

https://dspace.org/

The optimal assignment problem: an investigation into current solutions, new approaches and the doubly stochastic polytope

Vermaak, Frans-Willem

2011-05-23
http://hdl.handle.net/10539/9894

Downloaded from a DSpace repository. https://dspace.org/

The Optimal Assignment Problem: An Investigation Into Current Solutions, New Approaches and the Doubly Stochastic Polytope

Frans-Willem Vermaak

A Dissertation submitted to the Faculty of Engineering, University of the Witwatersand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering

Johannesburg 2010

Declaration

I declare that this research dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

_____

signed this _____ day of _____2010

Abstract

This dissertation presents two important results: a novel algorithm that approximately solves the optimal assignment problem as well as a novel method of projecting matrices into the doubly stochastic polytope while preserving the optimal assignment. The optimal assignment problem is a classical combinatorial optimisation problem that has fuelled extensive research in the last century. The problem is concerned with a matching or assignment of elements in one set to those in another set in an optimal manner. It finds typical application in logistical optimisation such as the matching of operators and machines but there are numerous other applications. In this document a process of iterative weighted normalization applied to the benefit matrix associated with the Assignment problem is considered. This process is derived from the application of the Computational Ecology Model to the assignment problem and referred to as the OACE (Optimal Assignment by Computational Ecology) algorithm. This simple process of iterative weighted normalisation converges towards a matrix that is easily converted to a permutation matrix corresponding to the optimal assignment or an assignment close to optimality. The document also considers a method of projecting a matrix into the doubly stochastic polytope while preserving the optimal assignment. Various methods of projecting square matrices into the doubly stochastic polytope exist but none that preserve the assignment. This novel result could prove instrumental in solving assignment problems and promises applications in other optimisation algorithms similar to those that Sinkhorn's algorithm finds.

CONTENTS

Chapter 1: Introduction 1

1.1 Introduction 1

1.2 Problem statement 3

Chapter2: Literature survey 4

2.1 General concepts 4

2.1.1 Permutation matrices 4

2.1.2 Doubly stochastic matrices 5

2.1.3 The assignment problem 7

2.2 Applications of the assignment problem 9

2.2.1 Reformulating the shortest path problem as an assignment problem 9

2.3 Methods of solving the assignment problem 10

2.3.1 The assignment problem considered as a linear program 10

2.3.2 The Dual Problem 11

2.3.3 Hungarian method 12

2.3.4 Auction algorithm 19

2.3.5 Invisible hand algorithm: Solving the assignment problem using a statistical physics approach 24

2.4 Other Important Concepts 30

2.4.1 Hogg and Huberman's Computational Ecology 30

Chapter 3: The OACE Algorithm 36

3.1 Introduction 36

3.2 Application of the Computational Ecology structure to the assignment problem 37

3.3 OACE algorithm 39

3.3.1 Initialisation of the algorithm 40

3.3.2 Termination of the algorithm 40

3.3.3 OACE Algorithm viewed as a process of iterative weighted normalisation 42

3.3.4 Convergence of the population matrix 43

3.3.5 Results 46

3.3.6 Simulation results 46
3.3.7 Computational efficiency of the algorithm 49
3.3.8 Examples and applications of the OACE algorithm 54
3.4 Comparison of the OACE algorithm to other algorithms 59
3.4.1 Brute force approach 59
3.4.2 OACE algorithm 60
3.4.3 Auction algorithm 60
3.4.4 Hungarian algorithm 60
3.4.5 Graphical comparisons of the computational complexity of the different algorithms 61
3.4.6 Tabled Summary of comparisons 63
Chapter 4: Doubly stochastic approximation of a matrix with the same optimal assignment 64
4.1 Introduction 64
4.2 Derivation 64
4.3 Computational complexity of algorithm 71
4.4 Example 72
4.5 Discussion of assignment preserving projection method 74
Chapter 5 75
5.1 Conclusion 75
5.2 Future work 76
5.2.1 Analytical derivation of OACE algorithm 76
5.2.2 Applications of the OACE algorithm 76
5.2.3 Application of assignment preserving doubly stochastic projection 76
Bibliography 77

LIST OF APPENDICES

Appendix A: **Matlab Code**

A1 Standard OACE algorithm

A2 Assignment preserving doubly stochastic matrix projection

Appendix B: **Publications**

B1 Approximate Solution to the Optimal Assignment Problem using a Computational Ecology based Approach, F. Vermaak, M. A. Van Wyk

TABLE OF FIGURES

Figure 2.1: Different representations of an assignment: permutation, permutation matrix and a bipartite graph. 8

Figure 2.2: Digraph for illustration of the shortest path problem. 9

Figure 2.3: Graphical demonstration of the agent resource allocation process. 31

Figure 3.1: Graphical illustration of the relations and interactions of the different agent groups and resources for a $\mathbf{3} \times \mathbf{3}$ benefit matrix. 38

Figure 3.2: Graph that plots the number of exact optimal solutions found to the Optimal Assignment problem from a random trial of 1000 for each size matrix runs against the size of the matrix. 47

Figure 3.3: Graph that plots the proximity of the solutions as found by the OACE algorithm to the actual optimal solutions as a function of the size of the matrices. 47

Figure 3.4: Graph that shows the percentage error between the solutions found by the OACE algorithm and the actual optimal solutions. 48

Figure 3.5: Enlarged view of the percentage error as depicted in Figure 3.4, shown for larger matrices only. 48

Figure 3.6: Average number of iterations taken to find a solution using the OACE algorithm for different size matrices. 50

Figure 3.7: Performance of the OACE algorithm with preconditioning applied to $\boldsymbol{B}$ in the form of an assignment preserving projection into the doubly- stochastic polytope as described in Section 3.2. 51

Figure 3.8: Performance of the OACE algorithm under repeated projection of $\boldsymbol{B}$ into the doubly stochastic polytope using the Assignment preserving projection as described in Section 3.2. 52

Figure 3.9: Performance of the OACE algorithm with preconditioning applied to $\boldsymbol{B}$ in the form of a constant bias added to all entries of matrix $\boldsymbol{B}$. 53

Figure 3.10: Comparison of performance of OACE algorithm with various types of preconditioning applied to $\boldsymbol{B}$ 53

Figure 3.11: Comparison of performance of OACE algorithm for the standard algorithm and the algorithm with added bias. 54

Figure 3.12: Plot of the evolution of all entries of the population matrix as a function of iteration number for the OACE algorithm. 55

Figure 3.13: Plot of the development of all entries of the population matrix corresponding to the shortest path example as a function of the iteration number. Plot continues until convergence of matrix $\boldsymbol{F}$ 55

Figure 3.14: Plot of the development of all entries of the population matrix corresponding to the shortest path example as a function of iteration number. Plot continues until matrix $\boldsymbol{F}$ becomes row dominant……57
Figure 3.15: Plot that compares the computational complexity of different algorithms. The faster growing plots are truncated after a few to allow for a meaningful graphical comparison……61
Figure 3.16: A second plot that compares the different algorithms with a different truncation to indicate the growth of the Auction algorithm more clearly……61
Figure 3.17: Indication of the operational complexity of the OACE algorithm compared to the brute force approach……62
Figure 3.18: Graph that attempts to graphically indicate how fast the factorial function grows. This graph keeps this shape when truncated at almost any value on a linear graph……62

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

The optimal assignment problem is a classical combinatorial optimisation problem. A simple explanatory example of the problem is matching a number of persons and objects on a one-to-one basis where each person has a specific benefit associated with being matched to a certain object. The optimal matching corresponds to an assignment of persons to objects that maximizes the sum of benefits. Expressed formally we have a set of persons and a set of objects both of cardinality N and each person-object matching has an associated scalar b_{ij} that describes the benefit of matching person i with object j. Each assignment can be represented by a permutation matrix and the space of all $N \times N$ permutation matrices is isomorphic to the set of permutations of elements in the set $\{1, \dots, N\}$. To find the optimal assignment we need to find the permutation matrix or equivalently the permutation that maximizes the sum of the benefits. This can be expressed as finding the permutation π that maximizes

$$\max_{a} \sum_{a=1}^{N} b_{\pi(a)a}$$

where $S_{\pi,N}$ is the set of all permutations of the elements $\{1, \dots, N\}$ with $\pi \in S_{\pi,N}$.

One tale tells of a logistical manager proposing the assignment problem to a mathematician, the mathematician ignorantly answered: "The problem is trivial as all one needs to do is evaluate all the different possibilities." This of course seems possible but is not at all practical for problems where large sets are to be matched. In section 2.1 it is shown that the computational complexity of the assignment problem scales as the factorial of the cardinality of the sets being matched. For a 60 by 60 matching there is an order of 10^{80} different possible assignments. This is also according to modern approximations the order of the number of atoms in the known universe. Clearly a more computationally efficient solution is necessary.

Various approaches have been developed to find efficient solutions to the assignment problem. To name a few of the more important ones: the Hungarian method by Kuhn and Munkres was of the first [1,2]. Dantzig solved the problem using linear programming [3] and more recently Bertsekas developed the Auction algorithm that solves the problem very efficiently [4]. Another interesting approach by Kosowsky and Yuille [5] manages to find a solution to the assignment problem using an approach related to statistical physics. It is referred to as the Invisible hand algorithm.

The assignment problem is of great theoretical importance as it embodies a fundamental linear programming structure. It is possible to reformulate important types of linear programming problems such as the linear network flow and shortest path problems to take the form of an assignment problem [4]. Numerous other applications of the assignment problem are discussed in section 1.2.

In this dissertation a novel method of solving the assignment problem is introduced and analysed. The method originates from the idea of solving the optimal assignment problem using a Computational Ecology based approach as developed by Hogg and Hubermann [6,7,8]. The result is a simple algorithm referred to as the OACE (Optimal Assignment by Computational Ecology) algorithm that reduces to a process of weighted recursive iteration and converges to the optimal assignment or a solution close to optimality. These findings and results were presented at the ICMTMA2010 international conference, Changsha, China and published by the author in [9]. The relations between the assignment problem and the set of doubly stochastic matrices are also investigated and a novel result concerning an assignment preserving projection into the doubly stochastic polytope derived. These findings and results are to be submitted for publication. The OACE algorithm and assignment preserving projection method are the main emphasis in this document as they offer a contribution to our state of current knowledge.

Chapter 2 begins with an overview of the important mathematical concepts associated with the assignment problem. Next the formal definition of the problem and its dual is covered and then an overview of all the most significant solutions to the problem is given. This overview includes summaries of the most important algorithms and examples illustrating the dynamics of some of these algorithms. The derivation of the Invisible hand algorithm is comprehensively covered as the literature regarding this can do with clarification. The chapter concludes with the necessary background of the Computational Ecology model. Chapter 3 looks at the application of the Computational Ecology model to the assignment problem resulting in the OACE algorithm. The algorithm is explained and an analysis of the simulation results and possible alterations and improvements such as preconditioning of the benefit matrix follows. The next chapter, chapter 4 covers derivation of the assignment preserving doubly stochastic projection. Its performance, utility and possible application to the assignment problem is discussed. The dissertation concludes in Chapter 5 with a summary and discussion of the most important results. Possible related future work and natural extensions of the work in this dissertation are also discussed.

1.2 PROBLEM STATEMENT

The assignment problem finds many applications; the most obvious being that of matching such as the matching of operators and machines or delivery vehicles and deliveries. There are however numerous other interesting applications. In [10] Van Wyk shows that the graph matching problem can be rewritten to take the form of an optimal assignment, it turns out that many such optimisation procedures that are concerned with finding a closest approximation of an element in some vector space can be wholly or partly reformulated as assignment problems. Another such example by Imai [11] is the approximation of piecewise linear functions or human face detection by Ying [12]. Wästlund shows in [13] that the shortest path problem can be reformulated as an assignment problem. Such algorithms are used in the determination of routing tables for telecommunication networks or optimal routes in GPS navigation systems. Some other applications of the assignment problem include tracking moving objects in space [14], the matching of moving objects [15] and scheduling of an input queued switch [16].

The importance of the assignment problem is quite clear from the above and various very successful solutions to the problem already exist. This document introduces and investigates a novel approach to solving the assignment problem. From a research point of view casting old problems into new frameworks bares merit as it gives a clear grounding for comparison of different methods and enriches the understanding of the dynamics that govern these phenomena. The assignment problem is a classical and practical problem which has generated much theoretical and practical analyses and which certainly warrants thorough investigation.

CHAPTER2: LITERATURE SURVEY

2.1 GENERAL CONCEPTS

2.1.1 Permutation matrices

The set of matrices known as the permutation matrices is the collection of all square matrices with binary entries (entries from the set {0,1}) with only one 1 in every row and one 1 in every column.

Definition 2.1 Permutation matrix [17]

A matrix $\boldsymbol{A}$ *with entries* a_{ij} *is a permutation matrix if* $a_{ij} \epsilon \{0,1\}$ *with* $\sum_i a_{ij} = 1$ *and* $\sum_j a_{ij} = 1$.

Another way of thinking about a permutation matrix is that a permutation matrix is obtained when taking an identity matrix and interchanging any of its rows and/or any of its columns. For the set consisting of all $N \times N$ matrices there are exactly $N!$ different permutation matrices corresponding exactly to the number of permutations of placing a collection of N objects into N bins. As a matter of fact the set of all $N \times N$ permutation matrices is isomorphic to the set of permutations of the set of numbers {1,2,…,N}. Some other interesting properties of permutation matrices include: A permutation matrix satisfies

$$\boldsymbol{A}\boldsymbol{A}^{\boldsymbol{T}} = \boldsymbol{I}$$

with $\boldsymbol{I}$ the identity matrix and $\boldsymbol{A}^{\boldsymbol{T}}$ the transpose of permutation matrix $\boldsymbol{A}$. Furthermore a permutation matrix is always non-singular with determinant ± 1 and when some matrix $\boldsymbol{M}$ is front multiplied by $\boldsymbol{A}$ it renders matrix $\boldsymbol{M}$ with rows interchanged according to the permutation vector $\boldsymbol{p}$ corresponding to $\boldsymbol{A}$. If matrix $\boldsymbol{M}$ is back multiplied by $\boldsymbol{A}$ the columns of $\boldsymbol{M}$ are interchanged similarly.

2.1.2 Doubly stochastic matrices

The set of doubly stochastic matrices is the set of all positive square matrices with the quality that each row and each column sum to a total of unity.

Definition 2.2 Doubly stochastic matrix [18]

A matrix $\boldsymbol{C}$ *is called doubly- stochastic if* $c_{ij} \epsilon \mathbb{R}$ *with* $\sum_i c_{ij} = 1$, $\sum_j c_{ij} = 1$ *and* $c_{ij} \geq 0 \; \forall i, j$

Clearly the set of doubly stochastic matrices contains the set of permutation matrices. The set of doubly stochastic matrices actually forms a convex polytope with the permutation matrices at its extreme points and a famous theorem by Birkhoff states that any doubly stochastic matrix can be obtained from a convex combination of permutation matrices.

A polytope is a finite region of n-dimensional space bound by a finite number of hyperplanes [19]. Such a region represents a number of related but slightly different mathematical objects. The doubly stochastic polytope refers to the region of $n \times n$ dimensional space that contains all the doubly stochastic matrices of size $n \times n$.

Definition 2.3 Polytope [20]

An n*-dimensional polytope may be specified as the set of solutions to a system of linear inequalities*

$$\boldsymbol{Mx} \leq \boldsymbol{b}$$

where $\boldsymbol{M}$ *is a real* $m \times n$ *matrix and* $\boldsymbol{b}$ *a real vector of size* n*.*

Definition 2.4 Convexity [21]

A set S can be described as a convex set when, if $x_1, x_2 \, \epsilon \, S$*, then all points of the form* $\alpha x_1 + (1 - \alpha) x_2 \, \epsilon \, S$ *where* $0 \leq \alpha \leq 1$ and $\alpha \, \epsilon \, \mathbb{R}$.

Theorem 2.1 Birkhoff 's polytope [22]

Every doubly stochastic matrix can be written as a convex combination of permutation matrices.

Alternatively one can say that the permutation matrices correspond exactly to the extreme points or vertices of the doubly stochastic polytope.

Example of Birkhoff's theorem for the 3×3 case

For the case of the space of all 3×3 matrices the following permutation matrices exist

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix},\begin{bmatrix}1 & 0 & 0\\0 & 0 & 1\\0 & 1 & 0\end{bmatrix},\begin{bmatrix}0 & 1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix},\begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix},\begin{bmatrix}0 & 1 & 0\\0 & 0 & 1\\1 & 0 & 0\end{bmatrix},\begin{bmatrix}0 & 0 & 1\\0 & 1 & 0\\1 & 0 & 0\end{bmatrix}.$$

There are $N! = 3! = 6$ different permutation matrices. These matrices can now be combined as

$$\alpha\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}+\beta\begin{bmatrix}1 & 0 & 0\\0 & 0 & 1\\0 & 1 & 0\end{bmatrix}+\gamma\begin{bmatrix}0 & 1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix}+\delta\begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix}+\varepsilon\begin{bmatrix}0 & 1 & 0\\0 & 0 & 1\\1 & 0 & 0\end{bmatrix}+\zeta\begin{bmatrix}0 & 0 & 1\\0 & 1 & 0\\1 & 0 & 0\end{bmatrix}$$

where $\alpha, \beta, \gamma, \delta, \varepsilon, \zeta \in \mathbb{R}$ and $\alpha + \beta + \gamma + \delta + \varepsilon + \zeta = 1$ in accordance with Birkhoff's theorem and the concept of convexity. The following matrix is the result

$$\begin{bmatrix}\alpha+\beta & \gamma+\varepsilon & \delta+\zeta\\ \gamma+\delta & \alpha+\zeta & \beta+\varepsilon\\ \varepsilon+\zeta & \beta+\delta & \alpha+\gamma\end{bmatrix}$$

and all rows and columns will sum to unity as each row and column contains all the elements $\alpha, \beta, \gamma, \delta, \varepsilon, \zeta$ exactly once.

Theorem 2.2 Sinkhorn's result [23]

Given a positive square matrix $\boldsymbol{M}$*, there exists a unique doubly stochastic matrix* $\boldsymbol{\Phi}$ *such that* $\boldsymbol{\Phi} = \boldsymbol{D_1 M D_2}$ *where* $\boldsymbol{D_1}$*and* $\boldsymbol{D_2}$ *are diagonal matrices with positive main diagonals.* $\boldsymbol{D_1}$*and* $\boldsymbol{D_2}$ *are unique up to a scalar multiple. The matrix* $\boldsymbol{\Phi}$ *is obtained in the limit of the sequence of matrices obtained by the process of iterative normalisation of the rows and columns of* $\boldsymbol{M}$*.*

2.1.3 The assignment problem

2.1.3.1 Introduction

As explained in the introductory paragraph, the assignment problem is concerned with the concept of finding an optimal one-to-one matching between two sets. A very common example is the allocation of operators to machines, the one set being the set of operators and the other the set of machines. These two sets must be related in the sense that each element from set 1 has a certain scalar value associated with each element from set 2. In the context of the previous example this value would represent the skill level of the operator in operating the respective machine also referred to as the *benefit* of assigning the operator to that machine. Such an association can be concisely expressed in matrix format with the matrix referred to as the benefit matrix $\boldsymbol{B}$ [24,25,26] where entry b_{ij} represents the benefit of matching element i from set 1 with element j from set 2. It is possible to calculate optimal assignments in the case where set 1 and set 2 are of different cardinality [14,28] by making a few minor adjustments such as the introduction of dummy elements [14] to the formulation of the problem but this will not be the focus here. $\boldsymbol{B}$ will then be an $N \times N$ square matrix where the two sets being matched are both of cardinality N. Also note that any algorithm with the ability to find an optimal assignment can also be used to find a minimal assignment after a suitable reformulation of $\boldsymbol{B}$. In short all entries of $\boldsymbol{B}$ need to be negated and the absolute value of the smallest entry of the negated matrix added to all its entries, resulting in a matrix with positive entries. The optimal assignment of this new benefit matrix will now correspond to the minimal assignment of the initial matrix and vice versa. In this document the term "optimal assignment" is mostly used to refer to a maximising optimal assignment but can also be used to refer to a minimising assignment as the two concepts are related by a mere reformulation.

2.1.3.2 Assignment

An assignment is obtained when we match an element from set 1 to an element from set 2 in a one-to-one fashion. Mathematically the assignment problem can be viewed as a bijective mapping of a finite set into itself, i.e. a permutation. This can be represented by a permutation matrix or an element from any set isomorphic to the set of permutation matrices such as the set of all possible permutations of the set of numbers {1,2,...,N}.

Another natural interpretation of the assignment problem is from a graph theoretical perspective. To achieve this we view the permutation matrix as the adjacency matrix of a weighted bipartite graph $G_\varphi = (V, W; E)$, with the vertex sets V and W of cardinality N and the edge weights E corresponding to the entries in the benefit matrix [14]. Every connected edge indicates an admissible assignment of operator to machine.

$$\varphi = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 \end{bmatrix}$$

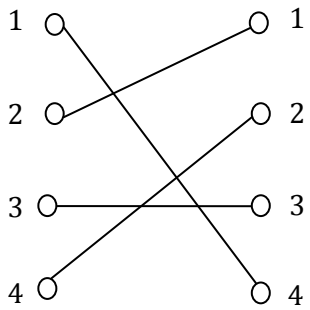


FIGURE 2.1: Different representations of an assignment: permutation, permutation matrix and a bipartite graph.

2.1.3.3 Optimisation

Solving the assignment problem corresponds to finding the assignment that maximizes the total benefit. This implies we must find the permutation π or corresponding permutation matrix Π that maximizes

$$\max_{i,j} \sum_{i,j=1}^{N} b_{ij}\, \Pi_{\mathrm{ij}} \qquad (2.1)$$

or equivalently

$$\max_{a} \sum_{a=1}^{N} b_{\pi(a)a} \qquad (2.2)$$

where $S_{\pi,N}$ is the set of all permutations of N elements with $\pi \in S_{\pi,N}$ and $S_{\Pi,\mathrm{N}}$ is the set of all $N \times N$ permutation matrices with $\Pi \in S_{\Pi,\mathrm{N}}$ and Π_{ij} the entry in row i and column j of Π.

2.2 APPLICATIONS OF THE ASSIGNMENT PROBLEM

Various other linear network flow problems can be reformulated as assignment problems. To name a few the classical shortest path problem [13,29], max-flow problems, linear transportation problems [30,31] and linear minimum cost flow (transhipment) problems[32]. An example of reformulating the shortest path problem as an assignment problem follows in section 2.2.1.

2.2.1 Reformulating the shortest path problem as an assignment problem

It is possible to encode the shortest path problem for a graph with $n+1$ vertices in an $n \times n$ matrix with the shortest path corresponding to the optimal assignment [13,29]. Consider a graph with $n+1$ vertices, we now reformulate the shortest path problem from node 0 to node n of such a graph as follows. Set up an $n \times n$ matrix with entry d_{ij} corresponding to the distance between node i and node j. The rows of matrix $\boldsymbol{D}$ need to be numbered from 0 to $n-1$ and the columns from 1 to n. It seems all entries $d_{ii} = 0$ but the way the rows and columns are numbered shifts these zero entries off the main diagonal removing the trivial minimal assignment. Any assignment in this matrix must contain a set of entries corresponding to a path from node 0 to node n hence the minimal assignment corresponds to the shortest path. With a suitable reformulation of matrix $\boldsymbol{D}$ as shown in section 3.1.8.2 the optimal assignment corresponds exactly to the shortest path.

