Software Clustering
Using Bunch

Software Clustering
Environment

The Structure of a
Graphical Editor
**The Software Clustering Problem**

- **“Find a good partition of the Software Structure.”**
- A **partition** is the decomposition of a set of elements (i.e., all the nodes of the graph) into mutually disjoint clusters.
- A **good partition** is a partition where:
  - highly interdependent nodes are grouped in the same clusters
  - independent nodes are assigned to separate clusters

**Not all Partitions are Created Equal ...**

**Our Assumption**

- **“Well designed software systems are organized into cohesive clusters that are loosely interconnected.”**
**Problem:** There are too many partitions of the MDG...

The number of MDG partitions grows very quickly, as the number of modules in the system increases...

\[
S_{n,k} = \begin{cases} 
1 & \text{if } k = 1 \vee k = n \\
S_{n-1,k-1} + kS_{n-1,k} & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>k</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>11</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>30</td>
<td>55</td>
<td>30</td>
<td>12</td>
<td>90</td>
<td>180</td>
<td>144</td>
<td>877</td>
<td>203</td>
<td>877</td>
<td>55</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>12</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>13</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>14</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td>19</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

**A 15 Module System is about the limit for performing Exhaustive Analysis**

---

**Edge Types**

- With respect to each cluster, there are two different kinds of edges:
  - \(\mu\) edges (Intra-Edges) which are edges that start and end within the same cluster
  - \(\epsilon\) edges (Inter-Edges) which are edges that start and end in different clusters

---

**A Partition of the Structure of the Graphical Editor**

---

Distributed Objects
The Module Dependency Graph (MDG)

We represent the structure of the system as a graph, called the MDG.

- The nodes are the modules/classes.
- The edges are the relations between the modules/classes.

Example: The Structure of the *dot* System

Dot is a reasonably small system, but its structure is complex and hard to understand...
The clustered view highlights...

Identification of "special" modules (relations are hidden to improve clarity)

The Partitioned Module Dependency Graph (MDG)
The partitioned MDG contains clusters that group related nodes from the MDG.
- The clusters represent the subsystems.
- The subsystems highlight high-level features or services provided by the modules in the cluster.

The clustered view of dot (its partitioned MDG) is easier to understand than the unclustered view...
Step 1: Creating the MDG

1. The MDG can be generated automatically using source code analysis tools.
2. Nodes are the modules/classes, edges represent source-code relations.
3. Edge weights can be established in many ways, and different MDGs can be created depending on the types of relations considered.

Example: The MDG for Apache’s Regular Expression class library

Source Code

Analysis Tools

Acacia

Java

Automatic MDG Generation

We provide scripts on the SERG web page to create MDGs automatically for systems developed in C, C++, and Java.

The MDG eliminates details associated with particular programming languages.

Why do we want to Cluster?

- To group components of a system.
- Grouping gives some semantic information.
  - We find out which components are strongly related to each other.
  - We find out which components are not so strongly connected to each other.
  - This gives us an idea of the logical subsystem structure of the entire system.

Reverse Engineering (Bunch) © SERG
**Clustering useful in practice?**

- Documentation may not exist.
- As software grows, the documentation doesn’t always keep up to date.
- There are constant changes in who uses and develops a system.

**Our Approach to Automatic Clustering**

- “Treat automatic clustering as a searching problem”
  - Maximize an objective function that formally quantifies of the “quality” of an MDG partition.
  - We refer to the value of the objective function as the modularization quality (MQ)

  MQ is a Measurement and not a Metric

**Why automatic clustering?**

- Experts are not always available.
- Gives the previously mentioned benefits without costing much.
How does clustering help?

- Helps create models of software in the absence of experts.
  - Gives the new developers a ‘road map’ of the software’s structure.
  - Helps developers understand legacy systems.
  - Lets developers compare documented structure with some ‘inherent’ structure of the system.

Module Dependency Graph

- Represents the dependencies between the components of a system.
- Nodes are components:
  - Classes, modules, files, packages, functions…
- Edges are relationships between components:
  - Inherit, call, instantiate, etc…

Software Clustering Problem

- For a given mdg, and a function that determines the quality of a partitioning of that mdg, find the best partition.
Possible Answer?

- We could look at every possible partition...
  - Exhaustive search always gives the best answer.
  - The number of partitions of a set are equal to the recurrence given by Sterling:
    \[
    S(n, k) = \begin{cases} 
    1 & \text{if } k = 1 \text{ or } k = n \\
    S(n - 1, k - 1) + k S(n - 1, k) & \text{otherwise}
    \end{cases}
    \]

Exhaustive Search...

