017	0.015	0.261	1.017	0.015	0.264
	5	1.11	0.048	0.149	1.054	0.016	1.008	1.008	0.007	0.318	1.008	0.007	0.315
	6	1.13	0.046	0.151	1.08	0.021	1.089	1.017	0.012	0.426	1.017	0.012	0.431
	7	1.17	0.051	0.155	1.114	0.027	0.979	1.01	0.009	0.402	1.01	0.009	0.399
	8	1.137	0.051	0.153	1.091	0.026	1.002	1.019	0.013	0.536	1.019	0.013	0.537
	9	1.185	0.045	0.155	1.132	0.026	1.022	1.018	0.009	0.503	1.018	0.009	0.508
	10	1.229	0.031	0.154	1.181	0.023	0.968	1.019	0.009	0.664	1.019	0.009	0.67
240	2	1.174	0.079	0.147	1.065	0.039	0.874	1.001	0.001	0.087	1.001	0.001	0.087
	3	1.204	0.144	0.151	1.047	0.032	0.929	1.002	0.003	0.15	1.002	0.003	0.15
	4	1.125	0.097	0.153	1.057	0.023	0.953	1.007	0.01	0.358	1.007	0.01	0.34
	5	1.113	0.058	0.152	1.059	0.021	1.047	1.012	0.009	0.389	1.012	0.009	0.367
	6	1.13	0.049	0.159	1.084	0.021	1.026	1.012	0.009	0.43	1.012	0.009	0.403
	7	1.189	0.036	0.157	1.144	0.026	1.065	1.014	0.01	0.472	1.014	0.01	0.45
	8	1.201	0.05	0.157	1.15	0.031	1.066	1.017	0.009	0.464	1.017	0.009	0.436

		Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
N	K	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	9	1.18	0.043	0.159	1.141	0.022	1.1	1.014	0.008	0.652	1.014	0.008	0.623
	10	1.215	0.044	0.159	1.167	0.036	1.103	1.024	0.008	0.712	1.023	0.008	0.682
250	2	1.217	0.095	0.155	1.076	0.058	0.857	1.002	0.001	0.089	1.002	0.001	0.084
	3	1.246	0.128	0.159	1.07	0.052	0.872	1.003	0.004	0.182	1.003	0.004	0.173
	4	1.095	0.058	0.159	1.047	0.019	1.054	1.013	0.013	0.338	1.013	0.013	0.316
	5	1.102	0.061	0.16	1.056	0.022	1.118	1.011	0.009	0.407	1.011	0.009	0.384
	6	1.154	0.045	0.166	1.111	0.021	1.125	1.011	0.011	0.391	1.011	0.011	0.366
	7	1.157	0.044	0.166	1.111	0.028	1.094	1.011	0.008	0.499	1.011	0.008	0.47
	8	1.187	0.042	0.165	1.143	0.025	1.089	1.025	0.013	0.574	1.025	0.013	0.542
	9	1.174	0.048	0.166	1.133	0.027	1.138	1.019	0.006	0.736	1.019	0.006	0.704
	10	1.228	0.048	0.164	1.18	0.027	1.058	1.027	0.01	0.655	1.027	0.01	0.617
260	2	1.199	0.087	0.152	1.063	0.049	0.914	1.001	0.001	0.121	1.001	0.001	0.116
	3	1.248	0.121	0.158	1.063	0.044	0.91	1.009	0.004	0.199	1.009	0.004	0.195
	4	1.122	0.096	0.161	1.051	0.028	1.122	1.004	0.004	0.421	1.004	0.004	0.406
	5	1.103	0.043	0.162	1.054	0.017	1.156	1.014	0.008	0.377	1.014	0.008	0.364
	6	1.142	0.044	0.163	1.092	0.03	1.101	1.013	0.011	0.392	1.013	0.011	0.383
	7	1.165	0.044	0.163	1.116	0.028	1.073	1.011	0.008	0.406	1.011	0.008	0.387
	8	1.17	0.05	0.167	1.114	0.02	1.165	1.017	0.009	0.672	1.017	0.009	0.661
	9	1.183	0.033	0.168	1.138	0.023	1.172	1.014	0.007	0.658	1.014	0.007	0.638
	10	1.2	0.041	0.173	1.155	0.029	1.177	1.021	0.007	0.706	1.021	0.007	0.712
270	2	1.208	0.1	0.166	1.056	0.047	0.957	1.001	0.001	0.122	1.001	0.001	0.118
	3	1.227	0.147	0.163	1.052	0.039	0.982	1.004	0.003	0.225	1.004	0.003	0.217
	4	1.12	0.105	0.163	1.044	0.02	1.151	1.005	0.002	0.43	1.005	0.002	0.412
	5	1.104	0.051	0.168	1.055	0.018	1.216	1.014	0.01	0.44	1.014	0.01	0.414
	6	1.129	0.035	0.172	1.092	0.021	1.182	1.013	0.01	0.463	1.013	0.01	0.444
	7	1.16	0.05	0.173	1.106	0.024	1.152	1.009	0.01	0.645	1.009	0.01	0.623
	8	1.154	0.035	0.175	1.12	0.019	1.193	1.022	0.012	0.641	1.022	0.012	0.632
	9	1.189	0.034	0.174	1.152	0.027	1.181	1.02	0.009	0.721	1.02	0.009	0.704

