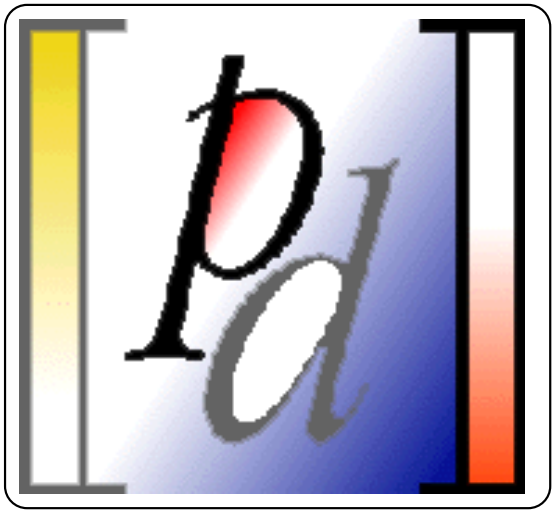
Improving the Solving Efficiency of TOY(FD) and its Application to Real-Life Problems

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 $\mathcal{TOY}(\mathcal{FD})$
3. Real-Life Applications of $\mathcal{TOY}(\mathcal{FD})$
4. Positioning $\mathcal{TOY}(\mathcal{FD})$ w.r.t. other CP($\mathcal{FD}$) Systems
5. General Conclusions

Outline

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: $F \equiv$ *minimize*/*maximize expr*.
- Solutions: Candidates entailing C and optimizing the *expr* of F.

Approaches to tackling CSP's and COP's

CSP's and COP's extensively studied. Different approaches:

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

CP($\mathcal{FD}$) Solving Techniques

- CP($\mathcal{FD}$) $\Rightarrow$ 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($\mathcal{FD}$).**
   - General-purpose. Declarative. Functional Logic Programming languages.
   - Some state-of-the-art systems: $\underline{\mathcal{TOY}(\mathcal{FD})}$, PAKCS.

$\mathcal{TOY}(\mathcal{FD})$ Main Description

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

Type	Constraints & Operators
Relational Constraints	==, /=, #>, #>=, #<, #<=
Arithmetic Operators	#+, #-, #*, #/
Propositional Constraint	post_implication
Domain Constraints	domain, domain_valArray
Global Constraints	all_different, count, sum, scalar_product

$\mathcal{TOY}(\mathcal{FD})$ Language Modeling Features

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

- Haskell-like syntax (variables upper-case & functions lower-case).
- Datatype declarations, operators & functions.

Functional Features	**Logic Features**
Functional notation	Relational notation
Curried expressions	Non-determinism
Higher-order functions	Backtracking
Lazy evaluation	Logical variables
Patterns	Domain variables
Partial application	Reasoning with models
Types	
Polymorphism	
Constraint composition	

$\mathcal{TOY}(\mathcal{FD})$ Model Example for *N*-Queens

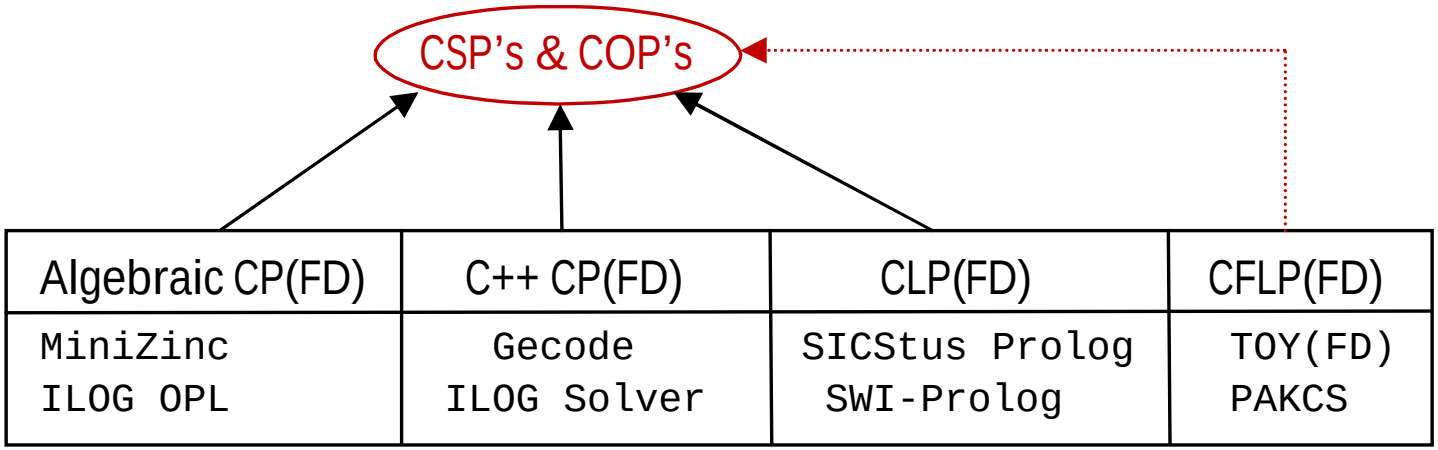
```
include "misc.toy"
include "cflpfd.toy"

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]
```

