ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Том 106

ANNUAL OF SOFIA UNIVERSITY „ST. KLIMENT OHRIDSKI“

FACULTY OF MATHEMATICS AND INFORMATICS
Volume 106

NEURAL NETWORKS FOR FACILITY LOCATION PROBLEMS

VLADISLAV HARALAMPIEV

This paper presents a new self-organizing neural network approach for solving graph-based facility location problems. It is designed to have small amount of parameters and to not need much tuning. We test our approach on several groups of problems and show that it consistently finds good feasible solutions.

Keywords: Neural networks, facility location, combinatorial optimization.

2010 Math. Subject Classification: 62M45, 90C27.

1. INTRODUCTION

Facility location problems are a large class of optimization problems, modelling the search for optimal placement of facilities to minimize costs. Many of these problems are known to be NP-hard to solve exactly. In this paper, we investigate the possibility to use neural networks for solving two graph variants of facility location problems.

There are two main neural approaches for solving difficult combinatorial optimization problems — Hopfield networks [7] and variations of Kohonen's Self-Organizing Feature Map [8]. Unfortunately, both of these approaches have problems. Hopfield's method has a tendency to settle in poor local minima, often not representing a feasible solution, and it is difficult to select appropriate parameters leading to a good solution. The problems, associated with Hopfield's approach, are well documented [11]. Vast majority of Self-Organizing Feature Maps, on the other hand, are based on the Elastic Net method [4]. This method relies on the fact that the 'elastic band' moves in Euclidean space, and distances between vertices are measured in the same space. This greatly limits the set of problems suitable for the method. In fact, most of the applications of Self-Organizing Feature Maps are to the Travelling Salesman Problem [5].

We propose a new neural network architecture for graph variants of facility location problems, inspired by the self-organizing approach to optimization. The network is designed to always find a feasible solution and to have small amount of parameters. It is often believed that involved mathematical instruments are more powerful than any heuristics, based on physical or biological analogies (see the preface in [1]). Our goal is not to outperform methods, designed for solving specific facility location problems, but to develop a robust neural network approach for facility location that needs minimal work with the specifics of the problem. This is important in practice, when we need to find acceptable solution with small investment. The proposed neural network is tested on several groups of problems and achieves good results, most of the time exactly solving the input instances.

2. GRAPH-BASED FACILITY LOCATION PROBLEMS

Let $G(V, E)$ be a connected, undirected and weighted graph with vertex set V and edge set E. We will denote the distance between two vertices $u, w \in V$ as $dist(u, w)$. The two facility location problems we are interested in are called $MiniSum$ and $MiniMax$. Good survey of the types of facility location problems is [3]. Intuitively, $MiniSum$ models the placement of several warehouse facilities, where the goal is to minimize the average travel distance. $MiniMax$ models the placement of fire stations, in which case we want to minimize the maximum time of travel to every vertex.

Definition 1 ($p - MiniSum$ problem)**.** Find a subset $u_1, u_2, \ldots, u_p$ of p vertices from V that minimizes $\sum_{w \in V} \min_{i \in \{1,\ldots,p\}} dist(w, u_i)$.

Definition 2 ($p - MiniMax$ problem)**.** Find a subset $u_1, u_2, \ldots, u_p$ of p vertices from V that minimizes $\max_{w \in V} \min_{i \in \{1,\ldots,p\}} dist(w, u_i)$.

The neural network we develop will only solve the $p - MiniSum$ problem. The following reduction is used for solving $p - MiniMax$.

Theorem 1 ($p - MiniMax$ to $p - MiniSum$ reduction)**.** *Solving a $p - MiniMax$ problem with any required positive precision ε can be reduced to solving a sequence of $p - MiniSum$ problems.*

Proof. Assume we need to solve a $p - MiniMax$ problem with input graph $G(V, E)$ and the optimal solution has value *opt*. For a given value c we can check if $opt \leq c$ by solving a $p - MiniSum$ problem in a modified version G' of G. G' is a fully connected graph with the same vertex set V. For every pair of vertices $u \neq w$ the weight in G' of the edge between u and w is one if $dist(u, w) \leq c$ in G, otherwise it is ten. Now, $opt \leq c \iff$ the optimal solution to the $p - MiniSum$ problem in G' has value $n - p$. This is because if $opt \leq c$, there is a solution in G' that only uses edges of weight one (and vice versa).

