DREXEL UNIVERSITY

A Heuristic Search Approach to Solving the Software Clustering Problem

Brian S. Mitchell

Software Engineering Research Group

Math & Computer Science Department

Drexel University

Outline

- Motivation & Background
- Search Based Software Clustering (Bunch)
- Evaluating Clustering Results
- Summary & Future Work

Background

- Software clustering simplifies program maintenance and program understanding
- Software clustering techniques help developers fix defects (maintenance), or add a features (program understanding) to existing software systems

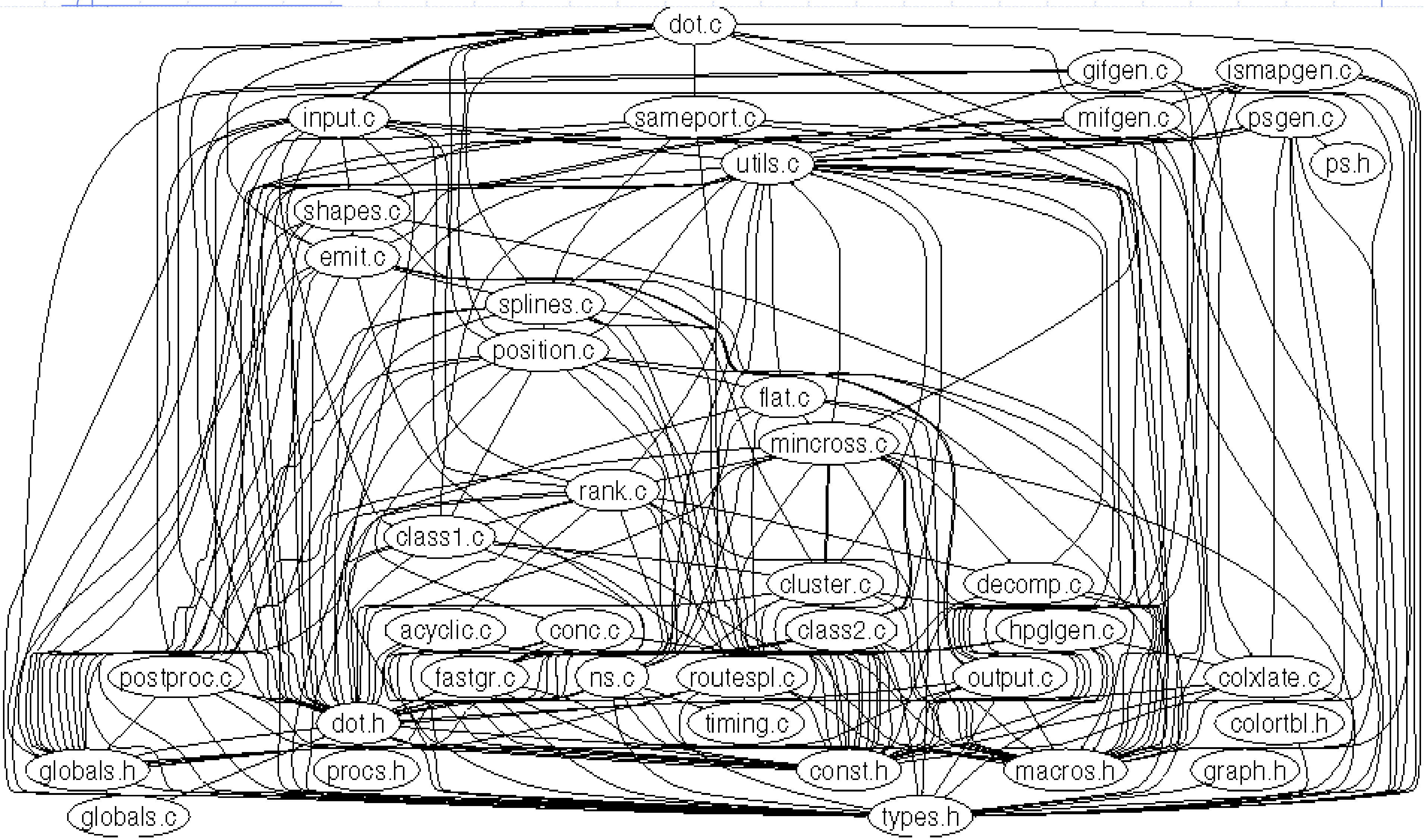
Understanding the Software Structure

- When fixing or extending a software system
  - Desirable to change as few of the existing modules/classes as possible
- Requires an understanding of the system's overall structure

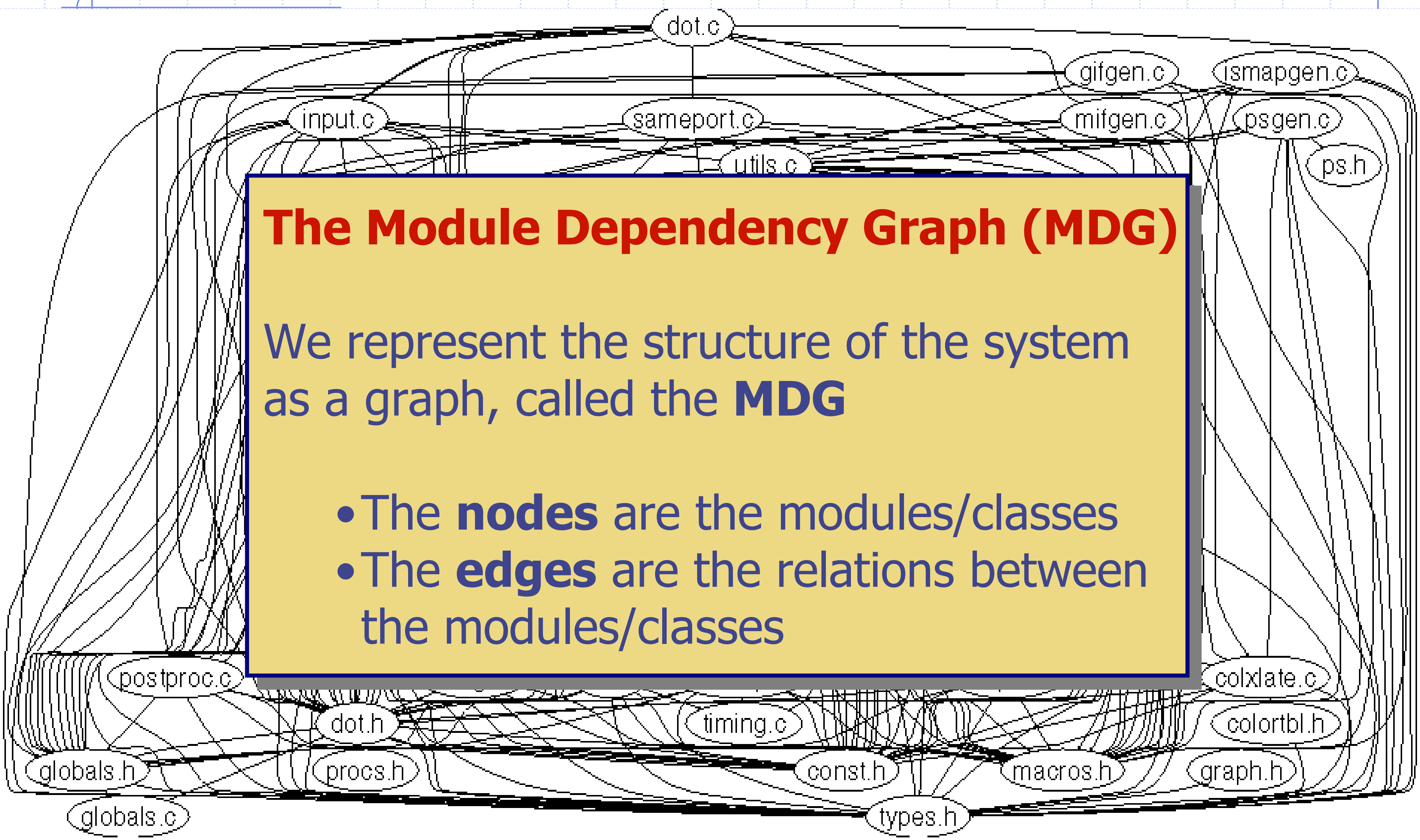
Problem 1: The structure is complex and often not documented for large systems

Problem 2: Ad hoc changes to the source code tend to deteriorate the system's structure over time

Example: The Structure of the **dot** System



Example: The Structure of the **dot** System

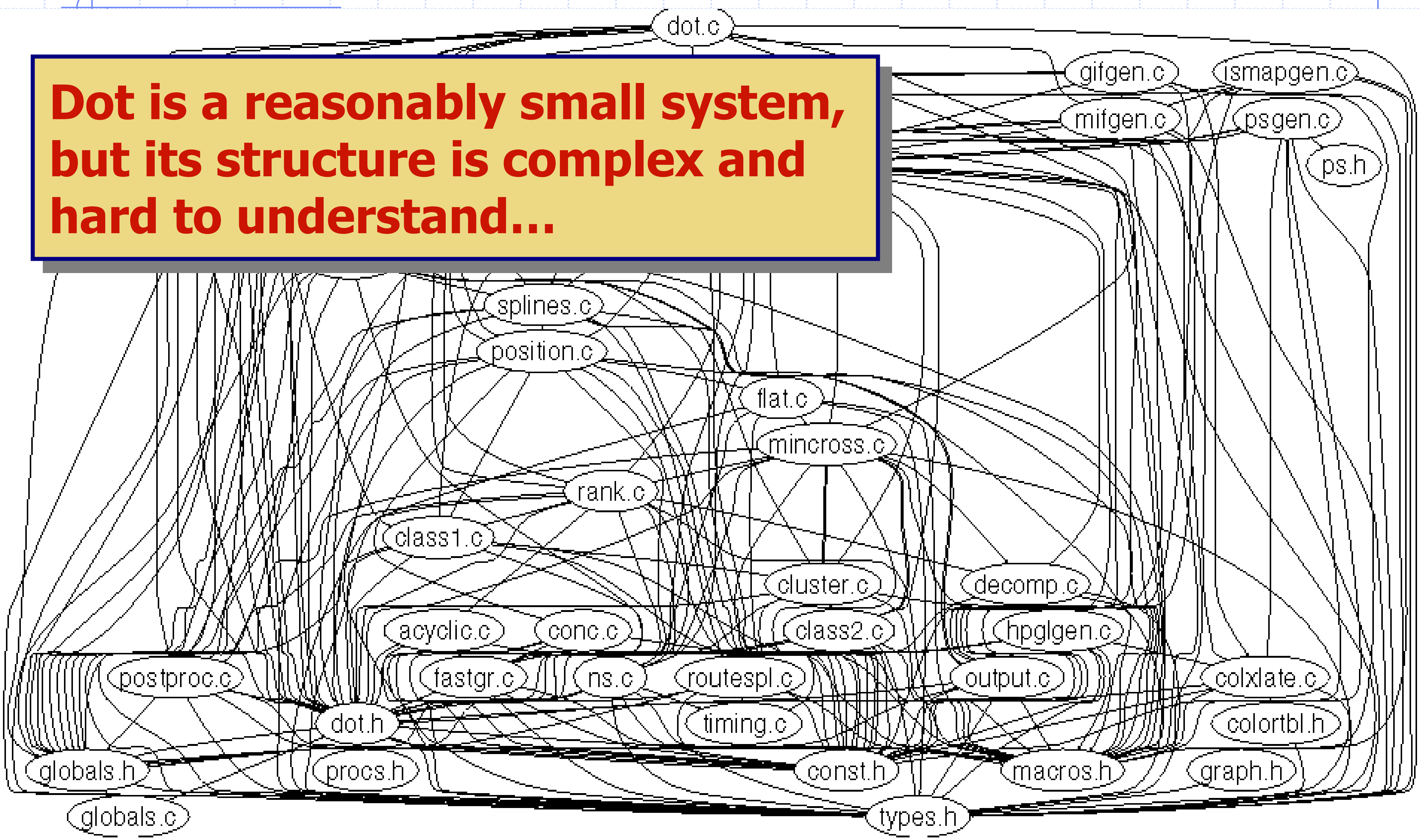


The Module Dependency Graph (MDG)

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

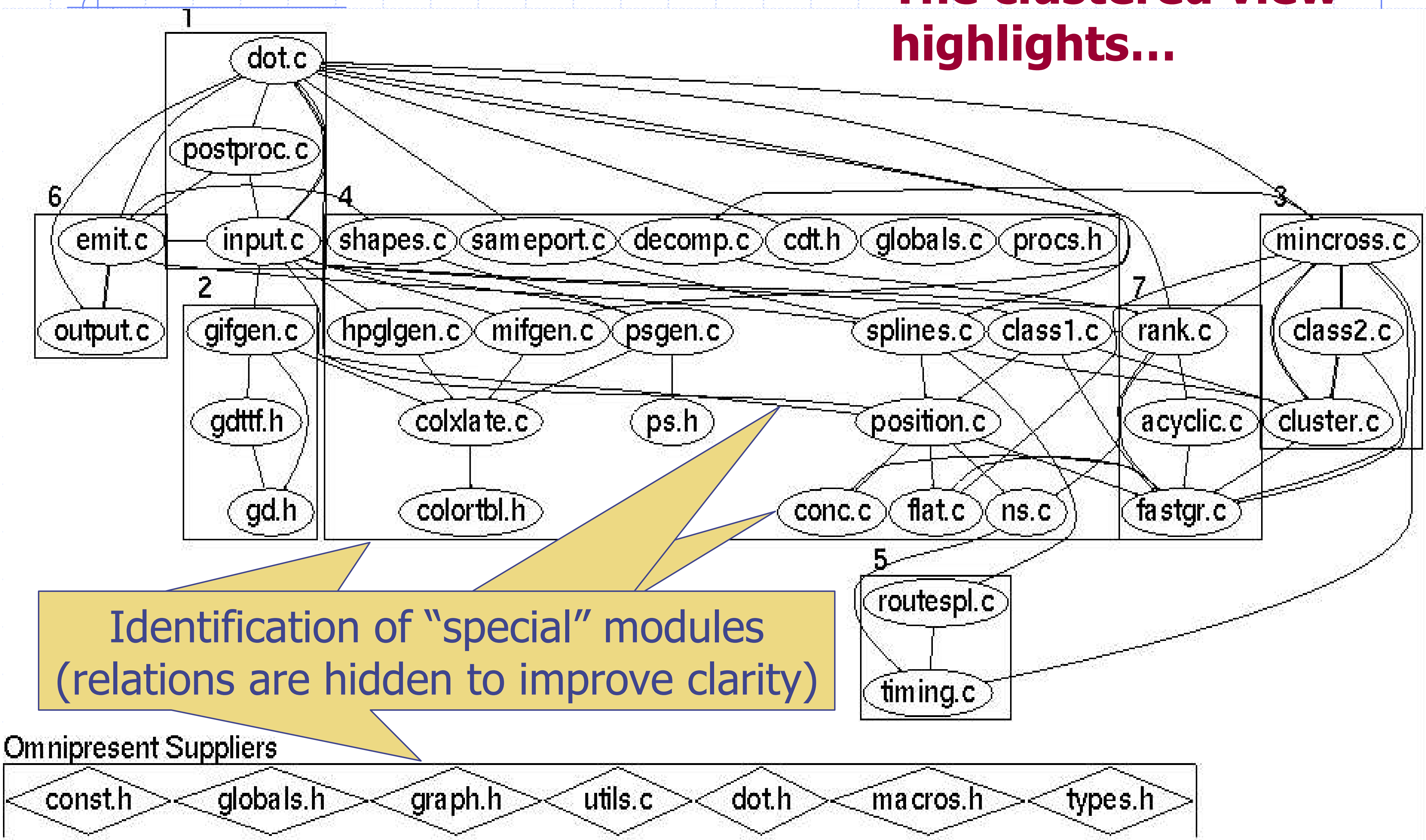
Example: The Structure of the **dot** System



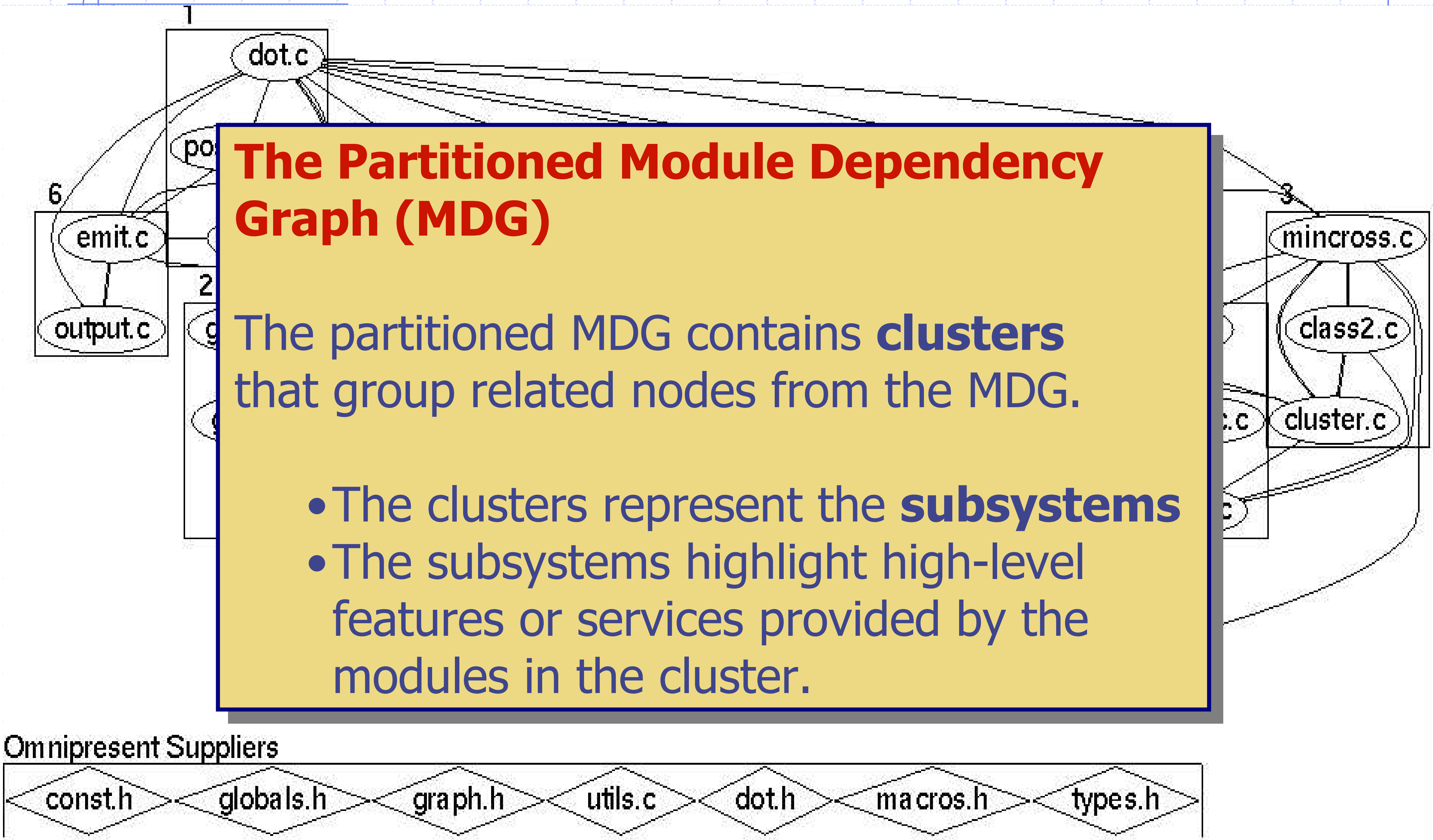
Dot is a reasonably small system, but its structure is complex and hard to understand…

The **dot** System after Clustering and Filtering

The clustered view highlights...



