Visualizing Co-occurrence of Self-Optimizing Fragment Groups

Edgar H. de Graaf   Walter A. Kosters

Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
email: edegraaf@liacs.nl

Abstract

In this paper we will use and extend a type of competitive neural network to visualize the co-occurrence of subgraphs in a dataset where molecules are considered as transactions or records.

We will adapt this algorithm, called push-and-pull, to plot each subgraph as a point in 2D space by repeatedly changing its position based only on the relative distances. Instead of each fragment/subgraph being a point ($\{x, y\}$-coordinate) in our visualization, we build a group for each point by "leaking" fragments to a point linked to a more fitting group.

In this way we create a 2D visualization by knowing only the distances between a user-defined number of groups (points), improving runtime and overview. Practically it will allow us to improve the analysis of co-occurring substructures, or fragments, within the molecules of the transactions, by improving the visualization with less points (centroids) in comparison with traditional push-and-pull.

The algorithm is beneficial for any data mining task where one only knows the distances between points, because the structure of the learning examples does not clearly allow for an input vector (e.g., graphs or trees), the dimension of the input vector grows exponentially or the input vectors are simply not given.

1 Introduction

Unsupervised learning methods allow us to visualize all kinds of bio-chemical data. The motivation for this paper is the search for co-occurring substructures in sets of molecules. These substructures are basically connected subgraphs of the bigger connected graph, the *molecule*. In the context of bio-chemical data we call these subgraphs *fragments*. A visualization of co-occurrence needs to handle many fragments, we seek to improve scalability.

For a bio-chemist it is very interesting to know which fragments occur often together, for example in so-called active molecules. This is because frequent co-occurrence implies that the fragments are needed simultaneously for biological activity. Furthermore, pharmaceutical companies provide generated libraries of molecules. A visualization of co-occurrences in molecule libraries gives a bio-chemist insight how the libraries are constructed by the company.

In this paper we will use and extend a type of competitive neural network called the *push-and-pull* algorithm, as published in [9], to visualize the co-occurrence of subgraphs in a dataset where molecules are considered as transactions. The push-and-pull algorithm is related to multi-dimensional scaling for which a rich literature exists, e.g., Sammon et al. in [16], Bronstein et al. in [1], and the ISOMAP algorithm in [17].

We will adapt the push-and-pull algorithm, that plots each point in 2D space by changing its position based only on the pair-wise relative distance. A *point* in this paper is defined as $\{x, y\}$-coordinate linked to or belonging to a fragment. Instead of each fragment being a point in our visualization, now we build a group for each point. We call a point linked with a group a *centroid* since the distance between groups is decided by the average distance between members. Furthermore we make these groups *self-optimizing* by "leaking" fragments to a centroid with a more fitting group.

The goal of this research was to make the push-and-pull algorithm more scalable to the number of fragments. An added benefit is that having less points improves the overview in the visualization. With our algorithm we hope to combine the advantages of Self-Organizing Maps (see [8]), e.g., one neuron for many similar points, with the advantages of push-and-pull, e.g., good intermediate approximations and

visualization with only distance information. To this end, this paper makes the following contributions:
— We will **define the “leaking” bi-dimensional centroids** and show how they fit in the push-and-pull algorithm.
— Furthermore we will **propose an algorithm** that allows us to visualize the co-occurrence of fragments (or subgraphs).
— We will empirically show that the algorithm can rediscover **2D synthetical data** based only on their relative distance.
— Finally we will empirically show **how scalability improves** in comparison with traditional push-and-pull.

In theory we can create a 2D visualization by knowing only the distances between a user-defined number of groups (points), improving runtime and overview. Practically it will allow us to improve the analysis of co-occurring fragments within the molecules of the transactions, by improving the visualization with less points (centroids).

