Improving the Solving Efficiency of TOY(FD) and its Application to Real-Life Problems

PhD Thesis:
Ignacio Castiñeiras Pérez

Advisors:
Fernando Sáenz Pérez
Francisco Javier López Fraguas

Declarative Programming Group
Complutense University of Madrid

Madrid, 5/12/2014
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Background

- Logistics: Key issue for the success of any company or organization.
  - Suitable allocation or scheduling of the resources of the company.
- Examples in nearly any manufacturing and service industry.
  - Procurement.
  - Production.
  - Transportation.
  - Distribution.
  - Information processing.
  - Communication.
CSP’s and COP’s

- Most of these logistics examples are NP-complete problems.
  - Time and expertise required for: Specification and Solving method.

- **Constraint Satisfaction and Optimization Problems:**
  (CSP’s and COP’s, resp.) Suitable formulation.

- **CSP** $\equiv (V, D, C)$
  - Variables: $V \equiv \{v_1, v_2, \ldots, v_n\}$
  - Domains: $D \equiv \{d_1, d_2, \ldots, d_n\}$
  - Constraints: $C \equiv \{c_1, c_2, \ldots, c_m\}$
  - Candidates: Combinations of assigning each $v_i$ to a value of $d_i$.
  - Solutions: Candidates entailing $C$.

- **COP** $\equiv (V, D, C, F)$
  - Optimization Function: $F \equiv \text{minimize}/\text{maximize expr}$.
  - Solutions: Candidates entailing $C$ and optimizing the expr of $F$. 
CSP’s and COP’s extensively studied. Different approaches:

1. Mathematical Programming.
2. Heuristics.
3. **Constraint Programming over Finite Domains: CP(FD)**.
   - Specially successful.
   - Captures constraint oriented nature of problems.
   - Clearly separates *Problem Specification* & *Problem Solving*.
CP(\textit{FD}) Solving Techniques

- CP(\textit{FD}) \implies Constraint solver: Propagation + Search.
- Constraint propagation of \( c \) involving \( \{ v_1 \ldots v_n \} \).
  - Inference process removing inconsistent domain values of \( \{ d_1 \ldots d_n \} \).
- CSP \equiv (V, D, [c_1, c_2 \ldots c_n]). Propagation on each \( c_i \) separately.
  - Not all remaining values of \( \{ d_1 \ldots d_n \} \) are part of some solution.
- Solving process is completed with a search exploration.
  - Dynamic modification CSP \equiv (V, D, [c_1 \ldots c_n, c_{n+1}, c_{n+2} \ldots c_{n+k}]).
  - Can be seen as traversing a search tree, where each node is a modified version of the original CSP.
CP(\(\mathcal{FD}\)) Modeling Languages

1. Algebraic CP(\(\mathcal{FD}\)).
   - Specific-purpose. Declarative. Based on algebraic formulations.
   - Some state-of-the-art systems: MiniZinc, ILOG OPL.

2. Object-oriented CP(\(\mathcal{FD}\)).
   - General-purpose. Imperative. Object-Oriented languages.
   - Some state-of-the-art systems: Gecode, ILOG Solver (C++).

3. CLP(\(\mathcal{FD}\)).
   - General-purpose. Declarative. Logic Programming languages.
   - Some state-of-the-art systems: SICStus Prolog, SWI-Prolog.

4. **Constraint Functional Logic Programming**: CFLP(\(\mathcal{FD}\)).
   - Some state-of-the-art systems: TOY(\(\mathcal{FD}\)), PAKCS.
TOY(FD) Main Description

- Implemented in SICStus Prolog.
  - Instead of using an abstract machine, a TOY(FD) program is compiled to SICStus Prolog and executed by the SICStus engine.
- Herbrand constraint solver.
  - Implemented on plain Prolog.
  - Syntactic == and /= constraints.
- Finite Domain constraint solver.
  - Relies on a CP(FD) library interfaced to the system.
    - Current TOY(FD) version interfaces SICStus clpfd library.

<table>
<thead>
<tr>
<th>Type</th>
<th>Constraints &amp; Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Constraints</td>
<td>==, /=, #&gt;, #&gt;=, #&lt;, #&lt;=</td>
</tr>
<tr>
<td>Arithmetic Operators</td>
<td>#+, #-, #*, #/</td>
</tr>
<tr>
<td>Propositional Constraint</td>
<td>post_implication</td>
</tr>
<tr>
<td>Domain Constraints</td>
<td>domain, domain_valArray</td>
</tr>
<tr>
<td>Global Constraints</td>
<td>all_different, count, sum, scalar_product</td>
</tr>
</tbody>
</table>
**TOY(FD) Language Modeling Features**

**TOY(FD) Program:**
- Haskell-like syntax (variables upper-case & functions lower-case).
- Datatype declarations, operators & functions.

<table>
<thead>
<tr>
<th>Functional Features</th>
<th>Logic Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional notation</td>
<td>Relational notation</td>
</tr>
<tr>
<td>Curried expressions</td>
<td>Non-determinism</td>
</tr>
<tr>
<td>Higher-order functions</td>
<td>Backtracking</td>
</tr>
<tr>
<td>Lazy evaluation</td>
<td>Logical variables</td>
</tr>
<tr>
<td>Patterns</td>
<td>Domain variables</td>
</tr>
<tr>
<td>Partial application</td>
<td>Reasoning with models</td>
</tr>
<tr>
<td>Types</td>
<td></td>
</tr>
<tr>
<td>Polymorphism</td>
<td></td>
</tr>
<tr>
<td>Constraint composition</td>
<td></td>
</tr>
</tbody>
</table>
**TOY(FD) Model Example for N-Queens**

```plaintext
include "misc.toy"
include "cflpfd.toy"

queens:: int -> [int]
queens N = L <==
    take N gen_v_list == L,
    domain L 1 N,
    all_different L,
    all_different (zipWith (#+) L (take N (from 0))),
    all_different (zipWith (#-) L (take N (from 0))),
    (head L) #< (last L),
    labeling [ff] L

from:: int -> [int]
from N = N : from (N+1)

gen_v_list:: [A]
gen_v_list = [X | gen_v_list]
```
**TOY(FD)** Goal for 6-Queens

**TOY(FD)**> queens 6 == L
  
  \{ L -> [ 2, 4, 6, 1, 3, 5 ] \}

sol.1, more solutions (y/n/d/a) [y]?
  
  \{ L -> [ 3, 6, 2, 5, 1, 4 ] \}

sol.2, more solutions (y/n/d/a) [y]?
  
  no
Nowadays big and alive CP($\mathcal{FD}$) community.

