Neural network approaches for a facility location problem

Vladislav Haralampiev
Faculty of Mathematics and Informatics – Sofia University “St. Kliment Ohridski”, Bulgaria
vladislav.haralampiev@gmail.com

Abstract: This paper examines the possibility to use neural networks for approximately solving the MiniSum problem, a classic facility location problem. For this we first create a set of realistic MiniSum instances, based on the Bulgarian road network. Two standard neural network approaches – Hopfield networks and Boltzmann machines, are then applied to the instances. Since the quality of solutions is not satisfactory, the reasons for the poor performance are discussed. An improved neural network approach is then proposed. This approach has excellent performance on the MiniSum instances. It always finds solutions just several percent worse than the optimum, and is often able to find the exact optimum.

Keywords: FACILITY LOCATION, NEURAL NETWORKS, COMBINATORIAL OPTIMIZATION

1. Introduction

Facility location problems are a large class of optimization problems, often occurring in practice. In this paper we are interested in the MiniSum problem, a classic facility location problem. MiniSum models the search for optimal placement of a set of warehouses with goal to minimize transportation costs. We give a formal definition of the problem in the next section.

The interest in using neural networks for approximately solving optimization problems started from the seminal paper of Hopfield and Tank [3]. They propose a certain type of network, which is able to find solutions to the Travelling salesmen problem. This network can be generalized to many combinatorial optimization problems. It is also implementable in hardware and is naturally massively-parallel, which makes the whole approach interesting for solving large instances. Sadly, the Hopfield-Tank (HT) approach has problems. As noted in [8] and in many other places, the approach does not scale to large instances and the quality of the solutions it produces is bad. There were many efforts to improve the HT approach. One promising direction are Boltzmann machines (further denoted BM), which add randomness to the network operation.

Neural networks for combinatorial optimization may be interesting from theoretical perspective, but if such approaches produce low-quality solutions, they will have no practical value. We start by generating a set of realistic MiniSum instances, based on the Bulgarian road network. We then apply the HT approach and BM to the instances and confirm that the quality of the solutions they give is not satisfactory. After analyzing the reasons for the poor performance, we propose an improved approach, which has excellent performance on the MiniSum instances. On average it finds solutions just 0.3% worse than the optimum and in 88% of the cases it finds the exact optimum. This confirms that asynchronous massively-parallel systems of “neurons” can be used to produce high-quality solutions to combinatorial optimization problems. The reasons for the poor performance on HT and BM are in their specifics, not in the idea of using such systems of “neurons”.

2. The MiniSum problem

Intuitively, we are given a directed, weighted graph. Nodes represent populated places and edges represent roads. We need to choose k nodes from the graph where to locate warehouses. The goal is to minimize the sum of distances from each populated place to the closest warehouse.

2.1 Formal definition

The k-MiniSum problem is defined over a directed, weighted and strongly connected graph $G(V, E)$ with vertex set $V$ and edge set $E$. The shortest distance in $G$ between two vertices $U$ and $W$ is denoted as $\text{dist}(U, W)$. Integer $k$ is a parameter of the problem and is usually much less than the number of vertices.

Definition (k-MiniSum problem)
Find a list of $k$ vertices $V_1, V_2, \ldots, V_k$ from $V$ which minimize
$$\sum_{V \in V_1 \cdots V_k} \text{dist}(U, V)$$

The k-MiniSum problem in general graphs is well-known to be NP-hard (by reduction, for example, from the Set cover problem [4]).

2.2 Test instances

The Bulgarian road network is used to generate realistic MiniSum instances. For each instance a rectangular area is selected. All populated places and roads in this area are extracted and used as underlying graph. We choose the areas so that there are between 20 and 90 populated places and set the number of facilities to a value between 2 and 6. The instances are small enough to be able to exactly compute the optimal solution in reasonable time. As source of geographic data OpenStreetMaps [6] is used (data is extracted using SPARQL queries). For computing the exact solution each instance is modelled as integer programming problem and solved using the Cbc mixed integer programming solver [1]. In total we have selected 50 different rectangular areas, which gives 250 MiniSum instances. The image below shows one of them.

Fig. 1 Example MiniSum instance with 4 warehouses. The dots represent populated placed and the 4 selected locations are marked with additional circular outline.

