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

F. Vermaak, M. A. Van Wyk
School of Electrical and Software engineering
University of the Witwatersrand
Johannesburg, South-Africa
email: franswillem.vermaak@gmail.com

***Abstract*—The assignment problem, a classical combinatorial optimization problem is attempted in a novel manner using an agent/resource allocation based model known as a Computational Ecology. A simple algorithm that reduces to an iterative weighted normalisation process is obtained that converges to the optimal or near optimal assignment of a given positive square benefit matrix.**

Keywords- optimal assignment, computational ecology, agent

I. INTRODUCTION

The optimal assignment problem is a classical combinatorial optimization problem. A common example of this problem is matching a number of machines and workers, all with different skills levels of operating the respective machines, such that each worker operates one machine and the total skill level is maximized. The problem finds many applications such as logistical matching, graph matching [1], tracking etc. A very important type of network flow problem the linear network flow problem can be reformulated as an assignment problem [2]. Various methodologies have been developed to find a solution to this problem, the Hungarian method by Kuhn and Munkres being of the first [3]. Dantzig solved the problem using linear programming [4] and more recently Bertsekas developed the Auction algorithm that solves the problem very efficiently [5]. Kosowsky and Juille also managed to find a solution to the problem using a statistical physics approach [6]. In this document the idea of solving the optimal assignment problem using a computational ecology based approach as developed by Hogg and Hubermann [7] is used. The result is a simple algorithm referred to as the OACE (Optimal Assignment by Computational Ecology) algorithm.

In section II an overview of the assignment problem is given followed by the necessary computational ecology background. We then look at the application of the computational ecology model to the assignment problem in section III and finally some analysis of the results and possible alterations and improvements such as preconditioning of the benefit matrix are considered.

II. BACKGROUND

A. Optimal Assignment Problem

The optimal assignment problem, also known as the bipartite weighted matching problem, is a classical combinatorial optimization problem. Given a set of individuals and tasks we intend to optimally match each individual with a certain task. Each individual-task pairing has an associated benefit, b_{ij}, the benefit of assigning individual j to a task i and collectively described by the benefit matrix $\boldsymbol{B}$. Given N individuals and N tasks we set up an assignment matrix $\boldsymbol{A}$ such that $a_{ij} = 1$ (a_{ij} is the entry found in row j, column i of matrix $\boldsymbol{A}$) when individual j is assigned to task i, and $a_{ij} = 0$ otherwise. The optimal assignment consists of assigning a single individual to each task such that the sum

$$T_b = \sum_{i}^{N}\sum_{j}^{N} a_{ij} b_{ij} \qquad (1)$$

is maximized. The matrix $\boldsymbol{A}$ is a type of matrix known as a permutation matrix, i.e. it has a single 1 in every row and a single 1 in every column with all other entries zeros.

B. Computational Ecology

The Hogg-Huberman computational ecology model is a set of dynamical equations that describe the dynamics of an agent-resource allocation problem. Agents are viewed as separate units deciding between different resources at a certain rate with some probability of preference for the available resources. It is possible to view the collection of agents and resources as a dynamical system and describe properties such as population density using differential equations. To achieve this we need to attribute properties to each agent such as decision rate and the probability of preferring one resource to the others. Hogg and Hubermann derives the following equation using this approach [7]

978-0-7695-3962-1/10 $26.00 © 2010 IEEE
DOI 10.1109/ICMTMA.2010.811

$$\frac{d\bar{n}_i}{dt} = \alpha(n\bar{\rho}_i - n_i) \qquad (1)$$

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

Or in normalized and discretized form [8]

$$f_i(k+1) = f_i(k) + \alpha(\rho_i(k) - f_i(k)) \qquad (3)$$

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

Eq. (3) can be interpreted as an update equation with α determining the degree of influence of ρ on the next state of $f_i(k)$. The population or agent distribution develops according to this probability term. Literature does however not stipulate how this probability is to be calculated, it is actually problem specific. It turns out that the probability term, ρ, captures much of the dynamics of the system.