2.2.1.1 Example of reformulating a shortest path problem as an assignment problem

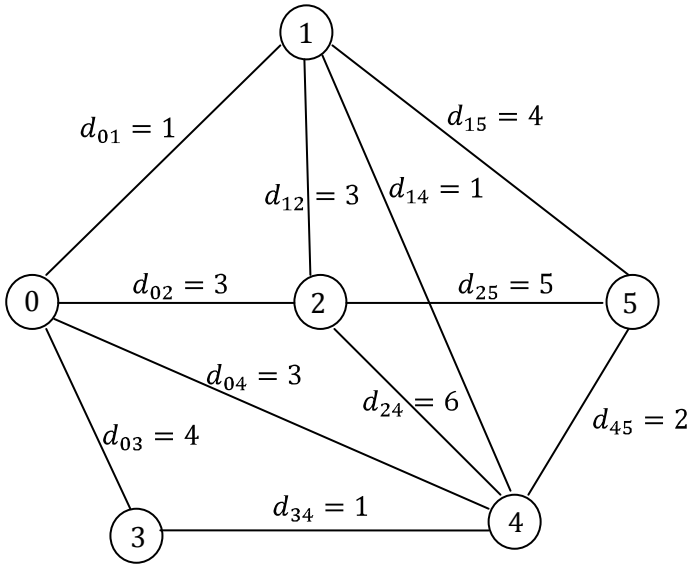


FIGURE 2.2: Digraph for illustration of the shortest path problem.

This graph is encoded in the following matrix

$$\begin{array}{c} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\ \begin{bmatrix} 1 & 3 & 4 & 3 & \infty \\ 0 & 3 & \infty & 1 & 4 \\ 3 & 0 & \infty & 6 & 5 \\ \infty & \infty & 0 & 1 & \infty \\ 1 & 6 & 1 & 0 & 2 \end{bmatrix} \end{array}$$

Solving the minimal assignment problem associated with this matrix now finds the shortest path in graph 2.2. As shown in Section 3.1.8.2 the solution is

$$\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} \end{bmatrix}$$

which corresponds to path

$$0 \rightarrow 1 \rightarrow 4 \rightarrow 5$$

with a total distance of

$$1 + 1 + 2 = 4.$$

Inspection of graph 2.2 shows that this is indeed the shortest path.

2.3 METHODS OF SOLVING THE ASSIGNMENT PROBLEM

2.3.1 The assignment problem considered as a linear program

It is possible to consider the assignment problem as a linear program as exemplified by Dantzig in [3] in his original example of matching 70 people with 70 jobs. Firstly we will reconsider the problem from a linear programming perspective. Consider having two sets of elements of cardinality N, one set being the set of objects $\{A(i)\}$ and the other the set of persons $\{i|j \in A(i)\}$. The assignment problem now corresponds to finding a set ψ of N pairs (i, j) of object-person pairings from the set of all possible pairings Ψ such that the total benefit is maximized. We can introduce a concise mathematical formulation by using assignment variables x_{ij} which take the value 1 when the pair (i, j) is selected and zero if not. The benefit of a selected pair would be given by b_{ij}. The problem can now be written as an integer linear programming problem.

$$\max_{\Psi} \sum_{(i,j)\in\psi} b_{ij}x_{ij} \tag{2.3}$$

such that

$$\sum_{j\in A(i)} x_{ij} = 1 \quad i = 1, \dots, N \tag{2.4}$$

$$\sum_{\{i|j\in A(i)\}} x_{ij} = 1 \quad j = 1, \dots, N \tag{2.5}$$

$$x_{ij} \in \{0,1\} \tag{2.6}$$

This linear program can be solved using methods such as the interior point method and renders the exact solution to the assignment problem as the constraint matrix is unimodular. Note that if condition (2.6) is replaced by $x_{ij} > 0$ we end up with a doubly stochastic matrix.

2.3.2 The Dual Problem

Any linear programming problem has a related dual problem. To obtain this we define a Lagrangian by appending the $2N$ constraints using Lagrange multipliers

$$L(x, \boldsymbol{u}, \boldsymbol{v}) = \sum_{(i,j)\in S} b_{ij}x_{ij} + \sum_{i=1}^{N} u_i \left(1 - \sum_{j\in A(i)} x_{ij}\right) + \sum_{j=1}^{N} v_j \left(1 - \sum_{\{i|j\in A(i)\}} x_{ij}\right), \tag{2.7}$$

which can be written as

$$L(x, \boldsymbol{u}, \boldsymbol{v}) = \sum_{(i,j)\in S} (b_{ij} - u_i - v_j)x_{ij} + \sum_{i=1}^{N} u_i + \sum_{j=1}^{N} v_j. \tag{2.8}$$

In the above equations the N dimensional multiplier vector $\boldsymbol{u}$ corresponds to constraint (2.4) while the vector $\boldsymbol{v}$ corresponds to constraint (2.5).

2.3.3 Hungarian method

2.3.3.1 Introduction

Possibly the first documented algorithmic method for finding a solution to the assignment problem aside from the brute force approach is the Hungarian method. The method was developed by H. W. Kuhn and based on the work of two Hungarian mathematicians, Evgary and König in the honour of which Kuhn named the algorithm the Hungarian algorithm. In this section a simple and practical overview of the Hungarian method is given. More detailed analysis and approaches can be found in [1,2,34,35].

The fundamental idea behind the Hungarian method is that the optimal assignment is conserved by the addition of a scalar to any row or column. This can easily be understood when considering that any assignment as defined in section 2.3.1.2 contains only one element from each row and one element from each column. The translation of a whole column (resp. row) by a constant corresponds to a translation of the optimal assignment by that same constant as any assignment must still contain one element from that column (resp. row). The Hungarian method uses this principle to transform a matrix into a sparse matrix by subtracting the minimal elements from each row and column, resulting in multiple zero entries while not affecting the optimal assignment of the matrix. When the matrix is sufficiently sparse an assignment of zeros entries that correspond to the minimal assignment would be possible and choosing such an assignment would solve the assignment problem for the matrix of interest. König's theorem gives an algorithmic method to check whether the matrix has been reduced to a sufficiently sparse matrix.

The Hungarian method can be described in a stepwise manner as follows:[34,36]

Step 1: Subtract the smallest entry in each row from all entries in that row.

Step 2: Subtract the smallest entry in each column from all entries in that column.

Step 3: Draw lines through the appropriate rows and columns ensuring that all zero entries are covered and use the minimum number of these lines (such a choice of lines is known as a zero cover and we use König's algorithm to find such a cover as discussed in section 2.3.3.2).

Step 4: If the minimum number of lines used is equal to N, an assignment of zeros is possible and the algorithm terminates.

If the minimum number of lines needed to cover all zeros is less than N an assignment is not yet possible and the algorithm continues.

Step 5: Find the smallest entry not covered by a line and subtract this from all other uncovered entries and add it to all entries covered by both a horizontal and vertical line. Return to step 3.

2.3.3.2 König's Theorem

The recursive subtraction of smallest elements from the rows and columns of a matrix will clearly result in the presence of a large number of zeros in the matrix (at least one in each row and column). If a selection of zeros from the matrix can be made corresponding to an assignment (a selection of zeros such that the position of each zero corresponds to a one in an associated permutation matrix) the algorithm terminates. We can consider this as covering all zero entries in the matrix by drawing lines through the rows and columns. The minimum number of lines needed to cover all zeros in the matrix is then known as its cover and when the cover equals the order of the matrix an assignment of zeros is possible. Determining whether such an assignment is possible is simple enough to see when matrices are small but could be very difficult for large matrices. Clearly there exists a need for an algorithmic method to find the cover and König's theorem allows the development of exactly such an algorithm.

Theorem 1.4 **König's theorem** [35,34]

For every rectangular matrix containing zeros as some of the entries, the number of zeros in a largest independent set of zeros is equal to the number of lines in a smallest cover of the zeros.

König's Algorithm

Step 1: Find a cover of independent zeros. This can easily be achieved by for example selecting the first zero from the first row, then the first zero from the next row that is not in a column which already contains a selected zero etc.

Step 2: Mark all rows containing a selected zero as covered (draw a horizontal line through the rows) and all other lines with 0.

Step 3: Find an unselected zero, if all zeros in the matrix are covered the selected zeros form an independent set of the same size as the cover and the algorithm terminates. If an unselected zero is found label the column containing this zero with the index of the row containing it and continue to step 4.

Step 4: See if the column containing the unselected zero contains a selected zero. If it does this selected zero will be covered. Label the row containing the selected zero with the index of the column containing the selected zero. Mark this column as covered (draw a vertical line through it) and remove the horizontal line through the row containing the selected zero, return to step 3. If there is no selected zero we reach a phenomena known as a breakthrough, continue to step 5.

Step 5: When a breakthrough is achieved it is possible to increase the size of the independent set of selected zeros by one. Draw a path from the unselected zero, to the column entry in the same row corresponding to the row label. Continue by tracing to the row in each column corresponding to the column label and then column in each row corresponding to the row label alternately until the label 0 is reached.

A new set of selected zeros is now found by selecting each unselected zero and unselecting each selected zero along the path. Since the path initially starts and ends with an unselected zero

there will now be one more selected zero than initially. Erase all labels and cover lines and return to step1.

2.3.3.3 Example of the Hungarian algorithm

Given a benefit matrix

$$\boldsymbol{B} = \begin{bmatrix} 2 & 5 & 7 & 3 \\ 2 & 3 & 3 & 1 \\ 5 & 7 & 5 & 4 \\ 1 & 9 & 2 & 0 \end{bmatrix}.$$

Firstly negate the matrix

$$\begin{bmatrix} -2 & -5 & -7 & -3 \\ -2 & -3 & -3 & -1 \\ -5 & -7 & -5 & -4 \\ -1 & -9 & -2 & -0 \end{bmatrix},$$

then add the absolute value of the smallest entry, 9 in this case

$$\begin{bmatrix} 7 & 4 & 2 & 6 \\ 7 & 6 & 6 & 8 \\ 4 & 2 & 4 & 5 \\ 8 & 0 & 7 & 9 \end{bmatrix}.$$

Now subtract the smallest entry in each row from all entries in the row

$$\begin{matrix} 2 \\ 3 \\ 2 \\ 0 \end{matrix} \begin{bmatrix} 5 & 2 & 0 & 4 \\ 1 & 0 & 0 & 2 \\ 2 & 0 & 2 & 3 \\ 8 & 0 & 7 & 9 \end{bmatrix}$$

then subtract the smallest entry in each column from all entries in the column

$$\begin{matrix} & 1 & 0 & 0 & 2 \\ 2 \\ 2 \\ 5 \\ 0 \end{matrix} \begin{bmatrix} 4 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 \\ 7 & 0 & 7 & 7 \end{bmatrix}.$$

Check for minimum zero cover

$$\begin{bmatrix} \cancel{4} & \cancel{2} & \cancel{0} & \cancel{2} \\ \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} \\ 1 & \cancel{0} & 2 & 1 \\ 7 & \cancel{0} & 7 & 7 \end{bmatrix}.$$

The zero cover only needs 3 lines in this case but $N = 4$. This implies we must continue with step 5, subtracting the minimum uncovered entry from all other uncovered entries and adding it to all the entries covered twice

$$\begin{bmatrix} 4 & 3 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 6 & 0 & 6 & 6 \end{bmatrix}.$$

We need to check for a zero cover again as there are now more zeros

$$\begin{bmatrix} 4 & 3 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 6 & 0 & 6 & 6 \end{bmatrix}.$$

In this case there are many configurations possible for the cover but all of them need at least four lines. This implies an assignment is possible. It is easily seen by inspection that the following two assignments of zeros are possible

$$\begin{bmatrix} 4 & 3 & \mathbf{0} & 2 \\ 0 & 1 & 0 & \mathbf{0} \\ \mathbf{0} & 0 & 1 & 0 \\ 6 & \mathbf{0} & 6 & 6 \end{bmatrix} \text{ and } \begin{bmatrix} 4 & 3 & \mathbf{0} & 2 \\ \mathbf{0} & 1 & 0 & 0 \\ 0 & 0 & 1 & \mathbf{0} \\ 6 & \mathbf{0} & 6 & 6 \end{bmatrix}$$

implying that this benefit matrix has two optimal solutions corresponding to the permutation matrices

$$\begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ 0 & \mathbf{1} & 0 & 0 \end{bmatrix}.$$

This example shows the execution of the Hungarian algorithm but does not show much of the working of König's algorithm as small matrices are not well suited to this. The following example illustrates König's agorithm.

Example of König's algorithm

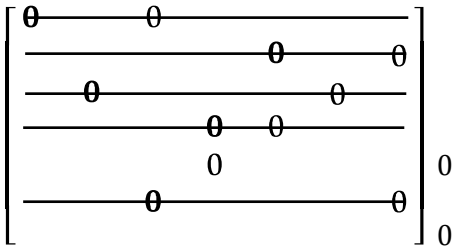
Consider a non-negative matrix containing the following zero entries

$$\begin{bmatrix} 0 & & 0 & & & & \\ & & & & 0 & & \\ & 0 & & & & 0 & \\ & & & 0 & 0 & & \\ & & & 0 & & & \\ & & 0 & & & & 0 \\ & & & & 0 & & \end{bmatrix}.$$

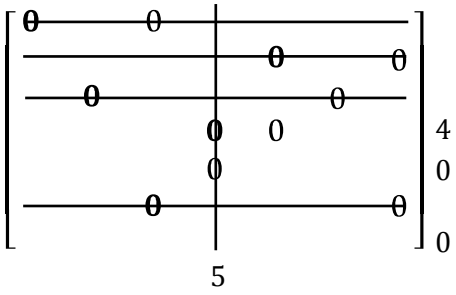
To obtain a zero cover follow the instructions as explianed in step 2 of König's agorithm, the bold zeros indicate the selected zeros

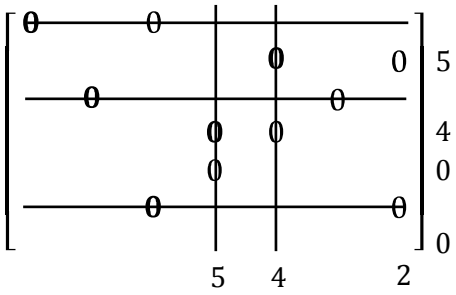
$$\begin{bmatrix} \mathbf{0} & 0 & & & \\ & & & \mathbf{0} & \\ & \mathbf{0} & & & 0 \\ & & \mathbf{0} & 0 & \\ & & 0 & & \\ & \mathbf{0} & & & 0 \\ & & & 0 & \end{bmatrix}.$$

Draw lines through the covered rows and apply labels (step 3)

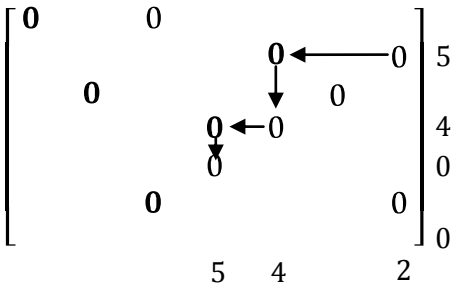


look for an uncovered zero and apply labels as explained in step 4

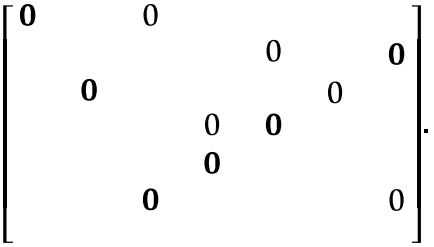




Now we have a breaktrough. Trace the path as described in step 5 indicated by the row and column labels.



By unselecting the selected zeros and selecting the unselected zeros on the path we increase the number of selected zeros by one



König's algorithm can once again be applied to this matrix with the new cover. The algorithm will however terminate at step 3 as this cover contains the maximum amount of independent zeros and no improvement is possible anymore. In this case it would be necessary to return to step5 of the Hungarian algorithm to obtain more zero entries in the matrix.

2.3.4 Auction algorithm

2.3.4.1 Introduction

Bertsekas developed a very efficient and rather intuitive algorithm for obtaining the optimal assignment of a given matrix using an auction based analogy. Every entry in the benefit matrix is viewed as the initial value a certain object has for the respective buyer. Buyers then bid on the objects offering them the greatest value and sets a price. As the price of an item increases its value decreases and after iteratively repeating this process a situation is obtained where each buyer obtains an object worth a maximum to him. In this section an intuitive explanation of the auction algorithm is given and the approach taken easy to follow. The algorithm is summarised in a stepwise fashion and an example of the algorithm executed on the same benefit matrix as used with the Hungarian method is also included. For a more detailed description of the algorithm the reader is referred to [4,14,28,37,32].

2.3.4.2 Auction algorithm

Let us consider the assignment problem as an economical situation where a group of buyers are bidding for a group of objects. Suppose that object has an associated benefit or value b_{ij} and also a price p_j. For any buyer i to obtain object j, buyer i would need to pay the price p_j. This implies that object j has a worth of $\{b_{ij} - p_j\}$ to buyer i. Any buyer is called 'happy' when he is assigned an object that maximizes this worth. This implies buyer i is *happy* when he is assigned object j that satisfies

$$b_{ij} - p_j = \max_{j=1,\dots,N}\{b_{ij} - p_j\}.$$

An assignment is said to be in equilibrium if all buyers are *happy* and an equilibrium assignment corresponds to an optimal assignment. The Auction algorithm gives us an iterative process for determining the prices of all the objects such that a buyer selects an object of maximal worth (this is actually only true for the naïve algorithm as a buyer just needs to be almost-happy as explained later on in the paragraph) and the process terminates when all objects are selected.

First a simple explanation of the auction algorithm referred to by Bertsekas in [4,29] as the *naïve* auction algorithm is introduced. This version is not complete as it does not always terminate but is useful to gain insight into the problem. The *naïve* algorithm is easily modified to obtain a fully functioning version of the auction algorithm.

The naïve Auction algorithm can be summarised as follows:

Set up an initial assignment: at this stage all buyers and objects are unassigned and all prices are zero. Consider the first buyer and allocate the object to him that he considers being worth a maximum. This would imply we now have one assigned pair $(1, j_1)$.

Raise the price of object j_1 by an amount γ_i such that the worth of this object to buyer i would be the same as the second best object. Raise p_i to

$$p_i = p_i + \gamma_i \qquad (2.9)$$

where

$$\gamma_i = v_i - w_i \qquad (2.10)$$

with v_i the best object value

$$v_j = \max_{j=1,\ldots,N} \{b_{ij} - p_j\} \qquad (2.11)$$

and w_i is the second best object value

$$w_j = \max_{j \neq j_i} \{b_{ij} - p_j\}. \qquad (2.12)$$

Continue this process by assigning the object of maximal worth to the respective buyer. If a buyer requires an object that is already assigned to another, the buyers exchange objects and the price of the required object is raised as described above.

The process terminates when all objects are assigned and all buyers are happy.

As in a regular auction process prices tend to increase, this is evident if one considers that the quantity γ_i, the bidding increment, can not be negative. The increase of prices leads to a situation where a buyer's initially favoured object could become less attractive. This simple process is almost sufficient to solve any assignment problem but there is one unfortunate setback. The trouble with the method becomes evident when one considers that raising the

price of one object to a level such that the worth of two objects is equivalent creates a situation where the bidding increment is zero. In such a case multiple objects offer maximal worth to the bidder. A situation is now created where several buyers contest objects of equivalent worth without raising the prices resulting in a loop without end. To prevent this possibility and break any endless cycles we introduce a disturbance mechanism that enforces an increase in the bidding increment every iteration. This can easily be done by introducing a small positive scalar ϵ, a person is then referred to as *almost happy* when the value assigned to the person is within ϵ of the optimal, that is,

$$b_{ij} - p_j \geq \max_{j=1,\ldots,N} \{b_{ij} - p_j\} - \epsilon. \quad (2.13)$$

To ensure a minimal bidding increment we redefine the bidding increment as

$$\gamma_i = v_i - w_i + \epsilon. \quad (2.14)$$

Any buyer will now be almost happy instead of happy at the end of each step and an assignment where we have only happy and almost happy buyers is referred to as an assignment almost at equilibrium. This reformulated version of the assignment problem can be shown to terminate in a finite number of steps. To see this consider that once an object receives a bid any buyer allocated to that object will be at least almost happy. An obvious and important question now arises; is this assignment optimal. The answer is that it depends on the size of ϵ. As a matter of fact it can be shown that if such an assignment is obtained after n iterations it is within $n\epsilon$ of being optimal. This is the case as we can consider such an assignment as a suboptimal assignment that differs from the optimal assignment by ϵ in at most n entries. Let us now consider an integer benefit matrix, which is generally the case, and if not such a matrix can be constructed by a suitable scaling of the benefit matrix given its entries are rational. The smallest possible variation from the optimal is now 1 as this is the smallest positive integer. Choosing ϵ such that $n\epsilon < 1$ then ensures that the assignment is optimal. It follows that we are ensured of achieving an optimal assignment for an integer benefit matrix if

$$\epsilon < \frac{1}{n}. \quad (2.15)$$

The algorithm will terminates quicker when choosing larger ϵ. This is analogous to more aggressive bidding by the buyers.

2.3.4.3 Example of the auction algorithm

For clarity's sake we'll construct a price vector $\boldsymbol{P}$, worth matrix $\boldsymbol{W}$ and assignment matrix $\boldsymbol{A}$ to complement the procedure.

Choose $\epsilon = 0.2$, this agrees with requirement (2.15).

Iteration 1

$$\boldsymbol{B} = \begin{bmatrix} 2 & 5 & 7 & 3 \\ 2 & 3 & 3 & 1 \\ 6 & 7 & 5 & 5 \\ 1 & 9 & 2 & 0 \end{bmatrix}$$

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 2.2 & 0 \end{bmatrix}$$

$$\boldsymbol{W} = \begin{bmatrix} 2 & 5 & 4.8 & 3 \\ 2 & 3 & 0.8 & 1 \\ 6 & 7 & 2.8 & 5 \\ 1 & 9 & -0.2 & 0 \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Iteration 2

$$\boldsymbol{B} = \begin{bmatrix} 2 & 5 & 7 & 3 \\ 2 & 3 & 3 & 1 \\ 6 & 7 & 5 & 5 \\ 1 & 9 & 2 & 0 \end{bmatrix}$$

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1.2 & 2.2 & 0 \end{bmatrix}$$

$$\boldsymbol{W} = \begin{bmatrix} 2 & 3.8 & 4.8 & 3 \\ 2 & 1.8 & 0.8 & 1 \\ 6 & 5.8 & 2.8 & 5 \\ 1 & 7.8 & -0.2 & 0 \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Iteration 3

$$\boldsymbol{P} = [1.2 \quad 1.2 \quad 2.2 \quad 0]$$

$$\boldsymbol{W} = \begin{bmatrix} 0.8 & 3.8 & 4.8 & 3 \\ 0.8 & 1.8 & 0.8 & 1 \\ 4.8 & 5.8 & 2.8 & 5 \\ -0.2 & 7.8 & -0.2 & 0 \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Iteration 4

$$\boldsymbol{P} = [1.2 \quad 9.2 \quad 2.2 \quad 0]$$

$$\boldsymbol{W} = \begin{bmatrix} 0.8 & -4.2 & 4.8 & 3 \\ 0.8 & -6.2 & 0.8 & 1 \\ 4.8 & -2.2 & 2.8 & 5 \\ -0.2 & -0.2 & -0.2 & 0 \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \end{bmatrix}$$

Iteration 5

$$\boldsymbol{P} = [1.2 \quad 9.2 \quad 2.2 \quad 0.4]$$

$$\boldsymbol{W} = \begin{bmatrix} 0.8 & -4.2 & 4.8 & 2.6 \\ 0.8 & -6.2 & 0.8 & 0.6 \\ 4.8 & -2.2 & 2.8 & 4.6 \\ -0.2 & -0.2 & -0.2 & -0.4 \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \end{bmatrix}$$

The algorithm now terminates as all buyers have been assigned objects and the resultant assignment matrix is the permutation matrix that corresponds to the optimal assignment.