- Building up a large number of systems and applications.
- Many papers in top conferences and journals.
  - Algebraic CP($\mathcal{FD}$), C++ CP($\mathcal{FD}$), CLP($\mathcal{FD}$) $\Rightarrow$ Lots of applications.
  - CFLP($\mathcal{FD}$) $\Rightarrow$ Lack of applications.
Perform an empirical evaluation of the applicability of CFLP($\mathcal{FD}$) to tackle CSP’s and COP’s

Select $\textit{TOY(FD)}$. Divide research in 3 parts:

1. Improving the Performance of $\textit{TOY(FD)}$.
2. Real-Life Applications of $\textit{TOY(FD)}$.
3. Positioning $\textit{TOY(FD)}$ w.r.t. other CP($\mathcal{FD}$) Systems.
Outline

1. Motivation
2. Improving the Performance of $\text{TOY}(FD)$
   2.1 Contributions
   2.2 Interface C++ CP($FD$) Solvers
   2.3 New Search Primitives
   2.4 Conclusions
3. Real-Life Applications of $\text{TOY}(FD)$
4. Positioning $\text{TOY}(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
   2.1 Contributions
   2.2 Interface C++ CP($FD$) Solvers
   2.3 New Search Primitives
   2.4 Conclusions
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Possible Improvements

- Amdahl’s law: Improve system performance by altering a component ⇒ Fraction of time component is used.
- Hypothesis: Combinatorial nature of CSP’s & COP’s.
  - As instances ↑ ⇒ Search exploration is the key performance factor.
- Conclusion: 2 ways to improve \( \text{TOY}(FD) \) performance.
  - Replace solver performing the search (i.e., same search exploration, more efficient solver).
  - Replace search being performed (i.e., same solver, more efficient search).
C++ CP($FD$) solvers

Develop an interface scheme for C++ CP($FD$) solvers into $TOY(FD)$
(Applicable to $TOY(FD)$, CLP($FD$) and CFLP($FD$) Prolog systems).

- Identify $TOY(FD)$ commands to coordinate the solver.
- Establish communication: Prolog and C++ components.
- Manage different variable, constraint and types of system & solver.
- Adapt the C++ CP($FD$) solver to the CFLP($FD$) requirements:
  - Model reasoning.
  - Multiple search strategies.
  - Incremental and Batch propagation modes.
- Targets: Gecode & ILOG Solver $\Rightarrow TOY(FDg)$ & $TOY(FDi)$. 
Develop an interface scheme for C++ CP(\(FD\)) solvers into \(TOY(FD)\) (Applicable to \(TOY(FD)\), CLP(\(FD\)) and CFLP(\(FD\)) Prolog systems).

- Establish communication: Prolog and C++ components.
- Manage different variable, constraint and types of system & solver.
- Adapt the C++ CP(\(FD\)) solver to the CFLP(\(FD\)) requirements:
  - Model reasoning.
  - Multiple search strategies.
  - Incremental and Batch propagation modes.
Search Strategies

Enhance $TOY(FD)$ language with search primitives.

(Applicable to CLP($FD$) and CFLP($FD$) Prolog systems interfacing C++ CP($FD$) solvers).

- Provide a more detailed search specification to the solver.
- Develop 8 new search primitives. Novel concepts:
  - Exhaustive breath exploration + sort criteria.
  - Fragment variable domain.
  - Apply strategy to a variable subset.
  - Specify variable, value and bound criteria at $TOY(FD)$ level.
- $TOY(FD)$ primitives combination + multiple search scenarios.
- Targets: Extend Gecode & ILOG Solver libraries $\Rightarrow$ $TOY(FDg)$ & $TOY(FDi)$. 
Search Strategies

**Enhance** \( TOY(FD) \) **language with search primitives.**

(Applicable to CLP(\(FD\)) and CFLP(\(FD\)) Prolog systems interfacing C++ CP(\(FD\)) solvers).

- Develop 8 new search primitives. Novel concepts:
  - Exhaustive breath exploration + sort criteria.
  - Fragment variable domain.
  - Apply strategy to a variable subset.
  - Specify variable, value and bound criteria at \( TOY(FD) \) level.)
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
   2.1 Contributions
   2.2 Interfacing C++ CP($FD$) Solvers
   2.3 New Search Primitives
   2.4 Conclusions
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Difficulties to Interface a C++ CP(\(\mathcal{FD}\)) Solver

- Communicating system and solver.
- Variable & constraint representations.
- Model reasoning.
- Search strategies.
- Incremental and batch propagation.
Communicating system and solver

(Before) \( \text{TOY}(FDs) \):

- Load the SICStus \( \text{clpfd} \) library.

(Now) \( \text{TOY}(FDg) \) & \( \text{TOY}(FDi) \):

- Use SICStus \textit{Prolog-C++ connection framework}.
- Use additional Prolog and C++ \textit{global} data structures.
Variable & Constraint Representations

(Before) $TOY(FDs)$:
- System & $FD$ solver = variable representation.
- $clpfd$ API exposes the store constraint network.

(Now) $TOY(FDg)$ & $TOY(FDi)$:
- System & $FD$ solver ≠ (variable, constraint) representation.
  - Additional mate $(pV, cV)$ & $(pC, cC)$ data structures.
  - Daemon constraints $d \Rightarrow$ Synchronize $cV$ binding with $pV$ unification.
- Solver API does not exposes the store constraint network.
  - Explicit traversing $pC \Rightarrow$ Display non-ground constraint on solutions.
Model Reasoning

**(Before)** $TOY(FDs)$:
- Automatic restoration of the store on backtracking.

**(Now)** $TOY(FDg)$ & $TOY(FDi)$:
- Explicit restoration of the store on backtracking.
  - Use automatic restoration of $pV$ & $pC$ to restore mate $cV$ & $cC$.
  - Solver API precludes single constraint removal $\Rightarrow$ clean & repost.
Multiple Searches & Constraint Posting

(Before) $\text{TOY}(FDs)$:
- clpfd supports it.

(Now) $\text{TOY}(FDg)$ & $\text{TOY}(FDi)$:
- Solver does not support it.
  - Isolate constraint posting to store from search strategies.
  - Each labeling $l_i$ managed by external unit.
  - Synchronize the store and external unit pre/post search.
Solution represented as a set of equality constraints.
Incremental & Batch Propagation

- Develop the new primitives `batch_on` & `batch_off`.
- Free use of them in `TOY(FD)` programs.

(Before) `TOY(FDs)`:
- Use delayed goals (`freeze` predicates).
- Additional flag modified by primitives.