$\mathcal{TOY}(\mathcal{FD})$ 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
```

CP($\mathcal{FD}$) Community Nowadays

- 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.

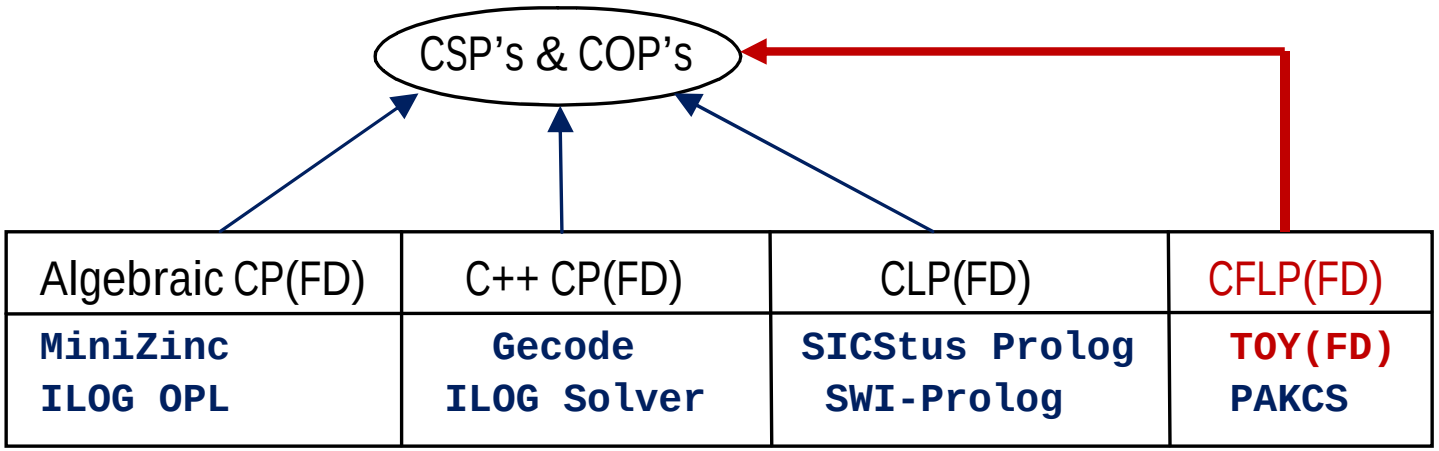


PhD Thesis Goal

Perform an empirical evaluation of the applicability of CFLP($\mathcal{FD}$) to tackle CSP's and COP's

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

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



Outline

Outline

Possible Improvements

- Amdahl's law: Improve system performance by altering a component $\Rightarrow$ Fraction of time component is used.
- Hypothesis: Combinatorial nature of CSP's & COP's.
  - As instances $\uparrow \Rightarrow$ Search exploration is the key performance factor.
- Conclusion: 2 ways to improve $\mathcal{TOY}(\mathcal{FD})$ 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($\mathcal{FD}$) solvers

Develop a interface scheme for C++ CP($\mathcal{FD}$) solvers into $\mathcal{TOY}(\mathcal{FD})$.

(Applicable to $\mathcal{TOY}(\mathcal{FD})$, CLP($\mathcal{FD}$) and CFLP($\mathcal{FD}$) Prolog systems).

- Identify $\mathcal{TOY}(\mathcal{FD})$ commands to coordinate the solver.
- Targets: Gecode & ILOG Solver $\Rightarrow$ $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$.
- Establish communication Prolog and C++ components.
- Manage different variable, constraint and types of system & solver.
- Adapt the C++ CP($\mathcal{FD}$) solver to the CFLP($\mathcal{FD}$) requirements:
  - Model reasoning.
  - Multiple search strategies.
  - Incremental and Batch propagation modes.

C++ CP($\mathcal{FD}$) solvers

Develop a interface scheme for C++ CP($\mathcal{FD}$) solvers into $\mathcal{TOY}(\mathcal{FD})$.

(Applicable to $\mathcal{TOY}(\mathcal{FD})$, CLP($\mathcal{FD}$) and CFLP($\mathcal{FD}$) Prolog systems).

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

Search Strategies

Enhance $\mathcal{TOY}(\mathcal{FD})$ language with search primitives.

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

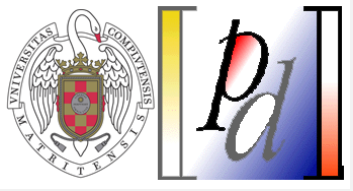
- Provide a more detailed search specification to the solver.
- Targets: Extend Gecode & ILOG Solver libraries $\Rightarrow$ $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$.
- Develop 8 new search primitives. Novel concepts:
  - Exhaustive breath exploration + sort criteria.
  - Fragment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

Search Strategies

Enhance $\mathcal{TOY}(\mathcal{FD})$ language with search primitives.

(Applicable to CLP($\mathcal{FD}$) and CFLP($\mathcal{FD}$) Prolog systems interfacing C++ CP($\mathcal{FD}$) 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 $\mathcal{TOY}(\mathcal{FD})$ level.



Outline

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

- Load the SICStus `clpfd` library.

(Now) $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$:

- Use SICStus *Prolog-C++ connection framework*.
- Use additional Prolog and C++ *global* data structures.

Variable & Constraint Representations

(Before) $\mathcal{TOY}(\mathcal{FD}s)$:

- System & $\mathcal{FD}$ solver = variable representation.
- `clpfd` API exposes the store constraint network.

(Now) $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$:

- System & $\mathcal{FD}$ solver $\neq$ (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) $\mathcal{TOY}(\mathcal{FD}s)$:

- Automatic restoration of the store on backtracking.

(Now) $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$:

- 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) $\mathcal{TOY}(\mathcal{FD}s)$:

- `clpfd` supports it.

(Now) $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$:

- 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 $\mathcal{TOY}(\mathcal{FD})$ programs.

(Before) $\mathcal{TOY}(\mathcal{FD}s)$:

- Use delayed goals (`freeze` predicates).
- Additional *flag* modified by primitives.

(Now) $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$:

- Additional tuple (B, cVs, cCs).
  - Variables and constraints are posted to cV & cC, but not to store$^{\mathcal{FD}}$.
  - 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.

Instance	%	Sp-Up	Instance	%	Sp-Up
Q-105 FDs	97.3	1.00	Q-120 FDs	99.9	1.00
Q-105 FDg	96.6	**3.33**	Q-120 FDg	99.9	**3.45**
Q-105 FDi	92.6	**2.78**	Q-120 FDi	99.9	**2.94**
L-127 FDs	97.7	1.00	L-131 FDs	99.9	1.00
L-127 FDg	96.3	**2.63**	L-131 FDg	99.8	**2.56**
L-127 FDi	88.9	**2.78**	L-131 FDi	99.4	**2.86**
G-10 FDs	99.6	1.00	G-11 FDs	99.9	1.00
G-10 FDg	99.1	**3.45**	G-11 FDg	99.9	**3.57**
G-10 FDi	99.2	**2.08**	G-11 FDi	99.9	**2.13**

Outline

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

$\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$: 8 new primitives

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

lab	frag
labB	fragB
labW	fragW
labO	fragO

Search Strategies lab & labB

lab:: varOrd $\rightarrow$ valOrd $\rightarrow$ int $\rightarrow$ [int] $\rightarrow$ bool

- Classical search. Collects solutions, one by one.

labB:: varOrd $\rightarrow$ valOrd $\rightarrow$ int $\rightarrow$ [int] $\rightarrow$ bool

- Equivalent to lab, but explores just one tree branch.

Variable & Value order criteria:

- Full Gecode catalog.
- *Specify criteria at $\mathcal{TOY}(\mathcal{FD})$ level*.
- *Label only a subset of the variables*.

Search Strategies lab & labB

```
-----------------------------------------------------------
myVarOrder:: [int] -> int
myrVarOrder Vs =
  fst (foldl 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 labW

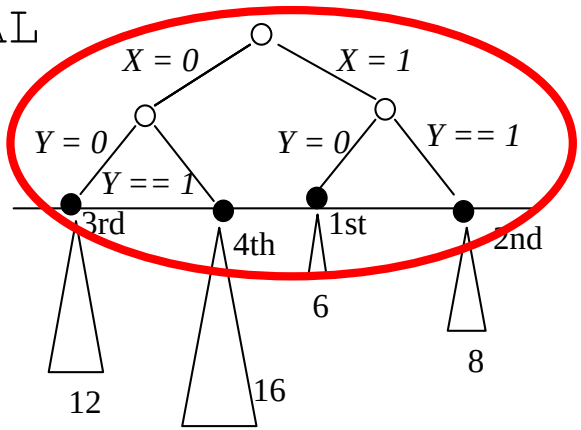
labW:: varOrd → bound → int → [int] → bool

- *Exhaustive Breadth exploration + criteria to sort solutions.*

Sort criteria:

- Smallest/largest local/global search space.
- Specify criteria at $\mathcal{TOY}(\mathcal{FD})$ level.

```
TOY(FD)> domain [X,Y] 0 1, domain [V1,V2] 0 3,
         (X #= 1) #=> (V1 #> 1), (Y #= 0) #=> (V2 #> 0),
         labW unassignedLeftVar smallestTree 2 [X,Y,V1,V2],
         ... REST OF TOY GOAL
```