3. Existing neural network approaches

Good description of combinatorial optimization using Hopfield networks can be found in [7], and using Boltzmann machines – in [5]. The two approaches have a lot in common. In fact, Boltzmann machines can be thought of as stochastic version of Hopfield networks. Both approaches assume a binary problem is given. There is a set of $n$ binary variables $V_1, V_2, \ldots, V_n$, an $n \times n$ weight matrix $W$ and a weight vector $D$. The approaches find a local minimum of the function $\sum W_{ij} V_i V_j - \sum D_i V_i$ (called the energy function). To approximately solve a combinatorial optimization problem we need to choose a weight matrix and weight vector such that local minima of the energy function correspond to good solutions of the problem. We call this step encoding. It should be noted that there are requirements, which matrix $W$ should meet (described in [7]). Usually, when encoding a problem, the requirements are naturally met.
Clearly, there are many ways to encode the MiniSum problem. Next, one possible way, inspired by the encoding of other problems in [7] and [8], is described. We believe that the encoding is reasonable, but there is a possibility for another encoding, giving better results, to exist.

Let n be the number of vertices in G and k be the number of warehouses we need to place. The neural network encoding has 2 ⋅ n ⋅ k binary variables. We call one group of n ⋅ k variables the client – facility variables and denote them as CF[i, j]. CF[i, j] is 1 iff the client in vertex i is serviced by facility (warehouse) number j. Similarly, another group of n ⋅ k variables correspond to the facility – location relation. We denote them as FL[i, s] and FL[j, s] is 1 iff facility j is located in vertex s. The transportation cost can now be expressed as ∑∈CF[i,j] ⋅ FL[i, s] ⋅ dist(Vi, Vs), which is in a form, compatible with the energy function notation. Fig. 2 illustrates the encoding.

Fig. 2 MiniSum encoding. Note that the binary variables correspond to edges (not all edges are shown). Dashed edges stand for variables with value 0, solid edges – with value 1. Solid edges connect a client with its facility and a facility with its location.

For a solution to be valid, each client should be serviced by only one facility, and each facility should be located in only one vertex. The typical approach to achieve this 1–of–many constraint with neural networks is to add penalty terms to the energy function. Similarly to [7] and [8] we add to the energy function a group of penalty terms of the form A ⋅ (1 − ∑∈CF[i,j])^2 for each client i. When A is large enough, these terms force the network to choose exactly one facility for each client. We analogously add term of the form B ⋅ (1 − ∑∈FL[j,s])^2 for each facility j to enforce the other 1–of–many constraint. Note that, when the brackets are open, the expressions are in a form, compatible with the notion of energy function. Choosing the values A and B is not straightforward and it affects the quality of the produced solution. The need to choose weights for the penalty terms is considered to be a major problem of the HT approach. Later in this section we will describe our approach for choosing the weights.

The problem encoding tells us how to construct, given a MiniSum instance, the set of binary variables, the weight matrix and weight vector. Variables are then mapped to simple computing units (neurons). Each unit corresponds to a variable. It has an output value equal to the value of the variable. The units are also connected to each other, matrix W gives the weights of the connections. Each unit has a procedure for updating its output, the procedure is different between HT and BM. The whole system of units starts from a random state (random output for each unit) and repeatedly chooses a unit and executes its update procedure. After enough time has passed, we read the output values of the neurons and this is the final solution. Again, the encoding tells us how to translate from the values of the binary variables Vi to the objects of the initial problem. In the case of the described encoding, the variables with value 1 tell which facility services each client and where each facility is located.

### 3.1 Hopfield network updates

Assume unit S wants to perform an update. It computes the value inp[S] = ∑∈W[ S,j] ⋅ output[j]. If inp[S] < 0[S] the unit sets its output to 1, otherwise it sets the output to 0. This computation is equivalent to checking the sign of the gradient of the energy function with respect to output[S]. The unit chooses the value of its output so that the energy function does not increase.

### 3.2 Boltzmann machine updates

In BM there is a notion of temperature. The temperature T controls the probability of accepting a change of the unit's output. Assume unit S wants to perform an update. It computes D, the difference in the energy function if we flip the output of S. If D < 0, S flips its output. Otherwise it flips its output with probability \((1 + e^D)T^{-1}\).

It can be noticed that, when the temperature is high, almost any change is accepted with high probability. When T is low, only changes which decrease the energy function are accepted. In this sense, when T is low, BM operates very similarly to a HT network. Temperature T gradually decreases with time. It starts from a value high enough to allow almost any change to be accepted (with high probability). Then every M steps it is multiplied by a constant a little less than 1 (where M is chosen to be proportional to the number of units).

