Synthesis of Search Algorithms from High-level CP Models

Samir A. Mohamed Elsayed & Laurent Michel

Computer Science & Engineering, University of Connecticut

Overview

UCONN

- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions & Future Work

Motivation

Motivation

CP = Model + Search

Motivation

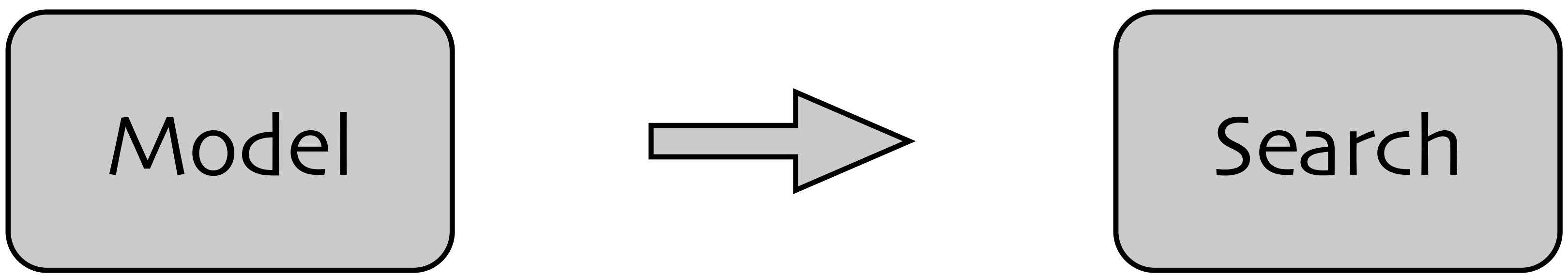
CP = Model + Search

VS.

X = Model

Motivation

- Automatic synthesis of search
- Retaining the ability to write custom search procedures
- Generate



Overview

UCONN

- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions

CP-AS

- Recognizes and classifies model structures
- Follows a rule-based approach
- Works on top of Comet

Example

```
Solver<CP> m();
minimize<m> s subject to {

  forall(i in V) m.post(c[i] <= s);
  forall(i in V, j in V : i < j)
    if (adj[i,j])
      m.post(c[i] != c[j]);

}
using {

  label(m);

}
```

Example

```
Solver<CP> m();
minimize<m> s subject to {

  forall(i in V) m.post(c[i] <= s);
  forall(i in V, j in V : i < j)
    if (adj[i,j])
      m.post(c[i] != c[j]);

}
using {

  label(m);

}
```

Example

```
Solver<CP> m();
minimize<m> s subject to {

  forall(i in V) m.post(c[i] <= s);
  forall(i in V, j in V : i < j)
    if (adj[i,j])
      m.post(c[i] != c[j]);

}
using {

  label(m);

}
```

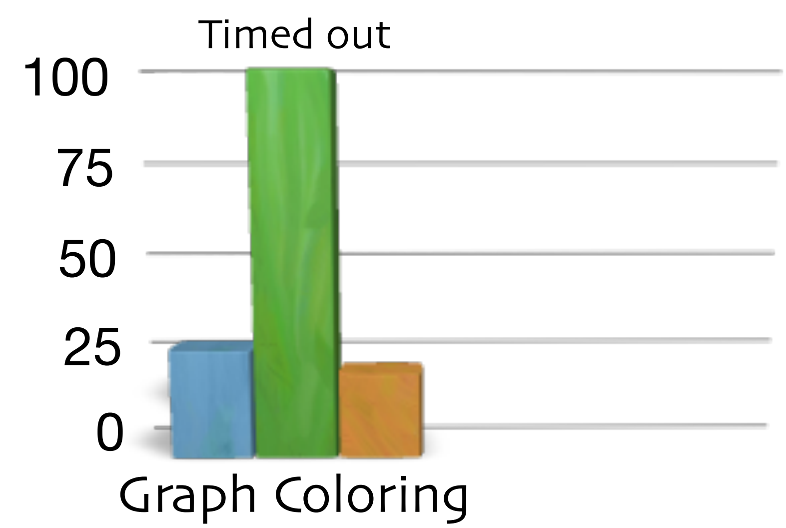
Example

```
Solver<CP> m();
minimize<m> s subject to {

  forall(i in V) m.post(c[i] <= s);
  forall(i in V, j in V : i < j)
    if (adj[i,j])
      m.post(c[i] != c[j]);

}
using {

   CPAS.generateSearch(m);

}
```

Example

```
Solver<CP> m();
minimize<m> s subject to {

  forall(i in V) m.post(c[i] <= s);
  forall(i in V, j in V : i < j)
    if (adj[i,j])
      m.post(c[i] != c[j]);

}
using {

   CPAS.generateSearch(m);

}
```

- Synthesized Search
- Comet Default search
- Tailored Search



Related Work

UCONN

- Aeon [Monette et al. 2009]
- Minion [Gent et al. 2006]
- Algorithm Portfolio (e.g., CPHYDRA) [O'Mahony et al. 2008]

Overview

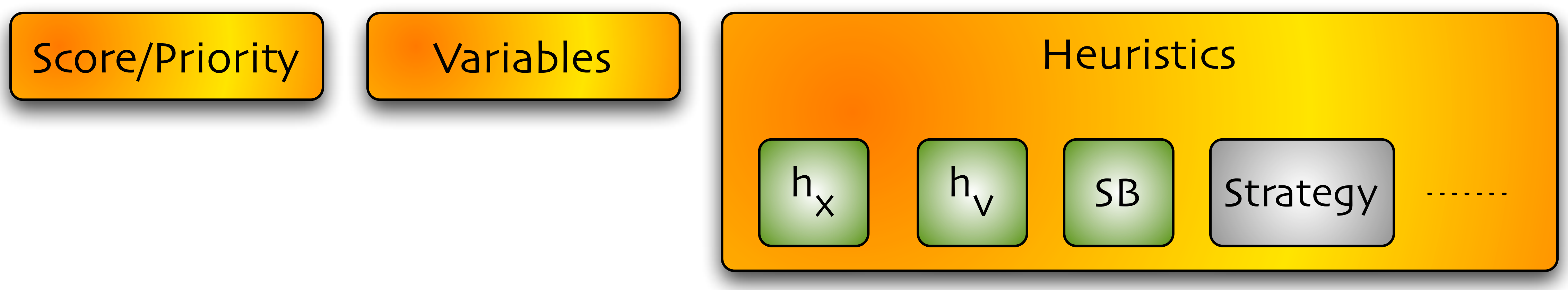
- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions

Rules

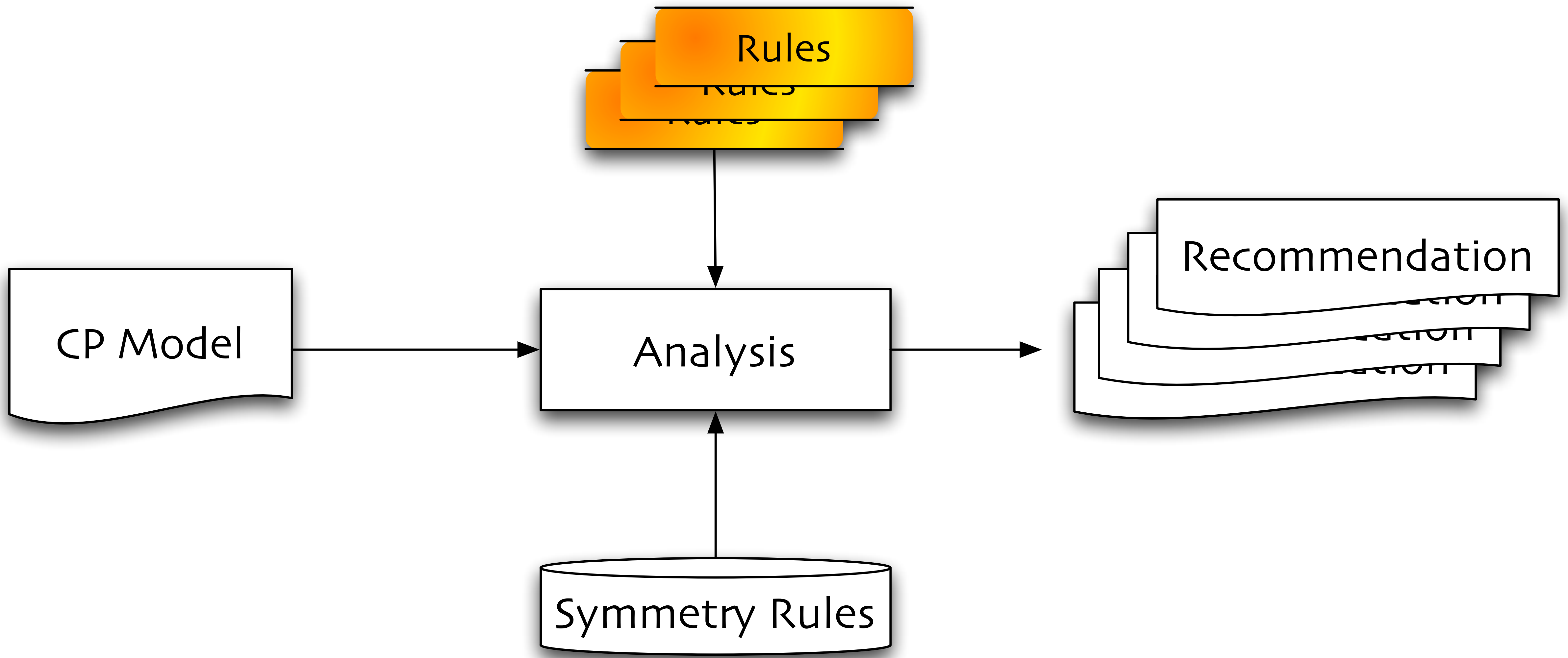
- Exploit model structures
- Generate a set of recommendations

