

# Declarative Heuristic Search for Pattern Set Mining



*Tias Guns* – Siegfried Nijssen – Albrecht Zimmermann – Luc De Raedt  
DTAI, KU Leuven, Belgium

**Declarative Pattern Mining workshop, ICDM 2011.**

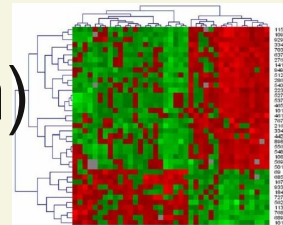
# Context: pattern mining

**Goal:** find patterns in (transactional) data

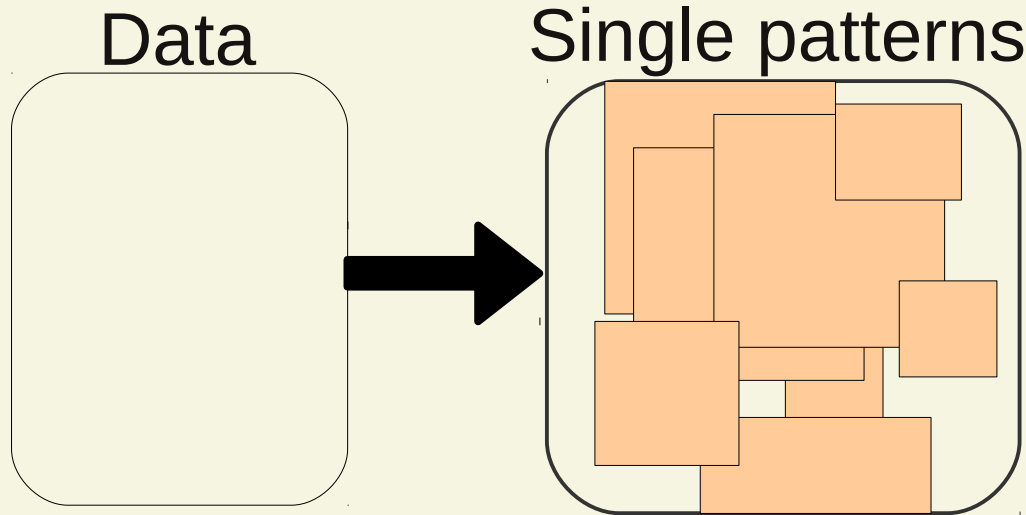
- better understanding of data
- find novel information

## Applications:

- online shops
- weblog analysis
- microarray analysis (gene expression)
- learning taxonomies
- text analysis (privacy leaks)
- ...