The **dot** System after Clustering and Filtering

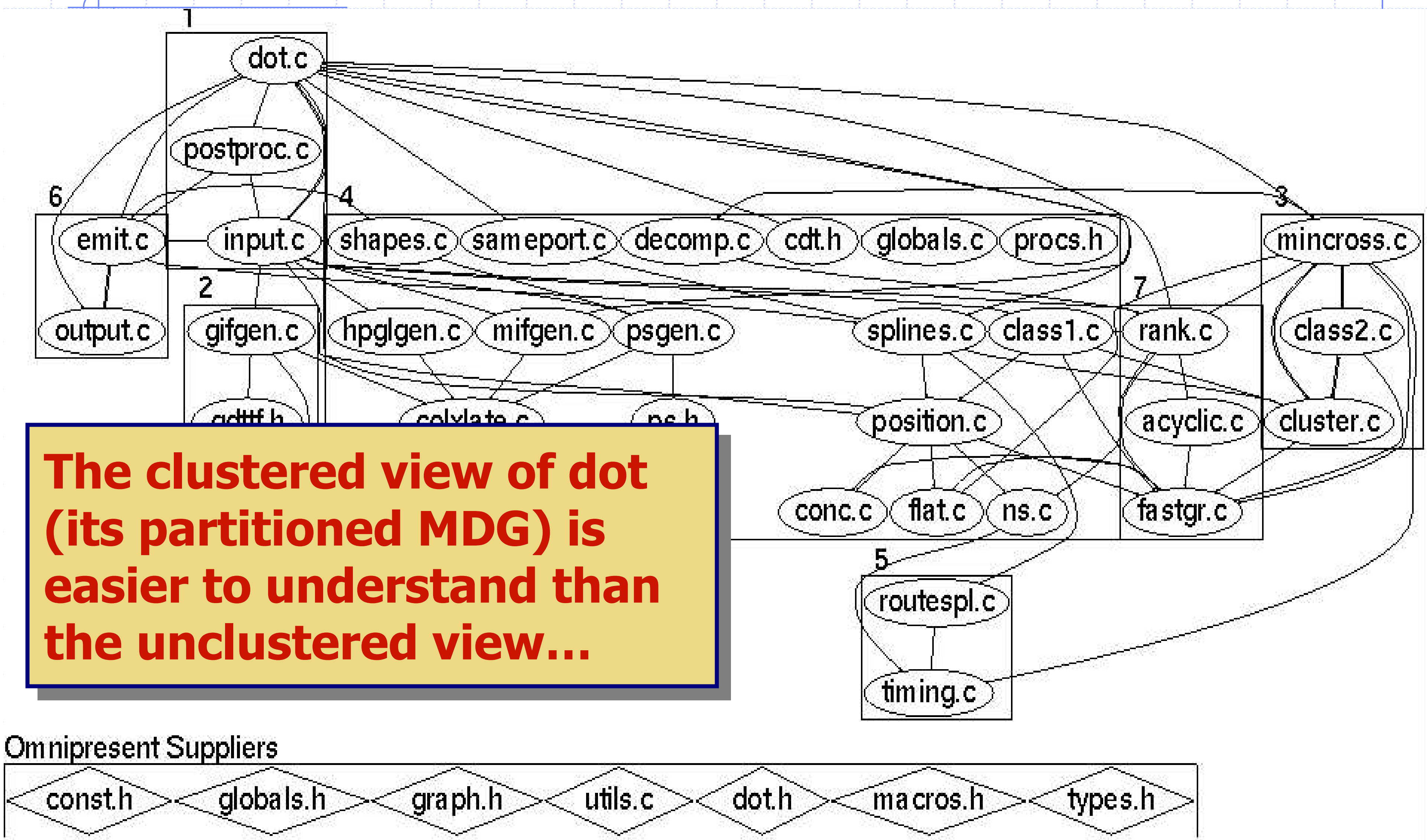


The Partitioned Module Dependency Graph (MDG)

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 **dot** System after Clustering and Filtering



The clustered view of dot (its partitioned MDG) is easier to understand than the unclustered view…

Clustering Techniques

- A variety of techniques for software clustering have been studied by the reverse engineering community:
  - Source code component similarity (or dissimilarity)
  - Concept Analysis
  - Subsystem Patterns
  - Implementation-Specific Information

Our clustering approach uses search algorithms

Software Clustering

Using Search Algorithms…

Step 1: Creating the MDG

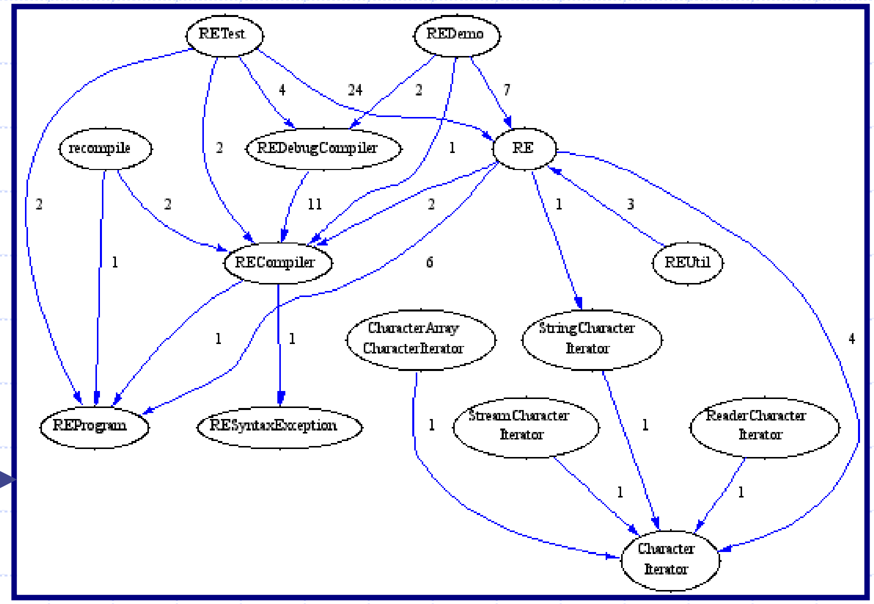
Source Code

```
void main()
{
 printf("hello");
}
```

Source Code Analysis Tools

Acacia **Chava**

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



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

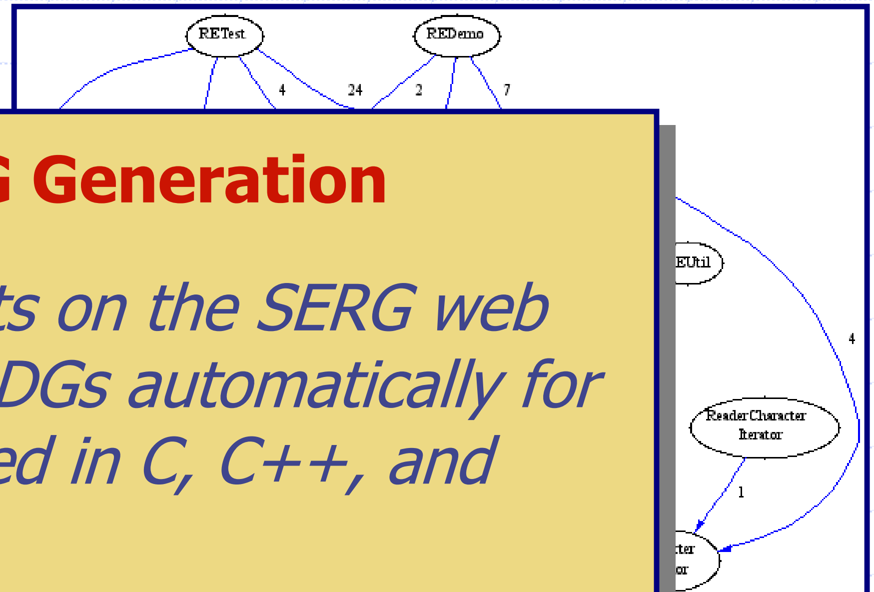
Step 1: Creating the MDG

Source Code

```
void main()
{
 printf
}
```

Sourc
Analys
Acacia

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



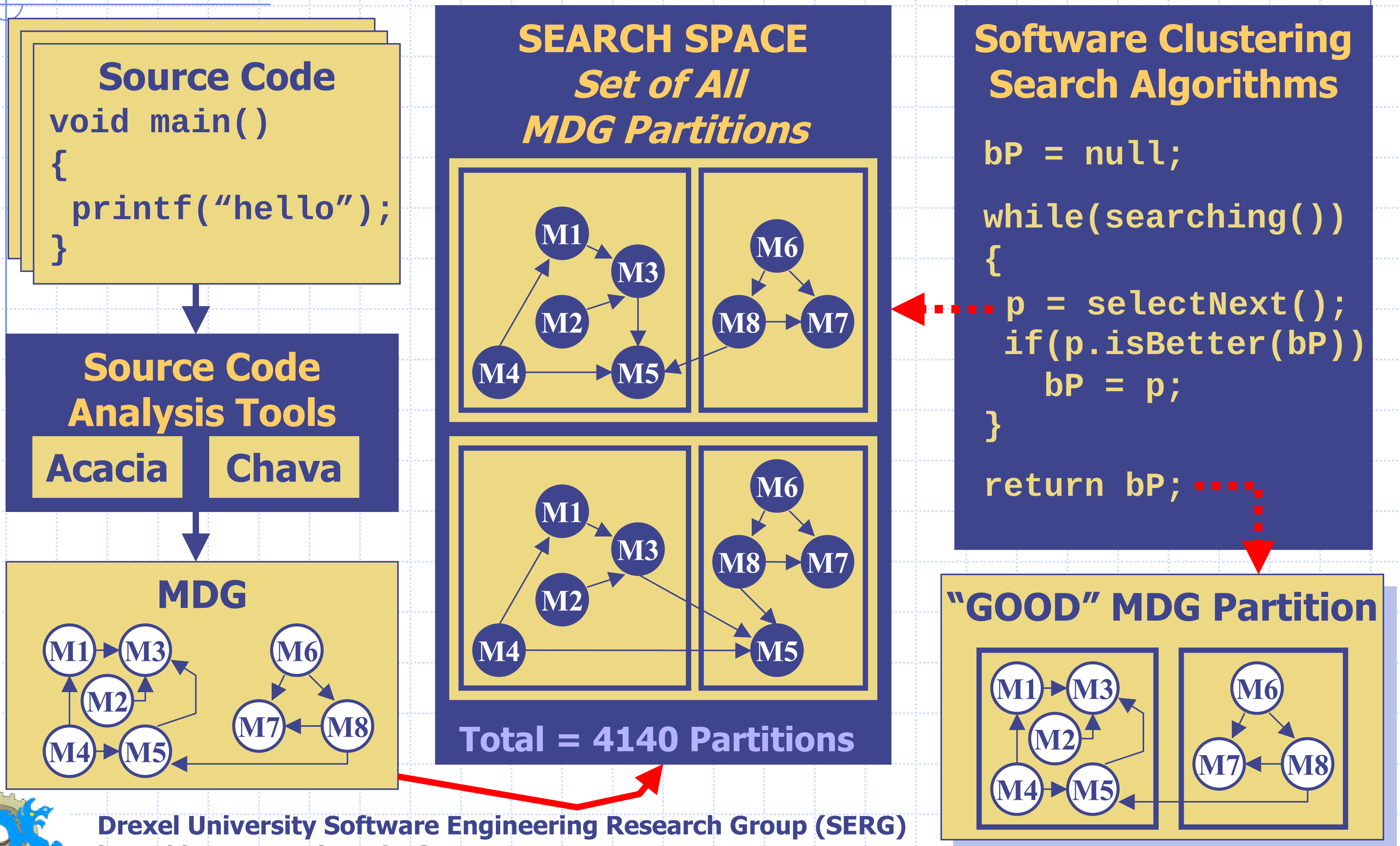
Automatic MDG Generation

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

1. The MDG … analysis tools
2. Nodes are … relations
3. Edge weig… t MDGs can be created depending on the types of relations considered

Software Clustering with Search Algorithms

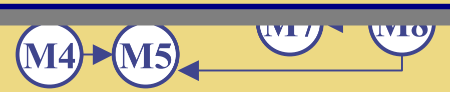


Software Clustering with Search Algorithms

Source Code

```
void main()
```

SEARCH SPACE
Set of All MDG Partitions



Total = 4140 Partitions

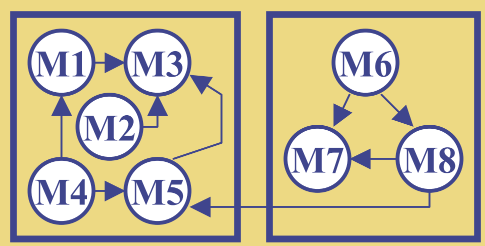
Search Algorithm Requirements

- Must be able to compare one partition to another objectively.
- We define the Modularization Quality (**MQ**) measurement to meet this goal.
  - ***Given partitions P1 & P2, MQ(P1) > MQ(P2) means that P1 "is better than" P2***

Software Clustering Search Algorithms

```
bP = null;
while(searching())
{
  p = selectNext();
  if(p.isBetter(bP))
    bP = p;
}
return bP;
```

"GOOD" MDG Partition



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…

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

1 = 1	6 = 203	11 = 678570	16 = 10480142147
2 = 2	7 = 877	12 = 4213597	17 = 82864869804
3 = 5	8 = 4140	13 = 27644437	18 = 682076806159
4 = 15	9 = 21147	14 = 190899322	19 = 5832742205057
5 = 52	10 = 115975	15 = 1382958545	20 = 51724158235372

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

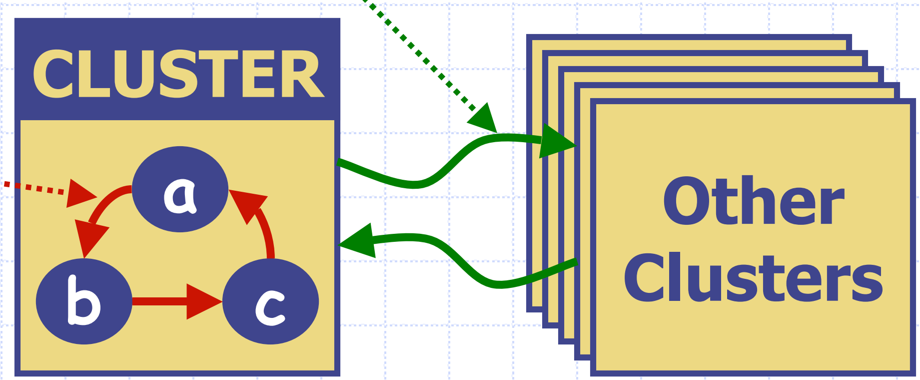
Our Approach to Automatic Clustering

- ***"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*

Edge Types

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

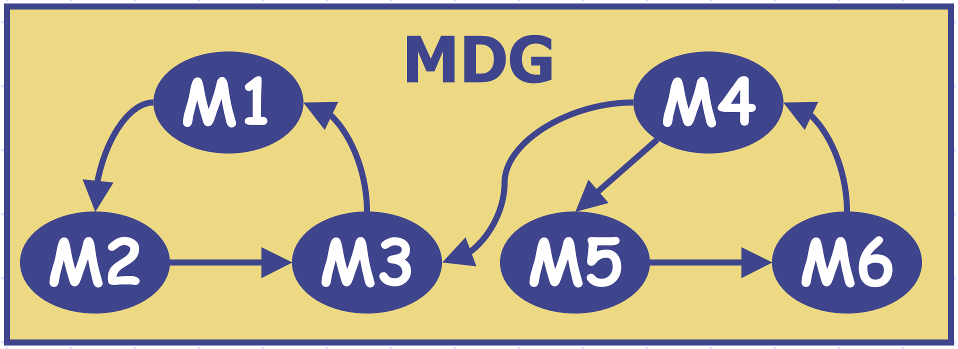


Our Assumption…

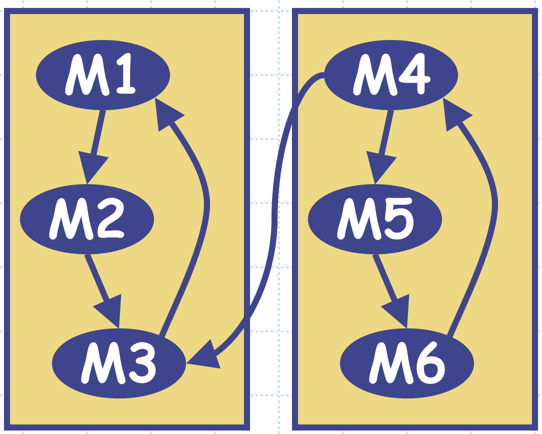
"Well designed software systems are organized into cohesive clusters that are loosely interconnected."

- The MQ measurement design must:
  - Increase as the weight of the intra-edges increases
  - Decrease as the weight of the inter-edges increases

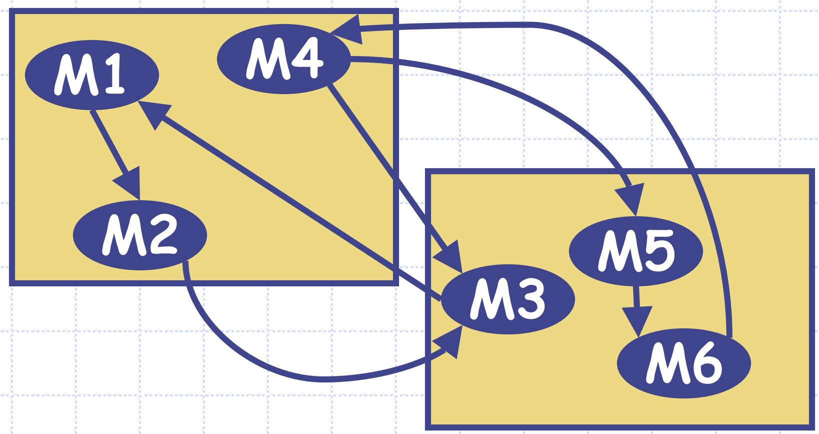
Not all Partitions are Created Equal ...



Good Partition!



Bad Partition!



