

Deep Greedy Switching: A Fast and Simple Approach For Linear Assignment Problems

Amgad Naiem^{*,†} and Mohammed El-Beltagy^{*,†}

^{*}Peerialism AB, Sweden

[†]Cairo University, Egypt

amgad@peerialism.com, mohammed@computer.org

Abstract. We present a novel approach for the linear assignment problem. The algorithm works by switching jobs to agents whilst considering the impact that the switch will have upon the overall objective. The algorithm itself is relatively easy to implement and is amenable to parallelization. Computational results on benchmark instances show that we attain solutions that slightly deviate from the best known solution and that it is at least an order of magnitude faster than other methods. The approach is best suited for solving assignment problems under tight time constraints as is the case in p2p applications, it sacrifices very little in terms of optimality for a massive speedup.

Keywords: assignment, linear assignment problems, heuristic, auction, deep greedy switching

PACS: 02.60.Pn

1. INTRODUCTION

When dealing with hard combinatorial optimization problems under tight time constraints, heuristics are often the only recourse. The goal here is to find a good enough solution within a given time limit, rather than seeking true optimality. In this paper we propose a deep greedy switching (DGS) heuristic approach to solve the *linear sum assignment problem* (LSAP) in which we want to minimize the cost of assigning of n jobs to n agents.

A large number of algorithms has been developed for the LSAP. Some these methods are: primal-dual algorithms (e.g. auction algorithm by Bertsekas [1]); interior point algorithms (e.g. approximate dual projective algorithm by Ramakrishnan et al. [2]); and algorithms using forest construction such as Achatz et al. [3]

Although an exact algorithm such the auction algorithm is efficient in solving the assignment problem, in large and complex instances of certain types the performance of the auction algorithm can be very slow. Ramakrishnan et al. [2] have discussed five types of problem instances (Geometric, Fixed-Cost proportional to i, j indices, High-Cost, Low-Cost, and Uniformly Random) where it was shown that for the first two types performance is really slow compared to the other types. Running time can reach hours versus a couple of seconds on other problem types of the same number of agents and jobs.

Moreover, in a highly dynamic system where it is necessary to match more than ten thousand agents to jobs in less than a minute in a live streaming peer-2-peer system, even the auction algorithm does not suffice. In this application optimally matching peers to one another generates many large instances of the assignment problem. If the algorithm used to find the optimal solution for the assignment problem takes too long, then the outcome would be useless as the peer-2-peer network probably would have changed by the time the optimal solution is found. Our experience has shown that the auction algorithm would be impractical and hence we developed the DGS to target such requirement with the sacrifice of less than 0.6% of optimality as will be shown in section 4.1

A plethora greedy heuristics have been developed for different optimization problems. Trick [4] employed such approach for the generalized assignment problem, where it was shown that some randomization to the greedy approach is better than a deep greedy approach. Our approach can be seen as simplified instance of the greedy randomized adaptive search procedure (GRASP) [5, 6], where we only employ randomization during initialization and proceed with local search only once, i.e. no multi-start. Also our approach is developed to best fit the assignment problem with a specially designed neighborhood construction. The fact that we obtained impressively good results with our simple procedure may suggest something about the nature the search space of the linear assignment problem.

Our approach starts with an initial solution and then iteratively seeks a better solution within a well defined neighborhood. Our greedy switching considers a switch as valid if it is strictly better than the current solution, in terms of our metric that is in direct correlation with the problem's objective function. It is greedy in a sense that it

always applies the switch that yields the largest impact on the objective function in the constructed neighborhood.

Extensive comparisons between our DGS and other existing algorithms have been conducted using known benchmark instances. Results shows that our solutions slightly deviates from the best known solutions but are an order of magnitude faster than any of the known algorithms.