(Now) `TOY(FDg)` & `TOY(FDi)`:
- Additional tuple `(B, cVs, cCs)`.
  - Variables and constraints are posted to `cV` & `cC`, but not to the store.
  - Explicit management for backtracking.
Experiments

Classical CP($\mathcal{FD}$) Benchmarks:

- Magic Series, N-Queens, Langford’s (CSP’s) & Golomb (COP).
- Instances solved in milliseconds, seconds & minutes.

<table>
<thead>
<tr>
<th>Instance</th>
<th>%</th>
<th>Sp-Up</th>
<th>Instance</th>
<th>%</th>
<th>Sp-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-105 FDs</td>
<td>97.3</td>
<td>1.00</td>
<td>Q-120 FDs</td>
<td>99.9</td>
<td>1.00</td>
</tr>
<tr>
<td>Q-105 FDg</td>
<td>96.6</td>
<td>3.33</td>
<td>Q-120 FDg</td>
<td>99.9</td>
<td>3.45</td>
</tr>
<tr>
<td>Q-105 FDi</td>
<td>92.6</td>
<td>2.78</td>
<td>Q-120 FDi</td>
<td>99.9</td>
<td>2.94</td>
</tr>
<tr>
<td>L-127 FDs</td>
<td>97.7</td>
<td>1.00</td>
<td>L-131 FDs</td>
<td>99.9</td>
<td>1.00</td>
</tr>
<tr>
<td>L-127 FDg</td>
<td>96.3</td>
<td>2.63</td>
<td>L-131 FDg</td>
<td>99.8</td>
<td>2.56</td>
</tr>
<tr>
<td>L-127 FDi</td>
<td>88.9</td>
<td>2.78</td>
<td>L-131 FDi</td>
<td>99.4</td>
<td>2.86</td>
</tr>
<tr>
<td>G-10 FDs</td>
<td>99.6</td>
<td>1.00</td>
<td>G-11 FDs</td>
<td>99.9</td>
<td>1.00</td>
</tr>
<tr>
<td>G-10 FDg</td>
<td>99.1</td>
<td>3.45</td>
<td>G-11 FDg</td>
<td>99.9</td>
<td>3.57</td>
</tr>
<tr>
<td>G-10 FDi</td>
<td>99.2</td>
<td>2.08</td>
<td>G-11 FDi</td>
<td>99.9</td>
<td>2.13</td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Improving the Performance of \textit{TOY} (FD)
   2.1 Contributions
   2.2 Interface C++ CP(FD) Solvers
   2.3 New Search Primitives
   2.4 Conclusions
3. Real-Life Applications of \textit{TOY} (FD)
4. Positioning \textit{TOY} (FD) w.r.t. other CP(FD) Systems
5. General Conclusions
Improving the Performance of $\text{TOY}(FD)$

New Search Primitives

Enhancing the $\text{TOY}(FD)$ Language

$\text{TOY}(FDg)$ & $\text{TOY}(FDi)$: 8 new primitives

- Increase user interaction with $FD$ search exploration.
- Exploit CFLP($FD$) expressivity & C++ CP($FD$) efficiency.

<table>
<thead>
<tr>
<th>lab</th>
<th>frag</th>
</tr>
</thead>
<tbody>
<tr>
<td>labB</td>
<td>fragB</td>
</tr>
<tr>
<td>labW</td>
<td>fragW</td>
</tr>
<tr>
<td>labO</td>
<td>fragO</td>
</tr>
</tbody>
</table>
Search Strategies lab & labB

lab:: varOrd → valOrd → int → [int] → bool
  ▶ Classical search. Collects solutions, one by one.

labB:: varOrd → valOrd → int → [int] → bool
  ▶ Equivalent to lab, but explores just one tree branch.

Variable & Value order criteria:
  ▶ Full Gecode catalog.
  ▶ Specify criteria at TOY(FD) level.
  ▶ Label only a subset of the variables.
Search Strategies lab & labB

myVarOrder:: [int] -> int
myrVarOrder Vs =
          fst (foldl1 cmp (0,0) (zip (take (length Vs) (from 0))
                          (map (length . get_dom) Vs)))

TOY(FD)> domain [X,Y,Z] 0 4, Y /= 1, Y /= 3, Z /= 2,
        lab userVar smallestVal 2 [X,Y,Z]
sol 1. {X in 0..4, Y -> 0, Z -> 0}
sol 2. {X in 0..4, Y -> 0, Z -> 1}
...

TOY(FD)> domain [X,Y,Z] 0 4, Y /= 1, Y /= 3, Z /= 2,
        labB userVar smallestVal 2 [X,Y,Z]
sol 1. {X in 0..4, Y -> 0, Z -> 0}
      no
Search Strategy \text{labW}

\text{labW}:: \text{varOrd} \rightarrow \text{bound} \rightarrow \text{int} \rightarrow \text{[int]} \rightarrow \text{bool}

- Exhaustive Breadth exploration + criteria to sort solutions.

Sort criteria:
- Smallest/largest local/global search space.
- Specify criteria at \text{TOPY(FD)} level.

\text{TOPY(FD)}> \text{domain [X,Y] 0 1}, \text{domain [V1,V2] 0 3}, \newline (X \#= 1) \Rightarrow (V1 \#> 1), (Y \#= 0) \Rightarrow (V2 \#> 0), \newline \text{labW \ unassignedLeftVar \ smallestTree \ 2 [X,Y,V1,V2]}, \newline \dots \text{REST OF TOPY GOAL}
Search Strategies frag, fragB, fragW

frag:: domFrag → varOrd → intervalOrd → int → [int] → bool
fragB:: domFrag → varOrd → intervalOrd → int → [int] → bool
fragW:: domFrag → varOrd → bound → int → [int] → bool

▶ Variable domains fragmented instead of labeled.

Example: A goal contains \( V \) variables and \( C \) constraints.

▶ \( V' \equiv \{V_1, V_2, V_3\} \) a subset of \( V \) (9 values each).

▶ Assume single solution.