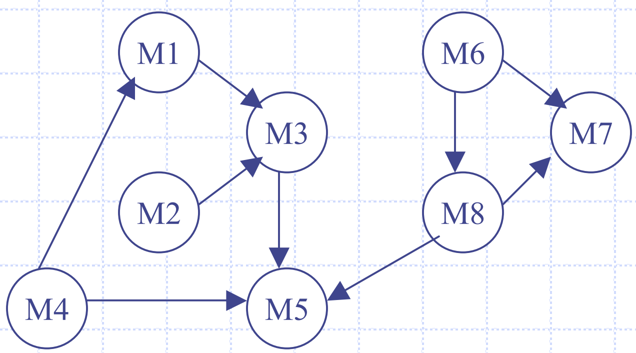
MQ(Good Partition) > MQ(Bad Partition)

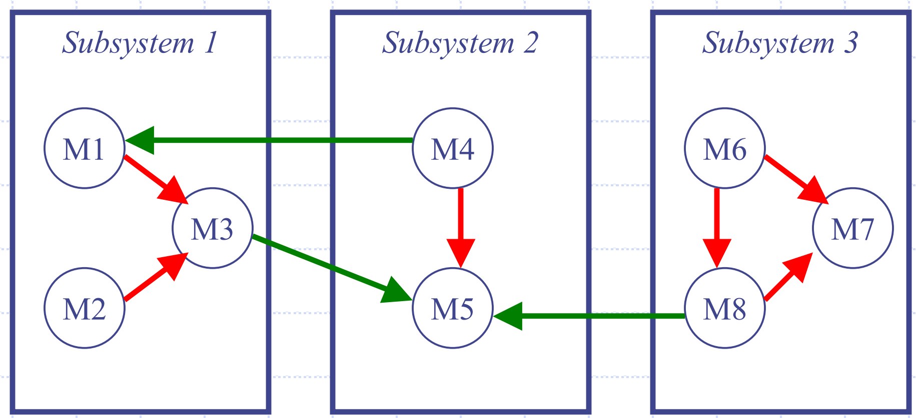
Measuring MQ – Step 1: The Cluster Factor

The Cluster Factor for cluster *i*, CF_i, is:

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \dfrac{2\mu_i}{2\mu_i + \sum\limits_{\substack{j=1 \\ j \neq i}}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & \textit{otherwise} \end{cases}$$

CF increases as the cluster's cohesiveness increases





$CF_1 = 4/6$ $CF_2 = 2/5$ $CF_3 = 6/7$

Modularization Quality (MQ):

$$MQ = \sum_{i=1}^{k} CF_i$$

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):

k re… e
curr…

◆ *Mo…*
mea…
par…

MQ

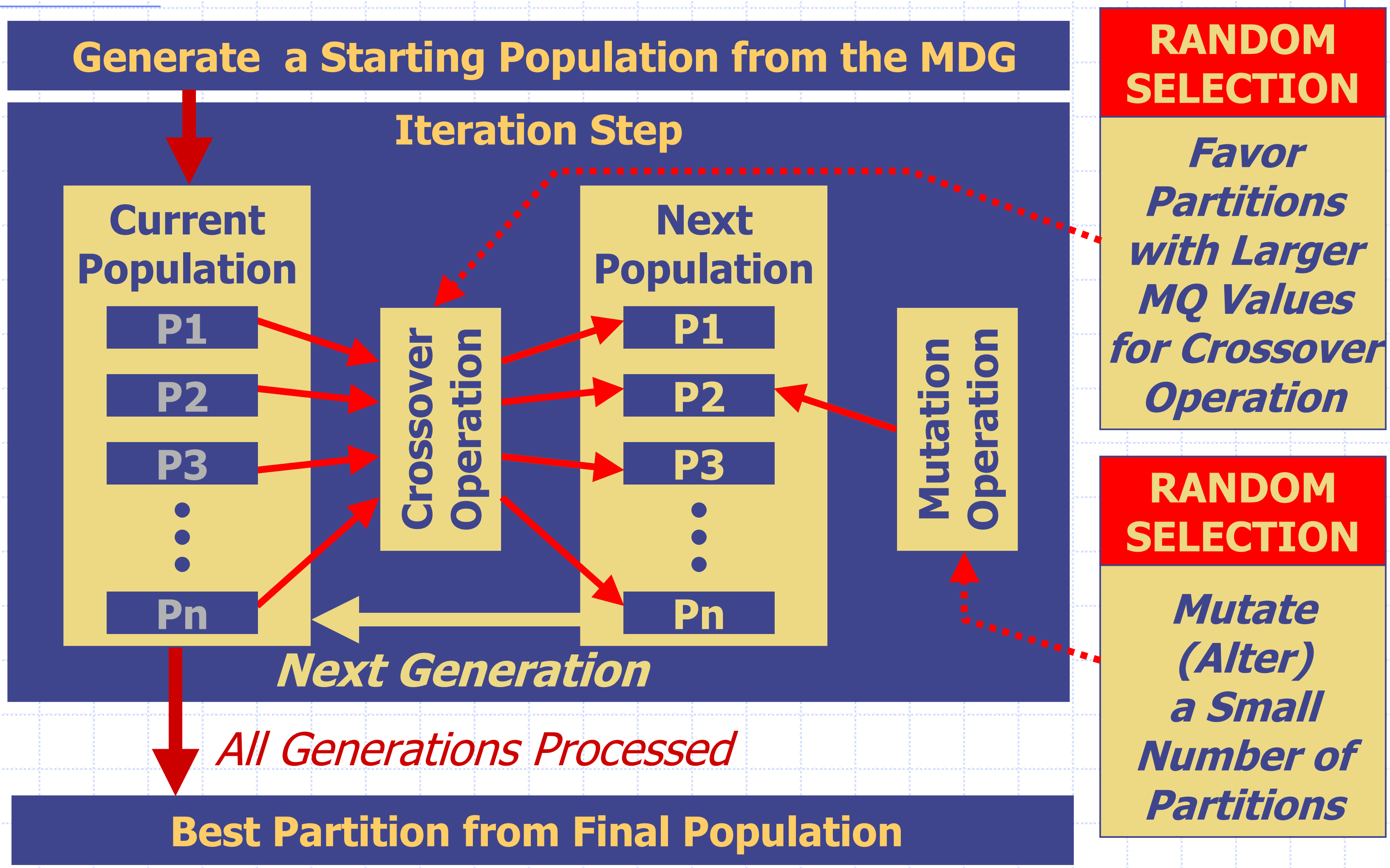
- We have implemented a family of MQ functions.
- MQ should support MDGs with edge weights
- Faster than older MQ (Basic MQ)
  - ***TurboMQ*** $\approx$ **O(|V|)**
  - **ITurboMQ** $\approx$ **O(1)**
- ITurboMQ incrementally updates, instead of recalculates, the $\mathbf{CF_i}$

The Software Clustering Problem: Algorithm Objectives

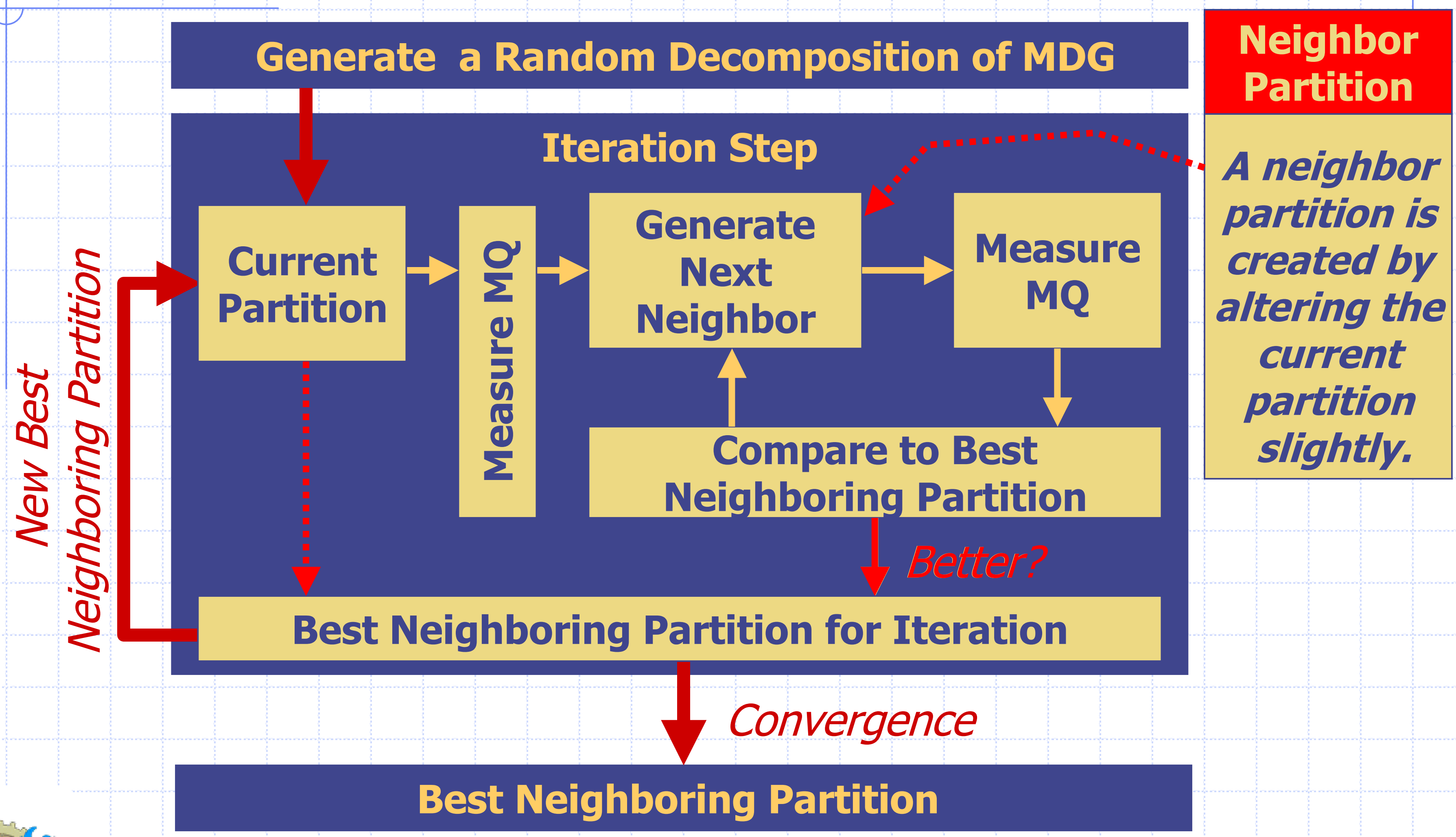
*"Find a **good** partition of the MDG."*

- 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**

Bunch Genetic Clustering Algorithm (GA)



Bunch Hill Climbing Clustering Algorithm



Bunch Hill Climbing Clustering Algorithm

Generate a Random Decomposition of MDG

Iteration Step

New Best Neighboring Partition

Hill-Climbing Algorithm Features

We have implemented a family of hill-climbing algorithms

- Control the “steepness” of the climb
- Simulated Annealing
- Population Size

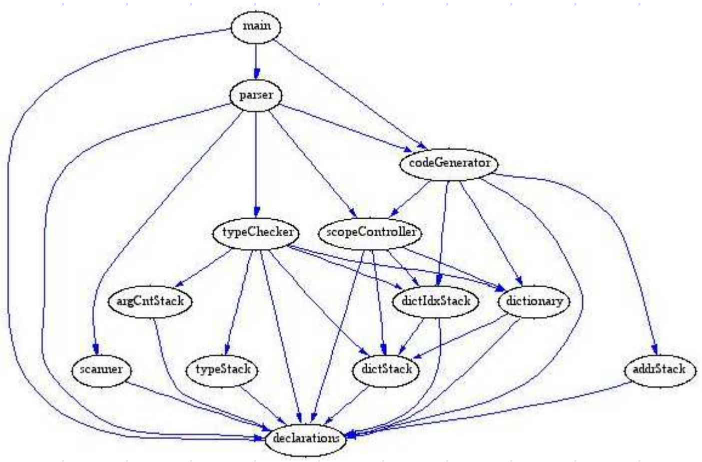
Best Neighboring Partition

Neighbor Partition

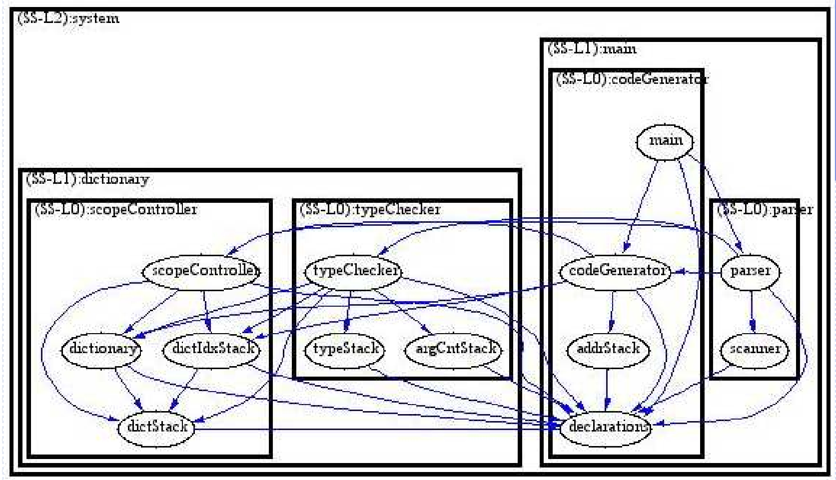
A neighbor partition is created by altering the current partition slightly.

Hierarchical Clustering (1): Tree View

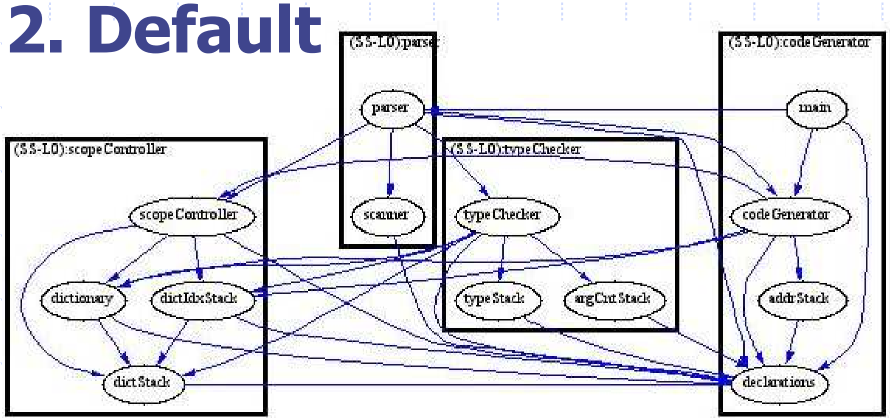
1.



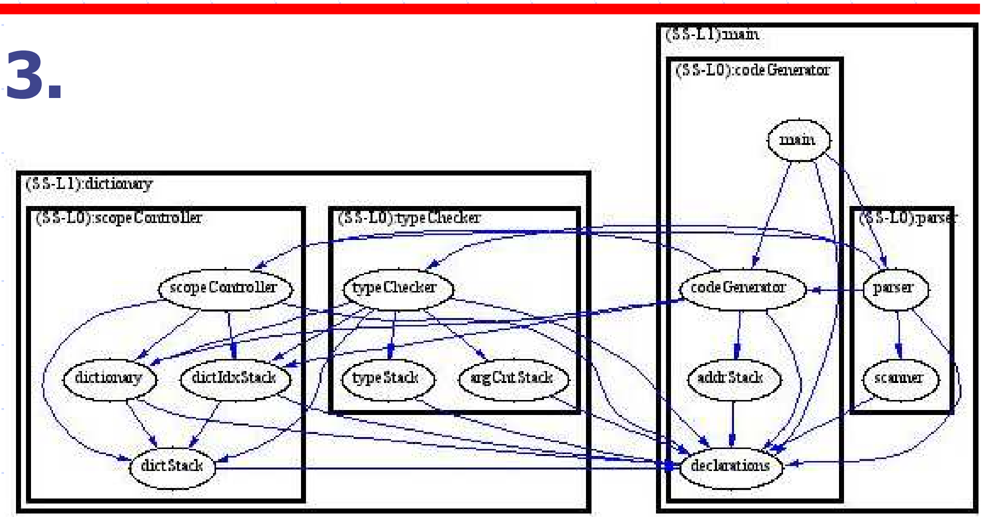
4.



2. Default

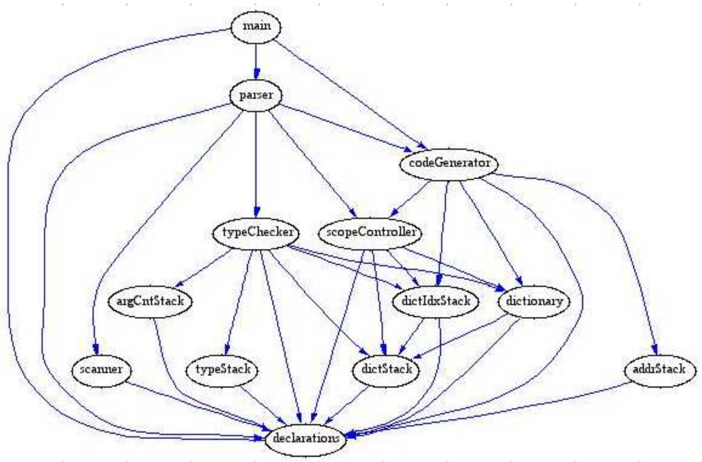


3.

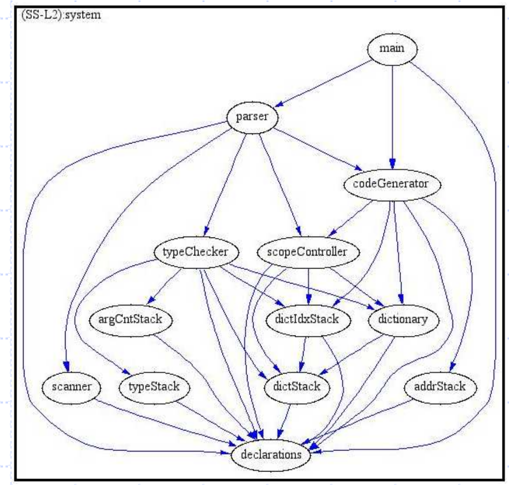


Hierarchical Clustering (2): Standard View

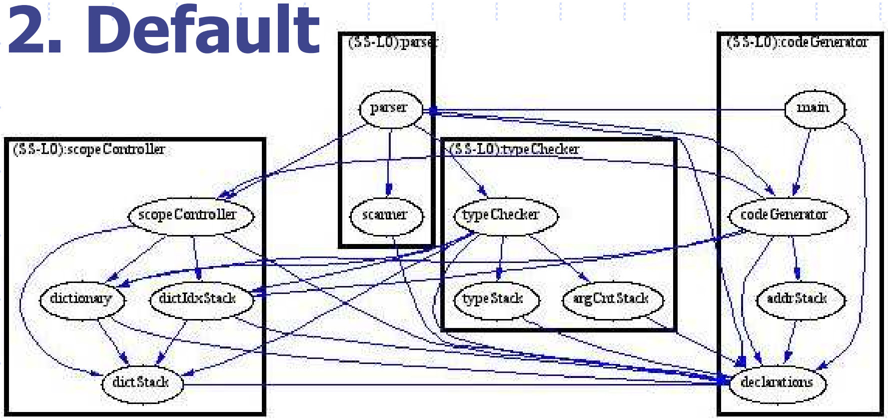
1.