Recommendations

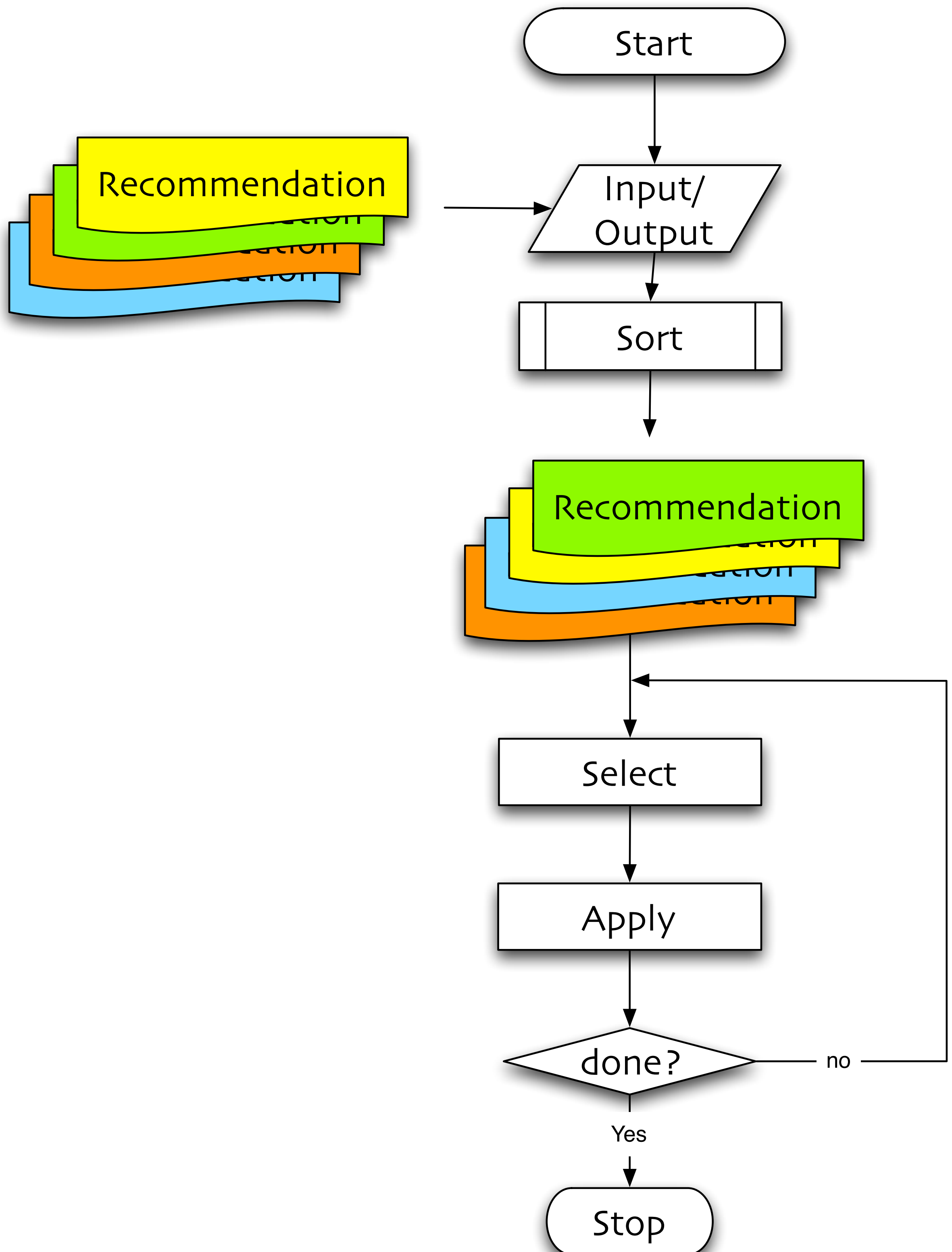
- Fully specify the search



Analysis



Search Generation



Overview

UCONN

- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions

Global Constraints Rules

- Insight
  - Capture global constraints structures
  - Currently support a subset of globals

Global Constraints Rules

- Insight
  - Capture global constraints structures
  - Currently support a subset of globals

AllDifferent
Knapsack
Sequence
Cardinality
Circuit
Regular
Spread
AtLeastNValue
....

Basics

UCONN

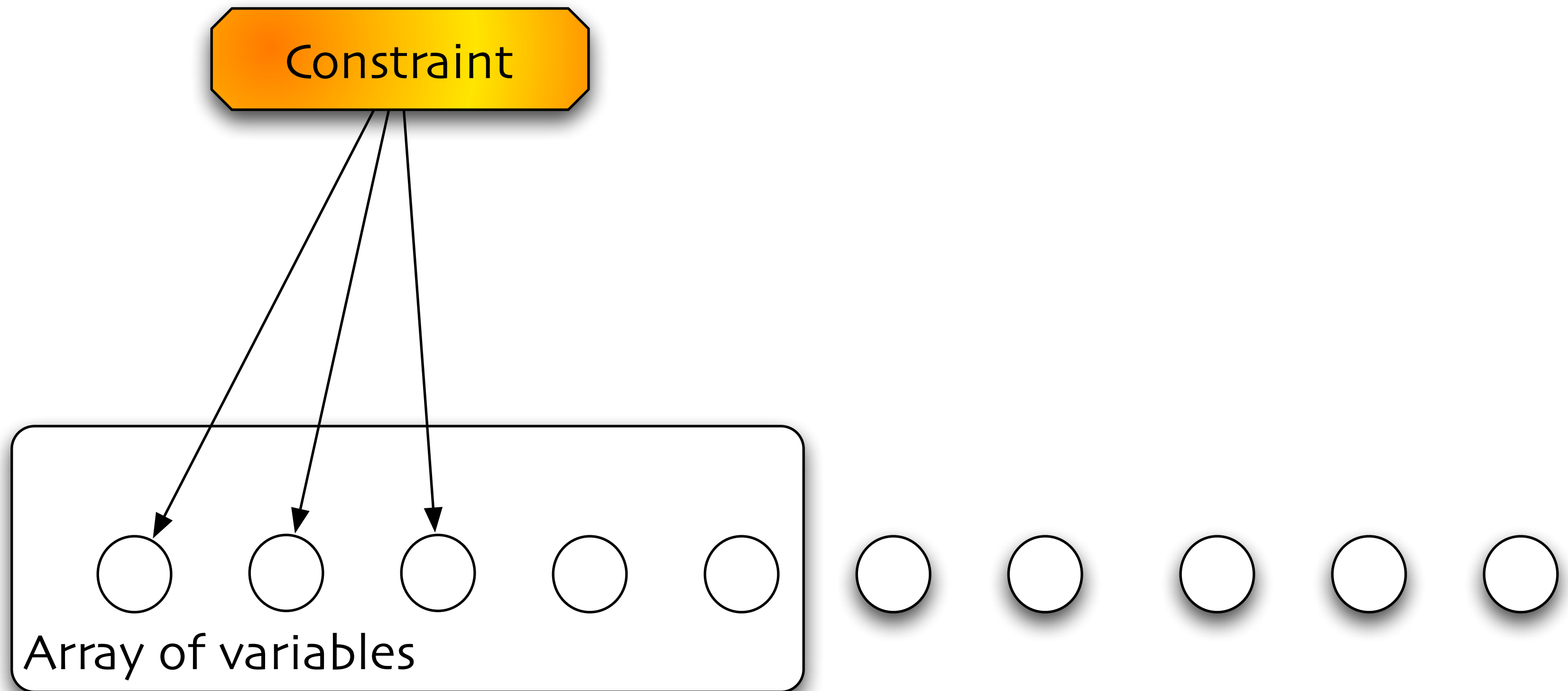
- Each global constraint (of a given type) issues one recommendation
  - Score
  - Variables affected
  - Heuristic selection

Scoring a global constraints

- Captures
  - Constraints covering
  - Constraints Homogeneity / Diversity
  - Constraints connectivity