The algorithm is beneficial for any data mining task where one only knows the distances between points, because the structure of the learning examples does not clearly allow for an input vector (e.g., graphs or trees), the dimension of the input vector grows exponentially or the input vectors are simply not given. In these cases more traditional algorithms, e.g., K-means in [11] and EM in [2], are harder to use (since we know only the distance between instances and not their corresponding input vector).

Fast access of the distances is important for the push-and-pull algorithm as it is for any competitive neural network algorithm. The algorithm will benefit from faster on-demand distance computation, since storing all distances in memory or even on the disk becomes impractical as the number of data points grows.

The overview of the rest of the paper is as follows. In Section 2 we start with a background discussion and continue, in Section 3, with defining molecules and fragments as graphs and subgraphs and the distance measure of co-occurrence for the fragments. In Section 4 we continue with the main contribution of this paper when we define our model and the principle of leaking, introducing our extended algorithm for visualization in Section 5. Finally, in Section 6, we discuss our experimental results. We conclude in Section 7.

2 Background

This work is related to work done on competitive neural networks, more specifically involving the push-and-pull algorithm. Furthermore, the work is related to work done on the analysis of molecular datasets.

Competitive neural networks in the area of biology are important because the dimensional reduction property provides an important basis for any visualization. In general our work is related to SOMs as developed by Kohonen (see [8]), in the sense that SOMs are also used to visualize data through a distance measure. A *Self-Organizing Map* (SOM) is a type of artificial neural network that is trained to produce a low dimensional representation of the training samples. A SOM is constructed by moving the best matching point and its neighbours (within a lattice of neurons) towards the input node.

SOMs have been used in a biological context many times, for example in [6, 12]. In some cases molecules are clustered via numeric data describing each molecule; in [19] clustering such data is investigated.

Points in a SOM can not, beforehand, be linked with a single pattern or group of patterns, which makes it less suitable for our purposes. Also by design a SOM needs to know the dimension of the input vector. In our setting we do not know the input vector, but only the distance between points. One could use the distance to other points as an input vector, but the number of dimensions will potentially be huge. With a more scalable push-and-pull algorithm we will not have these disadvantages while still being able to analyse many fragments. Furthermore the algorithm for a SOM needs time before neurons get into the neighbourhood of the correct group of items. The intermediate picture of push-and-pull always approaches the situation with different levels of quality.

Furthermore, our work is related to work done on the identification of *Structure Activity Relationships* (SARs) where one relates biological activity of molecules by analyzing their chemical structure [4, 7] in the sense that in our work the structure of a graph is used to build a model. In [3, 14, 15] a statistical analysis was done on the presence of fragments in active and inactive molecules. However, our work is not concerned with the discovery of SARs, but with co-occurrence of subgraphs occurring in a collection of graphs. More related is the work done by Lameijer et al. in [10]. This work is concerned with co-occurring fragments discovered with a graph splitting. Graph splitting breaks molecules at topologically interesting points. A frequency threshold is used to filter out some fragments after their generation, however no frequent pattern mining techniques are used. Furthermore, they do not build a co-occurrence model or a similar visualization of co-occurrence. Figure 1 shows two co-occurring subgraphs (fragments) discovered by Lameijer et al. in their dataset of molecules. The algorithm presented in this paper confirms these results.

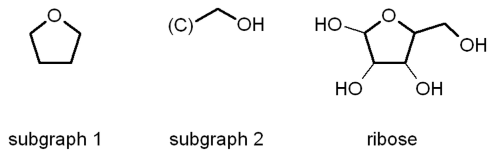

Figure 1: An example of co-occurring subgraphs from [10] with an example molecule.

3 Molecules and Fragments

First we define what a fragment and a molecule are in the context of this work. Let $G = (V, E)$ and $G' = (V', E')$ be connected graphs, where V and V' are finite, non-empty sets of vertices and E and E' are non-empty sets of edges (links between pairs of vertices). The graph G' is a *subgraph* or *fragment* of the graph or *molecule* G if $V' \subseteq V$ and $E' \subseteq E$. If G' is a subgraph of at least $minsupp$ graphs G in a dataset $\mathcal{D}$ of graphs then we call G' a *frequent subgraph*, where $minsupp$ is a user-defined threshold for frequency. Our algorithm is commonly used in the analysis of frequent fragments in order to minimize the pattern space.