2. DEEP GREEDY SWITCHING FOR LSAP

Given n jobs $J = \{1, \dots, n\}$ and n agents $I = \{1, \dots, n\}$, we want to find a minimum cost of assigning each job to exactly one agent and each agent to exactly one job. Assigning job j to agent i incurs a cost of c_{ij} . Finding the minimum cost assignment is equivalent to finding the maximum happiness assignment if we assume happiness defined as $a_{ij} = -c_{ij}$, $\forall i \in I, j \in J$. In our context we will assume we want to maximize the total happiness as this is how the DGS approach was initially developed, considering that any cost minimization assignment problem can be easily converted to the maximization problem version. Then the LSAP can be formulated as follows

$$\begin{aligned} & \text{maximize} && \sum_{i \in I} \sum_{j \in J} a_{ij} x_{ij} \\ & \text{subject to} && \sum_{i \in I} x_{ij} = 1, \quad \forall j \in J \\ & && \sum_{j \in J} x_{ij} = 1, \quad \forall i \in I \\ & && x_{ij} \in \{0, 1\} \quad \forall i \in I, \text{ and } \forall j \in J \end{aligned}$$

An assignment mapping $\sigma : J \rightarrow I$ is described as $\sigma(j) = i$ and means that job j is assigned to agent i , similarly we can define an assignment as mapping $\tau : I \rightarrow J$ where $\tau(i) = j$ would mean that agent i is assigned to job j . τ can be constructed from σ by a mapping function $\tau = M(\sigma)$. The objective function value for a given assignment σ is $f(\sigma)$. We start our approach by finding an initial feasible solution. We found DGS to quite robust with regards to the starting initial solution, optimal assignments obtained from different initial solutions have very small deviation from each other as will be shown in the results in section 4.1

2.1. Local Search and Neighborhood Construction

Starting from a random initial solution σ we construct a restricted 2-exchange neighborhood, $NA = \{NA_1, NA_2, \dots, NA_n\}$, with respect to agents. We also construct a 2-exchange neighborhood NJ , for jobs. For each agent i , NA_i would contain a valid assignment iff there exists a job j' , such that $j' \neq \tau(i)$, that carrying out a 2-exchange between j' and $\tau(i)$ would yied an improvement in the objective value, otherwise it will be \emptyset . This is done by the agent difference evaluation (ADE) algorithm. We define $f(\emptyset) = -\infty$.

Similarly, for each job j we define NJ_j to be a valid assignment for 2-exchange involving agents that could be switched with $\sigma(j)$. This is done by the job difference evaluation (JDE) algorithm, which we omit here for brevity.

ALGORITHM DGS (f, σ)**repeat** $\sigma_{start} \leftarrow \sigma, \tau = M(\sigma)$ $ADE(i, f, \tau, \sigma, NA) \quad \forall i \in I$ $JDE(j, f, \tau, \sigma, NJ) \quad \forall j \in J$ **while** $\exists NA_i \neq \emptyset \vee \exists NJ_j \neq \emptyset$ **do** $i^* \leftarrow \arg \max_{i=1..n} f(NA_i), j^* \leftarrow \arg \max_{j=1..n} f(NJ_j)$ **if** $f(NA_{i^*}) > f(NJ_{j^*})$ **then** $\sigma' \leftarrow NA_{i^*}$ $agents \leftarrow \{i^*, \sigma'(\tau(i^*))\}, jobs \leftarrow \{\sigma(i^*), \sigma'(i^*)\}$ **else** $\sigma' \leftarrow NJ_{j^*}, \tau' \leftarrow M(\sigma')$ $agents \leftarrow \{\tau(j^*), \tau'(j^*)\}, jobs \leftarrow \{j^*, \tau(\sigma'(j^*))\}$ **if** $f(\sigma') > f(\sigma)$ **then** $\sigma \leftarrow \sigma', \tau = M(\sigma)$ $ADE(i, f, \tau, \sigma, NA) \quad \forall i \in agents$ $JDE(j, f, \tau, \sigma, NJ) \quad \forall j \in jobs$ **until** $f(\sigma_{start}) = f(\sigma')$;output σ' **ALGORITHM ADE** (i, f, τ, σ, NA) $j \leftarrow \tau(i), \sigma_i^* \leftarrow \sigma, NA_i \leftarrow \emptyset$ **foreach** $j' \in \{J \mid j' \neq j\}$ **do** $i' \leftarrow \sigma(j')$ $\sigma_i' \leftarrow \sigma, \sigma_i'(j) = i', \sigma_i'(j') = i$ **if** $f(\sigma_i') > f(\sigma_i^*)$ **then** $\sigma_i^* \leftarrow \sigma_i'$ **if** $\sigma_i^* \neq \sigma$ **then** $NA_i \leftarrow \sigma_i^*$