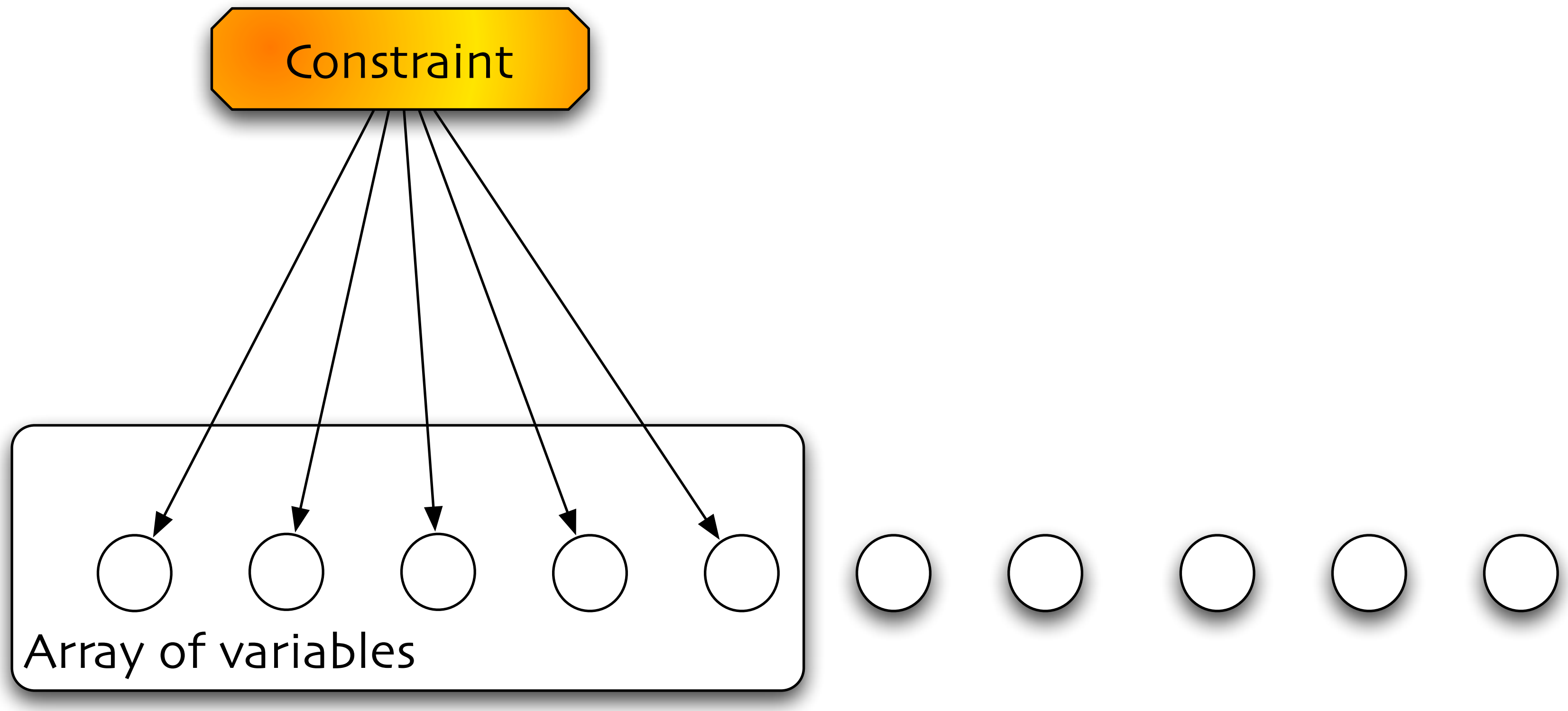
Covering

- Variable "coverage" Insight
  - Full vs. partial



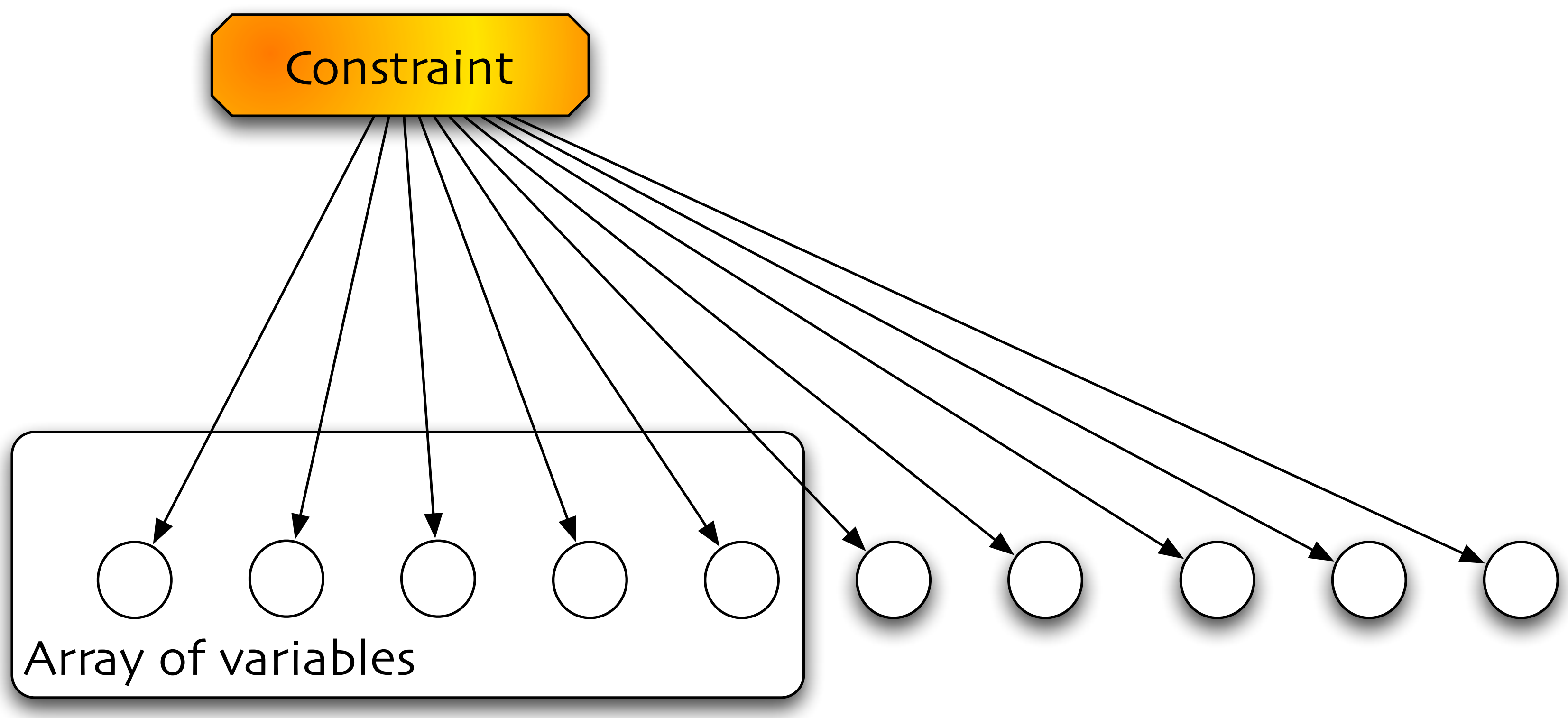
Covering

- Variable "coverage" Insight
  - Full vs. partial



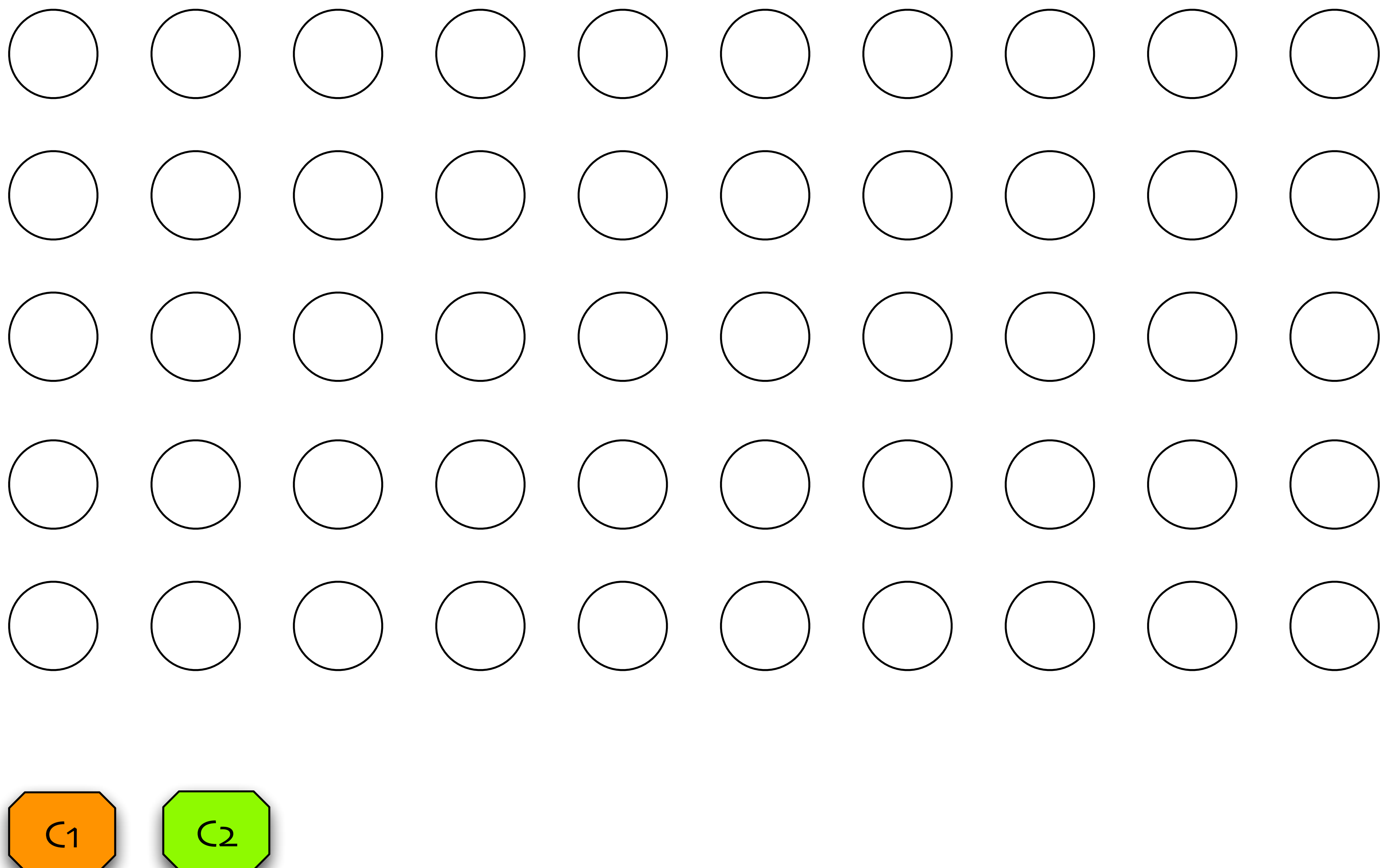
Covering

- Variable "coverage" Insight
  - Full vs. partial



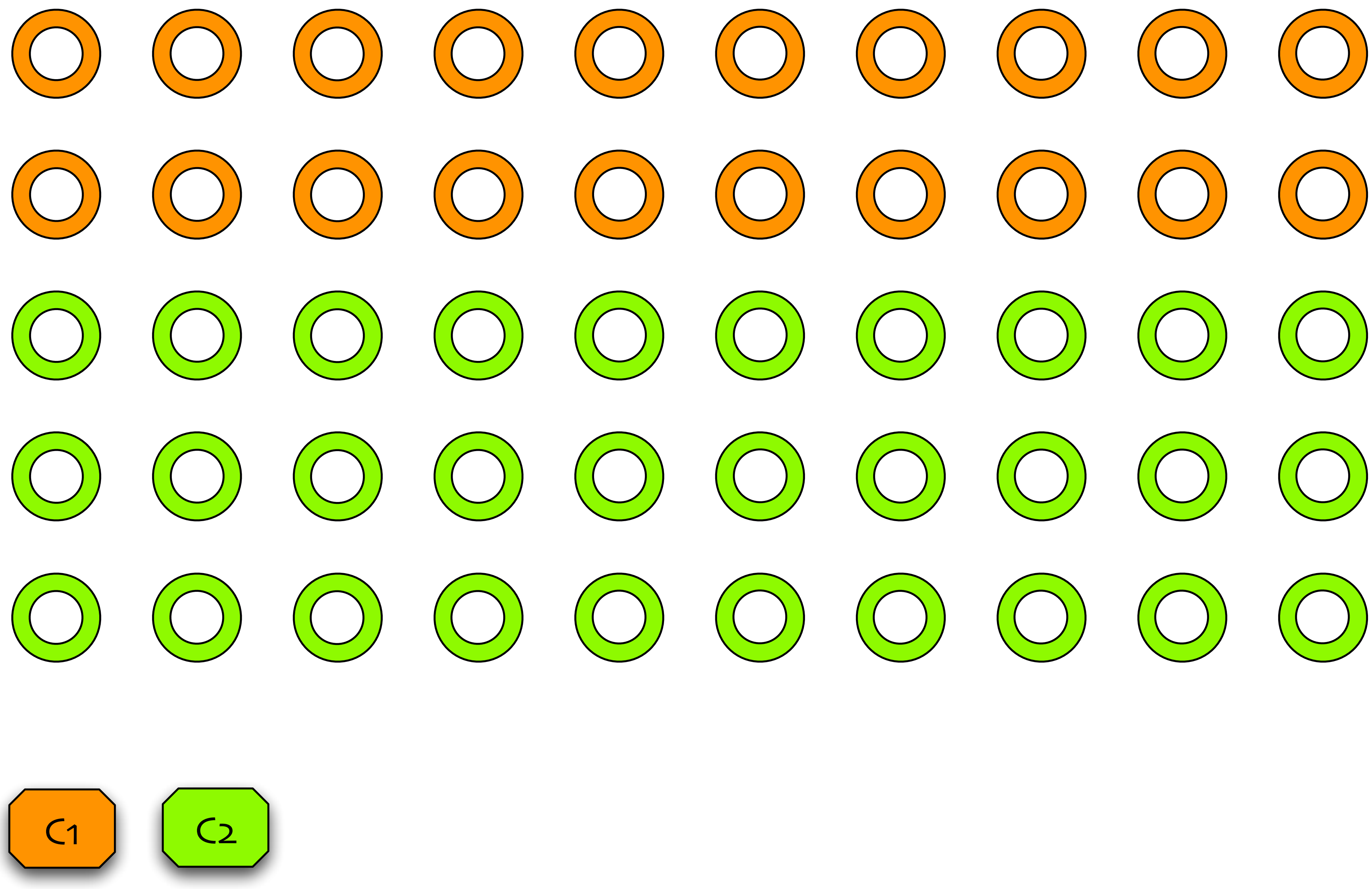
Homogeneity

- "coupling" Insight



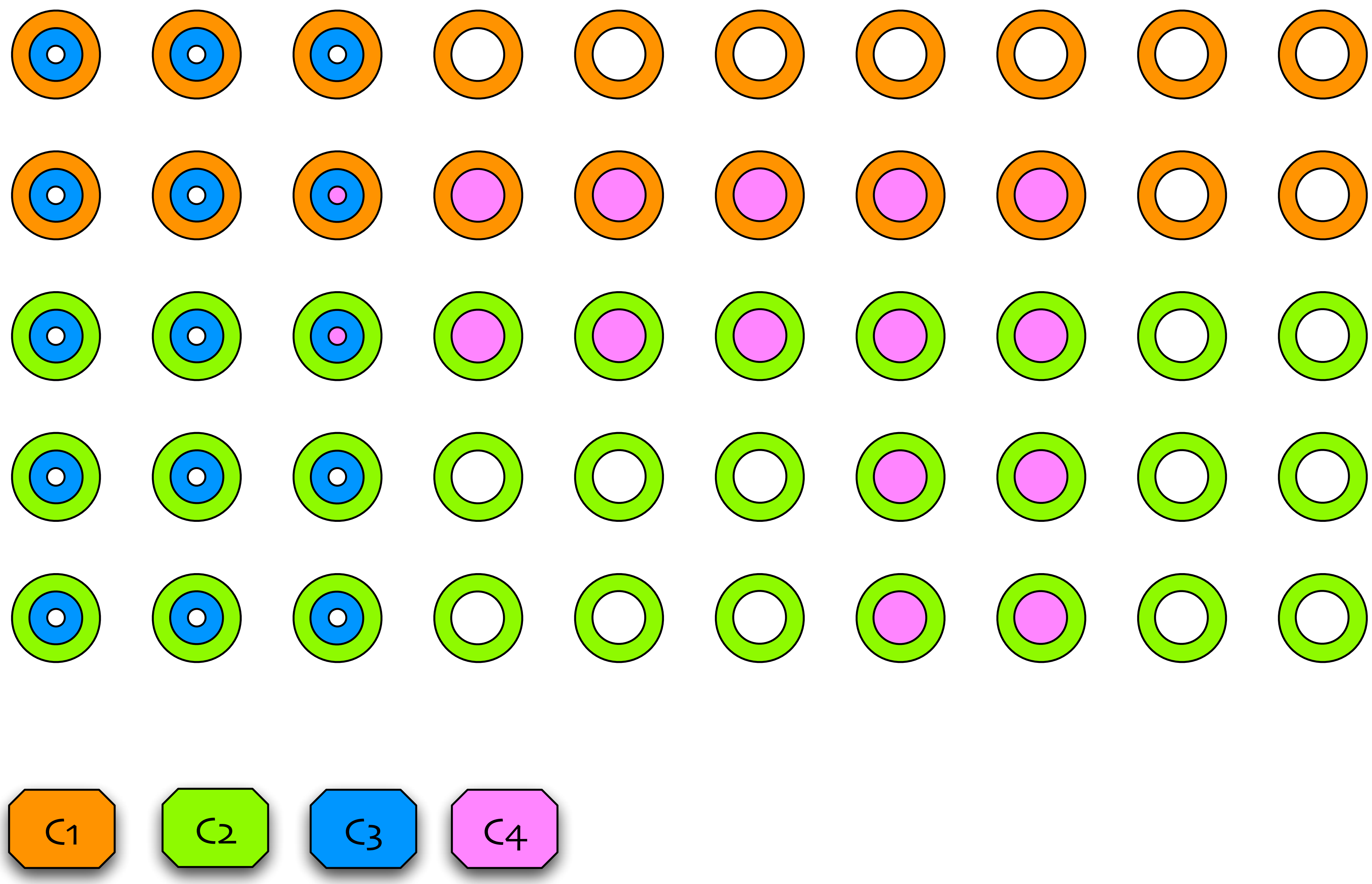
Homogeneity

- “coupling” Insight



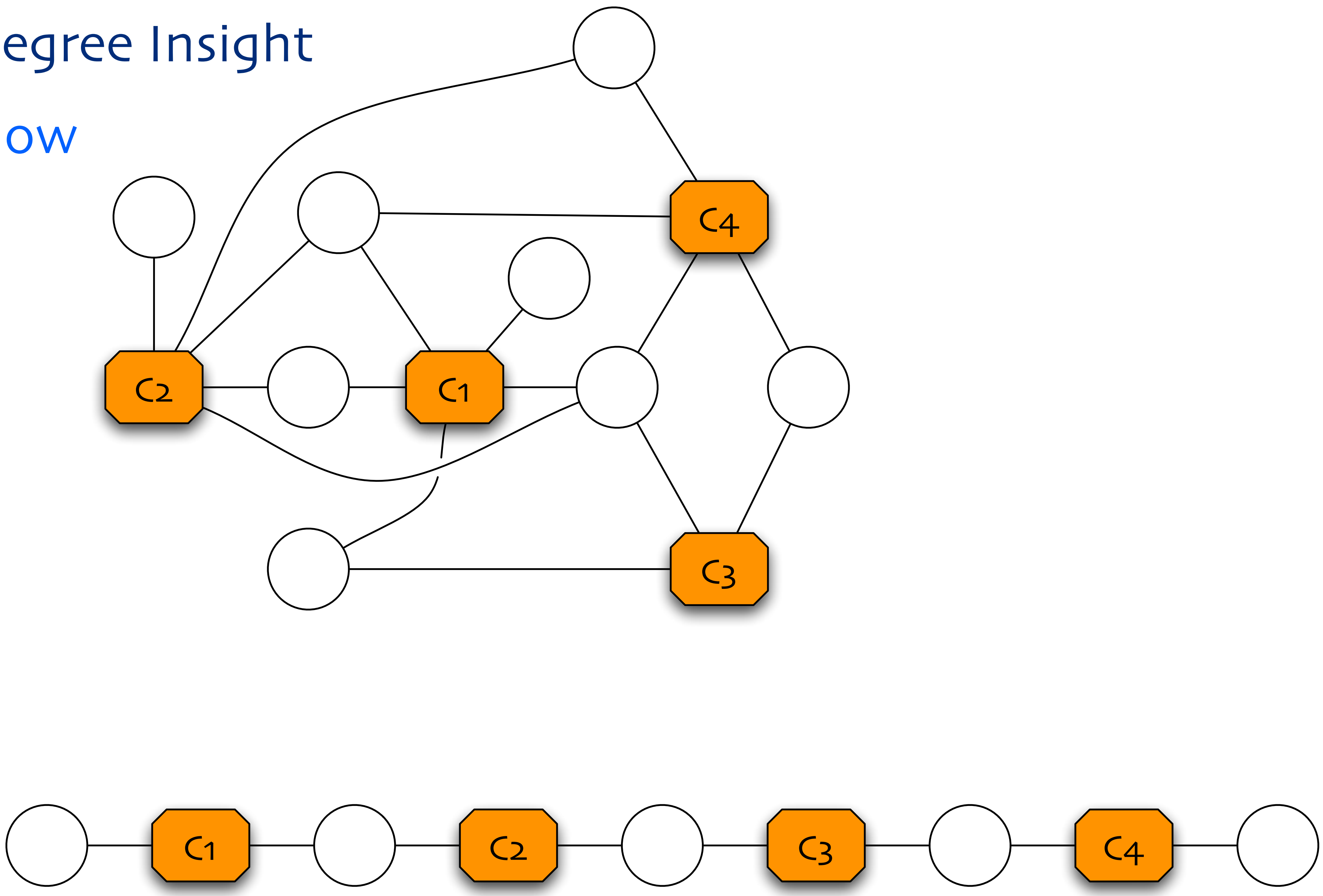