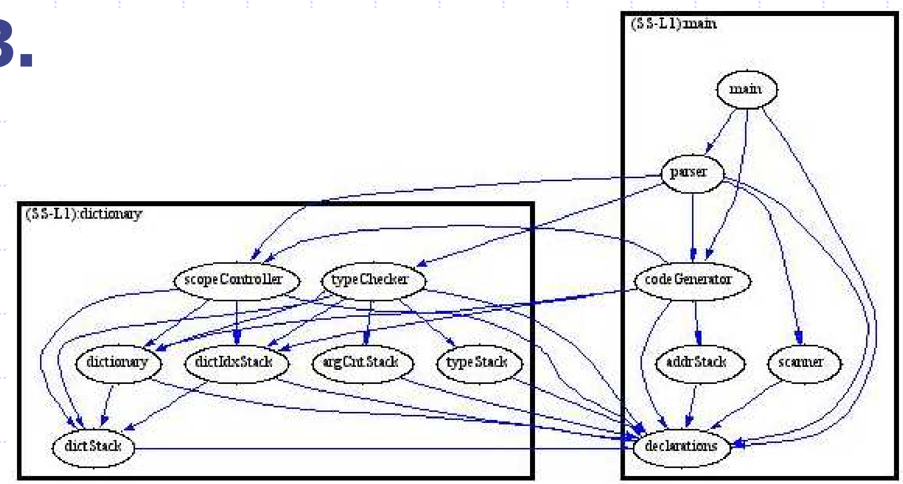
4.



2. Default



3.

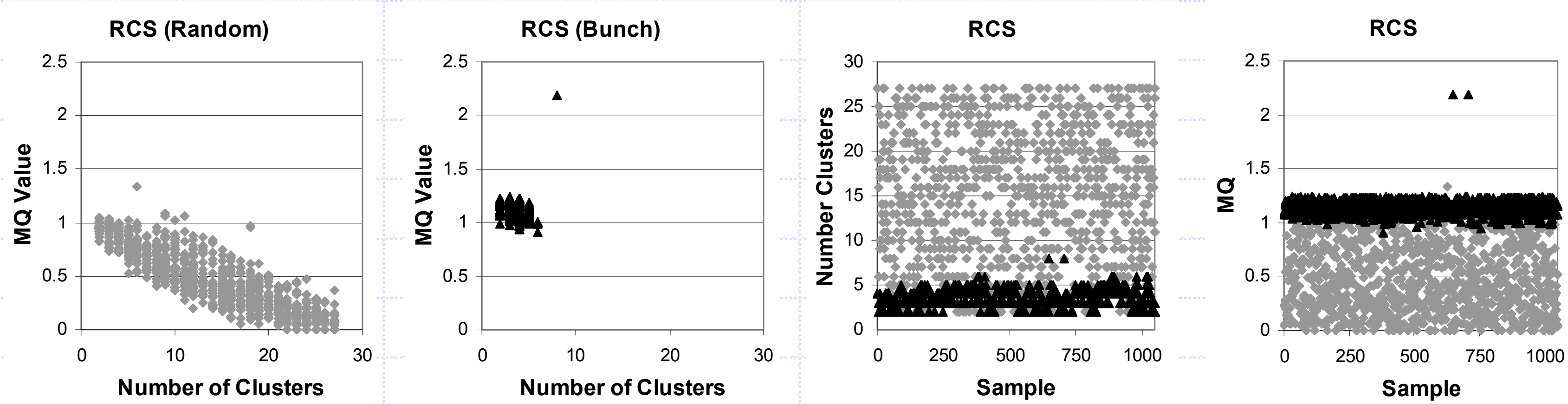


Drexel University Software Engineering Research Group (SERG)
http://serg.mcs.drexel.edu

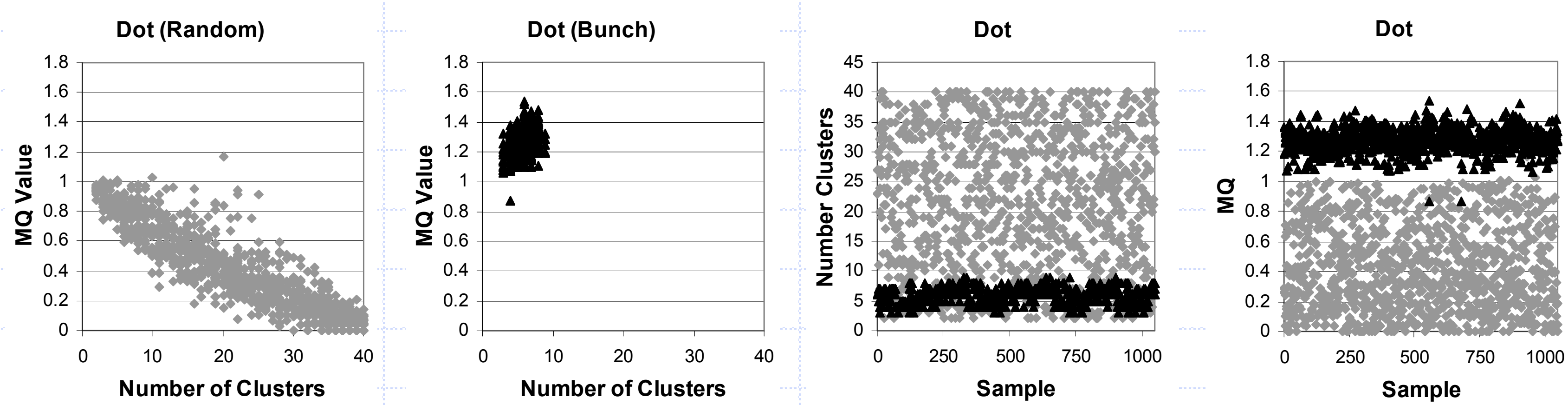
Some Results (1)...

◆ Random
▲ Bunch

RCS



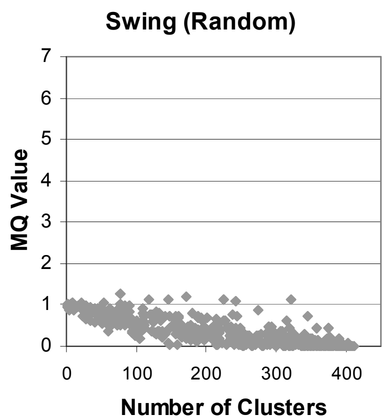
Dot

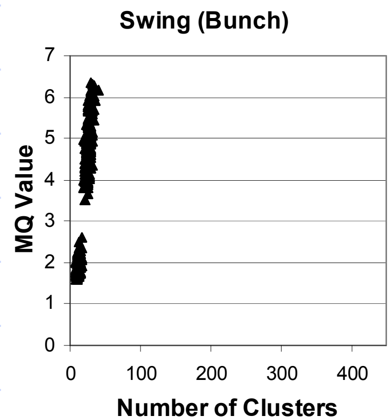


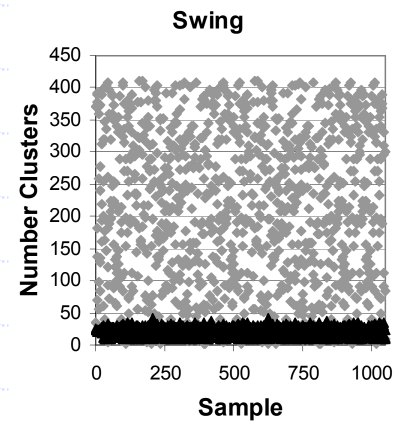
Some Results (2)...

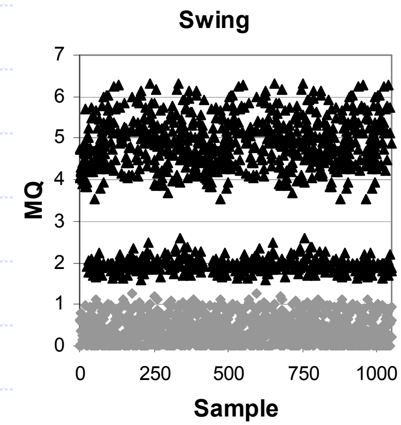
Random
Bunch

Swing

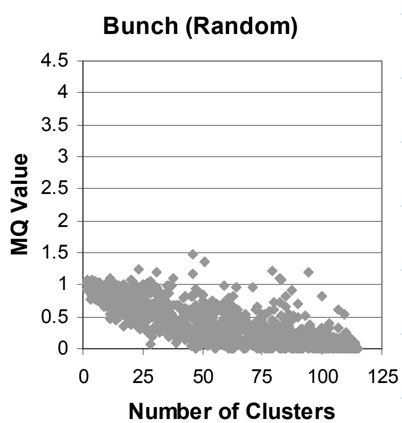


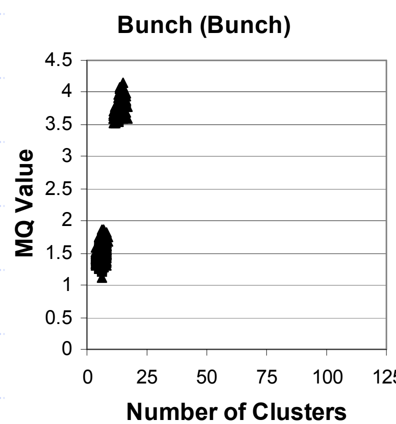


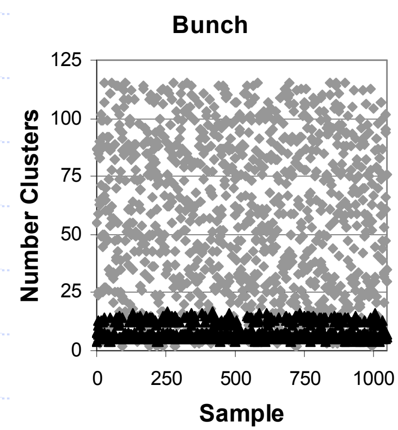


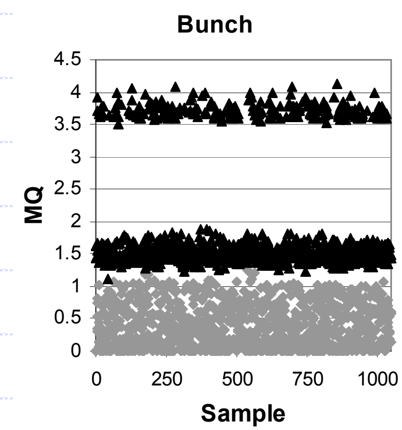


Bunch



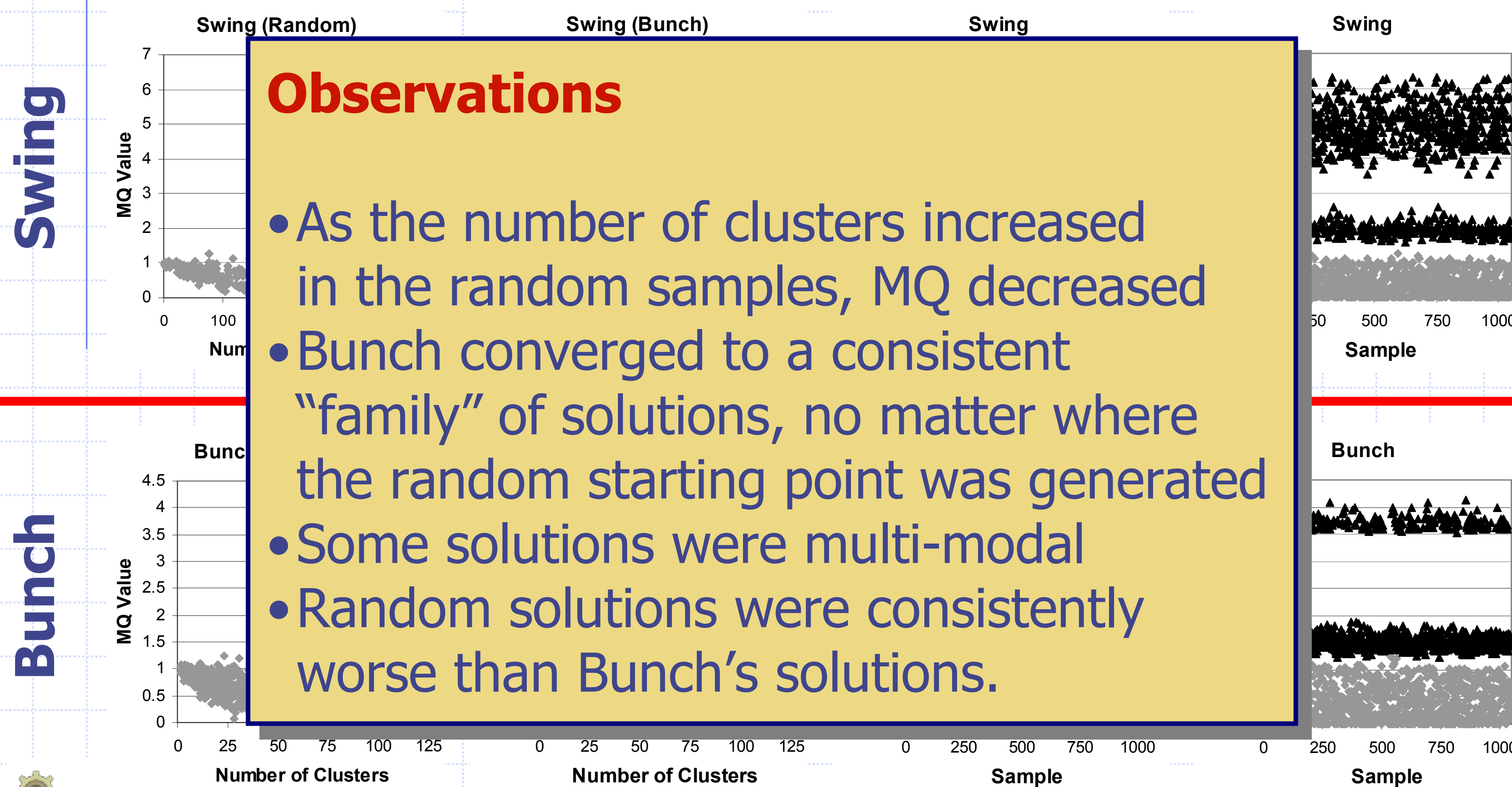






Some Results (2)…

Random
Bunch

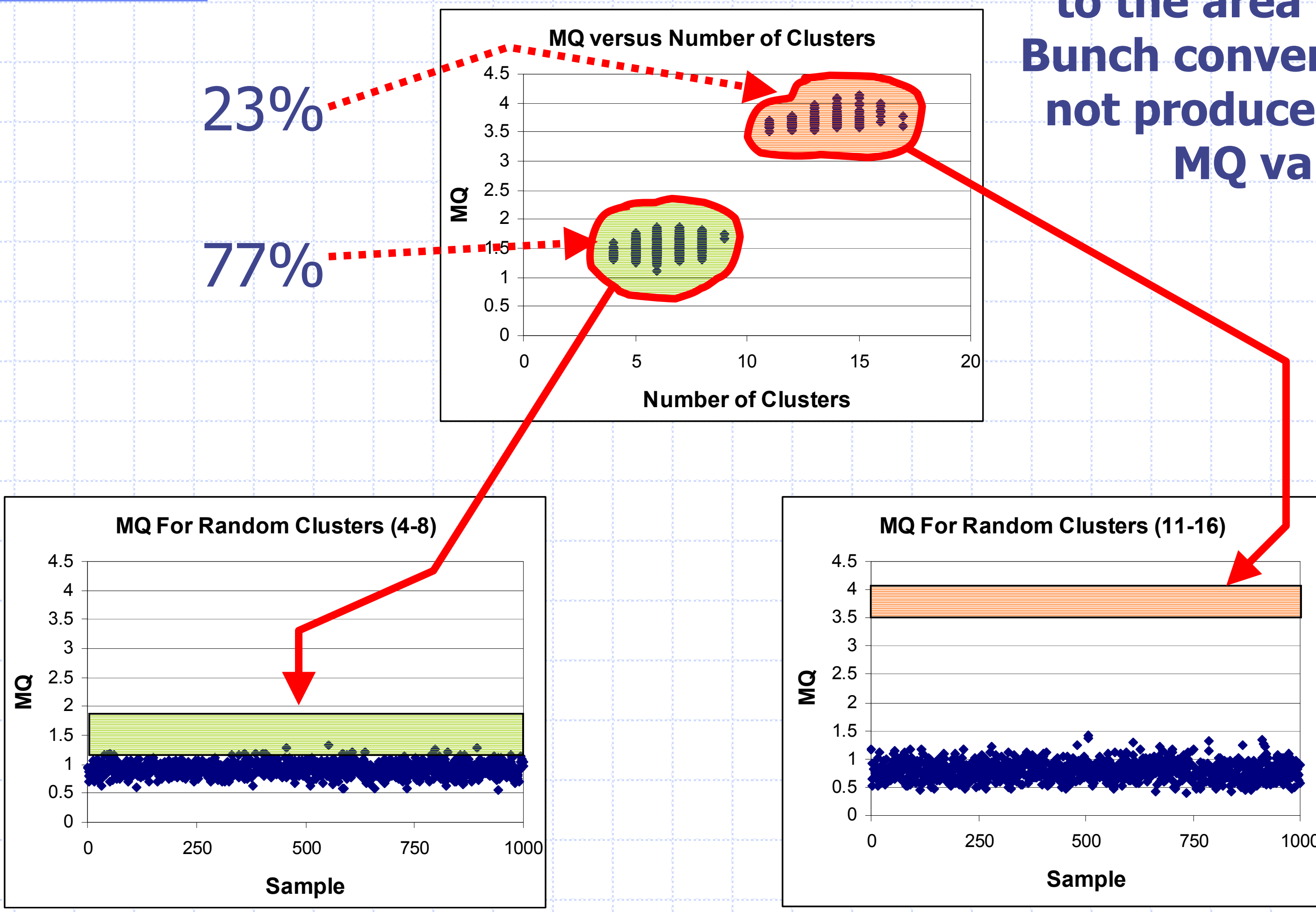


Observations

- As the number of clusters increased in the random samples, MQ decreased
- Bunch converged to a consistent "family" of solutions, no matter where the random starting point was generated
- Some solutions were multi-modal
- Random solutions were consistently worse than Bunch's solutions.

Example - Detailed Results: Bunch System

The search space has some inherent structure, as random clusters constrained to the area where Bunch converged did not produce better MQ values.



