ÉCOLE
POLYTECHNIQUE
MONTRÉAL

Ptidej
Pattern Trace Identification, Detection, and Enhancement in Java

Soccerlab
SOftware Cost-effective Change and Evolution Research Lab

OdMoMS: Multi-Objective Miniaturization of Software

Giuliano (Giulio) Antoniol

With the collaboration of Nasir Ali, Wei Wu, Yann-Gaël Guéhéneuc, Jane Huffman Hayes, Max Di Penta

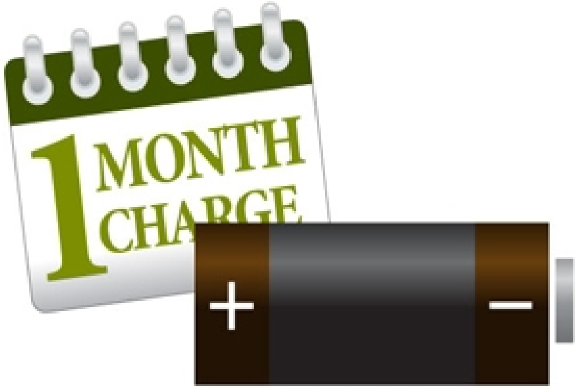
Motivation

PORT
HAND HELD DEVICES
How?
SMS
NEWS

Resources vs. Feature vs. Customers

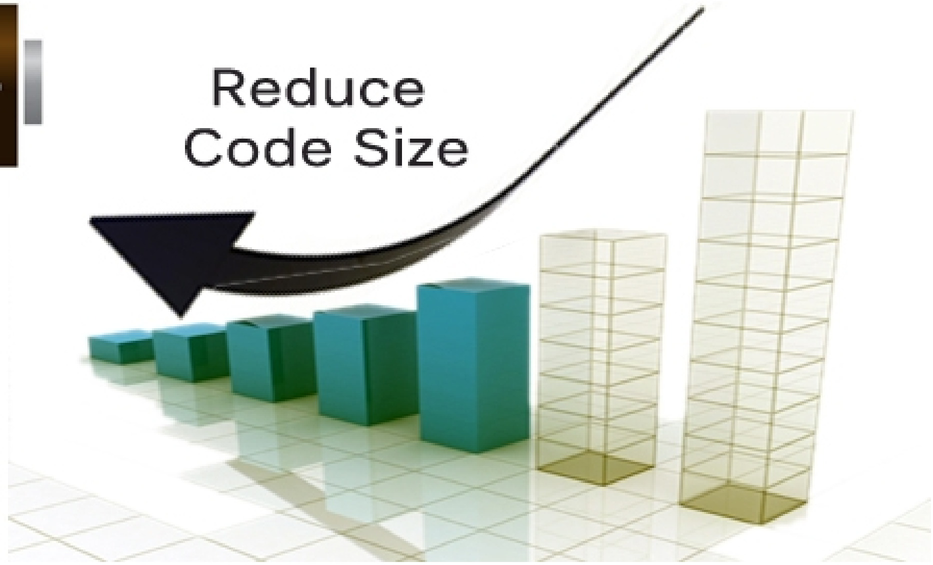
Our Goal

Better Performance

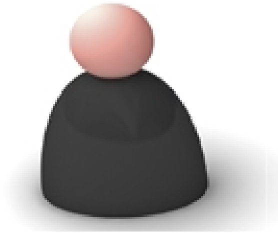


Maximize Customers' Satisfaction

Reduce Code Size



Different Customers – Different Features