After completing the neighborhood construction, we choose σ' that has the best objective value from NA and NJ . This is called the *greedy switch*. This procedure is repeated using σ' as the new solution until the neighborhoods NA and NJ contain no valid assignments, which would mean that no further improvements can be done.

DGS works by attempting to improve the initial solution by making 2-exchanges for agents and jobs that yield the highest positive impact on the objective function. At any point in time the maximum number of valid 2-exchanges around the σ_{start} that is being explored in DGS heuristics is $2n$, where $\forall i \in I NA_i \neq \emptyset \wedge \forall j \in J NJ_j \neq \emptyset$. As can be seen from the DGS algorithm, the stopping criteria for not repeating the neighborhood construction and the local search is when no changes in the solution occurred during the local search.

What would appear to be a very standard simple local search, produces near optimal results for a fraction of the computational effort. However, the algorithm is unique in that we consider 2-exchange neighborhood from the agents as well as the jobs permutations. This, in combination with only selecting the best 2-exchanges for each agent and job, yields a large computational speedup.

One of the key properties of the DGS approach is that the size of the valid assignment in the neighborhoods NA and NJ of σ_{start} is always decreasing. It can be shown that at least one of the jobs and agents involved in the 2-exchange process will have no further valid assignment (i.e. $\exists NA_i = \emptyset \vee \exists NJ_j = \emptyset \quad \forall i \in agents, \forall j \in jobs$), and hence any local search around σ_{start} will at most involve $2n$ switches, and in practice it is much less. This property allows our approach to finish rapidly for big and complex instances of the LSAP.

3. ASSIGNMENT PROBLEMS WITH PARTIAL CONNECTIVITY

Another way of representing the assignment problem is as a bipartite graph $G = (V, W; E)$, where vertex sets V and W have n vertices that represent agents and jobs respectively and E is the set of edges connecting any agent $i \in V$ to job $j \in W$. According to such definition we can associate each edge $(i, j) \in E$ to variable x_{ij} that is the decision variable in the LSAP. This is called a complete bipartite graph as any agent is allowed to be assigned to any object with no restrictions.

However, in many applications the problem is described as an incomplete bipartite graph where some edges $E' \in E$ are missing from the graph G or in other words some assignments are not possible. Mills-Tettey et al. [7] suggested that missing edges (i, j) may have a large cost value or in our case an a_{ij} value of $-\infty$ as we deal with maximization problem in our context. However, as we are using a heuristic approach, we may end up having a solution that contains an inapplicable assignments to the problem at hand. Also we need our approach to be able to determine if the current problem with the missing edges is feasible or not.

In order to determine feasibility, we deal with the problem as if such edges do not exist at all. This leaves us with challenging problem of finding an initial feasible solution, if any exist. Let J_i be the set of jobs that agent i is allowed to connect to. We find an initial feasible solution by iterating over all agents sequentially and trying to find an unassigned job for agent i from the set J_i . If no unassigned job can be found, we choose an assigned job j based on best difference

in the objective value that a switch between current agent i and the agent assigned to job j , $\sigma(j)$, have. When a switch is made between agent i and $\sigma(j)$, agent $\sigma(j)$ will be kicked out and become unassigned and this switch is recorded in agent i 's switch history so that this switch will not be fired again if for any reason agent i become unassigned (another agent kicked him). If for any agent the switch history becomes full (i.e. contains all possible switches), we deduce that a feasible solution for the given problem is unlikely. We then clear the switch history of that agent, and if it becomes full once more, we declare that a feasible assignment can not be found.