Software Clustering Tools

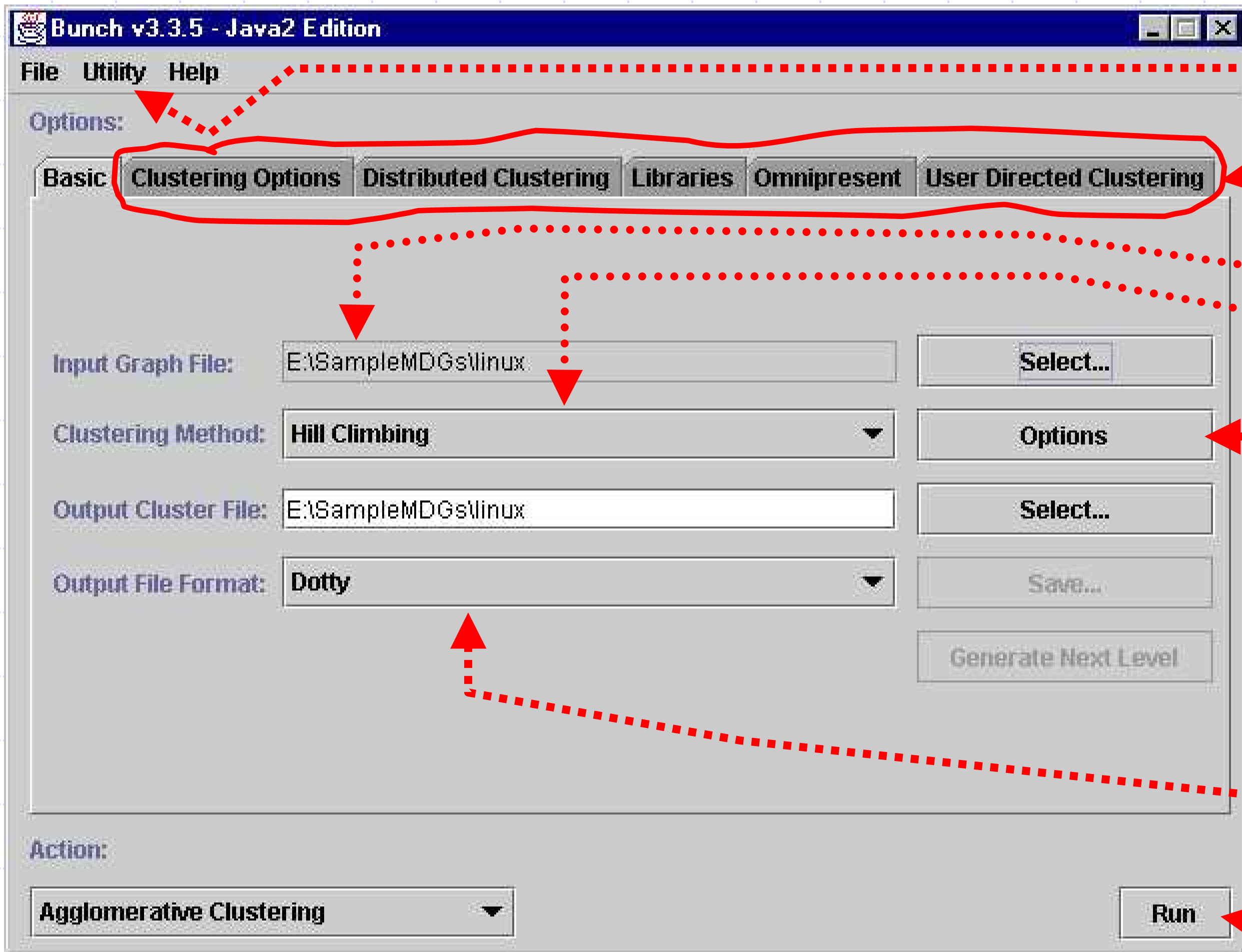
The Bunch Clustering Tool

The Bunch Clustering Tool

- Developed in Java
- Provides a GUI and an API
- Supports automatic and semi-automatic clustering
- Provides source code analysis tools
- "Pluggable" clustering algorithms

Bunch was designed to be used and extended by other researchers

The Bunch Tool: Automatic Clustering



Other Tools and Utilities

Clustering Services

MDG

Clustering Algorithm

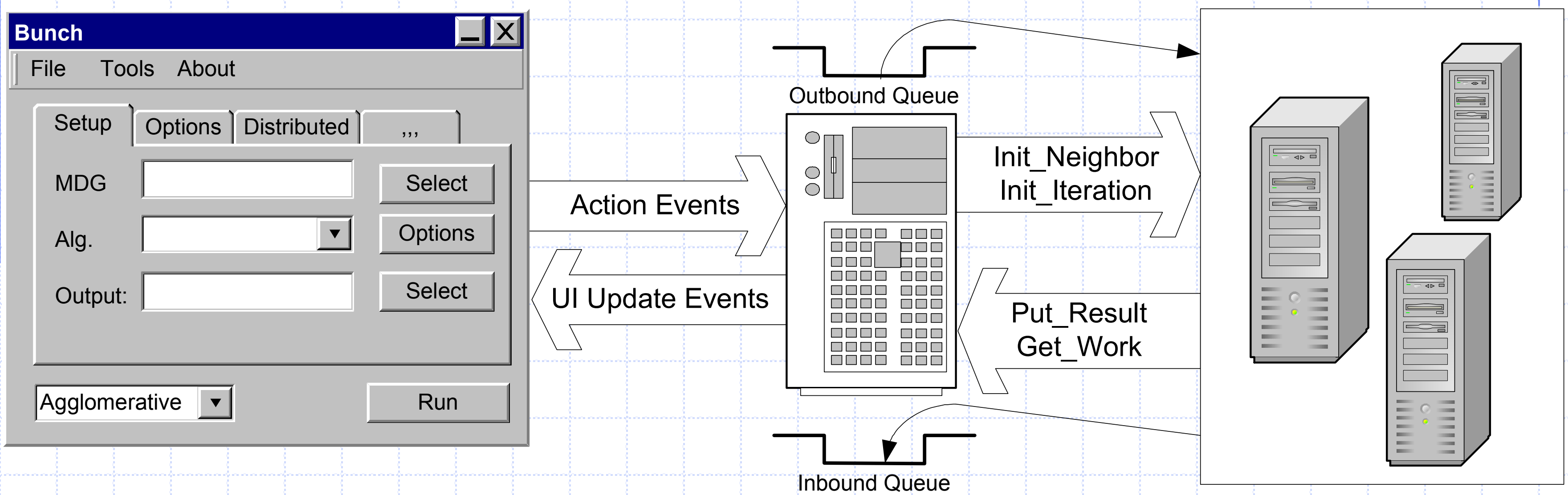
Clustering Algorithm Options

Output Format

START

Distributed Clustering

Distributed clustering allows large MDGs to be clustered faster…



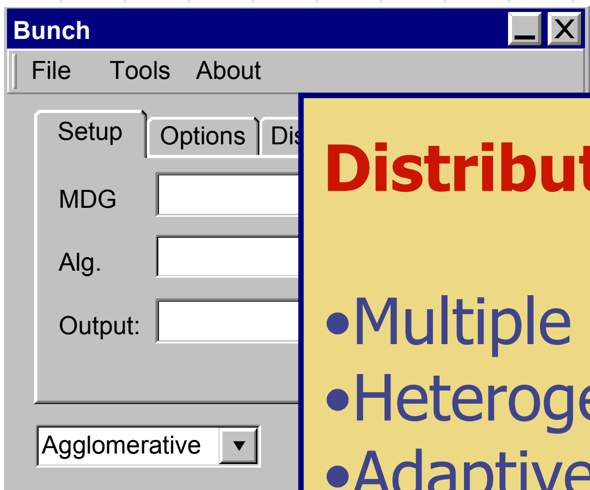
Bunch User Interface (BUI)

Bunch Clustering Service (BCS)

Neighboring Servers (NS)

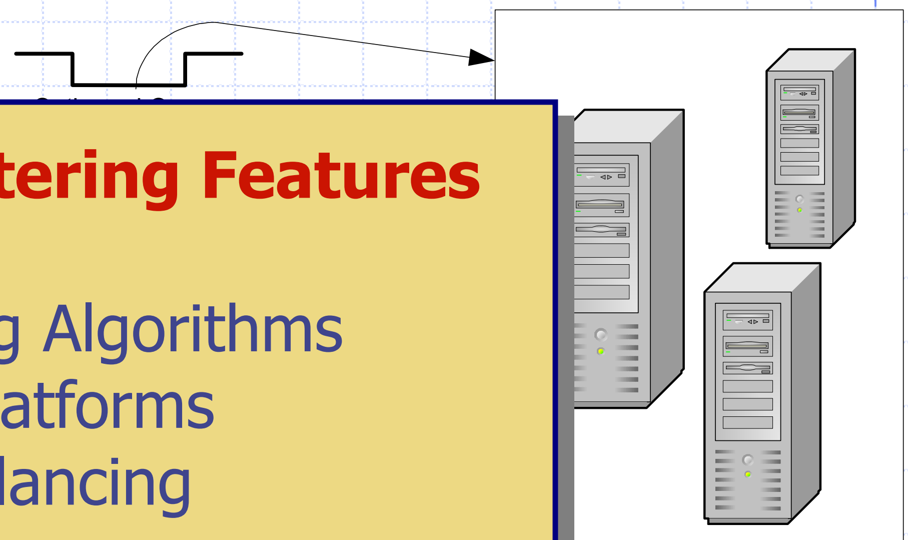
Distributed Clustering

Distributed clustering allows large MDGs to be clustered faster…



Distributed Clustering Features

- Multiple Clustering Algorithms
- Heterogeneous Platforms
- Adaptive Load Balancing



Bunch User Interface (BUI)

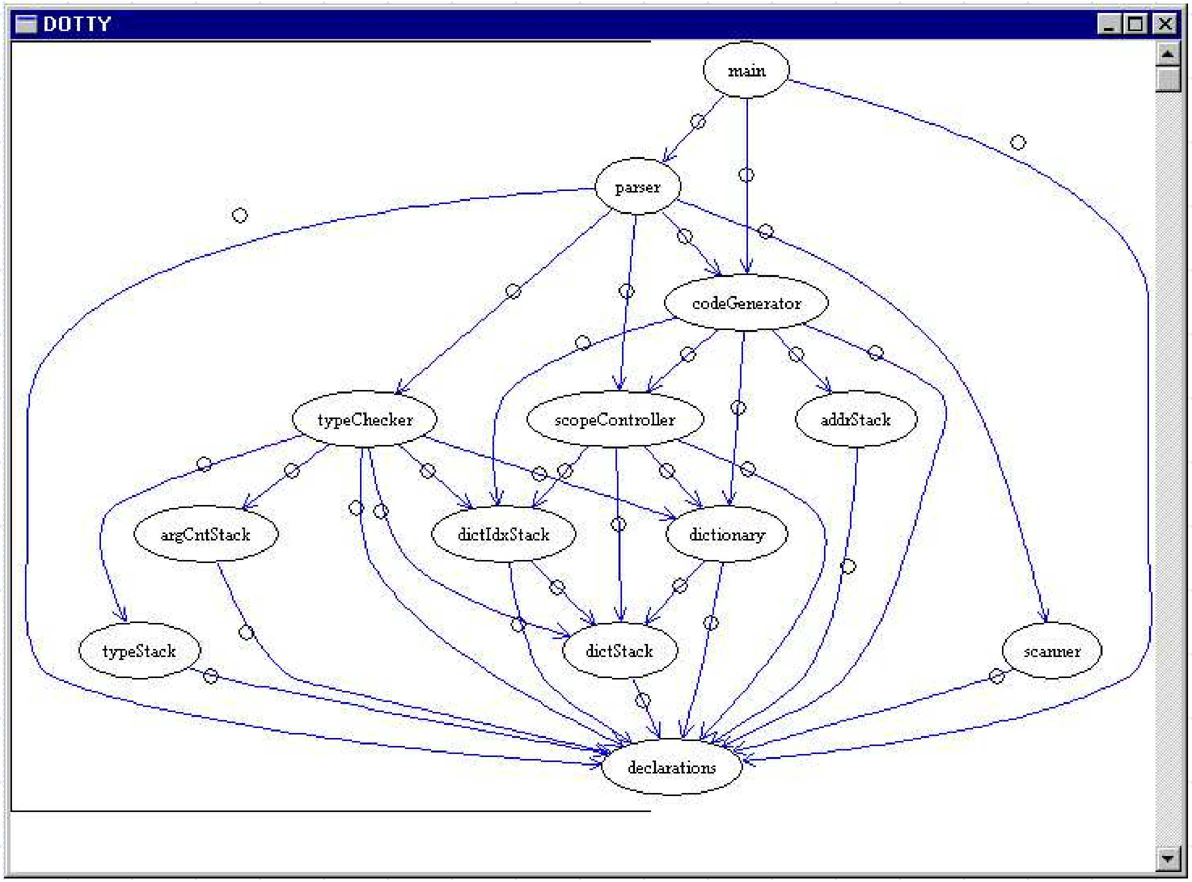
Bunch Clustering Service (BCS)

Neighboring Servers (NS)

Evaluation

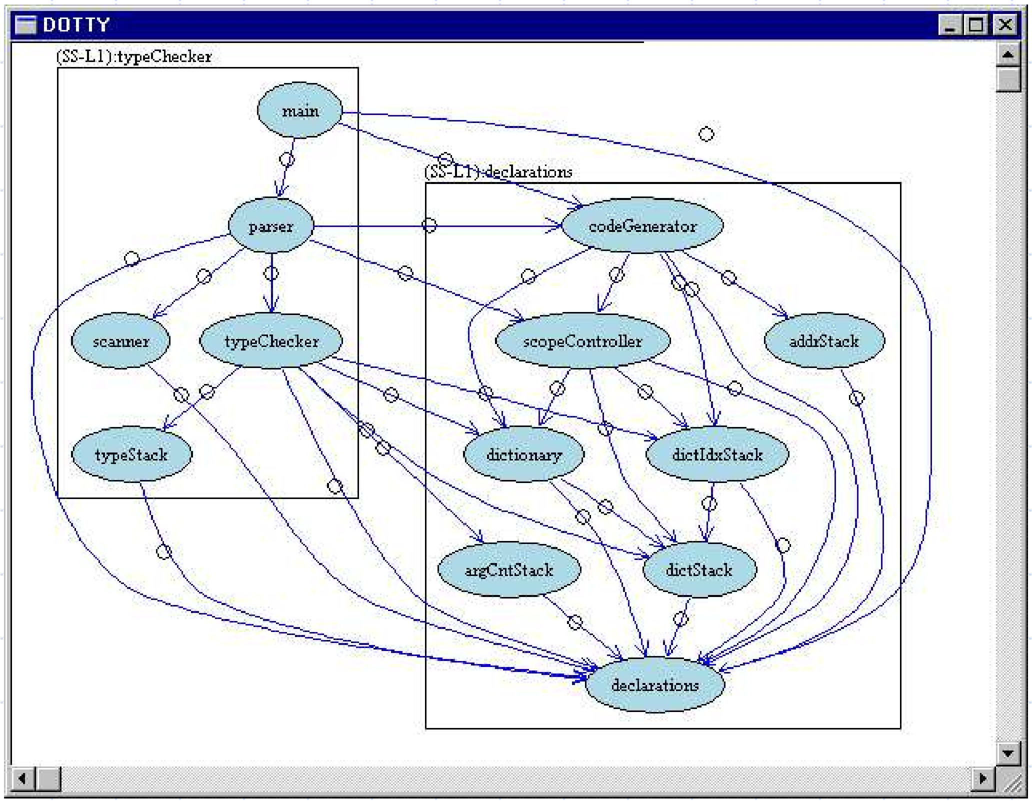
Clustering the Structure of a System (1)

Given the structure of a system…



Clustering the Structure of a System (2)

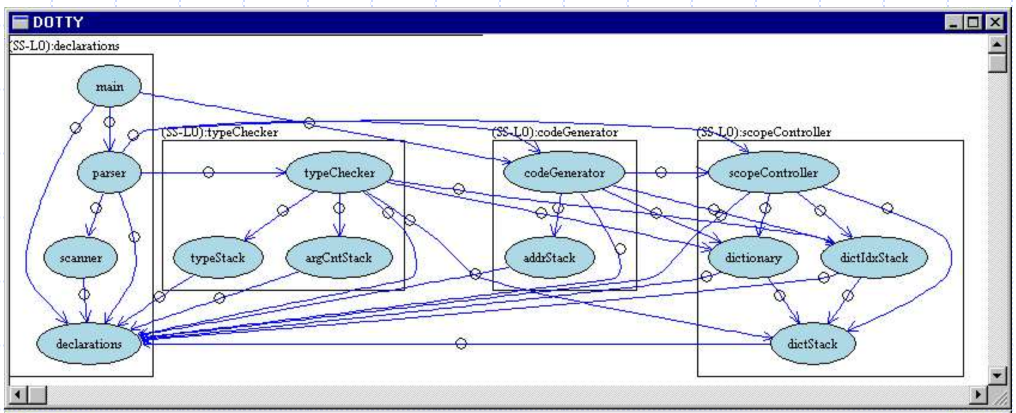
The goal is to partition the system structure graph into clusters…

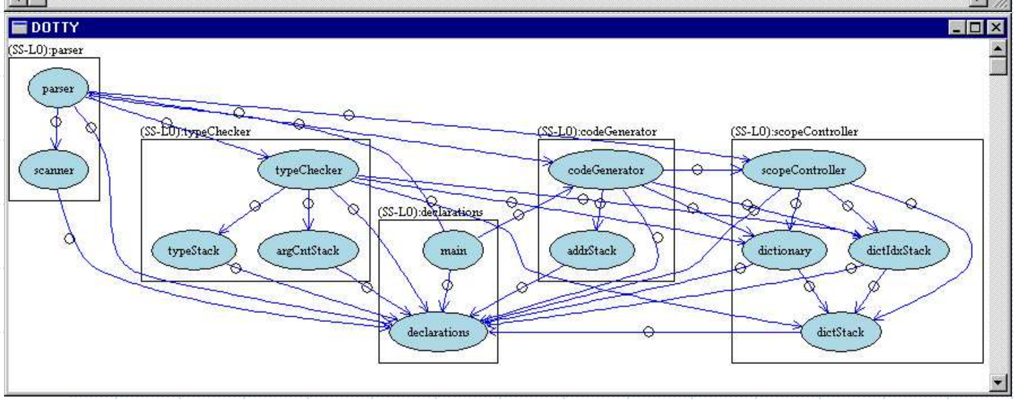


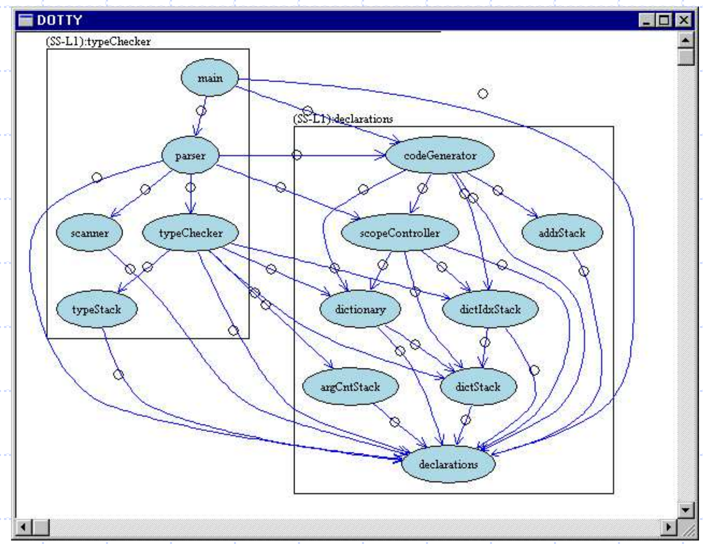
The clusters should represent the subsystems

Clustering the Structure of a System (3)

But how do we know that the clustering result is good?