Customer Relative Weight

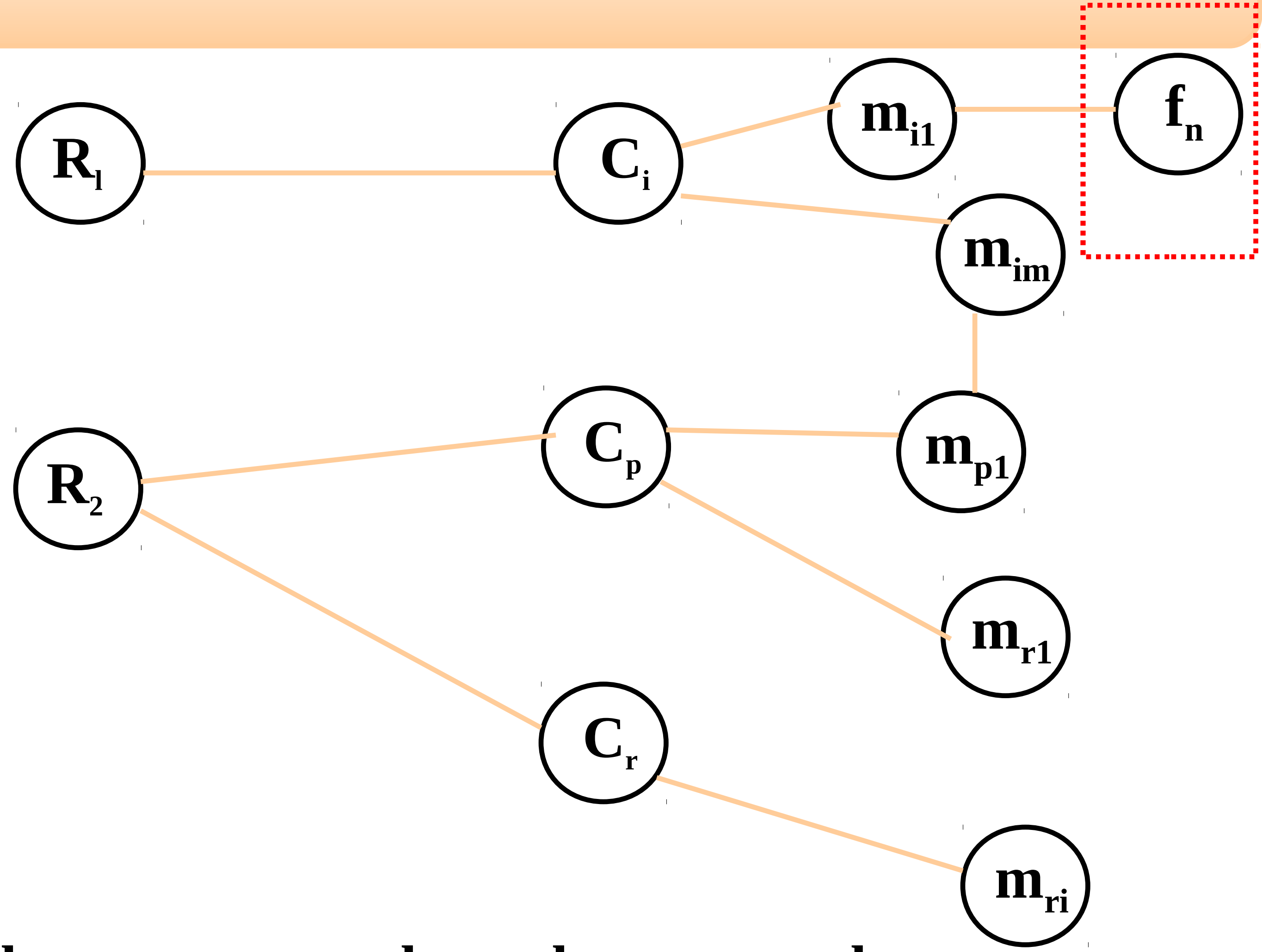
Overall

- What do customer want?
  - What do we already have ?
  - PREREQUIR + ReORe.
- How can we make customers happy?
  - Static vs. dynamic information
  - Size vs. features vs. happy customers vs. CPU consumption
- Miniaturization problem.
- Case Study.
- Conclusion.

PREREQIR in a Nutshell

- We need pre-requirement documents:
  - What the competitors' systems do?
  - What our customers want?
- We obtain and vet a list of requirements from diverse stakeholders.
- We structure requirements by mapping them into a representation suitable for grouping via pattern-recognition and similarity-based clustering.