N	K	Hopfield (1 Run)			Hopfield (10 Runs)			Local Search (1 Second)			Local Search (5 Seconds)		
		A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time	A. Ratio	σ	A. Time
	10	1.192	0.036	0.176	1.161	0.02	1.265	1.025	0.009	0.934	1.023	0.009	1.108
280	2	1.187	0.088	0.171	1.061	0.044	0.991	1.002	0.001	0.105	1.002	0.001	0.108
	3	1.214	0.121	0.17	1.068	0.043	1.016	1.005	0.004	0.248	1.005	0.004	0.249
	4	1.128	0.092	0.174	1.049	0.018	1.183	1.007	0.009	0.457	1.007	0.009	0.462
	5	1.093	0.037	0.174	1.052	0.019	1.263	1.011	0.009	0.431	1.011	0.009	0.446
	6	1.124	0.035	0.176	1.08	0.018	1.229	1.012	0.008	0.407	1.012	0.008	0.415
	7	1.155	0.046	0.178	1.102	0.029	1.239	1.014	0.011	0.533	1.014	0.011	0.546
	8	1.189	0.047	0.185	1.137	0.022	1.239	1.015	0.008	0.534	1.015	0.008	0.54
	9	1.194	0.043	0.184	1.153	0.023	1.234	1.018	0.01	0.735	1.018	0.01	0.761
	10	1.229	0.031	0.181	1.181	0.029	1.249	1.02	0.007	0.822	1.02	0.007	0.862
290	2	1.208	0.09	0.173	1.07	0.052	1.026	1.001	0.001	0.122	1.001	0.001	0.124
	3	1.208	0.115	0.182	1.065	0.046	1.085	1.003	0.004	0.221	1.003	0.004	0.227
	4	1.16	0.121	0.18	1.051	0.025	1.175	1.008	0.012	0.478	1.008	0.012	0.491
	5	1.096	0.05	0.18	1.052	0.019	1.293	1.009	0.008	0.508	1.009	0.008	0.518
	6	1.116	0.038	0.187	1.077	0.02	1.261	1.013	0.006	0.426	1.013	0.006	0.435
	7	1.139	0.046	0.184	1.085	0.025	1.25	1.017	0.01	0.508	1.017	0.01	0.526
	8	1.161	0.033	0.187	1.122	0.021	1.264	1.014	0.008	0.542	1.014	0.008	0.561
	9	1.188	0.033	0.183	1.152	0.024	1.296	1.021	0.012	0.74	1.021	0.012	0.776
	10	1.211	0.029	0.19	1.17	0.027	1.285	1.02	0.009	0.904	1.02	0.008	1.037
300	2	1.192	0.083	0.182	1.068	0.046	1.107	1.002	0.001	0.122	1.002	0.001	0.124
	3	1.236	0.124	0.183	1.069	0.049	1.097	1.002	0.002	0.244	1.002	0.002	0.247
	4	1.134	0.103	0.19	1.049	0.018	1.214	1.007	0.01	0.454	1.007	0.01	0.463
	5	1.099	0.041	0.192	1.052	0.019	1.354	1.01	0.008	0.482	1.01	0.008	0.497
	6	1.126	0.036	0.195	1.08	0.018	1.366	1.012	0.007	0.428	1.012	0.007	0.441
	7	1.156	0.047	0.191	1.096	0.024	1.284	1.014	0.011	0.505	1.014	0.011	0.518
	8	1.16	0.045	0.196	1.119	0.02	1.275	1.016	0.006	0.643	1.016	0.006	0.666
	9	1.18	0.033	0.187	1.142	0.018	1.323	1.022	0.012	0.779	1.022	0.012	0.82
	10	1.211	0.034	0.199	1.17	0.025	1.331	1.014	0.005	0.933	1.014	0.005	1.052