2.3.5 *Invisible hand algorithm: Solving the assignment problem using a statistical physics approach*

2.3.5.1 *Introduction*

Kosowsky and Yuille managed to use typical methods taken from statistical physics and cast the assignment problem into such a framework. Firstly an energy function is defined corresponding to an assignment and this energy function is negated with the optimal assignment corresponding to the assignment matrix that minimizes this energy function. Then a probability distribution and related partition function of the system is computed. The discrete partition function is embedded into a continuous form with the creative application of Dirac delta functions and their Fourier transforms. Using this approach allows the derivation of a continuous system model for this inherently discrete problem with the same extrema. The Saddle point approximation and method of steepest descent can now be applied to this new continuous problem to find these extrema. In [25] Kosowsky and Yuille shows that finding the critical points of the new continuous model under the right conditions exactly solves the assignment problem. This results in the Invisible hand algorithm that always converges to the optimal solution of the assignment problem. They also apply a result from Sinkhorn [23] that finds a doubly stochastic approximation to a matrix using an iterative normalisation process to find solutions to the saddle point approximation. The derivation and discussion below is very similar to the one found in [5,25] with more elaborate discussions concerning certain details that could do with some clarification.

2.3.5.2 *Invisible hand algorithm*

Consider $\boldsymbol{A}$ to be the assignment matrix, $\boldsymbol{A}$ is a permutation matrix with the optimal assignment corresponding to the matrix $\boldsymbol{A}$ that maximises the sum $T = \sum_{i,j} a_{ij}\, b_{ij}$ with $a_{ij} \in \boldsymbol{A}$ and $b_{ij} \in \boldsymbol{B}$ the benefit matrix. We can now define an energy function

$$E[\boldsymbol{A}] = -T = -\sum_{i,j} a_{ij}\, b_{ij} \qquad (2.16)$$

Finding an assignment matrix $\boldsymbol{A}$ that minimizes $E[\boldsymbol{A}]$ then corresponds to an optimal solution of the assignment problem. From the statistical physics viewpoint the assignment problem can be viewed as a system with an energy associated with every state as defined by equation (2.16) and an associated probability of occupying that state described by what is known in statistical physics as the partition function. Firstly we consider the probability of the system being in a particular state as proportional to the exponential of the energy function

$$P[\boldsymbol{A}] \propto e^{-\beta E_{ass}[\boldsymbol{A}]} \tag{2.17}$$

where $\beta = 1/T$ and T represents a temperature parameter that can be adjusted for optimisation purposes. The proportionality sign can be rewritten as an equality sign if correctly normalised

$$P[\boldsymbol{A}] = \frac{1}{\mathbf{Z}} e^{-\beta E_{ass}[\boldsymbol{A}]} \tag{2.18}$$

where $\mathbf{Z}$ is known as the partition function

$$\mathbf{Z} = \sum_{\boldsymbol{\mathcal{A}}} e^{-\beta E_{ass}[\boldsymbol{A}]} \tag{2.19}$$

and the sum is taken over the set $\boldsymbol{\mathcal{A}}$ consisting of all admissible configurations of matrix $\boldsymbol{A}$ (i.e. the set of all permutation matrices of size N). The partition function $\mathbf{Z}$ serves the purpose of acting as a normalizing factor in equation (2.18) and ensures that the sum of probabilities over all possible states is one. This leaves us with the well known Gibbs distribution. The computation of the partition function turns out to be a problem as summing over all valid states in equation (2.19) would be at least as computationally expensive as simply evaluating all valid assignments and choosing the maximum as discussed in Section 1.1. In [25] Kosowsky and Yuille derives a continuous version of the partition function by embedding the space of binary assignment matrices into the space of $N \times N$ matrices with entries from $\mathbb{R}$ using a technique from Peterson and Soderberg [39]. The partition function is rewritten as

$$\mathbf{Z} = \sum_{\boldsymbol{\mathcal{A}}} \int [ds] e^{-\beta E_{ass}[\boldsymbol{S}]} \prod_{i,a} \delta(s_{ia} - a_{ia}) \tag{2.20}$$

where a new set of continuous variables $\boldsymbol{S}$ is introduced. For each element from set $\boldsymbol{S}$ a Dirac delta function is introduced as well and together these allow one to write the partition function as continuous function that evaluates to the exact same value as its original form shown by equation (2.19). This is because only the discrete points corresponding to the original discrete system contribute due to the Dirac delta functions. A clearer way of writing equation (2.20) is

$$\mathbf{Z} = \sum_{\boldsymbol{\mathcal{A}}} \iint_{\dots} \int e^{-\beta E_{ass}[\boldsymbol{S}]} \prod_{i,a} \delta(s_{ia} - a_{ia}) ds_{11} \dots ds_{NN} \tag{2.21}$$

Illustrating that one integral and one Dirac delta are introduced for each of the entries a_{ia}. It is however still inconvenient to work with a function of this form as the Dirac delta functions are difficult to manipulate. This problem can be overcome by introducing a Fourier transformed representation of these delta functions. From the theory of Fourier transforms

$$\Im[\delta(x)] = 1 \tag{2.22}$$

or

$$\Delta[\omega] = \Im[\delta(x)] = \int_{-\infty}^{\infty} \delta(x) e^{-j\omega x}\, dx = 1 \tag{2.23}$$

and

$$\Im^{-1}[1] = \delta(x) \tag{2.24}$$

or

$$\Im^{-1}[1] = \int_{-\infty}^{\infty} 1.\, e^{j\omega x}\, d\omega \tag{2.25}$$

and when taking the integral along the imaginary axis we get

$$\delta(x) = \Im^{-1}[1] = \int_{-j\infty}^{j\infty} e^{-\omega x}\, d\omega. \tag{2.26}$$

In the above expressions constant scaling factors are neglected as they will not influence the extreme points of $\mathbf{Z}$. Using expression (2.26) we can rid equation (2.20) of all the Dirac deltas by introducing a set of N^2 new variables $\boldsymbol{U}$ corresponding to ω in equation (2.26). Equation (2.20) now becomes

$$\mathbf{Z} = \sum_{\boldsymbol{\mathcal{A}}} \int \int_{\mathrm{I}} [ds][du] e^{-\beta E_{ass}[\boldsymbol{S}]}\, e^{-\sum_{ia} u_{ia}(s_{ia} - a_{ia}).} \tag{2.27}$$

As the summation over all admissible states of matrix $\boldsymbol{A}$ is computationally expensive we impose the column constraints of set $\boldsymbol{\mathcal{A}}$ by adding an additional set of delta functions in the integral as follows

$$\mathbf{Z} = \sum_{\boldsymbol{\mathcal{A}}} \int \int_{\mathrm{I}} [ds][du] e^{-\beta E_{ass}[\boldsymbol{S}]}\, e^{-\sum_{ia} u_{ia}(s_{ia} - a_{ia})} \prod_{a} \delta(\sum_{i} s_{ia} - 1). \tag{2.28}$$

Here we can once again remove the delta functions using equation (2.26) again introducing a new vector of variables $[p_a]$ resulting in

$$\mathbf{Z} = \sum_{\boldsymbol{\mathcal{A}}} \int \int_{I} \int_{I} [ds][du][dp]\, e^{-\beta E_{ass}[\boldsymbol{S}]}\, e^{-\sum_{ia} u_{ia}(s_{ia} - a_{ia})} e^{-\sum_{a} p_a(\sum_{i} s_{ia} - 1)}. \tag{2.29}$$

To completely remove the summation sign at the front of equation (2.29) the row constraints of the elements in set $\boldsymbol{\mathcal{A}}$ needs to be enforced. It is possible to do this by interchanging the order of summation and integration and summing over all the relevant states. Rewriting the a_{ij} dependant terms in equation (2.29)

$$\sum_{\boldsymbol{\mathcal{A}}} e^{\sum_{ia} u_{ia} a_{ia}} = \sum_{\boldsymbol{\mathcal{A}}} \prod_{i} e^{\sum_{a} u_{ia} a_{ia}} \tag{2.30}$$

and recalling that

$$\sum_{j} a_{ij} = 1 \tag{2.31}$$

we can write

$$\sum_{\boldsymbol{\mathcal{A}}} \prod_{i} e^{\sum_{a} u_{ia} a_{ia}} = \prod_{i} \left(\sum_{j} e^{u_{ij}} \right) \tag{2.32}$$

$$= e^{\sum_{i} \ln \left(\sum_{j} e^{u_{ij}} \right)}.$$

Finally the partition function can be written as

$$\mathbf{Z} = \int \int_{I} \int_{I} [ds][du][dp]\, e^{-\beta E_{ass}[\boldsymbol{S}]}\, e^{-\sum_{ia} u_{ia} s_{ia}} e^{\sum_{i} \ln \left(\sum_{j} e^{u_{ij}} \right)} e^{-\sum_{a} p_{a} \left(\sum_{i} s_{ia} - 1 \right)} \tag{2.33}$$

or

$$\mathbf{Z} = \int \int_{I} \int_{I} [ds][du][dp]\, e^{-\beta E_{eff}[\boldsymbol{S}, \boldsymbol{U}, \boldsymbol{P}]} \tag{2.34}$$

where we define the new term the effective energy E_{eff} as

$$E_{eff}[\boldsymbol{S}, \boldsymbol{U}, \boldsymbol{P}: \beta] = E_{ass}[\boldsymbol{S}] + \frac{1}{\beta} \sum_{i,j} s_{ij}\, u_{ij} - \frac{1}{\beta} \sum_{i} \ln \left(\sum_{j} e^{u_{ij}} \right) + \sum_{j} p_{j} \left(\sum_{i} s_{ij} - 1 \right). \tag{2.35}$$

Note that $\boldsymbol{P}$ has been rescaled as $p_j \longmapsto \frac{1}{\beta} p_j$ ensuring that the critical points of $\boldsymbol{P}$ will scale properly with $\beta = \frac{1}{T}$.

As we now have a continuous form of the partition function it is possible to apply the saddle point approximation. First we determine the critical points of the partition function

$$0 = \frac{\partial E_{eff}}{\partial s_{ij}} = -b_{ij} + \frac{1}{\beta} u_{ij} + p_a$$

$$0 = \frac{\partial E_{eff}}{\partial u_{ij}} = \frac{1}{\beta} s_{ij} - \frac{1}{\beta} \frac{e^{u_{ij}}}{\sum_k e^{u_{ik}}} \qquad (2.36)$$

$$0 = \frac{\partial E_{eff}}{\partial p_j} = \sum_i s_{ij} - 1.$$

Using the above equations it is possible to eliminate $\boldsymbol{U}$ resulting in the following equations

$$s_{ij} = \frac{e^{\beta(a_{ij} - p_j)}}{\sum_b e^{\beta(a_{ib} - p_b)}}, \qquad \forall i, j$$

$$\sum_i s_{ij} = 1, \qquad \forall i. \qquad (2.37)$$

It is also possible to remove the $\boldsymbol{S}$ dependence of equations (2.37) resulting in a set of N equations for N unknowns, the entries of the vector $\boldsymbol{P}$

$$\sum_i \frac{e^{\beta(a_{ij} - p_j)}}{\sum_b e^{\beta(a_{ib} - p_b)}} = 1\,, \qquad \forall j. \qquad (2.38)$$

Given a vector $\boldsymbol{P}$ that solves equations (2.38) the critical points of the partition function are then completely characterised by the subsequent solutions to equations (2.36). This implies that the optimal assignment is solved by the solution to equations (2.38) as the temperature parameter tends to zero. Solving equations (2.38) proves to be troublesome when using an analytical approach but it is easily solved by iterative methods such as the method of steepest descent. Kosowsky and Yuille also show that it is possible to solve equations (2.38) by applying a result known as Sinkhorn's theorem [23]. Sinkhorn's theorem shows that it is possible to write any square matrix as a doubly stochastic matrix front and back multiplied by a diagonal matrix and that a square matrix converges to such a doubly stochastic matrix by the process of iterative row and column normalisation (see theorem 2.2).

The resulting algorithm can be summarised as follows:

Initialise $\beta \leftarrow \beta_0$

Process 1: Repeat until $\beta = 0$

Initialise s_{ij} $\forall i, j$

$s_{ij} \leftarrow e^{\beta(a_{ij} - p_j)}$

Process 2: Repeat until $\boldsymbol{S}$ converges

Normalise row-wise $\forall i, j$

$$s_{ij} \leftarrow \frac{s_{ij}}{\sum_j s_{ij}}$$

Normalise column-wise $\forall i, j$

$$s_{ij} \leftarrow \frac{s_{ij}}{\sum_i s_{ij}}$$

End

Use Sinkhorn's theorem to compute $\boldsymbol{P}$

Decrease β

End

Computational complexity

This approach used by Kosowsky and Yuille even though fascinating, is far inferior to an algorithm such as Bertsekas' auction algorithm when computational complexity is considered. Kosowsky and Yuille do not offer any bounds on their algorithm's computational performance.

2.4 OTHER IMPORTANT CONCEPTS

2.4.1 Hogg and Huberman's Computational Ecology

2.4.1.1 Introduction

Various man made and natural systems exist with individual units acting together in some sort of a group resulting in group dynamics that could be quite different from the behaviour of the individual units in the group. This combined behaviour is more complex and usually far more intelligent than that of the individual units. Various disciplines have looked into the science of describing such behaviour. With the modern development of computing power we find many strategies in the field of computational intelligence that solve problems using an approach where large numbers of simple individuals act together in a group, for example ant colony optimisation, particle swarm optimisation [40], population based incremental learning and genetic algorithms [41,42] to name a few. These methods have proven very successful in problem solving but it is important to understand that the approach taken here is one of using sheer computing power to model the individual units themselves and then subsequently observe the resultant overall dynamics. The subject described here as a Computational Ecology differs significantly from these methods when the methodology is considered. For the Computational Ecology the idea is firstly the identification of important elements in a large system, then describing these elements and their relation and finally deriving equations describing the overall dynamics of the system. Hogg and Huberman manage to derive such equations for certain systems particularly situations of resource allocation that would typically occur in structures such as computer networks or other large distributed data processing systems and they discuss these examples frequently [6,43]. In accordance with methods taken from statistical mechanics the system descriptions are mostly concerned with average behaviour in groups. In [43] they even compare their model to Adam Smith's famous "Invisible hand" principle [44]. The problem is restricted to an agent/resource allocation problem where a group of individual units referred to as agents compete for and share a finite amount of resources. In [45] Hogg and Huberman investigate the performance of such systems subject to phenomena such as delayed and imperfect information and show that such conditions can lead to chaotic behaviour. They also investigate some methods of stabilising such chaotic behaviour and in [43] one can find simulations that agree with the predictions of the model. The system description is encoded using differential equations and probability distributions and disturbances are artificially introduced using error functions. Considering the time that these developments were made it is quite clear why Hogg and Huberman attempted to develop these analytic models. Currently however the utility of this approach for gaining a handle on systems

that are hard to describe such as computer networks seems to be fading as brute force modelling techniques are much easier to develop and offer more accurate results. The result being that the Hogg and Huberman model is not finding much application and their results are mostly of theoretical significance. There are however some scientists that found application for the model recently such as Ushio and Yamasaki [46]. The OACE algorithm as described in Chapter 3 of this document uses a small part of the Computational Ecology model but it is more accurate to consider the Computational Ecology model as an inspiration to the OACE algorithm rather than a basis. In this paragraph the derivation of the Computational Ecology model is covered and the results relevant to the OACE algorithm interpreted. A thorough understanding of these derivations are however quite unnecessary for comprehension of the OACE algorithm and the reader without a redundant drive for inquisition is advised to only study equations (2.47) and (2.48) and their interpretation.

2.4.1.2 Derivation of dynamical equations

Assume that we are working in an environment were a finite number of agents, say N, are competing for a finite number of resources, each agent with a certain degree of *happiness* with each respective resource. This *happiness* or preference is described by a variable related to the probability of the agent moving to a certain resource in the future. Figure 2.3 attempts to demonstrate these relationships graphically.

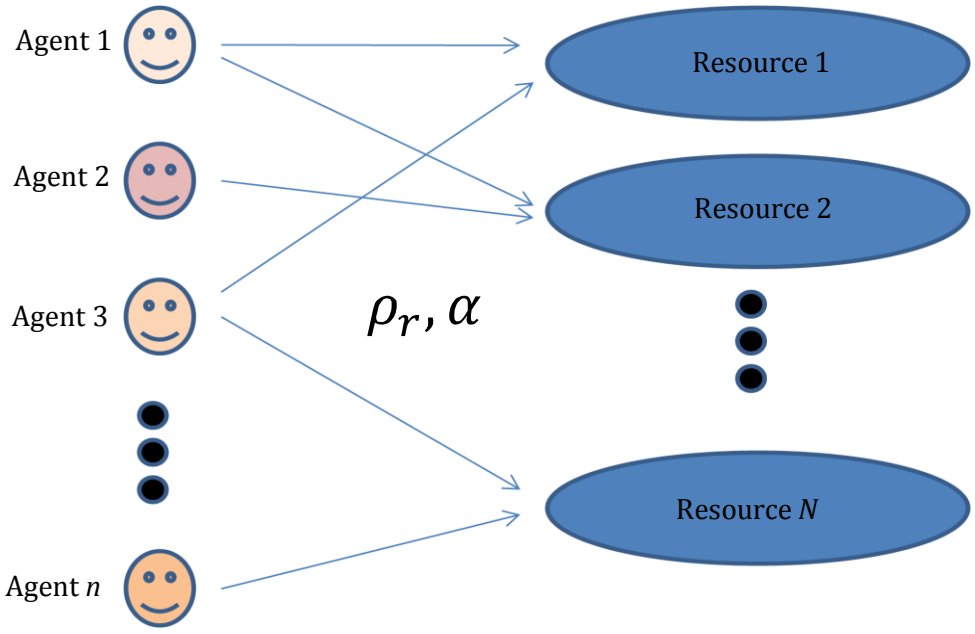


FIGURE 2.3: Graphical demonstration of the agent resource allocation process.

Let us consider how agents make choices in a given time interval Δt. Let α be the average number of choices made by an agent in a unit of time. If we choose Δt small enough such that $\alpha\Delta t < 1$ we can say that the probability of an agent making a choice in the interval Δt is proportional to $\alpha\Delta t$. The next step is to find the probabilities of an agent moving from one resource to another, this implies that the agent needs to evaluate the new option and see whether it will be happier with such a choice than its current choice. To do this, two things need to be quantified; the probability of an agent making a choice in a given amount of time and the probability of an agent preferring one resource to the other. Let us not now dwell on finding a way to calculate this probability as this could be very problem specific but assume that the probability of an agent preferring resource 1 to resource 2 is ρ. Then the probability of preferring resource 2 to resource 1 is obviously $(1-\rho)$ and we can now write the probabilities of an agent moving from one resource to the other as

$$P(1 \to 2) = \alpha\Delta t(1-\rho), \tag{2.39}$$

$$P(2 \to 1) = \alpha\Delta t\rho. \tag{2.40}$$

Defining these probabilities allows us to compute distributions for the probabilities of k_{12} agents changing from resource 1 to resource 2 using the well known binomial distribution given by

$$\mathrm{Bi}(N,\rho,k) = \binom{N}{k}\rho^k(1-\rho)^{N-k}. \tag{2.41}$$

To find the global behaviour from these local changes we consider the quantity $P_1(a,t)$ the probability that at time t, a agents are using resource 1. Having a total of n agents we can write

$$\sum_{a=0}^{n} P_1(a,t) = 1. \tag{2.42}$$

Now we can use methods pertaining to Calculus to derive dynamical equations. First we consider the quantity $P_1(a,t+\Delta t)$. We can write this in terms of $P_1(a',t)$ where a' is now the amount of agents using strategy 1 at time t and a net number of $a-a'$ agents moved from resource 1 to resource 2. Since a could take any value between 0 and n we can write

$$P_1(a,t+\Delta t) = \sum_{a'=0}^{n} P_1(a',t)P(a-a',a'). \tag{2.43}$$

Here $P_1(\delta \equiv a - a', a')$ is the probability of a net change of δ agents during the interval from strategy 2 to strategy 1. If the time interval Δt is chosen to be very small we can approximate δ in the lowest order as $\delta = -1, \delta = 0$ or $\delta = 1$. This means that for very small time intervals we only need to consider a change of one agent between the two resources. Now we can rewrite this probability using equations (2.39) and (2.40) as

$$P_1(\delta = -1, a') \propto a' P(1 \to 2)$$

or

$$P_1(\delta = -1, a') = a' a \Delta t (1 - \rho).$$

Similarly

$$P_1(\delta = 1, a') = (n - a') a \Delta t \rho$$

and from the assumption that we're only considering small δ

$$P_1(\delta = 0, a') = 1 - P_1(\delta = -1, a') - P_1(\delta = 1, a')$$

or

$$P_1(\delta = 0, a') = 1 - a' a \Delta t (1 - \rho) - (n - a') a \Delta t \rho.$$

Using this approximation equation (2.43) becomes

$$P_1(a, t + \Delta t) = P_1(a, t) P_1(\delta = 0, a') + P_1(a + 1, t) P_1(\delta = -1, a') + P_1(a - 1, t) P_1(\delta = 1, a')$$

as we only need to consider values of a' that is within one unit from a for this approximation, that is $a' = a$; $a = a + 1$ and $a = a - 1$. This assumption is similar to only considering linear orders of Δt to be significant in equation (2.43). On substitution of the above probabilities we get

$$\begin{aligned} P_1(a, t + \Delta t) = {} & P_1(a, t) + P_1(a, t)[-a(1 - \rho) - (n - a)\rho]\alpha \Delta t + P_1(a + 1, t)(a + 1)(1 - \rho_+)\alpha \Delta t \\ & + P_1(a - 1, t)(n - a + 1)(\rho_-)\alpha \Delta t \end{aligned}$$

which can be rewritten as

$$\begin{aligned} \frac{P_1(a, t + \Delta t) - P_1(a, t)}{\alpha \Delta t} = {} & P_1(a, t)[-a(1 - \rho) - (n - a)\rho] + \\ & P_1(a + 1, t)(a + 1)(1 - \rho_+) + P_1(a - 1, t)(n - a + 1)(\rho_-). \end{aligned} \quad (2.44)$$

Now we've managed to derive an equation that looks like the definition of a derivative so in the limit of $\Delta t \to 0$ we have

$$\alpha^{-1}\frac{dP_1}{dt} = \lim_{\Delta t \to 0} \frac{P_1(a, t + \Delta t) - P_1(a, t)}{\alpha \Delta t} \tag{2.45}$$

$$\alpha^{-1}\frac{dP_1}{dt} = P_1(a,t)[-a(1-\rho) - (n-a)\rho] + P_1(a+1,t)(a+1)(1-\rho_+) + P_1(a-1,t)(n-a+1)(\rho_-).$$

In the above equation the probabilities $\rho_\pm$ is ρ evaluated for $a \pm 1$. Equation (2.45) describes the system we defined completely but it is not a very useful format and to achieve more useable results we consider the average system behaviour. Hogg and Huberman arrive at a result bearing utility by calculating average system quantities. Using the definition of an expectation

$$f_{ave} = \sum_{a=0}^{n} f(a)\, P_1(a,t) \tag{2.46}$$

it is possible to calculate the average number of agents using a resource at a given time. We now find the Hogg and Huberman's result that is of greatest value to the development of the OACE algorithm

$$\frac{d\bar{n}_i}{dt} = \alpha(n\bar{\rho}_i - \bar{n}_i) \tag{2.47}$$

where

n	total number of agents
$\bar{n}_i$	average number of agents using resource *i*
α	degree of change/rate of decision making
$\bar{\rho}_i$	average probability of agents preferring resource *i* to other resources

Equation (2.47) describes the development of the number of agents using resource i with time in terms of the average probability of preferring that resource and the decision rate. To arrive at equation (2.47) the expectation operation was applied and the resulting equation is no longer of a stochastic nature. For the sake of digital implementation we write equation (2.47) in normalised and discretized form as [46]

$$f_i(k+1) = f_i(k) + \alpha\big(\rho_i(k) - f_i(k)\big) \tag{2.48}$$

where $f_i(k)$ is the fraction of agents using resource i at time k.

