EmbeddedMontiArc automated component clustering

(draft, version 2018-12-31)

Objective

Bundle interconnected top level components of the model into different clusters. The aim is to reduce connection and communication overhead between components by grouping affine components into different clusters which then are connected using ROS.

Procedure

1) Convert the symbol table of a component into an adjacency matrix
    - Order all sub components by name (neccessary for the adjacency matrix).
    - Create adjacency matrix to use with a clustering algorithm, with subcomponents as nodes and connectors between subcomponents as vertices. Sift out all connectors to the super component.
2) Feed (weighted) adjacency matrix into the selected clustering algorithm
    - Spectral clustering
        - We are using the machine learning library „smile ml“ (r_sml) which provides a broad range of different clustering and partitioning approaches. As a prime example we are using „spectral clustering“ here.
        - The clustring algorithm yields multiple cluster labels with the clustered entries of the adjacency matrix assigned to them. We have to convert them back to a set of symbol tables of components representing the clusters.
    - Markov Clustering (MCL)
        - We are using the machine learning library „java ml“ (r_jml).
        - The clustring algorithm yields a sparse matrix which is interpreted and converted into a multi-dimensional dataset, representing the clusters and associated data points. We use the adjacency matrix order (by subcomp name) to convert this dataset back to a set of symbol tables of components suitable for Monti.
    - DBScan
        - We are using the machine learning library „smile ml“ (r_sml)
        - The clustring algorithm yields multiple cluster labels with the clustered entries of the adjacency matrix assigned to them. We have to convert them back to a set of symbol tables of components representing the clusters.
3) Generate middleware tags separating the clusters
    - This will build the cluster-to-ROS connections.
    - We won’t take account of ports of the super component and only consider connected top level components.
    - A connection will be established if the target cluster label is different from the source cluster label thus connecting different clusters with each other.
4) Feed result into existing manual clustering architecture

Clustering algorithms in a nutshell

Spectral Clustering

The goal of spectral clustering is to cluster data which is connected but not compact or not clustered within convex boundaries. Data is basically seen as a connected graph and clustering is the process of finding partitions in the graph based on the affinity (similarity or adjacency) of vertices.

The general approach is to perform dimensionality reduction before clustering in fewer dimensions using a standard clustering method (like k-means) on relevant eigenvectors (the "spectrum") of the matrix representation of a graph (Laplacian matrix).

Basically we follow three steps in spectral clustering

1) Pre-processing
    a. Construct a matrix representation of a graph
2) Decomposition
    a. Compute eigenvalues and eigenvectors of the matrix
    b. Map each point to a lower-dimensional representation based on one or more eigenvectors
3) Grouping
    a. Assign points to two or more clusters, based on the new representation

Pre Pre-processing: How to define the affinity of data points and decide upon the connectivity of a similarity graph?

It is the affinity of data points, which in the end defines clusters, rather than the absolute (spatial) location or spatial proximity. We have to define both, a way to calculate affinity (similarity function) of data points, and a respective graph representation (from which then the similarity matrix is derived), i.e. which data points (vertices) will be connceted in the graph by (undirected) edges. Typically a similarity function evaluates the distance, this can either be the classic Euclidian distance or a Gaussian Kernel similarity function.

The most wide spread approach to a graph representation is kNN, the k nearest neighbors of a vertex. In this approach the k nearest neighbors of a vertex v vote on where v belongs and thus should be connected to. The goal is to connect vertex vi with vertex vj with an edge if vj is among the k-nearest neighbors of vi. Because this leads to a directed graph, we need an approach to convert it to an undirected one. This can be done in two ways: Either there's an edge if p is NN of q OR q is NN of p. Or there's an edge if p is NN of q AND q is NN of p (this is called mutual kNN, which is in practice a good choice).

Other possible approaches to decide on the connectivity of a similarity graph are "epsilon neighborhood" (a threshold based approach to construct a binary adjacency matrix) or a fully connected graph in combination with a Gaussian similarity function.

Affinity evaluation

Within an affinity matrix, data points belonging to the same cluster have a very similar affinity vector to all other data points (eigenvector). Each eigenvector has an eigenvalue which states how

prevalent its vector is in the affinity matrix. So those eigenvectors act like a fingerprint for different clusters, representing all datapoints belonging to a specific cluster, in a lower dimensional space.

