Dependence Communities in Software

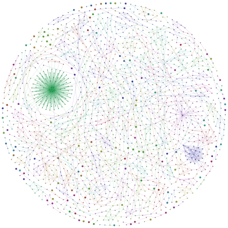
Crest COW

UCL

Sebastian Danicic and James Hamilton

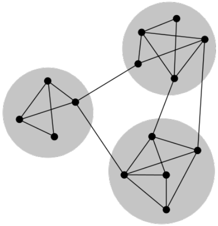
Goldsmiths, University of London

30th April 2012



Communities in Graphs

A network is said to have *community structure* if the nodes of the network can be easily grouped into (potentially overlapping) sets of nodes such that each set of nodes is densely connected internally, with few connections to the rest of the network.



Communities in Real World Graphs

Many real-world networks are known to have community structure.

Communities in Real World Graphs

Many real-world networks are known to have community structure.

- Social networks

Communities in Real World Graphs

Many real-world networks are known to have community structure.

- Social networks
- Biological networks

Communities in Real World Graphs

Many real-world networks are known to have community structure.

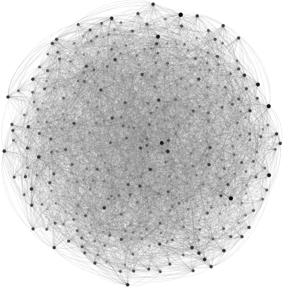
- Social networks
- Biological networks
- Computer networks

Communities in Real World Graphs

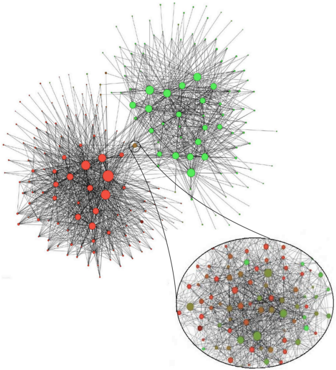
Many real-world networks are known to have community structure.

- Social networks
- Biological networks
- Computer networks

Not all networks have community structure e.g. random graphs



Communities in Real World Graphs



“Graphical representation of the network of communities extracted from a Belgian mobile phone network. About 2M customers are represented on this network. The size of a node is proportional to the number of individuals in the corresponding community and its colour on a red-green scale represents the main language spoken in the community (red for French and green for Dutch). Only the communities composed of more than 100 customers have been plotted. Notice the intermediate community of mixed colours between the two main language clusters. A zoom at higher resolution reveals that it is made of several sub-communities with less apparent language separation.”

(Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks.

Journal of Statistical Mechanics: Theory and Experiment, 2008(10), P10008. doi:10.1088/1742-5468/2008/10/P10008)

Does Software have Community Structure?

Does Software have Community Structure?

It depends how you turn the software into a graph.

Does Software have Community Structure?

It depends how you turn the software into a graph.

Consider, the graph:
$G_1(P) = n_1 \rightarrow n_2$ if and only if n_1 and n_2 are in the same function in program P.

Does Software have Community Structure?

It depends how you turn the software into a graph.

Consider, the graph:
$G_1(P) = n_1 \rightarrow n_2$ if and only if n_1 and n_2 are in the same function in program P.

Clearly G_1 has community structure but it's not very interesting!

Does Software have Community Structure?

It depends how you turn the software into a graph.

Consider, the graph:
$G_1(P) = n_1 \to n_2$ if and only if n_1 and n_2 are in the same function in program P.

Clearly G_1 has community structure but it's not very interesting!

Previous work has shown community structure exists in class dependence graphs.

'Interesting' Communities in Software

We are looking for communites which reflect *semantic* properties of programs.

'Interesting' Communities in Software

We are looking for communites which reflect *semantic* properties of programs.

Where do we start?

'Interesting' Communities in Software

We are looking for communites which reflect *semantic* properties of programs.

Where do we start?

We have to choose graphs which reflect semantic properties of programs.

'Interesting' Communities in Software

We are looking for communites which reflect *semantic* properties of programs.

Where do we start?

We have to choose graphs which reflect semantic properties of programs.

We then find communites in these graphs.

‘Interesting’ Communities in Software

We are looking for communites which reflect *semantic* properties of programs.

Where do we start?

We have to choose graphs which reflect semantic properties of programs.

We then find communites in these graphs.

Finally we see if these communities reflect anything semantic.

Slice Graphs

$S(P) = n_1 \rightarrow n_2$ if and only if n_2 is in the slice of P with respect to n_1.