Equation (2.48) can be interpreted as an update equation where α determines the degree of influence of ρ on the next state of f. The derivation of this result seems rather excessive if one considers the simplicity of this equation, but Hogg and Huberman use this result to illustrate some interesting aspects of distributed system dynamics. Consider constructing a problem such that $\alpha = 1$ then equation (2.48) reduces to $f_i(k+1) = \rho_i(k)$. This implies the dynamics of the next state is governed completely by the probability term ρ, a term that Hogg and Huberman gives little attention but is of significance to the OACE algorithm.

CHAPTER 3: THE OACE ALGORITHM

3.1 INTRODUCTION

It is tempting to try and cast the assignment problem into the Computational Ecology model as the relationship between the assignment problem and the Computational Ecology model seems rather natural; the Computational Ecology model is based on the idea of agents trying to find optimal resources and the assignment problem strives to match individuals and objects or jobs as best as possible. Hogg and Huberman's model consisting of a dynamic set of equations allows for the investigation of the behaviour of agent groups under various conditions. Although the model is not well suited to general applications or problem solving it offers information that increases insight into distributed systems and its various properties and can be used to improve the design process. Scientists and engineers will however try and find the maximum utility in the model as was done by Ushio and Yamasaki in [46] where they applied the model to a packet routing problem such as generally found in a telecommunications environment. In this document a far simpler version of the Computational Ecology model is applied to the assignment problem resulting in a simple algorithm called the OACE (Optimal Assignment by Computational Ecology) algorithm that converges to the optimal or near optimal assignment as summarised by Vermaak in [9]. The resulting algorithm can be viewed as an iterative weighted normalisation process when one removes the Computational Ecology shell and as such it may be more appropriate to consider the Computational Ecology model as an inspiration to the OACE algorithm rather than a necessary structure. It is also important to realise that this method is completely deterministic even though certain terms are referred to as probabilities and the name Computational Ecology may be associated with certain stochastic methods.

In section 3.1.2 the approach of considering the assignment problem from a Computational Ecology viewpoint is explained. It is based on the idea of groups of agents initially evenly distributed among various resources and moving towards certain preferred resources corresponding to the optimal assignment. The algorithm is then given in a step-wise manner in section 3.1.3 and it is quite possible to follow this without reading section 3.1.2 and view the algorithm merely as a process of iterative weighted normalisation. The Computational Ecology structure does however offer a comfortable mental platform and allows for an intuitive interpretation of the dynamics of the algorithm. An analysis of the convergence and performance follows in section 3.1.5 and 3.1.6&7 respectively and finally a discussion of the algorithm follows including its advantages and limitations.

3.2 APPLICATION OF THE COMPUTATIONAL ECOLOGY STRUCTURE TO THE ASSIGNMENT PROBLEM

The most obvious association between the assignment problem and the Computational Ecology model is to consider the *individuals* of the assignment problem as *agents* and the *objects* as *resources*. It turns out though that this simple association is not sufficient as it does not allow the equations in the Computational Ecology model to evolve meaningfully as the Hogg-Huberman model does not deal with individual agents but with average group behaviour. A formulation that proved successful was to consider each individual from the assignment problem as a group of agents. This implies that instead of allocating a single individual to a resource a fraction of this total group of agents is allocated to each resource. The agents are distributed between the different resources depending on the probabilities of preferring certain resources to others which in turn depends on the entries from the benefit matrix and the number of agents from different groups using the same resource.

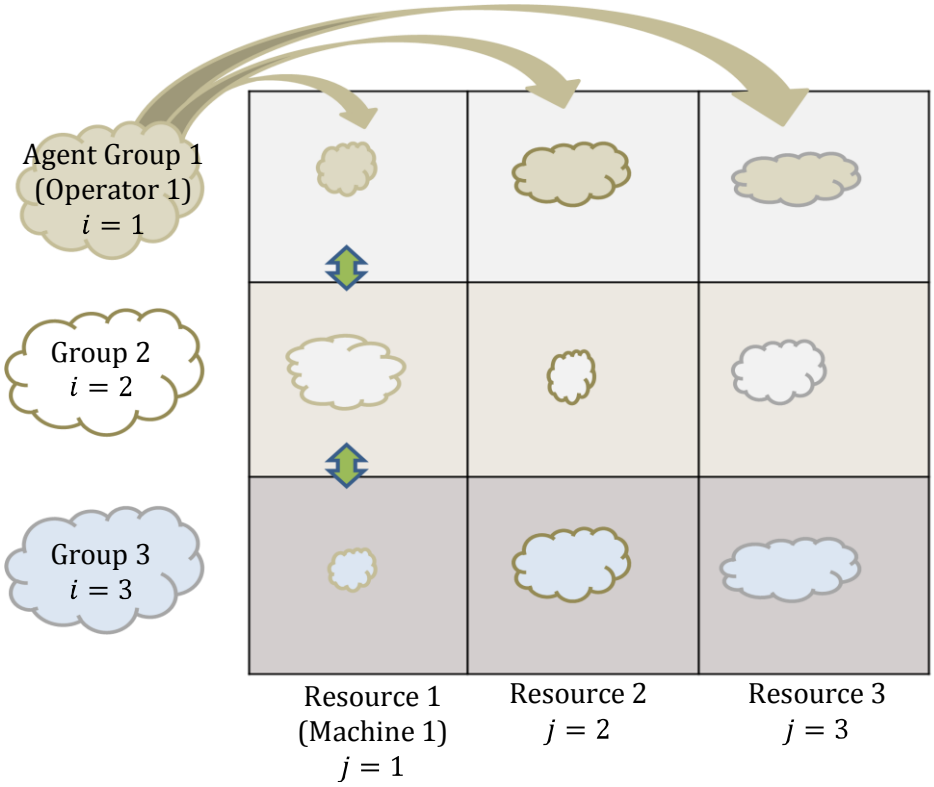


FIGURE 3.1: Graphical illustration of the relations and interactions of the different agent groups and resources for a $\mathbf{3 \times 3}$ benefit matrix.

Figure 3.1 shows a schematic representation of a possible way the agents could distribute themselves in the case of a 3×3 matrix. Every row corresponds to one agent group and every column to a resource. Each resource is allocated a certain fraction of agents from each group depending on the benefit of the specific match and the number of agents from other groups using that same resource. This step is executed in its simplest form as a weighted row-wise normalisation as described by equation (3.1). The total of all the fractions of agents allocated to a specific resource is now forced to sum to unity by the next step of a column-wise normalisation as described by equation (3.2). This is analogous to agents moving between different groups for the same resource. The column-wise normalisation disturbs the previous step's row-wise normalisation but rows will sum to a total in the proximity of one. The column-wise normalisation will ensure interdependency of row and column population densities. Populations are now updated according to equation (2.48) from the Computational Ecology model. Mathematically the development of the populations of agents between the different resources is described by a population matrix $\boldsymbol{F}$. Entry f_{ij} develops according to equation (2.48) and probability (3.1) and describes the fraction of agents in group i allocated to resource j. $\boldsymbol{F}$ develops per iteration, hence $\boldsymbol{F} = \boldsymbol{F}(k)$. The iterative implementation of this process results in a population distribution that either corresponds to the optimal assignment or a near optimal assignment.

3.3 OACE ALGORITHM

The OACE algorithm for solving the optimal assignment problem can be summarised in the following stepwise format.

1) Initialise $\boldsymbol{F}$

$$f_{ij} = \frac{1}{n} \quad \forall i, j$$

with n the dimension of the matrix.

2) Determine the probability of agent i choosing task j according to equation (3.1).

$$\rho_{ij}(k) = \frac{f_{ij}(k) \times b_{ij}}{\sum_j f_{ij}(k) \times b_{ij}} \qquad (3.1)$$

where

ρ_{ij} The probability that agents in group i would prefer resource j

f_{ij} Fraction of agents from group i allocated to resource j

b_{ij} Benefit of allocating agents in group i to resource j

3) Normalise ρ column-wise according to equation (3.2)

$$\rho_{ij} = \frac{\rho_{ij}}{\sum_i \rho_{ij}} \qquad (3.2)$$

4) Update f according to equation (2.48)

$$f_{ij}(k+1) = f_{ij}(k) + \alpha(\rho_{ij}(k) - f_{ij}(k))$$

5) Repeat until one of the termination criteria is met

- $\boldsymbol{F}$ becomes row-dominant
- $\boldsymbol{F}(k)$ converges (when $\boldsymbol{F}(k) = \boldsymbol{F}(k+1)$)

if neither criteria are met return to step 2.

6) Approximate $\boldsymbol{F}(k)$ with its closest permutation matrix.

3.3.1 *Initialisation of the algorithm*

The most natural initialisation of the algorithm is to initialise all entries of $\boldsymbol{F}$ as

$$f_{ij} = \frac{1}{n} \quad \forall i, j$$

with n the dimension of the matrix. This would agree with the idea of an agent group initially distributing itself equally between all the different resources. It is however actually possible to initialize f_{ij} as any positive, non-zero real number as long as all entries are initialized to the same number. This can be seen when considering that if the f_{ij} terms are independent of i and j in equation (3.1) they cancel.

3.3.2 *Termination of the algorithm*

The simplest criterion for termination of the algorithm is that the algorithm is repeated until the population matrix converges to a stationary matrix. This occurs when $\boldsymbol{F}(k) = \boldsymbol{F}(k+1)$ or for the sake of practicality when $\|\boldsymbol{F}(k) - \boldsymbol{F}(k+1)\| < \epsilon$, where ϵ is a sufficiently small scalar and an applicable norm is used.

Another criterion that is useful for termination of the algorithm is termination when the population matrix becomes row dominant. A matrix will be called row (resp. column) dominant if the maximum element in each row (resp. column) is located in a different column (resp. row). This generally leads to much improved results concerning the number of iterations the algorithm needs to produce a result. In [25] Kosowsky and Yuille use the same criterion for the termination of one of their algorithms. As the convergence to a row dominant matrix can not be guaranteed the algorithm terminates on whichever of the above criteria are met first.

3.3.2.1 *Approximation of $\boldsymbol{F}$ by a permutation matrix*

As the most natural representation of an assignment when working with matrices is a permutation matrix, matrix $\boldsymbol{F}$ needs to be approximated by a permutation matrix once the OACE algorithm terminates. The population matrix converges to a stationary sparse matrix or a row-dominant matrix under application of the OACE algorithm and we will refer to this matrix as $\boldsymbol{F}(k)|_{k\to\infty}$. Such a matrix is near the doubly stochastic polytope although not necessarily contained within it. To find the best permutation matrix approximation of $\boldsymbol{F}(k)|_{k\to\infty}$ we need to find the permutation matrix that is closest to $\boldsymbol{F}(k)|_{k\to\infty}$ under some norm, the Frobenius (least squares) norm is sufficient. We can also state this as finding the closest extreme point of the doubly stochastic polytope to $\boldsymbol{F}(k)|_{k\to\infty}$. Of course finding the closest permutation matrix to $\boldsymbol{B}$

actually solves the assignment problem and usually the most efficient way to do this is by using algorithms that solve the assignment problem such as the Auction algorithm. The advantage of applying the OACE algorithm is that we end up with a matrix $\boldsymbol{F}(k)|_{k\to\infty}$ that is very easy to approximate by a permutation matrix and we do this using a procedure that can be classified as a so-called greedy algorithm.

If $\boldsymbol{F}(k)|_{k\to\infty}$ is row-dominant the closest permutation matrix is found by replacing the largest element in each row with a 1 and all other elements by 0. A row-dominant $\boldsymbol{F}(k)|_{k\to\infty}$ will always render a valid permutation matrix by this simple procedure.

If $\boldsymbol{F}(k)|_{k\to\infty}$ is not row-dominant it will be a stationary sparse matrix. A permutation matrix can now generally be obtained by replacing the maximum entry in a row with a 1 and all other entries in that row with a 0 as in the row-dominant case. This procedure does however not always render a permutation matrix as it is possible for $\boldsymbol{F}$ to converge to a matrix with two or more maximal row entries occurring in the same column. In such a case a simple logical procedure can be used to convert the greedy algorithm's result (referred to as $\boldsymbol{G}$) to a valid permutation matrix.

Step 1: Test whether $\boldsymbol{G}$ is a valid permutation matrix (A simple method of doing this is to add the totals of all the entries in each row and see if it equals the order of the matrix. Do the same for the columns).

Step 2: If $\boldsymbol{G}$ is not a valid permutation matrix, find all the columns that do not contain a 1 and all the columns that contain more than one 1.

Step 3: Move any extra 1s in the row/column to a row/column not containing any 1s with the maximal entry of $\boldsymbol{F}$ in the same column/row.

Step 4: Repeat step 1. If the matrix is a permutation matrix, terminate the algorithm.

Step 5: Repeat step 2.

Step 6: Evaluate the validity of moving every misplaced 1 to every open column and also evaluate the total assignment benefit of such a move. Apply the optimal correction.

Of course step 3 reminds one of evaluating all possible assignments and in a sense it is the case here. The difference however is there will be only a few misplaced 1's compared to the size of

the matrix and the evaluation of all possibilities is computationally insignificant compared to the original problem.

3.3.3 OACE Algorithm viewed as a process of iterative weighted normalisation

If we consider the OACE algorithm in its simplest form it reduces to a process of iterative normalisation. This becomes clear when one considers equation (2.48) and the term that describes the decision rate α. If one carefully investigates the approach taken in section 3.2 where the assignment problem is written in terms of the Computational Ecology model it turns out that the decision rate α does not play an obvious role. Choosing $\alpha = 1$ reduces equation (2.48) to the rather mundane form

$$f_{ij}(k+1) = \rho_{ij}(k).$$

Now step 4 of the OACE algorithm in section 3.1.3 is consequently reduced to a process of merely updating $\boldsymbol{F}$.

The OACE algorithm can now be entirely described as a process of iterative weighted column-wise normalisation followed by a row-wise normalisation. Firstly a weighted normalisation is executed along each column according to equation (3.1) followed by a row-wise normalisation executed according to equation (3.2). The result is a matrix with rows summing to unity and columns summing to something around unity as the row-wise normalisation perturbs the column-wise normalisation preceding it. From this viewpoint it seems that the Computational Ecology structure is not necessary to find the OACE algorithm. This is true but the Computational Ecology structure was the inspiration for finding this result and even though the whole model peels away it still offers a useful viewpoint of the problem and an intuitive explanation of the dynamics of the algorithm.

Considering the OACE algorithm as a process of iterative normalisation allows a different viewpoint for the reason the algorithm converges towards assignments and more particularly the optimal assignment. Section 3.1.5 contains a discussion of the convergence of the process of weighted normalisation along a one dimensional matrix or vector resulting in a vector with the entry corresponding to the maximum original entry containing a 1 and all other entries 0. The larger the entry becomes the more rapidly it grows with this process of weighted normalisation. The row-wise normalisation that follows every step of weighted column-wise normalisation has a reshuffling effect on this process. It also establishes interdependency between the rows and columns. One can think of this process as that of trying to force the rows and columns to a

situation where both sum to unity, i.e. a doubly stochastic matrix but certain entries will start dominating. The matrix would then converge to something close to a permutation matrix.

3.3.4 Convergence of the population matrix

The process of iterative weighted normalisation along one dimension of a matrix followed by normalisation along the other dimension (the OACE algorithm) will always converge to some stationary matrix. In this section a formal proof for the one dimensional case is offered but only argumentatively extended to the general case.

Firstly we will consider the convergence of iterative weighted normalisation of a one dimensional matrix or vector.

Theorem 3.1 Convergence of a vector subject to iterative weighted normalisation

The process of iterative weighted normalisation of a vector or one dimensional matrix will always converge to a stationary vector. Moreover this vector will either be a vector with one entry equal to 1 and all other entries equal to zero or a vector with a number of n *entries equal to* $1/n$ *and all other entries zero. The unity entry corresponds to the maximum entry in the original matrix and in the case of the fractional entries there were* n *equal and maximal entries in the original matrix.*

(When the iterative weighted normalization is applied to a one dimensional matrix the procedure resembles the *softmax* algorithm as described in [26,47].)

Proof

Consider a one dimensional matrix with all entries positive containing a single maximal entry.

$$\boldsymbol{A} = [a_1 \quad \cdots \quad a_i \quad \ldots \quad a_N]$$

Initialising the population matrix as described in section 3.1.3

$$\boldsymbol{F} = [f_1 \quad \cdots \quad f_i \quad \ldots \quad f_N]$$

as

$$\boldsymbol{F} = [1 \quad \cdots \quad 1 \quad \ldots \quad 1]$$

Considering term one and applying the process of iterative weighted normalisation equation (3.1) once we get

$$f_1(2) = \frac{f_1 \times a_1}{\sum_i f_i \times a_i} = \frac{1 \times a_1}{\sum_i 1 \times a_i} = \frac{a_1}{\sum_i a_i}.$$

Iteration two gives

$$f_1(3) = \frac{f_1(2) \times a_1}{\sum_i f_i(2) \times a_i} = \frac{\frac{a_1}{\sum_i a_i} \times a_1}{\sum_i \frac{a_i}{\sum_i a_i} \times a_i} = \frac{\frac{{a_1}^2}{\sum_i a_i}}{\sum_i \frac{{a_i}^2}{\sum_i a_i}}.$$

Here $f_1(1)$ refers to the first term and first iteration of the population matrix.

Following this process it becomes evident that term k of the population matrix will take the form

$$f_1(k) = \frac{{a_1}^k}{C}$$

where C is an adjusted normalisation factor.

As term one was arbitrarily chosen this holds for any term in vector $\boldsymbol{A}$:

$$f_j(k) = \frac{{a_j}^k}{C} \qquad (3.3)$$

Considering equation (3.3) one can see that each term of the population matrix scales directly proportional to the power of the original corresponding entry in vector $\boldsymbol{A}$. Remembering that all the entries are fractions they will all fall away with higher iterations or powers. The normalising factor ensures summation to unity of all the entries though which would reinstate these entries. Clearly the smaller terms of $\boldsymbol{A}$ must diminish quicker than the larger as the normalising factor is identical for all terms. This implies that the largest term will grow with every iteration (after normalisation) and as there is no reason for slowing this growth all the smaller terms will diminish. The population matrix will tend to a matrix with all entries zero and one entry unity. In the case of having multiple equal and maximal elements of vector $\boldsymbol{A}$ all these elements would grow at the same rate as they would expand according to the equivalent expressions. All other entries would still diminish in accordance with the argument in the previous paragraph and as the normalising procedure ensures summation to unity the population matrix will take the form of all entries zero except the entries corresponding to the maximal elements of $\boldsymbol{A}$ with the values $1/n$.

General case

Generalising the above proof implies that we need to consider a matrix with multiple rows. In this case the procedure of the column-wise normalisation also plays a role. If multiple entries in the same column tend towards unity due to the row-wise weighted normalisation process the column normalisation forces the sum of all these entries to unity and they are fractionalised. A new matrix is now obtained that could have different maximum row entries than the initial matrix and consequently the row-wise weighted normalisation would result in this new maximum entry tending to unity. As this process forces maximal entries to be moved to different columns at some stage one of two things must happen. A situation will be obtained where the two normalisation processes result in a matrix with a maximal and growing entry in each of the rows but in separate columns. This matrix must converge to something very close to a permutation matrix. The second possibility is a situation where each row does not contain a maximal and/or growing entry in separate columns. This can only occur if the smaller entries have stopped growing which means the two normalisation processes have no effect on the matrix. This implies the matrix has already converged.

3.3.5 Results

It turns out that this iterative weighted normalisation process does not always converge to the optimal assignment. As a matter of fact a significant number of situations exist where the application of the greedy algorithm to the converged population matrix does not render the optimal permutation matrix. There even exist cases when the greedy algorithm does not render a permutation matrix at all and an additional step is needed to obtain a permutation matrix as explained in section 3.1.3.2. When a permutation matrix is obtained it is either optimal or very close to optimal. Simulation results show that the success of the algorithm in finding the optimal assignment decreases as larger matrices are considered. This is expected as the mapping from the benefit matrix to the assignment total grows denser as the matrices grow larger. From fig 3.2 can be seen that even though the exact optimal assignment is rarely found for matrices larger than 80×80 the solutions are very close to optimal (figure 3.3) and the error diminishes as larger matrices are considered (figure 3.4 & 3.5).

3.3.6 Simulation results

Simulations of the OACE algorithm were run for randomly initialised matrices with uniformly distributed random numbers from the set [0,1]. This domain is sufficient to cover all possible benefit matrices as the algorithm has a normalising effect and would reduce any initial positive matrix to a population matrix containing elements from the set [0,1]. A thousand runs of the algorithm were done on different random matrices of every order. Matrices of order 2-100 were tested and it turns out that for 100×100 matrices the algorithm was far less effective at finding the optimal assignment compared to smaller matrices as can be seen in figure 3.2 where the number of optimal solutions were counted and represented as a percentage. To illustrate that the algorithm still found near optimal results the proximity to the optimal solution was evaluated and is presented as a percentage in figure 3.2. The number of non-permutation matrix solutions is counted and after applying a corrective procedure to transform such matrices into permutation matrices the proximity to the optimal assignment is evaluated. Proximity here is defined as the difference between the sum of the total benefits corresponding to the optimal assignment and the sum of the total benefits corresponding to the OACE algorithm's result.

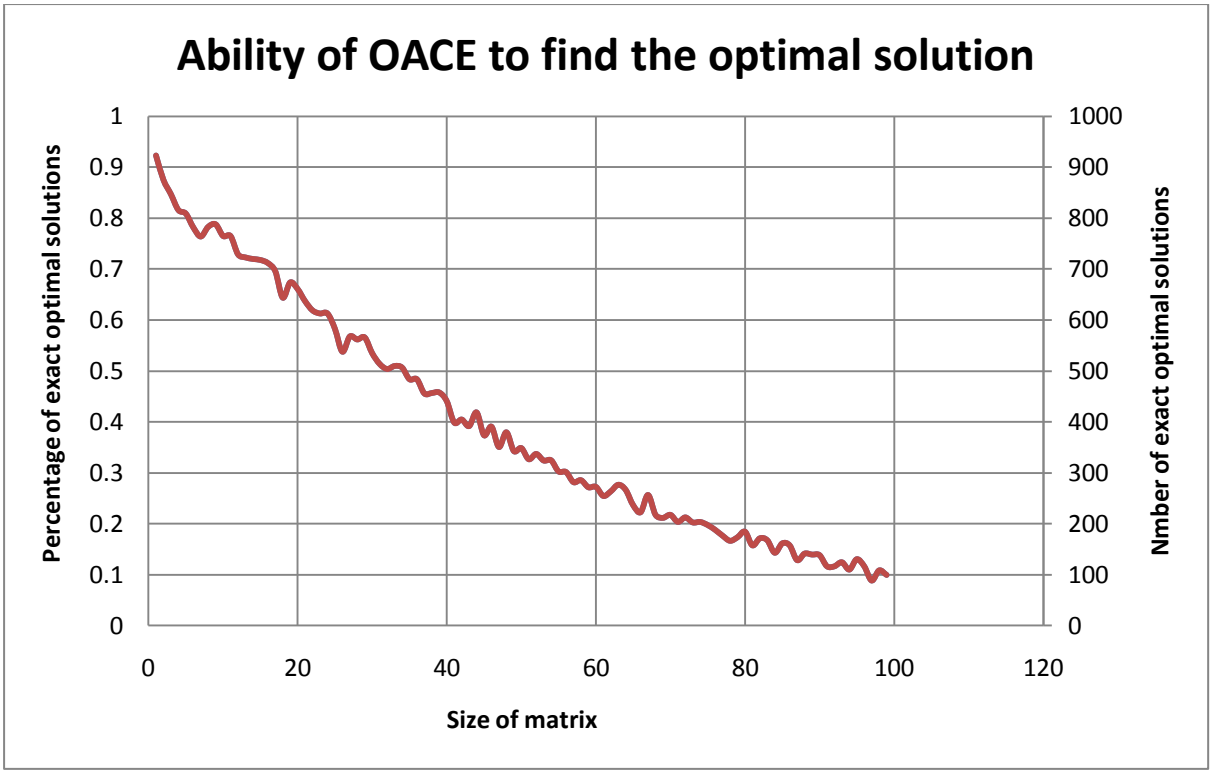


FIGURE 3.2: Graph that plots the number of exact optimal solutions found to the Optimal Assignment problem from a random trial of 1000 for each size matrix runs against the size of the matrix.