To solve the original $p - MiniMax$ problem, we can do a binary search on the value c. This way, we reduced the problem to $O(lg(MAX) + |lg(\varepsilon)|)$ instances of $p - MiniSum$, where MAX is the maximum distance between two vertices in G.□

Note that the reduction assumes all instances of $p - MiniSum$ are solved correctly. Our neural approach provides only approximate solutions to $p - MiniSum$ problems, so the solution we get for $p - MiniMax$ is also approximate. Intuitively, errors early in the sequence of $p - MiniSum$ problems directly lead to a poor quality solution for the $p - MiniMax$ problem. But, early in the sequence, the value opt from the reduction is far from c, which makes much simpler the corresponding $p - MiniSum$ problem. Another difficulty arises from the way we define distances in G'. The distances between vertices in this graph do not change smoothly, making it harder for local search methods to find good solutions.

3. NETWORK ARCHITECTURE

The architecture of the proposed network is shown in Figure 1. There are three layers, which we call A, B and C.

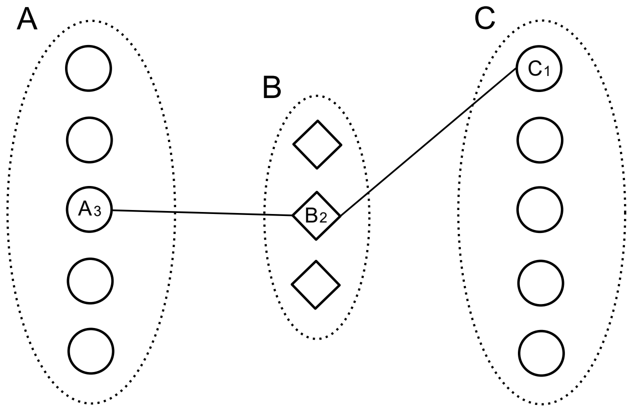

Figure 1. Neural network architecture for facility location problems

Layer A contains $|V|$ nodes, corresponding to the clients (vertices of the graph G). Layer B contains p nodes, corresponding to the facilities we need to locate. Layer C contains $|V|$ nodes, corresponding to the locations of the facilities (vertices of G). Layers $A - B$ and $B - C$ are fully connected. When we talk about outgoing edges, we assume edges are directed from A to B and from B to C. The weight of the edge between A_3 and B_2 (a number between zero and one) shows to what degree client A_3 uses facility B_2. The weight of the $B_2 - C_1$ edge shows to what degree facility B_2 is located in C_1. The interpretation of the other edges is analogous. For each node in layers A and B, we require that the sum of the weights of the outgoing edges is one. These weights are initialized randomly. As the algorithm progresses, for each node one of the outgoing edges starts to dominate and its weight approaches one. To produce the final solution from the network, we assign clients to facilities and facilities to locations by following the dominating edges. As a side note, sometimes, because of symmetries in the graph, several edges start to dominate for a node (their weights become comparable and much larger than the weights of the other edges). We observed that in such situations choosing each one of these edges produces equally good solution. In our experiments, when multiple edges dominate for a node, we always pick the edge with the smallest index.

4. OPTIMIZATION

The neural network minimizes the function

$$\sum_{a_i \in A} \sum_{b_j \in B} \sum_{c_k \in C} weight(a_i, b_j) \cdot weight(b_j, c_k) \cdot dist(a_i, c_k) \tag{1}$$

Here $weight(a_i, b_j)$ represents to what degree client a_i uses facility b_j, $weight(b_j, c_k)$ represents to what degree facility b_j is located in c_k, and $dist(a_i, c_k)$ is the distance in G between the vertices, corresponding to a_i and c_k. As for each node one of the outgoing edges starts to dominate and its weight approaches one, this function becomes equivalent to the $MiniSum$ function.