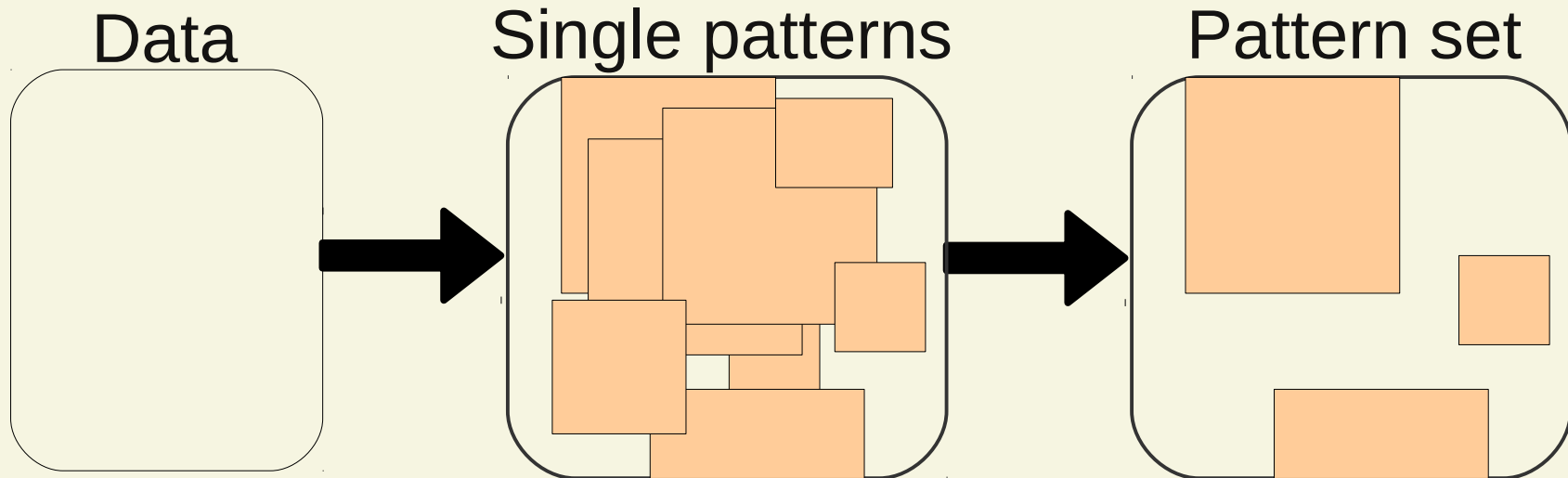


# Pattern Mining



**In practice:** Too many (redundant) patterns

# Pattern **Set** Mining

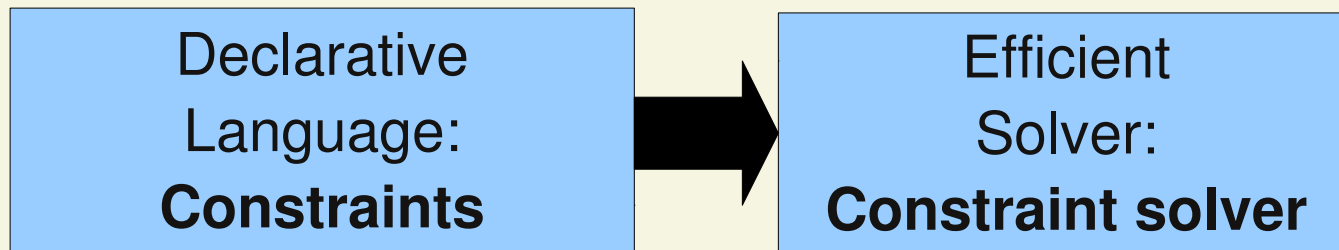


**Goal:** find *small* set of coherent patterns

- Classifier construction
- Clustering
- Summarization

# Declarative pattern (set) mining

CP4IM: Constraint Programming for Itemset Mining



## Constraint Programming

- Declarative: model + search
- Flexible: independent constraints
- General: easy to combine constraints

# Example specification (comet)

```
Solver<CP> cp();

var<CP> {bool} Items[1..NrI](cp);
var<CP> {bool} Trans[1..NrT](cp);

solveall<cp> {
  forall (t in 1..NrT) // Trans covered by Items
    cp.post(Trans[t] ==
      sum(i in 1..NrI: !TDB[t].contains(i)) Items[i] <= 0 );
  forall (i in 1..NrI) // all Items frequent
    cp.post(Items[i] =>
      sum(t in 1..NrT: TDB[t].contains(i)) Trans[t] >= Freq);
}
using { label(Items); }
```

# CP4IM: generality

## Pattern mining tasks:

- Frequent, closed and maximal itemset mining
- Mining with costs (convertible and more)
- Discriminative I.M. (correlated I.M./ subgroup discover/.)

## Pattern set mining tasks: (fixes size $k$ )

- $k$  concept learning ( $k$  concepts/patterns)
- conceptual clustering ( $k$  clusters)
- $k$ -tiling

+ combinations!

# CP4IM search strategy

- Standard Gecode CP solver (not optimised for DM)
- Exhaustive search
  - not common in pattern *set* mining,