Homogeneity

- "coupling" Insight



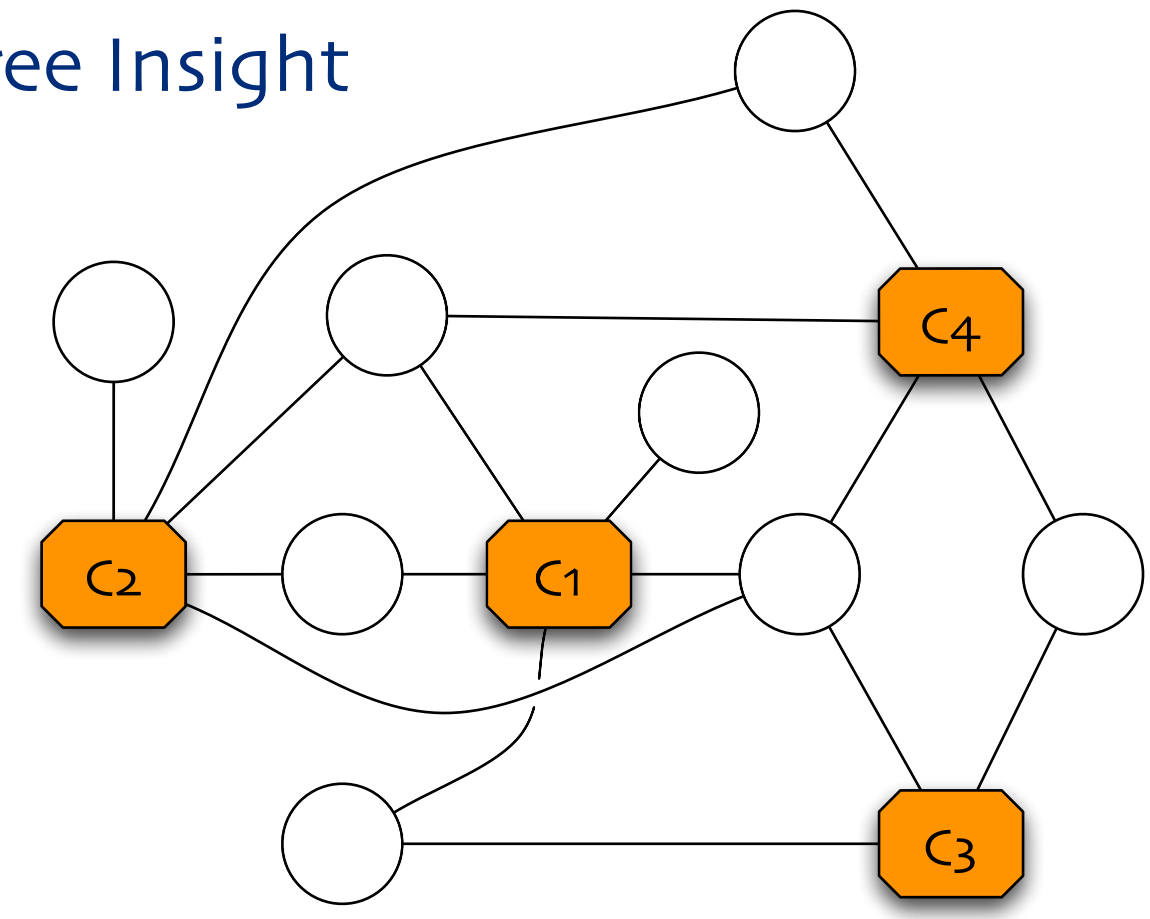
Connectivity

- Variables Degree Insight
  - High vs. low

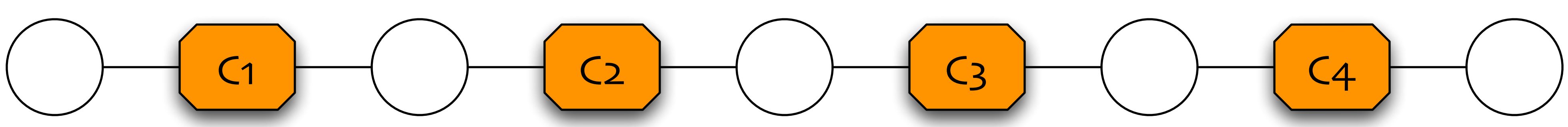


Connectivity

- Variables Degree Insight
  - High vs. low



High variables degree



Lower variables degree

Overall scoring

UCONN

- Integrate three elements
  - The more uniform the constraint types, the stronger the fit
  - The higher the variables degree, the stronger the fit
  - The higher the variables coverage, the stronger the fit