Search Strategies frag, fragB, fragW

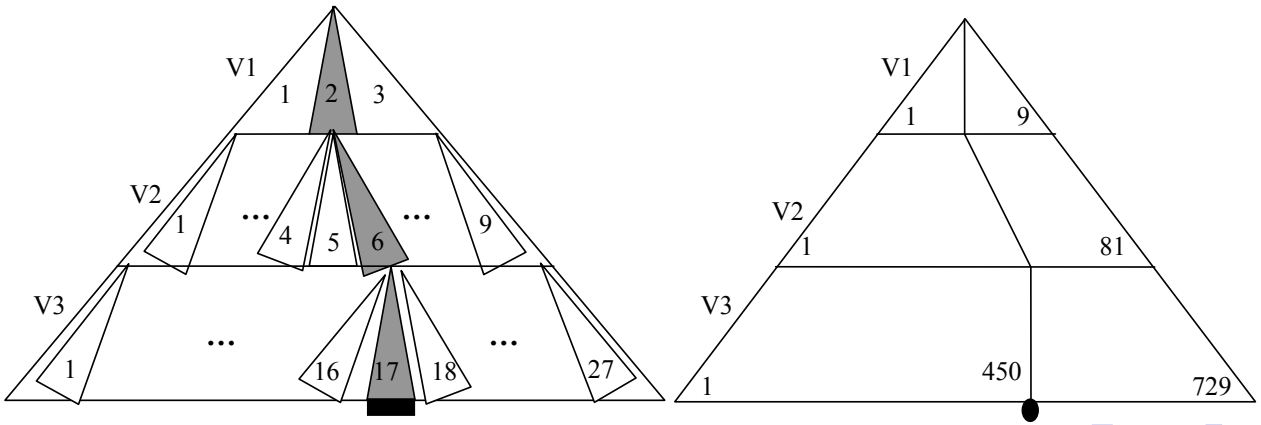
frag:: domFrag → varOrd → intervalOrd → int → [int] → bool
fragB:: domFrag → varOrd → intervalOrd → int → [int] → bool
fragW:: domFrag → varOrd → bound → int → [int] → bool

- *Variable domains fragmented instead of labeled*.

Example: A goal contains `V` variables and `C` constraints.

- $\texttt{V}' \equiv \{\texttt{V1}, \texttt{V2}, \texttt{V3}\}$ 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.

Instance	Sp-Up-Bs	Sp-Up-Is	off/on
Q-105 FDi	1.00	1.00	**1.61**
Q-105 FDg	1.19	**10.00**	**12.50**
Q-120 FDi	1.00	1.00	**138.74**
Q-120 FDg	1.19	**12.50**	**1443.11**
L-127 FDi	1.00	1.00	**3.70**
L-127 FDg	0.94	**3.03**	**12.50**
L-131 FDi	1.00	1.00	**73.11**
L-131 FDg	0.88	**3.57**	**298.58**
G-10 FDi	1.00	1.00	**2.44**
G-10 FDg	1.69	**1.75**	**2.50**
G-11 FDi	1.00	1.00	**1.64**
G-11 FDg	1.72	**1.75**	**1.69**

Outline

Conclusions

Interfacing C++ CP($\mathcal{FD}$) solvers:

- $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i) \Rightarrow$ C++ solver coordination overhead.
- Benchmark: Instances $\uparrow \Rightarrow$ Search Time $\uparrow$ & overhead $\downarrow$.
- $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$ outperform $\mathcal{TOY}(\mathcal{FD}s)$ 1.15-3.57$\times$.
- Correlation: Search Time $\uparrow \Rightarrow$ outperform $\uparrow$.
- Propagation mode: Not relevant.

Conclusions

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

- Benchmark: Problem solution structures $\Rightarrow$ *Ad hoc* search strategy.
- New search strategies: $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$ outperform themselves up to 1,000$\times$.
- $\mathcal{TOY}(\mathcal{FD}g)$ outperform $\mathcal{TOY}(\mathcal{FD}i)$ $\Rightarrow$ More efficient library extension.
- $\mathcal{TOY}(\mathcal{FD})$ criteria: Overhead due to interaction of different $\mathcal{TOY}(\mathcal{FD})$ architecture layers.

Outline

Outline

Contributions

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

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

Contributions

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

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

Contributions

Applying $\mathcal{TOY}(\mathcal{FD})$ + Heuristics to Bin Packing Problem.
(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 $\mathcal{TOY}(\mathcal{FD})$ as an appealing CP($\mathcal{FD}$) system to be used in the analysis.

Contributions

Applying $\mathcal{TOY}(\mathcal{FD})$ + Heuristics to Bin Packing Problem.
(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 $\mathcal{TOY}(\mathcal{FD})$ as an appealing CP($\mathcal{FD}$) system to be used in the analysis.

Outline

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 *working_days = (20,22,24), 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:
  - Have 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)*.