B.4 OR-Library P-Median Dataset Results

The OR-Library [5] contains 40 k-median tests labelled pmed1 through pmed40. Each test uses a single n and k value and includes a list of locations as well as the optimal distance value for the k-median problem. The n values range from 10 to 900 and the k values range from 5 to 200.

We report the approximation ratio and total runtime in seconds for each test.

B.4.1 Hopfield Network and Haralampiev Network Results

Test	N	K	Hopfield (1 Run)		Hopfield (4 Runs)		Haralampiev (1 Run)		Haralampiev (4 Runs)	
			Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time
pmed1	100	5	1.038	0.052	1.032	0.185	1.009	0.122	1.009	0.191
pmed2	100	10	1.14	0.05	1.097	0.185	1.163	0.623	1.105	1.125
pmed3	100	10	1.21	0.059	1.063	0.177	1.133	0.239	1.107	0.367
pmed4	100	20	1.173	0.063	1.142	0.178	1.242	0.389	1.161	0.834
pmed5	100	33	1.398	0.061	1.316	0.201	1.375	0.857	1.314	1.062
pmed6	200	5	1.029	0.142	1.031	0.414	1.016	0.173	1.048	0.419
pmed7	200	10	1.048	0.13	1.084	0.401	1.102	1.379	1.013	1.715
pmed8	200	20	1.18	0.145	1.112	0.446	1.151	1.18	1.184	1.99
pmed9	200	40	1.233	0.126	1.172	0.419	1.246	1.964	1.29	3.977
pmed10	200	67	1.459	0.141	1.406	0.499	1.485	3.747	1.407	13.242
pmed11	300	5	1.1	0.162	1.031	0.631	1.028	0.998	1.033	1.532
pmed12	300	10	1.054	0.219	1.06	0.559	1.025	1.337	1.109	1.666
pmed13	300	30	1.163	0.214	1.14	0.763	1.139	4.548	1.106	5.781
pmed14	300	60	1.215	0.198	1.198	0.72	1.241	6.408	1.248	16.565
pmed15	300	100	1.359	0.214	1.337	0.824	1.367	6.031	1.287	20.285
pmed16	400	5	1.04	0.289	1.009	0.773	1.018	0.407	1.021	1.715
pmed17	400	10	1.04	0.253	1.037	0.72	1.019	2.608	1.037	2.345
pmed18	400	40	1.109	0.288	1.074	0.909	1.094	5.357	1.1	12.895
pmed19	400	80	1.19	0.369	1.148	1.136	1.247	7.697	1.292	13.4
pmed20	400	133	1.311	0.357	1.329	1.083	1.363	10.978	1.397	32.046
pmed21	500	5	1.042	0.397	1.049	1.092	1.055	0.998	1.031	2.125
pmed22	500	10	1.059	0.425	1.052	0.966	1.06	1.653	1.078	2.68
pmed23	500	50	1.105	0.383	1.104	1.283	1.15	6.62	1.079	21.383
pmed24	500	100	1.158	0.412	1.18	1.525	1.118	18.13	1.202	30.242
pmed25	500	167	1.369	0.353	1.338	1.212	1.4	30.997	1.383	47.051
pmed26	600	5	1.045	0.387	1.026	1.236	1.049	1.443	1.049	2.195
pmed27	600	10	1.06	0.44	1.056	1.254	1.18	4.287	1.096	25.046
pmed28	600	60	1.087	0.535	1.094	1.643	1.148	13.053	1.137	30.878

