This page is here to make the page numbers come out correctly.

Do not print this page.

Derivation of Consistent Pairwise Matrices

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science

by

Christopher Kuske

April 5, 2018

Signature page for the Masters in Computer Science Thesis of Christopher Kuske

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

Dr. Michael Soltys, Thesis Advisor   Date

Dr. Konrad Kulakowski, Thesis Committee   Date

Dr. Pawel Pilarczyk, Thesis Committee   Date

APPROVED FOR THE UNIVERSITY

Dr. Joseph Shapiro, AVP Extended University   Date

Acknowledgements

To my wife Kendra and my children Evan and Emma, in gratitude for their unending encouragement and support while developing this thesis. Without them, I would have not been able to reach this goal that I had set for myself. I would like to extend special thanks to Dr. Michael Soltys for his knowledge, patience, and encouragement as my advisor.

Abstract

Derivation of Consistent Pairwise Matrices

by Christopher Kuske

A method of generating a consistent Pairwise Comparison Matrix from an inconsistent matrix will be presented, a proposal for defining "closeness" between matrices will be discussed, and finally, various methods will be examined to find consistent matrices that are as close to the original inconsistent matrix as possible using a calculated distance described within.

This thesis will propose several algorithms, compare their performance, and examine their respective merits.

Contents

List of Figures

1. Introduction

Technology has given society a wide array of choices, whether those choices are concerned with the selection of material goods or consideration of different ideas and methods for solving different problems. One method of arriving at the final decision can be made by ranking the different choices in question. The concepts behind each of these trade-offs are called *criteria*.

Over the past few decades, several methodologies have been used more and more frequently to help decision makers in the evaluation of multiple criteria. *Pairwise Comparisons* (*PC*) and the *Analytical Hierarchical Process* (*AHP*) have given decision makers a new set of tools that empower them to make more informed decisions. AHP uses Pairwise Comparisons to help rank the evaluation criteria based on importance. This type of problem is also known as *Multiple Attribute Decision Making (MADM).*

The method of "Pairwise Comparisons" (PC) has a surprisingly old history for a method that is not widely known outside certain circles in society. The beginnings of PC are attributed to Ramon Llull (see [3]) then further popularized by the Marquis de Condorcet (see [1], written four years before the French Revolution, and nine years before losing his head to the same [10]). Condorcet applied the PC method to analyzing voting outcomes, where the choice was binary (win/lose situations). Almost a century and a half later, Thurstone [11] refined the method and employed a psychological continuum with the scale values as the medians of the distributions of judgments[9].

1.1. **Contributions.**

(1) Development of new algorithms that arrive to a "closer" distance between matrices faster than generating random consistent matrices. These new algorithms can also be adaptive with respect to the effort (in time and computation) that the user wishes the computer to expend on the problem. This thesis will describe the concept of each algorithm, details regarding the implementation of each algorithm, and how it was tested.

(2) Performed analysis of results from the algorithm that provides recommendations to a user of the algorithm concerning parameter values to use in proportion to the size of the matrix.

(3) Investigated alternate algorithms, and demonstrated how the proposed algorithm is a good compromise between accuracy (smallest distance between M and M') and speed.

1.2. **Pertinent Concepts.**

1.2.1. *Pairwise Matrices*

In pairwise matrices, the matrix is always *square*. Additionally, when defining a pairwise matrix each item in the matrix has a relative rank that is considered against another item.

Consider the following pairwise matrix:

$$\begin{array}{c c} & \begin{array}{ccc} \mathit{Apple} & \mathit{Banana} & \mathit{Cherry} \end{array} \\ \begin{array}{c} \mathit{Apple} \\ \mathit{Banana} \\ \mathit{Cherry} \end{array} & \begin{bmatrix} 1 & 2 & 10 \\ 1/2 & 1 & 5 \\ 1/10 & 1/5 & 1 \end{bmatrix} \end{array}$$

The elements above the diagonal in the matrix are the items that are being considered for evaluation. Using this matrix as an example, the first line indicates that bananas are preferred two times over apples, and cherries are preferred ten times over apples. Finally, bananas are preferred five times over cherries.

When decision makers are trying to make their evaluation(s), they will often bring in subject matter experts to help develop the relative rankings of how one preference should be ranked compared to another. When these experts define their preferences in a PC matrix, they often generate matrices that do not meet the criteria for consistency for one reason or another. Consistency will be discussed in detail in Section 1.2.3, but it is defined as follows: $\forall ijk, a_{ij} = a_{ik} * a_{kj}$, i.e, the ranking is internally coherent.

Pairwise Comparison matrices can sometimes be involved in situations where the correct decision can mean life or death for someone. Consider a Pairwise Comparison matrix that considers different criteria on whether a doctor should operate on a patient or not. Rather than relying on a "gut" feel, pairwise comparisons help the situation by introducing a process and strategy for arriving at a decision. The decision may involve many factors that need to be considered (weighted) against each other to arrive at a decision. In the situation given, there may be many factors involved that need to be distilled down to a simple "yes", "no", or "maybe".

When generating matrices, the table on the following page can help guide on how these comparisons should be weighted:

Intensity of Importance	Definition	Explanation
1	Equal importance	Two activities contribute equally to the objective
3	Weak importance of one over another	Experience and judgment slightly favor one activity over another
5	Essential or strong importance	Experience and judgment strongly favor one activity over another
7	Demonstrated importance	An activity is strongly favored and its dominance is demonstrated in practice.
9	Absolute importance	The evidence favoring one activity over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between the two adjacent judgments	When compromise is needed

This table of values was created by Saaty in his seminal 1977 work[8].

1.2.2. *Properties of Pairwise Matrices*

A pairwise comparison matrix M has the following properties that must be present, even if the matrix does not meet the additional criteria for consistency listed in Section 1.2.3:

(1) M is *square* (equal number of n rows and n columns).

(2) All elements on the diagonal of M have a value of 1.

(3) M has the property where each element a_{ij} has an element that is the *reciprocal*, located at a_{ji} as shown below:

$$\begin{bmatrix} 1 & a_{12} & a_{13} & a_{1n} \\ \frac{1}{a_{21}} & 1 & a_{23} & a_{2n} \\ \frac{1}{a_{31}} & \frac{1}{a_{23}} & 1 & a_{3n} \\ \frac{1}{a_{n1}} & \frac{1}{a_{n2}} & \frac{1}{a_{n3}} & 1 \end{bmatrix}$$

1.2.3. *Consistency in Pairwise Matrices*

For a matrix to be considered consistent, the following condition must be met:

For each element a_{ij} in matrix M, $a_{ij} = a_{ik} * a_{kj}$ must hold true that for all i, j, k in order for the matrix to be considered consistent.

Throughout this thesis, W is defined as a one dimensional vector that is comprised of a sequence of integers.

For example:

$W = [w_1, w_2, ..., w_n]$

The matrix $M = \langle W \rangle$ is a matrix generated by vector W.

$\langle W \rangle = [w_i/w_j]$, i.e., the $(i$,$j)$ entry of $\langle W \rangle$ is w_i divided by w_j.

Claim 1. Given $W \in (\mathbb{R}^+)^n$, if $M = \langle W \rangle$ ($M = \langle W \rangle$ is the matrix generated by the numbers present in set W) then M is reciprocal and consistent.

Proof. For all $i, j \in [n]$, $a_{ij} = w_i/w_j$

$= 1/(w_j/w_i) = 1/a_{ji}$

Also for all $i, j, k \in [n]$, $a_{ij} = w_i/w_j$

$= (w_i w_k)/(w_j w_k)$

$= (w_i/w_k)(w_k/w_j)$

$= a_{ik}a_{kj}$.

$\square$

2. Literature Review

Modern PC can be said to have started with the work of Saaty in 1977 [8], who proposed a finite nine-point scale of measurements. Furthermore, Saaty introduced the *Analytic Hierarchy Process* (AHP), which is a formal method to derive ranking orders from numerical pairwise comparisons. AHP is widely used around the world for decision making, in education, industry, government, etc. Koczkodaj's [6] proposed smaller five-point scale, which is less fine-grained than Saaty's nine-point scale, is less precise but easier to use.

Note that while AHP is a respectable tool for practical applications, it is nevertheless considered by many [2] as a less-than-perfect procedure that yields arbitrary rankings. The belief is that the shortcomings of AHP arise from the following two observations [5]:

(1) The final outcome is forced to be totally ordered, which might be too strong a requirement;
(2) numbers, whose assignment is very subjective, are assigned to all items to calculate the final outcome.

.

It is also important to note that AHP uses a fixed scale that makes it a subset of Pairwise Comparisons. Pairwise Comparisons allow for a non-numeric ranking system. In fact, it does not assume a particular scale at all in contrast to AHP.

3. Methodology

3.1. **Introduction.** To investigate the properties of both consistent and inconsistent matrices, several new algorithms were developed. The problem we wish to solve is that of approximating a reciprocal matrix (which may or may not be consistent) with a consistent matrix. That is, given the matrix M is reciprocal, we want to find a consistent matrix M' so that the distance between M and M' is minimized.

Additionally, a measure of determining the "closeness" between inconsistent matrix M and consistent matrix M' as an index of inconsistency is proposed.

Each algorithm also takes advantage of the fact that a consistent pairwise matrix can be constructed from any row or column of an inconsistent matrix. Stated more formally:

For an inconsistent matrix M, a consistent matrix M' can be constructed by generating a set of sequence entries W (also called a vector of weights), where W is comprised of the elements of any given row/column in M.

Given this goal, each algorithm takes a different approach to arrive at M' where M' has the smallest possible distance between M and itself. Each approach has trade-offs concerning the computed distance between M and M' and the computational cost required to arrive at M'. This section of the paper will describe the operation of each algorithm. For analysis of the results for each algorithm, please see Section 4.

Of course, it is important to note that a consistent pairwise matrix that was initially inconsistent is always more meaningful if the initial degree of inconsistency is under a certain amount. When the largest inconsistencies in a matrix are brought under control, the subsequent consistent matrix can be of greater utility.

In other words, the inconsistency measure (however that measure is defined) of a PC matrix is the measure of the quality of knowledge [4].

3.2. **Formation of Consistent Matrices.** If M is consistent, any row or column of M may be selected such that:

$[w_1, w_2, ..., w_n] = [a_{11},\ a_{21},\ a_{31},\ ...,\ a_{n1}]$ using matrix a where n is the size of the matrix.

Using Saaty's seminal work, the most often used definition of consistency in pairwise matrices is as follows:

A pairwise comparison matrix A is consistent if and only if there exists a vector $[w_1, w_2, ..., w_n]$ such that $a_{ij} = w_i/w_j$.[8]

Claim 2. Suppose that M is reciprocal and consistent. Then $M = \langle W \rangle$, where W is any row or any column of M.