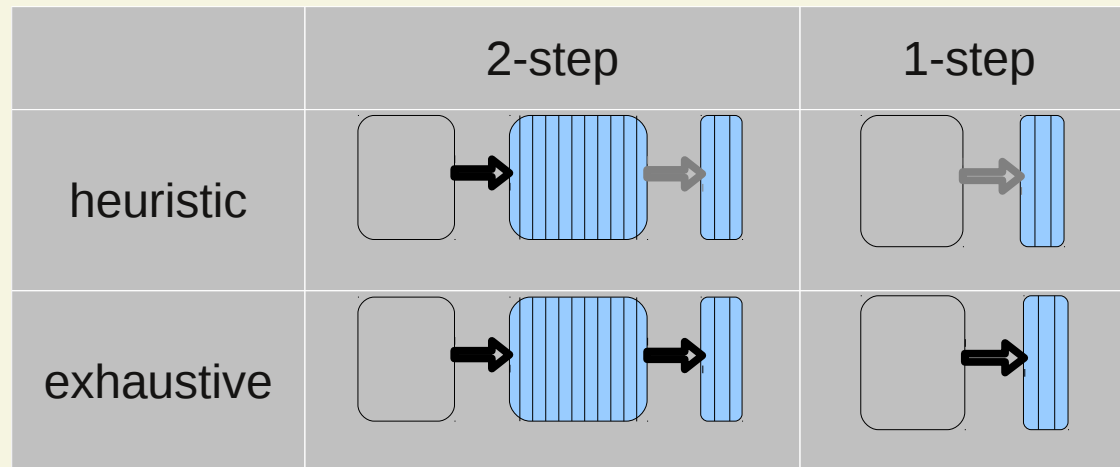
For pattern set mining:

- search space  $O(2^k \cdot |I|)$  for  $k$  patterns
- often heuristic techniques  
(e.g. post-processing, greedy covering, beam search, ...)

# Search strategies

Evaluating pattern set mining strategies, in a constraint programming framework

(T. Guns, S. Nijssen, L. De Raedt, PAKDD 2011).



→ requires ad-hoc wrapper scripts, **NOT** declarative

Heuristic search  
in declarative framework?

# Declarative Exhaustive Search In Constraint Programming

# Declarative exhaustive search

In many languages: parameterized 'label' procedure:  
pick a variable, pick a value.

New approach allowing more expressive search:  
'Comet' system (Pascal Van Hentenryck)

```
Solver<CP> cp();
```

```
maximize<cp>
```

```
...
```

```
subject to {
```

```
...
```

```
}
```

```
using { ... }
```

```
} model
```

```
} search
```


# Declarative exhaustive search

- pattern-first

```
using {  
  forall (k in 1..K)  
    forall (i in 1..NrI) by (freqs[i])  
      try<cp> cp.label(Items[k,i], 0);  
        | cp.diff(Items[k,i], 0);  
}
```

- item-first

```
using {  
  forall (i in 1..NrI) by (freqs[i])  
    forall (k in 1..K)  
      try<cp> cp.label(Items[k,i], 0);  
        | cp.diff(Items[k,i], 0);  
}
```



# Declarative Heuristic Search In Constraint Programming

# Declarative heuristic search

New in Constraint Programming (~4 years)

Only in 'Comet' language (for now)

```
Solver<CP> cp();  
cp.InsOnFailure(1000);
```

```
maximize<cp>
```

```
...
```

```
subject to {
```

```
...
```

```
}
```

} model

```
using { ... }
```

```
onRestart {
```

```
...
```

```
}
```

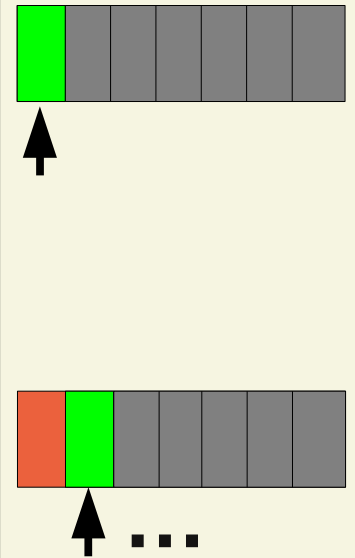
} search

# Different search

## 3) Greedily construct pattern set:

find one pattern, add a pattern, add a pattern, ...

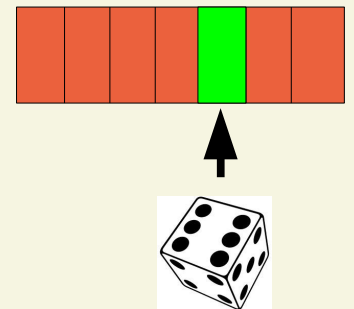
```
onRestart {  
  if (iter == 1) {  
    forall (k in 2..K, i in 1..NrI)  
      // ignore all but first pattern in search  
      cp.label(Items[k,i],1);  
  } else {  
    if (iter > K) { cp.exit(); }  
    Solution s = cp.getSolution();  
    forall (k in 1..K: k!=iter, i in 1..NrI)  
      // ignore all but k'th pattern in search  
      cp.post( Items[k,i] == Items[k,i].getSnapshot(s));  
  }  
  iter := iter+1;  
}
```