			Hopfield (1 Run)		Hopfield (4 Runs)		Haralampiev (1 Run)		Haralampiev (4 Runs)	
Test	N	K	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time
pmed29	600	120	1.197	0.478	1.18	1.556	1.217	28.375	1.171	59.436
pmed30	600	200	1.305	0.463	1.314	1.748	1.416	25.584	1.366	76.523
pmed31	700	5	1.061	0.509	1.029	1.554	1.061	0.812	1.074	2.742
pmed32	700	10	1.055	0.49	1.043	1.779	1.091	3.391	1.099	6.169
pmed33	700	70	1.105	0.703	1.083	2.246	1.151	19.893	1.178	35.905
pmed34	700	140	1.223	0.761	1.204	1.985	1.215	28.612	1.21	89.728
pmed35	800	5	1.032	0.757	1.035	2.056	1.055	1.693	1.009	3.446
pmed36	800	10	1.016	0.733	1.026	2.079	1.113	2.514	1.066	6.961
pmed37	800	80	1.078	0.736	1.091	2.499	1.17	23.989	1.141	42.994
pmed38	900	5	1.027	0.783	1.033	1.981	1.132	4.249	1.063	4.027
pmed39	900	10	1.059	0.738	1.037	2.377	1.035	2.519	1.045	6.93
pmed40	900	90	1.117	0.871	1.099	2.724	1.126	27.969	1.175	58.165

B.4.2 Hopfield Network and Local Search Results

			Hopfield (1 Run)		Hopfield (10 Runs)		Local Search (1s)		Local Search (5s)		Local Search (15s)	
Test	N	K	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time
pmed1	100	5	1.038	0.052	1.032	0.48	1	0.258	1	0.264	1	0.27
pmed2	100	10	1.14	0.05	1.102	0.459	1.003	0.641	1.003	0.645	1.003	0.643
pmed3	100	10	1.21	0.059	1.052	0.511	1.001	0.579	1.001	0.608	1.001	0.586
pmed4	100	20	1.173	0.063	1.143	0.59	1.046	1	1.004	1.731	1.004	1.677
pmed5	100	33	1.398	0.061	1.32	0.522	1.058	1	1.002	2.142	1.002	2.132
pmed6	200	5	1.029	0.142	1.038	0.996	1	0.551	1	0.536	1	0.532
pmed7	200	10	1.048	0.13	1.042	1.117	1.044	1	1.001	2.574	1.001	2.567
pmed8	200	20	1.18	0.145	1.147	0.984	1.17	1	1.022	5	1.003	7.788
pmed9	200	40	1.233	0.126	1.139	1.086	1.296	1	1.116	5	1.01	15
pmed10	200	67	1.459	0.141	1.424	1.011	1.387	1	1.222	5	1.039	15
pmed11	300	5	1.1	0.162	1.021	1.213	1	0.663	1	0.71	1	0.711
pmed12	300	10	1.054	0.219	1.045	1.617	1.018	1	1	2.797	1	2.838
pmed13	300	30	1.163	0.214	1.099	1.774	1.158	1	1.065	5	1.007	15
pmed14	300	60	1.215	0.198	1.174	1.8	1.337	1	1.253	5	1.139	15
pmed15	300	100	1.359	0.214	1.312	1.684	1.473	1	1.331	5	1.228	15
pmed16	400	5	1.04	0.289	1.037	1.754	1.003	1	1.003	1.036	1.003	1.127
pmed17	400	10	1.04	0.253	1.015	1.988	1.1	1	1.007	5	1.001	8.405
pmed18	400	40	1.109	0.288	1.072	2.654	1.201	1	1.141	5	1.074	15
pmed19	400	80	1.19	0.369	1.153	2.394	1.32	1	1.229	5	1.137	15
pmed20	400	133	1.311	0.357	1.334	2.54	1.527	1	1.454	5	1.387	15
pmed21	500	5	1.042	0.397	1	2.445	1.02	1	1	2.402	1	2.463
pmed22	500	10	1.059	0.425	1.035	2.964	1.048	1	1.018	5	1	11.907
pmed23	500	50	1.105	0.383	1.094	3.377	1.277	1	1.185	5	1.092	15
pmed24	500	100	1.158	0.412	1.189	3.156	1.293	1.001	1.249	5	1.196	15
pmed25	500	167	1.369	0.353	1.382	3.356	1.49	1	1.417	5	1.335	15
pmed26	600	5	1.045	0.387	1.004	3.179	1.042	1.001	1	4.668	1	4.755
pmed27	600	10	1.06	0.44	1.031	3.654	1.111	1.001	1.03	5	1	12.417
pmed28	600	60	1.087	0.535	1.088	3.887	1.286	1	1.236	5	1.148	15