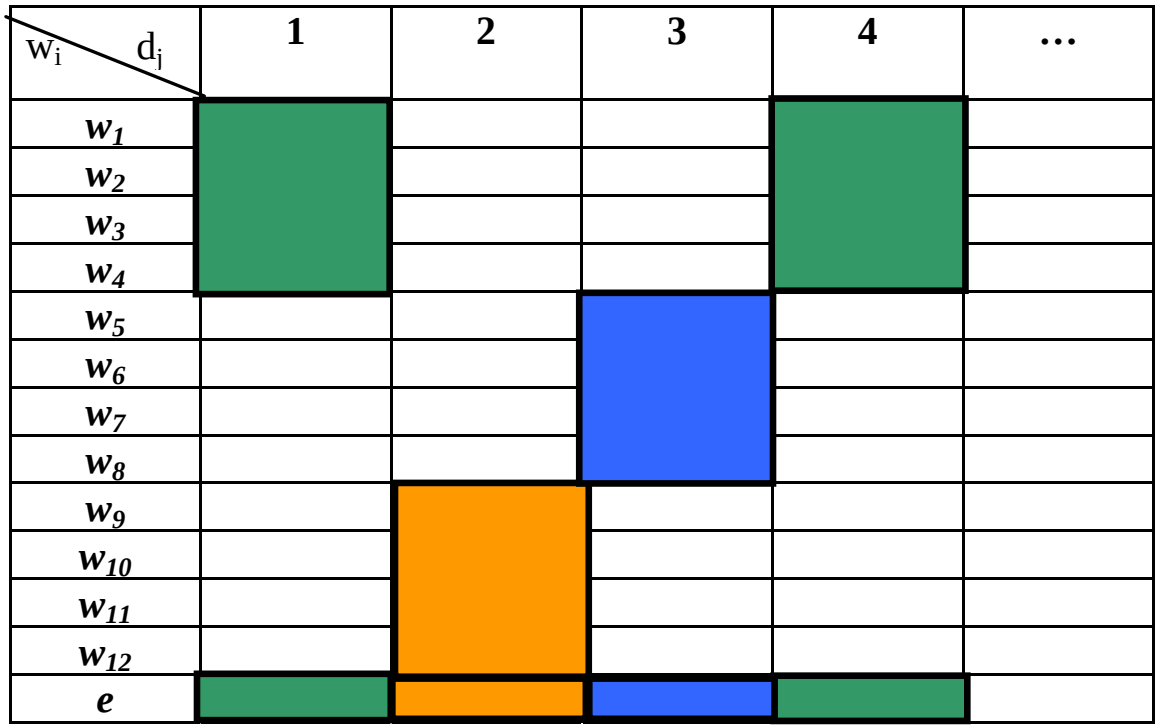
ETP Solving Approach

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

w_i \ d_j	**1**	**2**	**3**	**4**	**…**
w_1	$tt_{1,1}$	0	0	$tt_{4,1}$	
w_2	20	0	0	$tt_{4,2}$	
w_3	$tt_{1,3}$	0	0	$tt_{4,3}$	
w_4	$tt_{1,4}$	0	0	$tt_{4,4}$	
w_5	0			0	
w_6	0			0	
w_7	0			0	
w_8	0			0	
w_9	0			0	
w_{10}	0			0	
w_{11}	0			0	
w_{12}	0			0	
e	$tt_{1,13}$	$tt_{2,13}$	$tt_{3,13}$	$tt_{4,13}$	…

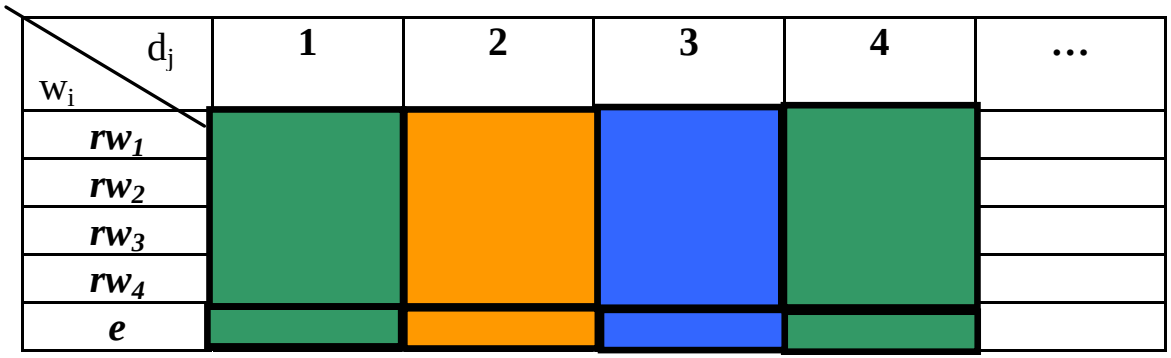
ETP Solving Approach

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



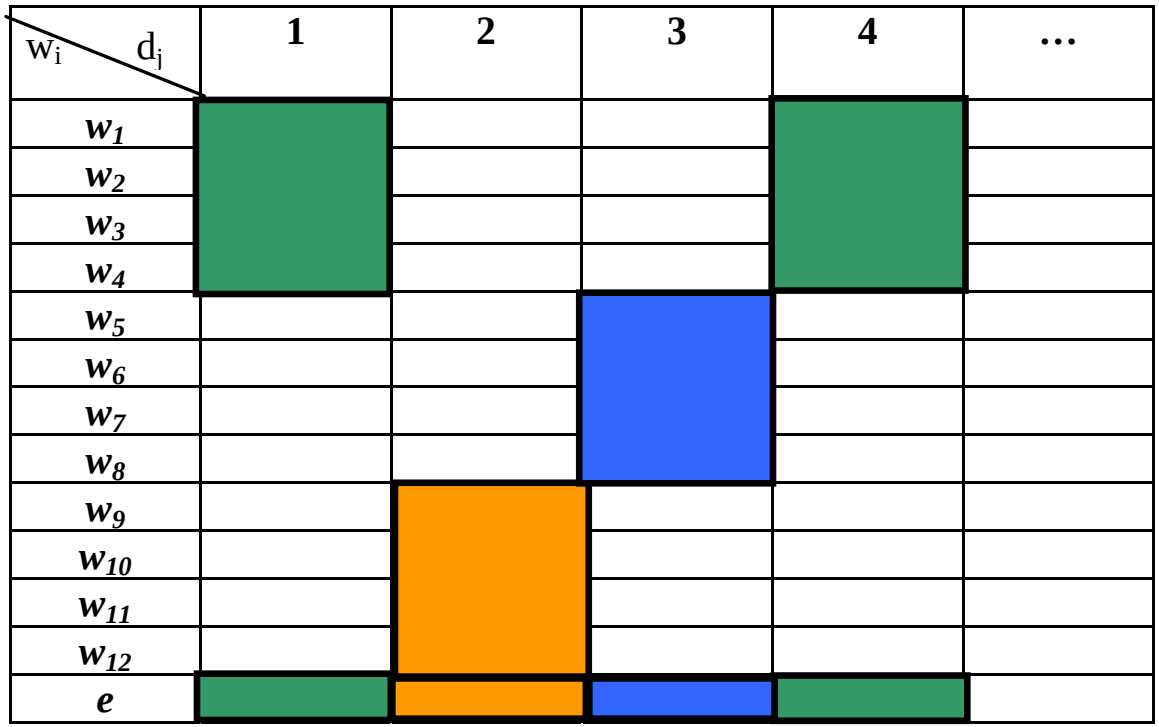
#*Table* Needed to Map *timetabling*?

- 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.

w_i d_j	1	2	3	4	...
w_1					
w_2					
w_3					
w_4					
w_5					
w_6					
w_7					
w_8					
w_9					
w_{10}					
w_{11}					
w_{12}					
e					

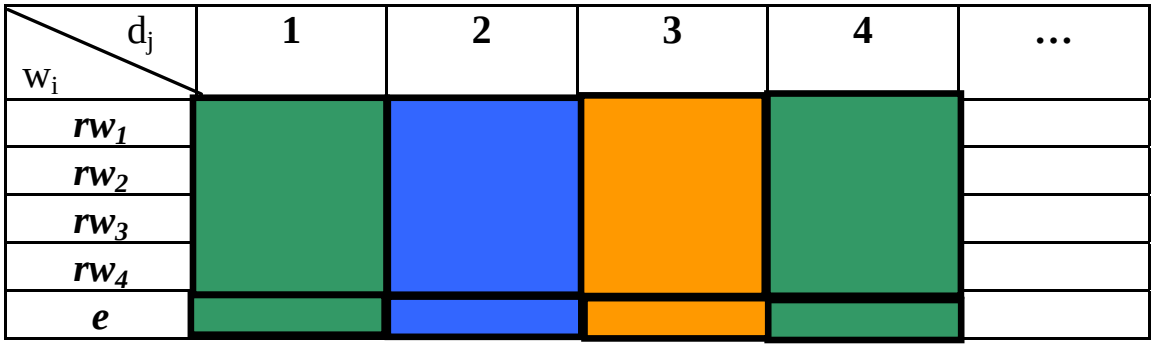
#*Table* Needed to Map *timetabling*?

- 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.