The optimization starts from a randomly initialized state and consists of a series of iterations, until the weights converge. In each iteration we go through all the nodes in layers A and B in random order and update the weights of their outgoing edges. Assume we are processing node $a_1 \in A$. The update consists of three steps:

- *Evaluate.* For each facility $b_j \in B$ calculate the cost of assigning a_1 to it, $cost_j = \sum_{c_k \in C} weight(b_j, c_k) \cdot dist(a_1, c_k)$. After this compute the value $prefer_j = \frac{min_{b_s \in B} cost_s}{cost_j}$ which is between zero and one. Intuitively, values closer to one are more preferable for the client.

- *Strengthen.* Differences between the $prefer$ values are often small. We increase them by settings $prefer'_j = \frac{e^{mult \cdot prefer_j}}{max_{b_s \in B} e^{mult \cdot prefer_s}}$. Here $mult$ is a parameter.

- *Update.* First transform the weights of the outgoing edges to have the same meaning as the $prefer'$ values. This is achieved by setting $weight(a_1, b_j)$ to $\frac{weight(a_1, b_j)}{max_{b_s \in B} weight(a_1, b_s)}$. Then update each $weight(a_1, b_j)$ to be equal to

$(1-\alpha)\cdot weight(a_1, b_j) + \alpha \cdot prefer'_j$. Finally, normalize the weights so that they sum to one (by dividing each weight by the sum of all weights). α is a parameter, analogous to learning rate in the learning algorithms of classical neural networks.

The updates are done similarly for all other nodes in layers A and B. There are two parameters, the learning rate α and the $mult$ parameter that scales the $prefer$ values. Exponential grid search is used to select the parameters. More specifically, the $mult$ dimension of the grid consists of the values 1.2^x for $x \in 1, 2, \ldots, 50$. The α dimension consists of $0.2 \cdot 0.8^y$ for $y \in 0, 1, 2, 3, 4$. For each cell of the grid we run the optimization with the corresponding parameters. We then pick the best solution found. To guarantee convergence in the allocated time, during each optimization run the learning rate decreases exponentially with the number of iterations. From our experience, the initial value of the learning rate affects the speed of convergence, but does not affect significantly the quality of the final solution (assuming the optimization runs long enough). On the other hand, $mult$ affects the quality of the solution.

5. TEST PROBLEMS AND RESULTS

The proposed network is tested on four groups of problems:

- *Unweighted trees (TU).* Random trees with 50 to 100 nodes and p (number of facilities) between two and six. All edges are of length one. The random trees are generated using Prüfer's code, a mapping of trees to number sequences [10].
- *Weighted trees (TW).* Trees with the same parameters as the unweighted trees above, but with random floating point edge lengths between 1 and 100.
- *Chordal graphs (CH).* Chordal graphs are graphs without induced s-cycles for s more than three [2]. They have more complex structure than trees, but still are simple enough to allow efficient algorithms for many problems that are hard in general graphs. We generate chordal graphs with 50 to 100 vertices and set p (number of facilities) to a value between three and six. To generate them we use two methods — producing a perfect elimination order and using the equivalence to intersections of subtrees of a tree [6].
- *Bulgarian road network (BR).* For various geographic regions in Bulgaria we take the populated places and the roads connecting them. We choose regions with 60 to 400 populated places and set p (number of facilities) to a value between two and four. Data about populated places and roads is taken from OpenStreetMaps [9] (using Overpass queries).

For each of the first three groups, we randomly generate 200 graphs. For the last group, we generate 30 graphs.

For each of the instances above we compute the optimal answer by trying all possibilities of locating the facilities. Constraints in the instances are chosen small enough so that this computation runs in reasonable time. We also run a local search for each instance to compare its results with the results of the proposed neural network. Local search [1] is a classical general method for solving optimization problems that often gives very good results. It starts from a random solution and repeatedly tries to improve it by choosing a better solution from some neighbourhood of the current one. In our case, the neighbourhood is defined by changing the location of a facility, or by changing the facility that services a client. Since the results of local search depend on the initial solution, for each instance we run 1000 independent local searches and pick the best value they return.