$$\frac{\max_{x \in vars(c)} deg(x)}{\max_{x \in X} deg(x) \cdot |G(C)|}$$

If covering

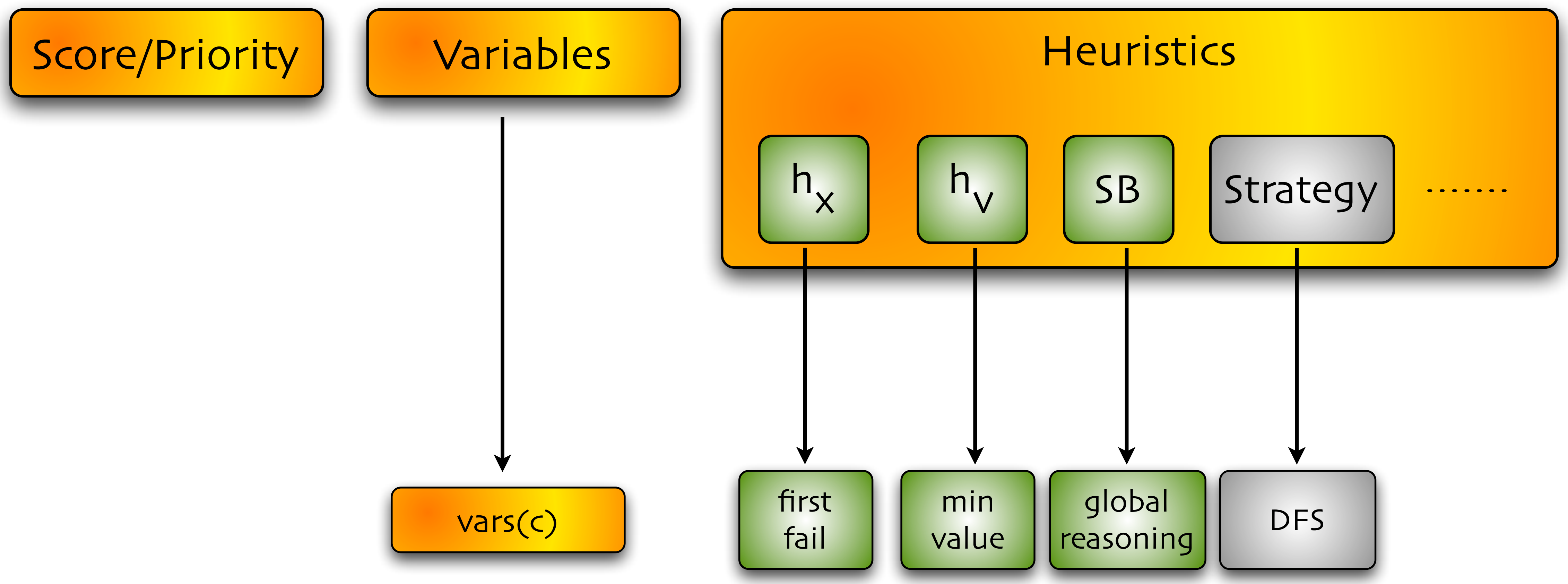
$$\frac{|vars(c)|}{\max_{k \in C: type(k)=type(c)} |vars(k)| \cdot |G(C)|}$$

Overall scoring

- Integrate three elements
  - The more uniform the constraint types, the stronger the fit
  - The higher the variables degree, the stronger the fit
  - The higher the variables coverage, the stronger the fit

$$\text{score} = \begin{cases} \dfrac{\max_{x \in vars(c)} deg(x)}{\max_{x \in X} deg(x) \cdot |G(C)|} & \text{If covering} \\[2ex] \dfrac{|vars(c)|}{\max_{k \in C : type(k) = type(c)} |vars(k)| \cdot |G(C)|} & \end{cases}$$

Global Constraints Rules

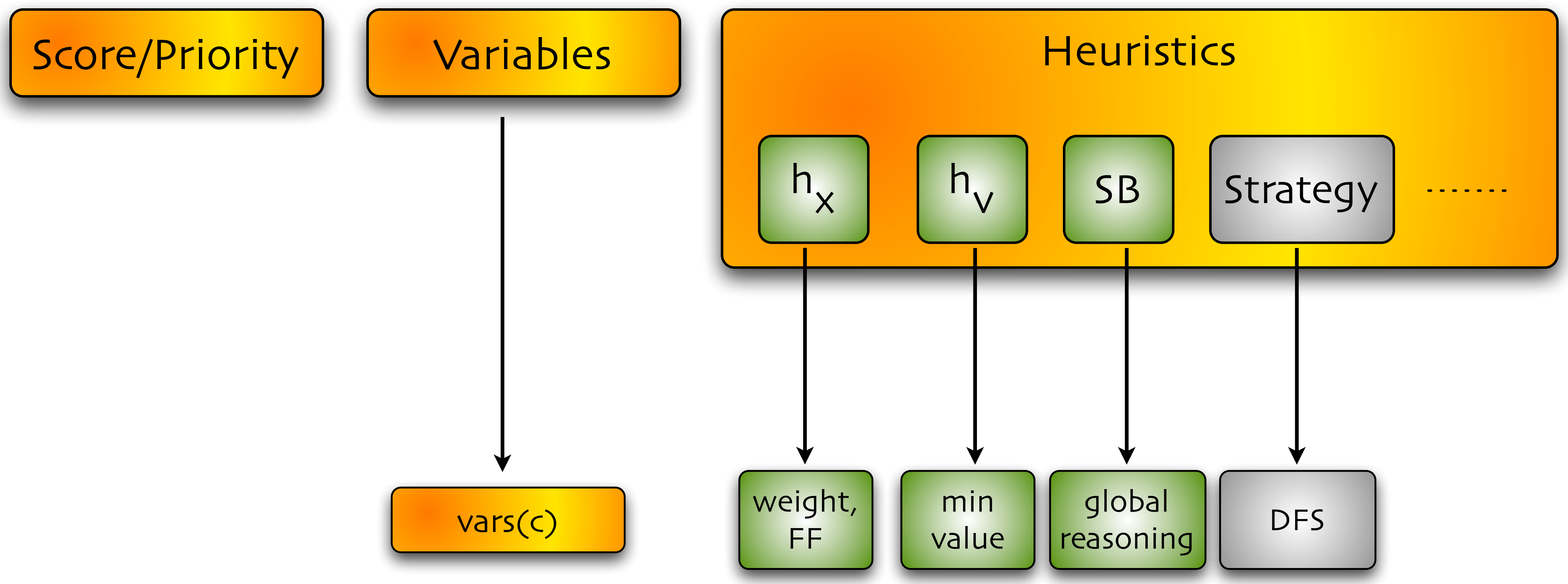


Knapsack Rule

- Full coverage of variables array
- Variable ordering by weight

$$\sum_{i \in N} w_i \cdot x_i \leq b$$

Knapsack Rule



Degree Rule

- Desirable if the static variable degrees are sufficiently diverse
- One recommendation for each model array
  - Compute relative degree frequencies (in [0..1]) $p_i = freq_i / |X|$
  - Get its score as

$$S_i = \left(1 - \sum_{i=1}^{z} p_i^2\right) \cdot \frac{\max_{x \in a} deg(x)}{\max_{x \in X} deg(x)}$$