# Different search

2) Remove a random pattern, find best replacement

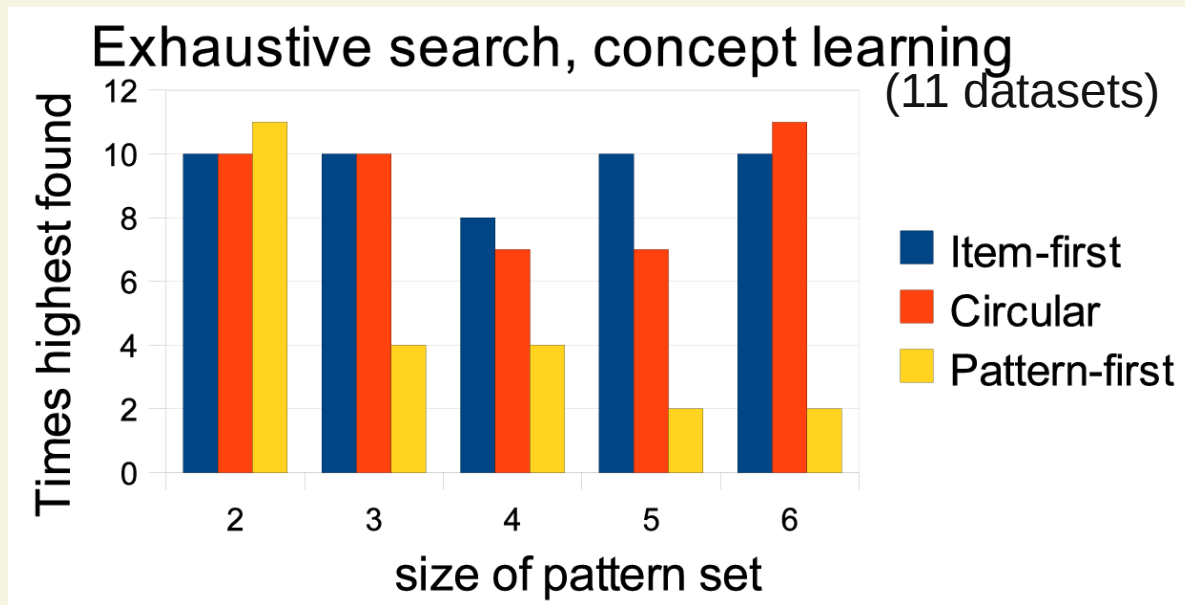
```
maximize<cp> { ... }  
subject to { ... }  
using {  
    label(Items);  
}  
onRestart {  
    Solution s = cp.getSolution();  
    UniformDistribution dist(1..K);  
    int not_k = dist.get();  
    forall (k in 1..K: k!=not_k, i in 1..NrI)  
        // ignore all but the random pattern in search  
        cp.post(Items[k,i] == Items[k,i].getSnapshot(s));  
}
```



Some experiments  
'because we can'

# Experiments

Concept learning, different search specifications



Same model, only 'using{ ... }' part changed!

# Experiments

## Dispersion set, different search strategies

	Gr VS All	Gr-it VS Gr	LNS-r VS LNS-p	LNS-r VS Gr
prim.-tumor	3/0/1	3/1/0	4/0/0	4/0/0
hepatitis	4/0/0	3/1/0	4/0/0	2/1/1
tic-tac-toe	2/1/0	1/3/0	0/4/0	1/3/0
audiology	3/0/0	2/1/0	4/0/0	1/0/3
germ.-credit	3/0/0	2/1/0	4/0/0	3/0/1
austr.-credit	3/0/0	1/1/1	4/0/0	1/0/3
soybean	3/0/0	2/0/1	3/0/1	4/0/0
lymph	3/0/0	2/0/1	4/0/0	2/0/2
vote	3/0/1	4/0/0	4/0/0	4/0/0
heart-clev.	4/0/0	0/1/0	4/0/0	3/0/1
total	31/1/3	20/9/3	35/4/1	25/4/11