Proof. Suppose $W = [a_{1k}, a_{2k}, \ldots, a_{nk}]$ where $M = [a_{ij}]$, that is, W is the k-th column of M. Then the i, j entry of $\langle W \rangle$ is $w_i/w_j = a_{ik}/a_{jk} = a_{ik}a_{kj} = a_{ij}$, where we used reciprocity and consistency of M. Note that M is reciprocal and consistent if and only if M^T (the transpose of M) is reciprocal and consistent, and so the claim follows for rows as well. $\square$

3.3. **Computation of "Distance".** In this paper, distance is used as a concept of the total difference between two matrices (M and M'). The distance between M and M' is computed by taking an element at the same position in M and M', and subtracting them. The absolute value is added to the total distance (which is initially zero). When this computation has taken place for each position in the matrices, the total distance has been calculated.

More formally, the computation of distance between two matrices can be stated as follows:

For both the upper triangle of the matrix ($i < j$) and the lower triangle ($i > j$):

$$d(M, M') = \sum_{i<j} \max\{|a_{ij} - a'_{ij}|, |a_{ji} - a'_{ji}|\},$$

3.3.1. *Distance Example*

The required work to calculate the distance between M and M' is shown below (using items except those on the diagonal):

$$\begin{bmatrix} 1 & 2 & 10 \\ \frac{1}{2} & 1 & 5 \\ \frac{1}{10} & \frac{1}{5} & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ \frac{1}{2} & 1 & 5 \\ \frac{1}{3} & \frac{1}{5} & 1 \end{bmatrix}$$

Distance of Upper Triangle $= |2 - 2| + |10 - 3| + |5 - 5| = \mathbf{7}$ $(\mathbf{7.00})$

Distance of Lower Triangle $= |\frac{1}{5} - \frac{1}{5}| + |\frac{1}{10} - \frac{1}{3}| + |\frac{1}{2} - \frac{1}{2}| = \frac{7}{30}(.23333)$

The final calculated distance for a given matrix is the higher of the two values (comparing the upper triangle distance to the lower triangle distance), and this maximum value is noted in **bold** throughout the rest of this document.

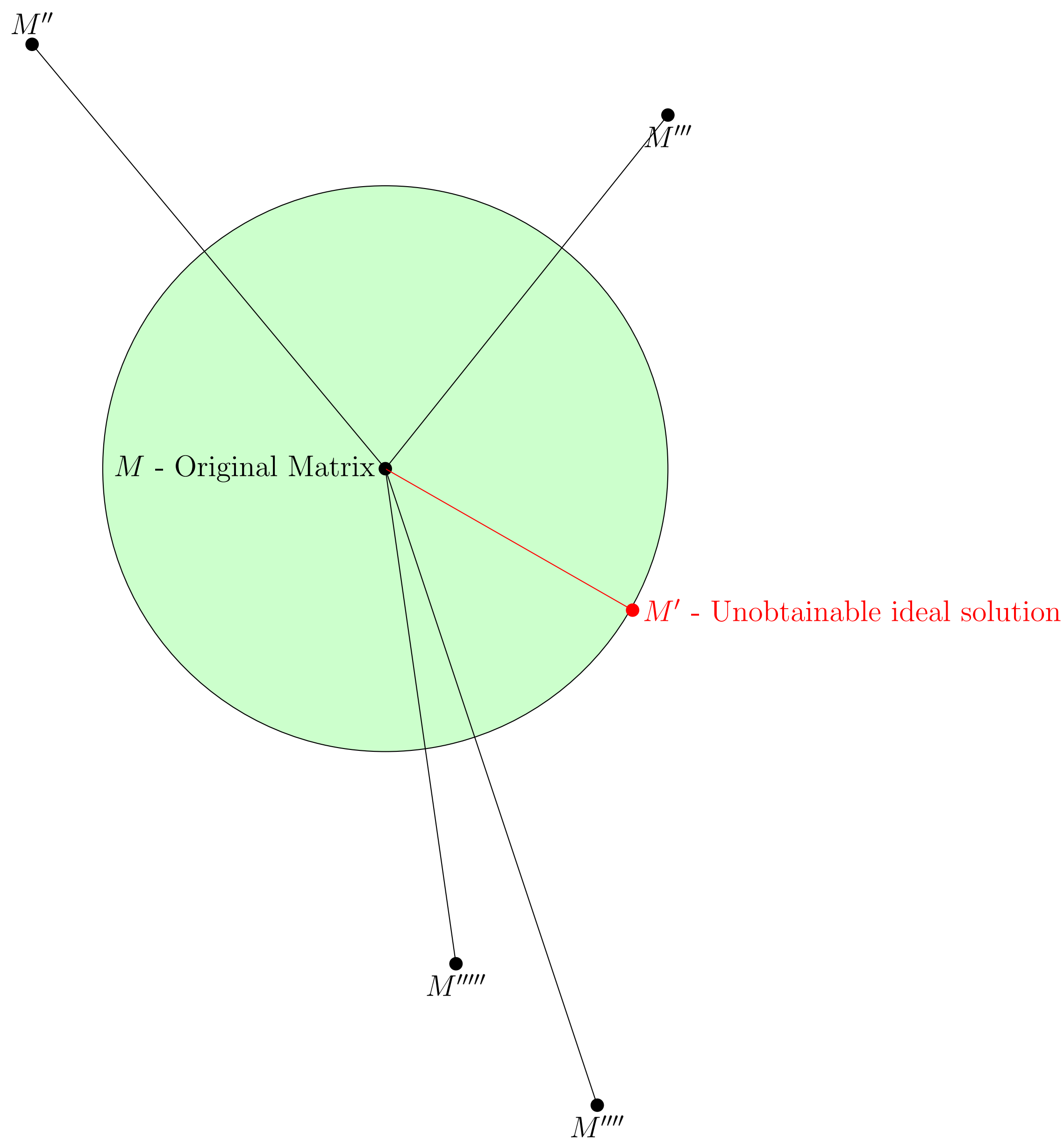
In this case, the distance between M and M' is seven, as the distance between the upper triangle of M and M' is greater than the distance between the lower triangles of M and M'.

3.4. **General Algorithm Philosophy.** The algorithms that are evaluated in the following sections of this paper are of the same general nature. They are intended to assist in the finding of consistent matrices where the matrix size (n) is fairly limited ($n \leq 20$). With matrices that have a size greater than 20, humans cannot accurately consider that many factors when making a decision. Therefore, when $n > 20$, the risk of accumulating large errors in the pairwise comparisons increases.

The algorithms proposed do not anticipate large variance in the pairwise comparisons present in the PC matrix. To attempt to rectify these large differences can have a 'ripple effect' on the matrix under examination as each element in the matrix has a relationship with the elements around it. In these situations, resolving the variance with human input is the most pragmatic approach because the variance is most likely not intentional.

Additionally, the proposed algorithms are trying to find **approximate** solutions. There is currently no known way to reliably and deterministically find the *ideal* solution for any given matrix where the number of inconsistencies and size of the matrix itself is non-trivial (in other words, the inconsistencies cannot be resolved with simple trial and error). It is important to note that the proposed measure of distance between M and M' is only one way to measure the quality of a solution, and that proposed measure is the basis for the algorithms in this thesis. The algorithms shown in the rest of this document make a trade-off of accuracy versus time to arrive at a solution, among other things.

Consider the drawing below. It represents a scenario where there is an ideal solution M' where the distance between original matrix M is as small as is possible. It also includes matrices M'', M''', M'''', and M''''' that represent Algorithms 1, 2, 3, and 4 in this thesis (but not representative of actual results).



Ideal solution M' vs. approximate solutions of varying accuracy.

3.5. **Algorithm 1.**

Require: m - a PC matrix given as input to the algorithm

```
y ← 0
sz ← m.length
worstSolutionDistance ← 0
bestSolutionDistance ← sys.maxsize
bestSolution - holds the M′ with the “best” distance
worstSolution - holds the M′ with the “worst” distance
mPrime - PC matrix derived from M
while y < sz do
    columnData = getColumn(y)
    mPrime ← generateConsistentMatrix(columnData)
    resultsDistance ← mPrime.getDistance(m)
    calculatedDistance = resultsDistance
    if calculatedDistance < bestSolutionDistance then
        bestSolutionDistance ← calculatedDistance
        bestSolution ← mPrime
    else
        worstSolutionDistance ← calculatedDistance
        worstSolution ← mPrime
    end if
    y ← y + 1
end while
```

3.5.1. *Definition*

Algorithm 1 is defined as follows:

For a given square matrix M of size n, Algorithm 1 will use each individual column of M and use it to generate new consistent matrix M'.

Using the columns of M, a new consistent matrix M' is generated and compared against matrix M. Then, the computed distance between M and M' is stored. As each column is used, if the computed distance of the new matrix is lower than any other previously computed distance, that lesser distance is saved as the "best" matrix solution. If a future iteration is better than the last, the best solution is updated/replaced.

3.5.2. *Example matrices*

$$\begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix}$$

$W_1 = [1\ \frac{1}{2}\ \frac{1}{6}\ \frac{1}{6}]$

$W_2 = [2\ 1\ \frac{1}{5}\ \frac{1}{4}]$

$W_3 = [6\ 5\ 1\ \frac{1}{2}]$

$W_4 = [6\ 4\ 2\ 1]$

The resulting matrices generated from W_1, W_2, W_3, W_4 are below:

$$< W_1 > = \begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 3 & 3 \\ \frac{1}{6} & \frac{1}{3} & 1 & 1 \\ \frac{1}{6} & \frac{1}{3} & 1 & 1 \end{bmatrix}$$

Computed distance between M and $< W_1 >$:

$[2-2] + [6-6] + [6-6] + [5-3] + [4-3] + [2-1] = \mathbf{4}\ (\mathbf{4.0})$

$$[\tfrac{1}{2} - 1] + [\tfrac{1}{4} - \tfrac{1}{3}] + [\tfrac{1}{6} - \tfrac{1}{6}] + [\tfrac{1}{5} - \tfrac{1}{3}] + [\tfrac{1}{6} - \tfrac{1}{6}] + [\tfrac{1}{2} - \tfrac{1}{2}] = \tfrac{43}{60}\ (0.716)$$

$$< W_2 > = \begin{bmatrix} 1 & 2 & 10 & 8 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{10} & \frac{1}{5} & 1 & \frac{4}{5} \\ \frac{1}{8} & \frac{1}{4} & \frac{5}{4} & 1 \end{bmatrix}$$

Computed distance between M and $< W_2 >$:

$[2-2] + [6-10] + [6-8] + [5-5] + [4-4] + [2-\frac{4}{5}] = \mathbf{6\frac{1}{5}}\ (\mathbf{6.20})$

$$[\frac{1}{2}-\frac{5}{4}]+[\frac{1}{4}-\frac{1}{4}]+[\frac{1}{6}-\frac{1}{8}]+[\frac{1}{5}-\frac{1}{5}]+[\frac{1}{6}-\frac{1}{10}]+[\frac{1}{2}-\frac{1}{2}]=\frac{43}{120}\ (0.358)$$

$$< W_3 > = \begin{bmatrix} 1 & \frac{6}{5} & 6 & 12 \\ \frac{5}{6} & 1 & 5 & 10 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{12} & \frac{1}{10} & \frac{1}{2} & 1 \end{bmatrix}$$

Computed distance between M and $< W_3 >$:

$$[2-\frac{6}{5}]+[6-6]+[6-12]+[5-5]+[4-10]+[2-2]=\mathbf{12\frac{4}{5}}\ (\mathbf{12.8})$$

$$[\frac{1}{2}-\frac{1}{2}]+[\frac{1}{4}-\frac{1}{10}]+[\frac{1}{6}-\frac{1}{12}]+[\frac{1}{5}-\frac{1}{5}]+[\frac{1}{6}-\frac{1}{6}]+[\frac{1}{2}-\frac{5}{6}]=\frac{17}{30}\ (0.566)$$

$$< W_4 > = \begin{bmatrix} 1 & \frac{3}{2} & 3 & 6 \\ \frac{2}{3} & 1 & 2 & 4 \\ \frac{1}{3} & \frac{1}{2} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix}$$

Computed distance between M and $< W_4 >$:

$$[2-\frac{3}{2}]+[6-3]+[6-6]+[5-2]+[4-4]+[2-2]=\mathbf{6\frac{1}{2}}\ (\mathbf{6.5})$$

$$[\frac{1}{2}-\frac{1}{2}]+[\frac{1}{4}-\frac{1}{4}]+[\frac{1}{6}-\frac{1}{6}]+[\frac{1}{5}-\frac{1}{2}]+[\frac{1}{6}-\frac{1}{3}]+[\frac{1}{2}-\frac{2}{3}]=\frac{19}{30}\ (0.633)$$

3.6. **Algorithm 2.**

3.6.1. *Definition*

For a given square matrix M of size n, Algorithm 2 will use a subset ('span') of items in (defined as an input parameter into the algorithm) each individual column of M (moving left to right) and use it to generate a new matrix $< W >$. During the execution of this algorithm, a span value of $sizeof(M)/3$ was used. For example if the size of M is 12, the 'span' used will be 4. This heuristic was chosen because as the algorithm is applied to larger matrices, it takes more and more columns into account for use in the algorithm without incurring a large time penalty. The span parameter will cause more operations to occur to generate W, which in turn generates M'.

With this W using $sizeof(M)/3$ as the span heuristic, a new consistent matrix M' is generated and compared against matrix M and the distance from M is stored. As each column is used, if the computed distance of the new matrix is lower than any other previously computed distance, that particular M' with the lesser distance is saved as the "best" matrix.

Require: m - a PC matrix given as input to the algorithm

$sz \leftarrow m.length$

$i, y \leftarrow 0$

$bestSolutionDistance \leftarrow 0$

$span \leftarrow$ - defaults to sizeof(M)/3

$bestSolution$ - holds the M' with the "best" distance

$worstSolution$ - holds the M' with the "worst" distance

$mPrime$ - PC matrix derived from M

$colsData \leftarrow \emptyset$

while $y < sz$ **do**

  $rowData = getRow(y)$

  $currentAnswer \leftarrow rowData[0]$

  while $i < span$ **do**

    $currentAnswer \leftarrow currentAnswer * rowData[i]$

    $colsData[y] \leftarrow currentAnswer$

    $i \leftarrow i + 1$

  end while

  $mPrime \leftarrow generateConsistentMatrix(colsData)$

  $resultsDistance \leftarrow mPrime.getDistance(m)$

  $calculatedDistance = resultsDistance[0]$

if $calculatedDistance < bestSolutionDistance$ **then**

$bestSolutionDistance \leftarrow calculatedDistance$

$bestSolution \leftarrow mPrime$

else

$worstSolutionDistance \leftarrow calculatedDistance$

$worstSolution \leftarrow mPrime$

end if

$y \leftarrow y + 1$

end while

3.6.2. *Example matrices*

$$\begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix}$$

The calculated span for this matrix is one, so $W_1 = [1\ \frac{1}{2}\ \frac{1}{6}\ \frac{1}{6}]$

The resulting matrix generated from W_1 is:

$< W_1 >$ matrix:

$$\begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 3 & 3 \\ \frac{1}{6} & \frac{1}{3} & 1 & 1 \\ \frac{1}{6} & \frac{1}{3} & 1 & 1 \end{bmatrix}$$

$$[2-2]+[6-6]+[6-6]+[5-3]+[4-3]+[2-1] = \mathbf{4}\ (\mathbf{4.0})$$

$$[\tfrac{1}{2}-1]+[\tfrac{1}{4}-\tfrac{1}{3}]+[\tfrac{1}{6}-\tfrac{1}{6}]+[\tfrac{1}{5}-\tfrac{1}{3}]+[\tfrac{1}{6}-\tfrac{1}{6}]+[\tfrac{1}{2}-\tfrac{1}{2}] = \tfrac{43}{60}\ (.71667)$$

If span had been specified as two, W_1 (let us now call it W_2) would be as follows:

$$\left. \begin{matrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{matrix} \right]$$

$W_2 = [2\ \frac{1}{2}\ \frac{1}{30}\ \frac{1}{24}]$

$< W_2 >$ matrix:

$$\begin{bmatrix} 1 & 4 & 60 & 48 \\ \frac{1}{4} & 1 & 15 & 12 \\ \frac{1}{60} & \frac{1}{15} & 1 & \frac{4}{5} \\ \frac{1}{48} & \frac{1}{12} & \frac{5}{4} & 1 \end{bmatrix}$$

$$[2-4]+[6-60]+[6-48]+[5-15]+[4-12]+[2-\tfrac{4}{5}] = \mathbf{117\tfrac{1}{5}}\ (\mathbf{117.20})$$

$$[\tfrac{1}{2}-\tfrac{5}{4}]+[\tfrac{1}{4}-\tfrac{1}{12}]+[\tfrac{1}{6}-\tfrac{1}{48}]+[\tfrac{1}{5}-\tfrac{1}{15}]+[\tfrac{1}{6}-\tfrac{1}{60}]+[\tfrac{1}{2}-\tfrac{1}{4}] = 1\tfrac{143}{240}\ (1.5958)$$

3.7. **Algorithm 3.**

3.7.1. *Definition*

$bestSolutionDistance \leftarrow 0$

$worstSolutionDistance \leftarrow 0$

$calculatedDistance \leftarrow 0$

$i, j, k \leftarrow 0$

$result \leftarrow 1$

$bestSolution$ - holds the M' with the "best" distance

$worstSolution$ - holds the M' with the "worst" distance

$allData \leftarrow \emptyset$

$w \leftarrow \emptyset$

$rowColCombos \leftarrow$ list of combinations generated (omitted for brevity)

$matrixSize \leftarrow sizeofdesiredmatrix$

m - original matrix M' is derived from

$mPrime$ - PC matrix derived from M

---

while $i < matrixSize$ **do**

$sizeofThisPairing \leftarrow len(rowColCombos[i])$

while $j < matrixSize$ **do**

$allData.append(rowColCombos[i][j])$

$j \leftarrow j + 1$

end while

$j \leftarrow 0$

while $j < matrixSize$ **do**

$result \leftarrow 1$

while $k < gridMultiplier$ **do**

$result = result * allData[k][j]$

$k \leftarrow k + 1$

end while

$w.append(result)$

$j \leftarrow j + 1$

end while

$mPrime \leftarrow generateConsistentMatrix(w)$

$resultsDistance \leftarrow mPrime.getDistance(m)$

$calculatedDistance = resultsDistance$

if $calculatedDistance < bestSolutionDistance$ **then**

$bestSolutionDistance \leftarrow calculatedDistance$

$bestSolution \leftarrow mPrime$

else

$worstSolutionDistance \leftarrow calculatedDistance$

$worstSolution \leftarrow mPrime$

end if

$i \leftarrow i + 1$

end while

For a given square matrix M of size n, Algorithm 3 will use a subset ('span') of items in (defined as a input parameter into the algorithm) each individual column of M and use it to populate an entry in sequence W. The span determines the number of column combinations that are generated. The smaller the span, the more combinations that are generated. This approach is different than Algorithm 2 in that every combination of columns (given a specific span) will be used. This ensures that no potential "best" solutions are missed as they might possibly be using Algorithm 2.

The span parameter is dynamic, based on the current size of the matrix being examined. The value of span is defined as follows:

$$span = sizeof(M)/n$$

With n set to 3, the number of matrices with respect to the span increases in a roughly exponential fashion as shown in the following table. For a graphical representation, see Section 4.6.3.