### 3.3 Choosing penalty weights

Penalty terms were introduced to guarantee the validity of the solution. When their weight is too small, the system can arrive at a local minimum, in which a client is not connected to any facility, or no location is chosen for a facility. If the penalty weights are too high, they will dominate the energy function and will drive the system into a state representing a valid, but low-quality solution. Our approach to choosing the weights is to try to make them as small as possible. For this we first start from small values. We then repeatedly double the weights and run the network optimization procedure, until the network finds a valid solution. After this we repeatedly try to decrease the weights by a small fraction (10% or 5%) until the system still finds valid solutions. Actually, since in our encoding there are two separate weights (denoted A and B), we alternate between them in the decreasing steps.

### 3.4 Discussion on the performance

Here by performance we mean the quality of the solution. Since in both HT and BM there is some randomness, multiple runs were performed. More specifically, the described procedure for choosing penalty weights by itself performs multiple runs with different weights. Also, multiple runs were performed with the best weights found. In total this gives several hundred runs of the optimization procedure for each instance. The best solution found during the runs is selected.

The results of both HT and BM are very discouraging. While they are always able to find valid configurations, on average they find solutions more than two times more expensive than the optimum. The results of BM are slightly better than HT, but the runtime is significantly longer. Also, the cost of the found solutions just slightly decreases when increasing the number of facilities. And the quality of the solutions is not much better than a random solution. This points that both HT and BM put emphasis on finding valid solutions and almost do not optimize for solution quality.

It can be noticed that both HT and BM are variations of local search. They operate on a set of M binary variables and find local minimum in a solution space of size \(2^M\) by starting from a random state and iteratively improving it. Local search is a well-researched area and is the base of many of the best-performing general approaches for (approximately) solving combinatorial optimization problems. Good overview of local search methods and their performance is [2]. We can borrow ideas from the analysis of local search methods to understand the problems with HT and BM, and improve their performance.

### Neighbourhood

To choose the next state, local search performs small modifications of the current. The states, reachable from the current state in one modification step, are called its neighbours. The performance of local search highly depends on the neighbourhood definition, at least because local minimum is defined with respect to
a neighbourhood. For HT and BM the neighbourhood is formed by
flipping one binary variable. This seems to be not flexible enough.
Flipping one variable changes the validity of the current solution,
which makes the penalty terms a major factor for deciding whether
to accept a solution. Once a facility is connected to a location, for
example, it becomes hard to change the location. The
neighbourhood puts more emphasis on finding a valid solution than
on finding a good solution. This may be a problem of our encoding
of MiniSum. Yet, the mechanics of HT and BM seem to not be able
to represent better neighbourhoods without excessive redundancy in
the encoding.

Solution space

Since MiniSum is a hard problem, it is expected that the
solution space is complex. But the penalty terms in the energy
function create additional problems. It was already noted that
choosing weights for the penalty terms is not straightforward. With
respect to the solution space, the penalty terms create many
(possibly poor) local minima from which it is hard to escape.

Randomization

Randomization often improves the quality of the solutions,
found by local search. One common way to add randomization is to
allow the search to occasionally accept modifications, which
decrease the quality of the solution. This is done in an effort to
escape from poor local minima. Since HT always decreases the
value of the energy function, in this sense the method is completely
deterministic. There is a historic explanation for this – Hopfield
networks were developed as associative memory, not as machinery
for optimization. Because of their determinism, we can expect HT
to perform worse than stochastic local search.

BM allows transitions, increasing the value of the energy
function. In this sense, they are better than HT. In fact, there is a
theoretical result that, given enough time, BM will find the optimal
solution (in probabilistic sense). But the time necessary for this is
larger than the time for iterating through the whole solution space,
so the result has little practical value.

Summary

As summary, the poor performance of HT and BM seems to
follow from a combination of the penalty terms as mechanism for
enforcing solution validity and the inflexible neighbourhood
definition. For the HT approach there is the additional drawback of
clean deterministic (in the sense of the previous paragraph).

4. Proposed neural network approach

The proposed model uses the same binary variables CF and FL
as in the already described MiniSum encoding. It minimizes the
function \( \sum_{i,j} CF[i,j] \cdot FL[j,s] \cdot \text{dist}(V_i,V_j) \). Note that there are no
penalty terms in this function, another mechanism will be used to
guarantee the validity of the solution. The binary variables are again
mapped to units which have outputs, equal to the value of the
responding variable. The whole system repeatedly updates the
outputs of the units, same as the HT and BM approaches.

The units in the proposed approach have two more properties.
Let say that units \( X \) corresponds to the variable \( V_k \). Then:

• \( \text{group}[X] \) is a list of units which, intuitively, compete with \( X \)
  for activation. Eventually only one of the units in the group will
  have output equal to 1 (will be on). This is the mechanism for
  enforcing solution validity.