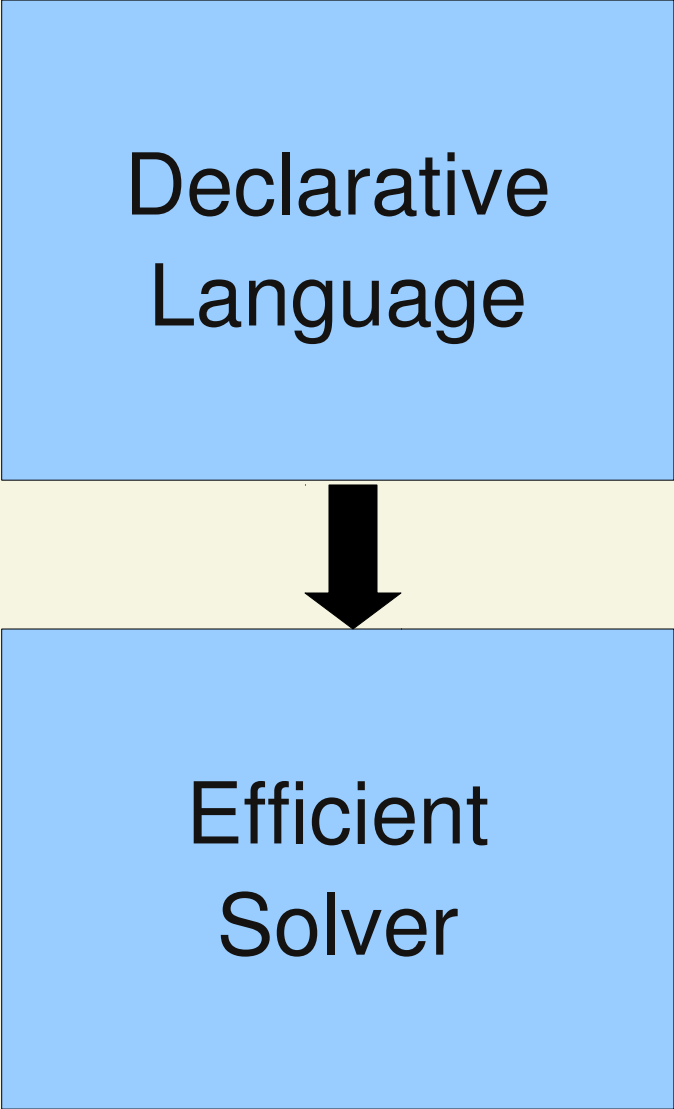
Table IV: win/draw/loss count of different search strategies for the dispersion score. For each dataset, pattern sets of size 3, 6, 9 and 12 were searched for. For LNS, average results over 5 runs were used.

Same model, only 'onRestart{ ... }' changed ( $\pm$ )

Concluding remarks &  
open questions

# Our Vision for pattern mining

Declarative  
Language



Efficient  
Solver

- Pattern and pattern *set* tasks
- Exhaustive and *heuristic* search
- **Declarative** specification of both task *and* search

# Declarative heuristic search

## The 'Comet' system

```
Solver<CP> cp();  
cp.lnsOnFailure(1000);
```

```
maximize<cp>
```

```
...
```

```
subject to {
```

```
...
```

```
}
```

} model

```
using { ... }
```

```
onRestart {
```

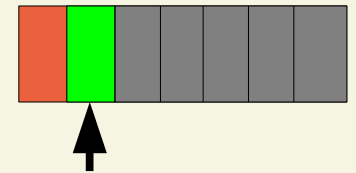
```
...
```

```
}
```

} search

# Comet experience

- + elegant declarative language
- + can study tasks and search separately
- + heuristic search without changing model ( $\pm$ )
- X** Low-level considerations (pass-by-value/reference)
- X** For greedy, 'tricks' needed to initialise patterns
- X** Limited scalability to large datasets



```
Solver<CP> cp();  
cp.InsOnFailure(1000);
```

```
maximize<cp>  
subject to {  
  ...  
}  
using { ... }  
onRestart {  
  ...  
}
```

} model

} search

# Open Discussion

- Declarative local search (no CP)?
- Do all tasks allow reuse of search specifications?
- Some tasks hard to model in constraints?
- What is needed to make it more scalable?
- General declarative tools for data mining?

Declarative  
Language



Efficient  
Solver