Slice Graphs

$S(P) = n_1 \rightarrow n_2$ if and only if n_2 is in the slice of P with respect to n_1.

In other words, $n_1 \rightarrow n_2$ if and only if n_1 <u>depends on</u> n_2 in P.

Slice Graphs

$S(P) = n_1 \to n_2$ if and only if n_2 is in the slice of P with respect to n_1.

In other words, $n_1 \to n_2$ if and only if n_1 depends on n_2 in P.

Clearly, slice graphs can be considered 'semantic'.

Slice Graphs

$S(P) = n_1 \rightarrow n_2$ if and only if n_2 is in the slice of P with respect to n_1.

In other words, $n_1 \rightarrow n_2$ if and only if n_1 depends on n_2 in P.

Clearly, slice graphs can be considered 'semantic'.

Question: Do slice graphs have community structure, and if so are the communites 'interesting' or 'useful'?

Slice Graphs

$S(P) = n_1 \to n_2$ if and only if n_2 is in the slice of P with respect to n_1.

In other words, $n_1 \to n_2$ if and only if n_1 depends on n_2 in P.

Clearly, slice graphs can be considered 'semantic'.

Question: Do slice graphs have community structure, and if so are the communites 'interesting' or 'useful'?

Intuitively, a community in a slice graph is a part of a program where there is strong internal inter-dependence.

Slice Graphs

$S(P) = n_1 \rightarrow n_2$ if and only if n_2 is in the slice of P with respect to n_1.

In other words, $n_1 \rightarrow n_2$ if and only if n_1 depends on n_2 in P.

Clearly, slice graphs can be considered 'semantic'.

Question: Do slice graphs have community structure, and if so are the communites 'interesting' or 'useful'?

Intuitively, a community in a slice graph is a part of a program where there is strong internal inter-dependence.

Perhaps dependence communities will highlight different semantic concerns within a program.

Modularity

Given a partition of a network, modularity is a measure of the 'strength' of the community structure of this partition.

$$Q = \begin{array}{c}\text{(fraction of edges that fall}\\ \text{within communities in the}\\ \text{given graph)}\end{array} - \begin{array}{c}\text{(expected number of edges}\\ \text{within those communities}\\ \text{in the null model)}\end{array} \tag{1}$$

Modularity

Given a partition of a network, modularity is a measure of the 'strength' of the community structure of this partition.

$$Q = \begin{matrix}\text{(fraction of edges that fall} \\ \text{within communities in the} \\ \text{given graph)}\end{matrix} - \begin{matrix}\text{(expected number of edges} \\ \text{within those communities} \\ \text{in the null model)}\end{matrix} \qquad (1)$$

Modularity, of a weighted undirected graph, is defined as

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - E_{ij}\right]\delta(c_i, c_j) \qquad (2)$$

where A_{ij} is the weight of the edge incident to i and j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges incident to vertex i, c_i is the community to which vertex i is assigned, $\delta(u, v)$ is 1 if i and j are in the same community and 0 otherwise and $m = \frac{1}{2}\sum_{i,j} A_{ij}$. E_{ij} is the expected number of edges between i and j in a random graph of the same degree distribution which can be calculated as $\frac{k_i k_j}{2m}$.

Algorithms for Finding Comunities

Finding partitions with the best modularity is NP-hard but tractable algortihms exist for aproximation best possible exist.

Algorithms for Finding Comunities

Finding partitions with the best modularity is NP-hard but tractable algortihms exist for aproximation best possible exist.

The *Louvain method* is a fast algorithm for detecting communities in large networks based upon modularity maximisation.

Algorithms for Finding Comunities

Finding partitions with the best modularity is NP-hard but tractable algortihms exist for aproximation best possible exist.

The *Louvain method* is a fast algorithm for detecting communities in large networks based upon modularity maximisation.

The algorithm combines neighbouring nodes until a local maximum of modularity is reached and then creates a new network of communities; these two steps are repeated until there is no further increase in modularity.

Algorithms for Finding Comunities

Finding partitions with the best modularity is NP-hard but tractable algortihms exist for aproximation best possible exist.

The *Louvain method* is a fast algorithm for detecting communities in large networks based upon modularity maximisation.

The algorithm combines neighbouring nodes until a local maximum of modularity is reached and then creates a new network of communities; these two steps are repeated until there is no further increase in modularity.

This creates a hierarchical decomposition of the network - at the lowest level all nodes are in their own community, and at the highest level nodes are in communities which gives the highest gain in modularity.

Algorithms for Finding Comunities

Finding partitions with the best modularity is NP-hard but tractable algortihms exist for aproximation best possible exist.