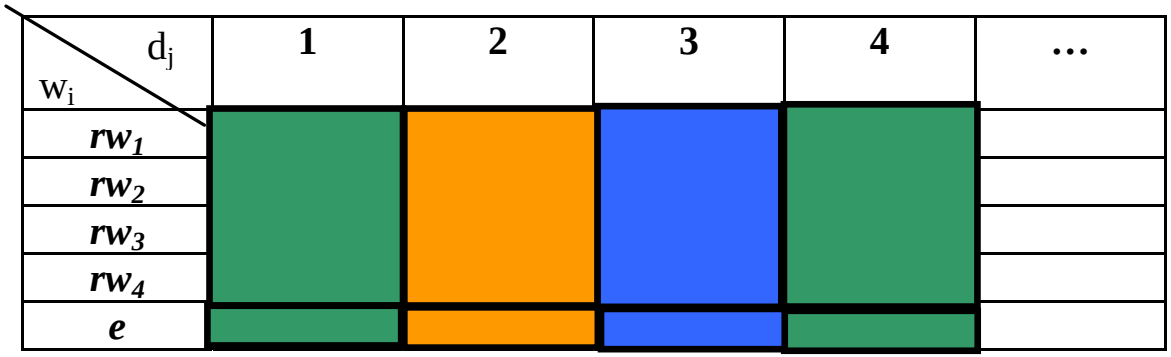
#*Table* Needed to Map *timetabling*?

- 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* 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.



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 $\mathcal{TOY}(\mathcal{FD})$ Useful?

Suitable for the architecture of the solution:

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

Suitable for specifying the problem:

- $\mathcal{TOY}(\mathcal{FD})$ 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.

Instance	%	Sp-Up
ETP-7 FDs	35.5	1.00
ETP-7 FDg	2.1	**1.30**
ETP-7 FDi	0.3	**0.18**
ETP-15 FDs	91.9	1.00
ETP-15 FDg	60.0	**3.57**
ETP-15 FDi	51.2	**0.98**
ETP-21 FDs	99.9	1.00
ETP-21 FDg	99.1	**6.67**
ETP-21 FDi	98.6	**3.13**

Experiments

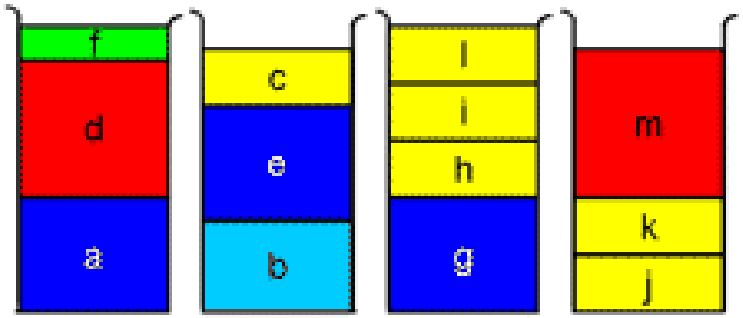
Ad hoc strategy.

Instance	Sp-Up-Bs	Sp-Up-Is	off/on
ETP-7 FDi	1.00	1.00	**0.95**
ETP-7 FDg	7.14	**7.14**	**0.95**
ETP-15 FDi	1.00	1.00	**1.25**
ETP-15 FDg	3.70	**5.56**	**1.89**
ETP-21 FDi	1.00	1.00	**4.76**
ETP-21 FDg	2.17	**2.38**	**5.26**

Outline

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).
- Apply CP($\mathcal{FD}$) & Heuristics methods to solve the benchmark.

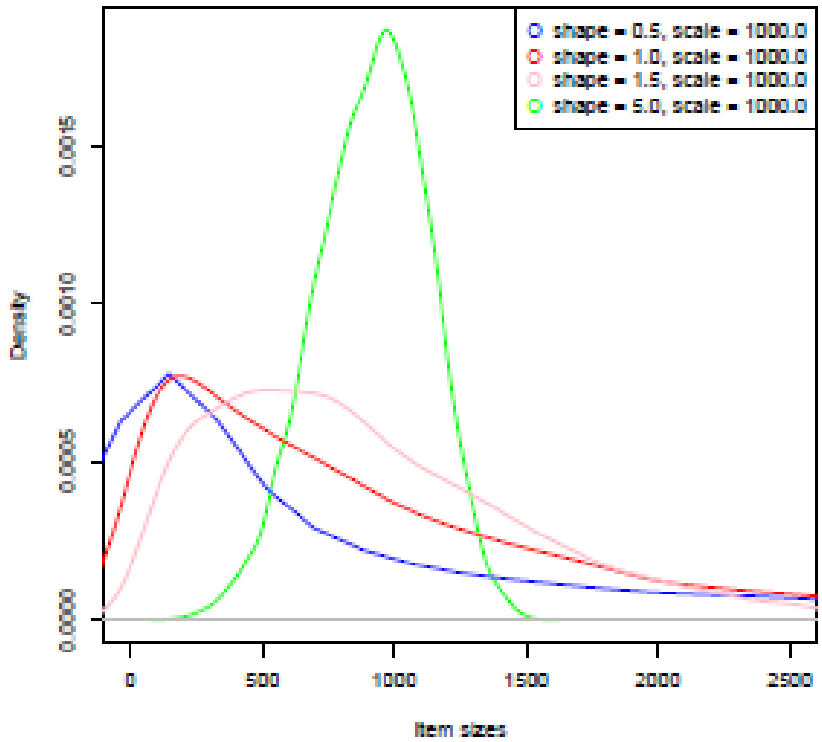
Parameterizable benchmark: Weibull statistical model.

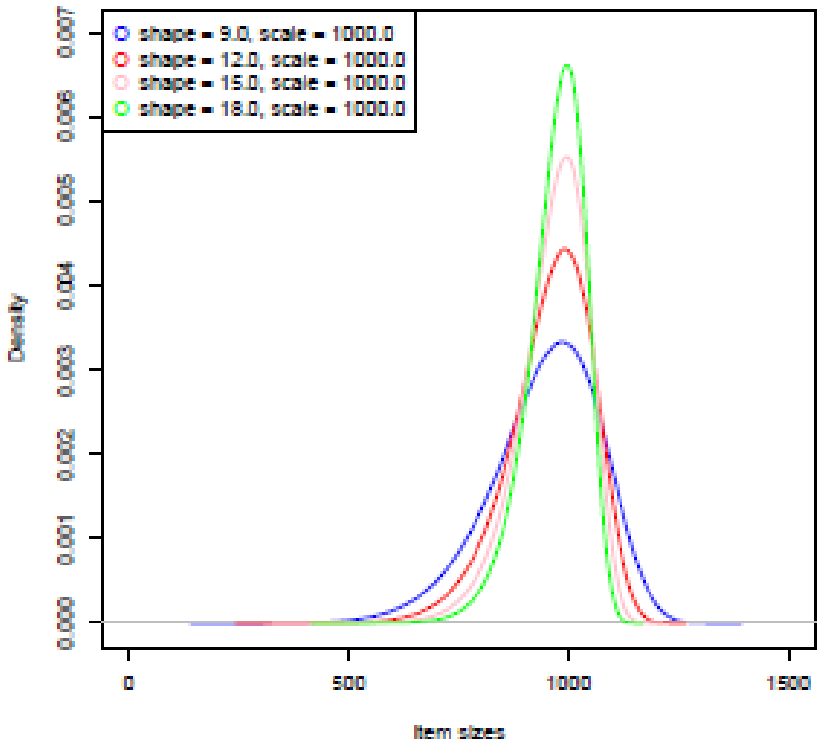
- Continuous probability distribution.
- Allows to represent nearly any unimodal distribution.
- Very flexible (parameterizable on its shape k and scale λ).