Tradeoff: Right path probability vs. less propagation.
Experiments

- Magic Series, N-Queens, Langford’s (CSP’s) & Golomb (COP).
- Instances solved in milliseconds, seconds & minutes.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Sp-Up-Bs</th>
<th>Sp-Up-Is</th>
<th>Bs/Is</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-105 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>1.61</td>
</tr>
<tr>
<td>Q-105 FDg</td>
<td>1.19</td>
<td>10.00</td>
<td>12.50</td>
</tr>
<tr>
<td>Q-120 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>138.74</td>
</tr>
<tr>
<td>Q-120 FDg</td>
<td>1.19</td>
<td>12.50</td>
<td>1443.11</td>
</tr>
<tr>
<td>L-127 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>3.70</td>
</tr>
<tr>
<td>L-127 FDg</td>
<td>0.94</td>
<td>3.03</td>
<td>12.50</td>
</tr>
<tr>
<td>L-131 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>73.11</td>
</tr>
<tr>
<td>L-131 FDg</td>
<td>0.88</td>
<td>3.57</td>
<td>298.58</td>
</tr>
<tr>
<td>G-10 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>2.44</td>
</tr>
<tr>
<td>G-10 FDg</td>
<td>1.69</td>
<td>1.75</td>
<td>2.50</td>
</tr>
<tr>
<td>G-11 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>1.64</td>
</tr>
<tr>
<td>G-11 FDg</td>
<td>1.72</td>
<td>1.75</td>
<td>1.69</td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
   2.1 Contributions
   2.2 Interface C++ CP($FD$) Solvers
   2.3 New Search Primitives
   2.4 Conclusions
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Interfacing C++ CP(\textit{FD}) solvers:

- \textit{TOY(FDg)} & \textit{TOY(FDi)} $\Rightarrow$ C++ solver coordination overhead.
- Benchmark: Instances $\uparrow$ $\Rightarrow$ Search Time $\uparrow$ & overhead $\downarrow$.
- \textit{TOY(FDg)} & \textit{TOY(FDi)} outperform \textit{TOY(FDs)} $1.15-3.57 \times$.
- Correlation: Search Time $\uparrow$ $\Rightarrow$ outperform $\uparrow$.
- Propagation mode: Not relevant.
Conclusions

Enhancing $\text{TOY}(\text{FD})$ with new search primitives:

- Benchmark: Problem solution structures $\Rightarrow$ Ad hoc search strategy.
- New search strategies: $\text{TOY}(\text{FD}_g)$ & $\text{TOY}(\text{FD}_i)$ outperform themselves up to $1,500 \times$.
- $\text{TOY}(\text{FD}_g)$ outperform $\text{TOY}(\text{FD}_i) \Rightarrow$ More efficient library extension.
- $\text{TOY}(\text{FD})$ criteria: Overhead due to interaction of different $\text{TOY}(\text{FD})$ architecture layers.
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
   3.1 Contributions
   3.2 Employee Timetabling Problem
   3.3 Bin Packing Problem
   3.4 Conclusions
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Outline

1. Motivation
2. Improving the Performance of \textit{TOY(FD)}
3. Real-Life Applications of \textit{TOY(FD)}
   3.1 Contributions
   3.2 Employee Timetabling Problem
   3.3 Bin Packing Problem
   3.4 Conclusions
4. Positioning \textit{TOY(FD)} w.r.t. other CP(FD) Systems
5. General Conclusions
Contributions

Applying $\mathcal{TOY}(FD)$ to Employee Timetabling Problem (ETP).
(Coming from the media industry).

- Present a fully parameterizable version of the problem.
- Present a solving approach, based on problem decomposition:
  - $\mathcal{TOY}(FD) \Rightarrow$ suitable for modeling the problem with this solving approach.
- Describe an algorithm to implement the solving approach.
Contributions

Applying \( TOY(FD) \) to Employee Timetabling Problem (ETP).
(Coming from the media industry).

- Present a fully parameterizable version of the problem.
- Present a solving approach, based on problem decomposition:
  - \( TOY(FD) \) ⇒ suitable for modeling the problem with this solving approach.
Contributions

Applying $\text{T0Y(FD)} + \text{Heuristics}$ to Bin Packing Problem (BPP).
(Benchmark of real-life instances from the data centre industry).

ETP $\Rightarrow$ Focus on the problem.
BPP $\Rightarrow$ Focus on the methodology.

- Describe a methodology to perform an empirical analysis of the hardness of the problem.
- Use a statistical model to generate a parameterizable benchmark.
- Apply CP($\mathcal{FD}$) & Heuristics methods to:
  - Model the problem.
  - Automatize the solving of the benchmark.
- Discuss $\text{T0Y(FD)}$ as an appealing CP($\mathcal{FD}$) system to be used in the analysis.
Contributions

Applying $\text{T}O\text{Y}(FD)$ + Heuristics to Bin Packing Problem (BPP).
(Benchmark of real-life instances from the data centre industry).

- ETP $\Rightarrow$ Focus on the problem.
- BPP $\Rightarrow$ Focus on the methodology.

- Describe a methodology to perform an empirical analysis of the hardness of the problem.
- Use a statistical model to generate a parameterizable benchmark.

- Discuss $\text{T}O\text{Y}(FD)$ as an appealing CP$(FD)$ system to be used in the analysis.
Outline

1. Motivation
2. Improving the Performance of TOY(FD)
3. Real-Life Applications of TOY(FD)
   3.1 Contributions
   3.2 Employee Timetabling Problem
   3.3 Bin Packing Problem
   3.4 Conclusions
4. Positioning TOY(FD) w.r.t. other CP(FD) Systems
5. General Conclusions
Employee Timetabling Problem Description

- A department employs $w$ (13) workers, divided into $nt$ teams (3) of $ntw$ workers (4), plus 1 extra worker $ew$.
- The department has to schedule the $w$ workers for the next $nd$ days (21), where the workload of the days may be different $\text{working\_days} = (20, 22, 24)$, $\text{weekend\_days} = (24, 24)$.
- The goal is to schedule the workers for the days, minimizing the extra hours to be paid.

- Interesting: The teams rotate (each $t_i$ works each $nt$ days).
- Due to the absences, the $t_i$ selected for $d_j$:
  - Has enough available workers.
  - Needs to schedule $ew$.
  - Cannot manage the amount of shifts required.
- $ew$ can just be selected to work 1 out of each $er$ days (3).
ETP Solving Approach

- **Idea:** Use *timetabling*, an $\mathcal{FD}$ variables $w \times nd$ matrix.
- **Better idea:** Exploit the dependencies of the problem, to solve an simple version of it.
  - *Table*, an $\mathcal{FD}$ variables $(ntw + 1) \times nd$ matrix.
ETP Solving Approach

- **Idea:** Use *timetabling*, an \( \mathcal{FD} \) variables \( w \times nd \) matrix.

- **Better idea:** Exploit the dependencies of the problem, to solve an simple version of it.
  - *Table*, an \( \mathcal{FD} \) variables \( (ntw + 1) \times nd \) matrix.
ETP Solving Approach

- Idea: Use *timetabling*, an $\mathcal{FD}$ variables $w \times nd$ matrix.
- Better idea: Exploit the dependencies of the problem, to solve an simple version of it.
  - *Table*, an $\mathcal{FD}$ variables $(ntw + 1) \times nd$ matrix.
# Table Needed to Map timetabling?