After an initial feasible solution is found, the neighborhood construction and the local search of the DGS approach is applied using the ADE / JDE as described earlier while taking care not to make a switch to a non-existing or missing edge and having the same stopping criteria for the DGS algorithm as is the LSAP.

4. COMPUTATIONAL RESULTS

4.1. Fully Connected Problems

We present here computational results for experiments comparing DGS performance with the best known results on benchmark problems as with the auction algorithm by Bertsekas [1] on some randomly generated problem instances. The auction algorithm is well known to be the fastest algorithm for solving the LSAP while guaranteeing near optimal solutions. All the experiments conducted throughout this paper were made on a macbook pro with a 2.66 GHz Intel Core 2 Duo processor and 4GB of memory.

The instances used in our tests are of three types. The first category is composed of benchmark instances used by Beasley [8, 9], and the other two types are RAND and GEOM used by Bus and Tvrdík [10] and dense instances of those types are generated as follows:

RAND: Each a_{ij} value is a random integer value between 1 and C , where C is a constant input parameter.

GEOM: First we generate n points randomly in a 2D graph square of dimensions $[0, C] \times [0, C]$. Then each a_{ij} value is set as the Euclidean distance between points i and j from the n generated points.

For the benchmark problems, from 100 to 800 agents and jobs, our approach has a deviation of 0.6% at its worst performance, and an average deviation of 0.5% from the exact optimal solution which shows that the quality of the solution is not affected by the initial solution used. The results also shows no significant difference in terms of running time on such small problems.

When comparing the results of our approach in terms of optimality, we used the known optimal solutions for the benchmark problems that are obtained using exact methods which are available at the OR-Library [9]. We also have used those results to verify the results of our implementation of the auction algorithm which came within 0.1% from the exact optimal solution.

The RAND generated instances have $C = 100$ and the number of agents / jobs (n) from 5,000 to 25,000 with increments of 5,000. The results shows that our approach has a deviation no more than 0.003% from the ϵ -optimal solution of the auction algorithm. These results was among five runs of the DGS approach and a comparison between the average running time of the DGS approach and the running time of our implementation of the auction algorithm is shown in Fig. 1(a).

The GEOM instances was generated with $C = 300$ and with n from 1,000 to 10,000 with increments of 1,000. The deviation of the results between the DGS solution and the auction solution was 0.058% at the worst case and 0.056% on the average among five runs. In the GEOM instances, which are considered a hard problems to find an optimal solution for, the auction algorithm running time increased exponentially while our runs at the same speed as in the RAND scenarios as can be shown in Fig. 1(b). This demonstrates the stability of the DGS performance under different kinds of problems with a very moderate sacrifice in optimality.

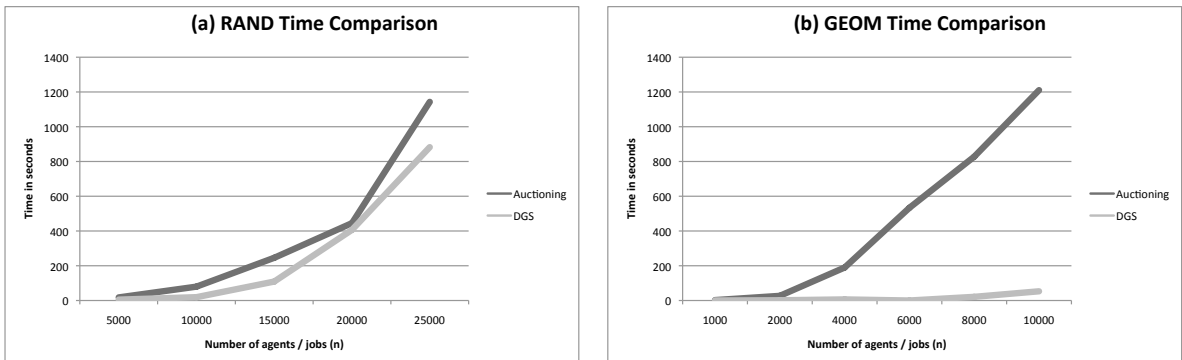


FIGURE 1. Comparison of the running time of the auction algorithm and the DGS approach on the RAND and GEOM problem instances