$$f(x, \lambda, k) = \begin{cases} \frac{k}{\lambda} \cdot (\frac{x}{\lambda})^{k-1} \cdot e^{-(x/\lambda)^k} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Precise Control of Generated Instances

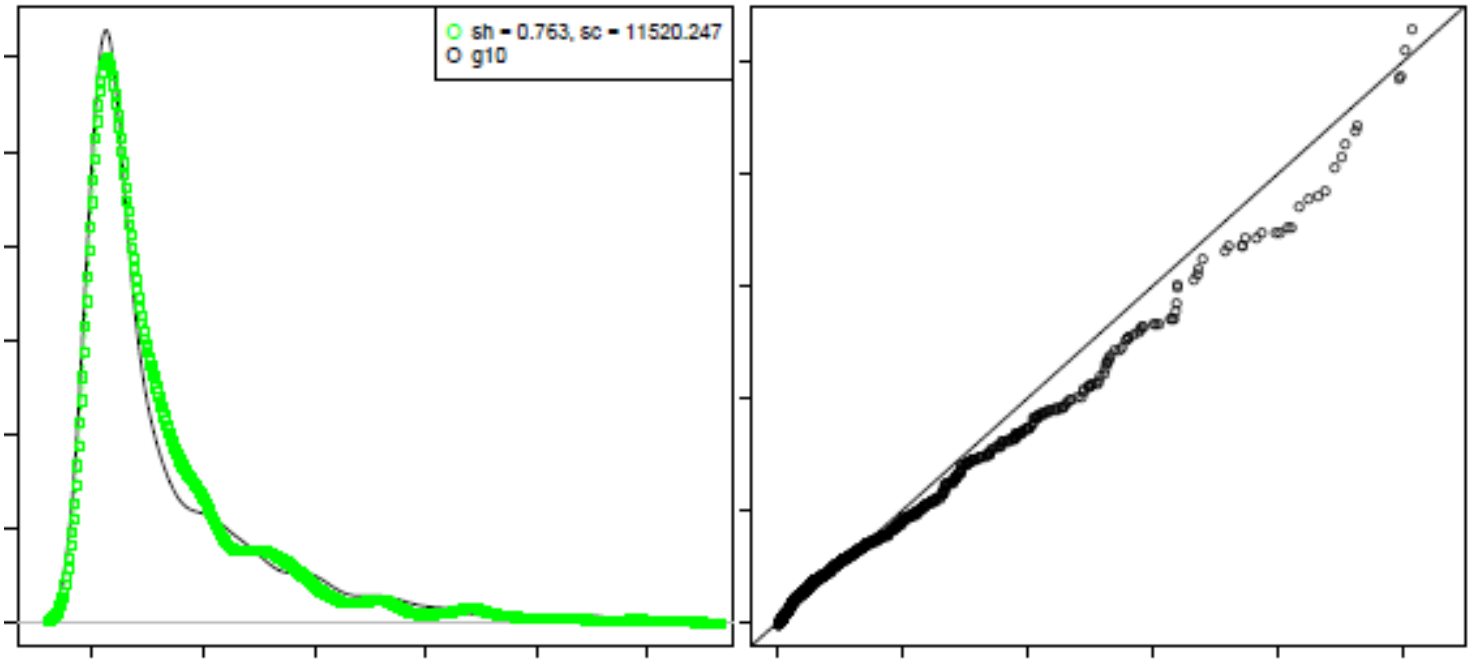
- A benchmark generator: Weibull (k,λ).
  - Scale (λ) = 1000.
  - Shape (k) $\in \{0.1, 0.2, \ldots, 19.9\}$.
  - 100 instances of 100 items per (k, λ) combination.





Well-Fitting Existing Real-Life BPP Instances

- By observation:
  - R: Maximum Likilihood Fitting & Quantile-Quantile Plots.
- By statistical tests:
  - Kolmogorov-Smirnov & χ^2



Benchmark Solving: CP($\mathcal{FD}$) & Heuristics

- CP($\mathcal{FD}$) model.
  - Based on the most efficient model of the Gecode distribution.
  - Lower & Upper bound.
  - Bin packing global constraint.
  - Complete Decreasing Best Fit 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.

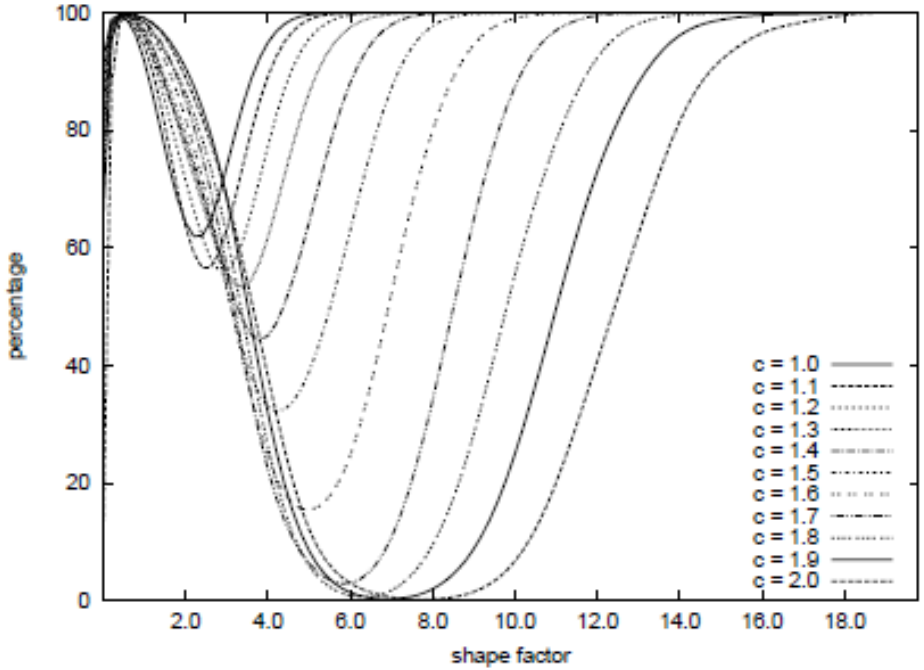
Why Use $\mathcal{TOY}(\mathcal{FD})$ as a CP($\mathcal{FD}$) System?