- Each feasible assignment of teams to days $\Rightarrow$ 1 Table.
  - Enough workers of each team per day of the timetable.
  - $ew$ resting constraint is satisfied.
Each feasible assignment of teams to days ⇒ 1 Table.

- Enough workers of each team per day of the timetable.
- Every resting constraint is satisfied.
# Table Needed to Map *timetabling*?

- Each feasible assignment of teams to days $\Rightarrow 1$ *Table*.
  - Enough workers of each team per day of the timetable.
  - $ew$ resting constraint is satisfied.

<table>
<thead>
<tr>
<th>$wi$</th>
<th>$d_j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$rw_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$rw_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$rw_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$rw_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table Needed to Map *timetabling*?

- Each feasible assignment of teams to days ⇒ 1 *Table*.
  - Enough workers of each team per day of the timetable.
  - *ew* resting constraint is satisfied.

![Diagram showing a table with days and teams]
Architecture of the Solution

4 Stages Process:

1. **team_assign**: Finds each feasible *teams to days assignment*: $tda$.
   - CSP: Collects the solutions one by one.

2. **tt_split**: Creates *Table* and split it by teams.
   - The problem associated to each team is independent from the one of the other teams.

3. **tt_solve**: Sequentially solves each *subTable Table$_{ti}$*.
   - COP: Finds suboptimal scheduling for $t_i$.

4. **tt_map**: Maps suboptimal *Table* to *timetabling* format.
   - Process backtracks again to 1, to find a new $tda$.

Once no more feasible $tda$ are found, computed suboptimal *timetabling* are compared, and the one minimizing the extra hours is outputted as a result.
Why Is \textit{TOY(FD)} Useful?

Suitable for the architecture of the solution:

- Easy coordination of the stages:
  - Main function calling stages in order.
- Reasoning with models:
  - Explicit \texttt{labeling} in stage 1 computes feasible \textit{tda} one by one.
  - It implicitly triggers backtracking after stage 4.
  - Using a \texttt{collect} to find all solutions.

Suitable for specifying the problem:

- \textit{TOY(FD)} Expressiveness is fully exploited: Functional notation, curried expressions, higher-order functions, lazy evaluation, patterns, partial application, types, polymorphism, constraint composition, backtracking, logical and decision variables.
Experiments

Instances solved in milliseconds, seconds & minutes.

<table>
<thead>
<tr>
<th>Instance</th>
<th>%</th>
<th>Sp-Up</th>
<th>Instance</th>
<th>%</th>
<th>Sp-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETP-15 FDs</td>
<td>91.9</td>
<td>1.00</td>
<td>ETP-21 FDs</td>
<td>99.9</td>
<td>1.00</td>
</tr>
<tr>
<td>ETP-15 FDg</td>
<td>60.0</td>
<td>3.57</td>
<td>ETP-21 FDg</td>
<td>99.1</td>
<td>6.67</td>
</tr>
<tr>
<td>ETP-15 FDi</td>
<td>51.2</td>
<td>0.98</td>
<td>ETP-21 FDi</td>
<td>98.6</td>
<td>3.13</td>
</tr>
</tbody>
</table>

Ad hoc strategy.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Sp-Up-Bs</th>
<th>Sp-Up-Is</th>
<th>Bs/Is</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETP-15 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>1.25</td>
</tr>
<tr>
<td>ETP-15 FDg</td>
<td>3.70</td>
<td>5.56</td>
<td>1.89</td>
</tr>
<tr>
<td>ETP-21 FDi</td>
<td>1.00</td>
<td>1.00</td>
<td>4.76</td>
</tr>
<tr>
<td>ETP-21 FDg</td>
<td>2.17</td>
<td>2.38</td>
<td>5.26</td>
</tr>
</tbody>
</table>
Real-Life Applications of $TOY(FD)$

Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
   3.1 Contributions
   3.2 Employee Timetabling Problem
   3.3 Bin Packing Problem
   3.4 Conclusions
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
BPP Empirical Analysis: Background

- One-dimensional Bin Packing (BPP):
  - Assign items to bins without breaking the capacity constraints.
  - Minimize the number of bins used.

- Lots of previous analysis on BPP:
  - Many method proposals.
  - Each method incorporating its own benchmark.

- Problem:
  - No standardized benchmarks.
  - Most of the available benchmarks are often unrealistic and/or trivial to solve.
**BPP Empirical Analysis: Our Proposal**

**Goal:** Perform another empirical analysis of the BPP hardness.

- Use a parameterizable generated **new** benchmark.
  - Representing real-life BPP instances.
  - Allowing to control the instances being generated (to perform very controlled experiments).

Parameterizable benchmark: Weibull statistical model.

- Continuous probability distribution.
- Allows to represent nearly any unimodal distribution.
- Very flexible (parameterizable on its shape $k$ and scale $\lambda$).

Parameterization:

$$f(x, \lambda, k) = \begin{cases} \frac{k}{\lambda} \cdot \left(\frac{x}{\lambda}\right)^{k-1} \cdot e^{-\left(\frac{x}{\lambda}\right)^k} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$
Precise Control of Generated Instances

- A benchmark generator: Weibull $(k, \lambda)$.
  - Scale $(\lambda) = 1000$.
  - Shape $(k) \in \{0.1, 0.2, \ldots, 19.9\}$.
  - 100 instances of 100 items per $(k, \lambda)$ combination.
Well-Fitting Existing Real-Life BPP Instances

- By observation:
  - R: Maximum Likelihood Fitting & Quantile-Quantile Plots.
- By statistical tests:
  - Kolmogorov-Smirnov & $\chi^2$
Benchmark Solving: CP(\(FD\)) & Heuristics

- CP(\(FD\)) model.
  - Based on the most efficient model of the Gecode distribution.
  - Lower & Upper bound.
  - Bin packing global constraint.
  - Complete Decreasing Best Fit (CDBF) Search.
  - Timeout of 10secs.
- Analyze solving time, % of instances solved, # bins.

- Heuristics:
  - MAXREST: Place in bin with more remaining space.
  - FIRSTFIT: Place in first bin with enough remaining space.
  - BESTFIT: Place in bin with enough but less remaining space.
  - NEXTFIT: Place in last bin.
- Analyze # bins.

- Solving sessions:
  - Bin Capacity \(C \in \{1.0, 1.1, \ldots, 2.0\}\).
  - Scripting techniques to automatize the solving of the benchmark.
Why Use $TOY(FD)$ as a CP($FD$) System?