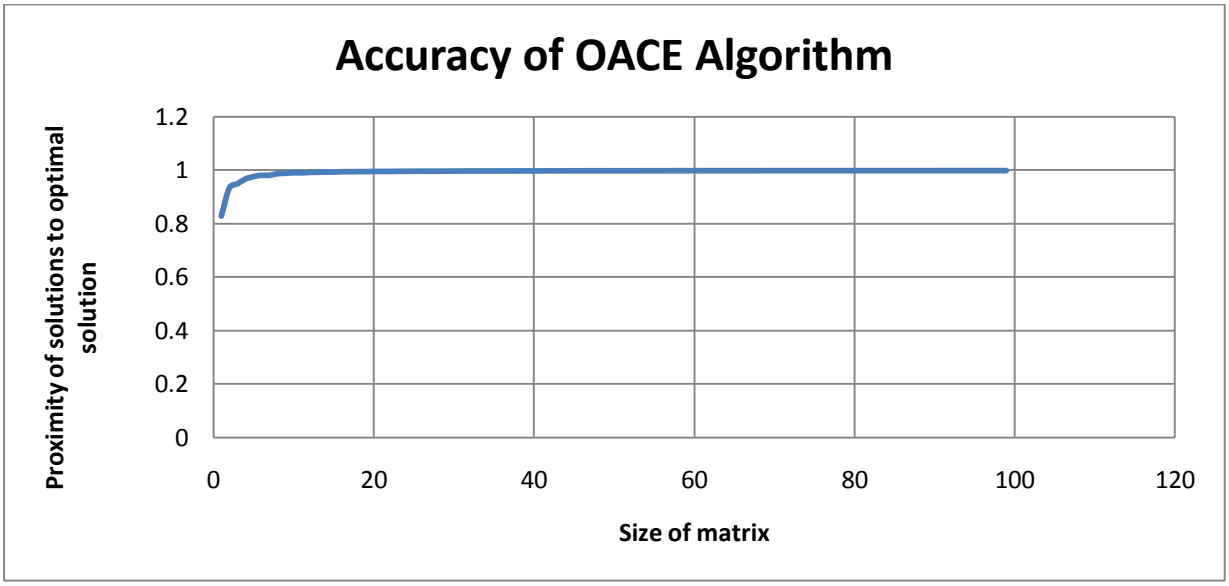


FIGURE 3.3: Graph that plots the proximity of the solutions as found by the OACE algorithm to the actual optimal solutions as a function of the size of the matrices.

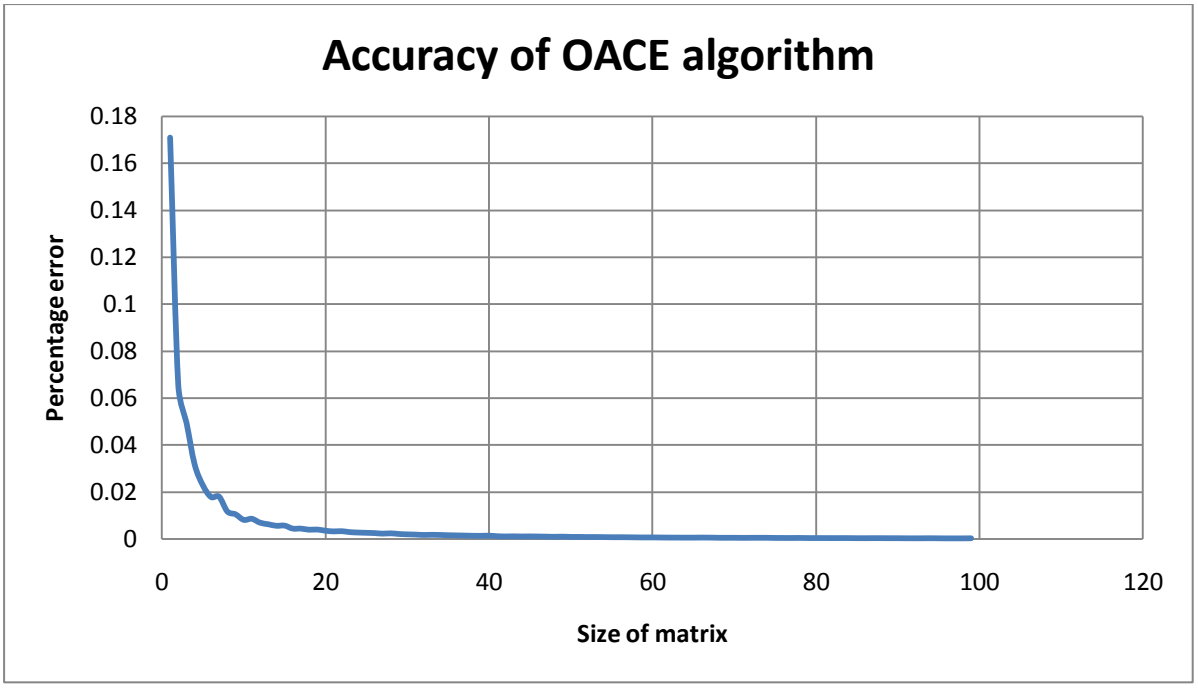


FIGURE 3.4: Graph that shows the percentage error between the solutions found by the OACE algorithm and the actual optimal solutions.

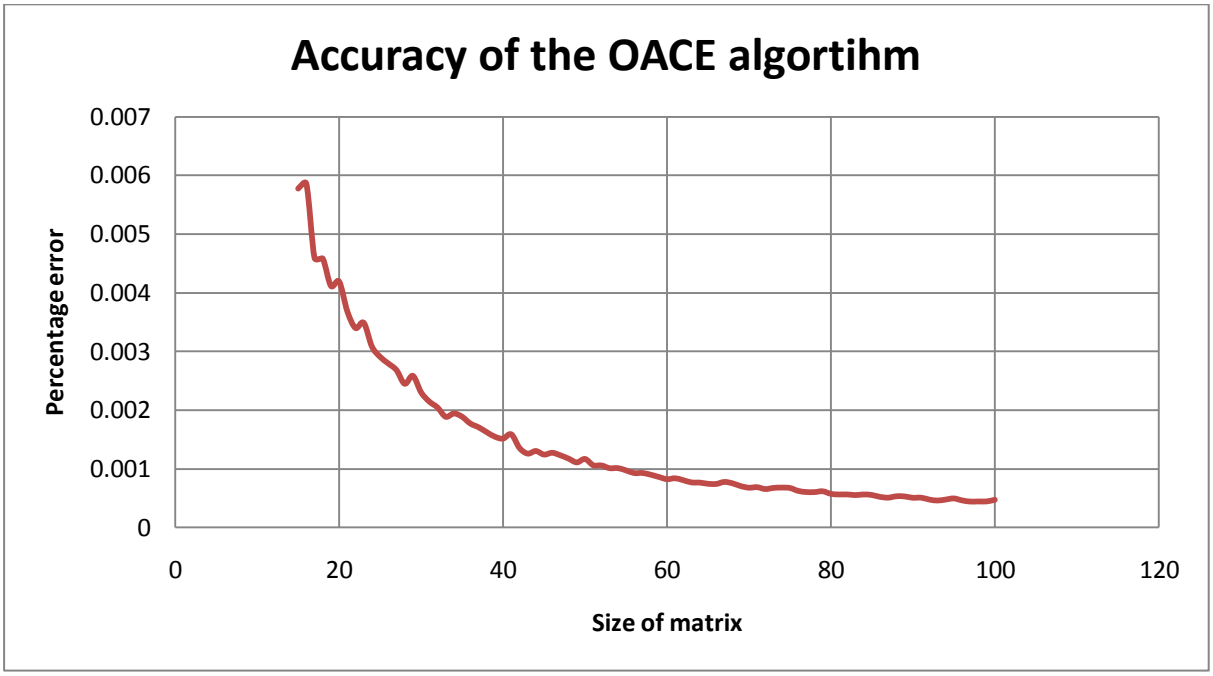


FIGURE 3.5: Enlarged view of the percentage error as depicted in Figure 3.4, shown for larger matrices only.

Discussion of results

As seen in Figure 3.4 & 3.5 the error between the sum of benefits corresponding to the optimal assignment and the sum of benefits corresponding to the OACE algorithm's result diminishes for larger matrices. This is simply the result of the range of the mapping from the benefit matrix to the real line growing denser as the size of the matrices increase. More simply put there will be far more different possible assignments for larger matrices and consequently there would be far more assignments with a total benefit close to that of the optimal assignment. For a method such as the OACE algorithm that is not specifically designed to find the exact optimal solution it is natural that the OACE algorithm would start finding more and more results very close to the optimal solution and less actual optimal solutions as the matrices grow.

Another interesting feature of the OACE algorithm is that it prefers average solutions to extreme solutions. Stated differently, the OACE algorithm would prefer a solution with entries from the benefit matrix of similar size to a solution were some entries are very large and other very small. This makes sense if one remembers that the algorithm is a process of iterative normalisation and there is great interdependency between entries in the same row or column. A very small entry would cause its fellow entries to grow as much as a very large entry would cause its fellow entries to diminish.

3.3.7 Computational efficiency of the algorithm

An empirical approximation of the computational complexity of the algorithm is made by plotting the average number of iterations until termination of the algorithm as a function of the matrix size for the simulation results as can be seen in figure 3.6. After fitting a polynomial function to the data a polynomial with a small but significant second order term proved the best fit. This implies the algorithm terminates in $\mathrm{O}(N^2)$ iterations.

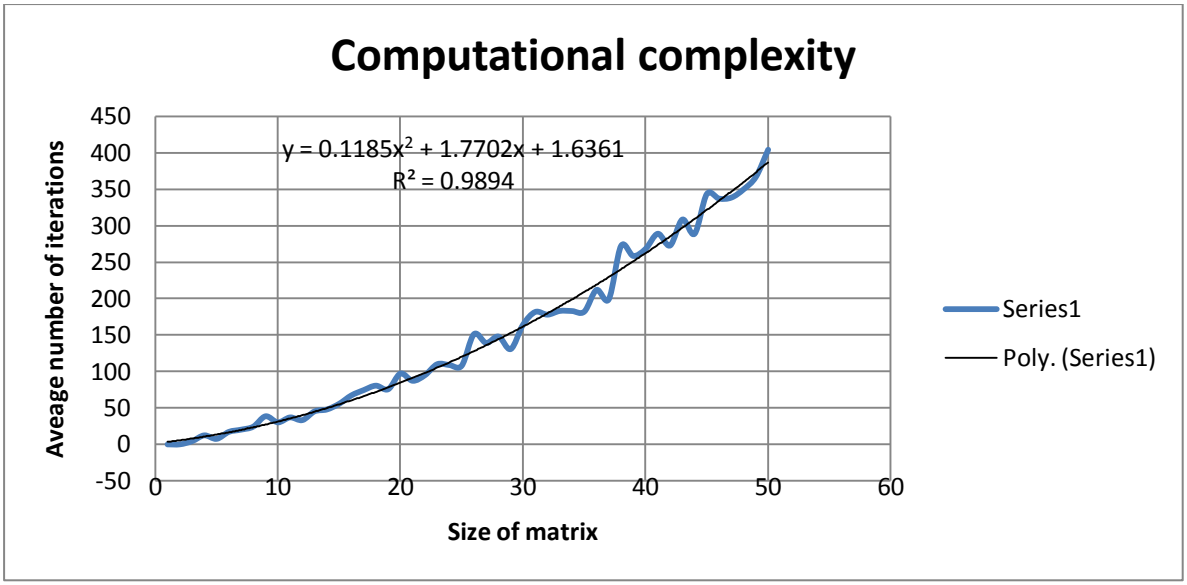


FIGURE 3.6: Average number of iterations taken to find a solution using the OACE algorithm for different size matrices.

Using this empirical approximation of the complexity of the algorithm in terms of number of iterations we can also calculate a computational complexity in terms of number of operations. The OACE algorithm runs in two steps

-Weighted normalisation that takes $2N^2$ operations per iteration

-Normalisation that takes N^2 operations per iteration

for a total of $3N^2$ operations per iteration. As the empirically approximated computational complexity of the algorithm is $O(N^2)$ iterations this implies a computational complexity of $O(N^4)$ operations. This is not a worst case performance bound but a general indication of practical algorithmic performance.

3.3.7.1 Preconditioning of the Benefit matrix

The straightforward application of the OACE algorithm certainly allows for improvement but there are no obvious alterations to the algorithm itself without incurring a large increase in complexity. One other element that can be easily altered or adjusted is the benefit matrix. We refer to this as preconditioning of the benefit matrix and investigate several methods of adjusting the Benefit matrix and how it influences the performance of the algorithm. The following precondition methods are investigated.

- Projection of the benefit matrix into the doubly stochastic polytope but conserving the assignment
- Repeated projection into the doubly stochastic polytope with assignment conservation
- Adding a bias to each entry of the Benefit matrix

Projection of the benefit matrix into the doubly stochastic polytope with conservation of the optimal assignment.

The method of projection into the doubly stochastic polytope and preserving the assignment as described in section 3.2 allows for some interesting applications. The idea here is simply that the new $\boldsymbol{B}$ matrix is closer to a permutation matrix than the original. As can be seen in Figure 3.10 the resulting effect on the OACE algorithm is quite impressive for matrices up to size 5×5 with a substantial improvement over the standard OACE algorithm but for larger matrices there is a rapid degeneration in performance resulting in a process that hardly finds the optimal solution for matrices above size 30×30.

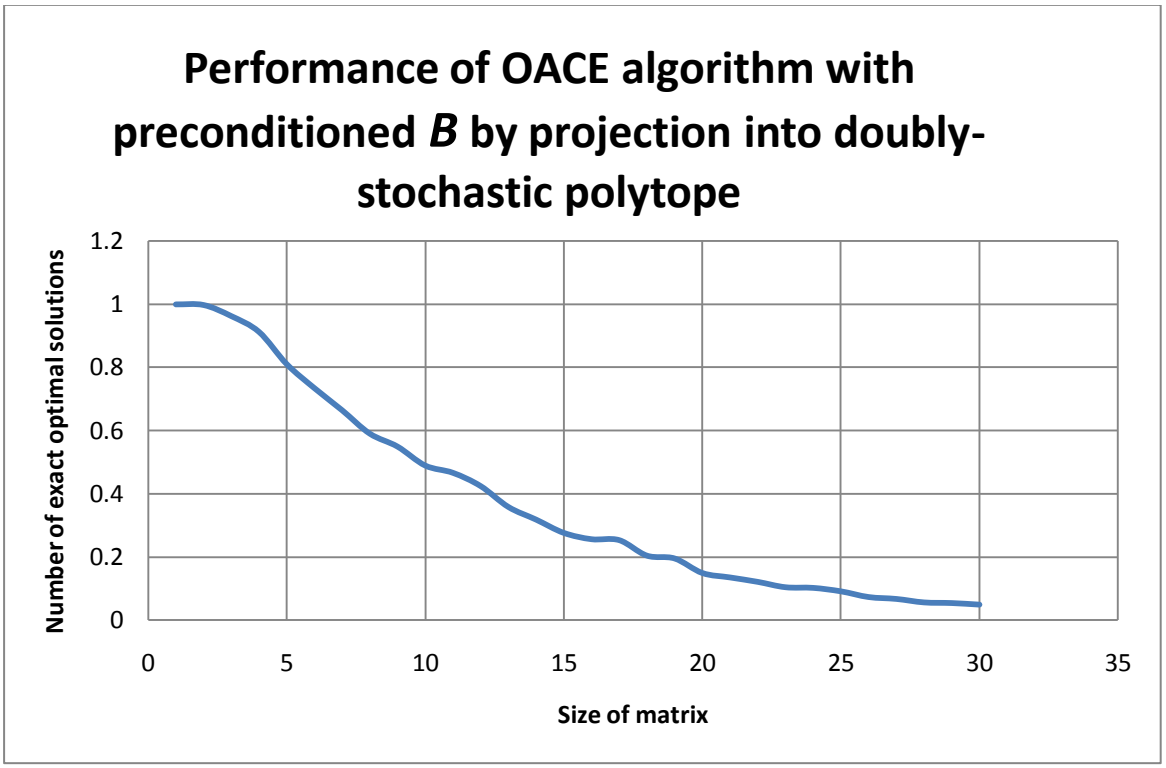


FIGURE 3.7: Performance of the OACE algorithm with preconditioning applied to $\boldsymbol{B}$ in the form of an assignment preserving projection into the doubly- stochastic polytope as described in Section 3.2.

Repeated projection into the doubly stochastic polytope with conservation of the optimal assignment.

The idea to repeatedly project the $\boldsymbol{F}$ matrix into the doubly stochastic polytope originates from similar methods such as the Invisible hand method by Kosowsky with application of Sinkhorn's algorithm where matrix $\boldsymbol{S}$ is continually projected into the doubly stochastic polytope using Sinkhorn's iterative algorithm. This method performs very poorly and fails to find optimal solutions for Benefit matrices larger than 30×30 are considered.

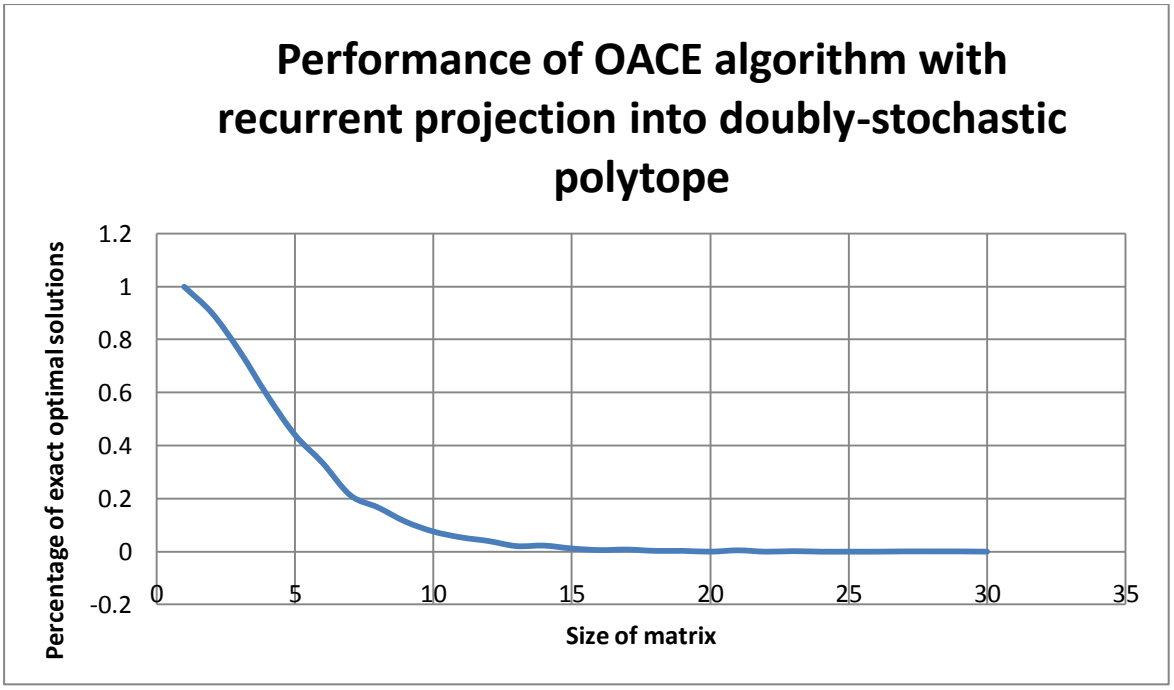


FIGURE 3.8: Performance of the OACE algorithm under repeated projection of $\boldsymbol{B}$ into the doubly stochastic polytope using the Assignment preserving projection as described in Section 3.2.

Adding a bias to each entry of the Benefit matrix

The most promising preconditioning method is also the simplest. Adding a constant to each entry in the Benefit matrix led to a rather substantial improvement for smaller matrices with this preconditioning method outperforming the standard OACE algorithm for matrices up until about size 20×20 as figure 3.10 depicts. For larger matrices the standard algorithm performs slightly better with an increasing improvement in performance as the matrices grow as can be seen in figure 3.11. Remember that the way the Benefit matrix was initialised was with numbers from the set [0,1] and any Benefit matrix can be rewritten in this format after suitable normalisation. Considering this a bias that would neither be insignificantly small or dominatingly large needs to be chosen and it was found that something between 0.5 and 1 proved effective as it is of similar size to the entries. The bias added to the results depicted in figure 3.10 and 3.11 is 1. Adding a bias to the entries of $\boldsymbol{B}$ has a positive effect on the convergence of the algorithm in that it removes the influence that very small numbers have on the process of normalization. Numbers close to zero make no contribution to the Optimal Assignment but will blow up other numbers in the same row or column during the normalization procedure. This agrees with the argument in the discussion of section 3.1.7 that the OACE algorithm prefers average solutions to extreme solutions as adding the bias has an averaging effect on the entries of the matrix.

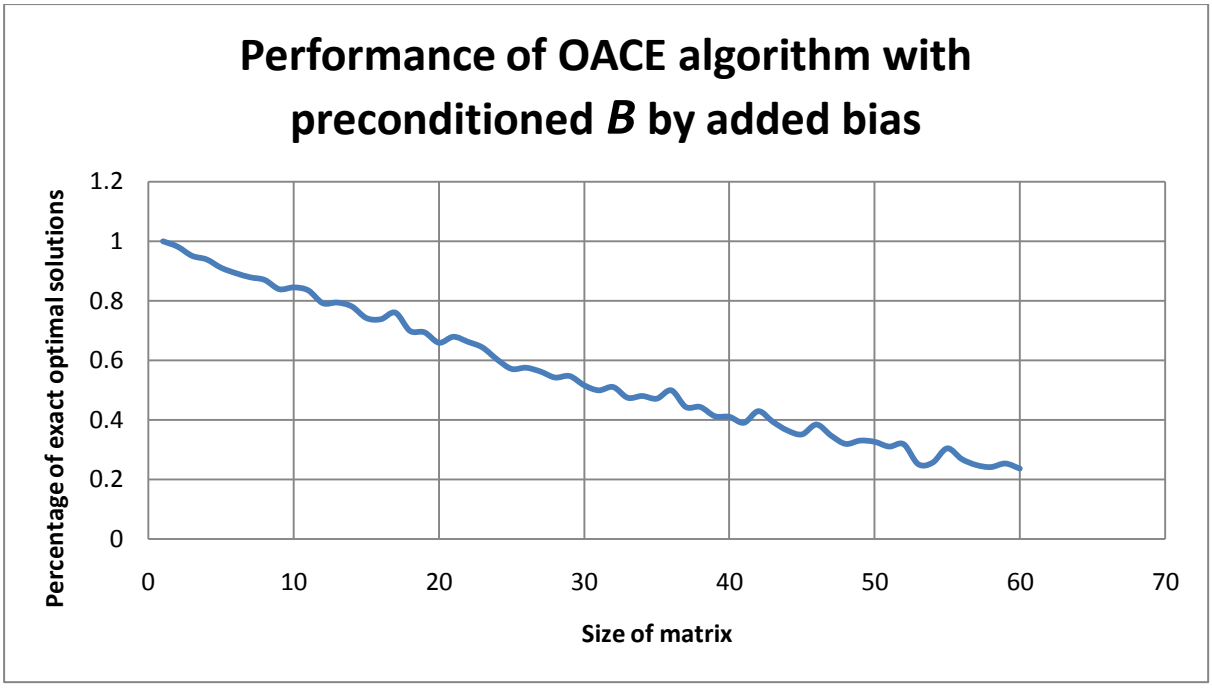


FIGURE 3.9: Performance of the OACE algorithm with preconditioning applied to $\boldsymbol{B}$ in the form of a constant bias added to all entries of matrix $\boldsymbol{B}$.