Static Traceability Map



The system may depend on external components e.g., an LDAP server

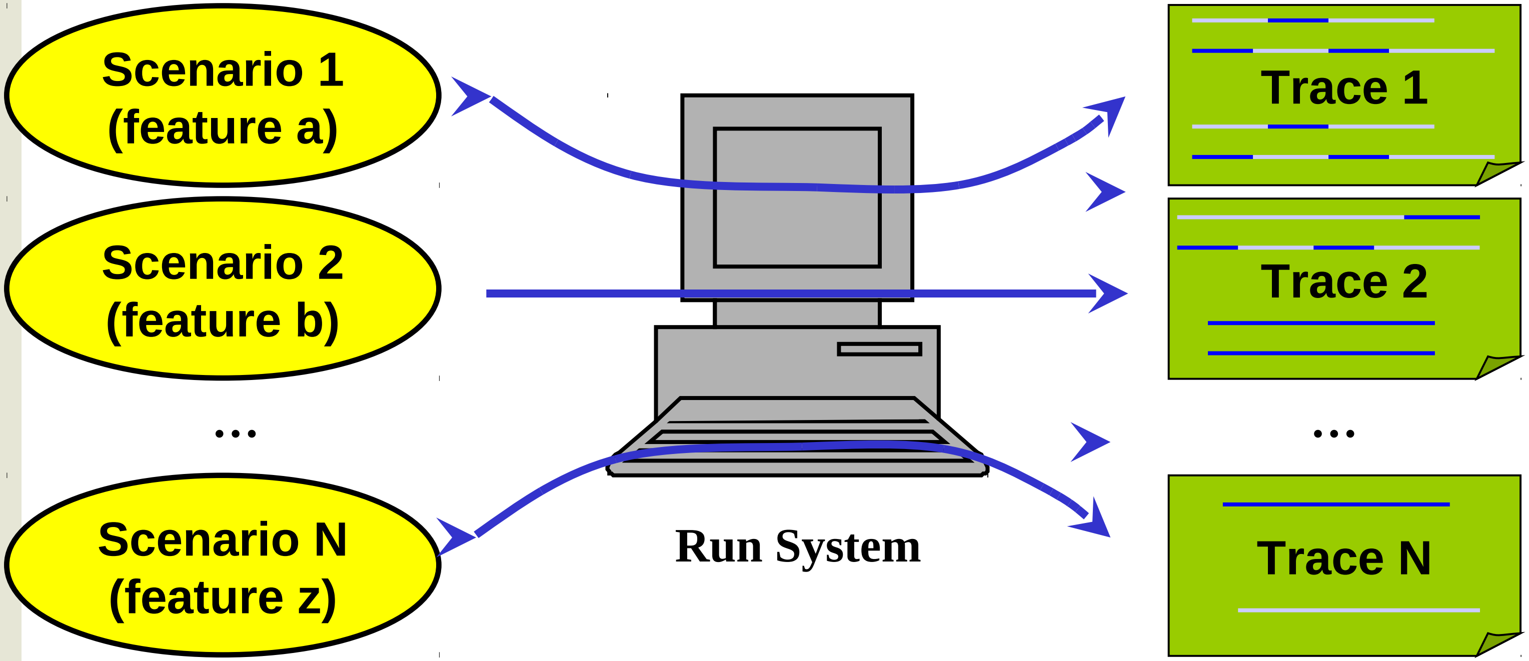
Features to Size

- Traceability relations are tagged with:
  - Size information.
  - IDs of customers requiring the given feature.
- Features are divided into:
  - Compulsory.
  - Cherry on the pie.
- Selected features must lead to a compilable system:
  - Extra code may be needed just to make sure that the system compiles and runs.

Features to CPU Consumption

- Assumption: CPU cycles/consumption is related to energy consumption:
  - The higher the CPU consumption, the lower the battery life.
- Binder’s JP2 profiling tool: comprehensive calling-context profiles:
  - Exact number of executed bytecodes for each calling context.
- Caveat: modern hardware architecture prevent exact estimation based on bytecode counting
- Bytecode counting is a good approximation of run time algorithmic complexity.
  - The lower the number of executed bytecodes, the lower the CPU time, the lower the battery consumption.

Requirements to Features



Dynamic Information

- Call tree:
  - Integrate call tree information for each executed feature with static traceability relations to count executed bytecodes.
- Evaluate CPU consumption at method level: accumulate into call tree top nodes the counts of lower nodes
  - Top nodes thus stores sub-tree bytecode counts.
  - Top nodes account for all executed bytecodes, including JARs and utility methods.
- Caveat:
  - Some feature may not be completely implemented.
  - Some feature may not be executed due to missing components.