Appealing alternative to Gecode

► Modeling:
  ▶ ETP expressiveness also holds for BPP.
  ▶ CDBF Search implemented as another new search primitive.
  ▶ Bin Packing implemented as a new primitive constraint.
  ▶ Achieves the same granularity as Gecode.

► Solving:
  ▶ $TOY(FDg)$ better performance also holds for BPP.
  ▶ Same constraint network & search exploration as in Gecode.
  ▶ Achieves the same performance as Gecode.

► Scripting techniques:
  ▶ Non-deterministic functions allows to automatize the solving of the benchmark.
Experiments

- Graphs showing the relationship between percentage or number of bins and shape factor for different values of parameter c.

- Curves for different values of c, indicating how the distribution of bins changes with shape factor.
Outline

1. Motivation
2. Improving the Performance of TOY(FD)
3. Real-Life Applications of TOY(FD)
   3.1 Contributions
   3.2 Employee Timetabling Problem
   3.3 Bin Packing Problem
   3.4 Conclusions
4. Positioning TOY(FD) w.r.t. other CP(FD) Systems
5. General Conclusions
Conclusions

ETP:

- Complex formulation exploits $\text{T}_\text{O}Y(\text{FD})$ high expressivity.
- Solving results are more instance dependent.
  - $\text{T}_\text{O}Y(\text{FD}_g)$ outperforms $\text{T}_\text{O}Y(\text{FD}_s)$ more.
  - $\text{T}_\text{O}Y(\text{FD}_i)$ worse, similar and better than $\text{T}_\text{O}Y(\text{FD}_s)$, resp.
- Small instances: $\text{T}_\text{O}Y(\text{FD}_g)$ & $\text{T}_\text{O}Y(\text{FD}_i)$ \uparrow overhead C++ solver.
- Propagation mode: More relevant in some cases.
- Ad hoc search strategy: Less relevant.
Conclusions

BPP:

- Both CP($\mathcal{FD}$) and Heuristics suitable for solving it.
- CP($\mathcal{FD}$): Any $C \Rightarrow$ hard interval $(k, \lambda)$.
  - Hardness classification by %:
    - 80%-100%, 60%-80%, 40%-60%, 20%-40%, 0%-20%.
  - As $C \uparrow \Rightarrow$ Hardness $\uparrow$.
- As $k \uparrow \Rightarrow$ Number of bins $\uparrow$.
  - No general patterns for relation:
    - Hard interval $(k, \lambda) + C +$ Number of bins.
- Heuristic MAXREST: Good alternative to CP($\mathcal{FD}$).
  - Quality of solution deviation (0.0-1.4) bins.
  - As CP($\mathcal{FD}$) hardness $\uparrow \Rightarrow$ MAXREST gap $\uparrow$. 
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
   4.1 Contributions
   4.2 Modeling Comparison
   4.3 Solving Comparison
   4.4 Conclusions
5. General Conclusions
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
   4.1 Contributions
   4.2 Modeling Comparison
   4.3 Solving Comparison
   4.4 Conclusions
5. General Conclusions
Contributions

Perform an in-depth modeling comparison of CP(FD) systems.

- Two COP’s:
  - Classical CP(FD) puzzle benchmark: Golomb Rulers.
  - Real-life application: ETP.
- Eight state-of-the-art systems:
  - Algebraic CP(FD): MiniZinc & ILOG OPL.
  - C++ CP(FD): Gecode & ILOG Solver.
  - CLP(FD): SICStus Prolog & SWI-Prolog.
  - CFLP(FD): TOY(FD) & PAKCS.
- General insights (Golomb) and full analysis (ETP):
  - Abstraction of the constraint solver.
  - FD variables, constraints, search strategies.
  - Data structures.
  - Display solutions.
  - General expressiveness (including neatness).
- Code examples (to point out issues) + full code of models.
Contributions

Perform an in-depth modeling comparison of CP(\(FD\)) systems.

- Eight state-of-the-art systems:
  - Algebraic CP(\(FD\)): MiniZinc & ILOG OPL.
  - C++ CP(\(FD\)): Gecode & ILOG Solver.
  - CLP(\(FD\)): SICStus Prolog & SWI-Prolog.
  - CFLP(\(FD\)): \(TOY(FD)\) & PAKCS.

- General insights (Golomb) and full analysis (ETP):
  - Abstraction of the constraint solver.
  - \(FD\) variables, constraints, search strategies.
  - Data structures.
  - Display solutions.
  - General expressiveness (including neatness).
Contributions

Perform an in-depth solving comparison of CP(FD) systems.

- Same COP’s and systems (TOY(FDg), TOY(FDi), TOY(FDs)).
- Common framework for experiments:
  - System versions.
  - Global constraints (filtering algorithms).
  - Elapsed time measurement.
  - Set of instances.
- General performance comparison.
  - Ranking + slow-down.
  - Order among CP(FD) libraries.
- Gecode, ILOG Solver & SICStus clpfd specialized comparison:
  - Ranking + slow-down.
  - Search exploration analysis.
  - Head to head: TOY(FD) vs. native solver.
Contributions

Perform an in-depth solving comparison of CP(FD) systems.

- Gecode, ILOG Solver & SICStus clpfd specialized comparison:
  - Ranking + slow-down.
  - Search exploration analysis.
  - Head to head: TOY(FD) vs. native solver.
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
   4.1 Contributions
   4.2 Modeling Comparison
   4.3 Solving Comparison
   4.4 Conclusions
5. General Conclusions
Constraint Solver Abstraction

Constraint solver & its management:

- Transparent in Algebraic CP($\mathcal{FD}$), CLP($\mathcal{FD}$) & CFLP($\mathcal{FD}$).
- C++ CP($\mathcal{FD}$): Solver-targeted models.
  - Manage control of $\mathcal{FD}$ variables, constraints, objective function, constraint store, constraint propagation, search engine & search control (+ garbage collection of these elements).
  - Gecode: Modeling by inheritance. Space class constructor.
  - ILOG Solver: IloModel (store) & IloSolver (engine).

Variable access:

- Free in Algebraic CP($\mathcal{FD}$), CLP($\mathcal{FD}$) & CFLP($\mathcal{FD}$).
- C++ CP($\mathcal{FD}$): Associated to the solver hosting it.
Multiple Stage Formulation

- Algebraic CP($\mathcal{FD}$): Several models (files).
  - ETP: 4 models, as $1^{st}$ & $3^{rd}$ isolated:
    - Same variable treated as decision & parameter.
    - To solve each team independently.
  - Additional script to coordinate the models.
    - Generate the input arguments for them.
    - Execution order.