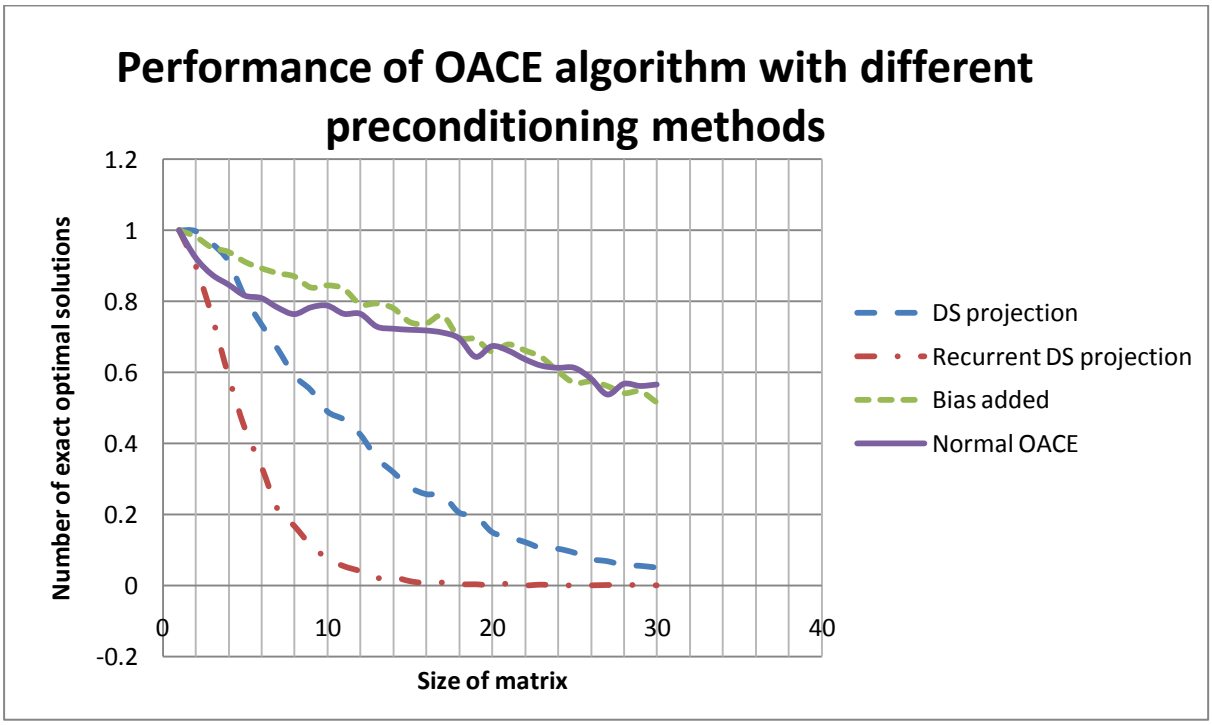


FIGURE 3.10: Comparison of performance of OACE algorithm with various types of preconditioning applied to $\boldsymbol{B}$.

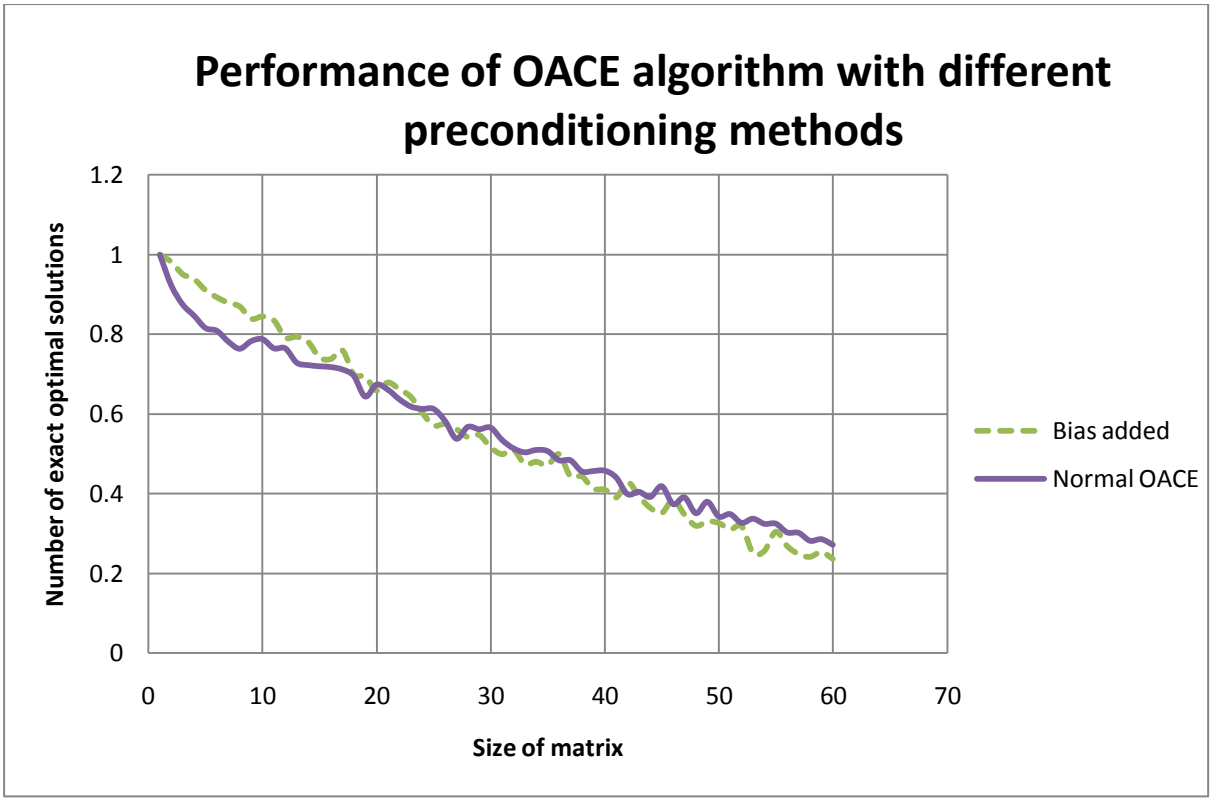


FIGURE 3.11: Comparison of performance of OACE algorithm for the standard algorithm and the algorithm with added bias.

3.3.8 Examples and applications of the OACE algorithm

3.3.8.1 Example of OACE algorithm

To allow some insight into the operation of the algorithm a 3×3 matrix is randomly initialised with numbers from the set $[0,1]$ rounded to the third decimal and the optimal assignment found using the OACE algorithm. Figure 3.12 shows the evolution of all the entries of the population matrix corresponding to the benefit matrix below. The algorithm is continued until the entries in matrix $\boldsymbol{F}$ become stationary but could actually have been terminated once the matrix became row-dominant. This occurred after iteration 2. Figure 3.12 shows that three of the entries tend towards unity or something close by while all the other tends towards zero or something close by. The randomly initialised benefit matrix is

$$\boldsymbol{B} = \begin{bmatrix} 0.419 & 0.753 & 0.793 \\ 0.919 & 0.884 & 0.367 \\ 0.620 & 0.731 & 0.193 \end{bmatrix}.$$

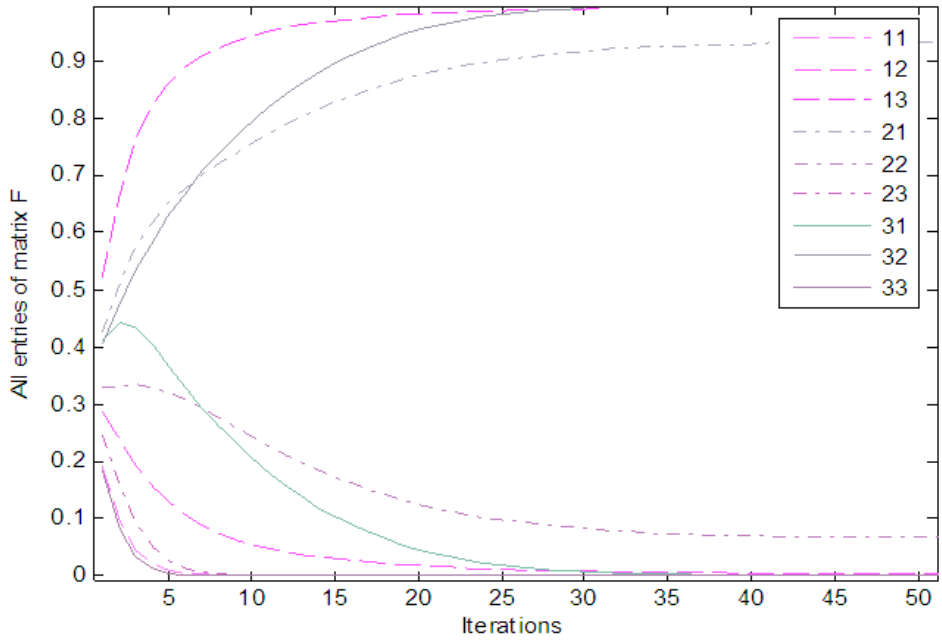


FIGURE 3.12: Plot of the evolution of all entries of the population matrix as a function of iteration number for the OACE algorithm.

To find a permutation matrix approximation one takes the three maximum entries in figure 3.12 and allocate a one to each one of them and a zero to all other entries. This results in

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

Representing the solution to the optimal assignment problem for the above $\boldsymbol{B}$.

3.3.8.2 Solving Example 2.2.1.1 the shortest path problem with the OACE algorithm

Example 2.2.1.1 illustrates that the shortest path problem can be written in a matrix format where the shortest path is found by finding the minimal assignment of this matrix. The matrix arrived at in example 2.2.1.1 is

$$\boldsymbol{D} = \begin{bmatrix} 1 & 3 & 4 & 3 & \infty \\ 0 & 3 & \infty & 1 & 4 \\ 3 & 0 & \infty & 6 & 5 \\ \infty & \infty & 0 & 1 & \infty \\ 1 & 6 & 1 & 0 & 2 \end{bmatrix}.$$

In this form the matrix is not suitable for application of the OACE algorithm but a few small changes can fix this. Firstly as the shortest path is concerned with finding a minimal assignment the matrix needs to be rewritten. This is done by negating the matrix and adding the absolute value of its smallest entry to ensure all entries are positive. For the above matrix this would be a

problem as some of the entries are infinity but we can easily circumnavigate this problem by replacing the infinity sign by a large number. A number larger than the sum of entries corresponding to the optimal assignment would ensure that such a number can not influence the optimal assignment of the new matrix. A simple way to find a number that would satisfy this constraint is to take the maximum entry of $\boldsymbol{D}$ smaller than infinity and multiply it by the size of the square matrix. That is

$$N \times \max_{ij} d_{ij}.$$

Finding the optimal assignment of such a matrix now corresponds to the minimal assignment on the initial matrix.

$$(\max_{ij} d_{ij} < \infty) = 6$$

Then replace ∞ with a number greater than $N \times 6 = 5 \times 6 = 30$ say 31

$$\begin{bmatrix} 1 & 3 & 4 & 3 & 31 \\ 0 & 3 & 31 & 1 & 4 \\ 3 & 0 & 31 & 6 & 5 \\ 31 & 31 & 0 & 1 & 31 \\ 1 & 6 & 1 & 0 & 2 \end{bmatrix}.$$

Negating the matrix yields

$$\begin{bmatrix} -1 & -3 & -4 & -3 & -31 \\ 0 & -3 & -31 & -1 & -4 \\ -3 & 0 & -31 & -6 & -5 \\ -31 & -31 & 0 & -1 & -31 \\ -1 & -6 & -1 & 0 & -2 \end{bmatrix}.$$

After adding an offset and we have a benefit matrix in the traditional sense again

$$\boldsymbol{B} = \begin{bmatrix} 30 & 28 & 27 & 28 & 0 \\ 31 & 28 & 0 & 30 & 27 \\ 28 & 31 & 0 & 25 & 26 \\ 0 & 0 & 31 & 30 & 0 \\ 30 & 25 & 30 & 31 & 29 \end{bmatrix}.$$

Applying the OACE algorithm to this matrix results in the following graph.

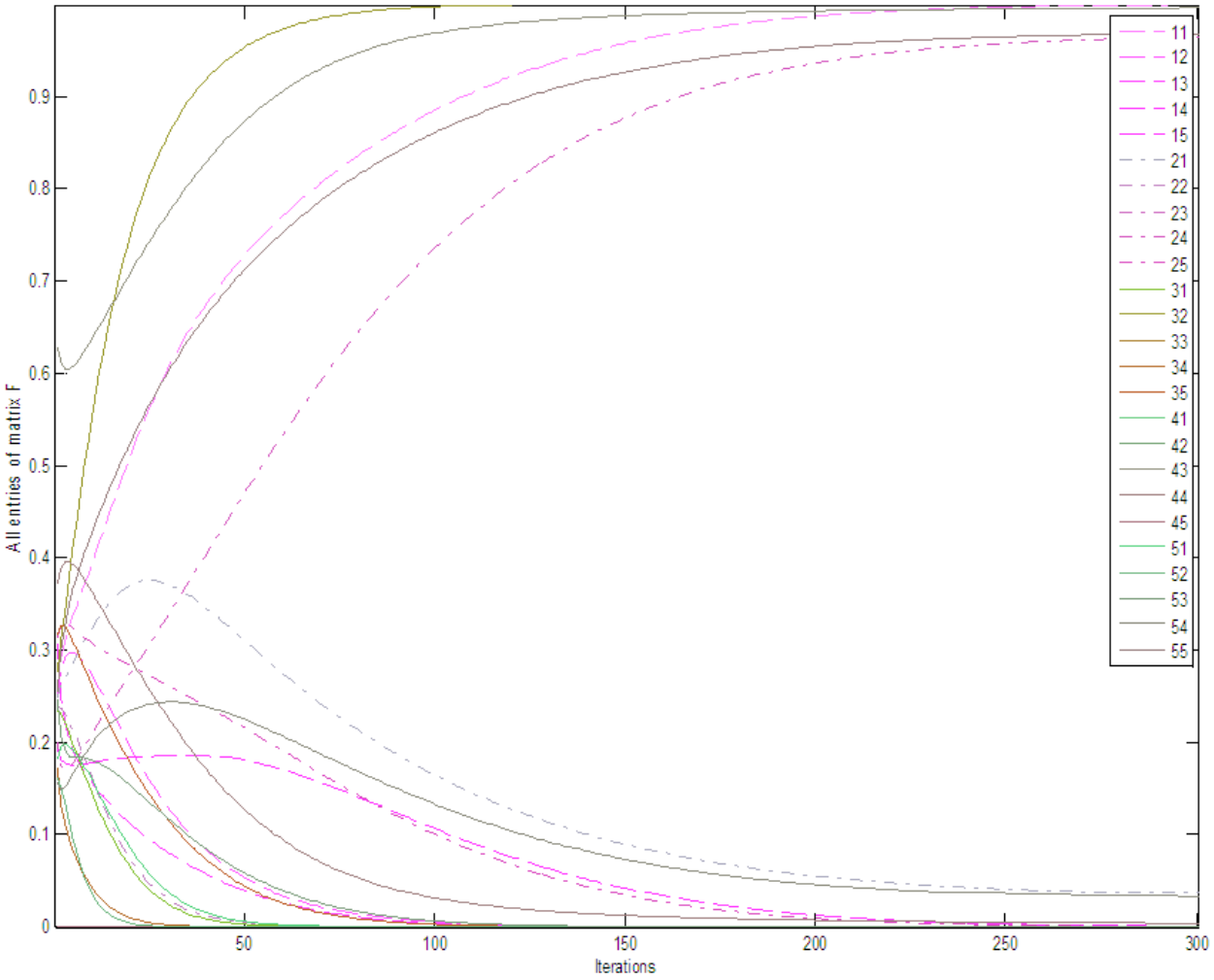


FIGURE 3.13: Plot of the development of all entries of the population matrix corresponding to the shortest path example as a function of the iteration number. Plot continues until convergence of matrix ***F***.

Figure 3.13 illustrates the development of all the entries of the population matrix ***F*** as the OACE algorithm is iterated. Upon inspection of figure 3.13 it is evident that five of the entries of ***F*** tend towards larger numbers near unity while all other entries tend towards smaller numbers near zero. These five maximum entries are indicative of the optimal assignment associated with matrix ***B***. Matrix ***F*** converges after about 300 iterations and the converged matrix looks as follows

$$\boldsymbol{F}(k=300)=\begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0335 & 0 & 0 & 0.9664 & 0 \\ 0 & 1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0.9976 & 0.0024 & 0 \\ 0 & 0 & 0 & 0.0312 & 0.9688 \end{bmatrix}$$

and is approximated by the following permutation matrix

$$\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} \end{bmatrix},$$

representing the solution to the assignment problem.

The algorithm can be terminated once $\boldsymbol{F}$ becomes row-dominant rather than waiting for $\boldsymbol{F}$ to converge. This allows for far quicker termination of the algorithm after only 35 iterations. In figure 3.14 the development of $\boldsymbol{F}$ until it becomes row dominant is illustrated.

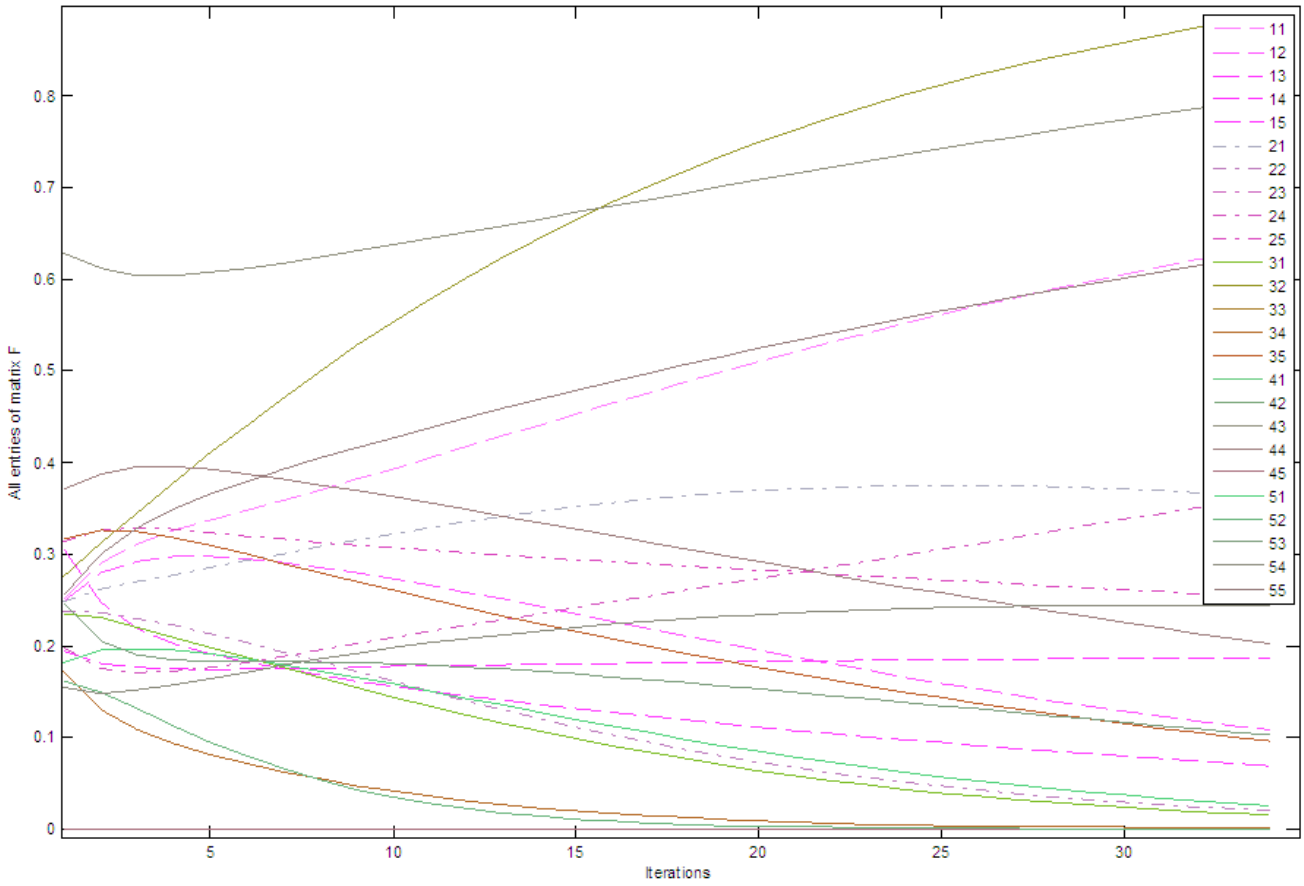


FIGURE 3.14: Plot of the development of all entries of the population matrix corresponding to the shortest path example as a function of iteration number. Plot continues until matrix $\boldsymbol{F}$ becomes row dominant.

The row dominant $\boldsymbol{F}$ matrix obtained after 35 iterations

$$\boldsymbol{F}(k=35)=\begin{bmatrix}0.6356 & 0.1083 & 0.0699 & 0.1862 & 0.0000\\ 0.3630 & 0.0203 & 0 & 0.3642 & 0.2525\\ 0.0157 & 0.8872 & 0 & 0.0010 & 0.0961\\ 0 & 0 & 0.7976 & 0.2024 & 0\\ 0.0261 & 0.0001 & 0.1031 & 0.2431 & 0.6277\end{bmatrix}$$

which is approximated by the following permutation matrix

$$\begin{bmatrix}\mathbf{1} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & \mathbf{1} & 0\\ 0 & \mathbf{1} & 0 & 0 & 0\\ 0 & 0 & \mathbf{1} & 0 & 0\\ 0 & 0 & 0 & 0 & \mathbf{1}\end{bmatrix},$$

representing the solution to the assignment problem.

3.1 COMPARISON OF THE OACE ALGORITHM TO OTHER ALGORITHMS

It is useful to see how the OACE algorithm matches up to some of its competitors. To do this we use the empirical computational complexity approximation as derived in Section 3.1.7.1 and compare it to complexity approximations of the other algorithms. It is notable that all the complexity measures given below are approximations of the practical complexity of the algorithms rather than complexity bounds. The distribution of the sets of coefficients constituting assignments are assumed to be nearly uniform and other more specific scenarios could influence these approximations significantly. These approximations do however offer an indication of the order of complexity and how the OACE algorithm compares with some of the best known and best solutions to the optimal assignment problem.

3.1.1 Brute force approach

The exact number of calculations needed to determine the optimal answer in using the brute force approach is easy to establish analytically. This number of operations will always be required and is a very fast growing number.

In calculating the number of operations needed for the brute force approach, consider firstly the number of different possibilities. There are $N!$ Different combinations that need to be evaluated and each requires N summations. All these assignment totals need to be compared, i.e. $N!$ Comparisons. This gives a total of

$$N! \times N + N!$$

operations.

This is such a fast growing number that any graphical comparisons to the other methods considered are meaningless for anything but very small numbers.