Miniaturization Problem

- We would like to:
  - Minimize size and CPU consumption.
  - Maximize customer satisfaction.
- Constraints may be imposed on the search space
  - Max available memory, max CPU power, customers that must be satisfied.
- Generate a Pareto surface:
  - Project Pareto surface onto a Pareto front.
- Final decision to the manager.

Miniaturization Problem (cont’d)

$C = \{c_1 \ldots c_L\}$ L customers;

Each customer has "value" $0 \le \mathrm{v} \le \mathrm{V}_{\max}$ assigned;

A set of ComF compulsory features;

OF $= \{F_1 \ldots F_L\}$ customer desired optional features;

Each $\mathrm{F_i} = \left\{ f_{i1} \ldots f_{iN_i} \right\}$ list customer i desired features;

A miniaturized program implements $\mathrm{F}' \subseteq$ OF features;

We have a set of implementation units IU $= \{iu_1 \ldots iu_M\}$;

There are properties $\mathrm{P} \subset \mathrm{R}^{\mathrm{K}}$ that must meet a set of constraints HC $= \{hc_1 \ldots hc_k\}$ where hc_j is an interval

Miniaturization Problem (cont'd)

- Traceability creates a function Impl that given a feature assigns implementation units.
- Each implementation unit has assigned properties values, e.g., each method has assigned a size and a CPU consumption.
- The Customer Satisfaction Ratio (CSR) is defined as:

$$CSR(F') = \frac{\sum_{i=1}^{L} \frac{|F_i \cap F'|}{|F_i|} \times \frac{v_i}{V_{\max}}}{L}$$

Miniaturization Problem (cont’d)

- Maximize CSR(F’) means minimize –CSR(F’)

- For a given set of features F’, the implementation units and the overall properties are:

$$IU^{'} = \mathrm{Impl}\left(\mathrm{F}' \cup \mathrm{ComF}\right)$$

$$P' = \mathrm{Prop}\left(\mathrm{Impl}\left(\mathrm{F}' \cup \mathrm{ComF}\right)\right)$$

- We assume that properties are additive: size (CPU consumption) of two units is the sum of units sizes (CPU consumptions).

Miniaturization Problem (cont’d)

$$\min_{F' \in 2^{OF}} \left\{ -CSR(F'), \mathrm{Prop}\left[\mathrm{Impl}\left(\mathrm{F'} \cup \mathrm{ComF}\right)\right] \right\}$$

$$\text{such that :}$$

$$\forall\, \mathrm{p_i} \mid \left(p_1 \ldots p_i \ldots p_K \right) = \mathrm{Prop}\left[\mathrm{Impl}\left(\mathrm{F'} \cup \mathrm{ComF}\right)\right] : \mathrm{p_i} \in hc_i$$

Notice that $\mathrm{Prop}\left(\mathrm{Impl}\left(\mathrm{F}' \cup \mathrm{ComF}\right)\right)$

is actually an array of sizes and CPU consumptions. Thus, a solution is a surface:

CSR = FUNC(size, CPU consumption)

Case Studies

- 350 questionnaires, 73 completed surveys
- Pooka V2.0 e-mail client:
  - 208 classes.
  - 20,868 methods.
  - 245 KLOCs.
  - 599 pre-requirements.
  - 30 traced features.
  - Code size 5.39 MB.
- SIP V1.0 audio/video internet phone:
  - 1,771 classes.
  - 31,302 methods.
  - 486 KLOCs.
  - 639 pre-requirements.
  - 36 traced features.
  - Code size 27.3 MB.

NSGA-II Parameters

- We used JMETAL:
  - Mutation probability 4%.
  - Crossover 90%.
  - Evaluation number 25,000.