Matrix Size	Span	Matrices Evaluated Per Matrix Size
10	3	120
11	3	165
12	4	495
13	4	715
14	4	1001
15	5	3003
16	5	4368
17	5	6188
18	6	18564
19	6	27132
20	6	38760
21	7	116280
22	7	170544
23	7	245157
24	8	735471
25	8	1081575

Table 1. Algorithm 3 Matrix Relationships

3.7.2. *Example matrices using column combinations*

Span = 2

$$\begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{16} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 6 & 6 \\ \frac{1}{2} & 1 & 5 & 4 \\ \frac{1}{6} & \frac{1}{5} & 1 & 2 \\ \frac{1}{6} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix}$$

$W_1 = [2 \; \frac{1}{2} \; \frac{1}{80} \; \frac{1}{24}]$

$W_2 = [6 \; \frac{5}{2} \; \frac{1}{6} \; \frac{1}{12}]$

$W_3 = [6 \; 2 \; 3 \; \frac{1}{3}]$

$W_4 = [12 \; 5 \; \frac{1}{5} \; \frac{1}{8}]$

$W_5 = [12 \; 4 \; \frac{2}{5} \; \frac{1}{4}]$

$W_6 = [36 \; 20 \; 2 \; \frac{1}{2}]$

The resulting matrices generated from $W_1, W_2, W_3, W_4, W_5, W_6$ are below:

$$< W_1 > = \begin{bmatrix} 1 & 4 & 160 & 48 \\ \frac{1}{4} & 1 & 40 & 12 \\ \frac{1}{160} & \frac{1}{40} & 1 & \frac{3}{10} \\ \frac{1}{48} & \frac{1}{12} & \frac{10}{3} & 1 \end{bmatrix}$$

Computed distance between M and $< W_1 >$:

$[2-4]+[6-160]+[6-48]+[5-40]+[4-12]+[2-\frac{3}{10}] = \mathbf{242\frac{7}{10}}$ $(\mathbf{242.7})$

$$[\tfrac{1}{2}-\tfrac{10}{3}]+[\tfrac{1}{4}-\tfrac{1}{12}]+[\tfrac{1}{6}-\tfrac{1}{48}]+[\tfrac{1}{5}-\tfrac{1}{40}]+[\tfrac{1}{6}-\tfrac{1}{160}]+[\tfrac{1}{2}-\tfrac{1}{4}] = 3\tfrac{117}{160}\ (3.73125)$$

$$< W_2 > = \begin{bmatrix} 1 & \frac{12}{5} & 36 & 72 \\ \frac{5}{12} & 1 & 15 & 30 \\ \frac{1}{36} & \frac{1}{15} & 1 & 2 \\ \frac{1}{72} & \frac{1}{30} & \frac{1}{2} & 1 \end{bmatrix}$$

Computed distance between M and $< W_2 >$:

$[2-\frac{12}{5}]+[6-36]+[6-72]+[5-15]+[4-30]+[2-2] = \mathbf{130\frac{2}{5}}$ $(\mathbf{130.4})$

$$[\tfrac{1}{2}-\tfrac{1}{2}]+[\tfrac{1}{4}-\tfrac{1}{30}]+[\tfrac{1}{6}-\tfrac{1}{72}]+[\tfrac{1}{5}-\tfrac{1}{15}]+[\tfrac{1}{6}-\tfrac{1}{36}]+[\tfrac{1}{2}-\tfrac{5}{12}] = \tfrac{919}{1260}\ (.72937)$$

$$< W_3 > = \begin{bmatrix} 1 & 3 & 2 & 18 \\ \frac{1}{3} & 1 & \frac{2}{3} & 6 \\ \frac{1}{2} & \frac{3}{2} & 1 & 9 \\ \frac{1}{18} & \frac{1}{6} & \frac{1}{9} & 1 \end{bmatrix}$$

Computed distance between M and $< W_3 >$:

$[2-3]+[6-2]+[6-18]+[5-\frac{2}{3}]+[4-6]+[2-9] = \mathbf{30\frac{1}{3}}\ (\mathbf{30.33333})$

$[\frac{1}{2}-\frac{1}{9}]+[\frac{1}{4}-\frac{1}{6}]+[\frac{1}{6}-\frac{1}{18}]+[\frac{1}{5}-\frac{3}{2}]+[\frac{1}{6}-\frac{1}{2}]+[\frac{1}{2}-\frac{1}{3}] = 2\frac{23}{60}\ (2.38333)$

$$< W_4 > = \begin{bmatrix} 1 & \frac{12}{5} & 60 & 96 \\ \frac{5}{12} & 1 & 25 & 40 \\ \frac{1}{60} & \frac{1}{25} & 1 & \frac{8}{5} \\ \frac{1}{96} & \frac{1}{40} & \frac{5}{8} & 1 \end{bmatrix}$$

Computed distance between M and $< W_4 >$:

$[2-\frac{12}{5}]+[6-60]+[6-96]+[5-25]+[4-40]+[2-\frac{8}{5}] = \mathbf{152\frac{4}{5}}\ (\mathbf{152.8})$

$[\frac{1}{2}-\frac{5}{8}]+[\frac{1}{4}-\frac{1}{40}]+[\frac{1}{6}-\frac{1}{96}]+[\frac{1}{5}-\frac{1}{25}]+[\frac{1}{6}-\frac{1}{60}]+[\frac{1}{2}-\frac{5}{12}] = \frac{2159}{2400}\ (.89958)$

$$< W_5 > = \begin{bmatrix} 1 & 3 & 30 & 48 \\ \frac{1}{3} & 1 & 10 & 16 \\ \frac{1}{30} & \frac{1}{10} & 1 & \frac{8}{5} \\ \frac{1}{48} & \frac{1}{16} & \frac{5}{8} & 1 \end{bmatrix}$$

Computed distance between M and $< W_5 >$:

$[2-3]+[6-30]+[6-48]+[5-10]+[4-16]+[2-\frac{8}{5}] = 84\frac{2}{5}$ (84.4)

$$[\tfrac{1}{2}-\tfrac{5}{8}]+[\tfrac{1}{4}-\tfrac{1}{16}]+[\tfrac{1}{6}-\tfrac{1}{48}]+[\tfrac{1}{5}-\tfrac{1}{10}]+[\tfrac{1}{6}-\tfrac{1}{30}]+[\tfrac{1}{2}-\tfrac{1}{3}] = \mathbf{105\tfrac{101}{120}}\ (\mathbf{105.84167})$$

$$< W_6 > = \begin{bmatrix} 1 & \frac{9}{5} & 18 & 72 \\ \frac{5}{9} & 1 & 10 & 40 \\ \frac{1}{18} & \frac{1}{10} & 1 & 4 \\ \frac{1}{72} & \frac{1}{40} & \frac{1}{4} & 1 \end{bmatrix}$$

Computed distance between M and $< W_6 >$:

$[2-\frac{9}{5}]+[6-18]+[6-72]+[5-10]+[4-40]+[2-4] = \mathbf{121\frac{1}{5}}$ $(\mathbf{121.2})$

$$[\tfrac{1}{2}-\tfrac{1}{4}]+[\tfrac{1}{4}-\tfrac{1}{40}]+[\tfrac{1}{6}-\tfrac{1}{72}]+[\tfrac{1}{5}-\tfrac{1}{10}]+[\tfrac{1}{6}-\tfrac{1}{18}]+[\tfrac{1}{2}-\tfrac{5}{9}] = \tfrac{161}{180}\ (.89444)$$

3.8. **Algorithm 4.**

3.8.1. *Definition*

$bestSolutionDistance \leftarrow 0$

$worstSolutionDistance \leftarrow 0$

$i \leftarrow 0$

$distanceDelta \leftarrow -1$

$listIdx \leftarrow 0$

$calculatedDistance \leftarrow 0$

$bestSolution$ - holds the M' with the "best" distance

$worstSolution$ - holds the M' with the "worst" distance

$myList \leftarrow$ - set of all values set to 1, length is the same as the size of M

$mPrime$ - PC matrix derived from M

$origValue$ - retained value as the value is modified/tested

$origmPrime$ - M' using the original value at a_{ij}

$amPrime$ - PC matrix derived from M

$bmPrime$ - PC matrix derived from M

$origDistanceResults \leftarrow 0$

$aDistanceResults \leftarrow 0$

$bDistanceResults \leftarrow 0$

---

while $i < math.ceil(largestValue/smallestValue)$ **do**

$distanceDelta \leftarrow -1$

while $listIdx < matrixSize$ **do**

$origValue \leftarrow myList[listIdx]$

$origmPrime \leftarrow generateConsistentMatrix(myList)$

$origDistanceResults \leftarrow origmPrime.getDistance(m)$

$myList[listIdx] \leftarrow myList[listIdx] * 2$

$amPrime \leftarrow generateConsistentMatrix(myList)$

$aDistanceResults \leftarrow amPrime.getDistance(m)$

$myList[listIdx] \leftarrow myList[listIdx]/2$

$bmPrime \leftarrow generateConsistentMatrix(myList)$

$bDistanceResults \leftarrow bmPrime.getDistance(m)$

if $aDistanceResults < bDistanceResults$ **then**

$distanceResults \leftarrow aDistanceResults$

$myList[listIdx] \leftarrow myList[listIdx] * 2$

end if

if $bDistanceResults < origDistanceResults$ **then**

$distanceResults \leftarrow bDistanceResults$

$myList[listIdx] \leftarrow myList[listIdx]/2$

else

$distanceResults \leftarrow origDistanceResults$

$myList[listIdx] \leftarrow origValue$

end if

if $distanceDelta < 0 \parallel (distanceResults < distanceDelta)$ **then**

$distanceDelta \leftarrow distanceResults$

end if

$listIdx \leftarrow listIdx + 1$

end while

$i \leftarrow i + 1$

end while

For a given random square matrix M of size n, Algorithm 4 will start with a matrix using a list named W of size n, where each element in W is initialized to one. With this W, a series of iterations are executed where the iteration count is user defined.

During each iteration, each element a_{ij} in W is modified to be $\frac{a_{ij}}{2}$, and then $a_{ij} * 2$. Upon each modification to a_{ij}, the new matrix M' is tested whether the distance to M is reduced. If the distance between M and M' has indeed been reduced, the value of a_{ij} that resulted in the smaller distance to M is kept and the next element is tested in the same manner. Otherwise, the original value of a_{ij} is retained. This procedure repeats until the last element in W is evaluated.

After the last element in W is evaluated, the procedure repeats again until all of the defined iterations have been exhausted.

3.8.2. *Example matrices*

Random Matrix generated as M:

$$\begin{bmatrix} 1 & 5 & 7 & 2 \\ \frac{1}{5} & 1 & 8 & 9 \\ \frac{1}{7} & \frac{1}{8} & 1 & 7 \\ \frac{1}{2} & \frac{1}{9} & \frac{1}{7} & 1 \end{bmatrix}$$

$W_1 = [1\ 1\ 1\ 1]$

$$< W_1 > = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Computed distance between M and $< W_1 >$:

$[5-1]+[7-1]+[2-1]+[8-1]+[9-1]+[7-1] = \mathbf{32}\ (\mathbf{32.00})$

$[\frac{1}{7}-1]+[\frac{1}{9}-1]+[\frac{1}{2}-1]+[\frac{1}{8}-1]+[\frac{1}{7}-1]+[\frac{1}{5}-1] = 4\frac{435}{559}\ (4.778)$

$W_2 = [2\ 1\ 1\ 1]$

$$< W_2 > = \begin{bmatrix} 1 & 2 & 2 & 2 \\ \frac{1}{2} & 1 & 1 & 1 \\ \frac{1}{2} & 1 & 1 & 1 \\ \frac{1}{2} & 1 & 1 & 1 \end{bmatrix}$$

Computed distance between M and $< W_2 >$:

$[5-2]+[7-2]+[2-2]+[8-2]+[9-1]+[7-1] = \mathbf{29}(\mathbf{29.00})$

$[\frac{1}{7}-1]+[\frac{1}{9}-1]+[\frac{1}{2}-\frac{1}{2}]+[\frac{1}{8}-1]+[\frac{1}{7}-\frac{1}{2}]+[\frac{1}{5}-\frac{1}{2}] = 3\frac{195}{701}$ (3.278125)

$W_3 = [2\ 2\ 1\ 1]$

$$< W_3 > = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ \frac{1}{2} & \frac{1}{2} & 1 & 1 \\ \frac{1}{2} & \frac{1}{2} & 1 & 1 \end{bmatrix}$$

Computed distance between M and $< W_3 >$:

$[5-1]+[7-2]+[2-2]+[8-2]+[9-2]+[7-1] = \mathbf{28}\ (\mathbf{28.00})$

$[\frac{1}{7}-1]+[\frac{1}{9}-\frac{1}{2}]+[\frac{1}{2}-\frac{1}{2}]+[\frac{1}{8}-\frac{1}{2}]+[\frac{1}{7}-\frac{1}{2}]+[\frac{1}{5}-1] = 2\frac{435}{559}$ (2.778175)

3.9. **Algorithmic Computational Complexity.** It is often useful to use "Big O" notation to describe the worst case computational complexity of a particular algorithm. Big O notation describes the worst-case scenario for the algorithm in question, and can be used to describe the execution time required or the space used (in memory or on disk, for example). In this case however, the thesis is most concerned with execution time. It is important to note that the following complexity assessments are based on the worst-case scenario for a given operation.