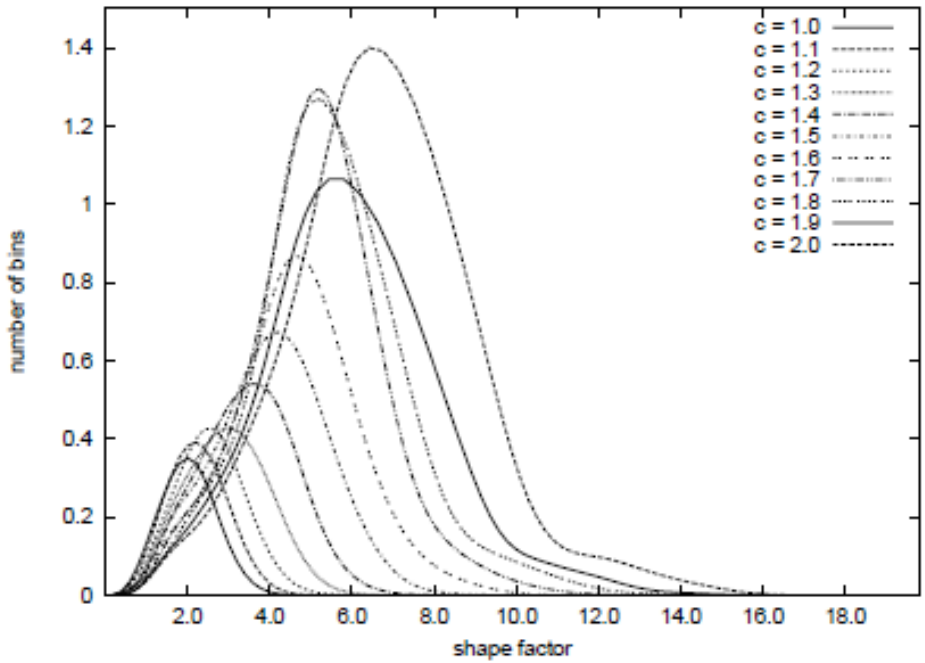
Appealing alternative to Gecode

- 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:
  - $\mathcal{TOY}(\mathcal{FD}g)$ 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

- CP($\mathcal{FD}$): Hardness categorization by combination C + (k,λ):
  - 80%-100%, 60%-80%, 40%-60%, 20%-40%, 0%-20%.
  - As $C \uparrow \Rightarrow$ hardness $\uparrow$.
- Heuristics:
  - Good alternative to CP($\mathcal{FD}$).
  - As CP($\mathcal{FD}$) hardness $\uparrow \Rightarrow$ Heuristics gap $\uparrow$.





Outline

Conclusions

- ETP complex formulation exploits $\mathcal{TOY}(\mathcal{FD})$ high expressivity.
- ETP Solving results are more instance dependent.
  - $\mathcal{TOY}(\mathcal{FD}g)$ outperforms $\mathcal{TOY}(\mathcal{FD}s)$ more.
    $\mathcal{TOY}(\mathcal{FD}i)$ worse, similar and better than $\mathcal{TOY}(\mathcal{FD}s)$, resp.
  - Small instances: $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$ $\uparrow$ overhead C++ solver.
  - Propagation mode: More relevant in some cases.
  - *Ad hoc* search strategy: Less relevant.

Conclusions

- Both CP($\mathcal{FD}$) and Heuristics suitable for solving BPP.
  Instance + $C \Rightarrow$ CP($\mathcal{FD}$) & Heuristics tradeoff (time, quality).
- CP($\mathcal{FD}$): Any $C \Rightarrow$ hard interval (k, λ).
  - 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, λ) + 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

Outline

Contributions

Perform an in-depth modeling comparison of CP($\mathcal{FD}$) systems.

- Two COP's:
  - Classical CP($\mathcal{FD}$) puzzle benchmark: Golomb Rulers.
  - Real-life application: ETP.
- Eight state-of-the-art systems:
  - Algebraic CP($\mathcal{FD}$): MiniZinc & ILOG OPL.
  - C++ CP($\mathcal{FD}$): Gecode & ILOG Solver.
  - CLP($\mathcal{FD}$): SICStus Prolog & SWI-Prolog.
  - CFLP($\mathcal{FD}$): $\mathcal{TOY}(\mathcal{FD})$ & PAKCS.
- General insights (Golomb) and full analysis (ETP):
  - Abstraction of the constraint solver.
  - $\mathcal{FD}$ 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($\mathcal{FD}$) systems.

- Eight state-of-the-art systems:
  - Algebraic CP($\mathcal{FD}$): MiniZinc & ILOG OPL.
  - C++ CP($\mathcal{FD}$): Gecode & ILOG Solver.
  - CLP($\mathcal{FD}$): SICStus Prolog & SWI-Prolog.
  - CFLP($\mathcal{FD}$): $\mathcal{TOY}(\mathcal{FD})$ & PAKCS.
- General insights (Golomb) and full analysis (ETP):
  - Abstraction of the constraint solver.
  - $\mathcal{FD}$ variables, constraints, search strategies.
  - Data structures.
  - Display solutions.
  - General expressiveness (including neatness).

Contributions

Perform an in-depth solving comparison of CP($\mathcal{FD}$) systems.

- Same COP's and systems ($\mathcal{TOY}(\mathcal{FD}g)$, $\mathcal{TOY}(\mathcal{FD}i)$, $\mathcal{TOY}(\mathcal{FD}s)$).
- 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($\mathcal{FD}$) libraries.
- Gecode, ILOG Solver & SICStus `clpfd` specialized comparison:
  - Ranking + slow-down.
  - Search exploration analysis.
  - Head to head: $\mathcal{TOY}(\mathcal{FD})$ vs. native solver.

Contributions

Perform an in-depth solving comparison of CP($\mathcal{FD}$) systems.

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

Outline

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 lenght $\Rightarrow$ 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 & Search

- 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 $\mathcal{TOY}(\mathcal{FD})$ explicit primitives to support both modes in the same program.

- Algebraic CP($\mathcal{FD}$), CLP($\mathcal{FD}$) & CFLP($\mathcal{FD}$): Expressive primitives.
- C++ CP($\mathcal{FD}$): Explicit declaration of some components:
  - Gecode: Low-level specification (attached to the tree exploration).
    - Copy + cost function methods. Search engine + `Space` objects.
  - ILOG Solver: Declaration with combination of some primitives.