- If Exhaustive search is optimal, then why not use it?
  - The recurrence to the Sterling recurrence grows very rapidly.
  - The search for partitioning for \( n \) nodes must go through partitions for all \( k \) in \( S(n, k) \). Thus the size of the search space is:
    \[
    \sum_{k=1}^{n} S(n, k) \quad \text{enormous}
    \]

Other Answers

- If we don’t use Exhaustive search, then we have to use a sub-optimal strategy.
- Just because a strategy is sub-optimal, though, doesn’t mean it won’t give good results.
Hill-Climbing

- Hill-Climbing refers to how to travel along the contour of the solution space.
- The idea is to find a starting position, and then move to a better state.
  - What if there is more than one state that we could move to?
  - What happens when we find a local maximum?
  - Can we do anything about getting a bad start?
**Bunch Hill Climbing Clustering Algorithm**

Generate a Random Decomposition of MDG

Iteration Step

Generate Next Neighbor

Measure MQ

Compare to Best Neighboring Partition

Better?

Best Neighboring Partition for Iteration

Convergence

Reverse Engineering (Bunch) © SERG

---

**Hill-Climbing Algorithm Features**

We have implemented a family of hill-climbing algorithms:

- Control the "steepness" of the climb
- Simulated Annealing
- Population Size

Reverse Engineering (Bunch) © SERG

---

**Genetic Algorithms**

- Genetic Algorithms take a problem, and model solution finding after nature.
  - Selection
  - Crossovers
  - Mutations
- Genetic Algorithms tend not to get stuck at local maxima.
**Bunch Genetic Clustering Algorithm (GA)**

Generate a Starting Population from the MDG

Iteration Step

Current Population

Next Population

Mutation Operation

Crossover Operation

Best Partition from Final Population

All Generations Processed

**Tools for Clustering**

- **Bunch** – Automatic Clustering
  - Performs automatic clustering.
  - Automatically produced results are often suboptimal.
  - Allows user-directed clustering:
    - Users can help the search for a good clustering with their knowledge.

**Using Bunch...**

- Bunch is a tool written in Java that will work on any machine supporting the JVM.

- The first step in using Bunch will be finding out how to make those mdg files mentioned earlier.
Creating mdg files...

- To create an mdg, we use the mdg script:
  - Used to create mdg files for Java, C, and C++.
  - Can view various relations (extends, implements, method-variable, method-method).
  - Can also give weights to each relation, along with providing its type.

Syntax of mdg

```
mdg --(java|c|c++) --(eivmA)+ --(wltA)*
```

- The first set of parameters represents the language to use.
- The second set of parameters describes the types of relationship to display.
- The last set determines what extra information to produce.

Syntax of bmdg

```
bmdg -f <file> -i -a -v
```

- The first parameter specifies the XML file created by bat during source code analysis.
- The -i (optional) parameter includes the “class A subclasses or implements class B” relations in the mdg.
- The -a (optional) parameter includes the “class A calls method in class B” and “class A accesses public variable in class B” relations in the mdg.
- The -v (optional) parameter includes the “class A has variables of type class B” relations in the mdg.
**MDG Results**

- The mdg script produces a series of pairs that represent directed edges.
- These edges can have extra information along with them, such as weight and type, if desired.

**Bunch Features**

- Bunch is a tool to perform automatic clustering of module dependency graphs.
  - Has various algorithms for finding a good solution.
  - Allows users to assist in automatic clustering.
  - Provides methods to exclude omnipresent libraries and consumers in clustering (provides a more succinct clustering).

**The Bunch Algorithms**

- Provides use for exhaustive search.
- Has hill-climbing algorithms
  - Steepest Ascent (greedy)
  - Nearest Ascent
- A genetic algorithm is also available.
Hill-Climbing Algorithms

• Hill-Climbing algorithms get their names because they ‘climb’ to the top of ‘the hills’ of a solution space.
  – Given the current solution, and some transition function, find some better solution.
  – Sub-optimal: Can get caught on one of ‘the smaller hills’ a.k.a. local maximum.

Steepest Ascent

• Steepest Ascent is a ‘greedy’ algorithm.
  – Greedy Algorithms take the most of what they want whenever possible.
  – Steepest Ascent computes all the states it can transition to, and takes the best one.
  – Greedy algorithms aren’t always optimal overall, even though they take the optimal step at each state.

Nearest Ascent

• Nearest Ascent is a non-greedy hill-climbing algorithm
  – Does not compute the set of all possible states to transition to.
  – Computes states until a better one is found.
  – Much faster in practice than Steepest Ascent.
Helping Hill-Climbers

- Hill-Climbing algorithms are inherently sub-optimal.
  - They still tend to give good results, though.
  - Even with good results, there are ways of improving the quality of the results, without that much effort (relatively).

Bunch Clustering

- Bunch, as it may already seem, views clustering as an optimization problem.
- The bunch objective function trades off between two things:
  - Cohesiveness of clusters (how intra-related is a cluster?)
  - Coupling of clusters (how inter-related are clusters?)

Random Restart

- One way to help Hill-Climbing algorithms is to restart the algorithm at some random state after finding a local maximum.
  - The idea is to get various local maxima.
  - After finding a set of various solutions, return the best one.
  - The cost isn’t really that much, in comparison to the cost in terms of system size.
Simulated Annealing

- Simulated Annealing is another technique used to assist Hill-Climbing algorithms.
  - Occasionally take a transition that is not an increase in quality.
  - The hope is that this will lead to climbing a better hill.