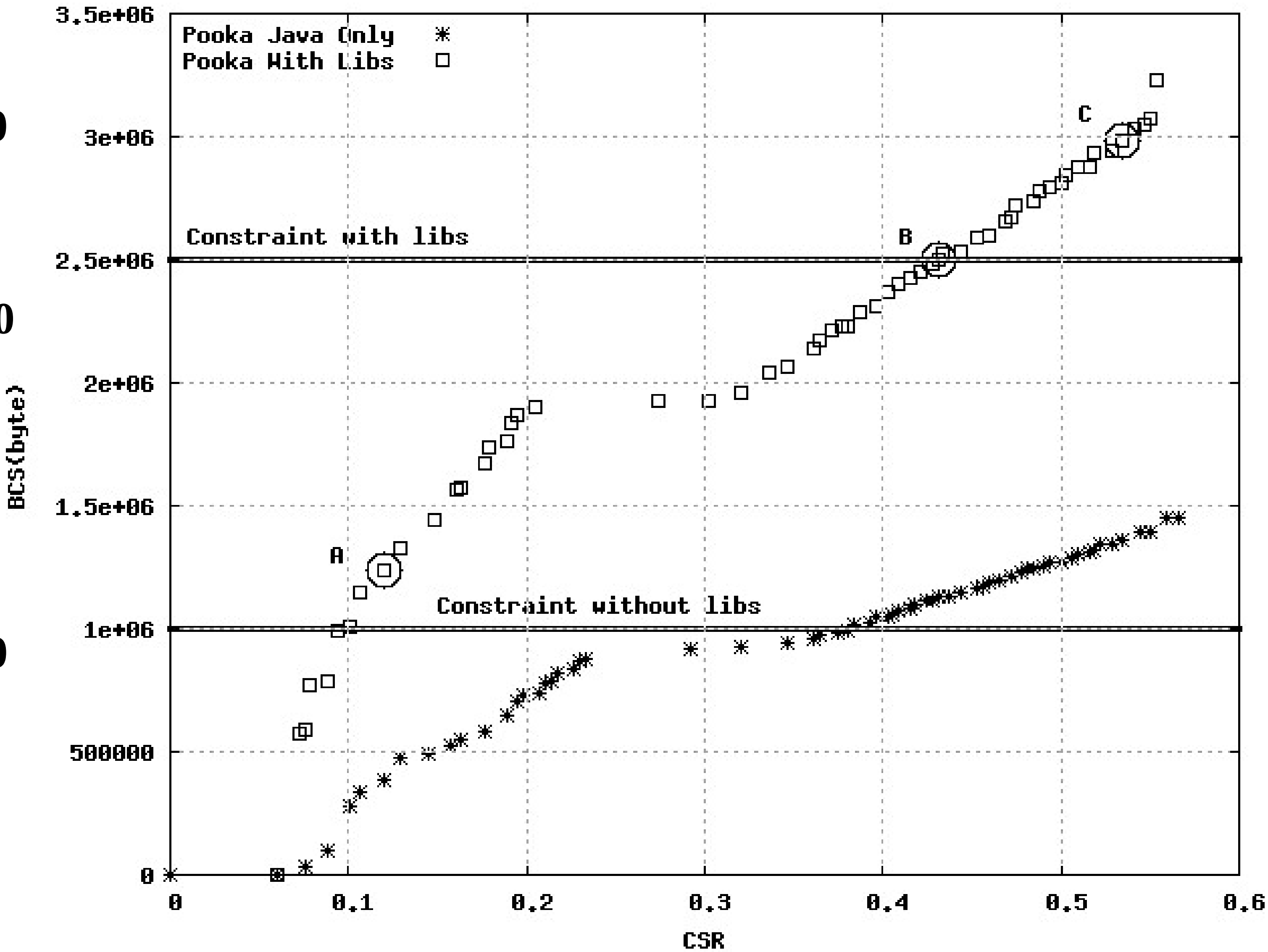
- High iteration number to ensure that we did not miss good solutions.