The distance function for calculating the distance between two learning examples can be different for each problem, however it needs to range between 0 and 1. In the case of mining molecular datasets and analysing co-occurrence, we take the distance measure given in [5]:

$$g\_dist(g_1, g_2) = \frac{supp(g_1) + supp(g_2) - 2 \cdot supp(g_1 \wedge g_2)}{supp(g_1 \vee g_2)} \tag{1}$$

where g_1 and g_2 are two subgraphs (or fragments) and the *support* function value $supp(g)$ computes the number of occurrences of g as subgraph in the dataset of molecules. Here $supp(g_1 \vee g_2)$ counts the occurrences of one of the two graphs and $supp(g_1 \wedge g_2)$ the occurrences of both graphs. For each molecule from the dataset we count only one occurrence. If $supp(g_1 \vee g_2) = 0$ we define $g\_dist(g_1, g_2) = 1$.

This distance measure is known as the Jaccard metric and was primarily chosen for its common use in Bio-informatics (see [18]). It is also easy to compute, given the appropriate supports; it doesn't make use of complicated graph comparisons, that would slow down the process.

4 Leaking Centroids

We will visualize co-occurrence by positioning a user-defined number n of centroids in a 2-dimensional area, where the ordered set of all centroids is indicated with $\mathcal{C}$. In this work a centroid is defined as a point linked to a group of fragments instead of to one fragment. Each centroid has coordinates within this 2-dimensional area consisting of a pair of two real numbers $0 \leq x \leq 1$ and $0 \leq y \leq 1$.

At first, as done in traditional push-and-pull, centroids are placed at random in the 2-dimensional area. At each iteration a random pair of centroids i and j ($0 \leq i, j < n$) is selected and their relative position is changed with the following formulas:

$$\begin{aligned}
x_{\mathcal{C}_i} &\leftarrow x_{\mathcal{C}_i} - \alpha \cdot (eucl\_dist(\mathcal{C}_i, \mathcal{C}_j) - m\_dist(\mathcal{C}_i, \mathcal{C}_j)) \cdot (x_{\mathcal{C}_i} - x_{\mathcal{C}_j}) \\
y_{\mathcal{C}_i} &\leftarrow y_{\mathcal{C}_i} - \alpha \cdot (eucl\_dist(\mathcal{C}_i, \mathcal{C}_j) - m\_dist(\mathcal{C}_i, \mathcal{C}_j)) \cdot (y_{\mathcal{C}_i} - y_{\mathcal{C}_j}) \\
x_{\mathcal{C}_j} &\leftarrow x_{\mathcal{C}_j} + \alpha \cdot (eucl\_dist(\mathcal{C}_i, \mathcal{C}_j) - m\_dist(\mathcal{C}_i, \mathcal{C}_j)) \cdot (x_{\mathcal{C}_i} - x_{\mathcal{C}_j}) \\
y_{\mathcal{C}_j} &\leftarrow y_{\mathcal{C}_j} + \alpha \cdot (eucl\_dist(\mathcal{C}_i, \mathcal{C}_j) - m\_dist(\mathcal{C}_i, \mathcal{C}_j)) \cdot (y_{\mathcal{C}_i} - y_{\mathcal{C}_j})
\end{aligned} \tag{2}$$

Here α ($0 \leq \alpha \leq 1$) is the user-defined learning rate. It should not be chosen too big for not making too large adaptation towards the pair-wise distance and never converging to a good approximation of distances between all points. An α of around 0.1 was found to be a good choice in most cases. The function $eucl\_dist$ calculates the Euclidean distance between centroid coordinates and $m\_dist(I_{\mathcal{C}_i}, I_{\mathcal{C}_j})$ is as defined below.