The results for the $MiniSum$ problem are presented in Table 1. Both methods have excellent performance, most of the time finding an optimal solution. Local search has slightly better performance, probably because it runs 1000 independent searches.

Table 1. Results of local search and the proposed neural network for the $MiniSum$ problem. *Max* is the maximum error over all inputs (as percentage from the optimal answer), *Avg* is the average error, and *Exact* is the percentage of inputs, solved exactly.

	Local search			Neural network		
	Max, %	*Avg*, %	*Exact*, %	*Max*, %	*Avg*, %	*Exact*, %
TU	0.312	0.009	97	0.295	0.007	98
TW	1.130	0.030	92	2.000	0.090	90
CH	3.191	0.072	98	4.000	0.500	80
BR	0.021	0.012	96	0.000	0.000	100

The results for the $MiniMax$ problem are presented in Table 2. Our approach to $MiniMax$ requires solving a sequence of harder $MiniSum$ instances, so, as expected, the results are worse than the ones for $MiniSum$. The neural network approach performs significantly better than local search. It is often able to exactly solve the instance. As a side note, the excellent results on chordal graphs (CH) are probably because they, intuitively, are a sequence of attached cliques, which makes $MiniMax$ simpler to solve.

Table 2. Results of local search and the proposed neural network for the $MiniMax$ problem. *Max* is the maximum error over all inputs (as percentage from the optimal answer), *Avg* is the average error, and *Exact* is the percentage of inputs, solved exactly.

	Local search			Neural network		
	Max, %	*Avg*, %	*Exact*, %	*Max*, %	*Avg*, %	*Exact*, %
TU	32.900	9.770	60	5.080	0.827	87
TW	31.640	7.180	47	3.042	0.253	87
CH	0.000	0.000	100	0.000	0.000	100
BR	36.150	16.210	0	10.000	3.090	67

6. CONCLUSION

We presented a new neural network architecture for solving graph-based facility location problems and evaluated its performance on several groups of $MiniSum$ and $MiniMax$ problems. Our method is based on the self-organizing approach to optimization and shows good performance. On simpler instances its results are comparable to local search, and it significantly outperforms local search on harder instances.

ACKNOWLEDGEMENT. Map data used for generating the facility location instances based on the Bulgarian road network is copyrighted by OpenStreetMap contributors and can be found at https://www.openstreetmap.org

7. REFERENCES

[1] Aarts, E., Lenstra, J. K.: *Local Search in Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, 1997.

[2] Blair, J. R. S., Peyton, B.: *An Introduction to Chordal Graphs and Clique Trees. Graph Theory and Sparse Matrix Computation.* IMA Volumes in Mathematics and its Applications, vol. 56, 1993.

[3] Cappanera, P.: A Survey on Obnoxious Facility Location Problems. University of Pisa, technical report, 1999.

[4] Durbin, R., Willshaw, D.: An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, **326**, 1987, 689–691.

[5] Favata, F., Walker, R.: A study of the application of Kohonen-type neural networks to the Travelling Salesman Problem. *Biological Cybernetics*, **64**, 1991, 463–468.

[6] Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combin. Theory Ser. B*, **16**, 1974, 47–56.

[7] Hopfield, J. J., Tank, D. W.: "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, **52**, 1985, 141–152.

[8] Kohonen, T.: Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, **43**, 1982, 59–69.

[9] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org. https://www.openstreetmap.org (2017)

[10] Prüfer, H.: Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik*, **27**, 1918, 142–144.

[11] Wilson, G. V., Pawley, G. S.: On the stability of the Travelling Salesman Problem algorithm of Hopfield and Tank. *Biological Cybernetics*, **58**, 1988, 63–70.

Received on April 18, 2019
Received in a revised form on February 29, 2020

Vladislav Haralampiev

Faculty of Mathematics and Informatics
"St. Kliment Ohridski" University of Sofia
5 blvd. J. Bourchier, BG-1164 Sofia
BULGARIA

E-mail: vladislav.haralampiev@gmail.com