- C++ CP($\mathcal{FD}$): 1 file.
  Stages $1^{st}$ & $3^{rd}$ need different solvers.
  - Gecode: Different Space classes.

- CLP($\mathcal{FD}$) & CFLP($\mathcal{FD}$): 1 file.
  Stages coordinated by placing them in order.
Data Structures

- **Algebraic CP($\mathcal{FD}$): Static Arrays.**
  - Extra input arguments to set array size.
  - n-dimensional arrays ($n \geq 2$):
    - Different length ⇒ Extra void $\mathcal{FD}$ variables.
  - Free access and free index.

- **C++ CP($\mathcal{FD}$), CLP($\mathcal{FD}$) & CFLP($\mathcal{FD}$): Dynamic vectors & lists.**
  - Additional variables (and even predicate/functions) to access/index the elements.
Save some $\mathcal{FD}$ variables

Team shifts: By days & by workers.

- **Algebraic CP($\mathcal{FD}$):** Saving is not possible.
  - Two different bi-dimensional arrays + link constraint.
- **C++ CP($\mathcal{FD}$):** Saving is possible.
  - 1 implementation variable with 2 different modeling representations.
  - Gecode: One-dimensional arrays.
    - More *unnatural* modeling.
    - More efficient (clone method in search).
    - Breaking isolation from modeling and solving issues.
- **CLP($\mathcal{FD}$) & CFLP($\mathcal{FD}$):** Saving is possible.
  - 2 logic variables unified in $\mathcal{H}$ before turning them into $\mathcal{FD}$ variables.
  - Breaks the pure declarative view of modeling.
Constraint Propagation

- Algebraic CP($\mathcal{FD}$): Just batch mode.
- C++ CP($\mathcal{FD}$): Inherent batch mode. Artificial incremental with additional variables.
- CLP($\mathcal{FD}$): Inherent incremental mode. Artificial batch with freeze flag variables (one per stage).
- CFLP($\mathcal{FD}$): In $TOY(\mathcal{FD})$ explicit primitives to support both modes in the same program.
Neatness

**Lines of Code:**
Important: For ETP coordination of stages is not being considered for algebraic CP($\mathcal{FD}$)

<table>
<thead>
<tr>
<th>System</th>
<th>Golomb</th>
<th>Ratio</th>
<th>ETP</th>
<th>Ratio</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniZinc</td>
<td>14</td>
<td>1.00</td>
<td>123</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>ILOG OPL</td>
<td>17</td>
<td>1.21</td>
<td>138</td>
<td>1.12</td>
<td>2</td>
</tr>
<tr>
<td>TOY($\mathcal{FD}$)</td>
<td>28</td>
<td>2.00</td>
<td>215</td>
<td>1.79</td>
<td>3</td>
</tr>
<tr>
<td>PAKCS</td>
<td>33</td>
<td>2.36</td>
<td>252</td>
<td>2.05</td>
<td>4</td>
</tr>
<tr>
<td>SICStus</td>
<td>57</td>
<td>4.07</td>
<td>599</td>
<td>4.87</td>
<td>5</td>
</tr>
<tr>
<td>SWI-Prolog</td>
<td>57</td>
<td>4.07</td>
<td>635</td>
<td>5.16</td>
<td>6</td>
</tr>
<tr>
<td>ILOG Solver</td>
<td>104</td>
<td>7.43</td>
<td>762</td>
<td>6.20</td>
<td>7</td>
</tr>
<tr>
<td>Gecode</td>
<td>127</td>
<td>9.07</td>
<td>828</td>
<td>6.73</td>
<td>8</td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Improving the Performance of $\text{TOY}(\mathcal{FD})$
3. Real-Life Applications of $\text{TOY}(\mathcal{FD})$
4. Positioning $\text{TOY}(\mathcal{FD})$ w.r.t. other CP($\mathcal{FD}$) Systems
   4.1 Contributions
   4.2 Modeling Comparison
   4.3 Solving Comparison
   4.4 Conclusions
5. General Conclusions
# Gecode Related Systems Analysis

<table>
<thead>
<tr>
<th>Instance</th>
<th>System</th>
<th>SI-Dw</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-11</td>
<td>MiniZinc</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Gecode</td>
<td>2.59</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>TOY(FDg)</td>
<td>2.62</td>
<td>3</td>
</tr>
<tr>
<td>ETP-21</td>
<td>Gecode</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>TOY(FDg)</td>
<td>1.01</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>MiniZinc</td>
<td>1.53</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>System</th>
<th>Time</th>
<th>Cons.</th>
<th>Propag.</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-11</td>
<td>Gecode</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>TOY(FDg)</td>
<td>1.01</td>
<td>1.00</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td>MiniZinc</td>
<td>0.39</td>
<td>1.00</td>
<td>0.30</td>
</tr>
<tr>
<td>ETP-21</td>
<td>Gecode</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>TOY(FDg)</td>
<td>1.01</td>
<td>1.00</td>
<td>1.01</td>
</tr>
<tr>
<td></td>
<td>MiniZinc</td>
<td>1.53</td>
<td>2.60</td>
<td>1.47</td>
</tr>
</tbody>
</table>
## ILOG Solver Related Systems Analysis

### Solving Comparison

<table>
<thead>
<tr>
<th>Instance</th>
<th>System</th>
<th>SI-Dw</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-11</td>
<td>TOY(FDi)</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ILOG Solver</td>
<td>1.06</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ILOG OPL</td>
<td>1.22</td>
<td>3</td>
</tr>
<tr>
<td>ETP-21</td>
<td>ILOG Solver</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>TOY(FDi)</td>
<td>1.17</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ILOG OPL</td>
<td>1.33</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>System</th>
<th>Time</th>
<th>Vars.</th>
<th>Cons.</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-11</td>
<td>TOY(FDi)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>ILOG Solver</td>
<td>1.06</td>
<td>1.20</td>
<td>1.20</td>
</tr>
<tr>
<td></td>
<td>ILOG OPL</td>
<td>1.22</td>
<td>1.38</td>
<td>2.36</td>
</tr>
<tr>
<td>ETP-21</td>
<td>ILOG Solver</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>TOY(FDi)</td>
<td>1.15</td>
<td>1.28</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>ILOG OPL</td>
<td>1.37</td>
<td>1.04</td>
<td>1.83</td>
</tr>
</tbody>
</table>
### SICStus clpfd Related Systems Analysis