This is a kind of push-and-pull algorithm which yields a visualization in which the distances in 2D correspond to distances in the pattern space. This approximation emerges due to the small adaptation of pair-wise distances. In a post processing step values are scaled to fit in coordinates ranging from 0 to 1. Note that in [9] the push-and-pull was shown to converge to an approximated 2D model of the relative distance between points.

Note that we always have a visualization: the longer we run the algorithm, the better the Euclidean distances correspond to the distances between centroids in the model. As is common to this type of algorithm, one might converge to a local minimum. However, in practice this seems to occur hardly ever.

Now we further refine our notion of centroids:

1. Each centroid $\mathcal{C}_i$ has one coordinate pair $(x_{\mathcal{C}_i}, y_{\mathcal{C}_i})$.
2. Each of them has a unique set $I_{\mathcal{C}_i}$ of learning examples (subgraphs in the molecular setting). The $I_{\mathcal{C}_i}$'s are mutually disjoint.

The distance between centroids $\mathcal{C}_i$ and $\mathcal{C}_j$ is decided by the *average distance* between the items of $I_{\mathcal{C}_i}$ and $I_{\mathcal{C}_j}$:

$$m\_dist(I_{\mathcal{C}_i}, I_{\mathcal{C}_j}) = \frac{\sum_{a \in I_{\mathcal{C}_i}} \sum_{b \in I_{\mathcal{C}_j}} g\_dist(a, b)}{|I_{\mathcal{C}_i}| \cdot |I_{\mathcal{C}_j}|}$$

Practically we store these distances in a $|\mathcal{C}| \times |\mathcal{C}|$ distance matrix where only $|\mathcal{C}|(|\mathcal{C}| - 1)/2$ distances are stored, since $m\_dist(I_{\mathcal{C}_i}, I_{\mathcal{C}_j}) = m\_dist(I_{\mathcal{C}_j}, I_{\mathcal{C}_i})$. In this way we save memory, in comparison with push-and-pull where we have one point for each fragment. Each centroid has two-dimensional coordinates and n-dimensional distance vector and that is why we call the *centroids bi-dimensional*.

We say that a centroid has a *"leaking" opportunity* if the learning examples $I_{\mathcal{C}_i}$ have a chance to be transferred to another centroid because it better fits with the items of this centroid. This requires an adaptation of the distance matrix without recalculating the distances between other items of each centroid; this is more formally described in Section 5. The term "leaking" and "updating", as seen in traditional clustering, have some similarities. However "updating" is usually done using known input vectors to update the position of the cluster. With "leaking" we move an instance to a better group and in the process the distance of this group to all other groups changes.

5 The Algorithm

The basis of our algorithm is the random placement of the centroids and interchanging iterations of model (distance) optimization and leaking of the centroids. For both iterations we can set the number of iterations, however these should not be set too high. This is because centroids should get the opportunity to exchange items and via pushing and pulling be able to adapt the model to the new situation. Formally we define this basis as Algorithm 1 (next page).

The affected distances are all distances between the centroids $\mathcal{C}_1$ and $\mathcal{C}_2$ and all other centroids (including the distance between $\mathcal{C}_1$ and $\mathcal{C}_2$). In Figure 2 we give an example distance matrix for a model of 9 centroids, where we only need to store the white area. E.g., if an item leaks from centroid 3 to 5 then the arrows indicate which distance values need to be adapted.