Pooka Projection CSR vs. Size

A: CSR = 0.21
Features: 15/30

B: CSR = 0.5
Features: 19/30

C: CSR = 0.56
Features: 23/30

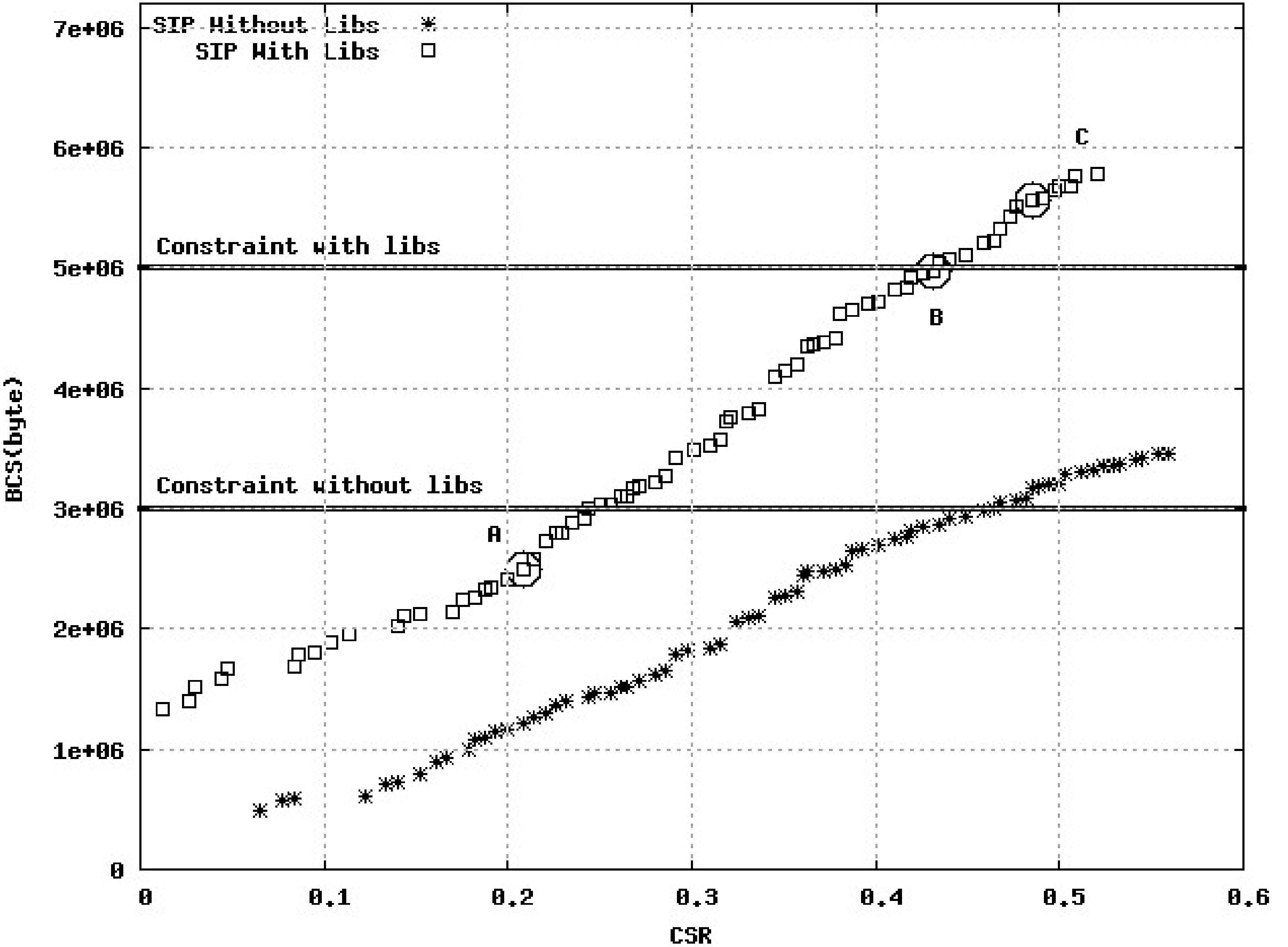


SIP Projection CSR vs. Size

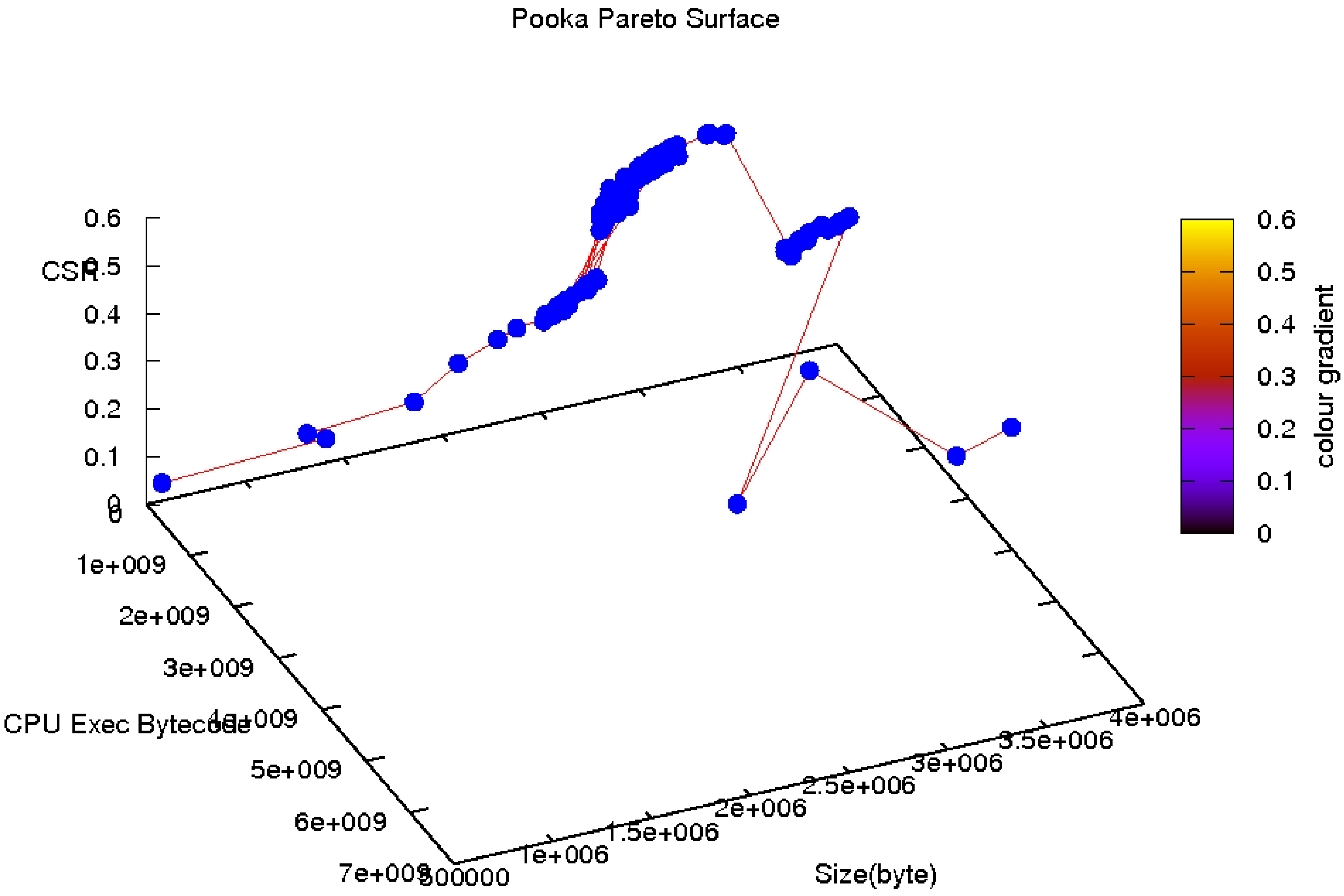
A: CSR = 0.20
Features: 10/36

B: CSR = 0.49
Features: 23/36

C: CSR = 0.56
Features: 31/36



Pooka Surface



Lessons Learned

- The miniaturization process is feasible but there are challenges:
  - Traceability recovery and accuracy of traced links.
  - Collecting dynamic information is difficult:
    - Missing or not 100% implemented features.
    - CPU consumption difficult to run:
      - We are still completing SIP.

- Some system (SIP) may exhibit tangled dependencies and there may be no sweet spot.

Conclusion

- The porting problem was modeled as a multi-objective minimization problem.

- Equations can accommodate a wide range of properties.

- The process can be automated thus saving considerable manual effort in selecting features to be ported:
  - Yet not in validating traceability links if links do not exist.

Questions