<table>
<thead>
<tr>
<th>Instance</th>
<th>System</th>
<th>SI-Dw</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-11</td>
<td>SICStus</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>PAKCS</td>
<td>1.06</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>TOY(FDs)</td>
<td>1.07</td>
<td>3</td>
</tr>
<tr>
<td>ETP-21</td>
<td>SICStus</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>PAKCS</td>
<td>1.05</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>TOY(FDs)</td>
<td>1.78</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>System</th>
<th>Time</th>
<th>Cons.</th>
<th>Resump.</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-11</td>
<td>SICStus</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>TOY(FDs)</td>
<td>1.07</td>
<td>1.00</td>
<td>0.92</td>
</tr>
<tr>
<td>ETP-21</td>
<td>SICStus</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>TOY(FDs)</td>
<td>1.74</td>
<td>1.00</td>
<td>1.03</td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP(FD) Systems
   4.1 Contributions
   4.2 Modeling Comparison
   4.3 Solving Comparison
   4.4 Conclusions
5. General Conclusions
Conclusions

*TOY(FD)* reveals as an appealing alternative for modeling COP’s.

- Abstracts constraint solver. Isolates use of several solvers and constraint distribution on them.
- Free access to variables.
- Homogeneous modeling.
- Uses dynamic data structures, handy access and index.
- Saves several *FD* variables with *H* solver.
- Batch and incremental primitives.
- CFLP: Very expressive.
- Only algebraic CP(*FD*) requires less code lines.
Conclusions

CP(FD) performance order: Gecode, ILOG Solver, SICStus clpfd.

TOY(FD) reveals as an appealing alternative for solving COP’s.

- **TOY(FDg)** vs. Gecode: Highly encouraged.
  - Achieving the same performance.
- **TOY(FDi)** vs. ILOG Solver: Encouraged.
  - Achieving nearly the same performance (low-level issues).
- **TOY(FDs)** vs. SICStus clpfd: Less encouraged.
  - Sometimes having a big overhead (low-level issues).
Outline

1. Motivation
2. Improving the Performance of $TOY(FD)$
3. Real-Life Applications of $TOY(FD)$
4. Positioning $TOY(FD)$ w.r.t. other CP($FD$) Systems
5. General Conclusions
Improved the Performance of $TOY(FD)$

- Developed the new system versions $TOY(FDg)$ & $TOY(FDi)$ (interfacing the C++ CP($FD$) solvers of Gecode and ILOG Solver), with the very same expressivity as the original $TOY(FDs)$ (interfacing the host solver of SICStus clpfd), but clearly outperforming it for all the classical benchmarks being tried.

- Enhanced the language of $TOY(FDg)$ & $TOY(FDi)$ with new search primitives, allowing to develop ad hoc search strategies for all the classical benchmarks being tried, clearly outperforming the previous performance achieved.
Real-Life Applications of $TOY(FD)$

- Applied $TOY(FDg)$, $TOY(FDi)$ & $TOY(FDs)$ to an Employee Timetabling Problem. The expressiveness of $TOY(FD)$ has been fully exploited, easing the specification of the decomposition approach used to tackle the problem.

- The better performance achieved by $TOY(FDg)$ & $TOY(FDi)$ and by the *ad hoc* strategies (specified using the new search primitives) has been confirmed for a real-life problem.

- Applied $TOY(FDg)$ to the Bin Packing Problem. Its expressiveness and solving performance (achieving the very same results as Gecode) have showed $TOY(FDg)$ as an appealing CP($FD$) system for the methodology proposed.
General Conclusions

**TOY(FD) w.r.t. CP(FD) Systems**

- The expressiveness of **TOY(FD)** for all the modeling aspects being evaluated reveals the system as an appealing alternative to the other state-of-the-art CP(FD) systems considered.
- The head-to-head solving comparison with Gecode highly encourages the use of **TOY(FDg)** (achieving the same performance).
- Also, the comparison with ILOG Solver encourages **TOY(FDi)** as well (almost achieving the same performance).
- Finally, the comparison with SICStus clpfd encourages less **TOY(FDs)** (as sometimes the overhead is quite big).
General Conclusions

Publications

- Improving the Performance of FD Constraint Solving in a CFLP System.

- Weibull-Based Benchmarks for Bin Packing.

- Applying CP(FD), CLP(FD) and CFLP(FD) to a Real-Life Employee Timetabling Problem.
  13th International Conference on Computational Science: ICCS’13. Procedia Computer Science, 18(0), pages 531-540.
A CFLP Approach for Modeling and Solving a Real Life Employee Timetabling Problem.
6th International Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems: COPLAS’11, pages 63-70.

Integrating ILOG CP Technology into TOY.
18th International Workshop on Functional and (Constraint) Logic Programming: WFLP’09. LNCS 5979, pages 27-43, Springer.

Improving the Search Capabilities of a CFLP(FD) System.
Invited for its submission to the Electronic Communications of the EASST.
Future Work

- Instantiate the interfacing scheme to other C++ CP($\mathcal{FD}$) solvers & Apply the new search strategies to other benchmarks.

- Reproduce a $\text{TOY}($FD&R$)$ version combining CP($\mathcal{FD}$) + MP solving techniques. Replace current SICStus clpfd & SICStus clpr by the C++ CP($\mathcal{FD}$) solvers ILOG Solver & ILOG CPLEX.

- Use scripting techniques to set up very controlled search scenarios. Apply different combinations of search primitives and parameters to infer relations between the problem structure and efficient search strategies to solve it.

- Add constraint generators (as the Algebraic CP($\mathcal{FD}$) forall) for improving the system neatness.
Future Work

- Use $TOY(FD)$ to model and solve other real-life problems & Use Weibull-based generated benchmarks to analyze their hardness.
- Exploit the parametericity of the ETP to solve multiple new instances. In particular use a single team to wide the applicability of the model to other Timetabling problems.
- Develop portfolio BPP solvers, fitting the intended instance to a Weibull $(k, \lambda)$ configuration and using the hardness analysis to decide applying either CP($FD$) or Heuristics to solve it.
- Add new CP($FD$) systems and benchmarks to the comparison, and also position $TOY(FD)$ w.r.t. Mathematical Programming and Heuristics techniques.
**TOY(FD):**

A CFLP(FD) system for tackling real-life CSP’s & COP’s with great expressivity and solving performance comparable to state-of-the-art CP(FD) systems.
PhD Thesis Result

$\textit{TOY}(\mathcal{FD})$:  

A CFLP(\mathcal{FD}) system for tackling real-life CSP’s & COP’s with great expressivity and solving performance comparable to state-of-the-art CP(\mathcal{FD}) systems.

Thank you for your attention!