We do not need to completely recalculate the affected distances, instead we can update the "old" distance. Indeed, if the centroid $\mathcal{C}_i$ *loses* an item a to a centroid $\mathcal{C}_j$ then for each distance with another centroid $\mathcal{C}_k$, where $k \neq i$ and $k \neq j$:

$$m\_dist(I_{\mathcal{C}_i}, I_{\mathcal{C}_k}) \leftarrow \frac{m\_dist(I_{\mathcal{C}_i}, I_{\mathcal{C}_k}) \cdot |I_{\mathcal{C}_i}| - m\_dist(\{a\}, I_{\mathcal{C}_k})}{|I_{\mathcal{C}_i}| - 1} \tag{3}$$

$$m\_dist(I_{\mathcal{C}_j}, I_{\mathcal{C}_k}) \leftarrow \frac{m\_dist(I_{\mathcal{C}_j}, I_{\mathcal{C}_k}) \cdot |I_{\mathcal{C}_j}| + m\_dist(\{a\}, I_{\mathcal{C}_k})}{|I_{\mathcal{C}_j}| + 1} \tag{4}$$

Algorithm 1 Leaking Centroid Algorithm: CENTROIDLEAK

```
Require: set $\mathcal{C}$ of centroids, database $\mathcal{D}$ of items
 1: Divide all $i \in \mathcal{D}$ evenly among the centroids
 2: for all $0 \leq j < |\mathcal{C}|$ do
 3:     Randomly choose $x_{\mathcal{C}_j}$ and $y_{\mathcal{C}_j}$ between 0 and 1
 4: end for
 5: for a user-defined number of interchanging iterations do
 6:     for a user-defined number of model optimizations do
 7:         Choose two random centroids
 8:         if Distance not in distance matrix then
 9:             Calculate the distance between the centroids and store it
10:         end if
11:         Use Equation 2 on these two centroids
12:     end for
13:     for a user-defined number of "leaking" opportunities do
14:         Choose two random centroids $\mathcal{C}_1$ and $\mathcal{C}_2$
15:         Choose one item $i \in \mathcal{C}_1$
16:         if $m\_dist(\{i\}, I_{\mathcal{C}_1} \backslash \{i\}) > m\_dist(\{i\}, I_{\mathcal{C}_2})$ then
17:             Transfer item $i$ to centroid $\mathcal{C}_2$
18:             Adapt the affected distances between the centroids using Equations 3 to 6
19:         end if
20:     end for
21: end for
```

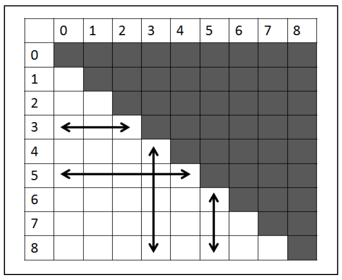

Figure 2: The distance matrix for model with 9 centroids (each white box represents a distance value).

Finally we need to adapt the distance between $\mathcal{C}_i$ and $\mathcal{C}_j$:

$$adaptation = m\_dist(\{a\}, I_{\mathcal{C}_i} \backslash \{a\}) \cdot (|I_{\mathcal{C}_i}| - 1) - m\_dist(\{a\}, I_{\mathcal{C}_j}) \cdot |I_{\mathcal{C}_j}| \tag{5}$$

$$m\_dist(I_{\mathcal{C}_i}, I_{\mathcal{C}_j}) \leftarrow \frac{m\_dist(I_{\mathcal{C}_i}, I_{\mathcal{C}_j}) \cdot |I_{\mathcal{C}_i}| \cdot |I_{\mathcal{C}_j}| + adaptation}{(|I_{\mathcal{C}_i}| - 1) \cdot (|I_{\mathcal{C}_j}| + 1)} \tag{6}$$

Note, in lines 8 and 9 of Algorithm 1, that distances are only calculated when needed. The number of distances to store is a little because of the user-defined number of groups, an advantage of CENTROIDLEAK.

6 Results and Performance

The experiments were done for two main reasons. Push-and-pull depends on the speed of the distance function. For large databases the g_dist function slows down. We want to show that CENTROIDLEAK can better deal with large databases if we can not store the distances between all fragments (items) in the main memory, *scalability*. We made no comparison with the SOM algorithm or any of the traditional clustering algorithms, because the reasons discussed in Section 2 made it less suitable for our purposes. Secondly we want to show that the models made by the algorithm actually approaches the model we expect.