3.9.1. *Common Code*

`generate_random_matrix` is a Python function that generates a sequence of random numbers between $\frac{1}{n}$ and $\frac{n}{1}$ where the probability of each type of number is $\frac{1}{2}$. This generated sequence is then used to generate a corresponding PC matrix. Due to a nested loop within the code, this function has a complexity of $O(n^2)$.

`generate_consistent_matrix` is a Python function that uses a sequence of integers, and that sequence in then used to generate a corresponding PC matrix. This function has a complexity of $O(n^2)$, due to a nested loop.

`get_distance` is a Python function that calculates the distance between two PC matrices. This function has a complexity of $O(n^2)$ due to a nested loop.

`get_column` is a Python function that gets the set of numbers representing a single column in a PC matrix. This function has a complexity of $O(n)$.

3.9.2. *Algorithm 1*

First, an input matrix is generated, using `generate_random_matrix`. So far, the complexity is $O(n^2)$.

The main loop in this algorithm iterates through each column in the input matrix, meaning the loop itself has a complexity of $O(n)$. Within the main algorithm loop, `get_column`, `generate_consistent_matrix`, and `get_distance` are called. This means the main loop has a algorithmic complexity of $O(2n^2 + n)$.

Finally, "best" matrix is generated from all the iterations of the algorithm as M', and the distance between M and M' is calculated using `generate_consistent_matrix` and `get_distance` respectively. The combined complexity of these two algorithms is $O(2n^2)$.

Adding all of these portions of the algorithm process together, the total number of steps for Algorithm 1 is $5n^2 + n$ which leads to a complexity of $O(n^2)$.

3.9.3. *Algorithm 2*

First, an input matrix is generated, using `generate_random_matrix`. So far, the complexity is $O(n^2)$.

The main loop in this algorithm iterates through each row in the input matrix, meaning the loop itself has a complexity of $O(n)$. Within the main algorithm loop, `get_row` is called to get the values of the PC matrix for the row in question. Then, in a loop, the product of one or more values is calculated for a complexity of $O(n)$ (remembering that the normal amount of columns to use in the product is the total size of the matrix divided by three). Next, `generate_consistent_matrix`, and `get_distance` are called. This means the main loop has a algorithmic complexity of $O(2n^2 + 2n)$.

Finally, "best" matrix is generated from all the iterations of the algorithm as M', and the distance between M and M' is calculated using `generate_consistent_matrix` and `get_distance` respectively. The combined complexity of these two algorithms is $O(2n^2)$.

Adding all of these portions of the algorithm process together, the total number of steps for Algorithm 2 is $5n^2 + 2n$ which leads to a complexity of $O(n^2)$.

3.9.4. *Algorithm 3*

First, an input matrix is generated, using `generate_random_matrix`. So far, the complexity is $O(n^2)$.

Then, the column combinations are calculated where the number of columns involved is bounded by the span heuristic previously defined. The complexity of this operation is $O(n^{(n/2)})$. Adding all of these combinations to our internal list to utilize has a complexity of $O(n)$.

The main loop in this algorithm iterates through the list of column combinations, meaning the main loop itself has a complexity of $O(n^n)$. Within the main algorithm loop, `get_column`, `generate_consistent_matrix`, and `get_distance` are called for complexity of $O(2n^2)$. This means the main loop has a algorithmic complexity of $O(n^{2n^2})$.

Finally, "best" matrix is generated from all the iterations of the algorithm as M', and the distance between M and M' is calculated using `generate_consistent_matrix` and `get_distance` respectively. The combined complexity of these two algorithms is $O(2n^2)$.

Adding all of these portions of the algorithm process together, the total number of steps for Algorithm 3 is $n^{2n^2} + n^{(n/2)} + 4n^2 + n$ which leads to a complexity of $O(n^n)$. This level of complexity makes Algorithm 3 only feasible when n is quite small.

3.9.5. *Algorithm 4*

First, an input matrix is generated, using `generate_random_matrix`. So far, the complexity is $O(n^2)$. Next, the matrix is scanned to determine the number of times the outer algorithm loop should be executed (ceil of the division of the largest value within M by the smallest value within M). The complexity of this operation is also $O(n^2)$.

The outer loop in this algorithm loops through each column in the input matrix, meaning the loop itself has a complexity of $O(n)$. The inner loop iterates through each element in matrix M'. This means the complexity of this inner loop is also $O(n)$. Within the inner algorithm loop, `get_column`, `generate_consistent_matrix`, and `get_distance` are called. The operations within this inner loop have a algorithmic complexity of $O(2n^2)$. The combined complexity of the loops and operations within them is $O(2n^2 + 2n)$.

Finally, "best" matrix is generated from all the iterations of the algorithm as M', and the distance between M and M' is calculated using `generate_consistent_matrix` and `get_distance` respectively. The combined complexity of these two algorithms is $O(2n^2)$.

Adding all of these portions of the algorithm process together, the total number of steps for Algorithm 4 is $6n^2 + 2n$ which leads to a complexity of $O(n^2)$.

3.9.6. *Algorithm Complexity Summary*

Name	Complexity
Algorithm 1	$O(n^2)$
Algorithm 2	$O(n^2)$
Algorithm 3	$O(n^n)$
Algorithm 4	$O(n^2)$

TABLE 2. Algorithm Performance Summary

3.10. **Algorithms Summary.** To provide the reader with an "at-a-glance" summary of the different algorithms, the following summary is provided:

(1) Algorithm 1: For a matrix of size n, Algorithm 1 generates n matrices, where each matrix is based on one individual column of matrix values. Of the resulting matrices (collectively called M'), the M' with the least distance from M is kept.
(2) Algorithm 2: Similar to Algorithm 1, except each candidate matrix M' is generated by taking a number of elements (defined as 'span' or 's') and multiplying them to create one element in the set W that will be used to generate a matrix M'.

(3) Algorithm 3: Similar to Algorithm 2, except every combination of column pairings is used over successive iterations when multiplying s column values together to form elements in W.

(4) Algorithm 4: A "brute force" algorithm that starts with a matrix M' set to all ones. Each element a_{ij} in M is modified to be $\frac{a_{ij}}{2}$, and then $a_{ij} * 2$. Upon each modification to a_{ij}, the new matrix M' is tested whether the distance to M is reduced. If the distance is indeed reduced, the value of a_{ij} that resulted in the smallest distance to M is kept and the next element is tested in the same manner. Otherwise, the original value of a_{ij} is retained. This repeats until all elements above the diagonal have been tested in this way. This process as a whole executes several times (depending on the matrix data).

4. Algorithm Performance

4.1. **Introduction.** The intention of the subsequent tests is to establish the performance of the four algorithms that were developed. In the evaluation of performance, two main factors are taken into account: the computed distance between M and M', and the time it takes to run each algorithm. All algorithms are tested with the same set of parameters to ensure as much fidelity as possible between each execution of the algorithms.

The tests below are meaningful because they encompass matrix sizes that are practical to most. That is, a matrix with a size of one thousand may be interesting from a computing point of view, but human subject matter experts will not be able to rank that many criteria in any meaningful way. The parameters chosen for execution of the algorithms also mirror a practical computing platform on which to solve these matrix inconsistencies.

Each algorithm has two graphs associated with it: Algorithm Accuracy and Algorithm Computational Cost (with respect to time).

The accuracy graph shows for each matrix size, what the average best and worst distances that M' is from random matrix M. The average is computed by tracking the best and worst results from each generation of a consistent matrix M. For example with Algorithm 1: if a matrix size is five, five distinct matrices are generated. The best distance and worst distances from among those five matrices is kept. This is repeated for the number of matrices specified to generate (program parameter *mps*). The average of the five best results is then calculated and displayed as *Avg. Best Solution*, then the process is repeated with the worst results and is displayed as *Avg. Worst Solution*. It is important to record both the best and worst distances so the reader can be aware of the range of possible distances that an algorithm would provide. Some algorithms have best and worst distances that are fairly close to each other, while other algorithms have a larger distance between the best and worst, especially as the matrix size increases.

The performance graph shows the total 'wall clock' time it took for the algorithm to execute with the given parameters for a certain size matrix noted on the graph.

The algorithms will test every size matrix specified between the minimum and maximum matrix sizes specified in the program parameters (inclusive of the min/max parameters specified).

A Python implementation of the algorithms is freely available for download at: `https://bit.ly/2uQCtKL`

4.2. **Testing Parameters.** The computer hardware and software configuration used to generate the test results below is as follows:

(1) Apple Macbook Pro (2015 model)
(2) 2.5GHz Intel Core i7
(3) 16 GB RAM
(4) Python 2.7.10
(5) Nuitka Python compiler 0.5.26 (for better performance) - http://nuitka.net/

When running each algorithm, the program parameter specified were:

```
consistency.py -minr 10 -maxr 50 -mps 8 -a all
```

This command line specifies that for each matrix size (10 to 50), eight matrices will be generated. A matrices-per-size setting of eight was chosen as the computer used for generating the results has a CPU capable of running eight simultaneous threads.

4.3. **Parallelization of the candidate algorithms.** Given the computationally intensive nature of the algorithms in this thesis, it was important to put some effort into taking advantage of a computers' hardware as much as was practically possible.

The Python program available for download is designed to dedicate each "run" of a particular algorithm to any free CPU thread. A run is defined as the execution of a particular algorithm to generate one matrix M' with the smallest distance between M and M' of size n. (Any intermediate M's that do not end up having the smallest distance to M within the algorithm as M' candidates are not considered here).