The Laplacian matrix

The Laplacian matrix L is defined as L= D-A, where D is the degree matrix (a diagonal matrix, containing the number of direct neighbors of a vertex) and A is the (binary) adjacency matrix (Aji=1 if vertecies i and j are connected with an edge, 0 otherwise).

Markov Clustering (MCL)

The basic idea is to run a flow simulation in a graph, representing the data to be clustered. The graph is explored by „random walks“ which utilize two basic property of clusters: Many edges within a cluster, few egdges between clusters. This means a random walk is more likely to stay within a cluster and less likely to leave it (so it’s possible to discover where the flow tends to „gather“). Unlike most other clustering approaches there’s no prior knowledge needed about the structure of your cluster data.

Basically we follow these steps in MCL

1) Pre-processing
    a. Transform the graph into a probability (or transition) matrix (e.g. an adjacency matrix can be easily transformed into a probability matrix) which simply shows node adjacency as 1/degree.
    b. Normalize the matrix (either each column of M or each row of M^T sums up to 1).
2) Expansion / Inflation
    a. Flow is easier within dense regions than across sparse boundaries. The edge weights will be higher within clusters, and lower between the clusters. This means there is a correspondence between the distribution of weight over the columns and the clusterings. MCL deliberately boosts this affect by adjusting for each vertex the transition values such that strong neighbors are further strengthened and less popular neighbors are demoted.
        i. Expand by taking the n^{th} power of the (markov chain) transition matrix. Allows for flow to connect different regions of the graph.
        ii. Inflate the resulting matrix (square and normalize). This operation is responsible for both strengthening and weakening of the flow current (further strengthens strong currents, and weaken already weak currents).
    b. Those two steps are alternated between repeatedly until a steady state is reached (convergence).
3) Convergence
    a. Convergence is not obvious and has not been proven in the paper the algorithm was introduced in originally.
    b. Nevertheless in practice, the algorithm is said to converge nearly always to a "doubly idempotent" matrix (i.e. every value in a single column has the same number), see (i21).
4) Interpret resulting matrix to discover clusters.

a. To interpret clusters, the vertices are split into two types. Attractors, which attract other vertices, and vertices that are being attracted by the attractors.
b. Attractors and the elements they attract are put together into the same cluster.

For further information see (i2), (i21)

DBSCAN (Density Based Spatial Clustering of Applications with Noise)

This method scans the data for proximity and density observations. It identifies the number of clusters itself, so it is useful in unsupervised learning (though a successful clustering heavily depends on the two mandatory parameters provided).

All data points eventually become a part of some cluster, or will be identified as outliers, even if the observations are spread widely across the vector space.

Clustering is performed based on two important parameters:

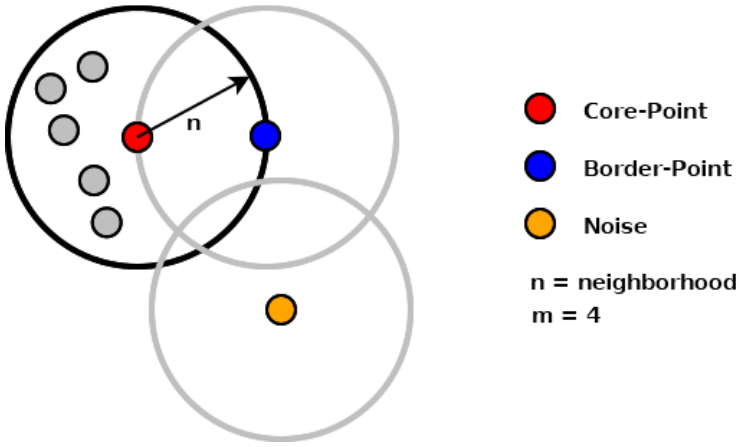
- ε - neighbourhood (n) - cutoff distance of a point to be considered as part of a cluster.
- minimum points (m) - minimum number of points required to form a cluster.

There are three types of points after the DBSCAN clustering is complete:

- Core - This is a point which has at least *m* points within distance *n* from itself.
- Border - This is a point which has at least one Core point at a distance *n*.
- Noise - This is a point which is neither a Core nor a Border. And it has less than *m* points within distance *n* from itself.

The DBSCAN clustering algorithm:

- For each point *P* in the dataset, identify points *pts* within distance *n* as follows
  - if $pts \geq m$, label *P* as a *Core* point
  - if $pts < m$ and a Core point exists at distance *n*, label *P* as a *Border* point
  - otherwise (if $pts < m$), label *P* as noise
- All the overlapping Core-Sets (i.e. a Core point and all the points within distance n) are grouped together into one cluster.