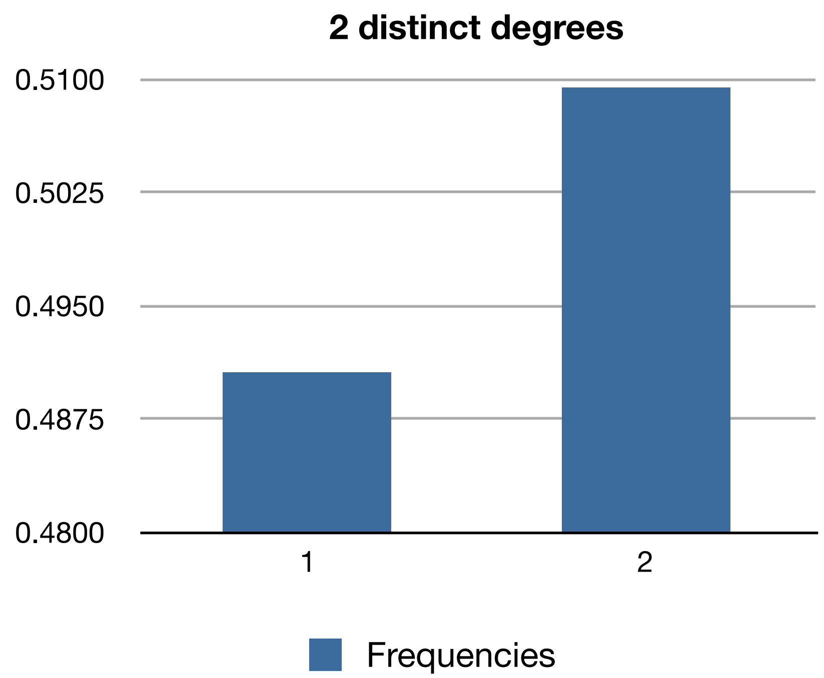
Frequencies

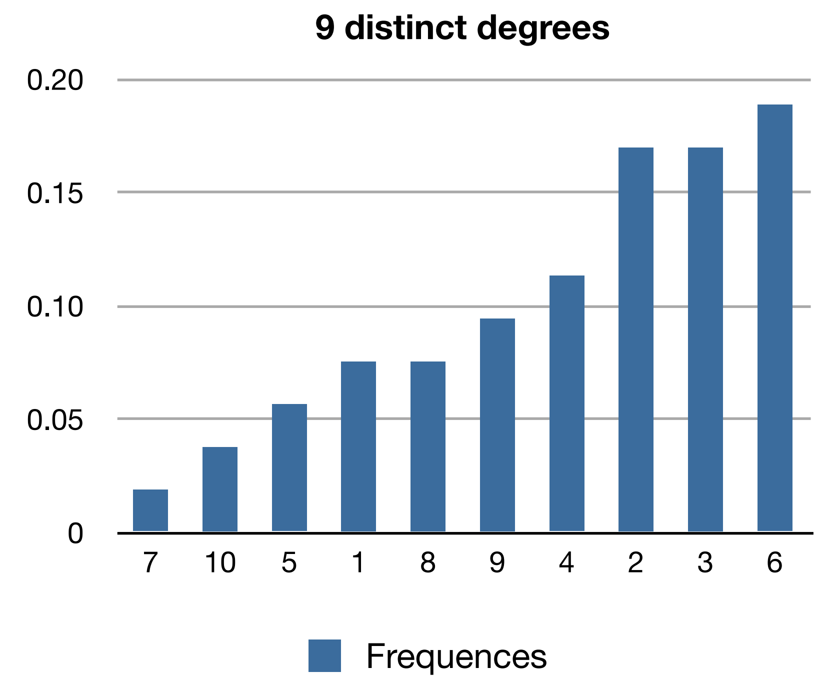
Frequences

Degree Rule

- Desirable if the static variable degrees are sufficiently diverse
- One recommendation for each model array
  - Compute relative degree frequencies (in [0..1]) $p_i = freq_i / |X|$
  - Get its score as

$$S_i = \left(1 - \sum_{i=1}^{z} p_i^2\right) \cdot \frac{\max_{x \in a} deg(x)}{\max_{x \in X} deg(x)}$$

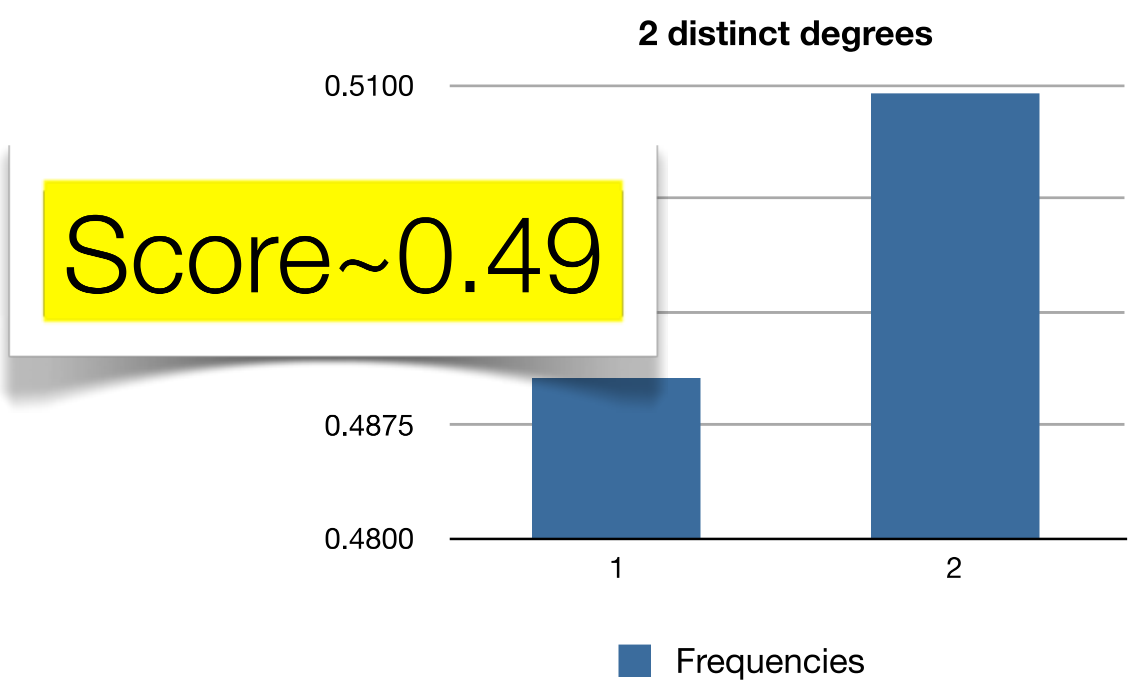


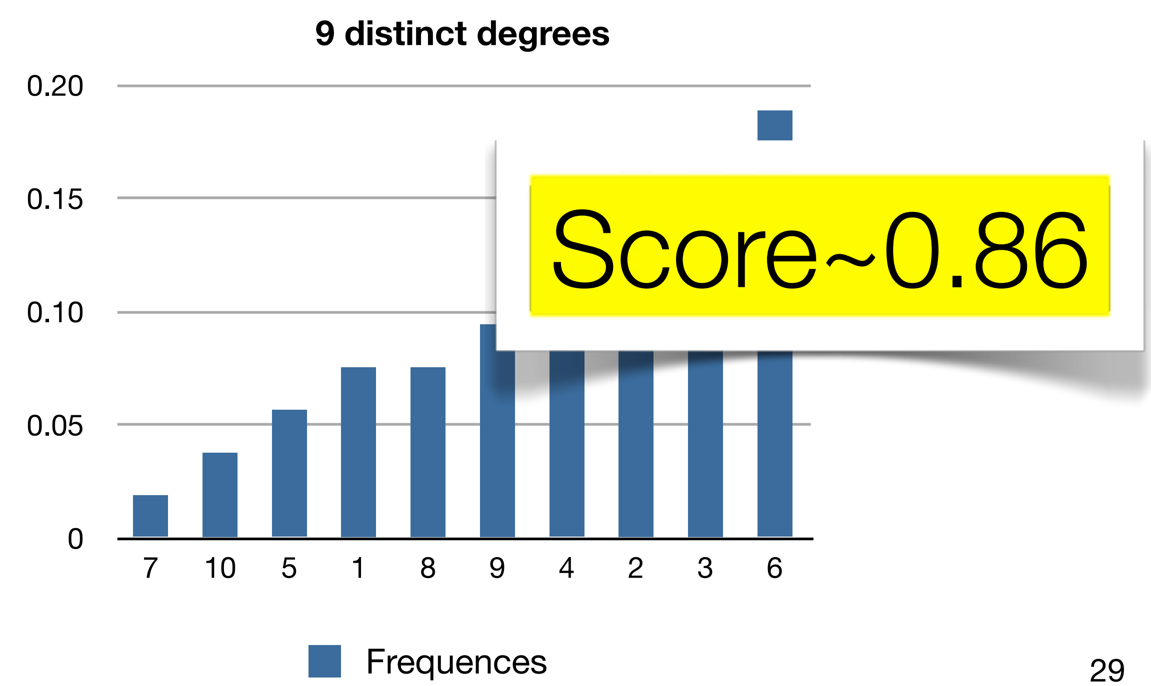


Degree Rule

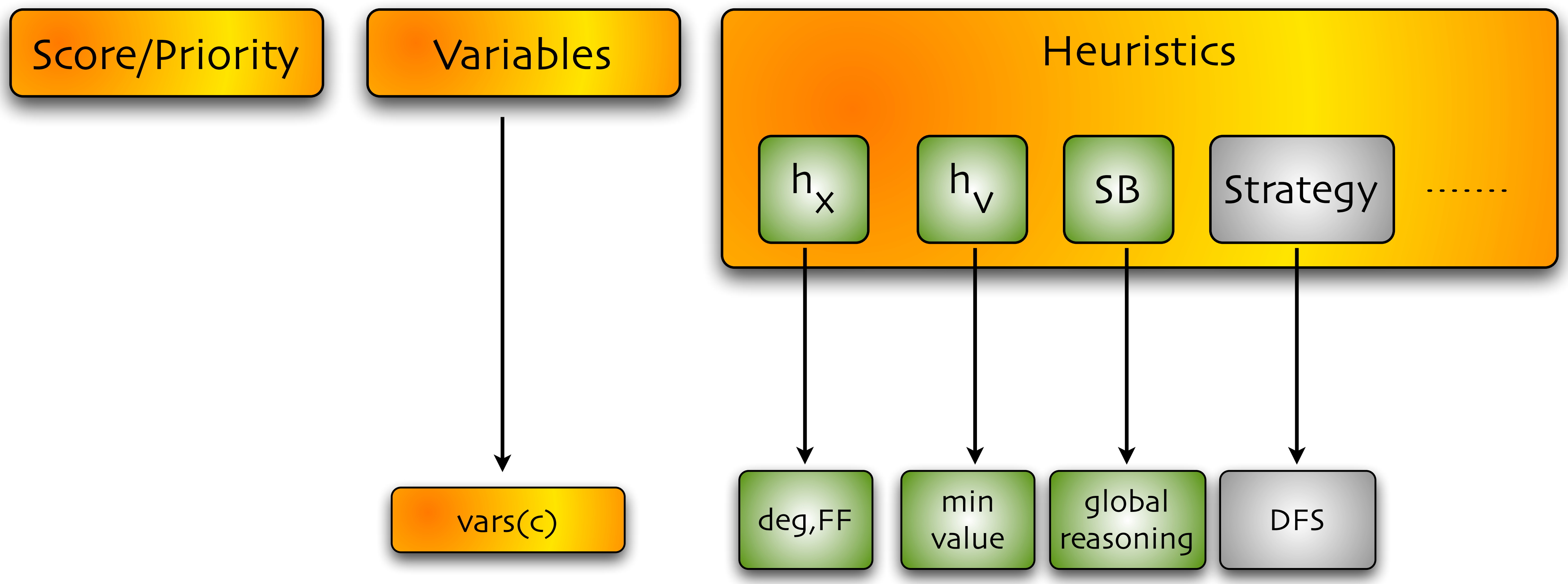
- Desirable if the static variable degrees are sufficiently diverse
- One recommendation for each model array
  - Compute relative degree frequencies (in [0..1]) $p_i = freq_i/|X|$
  - Get its score as

$$S_i = \left(1 - \sum_{i=1}^{z} p_i^2\right) \cdot \frac{\max_{x \in a} deg(x)}{\max_{x \in X} deg(x)}$$