Test	N	K	Hopfield (1 Run)		Hopfield (10 Runs)		Local Search (1s)		Local Search (5s)		Local Search (15s)	
			Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time
pmed29	600	120	1.197	0.478	1.149	3.833	1.445	1	1.379	5	1.295	15
pmed30	600	200	1.305	0.463	1.312	4.101	1.581	1.001	1.514	5	1.464	15
pmed31	700	5	1.061	0.509	1.017	3.079	1.075	1.001	1	5	1	5.449
pmed32	700	10	1.055	0.49	1.017	4.19	1.077	1.001	1.031	5.001	1	15.001
pmed33	700	70	1.105	0.703	1.093	4.637	1.29	1.001	1.215	5.001	1.164	15.001
pmed34	700	140	1.223	0.761	1.194	5.546	1.443	1	1.383	5.001	1.331	15.001
pmed35	800	5	1.032	0.757	1.041	4.577	1.031	1.001	1	3.927	1	3.964
pmed36	800	10	1.016	0.733	1.027	5.161	1.151	1.001	1.065	5.001	1.016	15.001
pmed37	800	80	1.078	0.736	1.076	5.247	1.327	1.001	1.281	5.001	1.213	15
pmed38	900	5	1.027	0.783	1.03	5.132	1.064	1.001	1.014	5.001	1.004	12.359
pmed39	900	10	1.059	0.738	1.019	5.527	1.134	1.001	1.067	5.001	1.007	15.001
pmed40	900	90	1.117	0.871	1.097	5.779	1.368	1.001	1.306	5.001	1.262	15.001

B.5 TSPLib Dataset Results

The TSPLib dataset [37] contains large problems for the travelling salesman problem. Each problem contains over 1000 locations and each location is specified by an x and y coordinate pair. This dataset has been adapted for the k-median problem by García et al. [15] and they provide solutions for several different k values. There are several tests per problem with k values ranging from 5 to 5000. Note that due to the size of the problems the solver of García et al. was unable to provide an optimal solution for a small number of test cases and as a result some k values are not included in the test results.

We report the approximation ratio and total runtime in seconds for each test.