III. Computational Ecology applied to the Optimal Assignment problem

A. Methodology

To cast the optimal assignment problem into the computational ecology structure suggests viewing the so called 'agents' as 'individuals' and the 'resources' as 'tasks'. Simply allocating the status of agent to individual does however not lend itself to an ecology that develops meaningfully. It proves more useful to view each individual as a separate group of agents that need to distribute themselves between all the different resources available as depicted in Fig.1. This implies that we are now not trying to match *N* individuals to *N* tasks but that we have *N* groups of agents distributing themselves between all the different tasks or resources. In Fig.1 individual 1 now corresponds to agent group 1. The advantage of having *N* separate groups of agents is that it allows a density dependant development of the agent distribution, i.e. if a large fraction of agents from group 1 are using resource 1 the other agents groups would be less inclined to use resource 1. The optimal resource indicated by a group of agents is the one preferred by the greatest fraction of agents in that group.

B. Structure

As a fraction of agents are distributed to every resource the result is a matrix $\boldsymbol{F(k)}$, referred to as the population matrix, of the same size as matrix $\boldsymbol{B}$ and dependant on *k*, the number of iterations. Entry f_{ij} represents the fraction of agents in group *j* allocated to task *i*. The entry f_{ij} develops according to the probability ρ_{ij} described by a matrix $\boldsymbol{P(k)}$. The determination of $\boldsymbol{P(k)}$ is discussed in the next section.

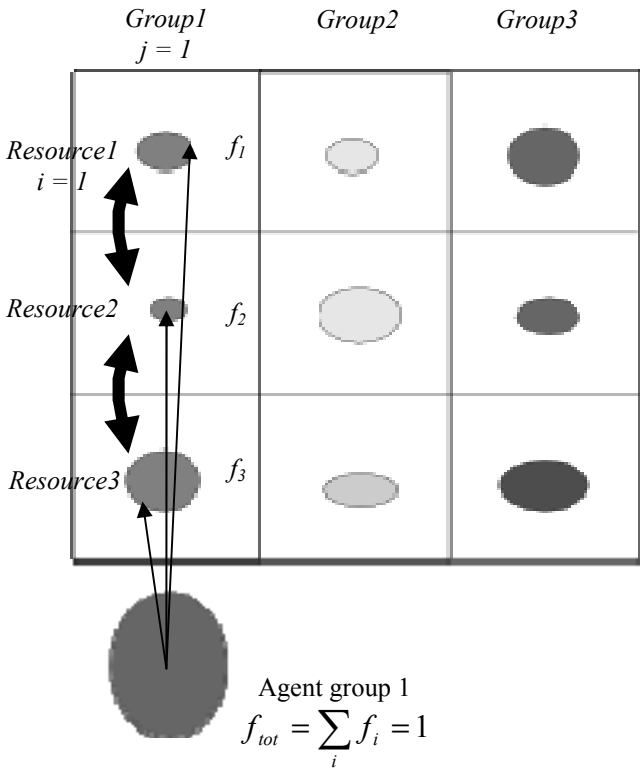

Figure 1. Illustration of agent distribution and dynamics between different tasks. The big group at the bottom of the figure distributes itself into smaller groups between the different tasks, these smaller groups then evolve according to the process described in section III.C.

C. Determination of probability term

The probability term according to which the dynamical equation develops is a density dependant term proportional to the density weighed benefit of being matched with a certain resource and indirectly proportional to sum of the density weighed benefits over all tasks (if a term contains $f_{ij}(k)$ it is density dependant)

$$\rho_{ij}(k) = \frac{f_{ij}(k)b_{ij}}{\sum_i f_{ij}(k)b_{ij}} \qquad (4)$$

The probabilities corresponding to one group of agents are then normalized over the different tasks to ensure summation to unity according to (5).

$$\rho_{ij}(k) = \frac{\rho_{ij}(k)}{\sum_j \rho_{ij}(k)} \qquad (5)$$

D. Algorithmic implementation

The OACE algorithm to solve the optimal assignment problem can be summarised as:

1) Initialise f

$$f_{ij} = 1/n$$

with n the dimension of the matrix (It is actually possible to initialize f_{ij} as any positive, non-zero real number as long as all f corresponding to the same group are initialized to the same number)

2) Determine the probability of agent j choosing task i according to (4).

$$\rho_{ij}(k) = \frac{f_{ij}(k) \times b_{ij}}{\sum_i f_{ij}(k) \times b_{ij}}$$

3) Normalise ρ row-wise according to (5)

$$\rho_{ij} = \frac{\rho_{ij}}{\sum_j \rho_{ij}}$$

4) Update f according to (2)

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

5) Return to step 2

6) Repeat until $\boldsymbol{F(k)}$ converges (When $\boldsymbol{F(k)} = \boldsymbol{F(k+1)}$)

7) Approximate $\boldsymbol{F(k)}$ with its closest permutation matrix.
The simplest method is to apply the so called Greedy Algorithm, assigning the value one to the maximum entry in a row and a zero to all other entries, other methods are discussed in the following section.

IV. Results and Analysis

A. Simulation

All matrix entries were randomly initialised with real numbers from the interval [0,1] (this method is sufficient as the algorithm normalises all entries). The OACE algorithm was then run until convergence to a stationary $\boldsymbol{F(k)}$ was obtained. Alternatively the algorithm could be terminated

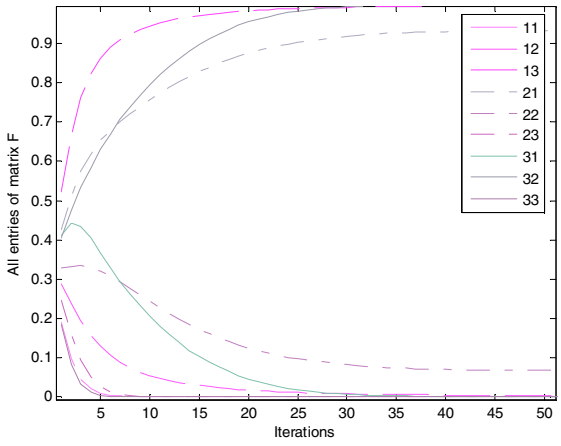

Figure 2. All entries of matrix **F** as a function of the number of iterations

once matrix $\boldsymbol{F}$ became row-dominant. A matrix will be described as row (resp. column) dominant if the maximum element in each row (resp. column) is located in a different column. This method led to slightly inferior performance as there are isolated situations where the greedy algorithm yields different results but $\boldsymbol{F}$ becomes row-dominant long before it is stationary, hence an assignment is obtained in far less iterations. Fig. 2 shows an example of the development of all the entries of $\boldsymbol{F}$ for

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

The algorithm reaches a stationary solution around 40 iterations and it is clear that three entries tend to numbers close to unity while all the others tend to numbers close to zero as expected for a 3×3 assignment. Fig. 2 also shows how quickly this example becomes row dominant.

B. General

The OACE algorithm as given in section III.C although simple does not always converge to the optimal solution, with deteriorating performance as the size of the benefit matrix grows. Fig. 3 shows the percentage of optimal solutions found when applying OACE to randomly generated benefit matrices of order 1 to 70. Clearly the algorithm struggles to find the optimal solution when the matrices are large. The algorithm does however find solutions very close to the optimal when not finding the optimal. The simple OACE algorithm does also not always render a permutation matrix yet always something very close to one. In such a case it is easy transform such a matrix into a permutation matrix by changing the position of a small number of 1's in the matrix.