Degree Rule

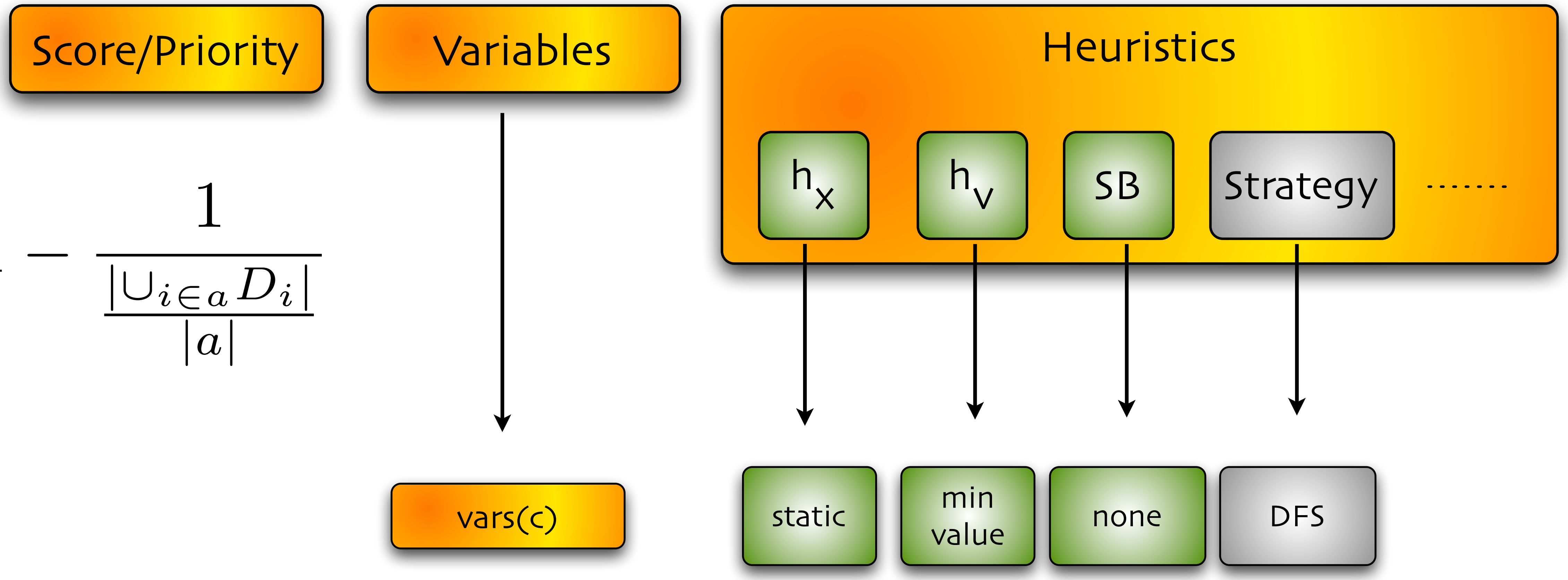


Score/Priority
Variables
Heuristics
h_x
h_v
SB
Strategy
....
vars(c)
deg,FF
min value
global reasoning
DFS

Pick Value First Rule

UCONN

- Insight
  - When there are far more values than variables!
  - Pick a value first. Choose a variable to assign second
- Score
  - Measure value density vs. array size



$$1 - \frac{1}{\frac{|\cup_{i \in a} D_i|}{|a|}}$$

Most Constrained Variables Rule

- Insight
  - Variable centric rule
  - Captures the traditional static degree.

First Fail Rule

- Insight
  - Simple “default” rule
  - Used to label variables not handled by any dedicated rule

Overview

UCONN

- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions

Symmetry Breaking

- Symmetry is everywhere!
- Compositional global constraints driven approach [Van Hentenryck et al., Eriksson 2005]
- Symmetry analysis with patterns
- Once symmetry detected, go and break it!

Overview

UCONN

- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions

Mechanics

- Recommendations are composable
- Lexicographic order of score & priority
- A polymorphic method '`label`'

```
forall(r in rec.getKeys()) by (-rec{r}.getScore(), rec{r}.getPriority()) {
  rec{r}.label();
  if (solver.isBound()) break;
}
```

Mechanics

```
class VariableRecommendation implements Recommendation { ...
   void label() {
      var<CP>{int}[] x = getVars();
      while(!bound(x)) {
         selectMin(i in unboundVars(x))(hx(x,i)) {
            set{int} values = getValues(x[i]);
            tryall<solver>(v in values) by (hv(values,v))
               solver.label(x[i], v);
            onFailure solver.diff(x[i], v);
         }
      }
   }
}
```

Mechanics

UCONN

```
class ValueRecommendation implements Recommendation { ...
   void label() {
      var<CP>{int}[] x = getVars();
      while(!bound(x)) {
         set{int} values = collect(s in x.getRange():!x[s].bound()) x[s].getMin();
         selectMin(v in values) (hv(values,v)) {
           tryall<solver>(i in unboundVars(x)) by (hx(x,i)) {
               solver.label(x[i], v);
            onFailure solver.diff(x[i], v);
           }
        }
      }
   }
}
```

Overview

UCONN

- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions

Benchmarks

- More than 10 models known for having non-trivial search

Progressive Party

```
Solver<CP> m();
var<CP>{int} boat[Guests,Periods](m,Hosts);
solve<m> {
   forall(g in Guests)
      m.post(alldifferent(all(p in Periods) boat[g,p]),onDomains);
   forall(p in Periods)
      m.post(multiknapsack(all(g in Guests) boat[g,p],crew,cap));
   forall(i in Guests, j in Guests : j > i)
      m.post(sum(p in Periods) (boat[i,p] == boat[j,p]) <= 1);}
CPAS.generateSearch(m);
```

Progressive Party

UCONN

```
Solver<CP> m();
var<CP>{int} boat[Guests,Periods](m,Hosts);
solve<m> {
   forall(g in Guests)
      m.post(alldifferent(all(p in Periods) boat[g,p]),onDomains);
   forall(p in Periods)
      m.post(multiknapsack(all(g in Guests) boat[g,p],crew,cap));
   forall(i in Guests, j in Guests : j > i)
      m.post(sum(p in Periods) (boat[i,p] == boat[j,p]) <= 1);}
CPAS.generateSearch(m);
```

Alldifferent **(0.25)**

Alldifferent **(0.25)**

Alldifferent **(0.25)**

Alldifferent **(0.25)**

Knapsack **(0.5)**

Knapsack **(0.5)**

Knapsack **(0.5)**

Knapsack **(0.5)**

Knapsack **(0.5)**

Progressive Party

```
Solver<CP> m();
var<CP>{int} boat[Guests,Periods](m,Hosts);
solve<m> {
   forall(g in Guests)
      m.post(alldifferent(all(p in Periods) boat[g,p]),onDomains);
   forall(p in Periods)
      m.post(multiknapsack(all(g in Guests) boat[g,p],crew,cap));
   forall(i in Guests, j in Guests : j > i)
      m.post(sum(p in Periods) (boat[i,p] == boat[j,p]) <= 1);}
CPAS.generateSearch(m);
```

Knapsack **(0.5)**

Knapsack **(0.5)**

Knapsack **(0.5)**

Knapsack **(0.5)**

Knapsack **(0.5)**

Alldifferent **(0.25)**

Alldifferent **(0.25)**

Alldifferent **(0.25)**

Alldifferent **(0.25)**

Symmetry Breaking **OFF**

Scene Allocation

UCONN

```
Solver<CP> m();
var<CP>{int} shoot[Scenes](m,Days);
var<CP>{int} nbd[Actor](m,Days);
int up[i in Days] = 5;
minimize<m> sum(a in Actor) fee[a] * nbd[a]
subject to {
   forall(a in Actor)
     m.post(nbd[a]==sum(d in Days) (or(s in which[a]) shoot[s]==d));
   m.post(atmost(up,shoot),onDomains);}
CPAS.generateSearch(m);
```

Scene Allocation

UCONN

```
Solver<CP> m();
var<CP>{int} shoot[Scenes](m,Days);
var<CP>{int} nbd[Actor](m,Days);
int up[i in Days] = 5;
minimize<m> sum(a in Actor) fee[a] * nbd[a]
subject to {
   forall(a in Actor)
     m.post(nbd[a]==sum(d in Days) (or(s in which[a]) shoot[s]==d));
   m.post(atmost(up,shoot),onDomains);}
CPAS.generateSearch(m);
```

Degree **(0.15)**

Degree **(0.03)**

Cardinality **(0.10)**

Scene Allocation

```
Solver<CP> m();
var<CP>{int} shoot[Scenes](m,Days);
var<CP>{int} nbd[Actor](m,Days);
int up[i in Days] = 5;
minimize<m> sum(a in Actor) fee[a] * nbd[a]
subject to {
   forall(a in Actor)
     m.post(nbd[a]==sum(d in Days) (or(s in which[a]) shoot[s]==d));
   m.post(atmost(up,shoot),onDomains);}
CPAS.generateSearch(m);
```

Degree **(0.25)**

Cardinality **(0.10)**

Degree **(0.03)**

Symmetry Breaking **ON**

Steel Slab Mill

```
Solver<CP> m();
var<CP>{int} x[Orders](m,Slabs);
var<CP>{int} l[Slabs](m,0..maxCap);
var<CP>{int} obj(m,0..nbSlabs*maxCap);
int loss[c in 0..maxCap] = min(i in Caps:capacities[i] >= c) capacities[i]-c;
minimize<m> obj subject to {
  m.post(obj == sum(s in Slabs) loss[l[s]]);
  m.post(multiknapsack(x,weight,l));
  forall(s in Slabs)
    m.post(sum(c in Colors) (or(o in colorOrders[c]) (x[o] == s)) <= 2);}
CPAS.generateSearch(m);
```

Steel Slab Mill

UCONN

```
Solver<CP> m();
var<CP>{int} x[Orders](m,Slabs);
var<CP>{int} l[Slabs](m,0..maxCap);
var<CP>{int} obj(m,0..nbSlabs*maxCap);
int loss[c in 0..maxCap] = min(i in Caps:capacities[i] >= c) capacities[i]-c;
minimize<m> obj subject to {
  m.post(obj == sum(s in Slabs) loss[l[s]]);
  m.post(multiknapsack(x,weight,l));
  forall(s in Slabs)
    m.post(sum(c in Colors) (or(o in colorOrders[c]) (x[o] == s)) <= 2);}
CPAS.generateSearch(m);
```