The Python implementation however has one shortcoming, in that is does not attempt to parallelize the execution of a particular algorithm run. For example, let us consider if the program is instructed to use Algorithm 1 to generate four matrices collectively named M' of size fifty, and the computer being used has eight CPU threads available for execution.

Ideally, the program would be able to use two of the available CPU threads within the algorithm run for each matrix, and all four matrices would be generated in parallel. This is not the case in the implementation provided, but it would be worth the effort especially for cases where you may want to generate one matrix of a very large size.

4.4. **Algorithm 1.**

4.4.1. *Algorithm Accuracy*

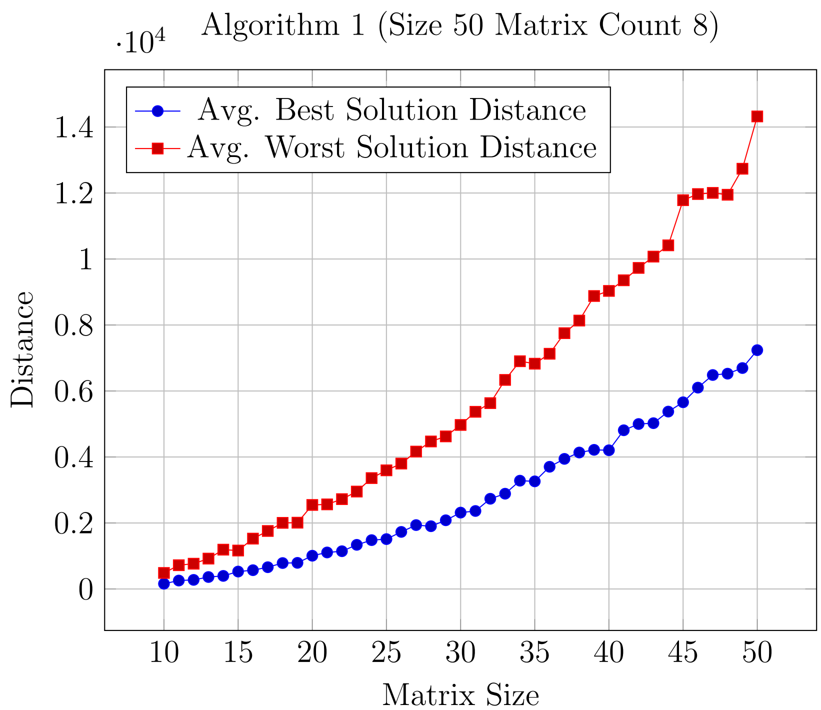


FIGURE 1. Algorithm 1 Accuracy

Algorithm 1 displays a consistent and smooth increase in distance for both the best and worst average solutions in each size of the matrices that were tested. The best solutions have a more consistent average with less variability between data points.

The average worst solutions (highest distance) overall increase in a linear fashion, but there are a few outliers where in some instances the distance between M and M' at the next larger size matrix is less than the previous size matrix distance between M and M'. This can more than likely be attributed to the random nature in which M is generated each time (see the definition of Algorithm 1 above).

4.4.2. *Algorithm Computational Cost*

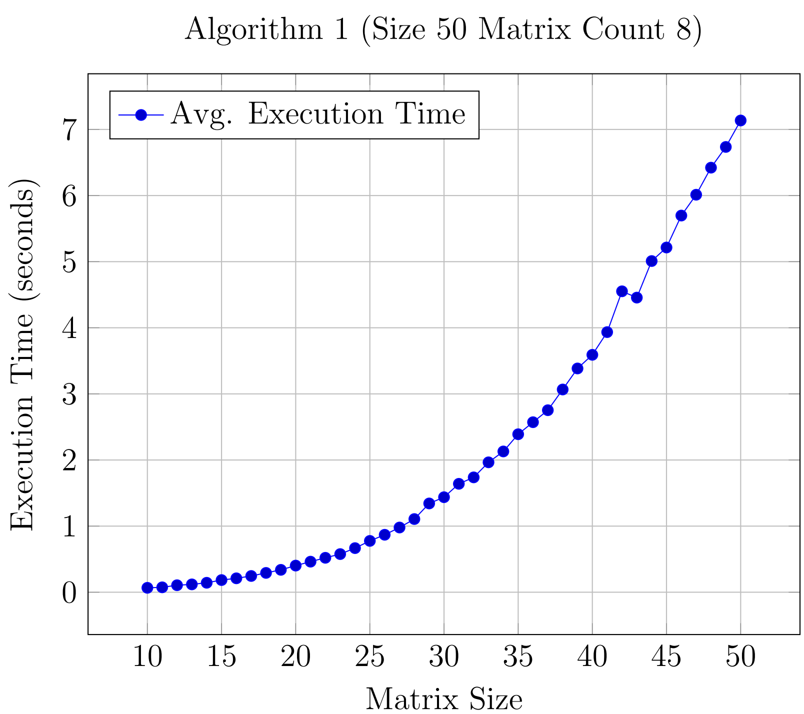


FIGURE 2. Algorithm 1 Computational Cost

Algorithm 1 performs rather well within the confines of the parameters of this thesis. It should be noted however that each time the size is increased above the size of twenty, the time for each successive matrix to complete begins increasing exponentially.

As the size of matrices increases, this algorithm will quickly become to inefficient to use in practical applications.

4.5. **Algorithm 2.**

4.5.1. *Algorithm Accuracy*

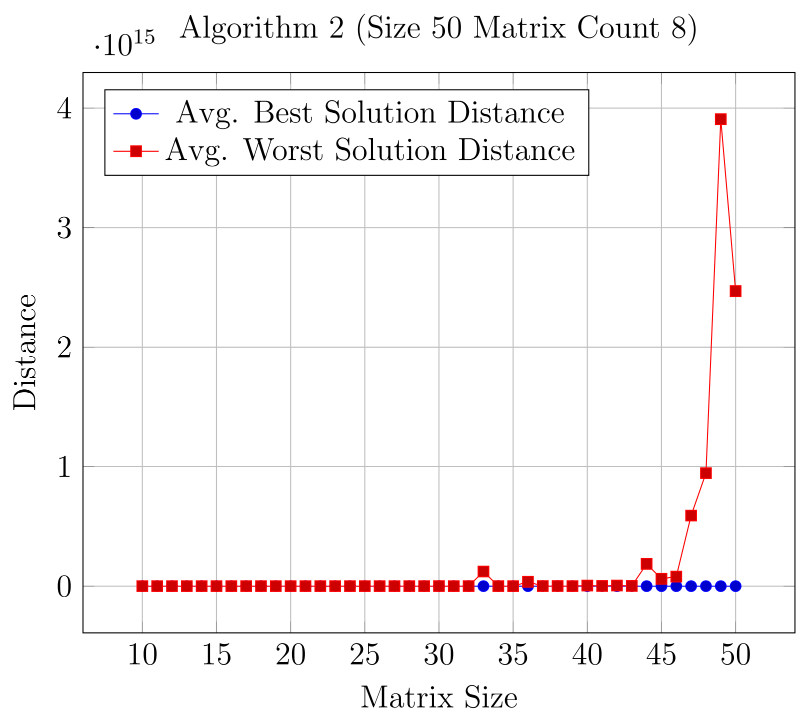


FIGURE 3. Algorithm 2 Accuracy

Algorithm 2 has a very large disparity between the best and worst average distances as the size of the matrices grows, especially with matrices larger than forty. Past forty, the worst distance diverges greatly from the best solution distance for reasons that are unknown at this time.

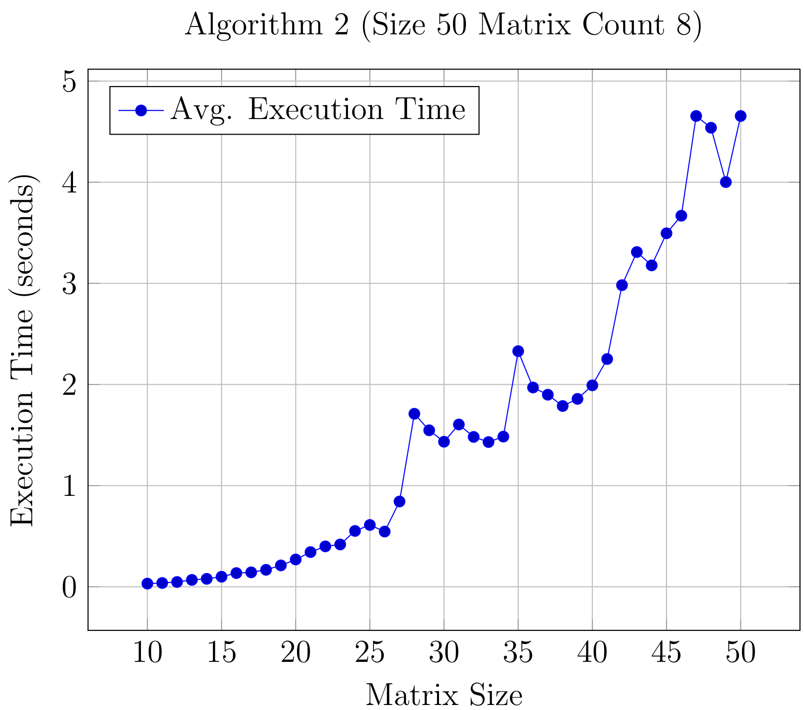


FIGURE 4. Algorithm 2 Computational Cost

4.5.2. *Algorithm Computational Cost*

Algorithm 2 performs rather well within the confines of the parameters of this thesis, and the trend as the size of the matrices grows is more linear in contrast to Algorithm 1.