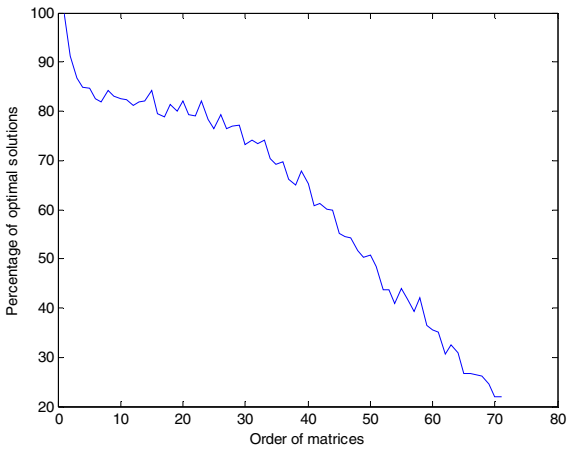

Figure 3. Percentage of optimal solutions found using the OACE algorithm against the order of the respective benefit matrices

C. *Preconditioning of the benefit matrix*

Certain preconditioning methods could be implemented to improve the performance of the OACE algorithm. The first simple heuristic approach constitutes adding an offset to all entries (adding 1 for $\boldsymbol{B}$ with all entries in the interval [0,1] showed good results) of the benefit matrix. This tends to lessen the influence of very small numbers in the normalization processes and led to an improvement in performance.

The second method consists of embedding the benefit matrix into the set of doubly stochastic matrices but conserving the optimal assignment and then applying the OACE algorithm. This leads to substantial improvements in accuracy for smaller matrices finding the optimal solution to about 98 percent of randomly initialized 3×3 benefit matrices but with worse performance for larger matrices.

D. *Discussion*

When α in (3) is chosen as $\alpha = 1$, (3) reduces to

$$f_{ij}(k+1) = \rho_{ij}(k) \tag{6}$$

This implies that the algorithm can be viewed as an iterative process of weighted column normalization followed by a row-wise normalization. This process of iterative normalization bares some similarity to an algorithm for solving the assignment problem derived by Kosowsky and Juille [6,9].

V. FUTURE WORK

Useful additional work would include a mathematical analysis of the convergence properties of the population matrices. A better understanding of when a given benefit matrix will converge to a permutation matrix and if so when will it be the optimal assignment or a slightly suboptimal solution need to be investigated.

VI. CONCLUSION

The application of the Hogg-Huberman computational ecology model to the optimal assignment problem shows some merit. A simple algorithm is derived that finds an optimal or close to optimal solution to the assignment problem. In terms of speed and computational efficiency the algorithm is inferior to algorithms such as Bertsekas' auction algorithm but performs comparable and superior to algorithms such as the Invisible hand algorithm [6]. Another important note is that it is possible to remove the computational ecology cast and view the algorithm as an iterative weighted normalization process. The computational ecology model was however the inspiration of the solution as presented here.

REFERENCES

[1] M.A. van Wyk,"Approximate least-squares attributed graph matching via bayesian inference," The transactions of the SA institute of electrical engineers, June 2005, pp 87-92

[2] D.P. Bertsekas,"Auction Algorithms for Network Flow Problems: A Tutorial Introduction," Computational Optimization and Applications, Kluwer Academic Publishers, Netherlands, Vol. 1, 1992, pp.7-66

[3] H.W. Kuhn,"The hungarian method for the assignment problem," Naval research logistics quarterly, Vol 2, 1955, pp 83-97

[4] B. Dantzig, "Linear programming and extensions," Princeton Univeristy Press, August 1998

[5] D.P. Bertsekas, "Auction algorithms for network flow problem: A tutorial introduction," Computational optimization and applications, No 1, 1992, pp 7-66

[6] J.J. Kosowsky, A.L. Juille, "The invisible hand Algorithm: Solving the assignment problem with statistical physics,'' Neural Networks, Pergamon, Vol. 7, No 3, 1994, pp. 477-490

[7] B. A. Huberman, T. Hogg, "The behavior of computational ecologies," The Ecology of computation, Elesvier, 1988, pp.77-115.

[8] T. Yamasaki, T. Ushio, "An application of a Computational Ecology Model to a routing Method in Computer Networks," IEEE Transactions on systems, man, and cybernetics- Part B: Cybernetics, Vol. 32, No 1, February 2002

[9] A. Rangarajan, S. Gold,' "A graduated assignment algorithm for graph matching," IEEE Transactions on pattern analysis and machine intelligence, Vol 18, No 4, 1996, pp. 377-388