All experiments were performed on an Intel Pentium 4 64-bits 3.2 GHz machine with 3 GB memory. As operating system Debian Linux 64-bits was used with kernel 2.6.8-12-em64t-p4.

The first dataset, called `SyntheticSmall`, is displayed in Figure 3. The dataset consists of 4 groups of 200 two-dimensional items. The second synthetic dataset, called `SyntheticLarge`, is displayed in Figure 5. The dataset consists of 11 groups of 500 two-dimensional items. Note that, contrary to the real-life datasets below, we know the original 2D locations of all data points, from which we compute the distances. The original coordinates are of course not employed by the algorithm, but can be used to verify the performance.

The first real-life dataset we use is the `4069.no_aro` dataset, containing 4,069 molecules; from this we extracted the 1,229 most frequent subgraphs using GSPAN (see [20]). This dataset was provided by Leiden/Amsterdam Center for Drug Research (LACDR). Other datasets we use are datasets of the National Cancer Institute (NCI), and can be found in [13]. One of these datasets contains 32,557 2D structures (molecules, average size is 26.3 nodes) with cancer test data as of August 1999; we will call this dataset the `NCI.CAN.99` dataset.

The approximation of `SyntheticSmall` displayed in Figure 4 shows the groups similarly positioned but slightly turned. The slight turn is caused by the algorithm only knowing the distance between points. Where in traditional push-and-pull you would plot all 800 items in 2D space, now we plot only 100 centroids representing self-optimizing groups of examples. Note that less points make it easier to distinguish (groups of) fragments that are co-occurring. This is essential for any user-interface that allows for further exploration of these centroids (or groups).

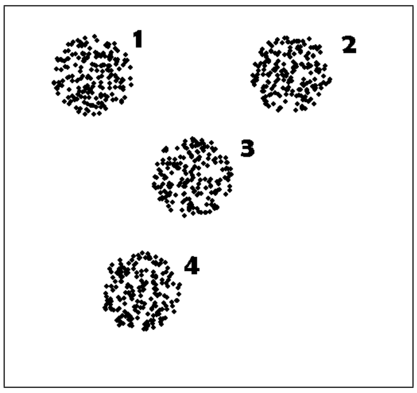

Figure 3: `SyntheticSmall`: Synthetic data when we also know the input vectors, 4 groups of 200 items.

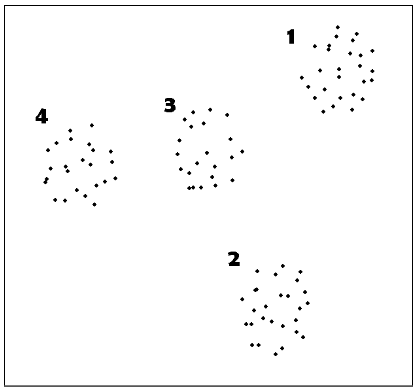

Figure 4: The approximation of `SyntheticSmall` with $n = 100$ centroids ($\alpha = 0.1$, 100 leaks, 10,000 model optimizations and 1,000 iterations).

In Figure 6 the algorithm approaches the `SyntheticLarge` dataset again with only 100 centroids instead of 5,500 (11 groups of 500). Note that we can calculate for the traditional push-and-pull, where we plot all items in 2D space, that $1/2 \cdot n(n-1)$ distances need to be stored and one will eventually run out of memory. Having many more than 5,500 points will eventually be a problem for traditional push-and-pull.

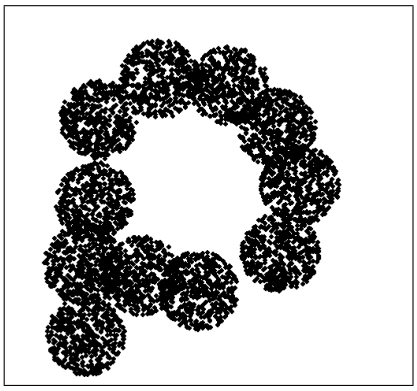

Figure 5: `SyntheticLarge`: Synthetic data when we also know the input vectors, 11 groups of 500 items.