4.6. **Algorithm 3.**

4.6.1. *Algorithm Accuracy*

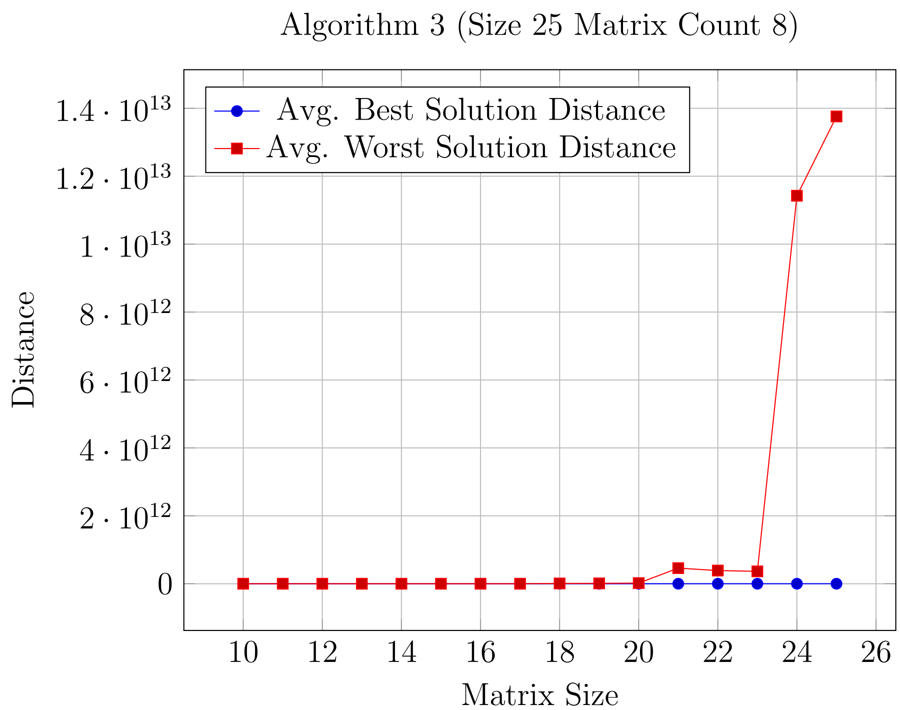


FIGURE 5. Algorithm 3 Accuracy

Algorithm 3 displays a consistent and smooth increase in distance for both the best and worst average solutions in each size of the matrices that were tested.

The best solutions have a more consistent average with less variability between data points. The average worst solutions (highest distance) overall increase in a linear (albeit steeper) fashion, but there are a few outliers where in some instances the distance between M and M' at the next larger size matrix is less than the previous size matrix distance between M and M'. This can more than likely be attributed to the random nature to which M is generated each time as (see the definition of Algorithm 1 above).

If there were more matrices generated in each matrix size, this variability would likely be greatly reduced as the number of samples to average against is increased, but the computational cost of this algorithm makes this prohibitive.

4.6.2. *Algorithm Computational Cost*

Algorithm 3 has a very high computational cost past matrix sizes near twenty-five and higher. Past this point, the time to compute a given number matrices of a certain size starts to increase exponentially. At a matrix size of 25, the time to compute is no longer practical. (Almost 34,000 seconds for eight matrices of a given size).

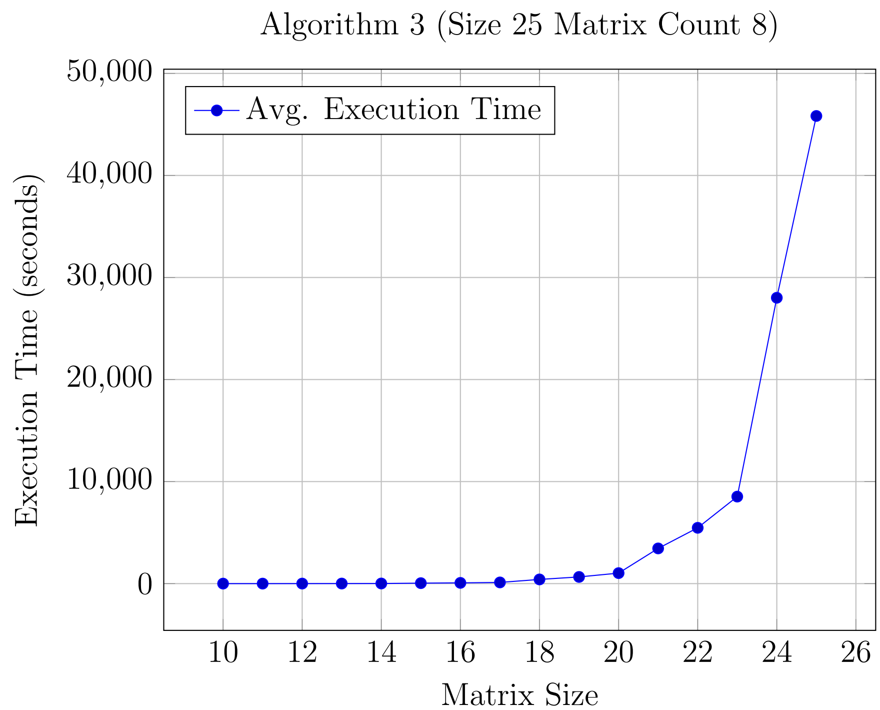
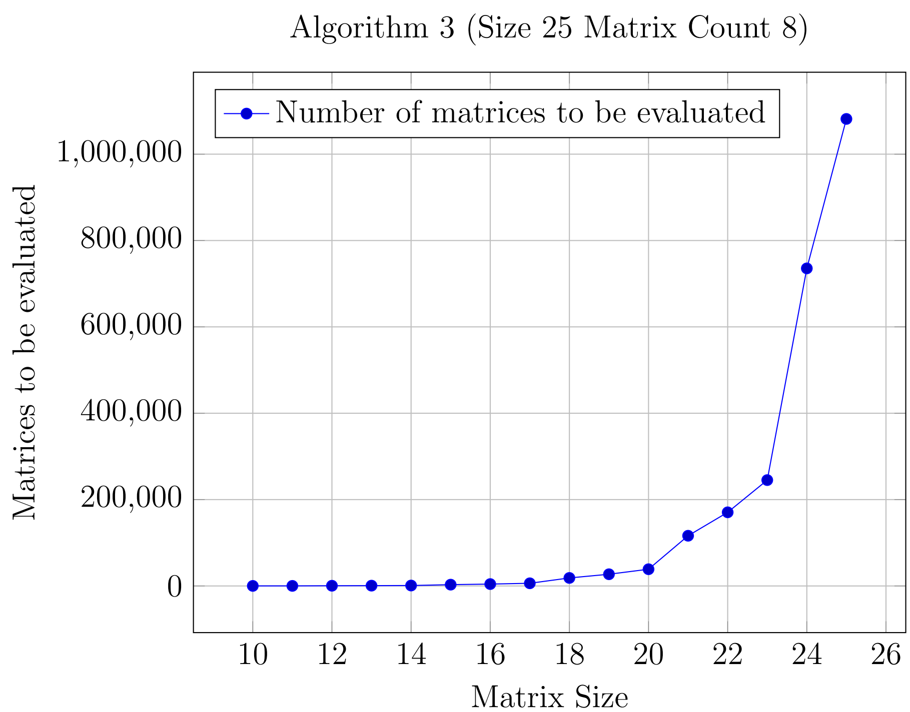


FIGURE 6. Algorithm 3 Computational Cost

4.6.3. *Modeling of Increasing Algorithmic Complexity*



The graph directly above shows that as the size of matrix M increases, the number of matrices to be evaluated increases dramatically past approximately a matrix size of 20. This can be attributed to the fact that based on the combination of the "span" parameter and the size of the matrix, the number of combinations of column values increases.

The modeling of this increasing complexity can be defined with the following binomial:

$$\binom{n}{k} = \frac{n!}{k!(floor(n/3) - k)!}$$

This binomial represents OEIS Integer Sequence A051033 [7]

4.7. **Algorithm 4.**

4.7.1. *Algorithm Accuracy*

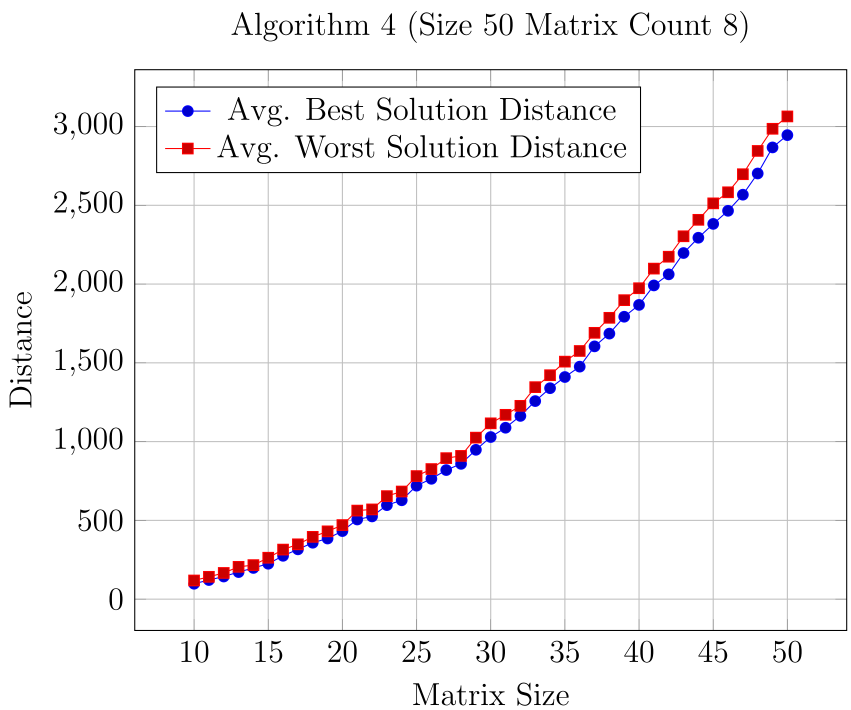


FIGURE 7. Algorithm 4 Accuracy

Algorithm 4 displays a consistent and smooth increase in distance for both the best and worst average solutions in each size of the matrices that were tested. The increase in distance as the size of the matrices grow is not quite linear, but definitely not exponential.

It is interesting to note that the distance between the best and worst solutions for each size of matrices are very close to each other compared to the other algorithms that were tested, and the trend for each is complimentary to the other.

This algorithm also shows very little variability between data points - the respective distances grow in a very orderly fashion as the size of the matrices increases.

