Evaluating Software Clustering
What is a good software decomposition?
The problem

- How do we know that a particular decomposition of a software system is good?
- What does “good” mean anyway?
- We can compare against a
  - Mental model
  - Benchmark standard
- Can be done either manually or automatically
Manual evaluation

- Have experts evaluate automatic decompositions
- Very time-consuming, impractical
- Also quite subjective
- Need an automatic, objective way of doing it
Automatic evaluation

- Usually measures the similarity of an automatic decomposition A to an authoritative decomposition B (prepared manually)
- Major drawback: Assumes there exists one “correct” decomposition
- Other evaluation approaches are possible, such as measuring the stability or the extremity distribution of a clustering algorithm
Standard Information Retrieval measures

Were applied in a software clustering context by Anquetil

Definitions:
- Intra pair: A pair of software entities in the same cluster
- Inter pair: A pair of entities in different clusters
• Precision: Percentage of intra pairs in A which are also intra in B

• Recall: Percentage of intra pairs in B also found in A

• A good algorithm should exhibit high values in both measures
Pair 1-5: Intra pair in A but not in B
Precision: $16/20 = 80\%$ Recall: $16/21 = 76.2\%$
Problems with Precision / Recall

- Sensitive to the size and number of clusters
  - Differences are exaggerated if you have small/many clusters

- Two values makes comparison harder

- The two values are interchangeable if there is no authoritative decomposition
Koschke - Eisenbarth measure (KE)

- Loosely based on Precision/Recall
- Attempts to be less strict
- Definitions:
  - GOOD match: Two clusters (one in A, one in B) with both precision and recall larger than a threshold $p$ (typical value 70%)
  - OK match: Two clusters with only one of the measures larger than $p$
Koschke - Eisenbarth metric

\[ \sum_{(a,b) \in \text{GOOD}} \text{overlap}(a, b) + \sum_{(a,b) \in \text{OK}} \text{overlap}(a, b) \]

\[ \left| \text{GOOD} \right| + \left| \text{OK} \right| + \left| \text{true negatives} \right| \]
- Does not take edges into account
- In extreme situations (each cluster contains only one element) may provide strange results (similarity of 100%)
- No penalty for joining clusters
Assignment tool: ke

- Takes two .kos files containing different decompositions of the same set of entities

- Example:
  `ke -cand dec1.kos -ref dec2.kos`

- Produces lots of output. We’re interested in the recall rate

- Transform an RSF file with contain facts to the KE format with
  `unitrans input.rsf output.kos`
MoJo distance

- The distance between two different partitions of the same software system is defined as the minimum number of Move and Join operations to transform one to the other
  - Move: Remove an object from a cluster and put it in a different cluster
  - Join: Merge two clusters into one
  - Split: Has to be simulated by Move operations
MoJo example

Decomposition A

Decomposition B

MoJo(A,B) = 2
MoJo(B,A) = 3
Why only Move and Join?

- Two clusters can be joined in only one way. One cluster can be split in two in an exponential number of ways.
- Joining two clusters only means that we performed more detailed clustering than required.
- Splitting is effectively assigned a weight equal to the cardinality of the smaller of the two resulting clusters.
Computing MoJo distance

- MoJo distance can be computed in polynomial time
- Worst case complexity is $O(n^3)$ but with real data it is no worse than $O(n \log n)$
Assignment tool: mojo

- Takes two .rsf files containing different decompositions of the same set of entities
- Example: `mojo dec1.rsf dec2.rsf`
- Output: 383
- If the two decompositions do not refer to the same set of entities, only the intersection of the two sets is considered.
MoJoFM Effectiveness Metric

\[ MoJoFM(A) = (1 - \frac{MoJo(A, B)}{\max_{\forall c}(MoJo(C, B))}) \times 100\% \]

- The denominator is the maximum possible MoJo distance to decomposition B
  - Can be computed by construction
EdgeSim

- Other measures do not consider edges
- Edges might convey important information
- EdgeSim penalizes clustering algorithms for changing the edge types
Inter-edge: Edge between clusters
Intra-edge: Edge within a cluster
Y: set of edges that are of the same type in both A and B
EdgeSim example

EdgeSim(A,B) = \frac{weight(Y)}{weight(E)} \times 100% 

In this example, EdgeSim(A,B) = 52.6%
EdgeSim counterexample
EdgeMoJo philosophy

- A similarity measure cannot make assumptions as to what cluster a particular object should belong to.

- Dissimilarity between decompositions should increase if the misplacement of an object results in the misplacement of a large number of edges.
EdgeMoJo calculation

- Apply MoJo and obtain a series of Move and Join operations
- Perform all Join operations
- The cost of each Move operations increases from 1 to

\[ m(o) = 1 + \frac{|W(o, A_{new}) - W(o, A_{old})|}{W(o, A_{new}) + W(o, A_{old})} \]
Decomposition A

Decomposition B

\[ m(5) = 1 + \frac{|4-1|}{4+1} = 1.6 \]
EdgeMoJo in practice

- Experiments with real and synthetic data indicated that EdgeMoJo distance is usually MoJo distance multiplied by a constant factor.
- The usefulness of edges in measuring similarity between partitions is still an open question.
What about nested decompositions?

- All measures we discussed so far assume a flat decomposition
  - No nested clusters
- Clustering algorithms typically create nested decompositions
- One needs to flatten decompositions to use these measures
Creating compact flat decompositions

- All objects are re-assigned to the first level cluster they are transitively contained in
Creating detailed flat decompositions

- All clusters are re-assigned to the root
The END framework

- Each decomposition is transformed into a sequence of decompositions
- One for each level in the containment tree
- Compute the value of a flat evaluation measure $M$ at each level
- Obtain a vector of values $S_{Mi}$
The END framework

- Compute the combined similarity/distance $S$ as
  \[ \sqrt{\sum (w_i S_{M_i}^2)} \]
  where $\sum w_i = 1$

- Weights need to be assigned to each level
A different solution: UpMoJo

- An extension of MoJo distance that includes an Up operation as well
  - Moves an object one level higher in the containment tree

- A series of Up operations ensures that the top level of the containment tree contains the same set of objects

- MoJo rearranges the top level

- The process repeats for each subtree of the top level