Ways to Evaluate Software Clustering Results…

Given a software clustering result, we can:

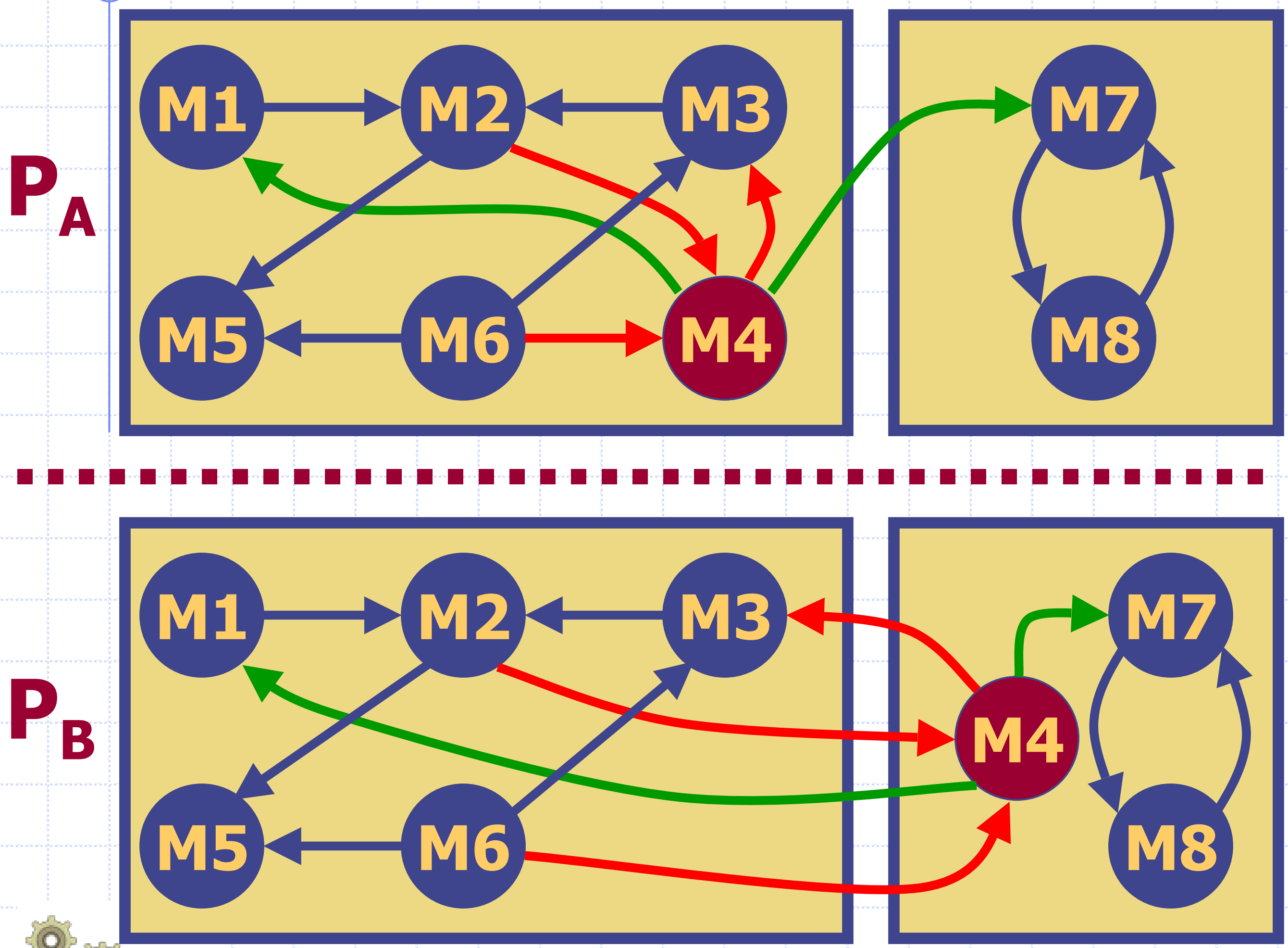
- Assess it against a mental model
- Assess it against a benchmark standard
  - Created Manually
  - Automatically Generated (CRAFT Tool)
- Techniques:
  - Subjective Opinions
  - Similarity Measurements (MeCl & EdgeSim)

Observation: Similarity Measurements

An important aspect of evaluation is being able to compare clustering results objectively…

- Edges are important for determining the similarity between decompositions
- Existing measurements don't consider edges:
  - Precision / Recall *(similarity)*
  - MoJo *(distance)*
- **Our idea:** Use the edges to determine similarity (MeCl & EdgeSim) [ICSM'01]
  - Our similarity measurements are integrated into the Bunch tool

Example: How "Similar" are these Decompositions?



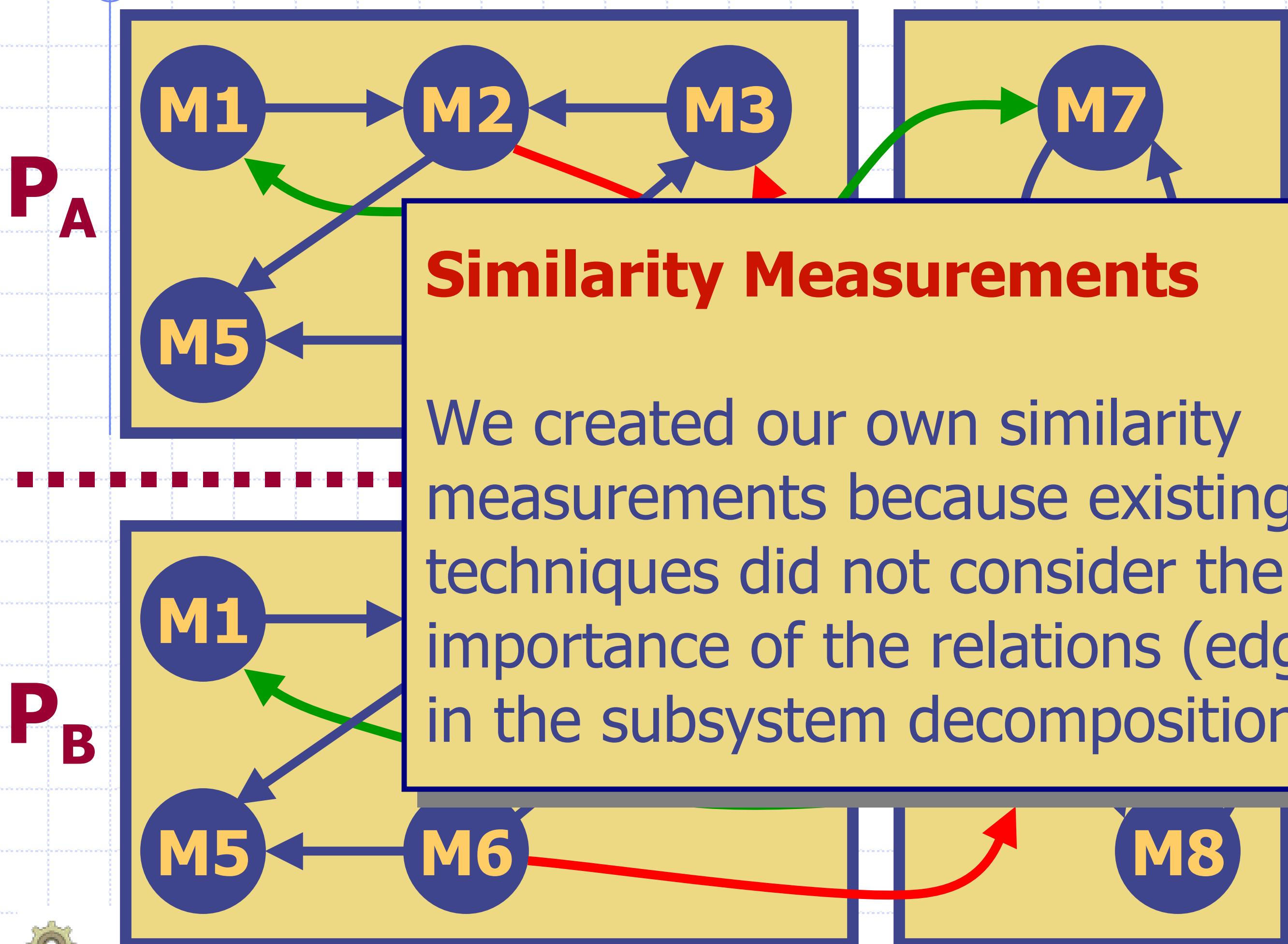
Blue Edges: Similarity still the same…

Green Edges: Similarity still the same…

Red Edges: Not as similar…

Conclusions: Once we add the **red** edges the similarity between $\mathbf{P_A}$ and $\mathbf{P_B}$ decreases

Example: How “Similar” are these Decompositions?



Similarity Measurements

We created our own similarity measurements because existing techniques did not consider the importance of the relations (edges) in the subsystem decomposition…

Blue Edges:
Similarity still the same…

Green Edges:
…arity still …ame…

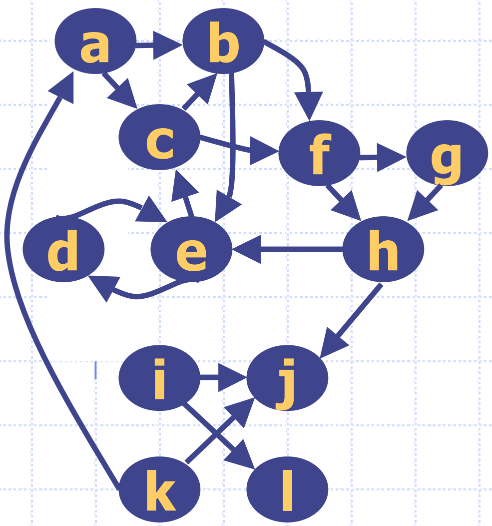
…Edges:
…s similar…

…clusions:
… we add the …dges the …ilarity between **P_A** and **P_B** decreases

EdgeSim Example

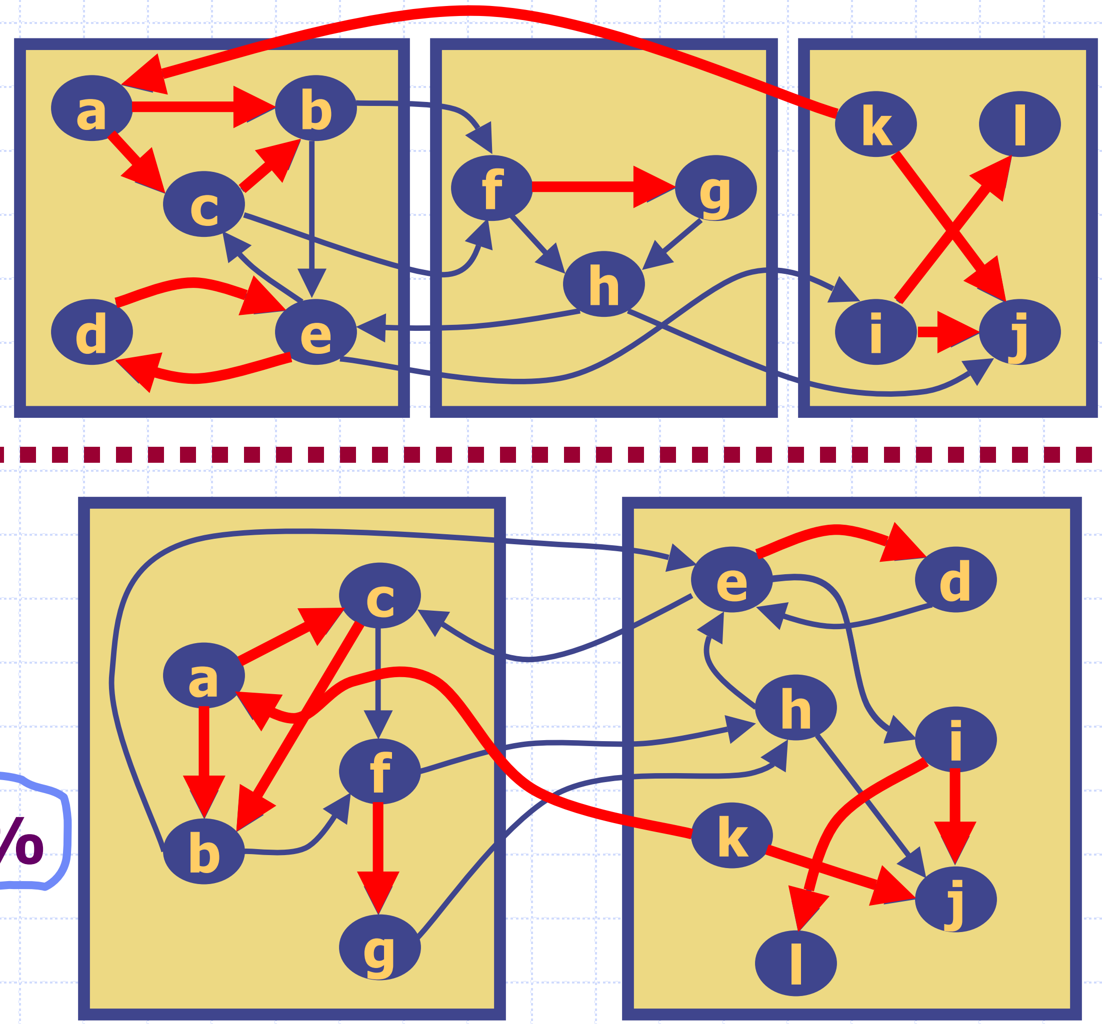
Common Edges

MDG



P_A

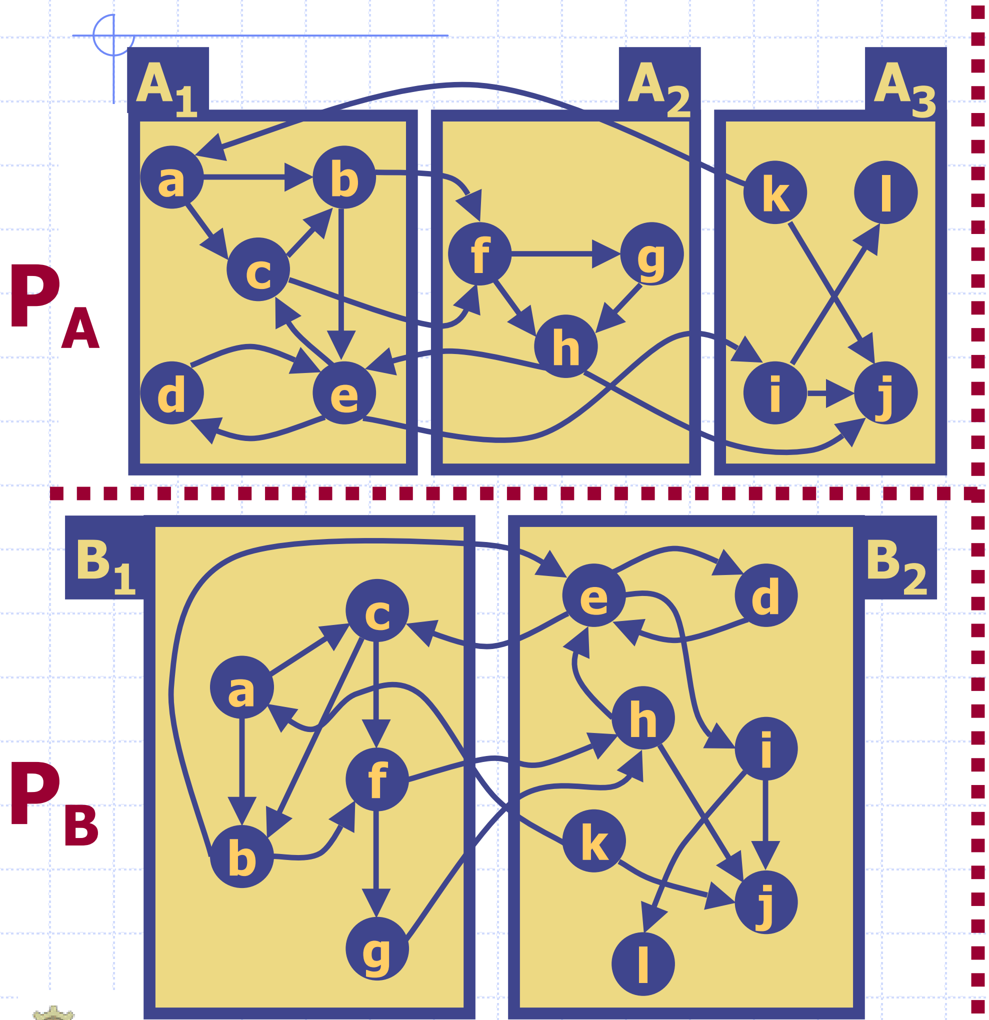
P_B

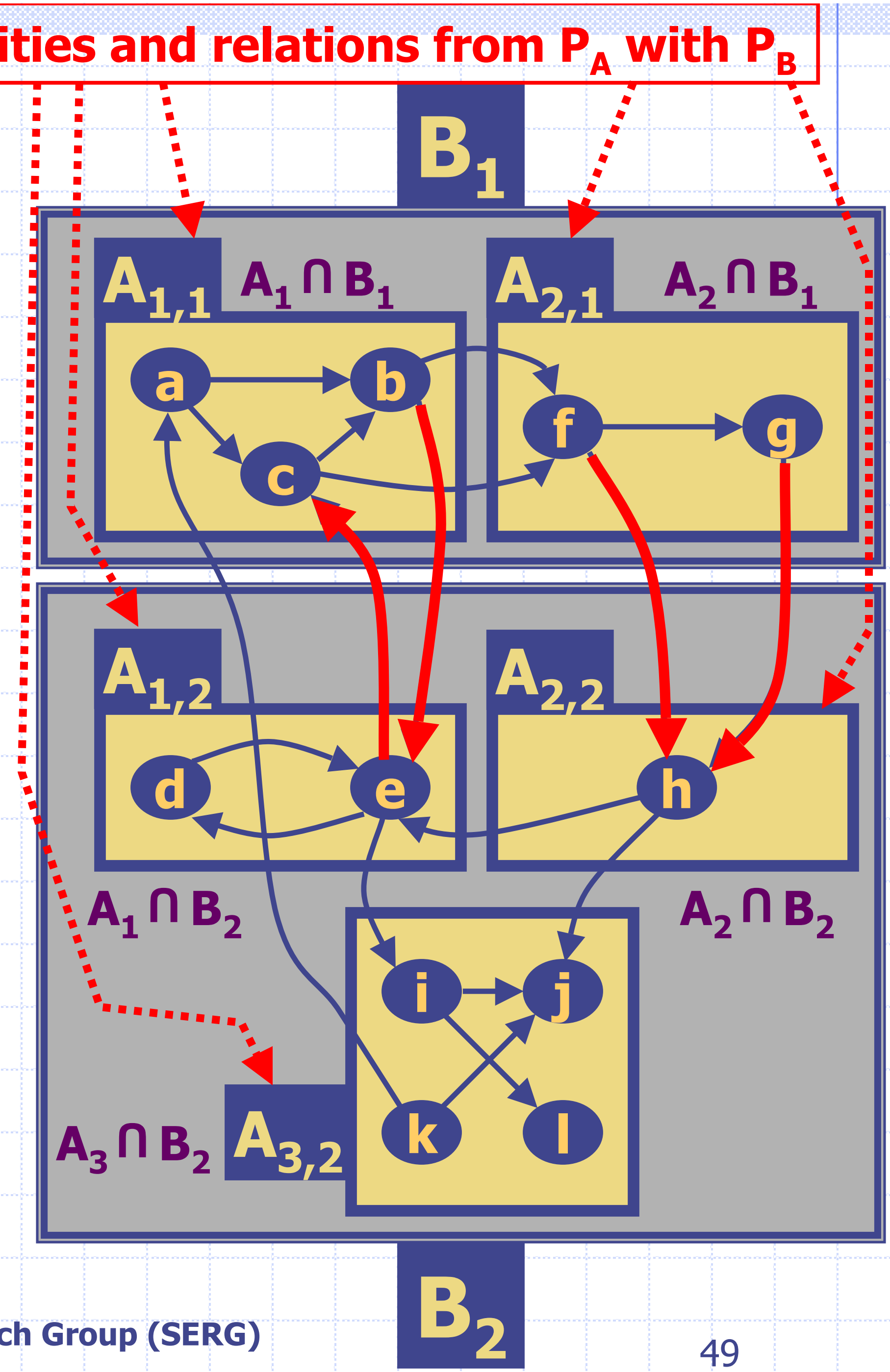


$$\frac{\text{Common Edge Weight}}{\text{Total Edge Weight}} = \frac{10}{19} = \mathbf{53\%}$$