B.5.1 Hopfield Network and Local Search Results

Test	N	K	Hopfield (1 Run)		Hopfield (10 Runs)		Local Search (1s)		Local Search (5s)		Local Search (15s)	
			Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time
rl1304	1304	5	1.44	1.861	1.312	12.883	1.411	1.002	1.334	5.001	1.271	15.001
		10	1.39	1.705	1.346	13.079	1.372	1.001	1.333	5.001	1.325	15.001
		20	1.387	1.746	1.377	11.236	2.136	1.002	2.054	5.001	1.525	15.002
		50	1.341	1.682	1.366	14.198	1.761	1.002	1.707	5.002	1.605	15.002
		100	1.258	1.798	1.233	15.72	1.841	1.001	1.831	5.001	1.817	15.002
		200	1.256	1.697	1.256	15.894	1.891	1.001	1.88	5.002	1.852	15.001
		300	1.32	1.91	1.302	15.817	2.104	1.001	2.099	5.001	2.06	15.001
		400	1.352	1.695	1.334	14.418	2.245	1.001	2.221	5.001	2.164	15.001
		500	1.399	2.04	1.362	14.03	2.097	1.001	2.082	5.001	2.071	15.001
fl1400	1400	5	1.823	1.605	1.779	11.427	2.692	1.002	2.202	5.003	2.082	15.002
		10	1.404	1.722	1.396	13.578	1.787	1.002	1.698	5.001	1.601	15.001
		20	1.954	2.009	1.887	7.367	1.933	1.001	1.918	5.001	1.837	15.001
		50	1.971	1.739	1.971	6.321	2.516	1.002	2.444	5.001	2.424	15.002
		100	2.564	1.878	2.454	12.526	2.796	1.001	2.735	5.002	2.712	15.001
		200	2.419	1.443	2.309	13.556	4.03	1.001	3.907	5.002	3.526	15.001
		400	1.746	1.557	1.709	14.199	3.792	1.002	3.435	5.001	3.024	15.002
u1432	1432	5	1.333	1.875	1.341	11.842	1.616	1.001	1.512	5.002	1.412	15.003
		10	1.33	2.074	1.365	14.626	1.57	1.002	1.543	5.003	1.537	15.001
		20	1.372	2.093	1.375	15.842	1.617	1.002	1.579	5.002	1.536	15.001
		50	1.348	2.621	1.335	16.579	1.625	1.001	1.621	5.001	1.619	15.002
		100	1.323	2.43	1.313	19.818	1.678	1.002	1.67	5.003	1.665	15.002
		200	1.281	2.328	1.29	18.122	1.715	1.002	1.713	5.002	1.707	15.003
		300	1.25	2.109	1.231	20.228	1.62	1.002	1.617	5.002	1.611	15.002
		500	1.109	2.442	1.085	19.875	1.403	1.002	1.402	5.001	1.401	15.002
vm1748	1748	10	1.404	3.782	1.309	11.827	1.521	1.004	1.438	5.002	1.373	15.002
		20	1.344	3.148	1.29	11.102	1.461	1.003	1.431	5.001	1.412	15.003
		50	1.311	3.745	1.294	22.266	1.635	1.003	1.622	5.003	1.571	15.003
		100	1.389	3.397	1.367	24.911	1.968	1.004	1.934	5.004	1.917	15.001

Test	N	K	Hopfield (1 Run) Ratio	Hopfield (1 Run) Time	Hopfield (10 Runs) Ratio	Hopfield (10 Runs) Time	Local Search (1s) Ratio	Local Search (1s) Time	Local Search (5s) Ratio	Local Search (5s) Time	Local Search (15s) Ratio	Local Search (15s) Time
		200	1.446	3.229	1.42	23.43	1.864	1.003	1.815	5.004	1.766	15.003
		300	1.501	3.064	1.482	24.768	1.963	1.003	1.91	5.003	1.859	15.004
		400	1.515	3.029	1.495	25.673	1.968	1.003	1.926	5.003	1.882	15.004
		500	1.471	3.001	1.451	26.636	2.046	1.003	2.025	5.003	1.985	15.003

B.5.2 Hopfield Network usa13509 Results

Test	N	K	Hopfield (1 Run) Ratio	Hopfield (1 Run) Time	Hopfield (10 runs) Ratio	Hopfield (10 runs) Time
usa13509	13509	300	1.50	838	1.47	6583
		400	1.51	831	1.48	6521
		500	1.51	830	1.49	3291
		800	1.53	825	1.52	5408
		1000	1.54	812	1.52	5448
		2000	1.55	785	1.55	6001
		3000	1.54	771	1.54	6614
		4000	1.63	750	1.63	5921
		5000	1.72	774	1.70	6261

Curriculum Vitae

Name:	Cody Rossiter
Post-Secondary Education and Degrees:	Simon Fraser University Burnaby, BC 2013 - 2016 BSc University of Western Ontario London, ON 2020 - 2023 MSc
Related Work Experience:	Teaching Assistant The University of Western Ontario 2020 - 2022