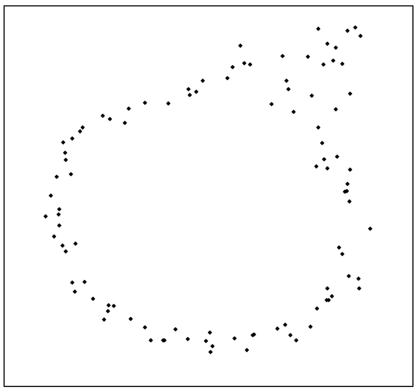

Figure 6: The approximation of `SyntheticLarge` with $n = 100$ centroids ($\alpha = 0.1$, 100 leaks, 10,000 model optimizations and 1,000 iterations).

For the `4069.no_aro` dataset the model of Figure 7 was build. We link those centroids for which items on average are co-occurring frequently and one can see these are often closer in 2D space.

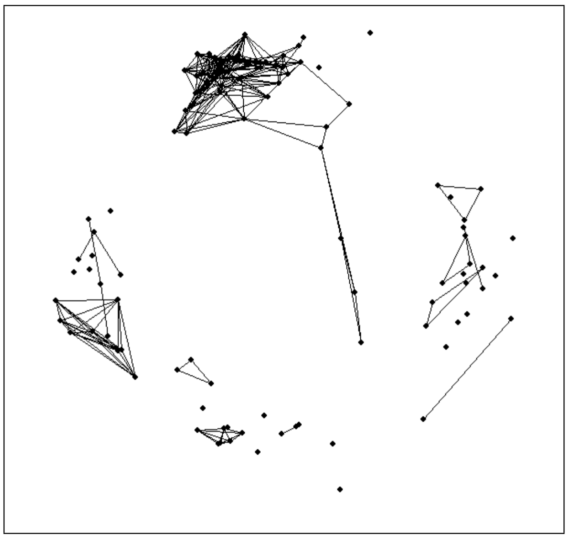

Figure 7: The approximation for the 4069.no_aro dataset with $n = 100$ centroids ($\alpha = 0.1$, 100 leaks, 10,000 model optimizations, 1,000 iterations).

In Figure 8 it is shown how runtime for a large dataset grows worse for traditional push-and-pull as the number of model optimizations increases. With Figure 8 we want to show that with "leaking" centroids the push-and-pull algorithm becomes more scalable. Due to the grouping of fragments and the lazy updating of the distance matrix there is much less need to count co-occurrence of the fragments.

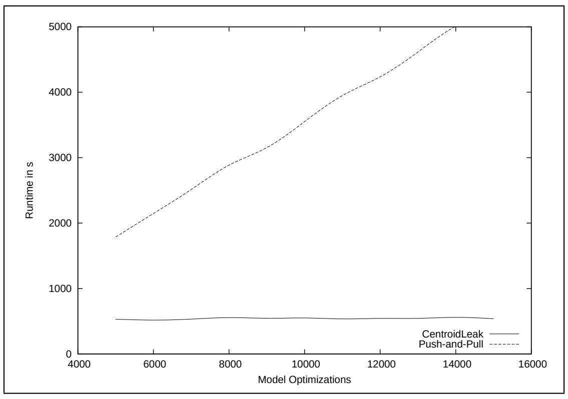

Figure 8: Runtime in seconds with variating model optimizations for the NCI.CAN.99 dataset with $n = 100$ centroids ($\alpha = 0.1$, 100 leaks, 1,000 iterations).

7 Conclusions and Future Work

In this work we proposed an algorithm for improving the scalability of push-and-pull, a competitive neural network algorithm. The algorithm was able to deal better with larger databases and a great amount of points in the model. This was done by having a fixed amount of centroids where the items are evenly divided among the centroids. Every few iterations certain items have the opportunity to "leak" to a more suitable group. We have experimentally shown that expected models are found. With our algorithm we hope to combine the advantages of Self-Organizing Maps, e.g., one neuron for many similar points, with the advantages of push-and-pull, e.g., good intermediate approximations and visualization with only distance information. The new method gives a better overview, using less points, it only employs intermediate distances, and is easy to understand and to adapt.