The *Louvain method* is a fast algorithm for detecting communities in large networks based upon modularity maximisation.

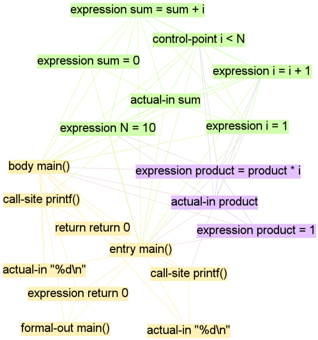
The algorithm combines neighbouring nodes until a local maximum of modularity is reached and then creates a new network of communities; these two steps are repeated until there is no further increase in modularity.

This creates a hierarchical decomposition of the network - at the lowest level all nodes are in their own community, and at the highest level nodes are in communities which gives the highest gain in modularity.

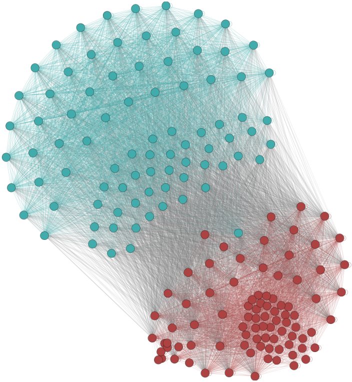
This technique is simple, fast and has good accuracy and has been tested on networks with millions of vertices/edges.

Example of Communities in the Slice Graph: Sum/Product

```
int main() {

        const int N = 10;
        int sum = 0;
        int product = 1;
        int i = 1;

        while(i < N) {
           sum = sum + i;
           product = product * i;
           i = i + 1;
        }

        printf("%d\n", sum);
        printf("%d\n", product);
}
```



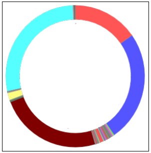
Example of Communities in the Slice Graph: Word Count Program



It separates out the code that does the counting from the code that does the I/O.

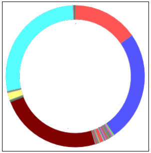
More Examples of Communities in the Slice Graph

More Examples of Communities in the Slice Graph

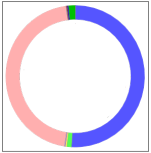


GNU Chess: frontend, adapter and engine

More Examples of Communities in the Slice Graph

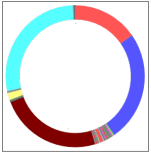


GNU Chess: frontend, adapter and engine

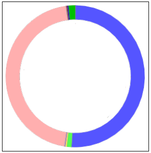


GNU bc: parser, calculator

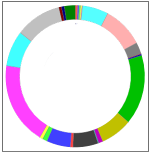
More Examples of Communities in the Slice Graph



GNU Chess: frontend, adapter and engine

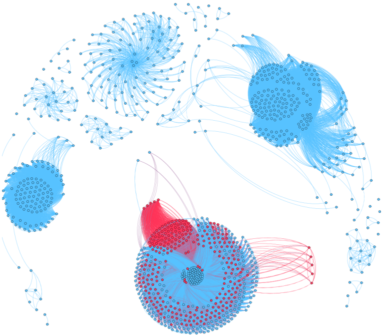


GNU bc: parser, calculator



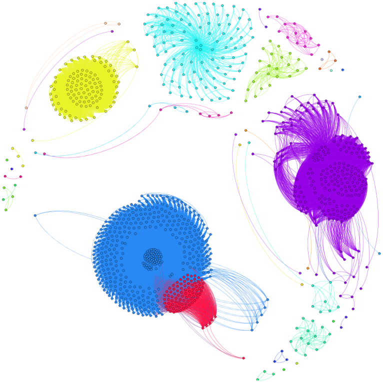
GNU robots: many communities due to low coupling

Applications - Detecting Dynamic Watermarks in Java Code



The red bits are the dynamic watermark we injected.

Applications - Detecting Dynamic Watermarks in Java Code



The red bits are the bits of the watermark discovered by the communites algorithm.

Dependence Clusters vs. Dependence Communities

A Dependence cluster is a maximal set of mutually dependent vertices i.e. a maximal clique in the slice graph.

Dependence Clusters vs. Dependence Communities

A Dependence cluster is a maximal set of mutually dependent vertices i.e. a maximal clique in the slice graph.

Finding maximal cliques is also NP-Hard.

Dependence Clusters vs. Dependence Communities

A Dependence cluster is a maximal set of mutually dependent vertices i.e. a maximal clique in the slice graph.