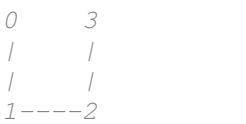
Clustering Algorithms Comparison Table

	Spectral Clustering	Markov (MCL)	DBScan
Parameters (* mandatory)	No. of clusters (*)	No mandatory params, but no. of clusters not applicable	Radius (*) for nearest neighbor search No. of minimum points for a cluster (*)
How much I need to know about my data up-front (domain knowledge)?	A little: How many clusters do I expect?	Parameter-wise literally nothing at all	Pretty much (structural/density info): What perimeter I consider a neighborhood? What's my expectation for a minimum data size per cluster? The outcome of the clustering process depends heavily on those two parameters.
Worst-case runtime complexity	$O(N^3)$	$O(N^3)$	$O(N^2)$
Prevailing complexity factor	eigen-decomposition, $O(N^3)$ (inflation step $O(N^2)$)	matrix multiplication on two matrices of dimension N	nearest neighbor search, ε-neighborhood
Runtime complexity of enhanced implementations	$O(L^3)$ for L-constrained Spectral Clustering (S1)	Depending on the implementation, logarithmic complexity can be achieved (M1)	O(N log N) on average (D1)
Unique characteristic	Good tradeoff between parametrization and utility of clustering, works well on cancave and sparse clusters	100% unsupervised clustering	Might be used to explicitly identify outliers in the data
Ref. implementation used, version	smile ml, 1.5.2 (r_sml)	java-ml, 0.1.7 (r_jml)	smile ml, 1.5.2 (r-sml)
Further information	(i1)	(i2)	(i3)

Example graphs

Tiny / minimal / proof-of-concept

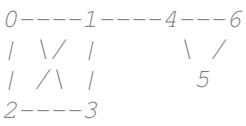
Please note: This is used for Spectral Clustering. MCL will naturally cluster this into 4 different parts because it's basically just a chain of 4 vertices.



Adjacency matrix

```
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
```

Small / simple example



Adjacency matrix

```
0 1 1 1 0 0 0
1 0 1 1 1 0 0
1 1 0 1 0 0 0
1 1 1 0 0 0 0
0 1 0 0 0 1 1
0 0 0 0 1 0 1
0 0 0 0 1 1 0
```

Transition matrix

```
  0, .33, .33, .33,   0,   0,   0
.25,   0, .25, .25, .25,   0,   0
.33, .33,   0, .33,   0,   0,   0
.33, .33, .33,   0,   0,   0,   0
  0, .33,   0,   0,   0, .33, .33
  0,   0,   0,   0,  .5,   0,  .5
  0,   0,   0,   0,  .5,  .5,   0
```

(row-major order as expected by java-ml)

Big / real-life

todo

Sources of information

Reference implementations used:

(r_sml) http://haifengl.github.io/smile/

(r_jml) http://java-ml.sourceforge.net/

Runtime complexity of enhanced implementations:

(S1) The complexity of the graph matrix construction – and thus the complexity of the eigen-decomposition $O(N^3)$ - can be bounded by $L << N$ if the affinity matrix is modified using pairwise constraints (so-called Nyström method).

For more information see: https://www.sciencedirect.com/science/article/pii/S1877050918302552

(M1) The specific kind of implementation used in java-ml is not clear. But there are implementations of Markov Clustering, capable of performing at O(L log K), where matrix square computation is done with a limited sparse matrix approach, with a maximum number K of nonzero entries per column and an upper bound L for the size of a newly computed matrix column, where L is bound by the smallest of the two numbers N and K^2.

For more information see: https://micans.org/mcl/man/mclfaq.html

(D1) Two factors can speed up the algorithm: (a) An appropriate indexing structure (e.g. R*-trees) is used which executes a fixed-radius nearest neighbor search query in O(log N) and (b) The Epsilon neighborhood parameter ε is chosen in a meaningful way, such that on average only O(log N) points are returned (small compared to the size of the whole data space).

For more information see: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.121.9220

Further algorithm information (date of page-view: 11-30-2018):

(i1) https://arxiv.org/abs/0711.0189

(i2) https://dspace.library.uu.nl/bitstream/handle/1874/848/full.pdf

(i21) https://www.cs.ucsb.edu/~xyan/classes/CS595D-2009winter/MCL_Presentation2.pdf

(i3) http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.121.9220