3.1.2 OACE algorithm

From Section 3.1.7.1 an empirical derivation of the computational complexity of the OACE algorithm is found as

$$0.12N^2 + 1.77N + 1.64$$

iterations with $3N^2$ operations per iterationgiving an operational complexity of

$$0.36N^4 + 5.31N^3 + 1.64.$$

3.1.3 Auction algorithm

Schwartz offers an estimation of the auction algorithm's computational complexity in [48]. Upon making assumptions regarding the distribution of the sets of coefficients constituting an assignment problem and some theoretical approximations a computational complexity of

$$1.757N^2 + \log_{10} N$$

is derived.

3.1.4 Hungarian algorithm

The Hungarian algorithm was originally found to be $O(N^4)$operations by Kuhn. Later improvements found running times of $O(N^3)$ and various similar performance estimations as seen in [49].

3.1.5 *Graphical comparisons of the computational complexity of the different algorithms*

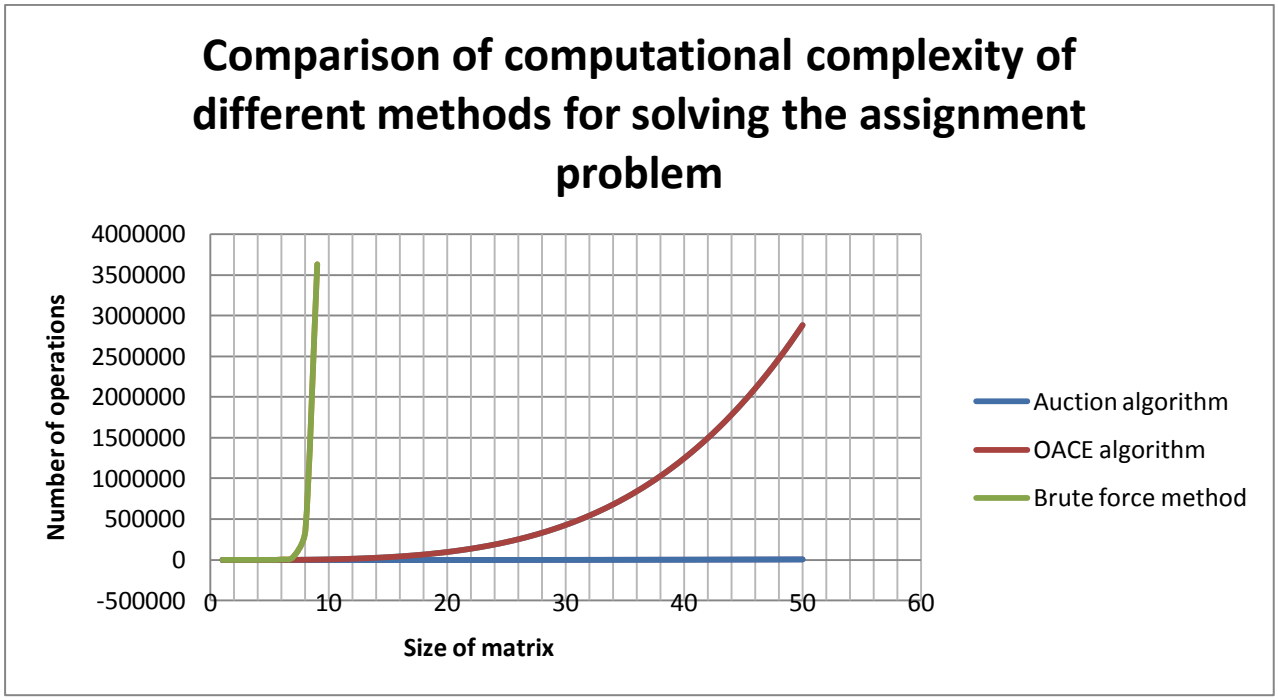


FIGURE 3.15: Plot that compares the computational complexity of different algorithms. The faster growing plots are only plotted for smaller matrices to allow for a meaningful graphical comparison.

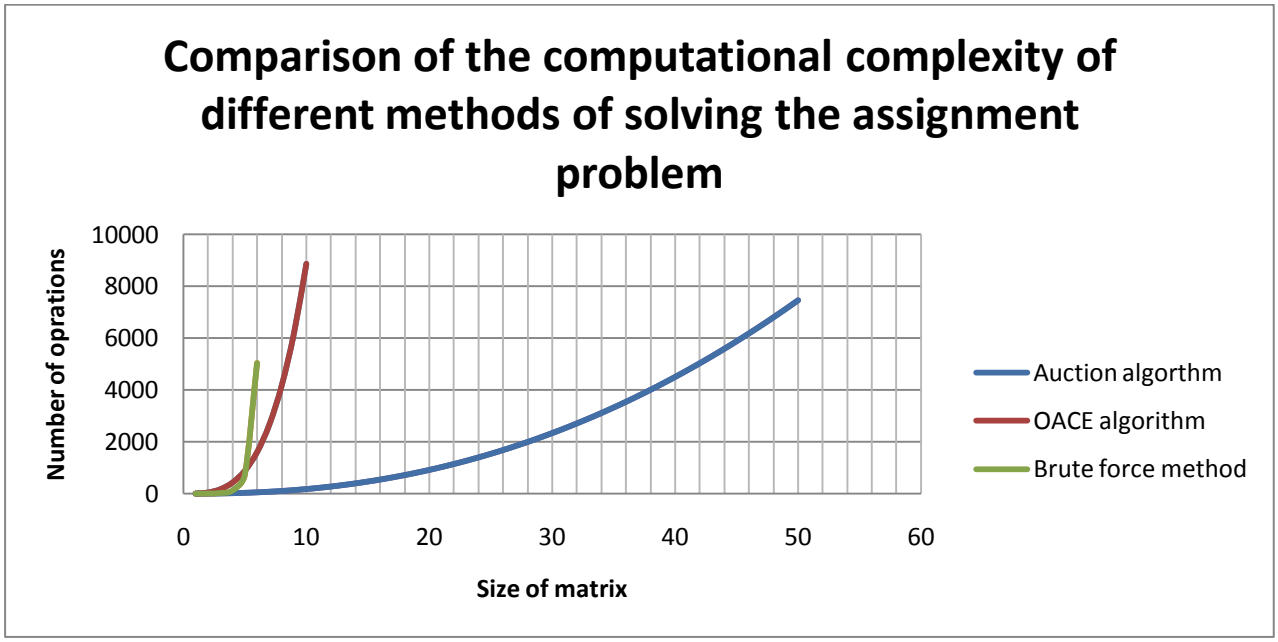


FIGURE 3.16: A second plot that compares the different algorithms with greater magnification of the y-axis.

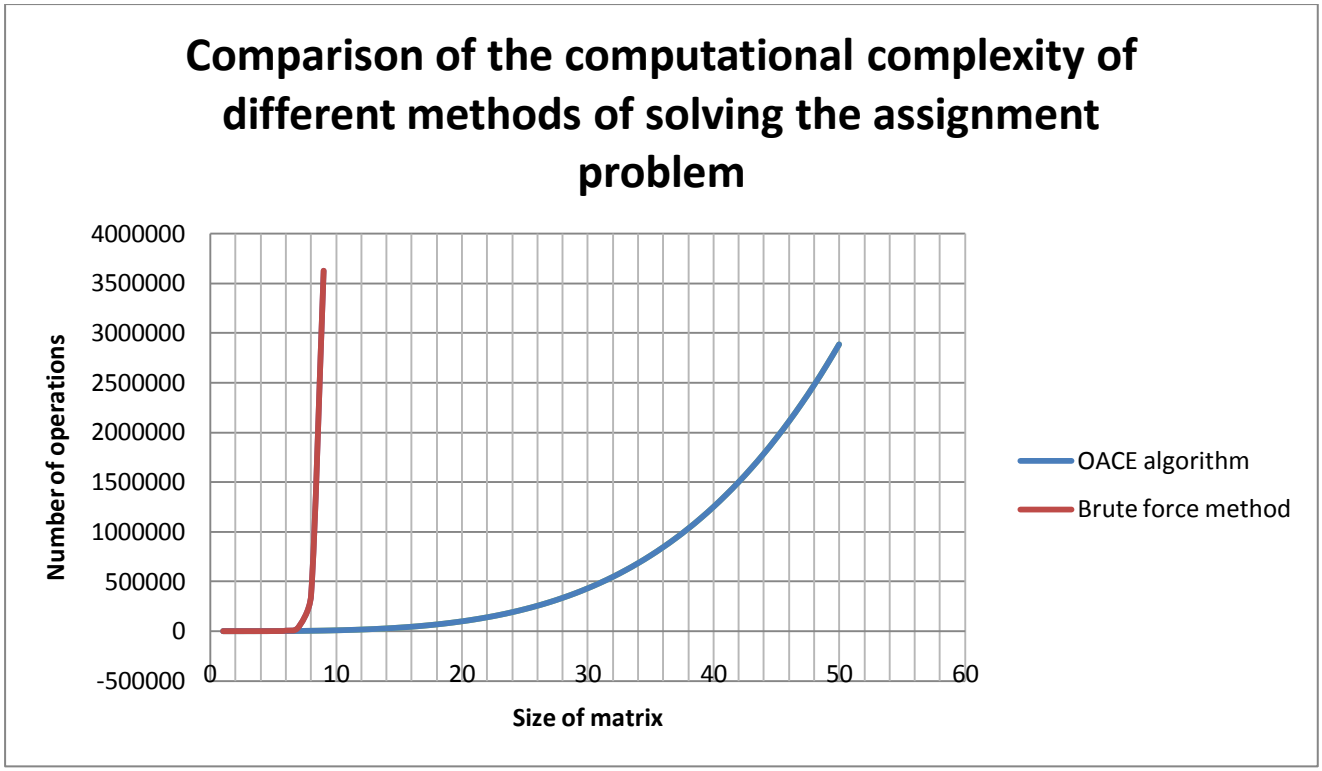


FIGURE 3.17: Indication of the operational complexity of the OACE algorithm compared to the brute force approach.

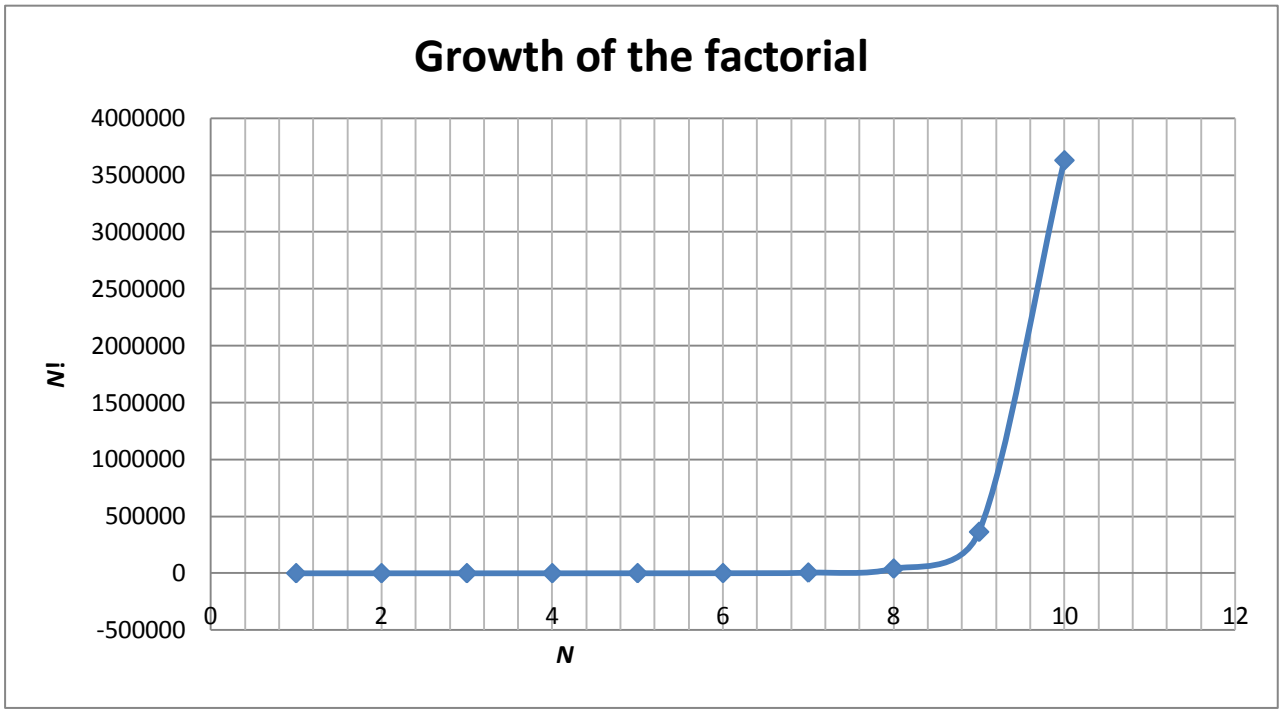


FIGURE 3.18: Graph that attempts to graphically indicate how fast the factorial function grows. This graph keeps this shape when truncated at almost any value on a linear graph.

3.1.6 Summary of comparisons

TABLE 3.1: Comparison of the computational complexity for different optimal assignment algorithms

Algorithm	**Empirical complexity**	**Complexity order**
Brute Force	$N! \times N + N!$	$\mathrm{O}(N!)$
OACE Algorithm	$0.35N^4 + 5.31N^3 + 1.63$	$\mathrm{O}(N^4)$
Hungarian Algorithm	$-$	$\mathrm{O}(N^3)$
Auction Algorithm	$1.757N^2 + \log_{10} N$	$\mathrm{O}(N^2)$

The previous section indicates that the OACE algorithm's performance is inferior to some of its best known and established competitors. It is still however of insignificant complexity when compared to the brute force method which does render it a valid solution to the problem. The auction algorithm clearly outperforms all the other algorithms considered.

CHAPTER 4: DOUBLY STOCHASTIC APPROXIMATION OF A MATRIX WITH THE SAME OPTIMAL ASSIGNMENT

4.1 INTRODUCTION

There are various methods of approximating a given matrix with a doubly stochastic matrix, some methods include finding the doubly stochastic approximation of a matrix that is nearest to it under the Frobenius (least squares) norm [50]. Another is the closest under Frobenius norm and with the same first moment [51] or same first and second moment [52] or with all entries non-negative. Sinkhorn shows an important method that decomposes a matrix into a doubly stochastic matrix front and back multiplied by diagonal matrices after iterative normalisation [23]. In this document we consider finding a doubly stochastic approximation to a matrix with the same optimal assignment. Stated differently, we find a doubly stochastic approximation in the vicinity of the same permutation matrix that is closest to the original matrix under some norm.

Any square matrix has a certain assignment matrix associated with it that represents the optimal assignment. This assignment matrix is also a permutation matrix as mentioned earlier. This implies that solving the assignment problem corresponds to finding the nearest permutation matrix to the benefit matrix under some norm. The Frobenius norm is sufficient and simple. It is also possible to find a doubly stochastic matrix with the same optimal assignment as the original benefit matrix as will be described later in this chapter. This implies that the assignment problem can then be viewed as that of finding the nearest permutation matrix to the new doubly stochastic matrix. It is worthwhile remembering that the set of doubly stochastic matrices forms a convex hull with the extreme points being the permutation matrices [22].

4.2 DERIVATION

The derivation will be done for the case where the matrix is of order 3 with a generalized extension to follow. It is important to remember that if a constant number is added to all the entries in any row or column the optimal assignment of the matrix remains unchanged as

described in section 2.3.3. This derivation uses scalar translation of rows and columns and matrix scaling, operations that do not affect the optimal assignment.

Consider a square matrix $\boldsymbol{A}$,

$$\boldsymbol{A} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}. \quad (4.1)$$

We need to find constants a, b and c such that

$$(a + \alpha_{11}) + (a + \alpha_{12}) + (a + \alpha_{13}) = K_r \quad (4.2)$$

$$(b + \alpha_{21}) + (b + \alpha_{22}) + (b + \alpha_{23}) = K_r \quad (4.3)$$

$$(c + \alpha_{31}) + (c + \alpha_{32}) + (c + \alpha_{33}) = K_r \quad (4.4)$$

with K_r any real number.

This amounts to solving equations (3.5),(3.6) and (3.7) simultaneously which would allow us to find values for a, b and c with K_r a free parameter.

Choose

$$K_r = \max_i \left(\sum_j \alpha_{ij} \right) \quad (4.5)$$

fixing the free parameter K_r and assuming, with no loss of generality, that this is the sum of the entries in the first row. Now substituting

$$K_r = \left(\sum_j \alpha_{i_{max}j} \right) \quad (4.6)$$

into equation (3.5) we see that

$$a = 0.$$

Then it follows from

$$(4.2) = (4.3)$$

$$(a+\alpha_{11})+(a+\alpha_{12})+(a+\alpha_{13})=(b+\alpha_{21})+(b+\alpha_{22})+(b+\alpha_{23}) \quad (4.7)$$

or

$$b=a+\frac{\alpha_{11}+\alpha_{12}+\alpha_{13}}{3}-\frac{\alpha_{21}+\alpha_{22}+\alpha_{23}}{3}$$

$$b=\frac{\alpha_{11}+\alpha_{12}+\alpha_{13}}{3}-\frac{\alpha_{21}+\alpha_{22}+\alpha_{23}}{3} \quad (4.8)$$

and similarly

$$c=a+\frac{\alpha_{11}+\alpha_{12}+\alpha_{13}}{3}-\frac{\alpha_{31}+\alpha_{32}+\alpha_{33}}{3}$$

$$c=\frac{\alpha_{11}+\alpha_{12}+\alpha_{13}}{3}-\frac{\alpha_{31}+\alpha_{32}+\alpha_{33}}{3}. \quad (4.9)$$

After suitable scaling all rows will now sum to unity

$$\boldsymbol{A}_S=\begin{bmatrix}a+\alpha_{11} & a+\alpha_{12} & a+\alpha_{13}\\ b+\alpha_{21} & b+\alpha_{22} & b+\alpha_{23}\\ c+\alpha_{31} & c+\alpha_{32} & c+\alpha_{33}\end{bmatrix}\times\frac{1}{K_r}. \quad (4.10)$$

$\boldsymbol{A}_S$ is now a right stochastic matrix. This same process can be followed to normalize the columns of $\boldsymbol{A}_S$ without changing the optimal assignment by simply applying the exact same process to the matrix $\boldsymbol{A}_S^T$, the transpose of matrix $\boldsymbol{A}_S$ and then transposing again once completed. It is important to note that applying the same procedure to the columns preserves the summation of the rows to unity and this is proven at the end of this chapter.

The general case

To generalize this procedure we represent $\boldsymbol{A}$ as

$$\boldsymbol{A}=\begin{bmatrix}\alpha_{11} & \cdots & \alpha_{1N}\\ \vdots & \ddots & \vdots\\ \alpha_{N1} & \cdots & \alpha_{NN}\end{bmatrix} \quad (4.11)$$

with N equations and N constants $r_1, \ldots, r_N$ such that

$$(r_1+\alpha_{11})+\cdots+(r_1+\alpha_{1N})=K_r$$

$$(r_2+\alpha_{21})+\cdots+(r_2+\alpha_{2N})=K_r$$

$$\vdots$$

$$(r_i+\alpha_{i1})+\cdots+\left(r_i+\alpha_{ij}\right)+\cdots+(r_i+\alpha_{iN})=K_r$$

$$\vdots$$

$$(r_N + \alpha_{N1}) + \cdots + (r_N + \alpha_{NN}) = K_r$$

and

$$r_k = r_{i_max} + \frac{1}{n}\sum_j \alpha_{i_{max}j} - \frac{1}{n}\sum_j \alpha_{kj}. \quad (4.12)$$

As we choose

$$K_r = \max_i \left(\sum_j \alpha_{ij} \right)$$

$$r_{i_max} = 0$$

and consequently

$$r_k = \frac{1}{n}\sum_j \alpha_{i_{max}j} - \frac{1}{n}\sum_j \alpha_{kj} \quad (4.13)$$

with

$$\boldsymbol{A}_S = \begin{bmatrix} r_1 + \alpha_{11} & \cdots & r_1 + \alpha_{1N} \\ \vdots & \ddots & \vdots \\ r_N + \alpha_{N1} & \cdots & r_N + \alpha_{NN} \end{bmatrix} \times \frac{1}{K_r}. \quad (4.14)$$

Which is rewritten for ease of use as

$$\boldsymbol{A}_S = \begin{bmatrix} \beta_{11} & \cdots & \beta_{1N} \\ \vdots & \ddots & \vdots \\ \beta_{N1} & \cdots & \beta_{NN} \end{bmatrix}. \quad (4.15)$$

The same process applied to the columns of $\boldsymbol{A}_S$ allows us to compute the N scalars $c_1, \ldots, c_N$ that add to the columns of $\boldsymbol{A}_S$. The resulting values are

$$K_c = \max_j \left(\sum_i \beta_{ij} \right) \quad (4.16)$$

where K_c is the value assigned to the free parameter corresponding to the column normalisation. The other scalars are

$$c_k = \frac{1}{n}\sum_i \beta_{ij_{max}} - \frac{1}{n}\sum_i \beta_{ik} \quad (4.17)$$

with $k = 1, \dots, N$. With the new doubly stochastic matrix

$$\boldsymbol{A}_{DS} = \left(\begin{bmatrix} c_1 + \beta_{11} & \cdots & c_N + \beta_{1N} \\ \vdots & \ddots & \vdots \\ c_1 + \beta_{N1} & \cdots & c_N + \beta_{NN} \end{bmatrix} \times \frac{1}{K_c} \right). \quad (4.18)$$

Proof that $\boldsymbol{A_{DS}}$ is doubly stochastic

To prove that $\boldsymbol{A}_{DS}$ is doubly stochastic we need to prove that both the rows and columns of $\boldsymbol{A}_{DS}$ all sum to unity. The columns of $\boldsymbol{A}_{DS}$ must all sum to unity as the constants $c_1, \dots, c_N$ are chosen in such a way as to enforce this. It is however necessary to prove that the rows still sum to unity as it seems that the normalisation of the columns would perturb the normalised structure of the rows.