MeCl Example (A→B)

Newly Introduced Inter-Edges

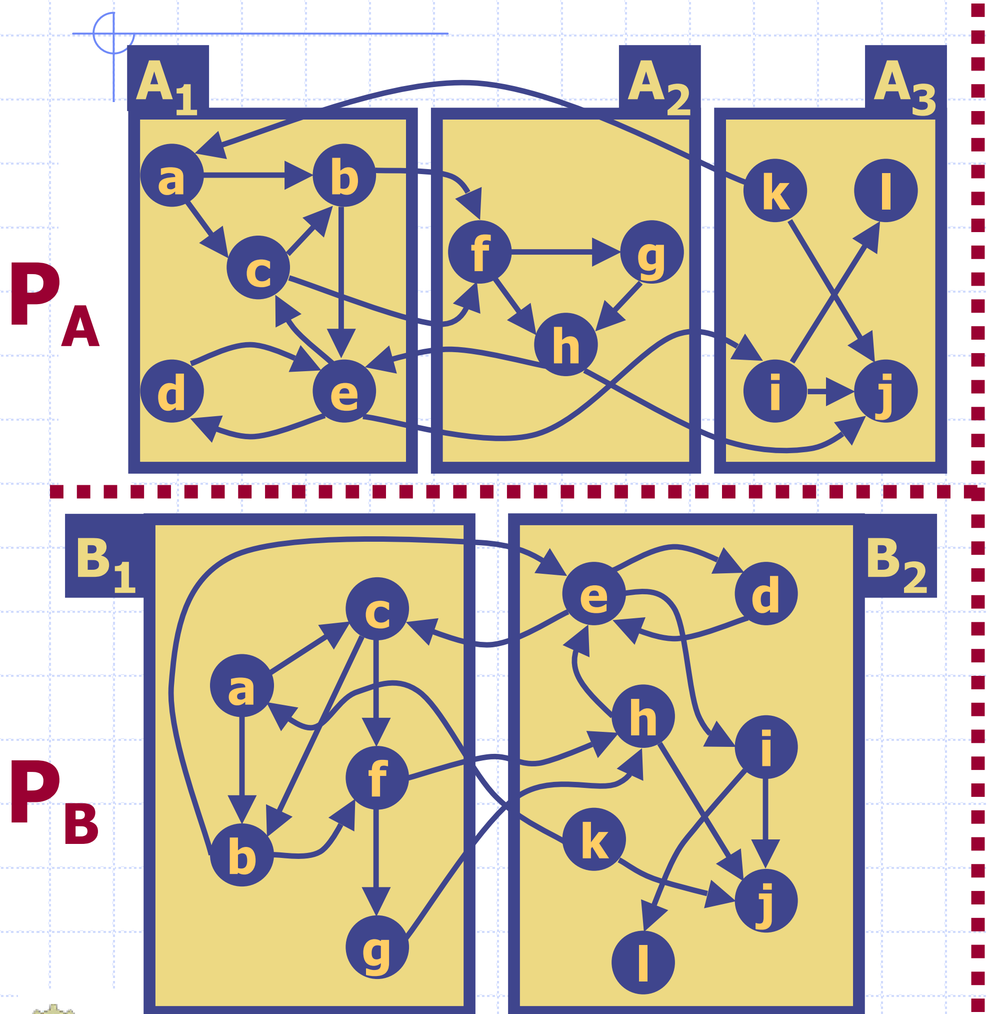


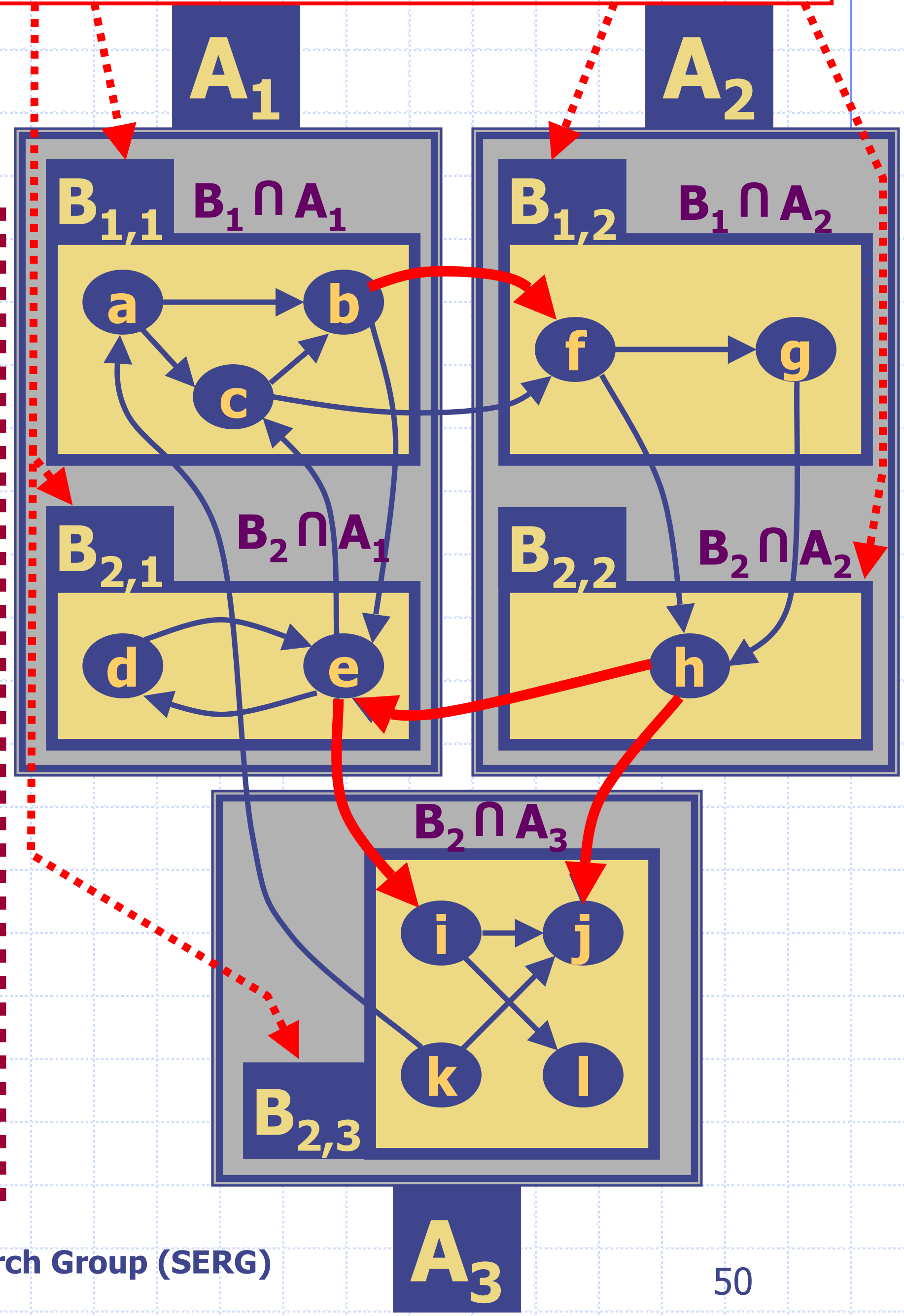


Intersection of common entities and relations from P_B with P_A

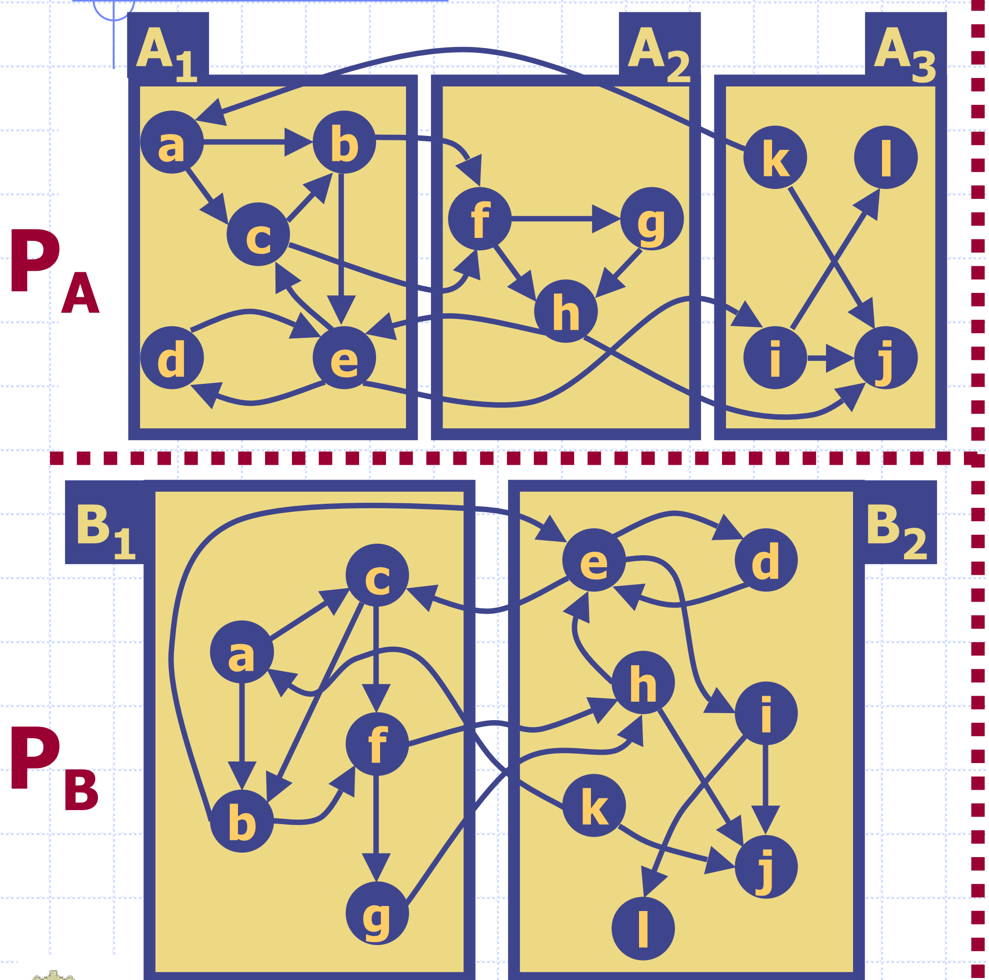
MeCl Example (B→A)

Newly Introduced Inter-Edges





MeCl Calculation



Inter-Edges Introduced

MeCl(A→B):
({b,e},{e,c},{g,h},{f,h})

MeCl(B→A):
({e,i},{h,j},{b,f},{c,f},{h,e})

$$\textbf{MeCl} = \left[1 - \frac{\max_W(M_{A \to B}, M_{B \to A})}{\text{Total Edge Weight}} \right]$$

$$\textbf{MeCl} = \left[1 - \frac{5}{19} \right] = \textbf{73.7\%}$$

Summary: Similarity Measurement

- Evaluation requires the ability to compare clustering results
- We created MeCl and EdgeSim to overcome some limitations that we found with other measurements
  - The edges are important!
- **MeCl:** Similarity improves as clustering algorithms create larger subsets of overlapping clusters
- **EdgeSim:** Similarity improves as clustering algorithms find common edge-types

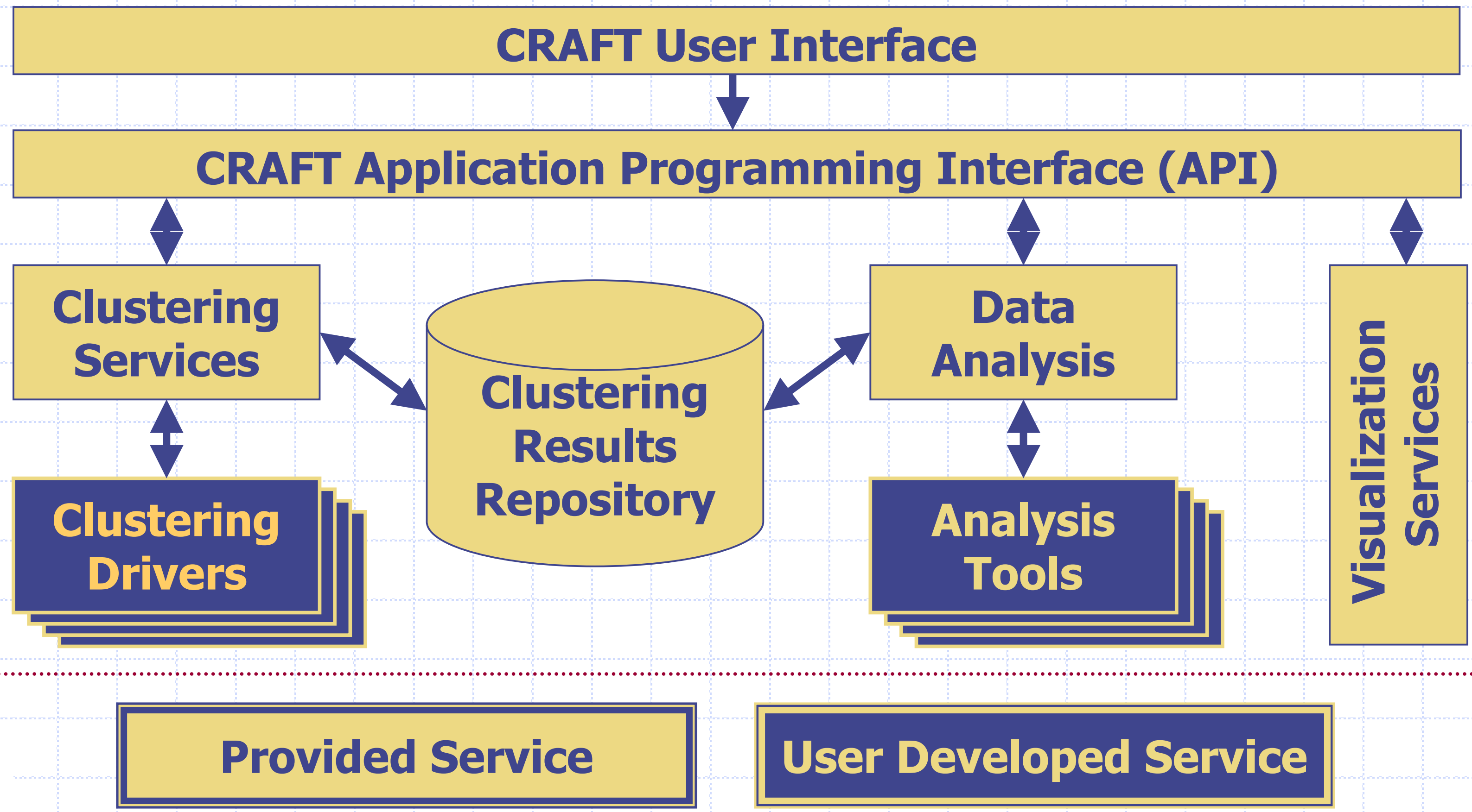
Reference Decompositions

- A ***Reference Decomposition***, (a.k.a *Gold Standard*, or *Benchmark)* is a "good" clustering result
- Difficult to create a *de facto* reference decomposition
- Our approach: If no benchmark is available, we create one based on commonalities in various clustering results

If a reference decomposition does not exist, the goal is to generate one that is "good enough"

The CRAFT Tool: A Reference Decomposition Generator

CRAFT = Clustering Results Analysis Framework and Tools



The CRAFT Tool: A Reference Decomposition Generator

CRAFT = Clustering Results Analysis Framework and Tools

CRAFT User Interface

Cl

Clusterin
Service

Clusterin
Drivers

Visualization Services

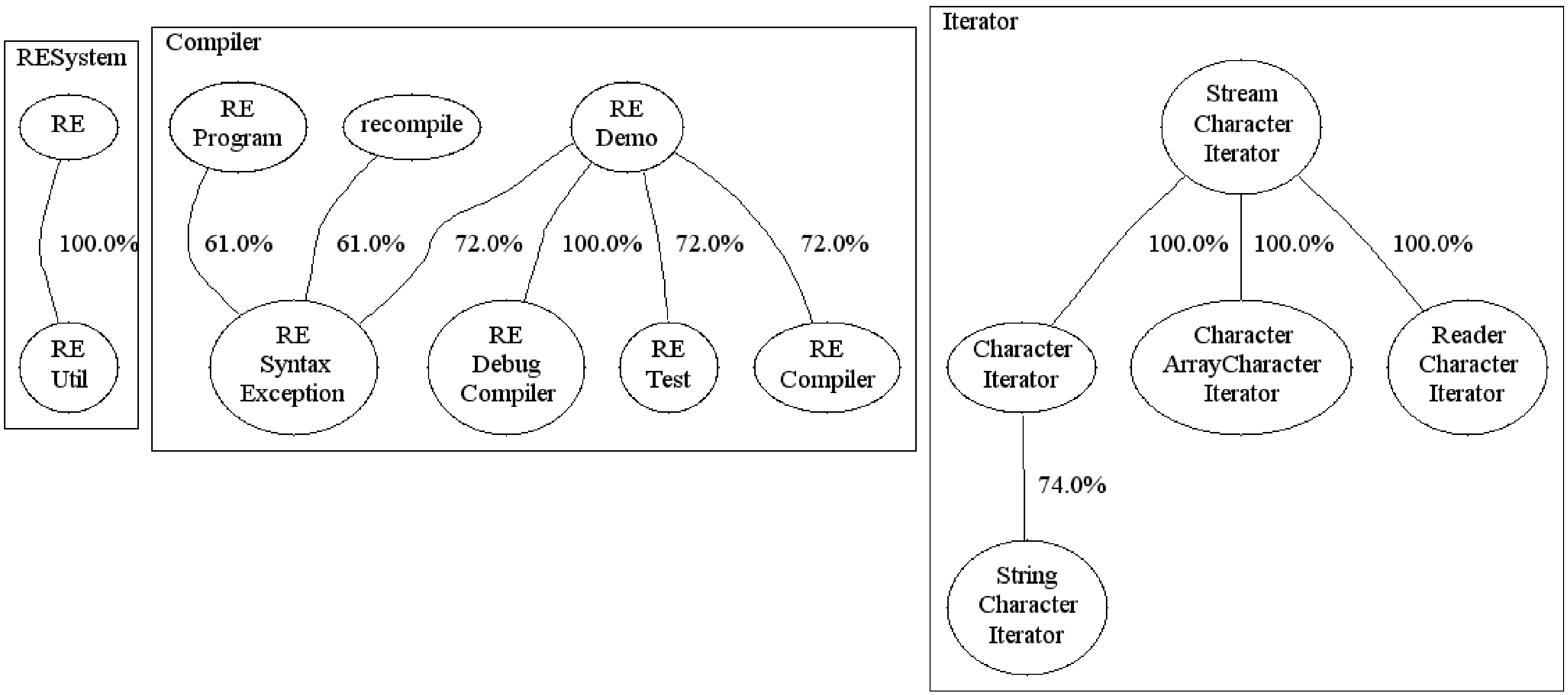
Generating a Reference Decomposition

CRAFT highlights areas of strong, weak and marginal agreement between different clustering algorithms