Finding maximal cliques is also NP-Hard.

A clique is a fully connected subgraph. This may be an overly strict requirement (Harman et al.).

Dependence Clusters vs. Dependence Communities

A Dependence cluster is a maximal set of mutually dependent vertices i.e. a maximal clique in the slice graph.

Finding maximal cliques is also NP-Hard.

A clique is a <u>fully</u> connected subgraph. This may be an overly strict requirement (Harman et al.).

Perhaps Dependence Communites are a 'good enough' approximation for what is required for Dependence Clusters.

Dependence Clusters vs. Dependence Communities

Because it is hard to compute Dependence Clusters, they approximate by saying:

Dependence Clusters vs. Dependence Communities

Because it is hard to compute Dependence Clusters, they approximate by saying: *two nodes are in the same Dependence Cluster if and only if they have the same slice.*

Dependence Clusters vs. Dependence Communities

Because it is hard to compute Dependence Clusters, they approximate by saying: *two nodes are in the same Dependence Cluster if and only if they have the same slice.*

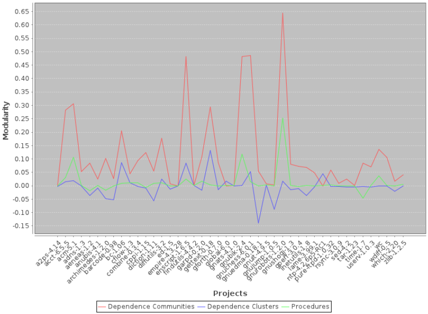
These are cliques, but not, in general, maximal ones.

Dependence Clusters vs. Dependence Communities

Because it is hard to compute Dependence Clusters, they approximate by saying: *two nodes are in the same Dependence Cluster if and only if they have the same slice.*

These are cliques, but not, in general, maximal ones.

It turns out that, If we apply the Louvain algorithm to the same graphs we get a partition with higher modularity. In other words it produces ‘clusters’ with a stronger ‘internal inter-dependence’ than those produced by Harman’s approximation.



What does this mean?

It could be argued, therefore, that Dependence Communites may be a better approximation to Dependence Clusters.

What does this mean?

It could be argued, therefore, that Dependence Communites may be a better approximation to Dependence Clusters.

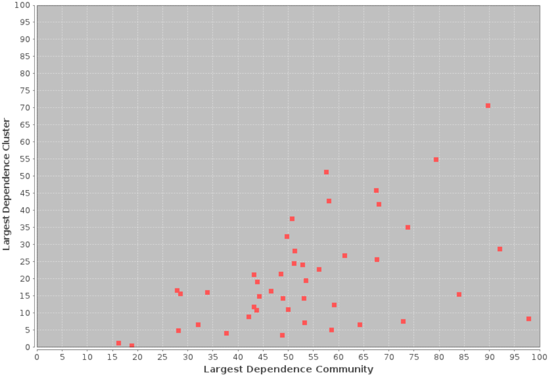
. . . or at least a better approximation to the properties of programs that authors are trying to capture using Dependence Clusters!

Programs with Large Dependence Clusters are bad!

Is there a correlation betwen large Dependence Clusters and Large Dependence Communities?

Programs with Large Dependence Clusters are bad!

Is there a correlation betwen large Dependence Clusters and Large Dependence Communities?



$$R = 0.8, p < 0.00001$$

Conclusions

Conclusions

We have introduced the concept of Dependence Communites.

Conclusions

We have introduced the concept of Dependence Communites.

We find these by using an algorithm that attempts to partition the slice graph in order maximise the modularity.

Conclusions

We have introduced the concept of Dependence Communites.

We find these by using an algorithm that attempts to partition the slice graph in order maximise the modularity.

There is a strong correlation between Dependence Communities and Dependence Clusters.

Conclusions

We have introduced the concept of Dependence Communites.

We find these by using an algorithm that attempts to partition the slice graph in order maximise the modularity.

There is a strong correlation between Dependence Communities and Dependence Clusters.

Programs that we investigated have a positive but, in most cases, not high modularity.

Programs that we investigated have a positive but, in most cases, not high modularity.

Conclusions

We have introduced the concept of Dependence Communites.

We find these by using an algorithm that attempts to partition the slice graph in order maximise the modularity.

There is a strong correlation between Dependence Communities and Dependence Clusters.

Programs that we investigated have a positive but, in most cases, not high modularity.

Programs that we investigated have a positive but, in most cases, not high modularity.

Dependence Communities reflect semantic properties of a program.

Thanks

Thanks for listening.

Any questions?