Auction Algorithm: A Brief Discussion for Linear Assignment Methods

Ali Emre Balcı
Middle East Technical University
Electrical Electronics Engineering Department
Ankara, Turkey
balci.ali@metu.edu.tr

Kurtuluş Kerem Şahin
Middle East Technical University
Electrical Electronics Engineering Department
Ankara, Turkey
kerem.sahin_02@metu.edu.tr

***Abstract*—This paper presents the so called Auction Algorithm, which was formulated by Dimitri Bertsekas in 1979 as a solution to the classical assignment problem. Formulation of the algorithm is revisited and simulated.**

***Index Terms*—auction, assignment**

I. Introduction

The Auction Algorithm was first presented as an alternative solution to the classical assignment problem in Dimitri Bertsekas' 1979 paper [1]. The key feature of this algorithm is that it is designed to be executed by independent agents simultaneously. This was a significant methodological change at the time, it had a great impact on development of multi agent decision mechanisms. In this document, we examined the Auction Algorithm, compared it empirically with another assignment problem solver algorithm and discussed the results.

II. Methodology

A. The Auction Algorithm

Let $n \in \mathbb{N}$, $L = \{(i,j) : i,j \leq n,\ i,j \in \mathbb{N}\}$ and define a function,

$$\begin{aligned} f &: L \to \mathbb{N} \\ f(i,j) &= b_{ij} \end{aligned} \tag{1}$$

where each b_{ij} is called the *benefit* associated with the pair (i,j). The auction algorithm aims to maximize the total benefit B defined as,

$$B = \sum_{(i,j)\in L} k_{ij} b_{ij} \tag{2}$$

where $k_{ij} \in \{0,1\}$ with the following constraints:
for any given $\alpha \leq n$,

$$\sum_{j \leq n} k_{\alpha j} = 1 \tag{3}$$

and, similarly, for any given $\beta \leq n$,

$$\sum_{i \leq n} k_{i\beta} = 1. \tag{4}$$

To maximize B, Bertsekas [1] proposes an auction scheme where *buyers* (we called them i's) bid for *products* (we called them j's) and, of course, products are *bought* by (or paired to) their highest bidders.

For the bidding process we will introduce a list of product prices $\mathbf{p} = (p_1, p_2, ..., p_n)$ where every p_j represents price of the product number j. At the beginning of the bidding process $p_j = 0\ \forall j$ and the process ends when buyers stop bidding since prices exceed benefits, i.e. $\max\{b_{ij} - p_j\} < 0$ for any $(i,j) \in L$.

The bidding process is iterated for buyers and begins with construction of a *net gains* vector, which defines as,

$$\mathbf{g} = (g_1, ..., g_n), \text{where } g_j = b_{ij} - p_j,\ j \leq n \tag{5}$$

for the buyer number i. Then, the best and the second best possible deals are found.

$$v = \arg\max(\mathbf{g}) \tag{6}$$

$$w = \arg\max(\mathbf{g} \setminus \{g_v\}) \tag{7}$$

Where v and w are the indexes of the best and second best deals respectively. If $g_v > 0$ buyer number i sets a bidding price for g_v , otherwise he does not bid higher but he *buys*, hence takes product number v off the market. Either he bids or buys, we set $k_{iv} = 1$ according to constraints (3) and (4) but if i buys v we are not allowed to change k_{iv} back to 0 in other iterations.

If $g_v > 0$, p_v increases by the bidding amount, let's call it γ_v, as

$$\gamma_v = \begin{cases} g_v & \text{if } g_w < 0 \\ g_v - g_w & \text{if } g_w \geq 0 \end{cases} \tag{8}$$

then p_v is updated as,

$$p_v = p_v + \gamma_v + \epsilon. \tag{9}$$

Note that in (9) we added an ϵ to the bidding price. If $g_v = g_w$, bidding price is 0 so algorithm may get stuck and never terminate. Bertsekas states that an additional pre-determined small amount, i.e. $\epsilon > 0$, needs to be added to every γ_v in order to ensure that the algorithm terminates in finite steps.

Bertsekas [2] also points out that if we want the process to terminate at the optimal state we need to set an upper bound to ϵ, that is

$$\epsilon < \frac{1}{n}. \tag{10}$$

Necessary proofs can be found in Bertsekas' paper [1].