The time results obtained in Fig. 1 is for our implementation of the auction algorithm which has the same behavior as in [10, 2] although the times are not identical as they have been conducted on different machines but they do have the same behavior for both the RAND and GEOM instances which shows the validity of the results obtained to produce this figure

4.2. Partial Connectivity Assignment Problems

The instances used in the tests of partial connectivity are categorized into two types. The first type is some benchmark instances used by Beasley [8], and the other type is gathered from Peerialism's peer-to-peer application. The properties of the instances gathered from the p2p application is that it has a high connectivity constraints and the a_{ij} values are so close to each which is considered a very hard instances to solve.

For the benchmark problems, from 800 to 5000 agents and jobs, the results of the DGS approach has an average deviation of 2.5% from the exact optimal solution obtained from [9]. The results also shows that our approach is more than 4 times faster than the auction algorithm. Notice that the auction algorithm ϵ -optimal solution has a deviation of 1.2% from the exact optimal solution.

For the p2p application instances, we used MyP2PWorld a reproducible emulation tool for p2p systems by Roverso et al. [11] to produce instances similar to the real p2p application instances. The results on such instances shows that the DGS approach achieves better results than the auction algorithm by 0.3% as an average of 50 different runs noticing that the a_{ij} matrix values are not integers. And in terms of computational time Fig. 2 shows that the DGS approach is faster than the auction algorithm by one or two orders of magnitude. The results shown in this graph is an average of 50 p2p optimization instances.

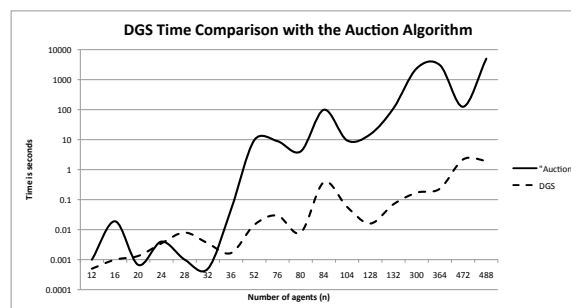


FIGURE 2. Computational time comparison between the DGS approach and the auction algorithm for partial connectivity LSAP from MyP2PWorld emulations

REFERENCES

1. D. Bertsekas, *Annals of Operations Research* **14**, 105–123 (1988).
2. K. Ramakrishnan, N. Karmarkar, and A. Kamath, *Network Flows and Matching: First DIMACS Implementation Challenge* **12**, 431–451 (1993).
3. H. Achatz, P. Kleinschmidt, and K. Paparrizos, *A dual forest algorithm for the assignment problem*, Fak. für Mathematik und Informatik, Univ. Passau, 1990.
4. M. Trick, *Naval Research Logistics* **39**, 137–151 (1992).
5. T. Feo, and M. Resende, *Journal of Global Optimization* **6**, 109–133 (1995).
6. M. Resende, and C. Ribeiro, *International Series in Operations Research and Management Science* pp. 219–250 (2003).
7. G. Mills-Tettey, A. Stentz, and M. Dias, *The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs* (2007).
8. J. Beasley, *Journal of the Operational Research Society* pp. 133–139 (1990).
9. J. E. Beasley, OR-Library (2005), <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/assigninfo.html>.
10. L. Bus, and P. Tvrdik, “Distributed Memory Auction Algorithms for the Linear Assignment Problem,” in *Proceedings of 14th IASTED International Conference of Parallel and Distributed Computing and Systems*, 2002, pp. 137–142.
11. R. Roverso, M. Al-Aggan, A. Naiem, A. Dahlstrom, S. El-Ansary, M. El-Beltagy, and S. Haridi, *Decentralized Self Management for Grids, P2P, and User Communities* p. 92 (2009).