Genetic Algorithms

- A totally different approach is to use Genetic Algorithms.
  - Genetic Algorithms tend to converge to a solution quickly when the solution space is small relative to the search space.
  - Genetic Algorithms tend to provide good results.
  - Genetic Algorithms don’t get stuck at local maxima.

Genetic Algorithms...

- Genetic Algorithms work by representing each state as a string, and then doing ‘genetic manipulation’ on a ‘generation’ in ways analogous to nature.
  - Crossing Over
  - Mutations
  - Selection
**Measuring MQ – Step 1: The Cluster Factor**

The Cluster Factor for cluster \( i \), \( CF_i \), is:

\[
CF_i = \begin{cases} 
0 & \text{if } \mu_i = 0 \\
\frac{2\mu_i}{2\mu_i + \sum (\epsilon_{ij}, \epsilon_{ji})} & \text{otherwise}
\end{cases}
\]

\( CF \) increases as the cluster’s cohesiveness increases.

---

**Modularization Quality (MQ):**

\[
MQ = \sum_k CF_i
\]

- \( k \) represents the number of clusters in the current partition of the MDG.

- Modularization Quality (MQ) is a measurement of the “quality” of a particular MDG partition.

---

**Modularization Quality (MQ):**

- We have implemented a family of MQ functions.
- MQ should support MDGs with edge weights.
- Faster than older MQ (Basic MQ)
  - TurboMQ \( \approx O(|V|) \)
  - ITurboMQ \( \approx O(1) \)
- ITurboMQ incrementally updates, instead of recalculates, the \( CF_i \)
The Clustering Problem: Algorithm Objectives

“Find a good partition of the MDG.”

- A partition is the decomposition of a set of elements (i.e., all the nodes of the graph) into mutually disjoint clusters.
- A good partition is a partition where:
  - highly interdependent nodes are grouped in the same clusters
  - independent nodes are assigned to separate clusters
- The better the partition the higher the MQ.

The Bunch Tool: Automatic Clustering

Starting Bunch

- Now that we know some of the stuff that Bunch does ‘under the hood,’ let’s start to use it.
  - Bunch is a Java program held in a jar file. To run it, type the following:
    
    `java -classpath Bunch.jar bunch.Bunch`
Basic Controls

Basic Controls Description

- Input Graph File: The mdg to be clustered.
- Clustering Method: Which algorithm to use? (Hill-Climbing, Genetic, etc).
- Output Cluster File: Where to put the results?
- Output File Format: Dotty? Text?

Clustering Actions

- Agglomerative Clustering: Bunch will cluster until the system is together in one large cluster.
- User-Driven Clustering: Here, the user clicks ‘Generate Next Level’ to proceed with each level of clustering.
Clustering Options Description

- Clustering Algorithm: Which ‘clustering’ algorithm to use.
- Graph File Delimiters: Specify delimiting characters in MDG files.
- Limiting Runtime: Used to limit the amount of time Bunch will run.
- Agglomerative Output Options: What type of output for agglomerative clustering.
- Generate Tree Format: Show clustering hierarchy.
Libraries

- Some Modules are just libraries used by the system.
  - These don’t add any semantics to the system.
  - While being a part of what makes the system, they are just small tools that the designers decided to use.

Omni-Present Modules

- Some systems have internal modules that are use across a large portion of the system.
  - Including such modules in a decomposition is pointless.
  - Bunch gives the option of removing such modules from the decomposition.
Sometimes, there will exist *a priori* knowledge about a system.

- Such knowledge, when used as constraints on a decomposition, makes it a better decomposition.
- Bunch allows pre-cluster grouping of modules.

Results...
**System Evolution**

- Systems change constantly.
  - New developers add to chaos in the maintenance.
  - New features do so, too.

**Out with the old?**

- Even though systems change, out of date decompositions don’t have to be thrown away.
  - There still may be some relations that the decomposition represents.
  - An old decomposition may be better than some random start state for clustering a system.

**Orphan Adoption**

- When a change is applied to one module of a system, the amount of change is relatively small.
  - We can take the module that changed, and see how it fits in with every cluster.
  - We can also see how good of a system we get when it’s alone.
  - Taking the best solution out of these solutions is how Bunch handles this.
Hierarchical Clustering (1):
Tree View

1. 

2. Default 

3. 

4. 

Hierarchical Clustering (2):
Standard View

1. 

2. Default 

3. 

4. 

Distributed Clustering

Distributed clustering allows large MDGs to be clustered faster...

Bunch User Interface (BUI) 
Bunch Clustering Service (BCS) 
Neighboring Servers (NS)
Distributed Clustering

Distributed clustering allows large MDGs to be clustered faster...

Distributed Clustering Features
- Multiple Clustering Algorithms
- Heterogeneous Platforms
- Adaptive Load Balancing

Bunch User Interface (BUI)  Bunch Clustering Service (BCS)  Neighboring Servers (NS)

Reverse Engineering (Bunch) © SERG

‘Good’ Clusterings

- Even with all these features, only the Exhaustive Search gives us the optimal clustering.
- How do we know how good of a clustering we have?