Provided Service

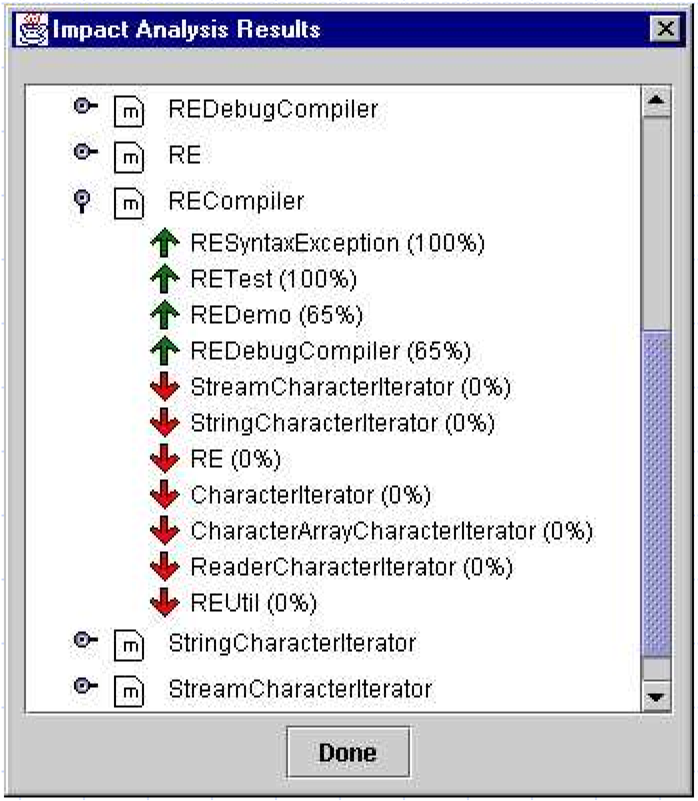
User Developed Service

Example: Recovered Reference Decomposition for Apache's RegExp



CRAFT's Confidence Analysis Reference Decomposition Generator

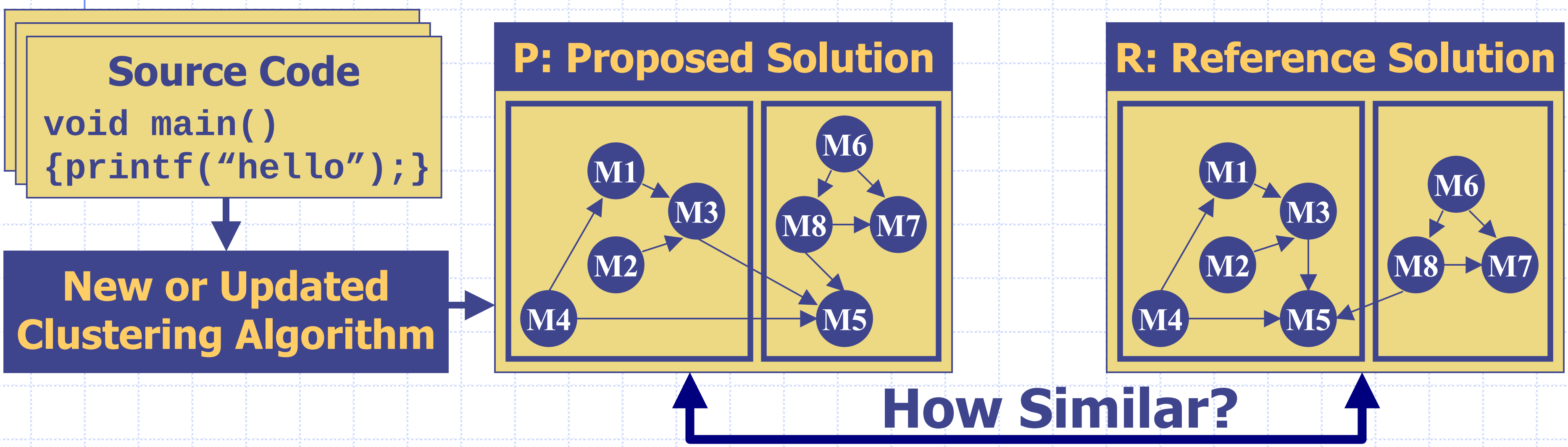
Example: Impact Analysis for for Apache's RegExp



Impact Analysis is good for helping users to understand the impact associated with making a local change…

Summary: Evaluation

Reference decompositions and similarity measurements are needed to evaluate software clustering results…



Compare[Sim(P),Sim(R)]

- **Good: High Similarity**
- **Bad: Low Similarity**
- **Can't Tell: Moderate Similarity**

Conclusions & Future Work

Contributions

- Search algorithms that partition the structure of a software system into subsystems
- Implementation of algorithms into the Bunch tool that supports clustering large systems (e.g., Linux)
- Evaluation techniques including similarity measurements and a reference decomposition generator

Contributions

- Sc ... ure of
- Ir ... ch to (e
- Ev m de

Impacts of our Research

- Our papers are widely referenced by researchers in the reverse engineering community.
- Our tools are used by software engineering students, researchers, and industry.
- Our tools are commercial quality, and are designed to be both used and extended by others.
- We are using Bunch services to create other tools in the SERG lab, including an online reverse engineering portal.

Research Opportunities

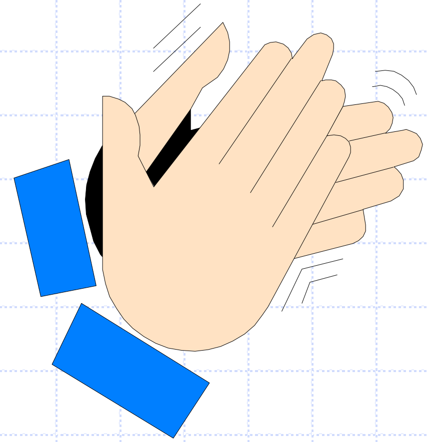
- Improved Visualization Services
  - Large systems are hard to visualize
- Investigate other types of MDG's
  - Example: Data flow, dynamic analysis, distributed systems
- Revisit Bunch's GA design and implementation
  - Improve encoding technique
- Addressing theoretical questions
  - How close is the solution to the "ideal" solution

Publications

1. “Search Based Reverse Engineering”. (with S. Mancoridis, M. Traverso). Submitted for Publication.
2. “Using Heuristic Search Techniques to Extract Design Abstractions from Source Code”. (with S. Mancoridis). Proc. of the Genetic and Evolutionary Computation Conference. (GECCO’02), New York, NY, July, 2002.
3. “Comparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements”. (with S. Mancoridis). Proc. of the IEEE International Conference on Software Maintenance (ICSM'01), Florence, Italy, November, 2001.
4. “CRAFT: A Framework for Evaluating Software Clustering Results in the Absence of Benchmark Decompositions”. (with S. Mancoridis). Proc. of the IEEE Working Conference in Reverse Engineering (WCRE'01), Stuttgart, Germany, October, 2001. **(Best Paper Award)**
5. “An Architecture for Distributing the Computation of Software Clustering Algorithms". (with S. Mancoridis, M.Traverso). Proc. of the IEEE/IFIP Working Conference on Software Architecture (WICSA'01), Amsterdam, Netherlands, August, 2001.
6. “Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures”. (with S. Mancoridis, Y.Chen, E.R.Gansner). Proc. of the IEEE International Conference on Software Maintenance (ICSM'99), Oxford, UK, August, 1999.
7. “Automatic Clustering of Software Systems using a Genetic Algorithm”. (with D. Doval, S. Mancoridis). Proc. of the IEEE International Conference on Software Tools and Engineering Practice (STEP'99), Pittsburgh, PA, August, 1999.
8. “Using Automatic Clustering to Produce High-Level System Organizations of Source Code”. (with S. Mancoridis, C.Rorres, Y.Chen, E.R.Gansner). Proc. of the IEEE International Workshop on Program Understanding (IWPC'98), Ischia, Italy, June, 1998.

Papers are available online

Recognition



Special Thanks To:

- **My Advisor:** Dr. Spiros Mancoridis
- **My Committee:** Dr. J. Johnson, Dr. C. Rorres, Dr. A. Shokoufandeh, Dr. R. Chen, and Dr. L. Perkovic (former member)
- **My Sponsors:** AT&T Research, Sun Microsystems, DARPA, NSF, US Army
- **Bunch Project Contributors:** D. Doval, M. Traverso, S. Mancoridis
- Dr. E. Gansner & Dr. R. Chen (AT&T Labs - Research) for test data and validation of Bunch's clustering results.
- The gang at the SERG lab…

DREXEL UNIVERSITY

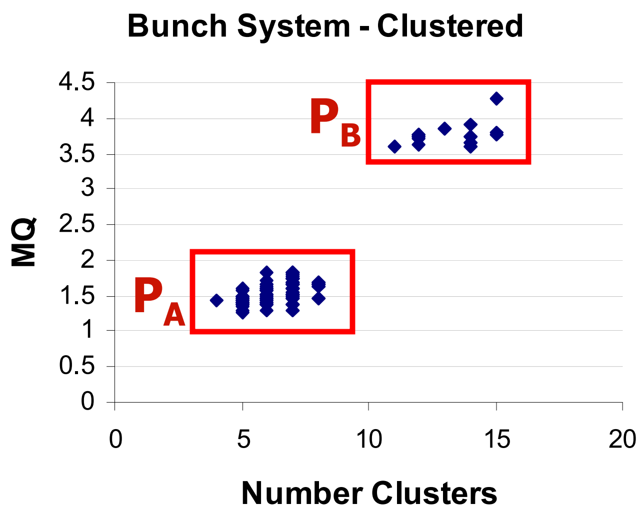
A Heuristic Search Approach to Solving the Software Clustering Problem

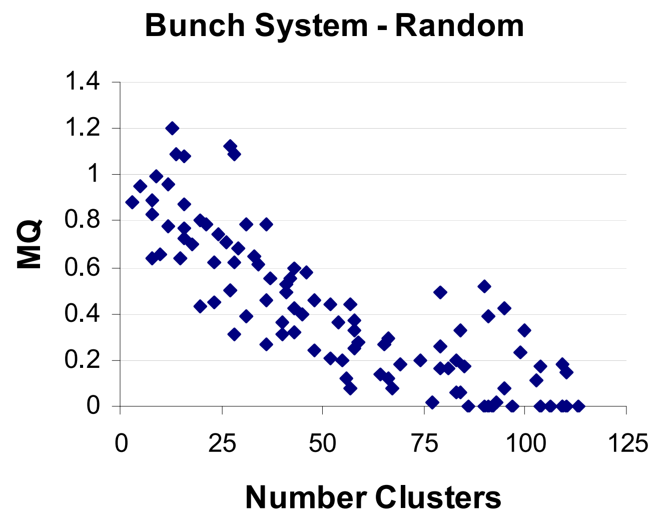
Questions

http://www.mcs.drexel.edu/~bmitchel/research

Extra Data

Results: Bunch System





P_A

	Min	Max	Avg
ES	59%	100%	80%
MC	80%	100%	91%

P_B

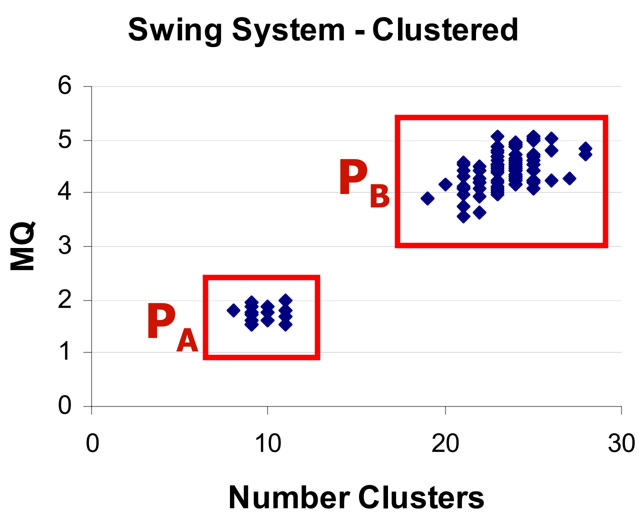
	Min	Max	Avg
ES	73%	100%	88%
MC	89%	100%	95%

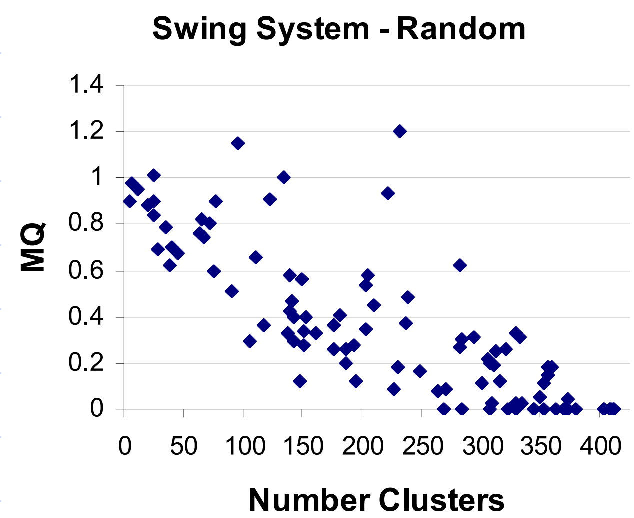
P_A versus P_B

	Min	Max	Avg
ES	63%	100%	71%
MC	69%	100%	84%

Sample Size = 100

Results: Swing System





P_A

	Min	Max	Avg
ES	64%	100%	78%
MC	86%	100%	92%

P_B

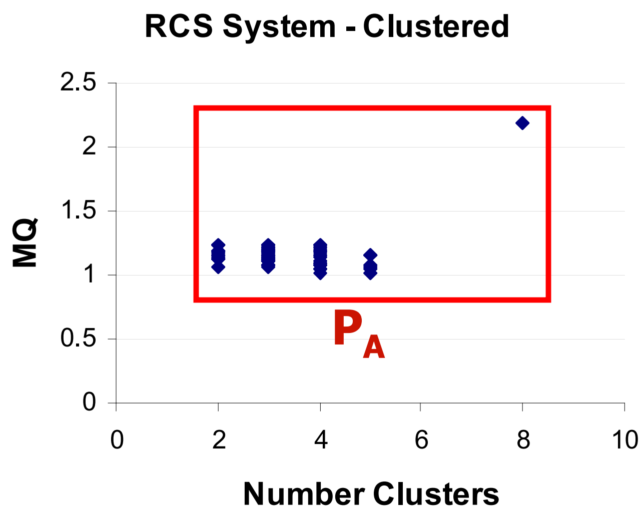
	Min	Max	Avg
ES	74%	100%	83%
MC	88%	100%	93%

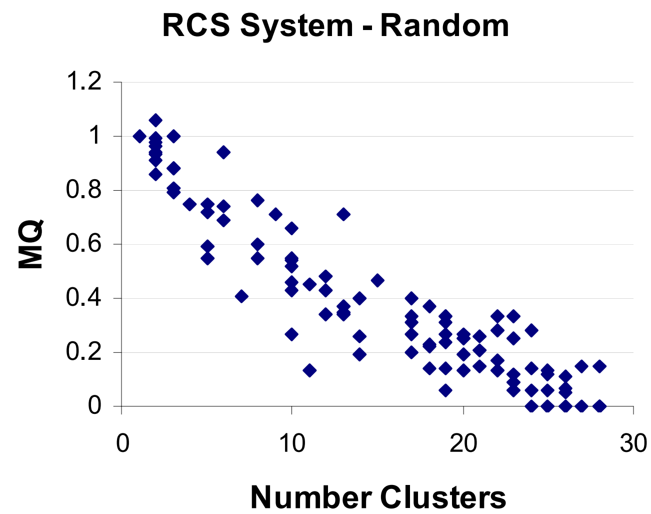
P_A versus P_B

	Min	Max	Avg
ES	63%	86%	74%
MC	70%	91%	79%

Sample Size = 100

Results: RCS System





P_A

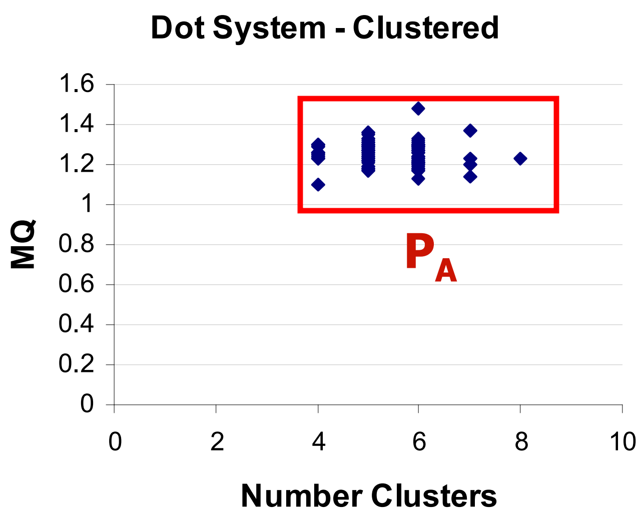
	Min	Max	Avg
ES	43%	100%	81%
MC	45%	100%	88%

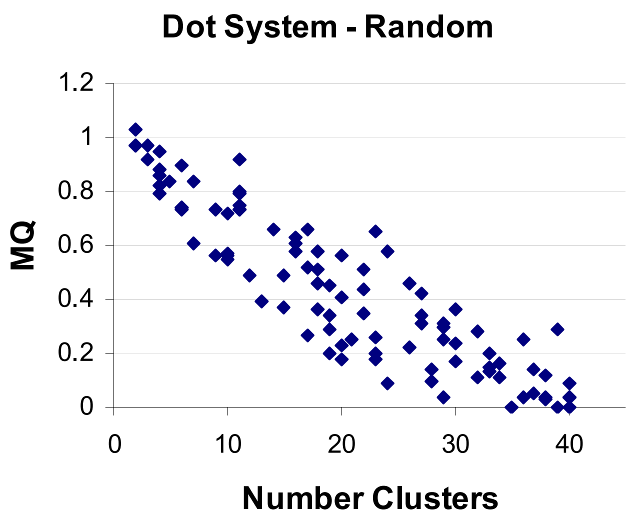
Sim_{MIN} versus Sim_{MAX}

	Sim Value	Max MQ	Min MQ
ES	43%	2.19	1.15
MC	45%	2.19	1.07

Sample Size = 100

Results: Dot System





P_A

	Min	Max	Avg
ES	63%	100%	81%
MC	79%	100%	88%

Sample Size = 100