Knapsack **(1)**

Degree **(0.1)**

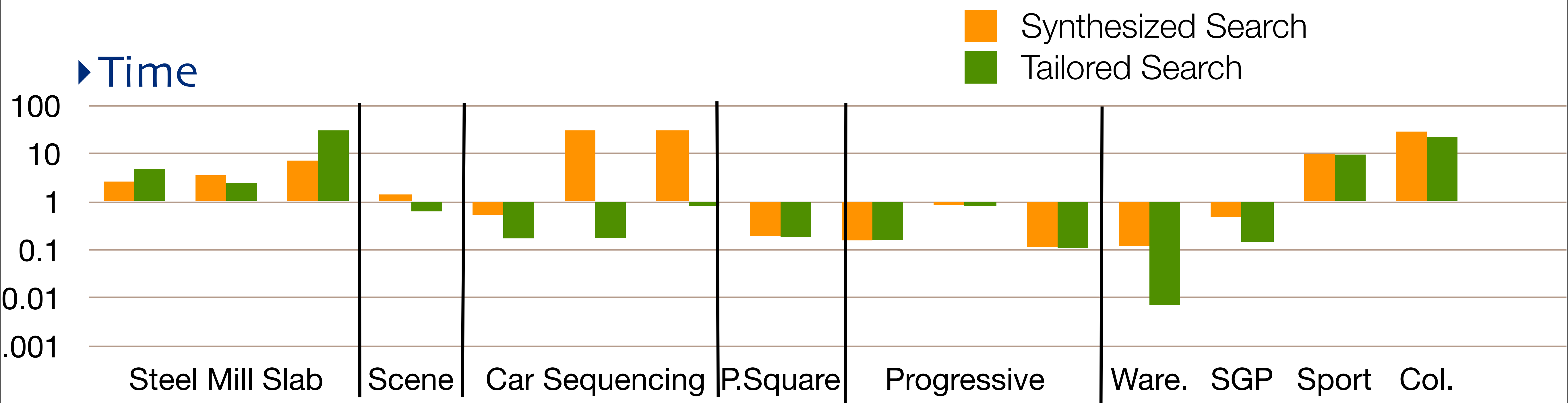
Symmetry Breaking **ON**

Overview

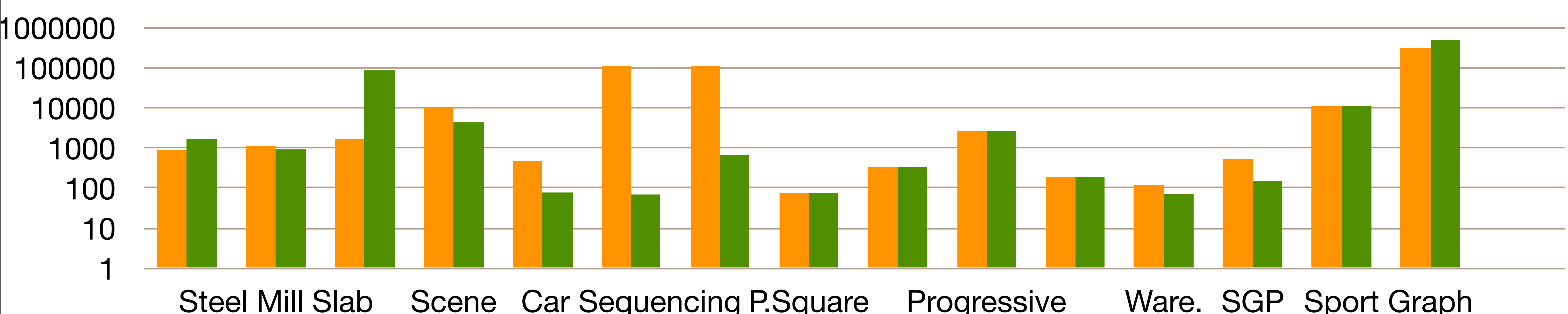
- Motivation
- Introducing CP-AS
- Synthesis Process
  - Rules Library
  - Symmetry Breaking
  - Implementation
- Example Applications
- Experimental Results
- Conclusions

Experimental Results

- Time

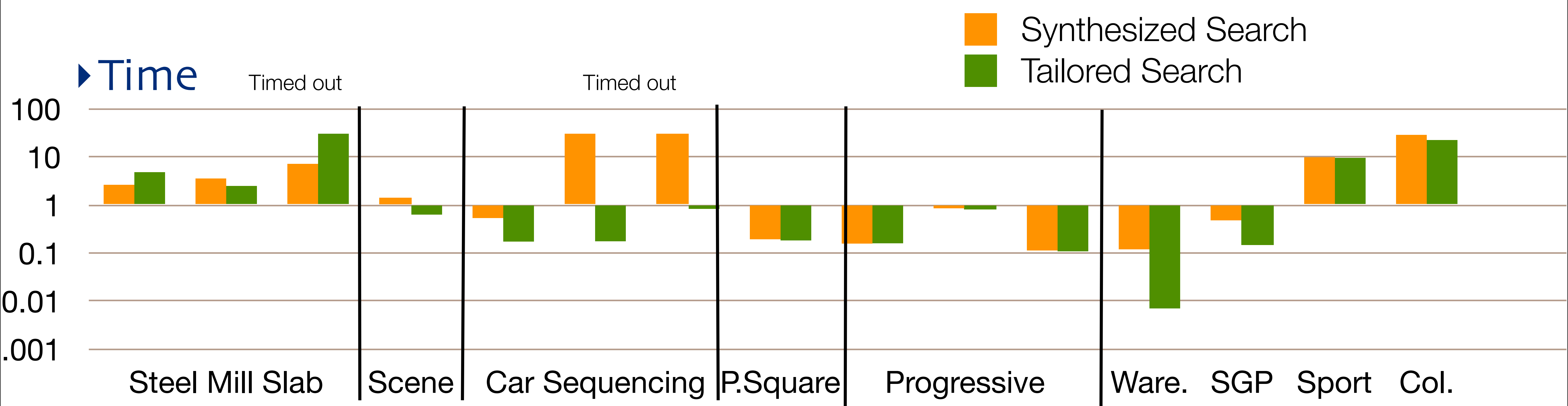


- Choice Points

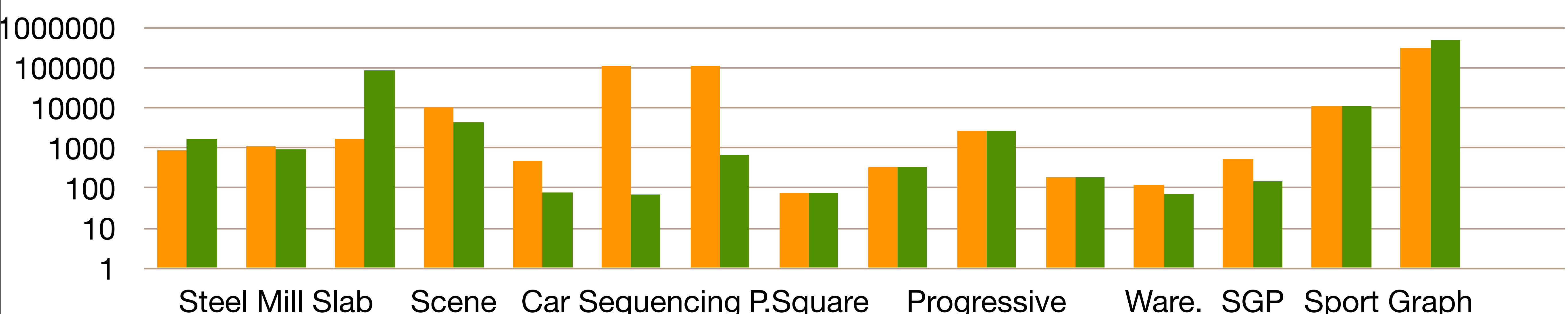


Experimental Results

- Time

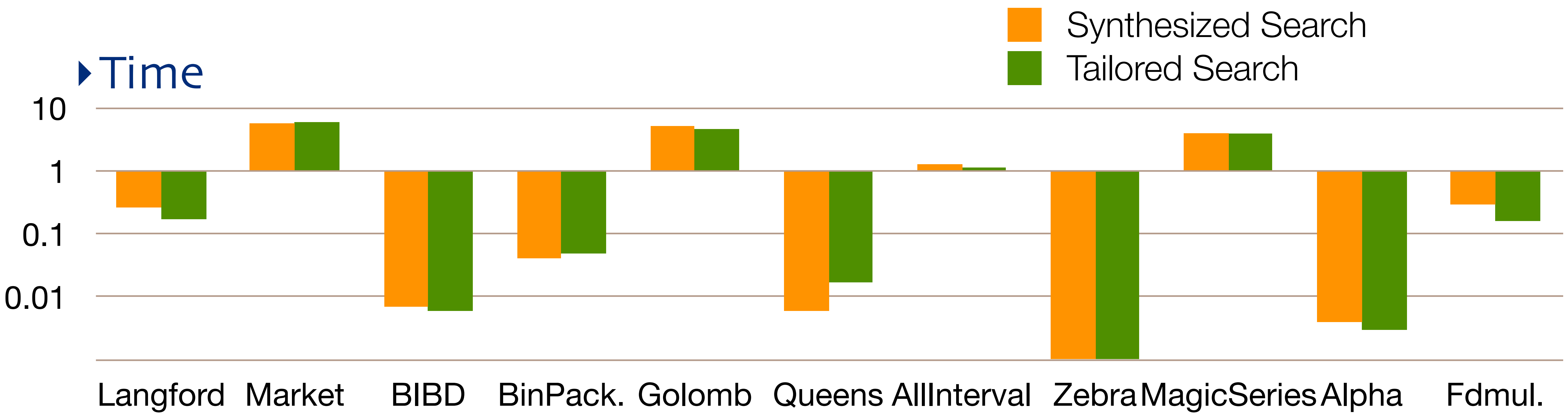


- Choice Points

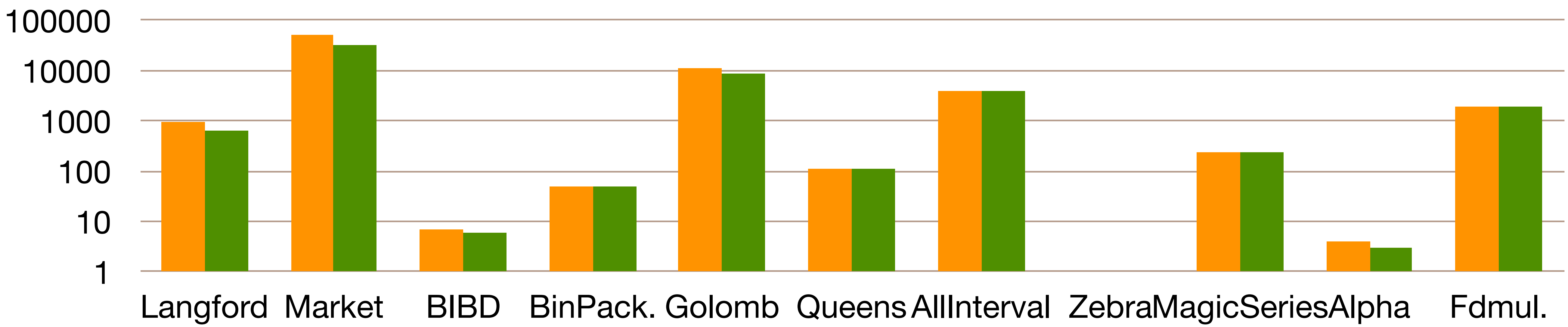


Experimental Results

- Time



- Choice Points



Conclusions & Future Work

- Synthesized search is competitive
- No significant degradation in performance
- Work remains to augment the rule set
- Improvements to:
  - Composition mechanism
  - Symmetry breaking inference engine
  - Value Heuristics & Search strategies
- An in-depth empirical evaluation is absolutely essential