• \( \text{con}[x] \) is a list of pairs of unit and weight (real number). This
  list represents the units to which \( X \) is connected (similarly to the
  \( W \) matrix in section 3).

The groups in the encoding naturally correspond to the 1-of-
many constraints. Each client needs to be connected to exactly one
facility, so for a \( CF[i,j] \) (client - facility) unit its group is
\( \{ CF[i,j] | t \in 1...k \} \). Here \( k \) is the number of facilities. Similarly,
for a \( FL[j,s] \) (facility – location) unit its group is \( \{ FL[j,t] | V \in V \} \).
This is because each facility needs to be located in exactly one
vertex of the graph.

Connections in the encoding are only between \( CF[i,j] \) and
\( FL[j,s] \) units. More specifically, \( CF[i,j] \) is connected to \( FL[j,s] \)
with weight \( \text{dist}(V_i,V_j) \). Similarly, \( FL[j,s] \) is connected to \( CF[i,j] \)
with the same weight. Here \( i, j \) and \( s \) iterate through all valid
values. This also has very natural interpretation – if client \( i \) is
serviced by facility \( j \) (\( CF[i,j] \)) which is located in vertex \( s \) (\( FL[j,s] \))
than we pay \( \text{dist}(V_i,V_j) \) for transportation.

As in Boltzmann machines, there is a notion of temperature \( T \),
which controls the probability of accepting transitions. This
temperature decreases exponentially and can be either local for each
unit, or global for the whole system.

If unit \( X \) wants to update its output, it first finds the set of units
which have value 1 and are connected to \( X \). Lets call this set \( ON \).
If \( ON \) is empty, \( X \) sets its output to 1. Otherwise, \( X \) computes
\( \text{value}[X] = (\sum_{e \in ON} \text{weight}[e]) / |ON| \). Here weight is the
weight of the connection to the corresponding unit. \( X \) also computes
\( \text{BEST} = \min \{ \text{value}[s] | s \in \text{group}[X] \} \). The unit sets its output
to 1 if \( \text{value}[X] < \text{BEST} \), else it sets it to 0. After this \( X \) flips its
output with probability \( 1 + e^{\frac{\text{B E S T} - \text{value}[X]}{T}}^{-1} \).

The operation of the system consists of a sequence of updates of
unit outputs (starting form a random state). Assume there are \( M \)
units in total. The sequence of updates is of size \( 5M^2 \) and is
separated into \( M \) groups. After each group the temperature \( T \) is
multiplied by a constant \( C \), which is chosen so that after the \( M \)
groups \( T \) becomes 0.001 (the initial temperature is chosen as the
maximum energy delta of a variable flip according to the initial
random state). Each group of updates consists of \( 5M \) individual
ones, in each of which we randomly choose a unit and update its
output, as we already described.

Note that, when the temperature is low, the way we process
groups guarantees the validity of the solution and the arrival at a
local minimum of the energy function. Additionally, on higher
temperatures there is a large probability to be in states, in which a
facility is connected to multiple locations, or a client is connected to
multiple facilities. Taking average in the value computation is done
to increase exploration by allowing cheap “drifting” of the chosen
location to neighbouring ones.

Since there is randomness in the system’s operation, it makes
sense to perform multiple runs. In our experiments we performed 4
independent runs for each instance and took the best solution found.
Actually, most of the time one run was enough to find the optimal
solution.

On the test MiniSum instances the described approach is able to
find the exact optimal solution in 88% of the cases and achieves
average error of 0.3%. By error we mean the difference
(percentage) between the returned solution and the optimal solution.
The maximum error on an instance is 8%, the next largest is 2%.
We haven’t tuned the parameters of the described model, so
probably it is possible to achieve slightly better results. Also, it is
worth nothing that, even on our small-sized MiniSum instances, the
proposed approach is much faster than both the integer
programming approach and BM.

5. Conclusion

We created a set of realistic MiniSum instances and proposed a
general neural network approach for facility location, which has
excellent performance on the instances. This shows that massively
parallel systems of “neurons” can find good quality solutions to
combinatorial optimization problems. Apart from being
theoretically interesting, such systems allow for efficient distributed
implementation. There are also developments in special hardware
for neural networks, which can significantly decrease the time of
the computation. As future work it is left to analyze the convergence properties of the proposed approach and to evaluate it on other facility location problems and combinatorial optimization problems with 1-of-many constraints.

References

[1] "Cbc (Coin-or branch and cut) mixed integer linear programming solver", https://github.com/coin-or/Cbc