4.7.2. *Algorithm Computational Cost*

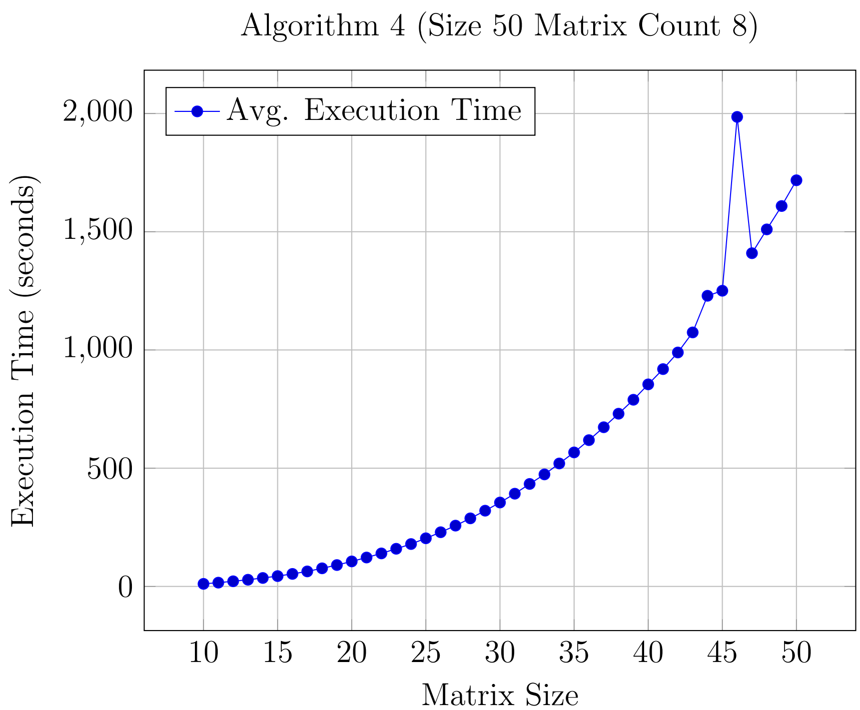
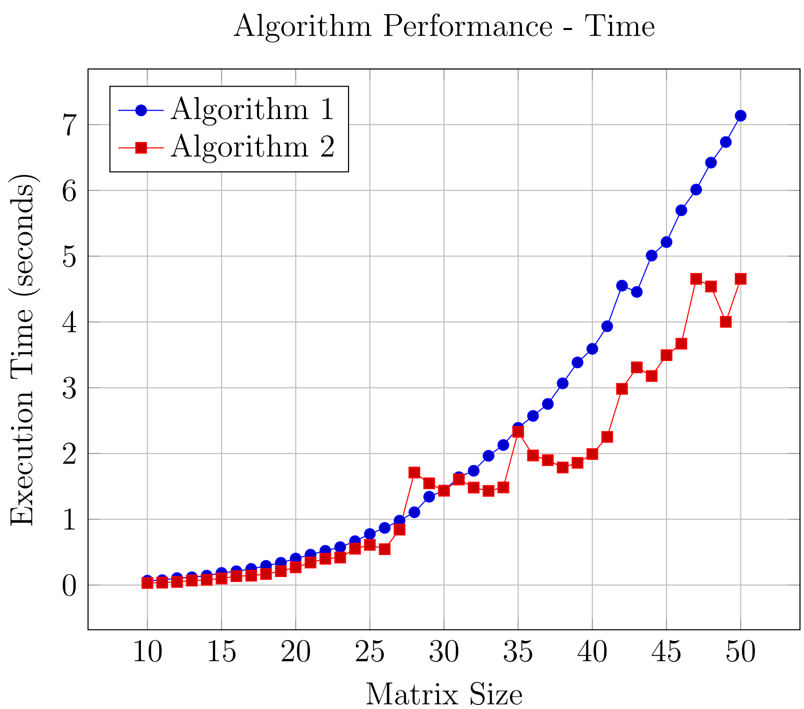
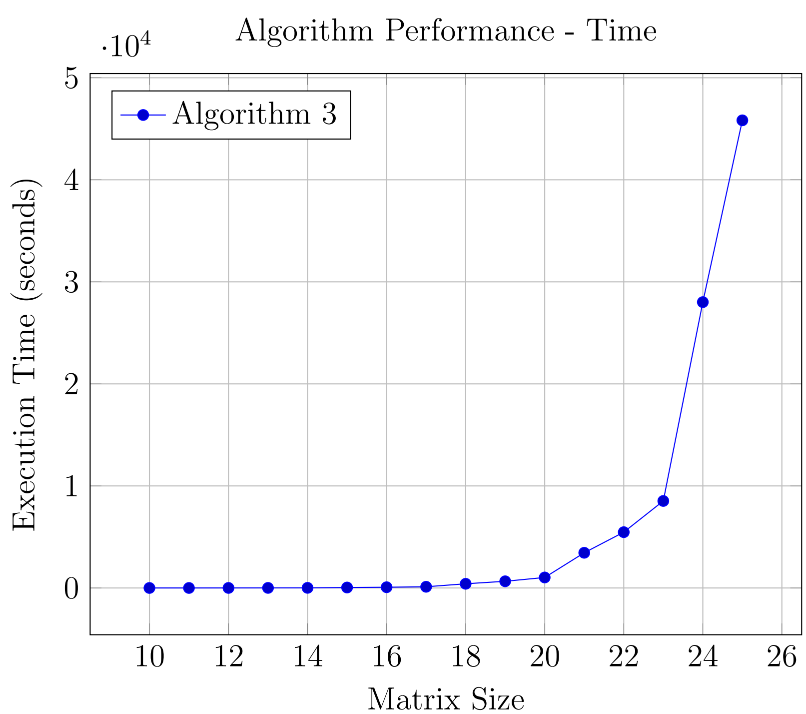


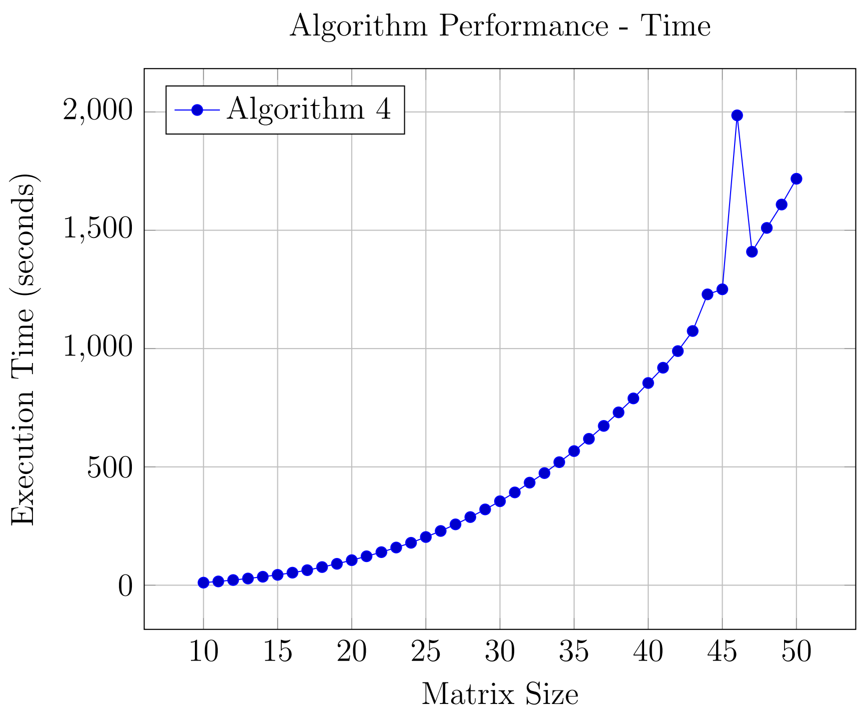
FIGURE 8. Algorithm 4 Computational Cost

Algorithm 4 performs rather well within the confines of the parameters of this thesis, although each time the size is increased above about the size of thirty, the time for each successive size to complete seems to start increasing exponentially. As the size of matrices increases (perhaps above one hundred), this algorithm will quickly become too inefficient to use in practical applications.

4.8. **Unified view of Algorithm Performance.**






Algorithm Performance - Time
Algorithm 4
2,000
1,500
1,000
500
0
Execution Time (seconds)
10
15
20
25
30
35
40
45
50
Matrix Size

5. Conclusions

5.1. **Summary.** When evaluating which algorithm to use for arriving at a consistent PC matrix, there are two elements considered in this thesis: The distance between M and M', and the computational cost (with respect to time) that was incurred to arrive at that solution. In practice, two of these algorithms were relatively well matched with respect to distance and computational cost, while the other two under consideration lacked the same performance of the two formerly mentioned algorithms in both distance and computational cost.

Based on examination of each algorithm's accuracy and performance, Algorithms 1 and 4 are the clear winners. Algorithm 2 has fast execution times that may allow very large matrices to processed, but its accuracy is much less when compared to Algorithms 1 and 4. The performance of Algorithm 3 with respect to accuracy rivals that of Algorithm 1 and 2 (comparatively), but this relative accuracy comes with a severe computing cost penalty which makes it prohibitively expensive to run for matrices where $size > 25$. Algorithm 4 had the best accuracy of all, but is more than seven times costlier to run than Algorithm 1.

If the most ideal solution is desired, Algorithm 4 can be used if results are not needed immediately. For slightly less accurate results and significantly less computational cost, Algorithm 1 is the preferred algorithm.

Name	Avg. Best Distance ($Size = 50$)	Execution time (secs)
Algorithm 1	4556	0.211
Algorithm 2	4.767^{24}	4.654
Algorithm 3 (size=25)	24797112	33816
Algorithm 4	3656	47.004

TABLE 3. Algorithm Performance Summary

5.2. **Future Directions.** Using this thesis as a jumping off point, future investigation can take several different directions. Future work can involve:

(1) Development of new alternative algorithms for deriving a consistent matrix from a non-consistent matrix.

(2) Refinement of the algorithms presented in this thesis with respect to both distance between M and M', but especially with regards to reducing the computing costs involved in Algorithm 3.

(3) Porting of this code to a more high performance programming language such as C or C++ for maximum performance.

(4) Implement support for more fine-grained parallelism in the implemented algorithms.

References

[1] Condorcet. Essai sur l'application de l'analyse 'a la probabilité des décisions rendues à la pluralité des vois. Paris, 1785.

[2] Dyer, J. S. Remarks on the analytic hierarchy process. *Manage. Sci. 36*, 3 (Mar. 1990), 249–258.

[3] Hagele, G., and Pukelsheim, F. Llull's writings on electoral systems. *Studia Lulliana 41* (2001), 3–38.

[4] Holsztynski, W., and Koczkodaj, W. W. Convergence of inconsistency algorithms for the pairwise comparisons. *Information Processing Letters*, 59 (1996), 197–202.

[5] Janicki, R. Approximations of arbitrary relations by partial orders: Classical and rough set models. In *Transactions on Rough Sets XIII, LNCS* (2011), J. F. P. et al, Ed., vol. 6499, Springer-Verlag Berlin Heidelberg.

[6] Koczkodaj, W. A new definition of consistency of pairwise comparisons. *Mathematical and Computer Modelling 18*, 7 (1993), 79–84.

[7] Online Encyclopedia of Integer Sequences. A051033 - OEIS. `http://oeis.org/A051033`, 2011. [Online; accessed 5-April-2018].

[8] Saaty, T. L. A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology 15* (1977), 234–281.

[9] Sandrasagra, B., and Soltys, M. Complex ranking procedures. *Fundamenta Informaticae Special Issue on Pairwise Comparisons 144*, 3-4 (2016), 223–240.

[10] Soltys, M. *An Introduction to the Analysis of Algorithms*, third ed. World Scientific, 2018.

[11] THURSTONE, L. L. A law of comparative judgement. *Psychological Review 34*, 278–286 (1927).