Neatness

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

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

Outline

Gecode Related Systems Analysis

Instance	System	SI-Dw	Ranking
G-11	MiniZinc	1.00	1
	Gecode	**2.59**	2
	$\mathcal{TOY}(\mathcal{FD}g)$	**2.62**	3
ETP-21	Gecode	**1.00**	1
	$\mathcal{TOY}(\mathcal{FD}g)$	**1.01**	2
	MiniZinc	1.53	3

Instance	System	Time	Cons.	Propag.
G-11	Gecode	1.00	1.00	1.00
	$\mathcal{TOY}(\mathcal{FD}g)$	**1.01**	1.00	**1.02**
	MiniZinc	**0.39**	1.00	**0.30**
ETP-21	Gecode	1.00	1.00	1.00
	$\mathcal{TOY}(\mathcal{FD}g)$	**1.01**	1.00	**1.01**
	MiniZinc	**1.53**	2.60	**1.47**

ILOG Solver Related Systems Analysis

Instance	System	SI-Dw	Ranking
G-11	$\mathcal{TOY}(\mathcal{FD}i)$	**1.00**	1
	ILOG Solver	**1.06**	2
	ILOG OPL	1.22	3
ETP-21	ILOG Solver	**1.00**	1
	$\mathcal{TOY}(\mathcal{FD}i)$	**1.17**	2
	ILOG OPL	1.33	3

Instance	System	Time	Vars.	Cons.
G-11	$\mathcal{TOY}(\mathcal{FD}i)$	**1.00**	**1.00**	**1.00**
	ILOG Solver	**1.06**	**1.20**	**1.20**
	ILOG OPL	1.22	1.38	2.36
ETP-21	ILOG Solver	**1.00**	**1.00**	**1.00**
	$\mathcal{TOY}(\mathcal{FD}i)$	**1.15**	**1.28**	**1.05**
	ILOG OPL	1.37	1.04	1.83

SICStus `clpfd` Related Systems Analysis

Instance	System	SI-Dw	Ranking
G-11	SICStus	**1.00**	1
	PAKCS	1.06	2
	$\mathcal{TOY}(\mathcal{FD}s)$	**1.07**	3
ETP-21	SICStus	**1.00**	1
	PAKCS	1.05	2
	$\mathcal{TOY}(\mathcal{FD}s)$	**1.78**	3

Instance	System	Time	Cons.	Resump.
G-11	SICStus	1.00	1.00	1.00
	$\mathcal{TOY}(\mathcal{FD}s)$	**1.07**	**1.00**	**0.92**
ETP-21	SICStus	1.00	1.00	1.00
	$\mathcal{TOY}(\mathcal{FD}s)$	**1.74**	**1.00**	**1.03**

Outline

Conclusions

$\mathcal{TOY}(\mathcal{FD})$ 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 $\mathcal{FD}$ variables with $\mathcal{H}$ solver.
- Batch and incremental primitives.
- FLP: Very expressive.
- Only algebraic CP($\mathcal{FD}$) requires less code lines.

Conclusions

CP($\mathcal{FD}$) performance order: Gecode, ILOG Solver, SICStus `clpfd`.
$\mathcal{TOY}(\mathcal{FD})$ **reveals as an appealing alternative for solving COP's.**

- $\mathcal{TOY}(\mathcal{FD}g)$: Highly encouraged.
  - Constraint network: $\mathcal{TOY}(\mathcal{FD}g)$ interface $\simeq$ Gecode native model.
  - Small instances: $\mathcal{TOY}(\mathcal{FD}g)$ overhead $\uparrow$.
    Big instances: Gecode & $\mathcal{TOY}(\mathcal{FD}g)$ $\simeq$ performance.
- $\mathcal{TOY}(\mathcal{FD}i)$: Encouraged.
  - Constraint network: $\mathcal{TOY}(\mathcal{FD}i)$ interface $\neq$ ILOG Solver native model (ILOG Concert).
  - Performance dependent of the problem formulation.
- $\mathcal{TOY}(\mathcal{FD}s)$: Less encouraged.
  - Constraint network: $\mathcal{TOY}(\mathcal{FD}s)$ interface $\neq$ ILOG Solver native model (Indexicals).
  - Small instances: $\mathcal{TOY}(\mathcal{FD}s)$ overhead $\downarrow$.
    Big instances: $\mathcal{TOY}(\mathcal{FD}s)$ overhead $\uparrow$.

Outline

Improved the Performance of $\mathcal{TOY}(\mathcal{FD})$

- Developed the new system versions $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$ (interfacing the C++ CP($\mathcal{FD}$) solvers of Gecode and ILOG Solver), with the very same expressivity as the original $\mathcal{TOY}(\mathcal{FD}s)$ (interfacing the host solver of SICStus `clpfd`), but clearly outperforming it for all the classical benchmarks being tried.
- Enhanced the language of $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$ 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 $\mathcal{TOY}(\mathcal{FD})$

- Applied $\mathcal{TOY}(\mathcal{FD}g)$, $\mathcal{TOY}(\mathcal{FD}i)$ & $\mathcal{TOY}(\mathcal{FD}s)$ to an Employee Timetabling Problem. The expressiveness of $\mathcal{TOY}(\mathcal{FD})$ has been fully exploited, easing the specification of the decomposition approach used to tackle the problem.
- The better performance achieved by $\mathcal{TOY}(\mathcal{FD}g)$ & $\mathcal{TOY}(\mathcal{FD}i)$ and by the *ad hoc* strategies (specified using the new search primitives) has been confirmed for a real-life problem.
- Applied $\mathcal{TOY}(\mathcal{FD}g)$ to the Bin Packing Problem. Its expressiveness and solving performance (achieving the very same results as Gecode) have showed $\mathcal{TOY}(\mathcal{FD}g)$ as an appealing CP($\mathcal{FD}$) system for the methodology proposed.

$\mathcal{TOY}(\mathcal{FD})$ w.r.t. CP($\mathcal{FD}$) Systems

- The expressiveness of $\mathcal{TOY}(\mathcal{FD})$ for all the modeling aspects being evaluated reveals the system as an appealing alternative to the other state-of-the-art CP($\mathcal{FD}$) systems considered.
- The head-to-head solving comparison with Gecode highly encourages the use of $\mathcal{TOY}(\mathcal{FD}g)$ (achieving the same performance).
- Also, the comparison with ILOG Solver encourages $\mathcal{TOY}(\mathcal{FD}i)$ as well (almost achieving the same performance).
- Finally, the comparison with SICStus `clpfd` encourages less $\mathcal{TOY}(\mathcal{FD}s)$ (as sometimes the overhead is quite big).

Publications

- **Improving the Performance of FD Constraint Solving in a CFLP System**.
  11th International Symposium on Functional and Logic Programming: FLOPS'12. LNCS 7294, pages 88-103, Springer.
- **Weibull-Based Benchmarks for Bin Packing**.
  18th International Conference on Principles and Practice on Constraint Programming: CP'12. LNCS 7514, pages 207-222, Springer.
- **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.

Publications

- **Integrating ILOG CP Technology into TOY**.
  18th International Workshop on Functional and (Consraint) Logic Programming: WFLP'09. LNCS 5979, pages 27-43, Springer.
- **Improving the Search Capabilities of a CFLP(FD) System**.
  XIII Spanish Conference on Programming and Computer Languages: PROLE'13, pages 273-287. Invited for its submission to the Electronic Communications of the EASST.
- **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.
- **Comparing TOY(FD) with State-of-the-Art Constraint Programming Systems**.
  Technical report DSIC-14.13 of Departamento de Sistemas Informáticos y Computación (DSIC) of the Complutense University of Madrid (UCM), pages 1-135.

Future Work

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

Future Work

- Use $\mathcal{TOY}(\mathcal{FD})$ 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, λ) configuration and using the hardness analysis to decide applying either CP($\mathcal{FD}$) or Heuristics to solve it.
- Add new CP($\mathcal{FD}$) systems and benchmarks to the comparison, and also position $\mathcal{TOY}(\mathcal{FD})$ w.r.t. Mathematical Programming and Heuristics techniques.

PhD Thesis Result

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

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.

Outline

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

End

Thank you for your attention!