To prove that the rows of $\boldsymbol{A}$ still sum to unity after the method described above is applied to the columns of $\boldsymbol{A}$ we need to prove that the sum of the row entries in matrix $\boldsymbol{A}_{DS}$ sum to unity. Written mathematically

$$\sum_j \frac{c_j + \beta_{ij}}{K_c} = 1. \quad (4.19)$$

The left hand side of equation (3.22) can be written as

$$\sum_j \frac{c_j}{K_c} + \sum_j \frac{\beta_{ij}}{K_c}$$

but

$$\sum_j \beta_{ij} = 1$$

as we calculated all β_{ij} to have this property. Then

$$\sum_j \frac{c_j}{K_c} + \sum_j \frac{\beta_{ij}}{K_c} = \sum_j \frac{c_j}{K_c} + \frac{1}{K_c}$$

and remembering that

$$c_j = \frac{\sum_i \beta_{ij_{max}} - \sum_i \beta_{ij}}{N}$$

this can be rewritten as

$$\sum_j \left(\frac{\sum_i \beta_{ij_{max}} - \sum_i \beta_{ij}}{NK_c} \right) + \frac{1}{K_c}$$

or

$$\left(\frac{\sum_j \sum_i \beta_{ij_{max}} - \sum_j \sum_i \beta_{ij}}{NK_c} \right) + \frac{1}{K_c}. \quad (4.20)$$

Also notice that

$$\sum_j \sum_i \beta_{ij_{max}} = \sum_i \beta_{ij_{max}} \text{ and } \sum_j \sum_i \beta_{ij} = N.$$

Then rewrite (3.23) as

$$\sum_i \frac{\beta_{ij_{max}}}{NK_c}. \quad (4.21)$$

Remembering that

$$\beta_{ij} = \frac{(\alpha_{ij_{max}} + r_i)}{K_r} \quad (4.22)$$

and

$$r_i = \left(\frac{\sum_j \alpha_{i_{max}j} - \sum_j \alpha_{ij}}{N} \right) \quad (4.23)$$

and

$$K_r = \sum_j \alpha_{i_{max}j} \quad (4.24)$$

and

$$K_c = \sum_i \beta_{ij_{max}} \quad (4.25)$$

we rewrite equation (3.28) using equation (3.25) and (3.26) as

$$\sum_i \frac{(\alpha_{ij_{max}} + r_i)}{K_r}$$

$$= \sum_i \frac{\sum_i \left(\alpha_{ij_{max}} + (\sum_j \alpha_{i_{max}j} - \sum_j \alpha_{ij})/N \right)}{\sum_j \alpha_{i_{max}j}}$$

$$= \frac{\sum_i \alpha_{ij_{max}} + \frac{1}{N}\sum_j \alpha_{i_{max}j} - \frac{1}{N}\sum_i \sum_j \alpha_{ij}}{\sum_j \alpha_{i_{max}j}}$$

$$K_c = \frac{N\sum_i \alpha_{ij_{max}} + \sum_j \alpha_{i_{max}j} - \sum_i \sum_j \alpha_{ij}}{N\sum_j \alpha_{i_{max}j}}$$

and expanding (3.24) again

$$\sum_i \frac{\beta_{ij_{max}}}{NK_c} = \sum_i \frac{\left(\alpha_{ij_{max}} + r_i\right)/K_r}{NK_c}$$

$$= \sum_i \frac{\left(\alpha_{ij_{max}} + \left(\sum_j \alpha_{i_{max}j} - \sum_j \alpha_{ij}\right)/N\right)}{NK_cK_r}$$

$$= \frac{\sum_i \alpha_{ij_{max}} + \frac{1}{N}\sum_i \sum_j \alpha_{i_{max}j} - \frac{1}{N}\sum_i \sum_j \alpha_{ij}}{NK_cK_r}$$

$$= \frac{\sum_i \alpha_{ij_{max}} + \frac{1}{N}\sum_j \alpha_{i_{max}j} - \frac{1}{N}\sum_i \sum_j \alpha_{ij}}{NK_cK_r}. \qquad (4.26)$$

Considering the denominator of (3.29)

$$NK_cK_r = N \times \sum_j \alpha_{i_{max}j} \times \left(\frac{N \times \sum_i \alpha_{ij_{max}} + \sum_j \alpha_{i_{max}j} - \sum_i \sum_j \alpha_{ij}}{N \times \sum_j \alpha_{i_{max}j}}\right)$$

$$= N\sum_i \alpha_{ij_{max}} + \sum_j \alpha_{i_{max}j} - \sum_i \sum_j \alpha_{ij.}$$

And substituting this back into (3.29) we get

$$\frac{\sum_i \alpha_{ij_{max}} + \frac{1}{N}\sum_j \alpha_{i_{max}j} - \frac{1}{N}\sum_i \sum_j \alpha_{ij}}{N \times \sum_i \alpha_{ij_{max}} + \sum_j \alpha_{i_{max}j} - \sum_i \sum_j \alpha_{ij}}$$

or

$$\frac{N\sum_i \alpha_{ij_{max}} + \sum_j \alpha_{i_{max}j} - \sum_i \sum_j \alpha_{ij}}{N\sum_i \alpha_{ij_{max}} + \sum_j \alpha_{i_{max}j} - \sum_i \sum_j \alpha_{ij}} = 1$$

which completes the proof.

4.3 COMPUTATIONAL COMPLEXITY OF ALGORITHM

To find the computational complexity of the algorithm we calculate the computational expense of normalising the rows and then double this number as the process applied to the columns will require the exact same number of operations. The row normalisation implies finding $\mathbf{A}_{\mathrm{S}}$ and to do this requires calculating expression (3.16) N times. Calculating a row total needed in expression (3.16) requires $N-1$ summations, expression (3.16) then requires 1 more summation and 1 scalar multiplication. This results in a total of

$$N\big((N-1)+1+1\big)$$

or

$$N^2+N$$

operations. Calculation of expression (3.20) requires another N^2 additions and N^2 scalar multiplications for a total of

$$N^2+N+N^2+N^2$$

or

$$3N^2+N$$

needed to find $\mathbf{A}_{\mathrm{S}}$. The exact same number of operations is needed for the column normalisation process needed to find $\mathbf{A}_{\mathrm{DS}}$. This results in a total of

$$2\times(3N^2+N)$$

or

$$6N^2+2N$$

operations. The result is an algorithm of complexity order $\mathrm{O}(N^2)$ operations.

4.4 EXAMPLE

To allow the reader an easier grasp of the operation of the algorithm an example of the algorithm is included for a simple 3×3 matrix.

Consider the Benefit matrix

$$\boldsymbol{B} = \begin{bmatrix} 2 & 2 & 3 \\ 2 & 1 & 4 \\ 6 & 3 & 1 \end{bmatrix}.$$

The optimal Assignment of this matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

With row and column totals

$$\begin{matrix} \begin{bmatrix} 2 & 2 & 3 \\ 2 & 1 & 4 \\ 6 & 3 & 1 \end{bmatrix} & \begin{matrix} 7 \\ 7 \\ 10 \end{matrix} \\ \begin{matrix} 10 & 7 & 8 \end{matrix} & \end{matrix}$$

now

$$r_1 = r_2 = \frac{10 - 7}{3} = 1$$

and

$$r_3 = 0.$$

Our new stochastic matrix is

$$\begin{bmatrix} 2+1 & 2+1 & 3+1 \\ 2+1 & 1+1 & 4+1 \\ 6+0 & 3+0 & 1+0 \end{bmatrix} \times \frac{1}{10}$$

or

$$\begin{bmatrix} 3/10 & 3/10 & 4/10 \\ 3/10 & 2/10 & 5/10 \\ 6/10 & 3/10 & 1/10 \end{bmatrix}.$$

Considering the row and column totals

$$\begin{matrix}\begin{bmatrix}3/10 & 3/10 & 4/10\\ 3/10 & 2/10 & 5/10\\ 6/10 & 3/10 & 1/10\end{bmatrix} & \begin{matrix}1\\ 1.\\ 1\end{matrix}\\ 12/10 \quad 8/10 \quad 10/10 & \end{matrix}$$

Clearly the rows sum to unity but not the columns, to ensure that the matrix becomes doubly stochastic we need to apply the same procedure to the columns of the normalised $\boldsymbol{B}$ matrix.

$$c_1 = 0$$

$$c_2 = \frac{12/10 - 8/10}{3} = 4/30$$

and

$$c_3 = \frac{12/10 - 10/10}{3} = 2/30.$$

Now

$$\begin{bmatrix}\frac{3}{10}+0 & \frac{3}{10}+\frac{4}{30} & \frac{4}{10}+\frac{2}{30}\\ \frac{3}{10}+0 & \frac{2}{10}+\frac{4}{30} & \frac{5}{10}+\frac{2}{30}\\ \frac{6}{10}+0 & \frac{3}{10}+\frac{4}{30} & \frac{1}{10}+\frac{2}{30}\end{bmatrix}\times\frac{1}{12/10}$$

or

$$\begin{bmatrix}\frac{3}{10} & \frac{13}{30} & \frac{14}{30}\\ \frac{3}{10} & \frac{10}{30} & \frac{17}{30}\\ \frac{6}{10} & \frac{13}{30} & \frac{5}{30}\end{bmatrix}\times\frac{10}{12}$$

or

$$\begin{bmatrix}\frac{9}{36} & \frac{13}{36} & \frac{14}{36}\\ \frac{9}{36} & \frac{10}{36} & \frac{17}{36}\\ \frac{18}{36} & \frac{13}{36} & \frac{5}{36}\end{bmatrix}$$

with row and column totals

$$\begin{bmatrix} \frac{9}{36} & \frac{13}{36} & \frac{14}{36} \\ \frac{9}{36} & \frac{10}{36} & \frac{17}{36} \\ \frac{18}{36} & \frac{13}{36} & \frac{5}{36} \end{bmatrix} \begin{matrix} 1 \\ 1 \\ 1 \end{matrix}$$

$$1 \quad 1 \quad 1$$

and the same optimal assignment as the original matrix $\boldsymbol{B}$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

4.5 DISCUSSION OF ASSIGNMENT PRESERVING PROJECTION METHOD

The assignment preserving doubly stochastic projection method offers a computationally inexpensive method of finding a doubly stochastic matrix near the original matrix and preserving the assignment. It turns out that this approximation could find use in many similar applications to Birkhoff's doubly stochastic approximation as both methods offer a manner of finding a doubly stochastic approximation to a given matrix. In some applications of Birkhoff's theorem the optimal assignment plays a very important role such as found in [25,26]. Originally the method was developed in an attempt to solve the assignment problem by first projecting a matrix into the doubly stochastic polytope and then continually projecting the matrix onto the convex edges of the polytope until a permutation matrix is obtained. Such a permutation matrix would then be a solution to the assignment problem. The author could however not find a method nearly as efficient as some of the existing methods for solving the assignment problem to find the nearest permutation matrix to this new doubly stochastic matrix and finding a solution to this problem probably constitutes finding a novel solution to the assignment problem.

CHAPTER 5

5.1 CONCLUSION

The OACE algorithm shows that a process as simple as iterative weighted normalisation applied to a square matrix converges to a sparse matrix in the vicinity of a permutation matrix that corresponds to an optimal or near optimal assignment for that matrix. Although the Computational Ecology model as derived by Hogg and Huberman was used to arrive at the OACE algorithm it turns out that the whole Computational Ecology shell can be removed and the OACE algorithm viewed simply as a process of iterative weighted normalisation. It must be acknowledged though that the Computational Ecology structure does offer a convenient basis for interpretation of the dynamics of the OACE algorithm and even if it is not necessary for the derivation of the algorithm it certainly acted as an inspiration in finding it. In general the practical use of the Computational Ecology Model turns out to be rather limited. Successful applications of the model are scarce and the original authors offer very limited examples.

The second important result in this document is the assignment preserving doubly stochastic projection. This allows one to find a doubly stochastic matrix with the same optimal assignment as the original matrix after only $O(N^2)$ operations. The method was originally developed in an attempt to solve the assignment problem with the idea of first projecting a matrix into the doubly stochastic polytope and then finding the nearest permutation matrix to this doubly stochastic matrix. We could however not find a method nearly as efficient as existing methods of solving the assignment problem to find the nearest permutation matrix to this new doubly stochastic matrix. This projection method does, however, seem to have some other possible applications especially in optimisation applications similar to those that Birkhoff's theorem finds.

5.2 FUTURE WORK

This document lays the foundation for much future work and the author believes the following points to be the most important.

5.2.1 Analytical derivation of OACE algorithm

A better analytical derivation of the OACE algorithm would be very welcome. The simplistic approach of a mere weighted normalisation of the original benefit matrix finds the optimal assignment with astonishing success but there is much room for improvement. With some alterations to the OACE algorithm it should be possible to find an algorithm that consistently finds the exact optimal solution to the assignment problem.

5.2.2 Applications of the OACE algorithm

The OACE algorithm does offer some utility but there are no applications of the algorithm where it outperforms all its competitors. There is still further scope for a study into advantages that this algorithm offers and applications it would excel at.

5.2.3 Application of assignment preserving doubly stochastic projection

This projection method is probably the result in this document that offers most utility. The study of the possible applications of the method especially for optimisation purposes similar to those that Birkhoff's theorem finds, needs to be expanded upon. The possibility of using this projection method as the basis of a method for solving the assignment problem also needs to be investigated.

BIBLIOGRAPHY

[1] H. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, 1955, pp. 83-97.

[2] J. Munkres, "Algorithms for the Assignment and Transportation Problems," *Journal of the Society of Industrial and Applied Mathematics*, vol. 5, 1957, pp. 32-38.

[3] G.B. Dantzig and M.N. Thapa, *Linear Programming: Theory and extensions*, Springer, 1997.

[4] D.P. Bertsekas, "Auction Algorithms for Network Flow Problems: A Tutorial Introduction," *Computational Optimization and Applications*, vol. 1, 1992, pp. 7-66.

[5] J. Kosowsky and A. Yuille, *Solving the Assignment Problem with Statistical Physics*, IEEE, 1991.

[6] B.A. Huberman and T. Hogg, "The Behavior of Computational Ecologies," *The Ecology of Computation*, 1988, pp. 77-115.

[7] J.O. Kephart, T. Hogg, and B.A. Huberman, "Dynamics of Computational Ecosystems," *Physical Review A*, vol. 40, 1989, pp. 404-421.

[8] T. Hogg and B.A. Huberman, "Controlling chaos in distributed systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, 1991, pp. 1325-1332.

[9] F. Vermaak and M. Van Wyk, "Approximate Solution to the Optimal Assignment Problem Using a Computational Ecology Based Approach," *2010 International Conference on Measuring Technology and Mechatronics Automation*, IEEE, IEEE, 2010, pp. 699-702.

[10] M. Van Wyk and J. Clark, "An Algorithm for Approximate Least-Squares Attributed Graph Matching," *Problems in Applied Math. and Computational Intelligence*, 2001, pp. pp. 67-72.

[11] H. Imai and M. Iri, "Computational-Geometric Methods for Polygonal Approximations of a Curve," *Computer Vision, Graphics and Image Processing*, vol. 36, 1986, pp. 31-41.

[12] Z. Ying and D. Castanon, "Statistical Model for Human Face Detection Using Multi-Resolution Features," *1999 International Conference on Information Intelligence and Systems (ICIIS'99)*, IEEE, 1999, p. 560.

[13] J. Wastlund, "Random Assignment and Shortest Path Problems," *Discrete Mathematics And Theoretical Computer Science*, 2003, pp. 1-11.

[14] R.E. Burkard and E. Çela, "Linear Assignment Problems and Extensions," *Handbook of Combinatorial Optimization*, 1998.

[15] W. Brogan, "Algorithm for Ranked Assignments with Aplication to Multi Object Tracking.," *Journal of Guidance*, 1989, pp. 357-364.

[16] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching Output Queuing With a Combined Input/Output-Queued Switch," *Selected Areas in Communications*, vol. 17, 2002, pp. 1030 - 1039.

[17] I.N. Bronshtein and K.A. Semendyayev, *Handbook of mathematics*, New York: Springer-Verlag, 2004.

[18] S. Sherman, "Doubly Stochastic Matrices and Complex Vector Spaces," *American Journal Of Mathematics*, vol. 77, 1955, pp. 245-246.

[19] H. Coxeter, *Introduction to Geometry*, Wiley, 1989.

[20] T. Bisztriczky, P. McMullen, R. Schneider, and A.I. Weiss, *Polytopes: Abstract, Convex and Computational*, Kluwer Academic, 1994.

[21] D.G. Luenberger, *Optimization by Vector Space Methods*, New York, NY, USA: John Wiley & Sons, Inc., 1997.

[22] G. Birkhoff, "Tres Observaciones Sobre el Algebra lLineal, Rev," *Rev. Univ. Nacionalde Tacuman A*, vol. 4, 1946, pp. 147-151.

[23] R. Sinkhorn and P. Knopp, "Concerning Nonnegative Matrices and Doubly Stochastic Matrices," *Pacific Journal of Mathematics*, vol. 21, 1967, pp. 10-11.

[24] J. Breslaw, "A linear Programming Solution to the Faculty Assignment Problem," *Socio-Economic Planning Sciences*, vol. 10, 1976, pp. 227-230.

[25] J. Kosowsky and A. Yuille, "The Invisible Hand Algorithm: Solving the Assignment Problem with Statistical Physics," *Neural Networks*, vol. 7, 1994, pp. 477-490.

[26] S. Gold and A. Rangarajan, "A Graduated Assignment Algorithm for Graph Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, 1996, pp. 377-388.

[27] R.E. Burkard and E. Çela, "Linear Assignment Problems and Extensions," *Handbook of Combinatorial Optimization*, 1998.

[28] D.P. Bertsekas, "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," *Annals of Operations Research*, vol. 14, 1988, pp. 105-123.

[29] D.P. Bertsekas, "An Auction Algorithm for Shortest Paths 1," *SIAM journal on Optimization*, 1990.

[30] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Englewood Cliffs N. J.: Prentice-Hall (re- published in 1997 by Athena Scientific, Belmont, MA), 1989.

[31] D.P. Bertsekas and D.A. Castañon, "The Auction Algorithm for Transportation Problems," *Annals of Operations Research*, vol. 20, 1989, pp. 67-96.

[32] D.P. Bertsekas and D.A. Castañon, "A Generic Auction Algorithm for the Minimum Cost Network Flow Problem," *Computational Optimization and Applications*, vol. 2, 1993, pp. 229-260.

[33] H. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, 1955, pp. 83-97.

[34] E.D. Nering and A.W. Tucker, *Linear Programs and Related Problems*, Academic press Inc, 1993.

[35] D. Kőnig, "Gráfok és Mátrixok," *Matematikai és Fizikai Lapok*, vol. 38, 1931, p. 116–119.

[36] H. Anton and C. Rorres, *Elementary Linear Algebra: Applications version*, John Wiley & Sons, Inc., 2000.

[37] D.P. Bertsekas, "The Auction Algorithm for Assignment and Other Network Flow Problems: A Tutorial," *Interfaces*, vol. 20, 1990, pp. 133 - 149.

[38] D.P. Bertsekas and D.A. Castañon, "A Generic Auction Algorithm for the Minimum Cost Network Flow Problem," *Computational Optimization and Applications*, vol. 2, 1993, pp. 229-260.

[39] C. Peterson and B. Soderberg, "Neural Optimization," *The Handbook of Brain Research and Neural Networks*, MIT Press, 1998, pp. 617-622.

[40] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, New York: Oxford University Press, 1999.

[41] J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, The MIT Press, 1992.

[42] C.A. Coello, "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques," *Knowledge and Information Systems*, vol. 1, 1998, pp. 269-308.

[43] B.A. Huberman and T. Hogg, "The Emergence of Computational Ecologies," *1992 Lectures in Complex Systems*, Addison-Wesley, 1993, pp. 185-205.

[44] A. Smith, *An Inquiry into the Nature and Whealth of Nations*, London: William Clowes and Sons, 1776.

[45] B.A. Huberman and T. Hogg, "Controlling Chaos in Distributed Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, 1991, pp. 1325-1332.

[46] T. Yamasaki and T. Ushio, "An Application of a Computational Ecology Model to a Routing Method in Computer Networks," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 32, 2002, pp. 99-106.

[47] J.S. Bridle, "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters," *Advances in neural information processing systems 2*, 1990, pp. 211-217.

[48] B.L. Schwartz, "A computational analysis of the Auction Algorithm," *European Journal Of Operational Research*, vol. 74, 1994, pp. 161-169.

[49] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Berlin Heidelberg: Springer- Verlag, 2003.

[50] R. Zass and A. Shashua, "Doubly Stochastic Normalization for Spectral Clustering," *Advances in Neural Information Processing Systems*, 2006, pp. 1569-1576.

[51] W. Glunt, T.L. Hayden, and R. Reams, "The Nearest 'Doubly Stochastic' Matrix to a Real Matrix with the same First Moment," *Numerical Linear Algebra with Applications*, vol. 5, 1998, pp. 475-482.

[52] W. Glunt, T.L. Hayden, and R. Reams, "The Nearest Generalized Doubly Stochastic Matrix to a Real Matrix With the Same First and Second Moments," *Computational & Applied Mathematics*, vol. 27, 2008, pp. 201-210.

APPENDIX A: MATLAB CODE

A1 Standard OACE algorithm

```
function [sim_ans rd] = OA_CE_mat_norm_func_check_rd_corr(b)

% Function that implements OA_CE using MATLAB's matrix operations
% This implementation runs the OACE algorithm untill f is row dominant or
% stationary. The code only allows for 10 000 iterations.

% the variable rd == 1 if f becomes row-dominant and rd == 0 if not

B = b;

siz = size(B);
n = siz(1);

f(:,:,1) = zeros(n,n) + 1;

k = 0;

stat_max_r1_prev = 0;

rd = 0;

while k < 10000

    k = k + 1;

    alpha_mat = eye(n);

    for i = 1:n
        alpha(i) = sum(B(:,i).*f(:,i,k));
        alpha_mat(i,i) = alpha_mat(i,i)*1/alpha(i);
    end

    B_col_norm = B.*f(:,:,k)*alpha_mat;

    f(:,:,k+1) = B_col_norm;

    gamma_mat = eye(n);

    for i = 1:n
        gamma(i) = sum(f(i,:,k+1));
        gamma_mat(i,i) = gamma_mat(i,i)* 1/gamma(i);
    end

    B_row_norm = (B_col_norm'*gamma_mat)';

    f(:,:,k+1) = B_row_norm;

    flag_row_dom = row_dominant(f(:,:,k+1));

        %%%         flag_stat_max = 1
        % check if the f mat is stationary by checking its max entry in the
```

```
        % first row

        stat_max_r1 = max(f(1,:,k+1));
        if abs(stat_max_r1 - stat_max_r1_prev) < 1e-5
            break
        end

        stat_max_r1_prev = stat_max_r1;

        %%%

    if (flag_row_dom == 1)
        rd = 1;
        break
    elseif k == 9999
        k
    end

end

Sim_ans = zeros(n,n);

for a =1:n

    max_d = max(f(a,:,k+1));

    for r = 1:n

        if f(a,r,k+1) == max_d

            Sim_ans(a,r) = 1;

        end
    end
end

sim_ans = Sim_ans;

end
```

```
function [row_dom] = row_dominant(b)
% function that evaluates whether a matrix is row dominant (row dominant if
% in every row the max element is located in a different column)

b_mat = b;

siz = size(b);

n = siz(2);

row_dom = 1;

max_n = zeros(n,1);
max_index = zeros(n,1);

for r = 1:n

    for col = 1:n

        if b_mat(r,col) > max_n(r)
            max_n(r) = b_mat(r,col);
            max_index(r) = col;
        end

    end

end

for k = 1:n

    eq_max = max_index(k) == max_index;

    if sum(eq_max) > 1

        row_dom = 0;

    end

end


end
```

A2 Assignment preserving doubly stochastic matrix projection

```
function [DS_b] = DS_same_OA_func(b)
% function that that returns a doubly stochastic approximation to a matrix
% b with the same optimal assignment

siz = size(b);
n = siz(1);

scale_mat = zeros(n,n);

% Finds all the row totals of b and also the indices off the max row
max_row = 1;
for i = 1:n
    row_tot(i) = sum(b(i,:));
    if i > 1
        if row_tot(i) > row_tot(i-1)
            max_row = i;
        end
    end
end

% Compute all the additive scalars
add_scalar_r(1) = (row_tot(max_row) - row_tot(1))/n;

for i = 2:n
    add_scalar_r(i) = add_scalar_r(1) + row_tot(1)/n - row_tot(i)/n;
end

%Change b by adding the scalars and then normalize

for i = 1:n
    scale_mat(i,:) = b(i,:) + add_scalar_r(i);
end

%normalize so all rows sum to 1

scale_mat = scale_mat*1/row_tot(max_row);

% Now follow the same procedure but for the columns

% Finds all the row totals of b and also the indices off the max col
max_col = 1;
for i = 1:n
    col_tot(i) = sum(scale_mat(:,i));
    if i > 1
        if col_tot(i) > col_tot(i-1)
            max_col = i;
        end
    end
end

% Compute all the additive scalars
add_scalar_c(1) = (col_tot(max_col) - col_tot(1))/n;

for i = 2:n
    add_scalar_c(i) = add_scalar_c(1) + col_tot(1)/n - col_tot(i)/n;
end

%Change b by adding the scalars and then normalize
```

```
for i = 1:n
    scale_mat(:,i) = scale_mat(:,i) + add_scalar_c(i);
end

%normalize so all rows sum to 1

scale_mat = scale_mat*1/col_tot(max_col);

DS_b = scale_mat;

end
```

APPENDIX B: PUBLICATIONS

B1 Approximate Solution to the Optimal Assignment Problem using a Computational Ecology based Approach

F. Vermaak, M. A. Van Wyk