In the future we want to make the updating of the distances more lazy, allowing for an even faster algorithm and we want to analyse the biological findings with the improved push-and-pull algorithm.

References

[1] A.M. Bronstein, M.M. Bronstein and R. Kimmel. Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching. *Proc. of National Academy of Sciences (PNAS)* **103** (2006) 1168–1172.

[2] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* **39** (1977) 1–38.

[3] H. Gao, C. Williams, P. Labute, and J.W. Bajorath. Binary quantitative structure-activity relationship (QSAR) analysis of estrogen. *Chemical Information and Computer Sciences* **39** (1999) 164–168.

[4] P. Gedeck and P. Willet. Visual and computational analysis of structure-activity relationships in high-throughput screening data. *Current Opinion in Chemical Biology* **5** (2001) 389–395.

[5] E.H. de Graaf, J.N. Kok, and W.A. Kosters. Visualization and grouping of graph patters in molecular databases. *Proc. of 27th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI-2007)*, pp. 267–280 (2007).

[6] J. Hanke, G. Beckmann, P. Bork, and J.G. Reich. Self-organizing hierarchic networks for pattern recognition in protein sequence. *Protein Science Journal* **5** (1996) 72–82.

[7] S. Izrailev and D.K. Agrafiotis. A method for quantifying and visualizing the diversity of QSAR models. *Molecular Graphics and Modelling* **22** (2004) 275–284.

[8] T. Kohonen. *Self Organizing Maps*. Springer Series in Information Sciences, Vol. 30, third extended edition (2001).

[9] W.A. Kosters and M.C. van Wezel. Competitive neural networks for customer choice models. *E-Commerce and Intelligent Methods of Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer*, **105** (2002) 41–60.

[10] E.W. Lameijer, T. Bäck, J.N. Kok, and A.P. IJzerman. Mining a chemical database for fragment co-occurrence: Discovery of “chemical clichés”. *Journal of Chemical Information and Modelling* **46** (2006) 553–562.

[11] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proc. of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297 (1967).

[12] S. Mahony, D. Hendrix, T.J. Smith, and A. Golden. Self-organizing maps of position weight matrices for motif discovery in biological sequences. *Artificial Intelligence Review Journal* **24** (2005) 397–413.

[13] National Cancer Institute (NCI) website, DTP/2D and 3D structural information, `http://cactus.nci.nih.gov/ncidb2/` [accessed 18.6.2008].

[14] N. Rhodes, P. Willet, J. Dunbar, and C. Humblet. Bit-string methods for selective compound acquisition. *Chemical Information and Computer Sciences* **40** (2000) 210–214.

[15] G. Roberts, G.J. Myatt, W.P. Johnson, K.P. Cross, and P.E. Blower. Leadscope: Software for exploring large sets of screening data. *Chemical Information and Computer Sciences* **40** (2000) 1302–1314.

[16] J.W. Sammon Jr. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers* **C-18** (1969) 401–409.

[17] J.B. Tenenbaum, V. de Silva, and J.C. Langford: A global geometric framework for nonlinear dimensionality reduction. *Science* **290(5500)** (2000) 2319–2323

[18] P. Willet, J.M. Barhad, and G.M.J. Downs: Chemical similarity searching. *Journal of Chemical Information and Computer Sciences* **38** (1999) 983–996.

[19] J. Xu, Q. Zhang, and C.-K Shih. V-cluster algorithm: A new algorithm for clustering molecules based upon numeric data. *Molecular Diversity* **10** (2006) 463–478.

[20] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. *Proc. 2002 IEEE International Conference on Data Mining (ICDM)*, pp. 721–724 (2002).