Algorithm 1 Auction Algorithm

```
1:  procedure AUCTION($\mathcal{B} \in \mathbb{R}^{n\times n}$(Benefit Matrix))
2:      Assign each buyer with a random product.
3:      $p_i \leftarrow 0$.
4:      Choose minimum bid $\epsilon$ according to. (10)
5:      while $\max_{i,j\in 1\ldots n} \{b_{ij} - p_j\} \geq 0$ do
6:          Choose $\text{buyer}_i$ randomly
7:          Find gain vector for buyer using (5)
8:          Find $v$ and $w$ using (6) and (7)
9:          Change the products so that $\text{buyer}_i$ gets the $v^{th}$ product
10:         if $\text{gain}_v > 0$ then
11:             Update the price $p_v$ using (9)
12:         else
13:             Remove $\text{product}_v$ and $\text{buyer}_i$ from the process
14:         end if
15:     end while
16: end procedure
```

B. Parallel Execution

Bidding process can be executed faster by allowing simultaneous bidding of different buyers. This can be accomplished by different microprocessors of the same computer operating simultaneously or by multiple agents operating independently equipped with a synchronised database of prices.

The main concern with parallel execution is synchronization, bidding agents may bid without obtaining the latest price data which may result in non-optimal results at termination. These problems considered more technically in Bertsekas' other papers [3], [4].

C. Performance Measurement

In the assignment problem one of the most popular methods is the Khun-Munkres (Hungarian) Algorithm. In our simulations we used the same benefit matrix in both Hungarian and Auction algorithms to compare two algorithms in both optimality of the solution and execution time.

III. Data and Results

We compared MATLAB implementations of both Hungarian and Auction algorithms [5], [6]. Auction Algorithm was implemented without parallel execution. Execution times can be seen in Figure 1.

Both algorithms yielded optimal results, no differences were observed. Optimality of results were checked using online tools [7].

IV. Discussions

Empirical data suggests that the Auction Algorithm without parallel execution lags behind the Hungarian Algorithm in terms of execution speed, however with parallel execution it is possible to obtain better results from the Auction Algorithm. It is safe to say that for multi-core CPUs or multiple computers

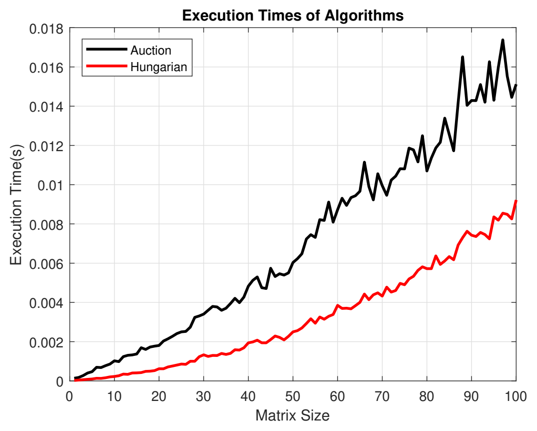

Fig. 1. Benefit matrix sizes vs Execution Time graph of the Hungarian Algorithm implemented by Y. Cao [5] and Auction Algorithm [6]

connected over a network would be better off using the Auction Algorithm. On the other hand, for single core processors the winner is the Hungarian Algorithm.

V. Conclusion

In this paper we described a widely used auction algorithm and compared it computational properties with famous Hungarian algorithm. The auction algorithm can be used in highly parallelised systems with increased speed.

Acknowledgments

This paper was written as an academic writing exercise. We would like to thank Assoc. Prof Emre Özkan and the Sensor Fusion METU team for this opportunity.

References

[1] D. Bertsekas, "A distributed algorithm for the assignment problem." [Online]. Available: https://www.mit.edu/~dimitrib/Orig_Auction.pdf

[2] D. Bertsekas, "Auction algorithms." [Online]. Available: http://www.mit.edu/~dimitrib/Auction_Encycl.pdf

[3] D. Bertsekas and J. Tsitsiklis, "Convergence rate and termination of asynchronous iterative algorithms." [Online]. Available: http://www.mit.edu/~jnt/Papers/C-89-rate_termination-Crete.pdf

[4] P. Tseng, D. Bertsekas, and J. Tsitsiklis, "Partially asynchronous, parallel algorithms for network flow and other problems." [Online]. Available: http://www.mit.edu/~jnt/Papers/J028-90-part-asyn.pdf

[5] Y. Cao. (2020) Hungarian algorithm for linear assignment problems (v2.3). Retrieved July 16 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/20652-hungarian-algorithm-for-linear-assignment-problems-v2-3

[6] A. Balcı and K. Şahin. (2020) An implementation of auction algorithm in matlab. Retrieved July 16 2020. [Online]. Available: https://github.com/aeblc/auction

[7] K. Conmuan, "Assignment problem calculator." [Online]. Available: https://comnuan.com/cmnn03/cmnn03005/