Analysis of the FD Variables and Constraints Generated by TOY(FDi) and ILOG for Solving Different Finite Domain Constraint Problems *

Technical Report 06/12
Ignacio Castiñeiras and Fernando Sáenz-Pérez
Complutense University of Madrid, Spain
ncasti@fdi.ucm.es, fernan@sip.ucm.es

Abstract

This technical report presents a comparison between the FD variables and constraints that TOY(FDi) and ILOG Solver generate for modeling and solving two different FD constraint problems: A real-life Employee Timetabling Problem and the Golomb Ruler Problem. A concrete instance of each problem is selected and described. Then, a trace of its execution in TOY(FDi) and ILOG is presented, pointing out the different function calls performed, and the amount of FD variables and constraints associated to each of them. Explanations for deviations on the number of variables and constraints between TOY(FDi) and ILOG are provided.

---

1. Motivation

The real-life Employee Timetabling Problem (ETP) presented in (Castiñeiras and Sáenz-Pérez 2011) and the Golomb Ruler Problem (P. Galinier and Pesant 2007) are two suitable problems to compare the modeling and solving capabilities of TOY(FDi) and ILOG Solver. Whereas to model the Golomb problem we have followed the formulation presented in Gecode (Schulte et al. 2010), to model the ETP we have departed from (Castiñeiras and Sáenz-Pérez 2011) and we have parameterized the formulation w.r.t. the structure an instance can take: The number of teams, the number of regular workers per working team, the extra worker rest constraint, the payment factor for the extra hours the extra worker does, and the different kind of working days.

These two problems have several things in common: (1) They are optimization problems: Whereas the ETP minimizes the extra hour payment, the Golomb problem minimizes the total sum of the rulers. (2) They are scalable problems (in the number of days to be scheduled and in the number of rulers, resp.) (3) As the size of their instances scales, almost the 100% of the total solving time is spent in solving their labelings, and thus the time spent in the rest of the tasks (constraint posting, incremental constraint propagation, etc.) becomes negligible. In the case of TOY(FDi), this tasks also include the system inherent surcharges arisen from lazy narrowing and the communication with its external solver ILOG Solver.

---

* This work has been partially supported by the Spanish projects TIN2008-06622-C03-01, UCM-BSCH-GR35/10-A-910502, and S2009TIC-1465

In this context, one would expect TOY(FDi) to match the solving time obtained when using directly ILOG Solver, as both systems are using the same formulation of the problem, the same algorithm to solve it, and the same constraint solving library. Regarding the labeling primitive (which is taking almost the 100% of the solving time) both systems are using the same API function, acting over the same variable set, and following the same search strategy. However, the solving of different ETP and Golomb instances reveal that the labeling primitive spends a different amount of time when being used in TOY(FDi) or in ILOG. Even more, no general rules can be inferred, as depending on the concrete instance being solved, the labeling primitive can be faster in either TOY(FDi) or ILOG.

Table 1 presents the results obtained for solving the labeling primitive in an instance of the ETP (scheduling 21 days) and in an instance of the Golomb problem (with 11 rulers.) In the case of Golomb, as there is only one labeling in the model, it is obviously the one being considered. In the case of the ETP, for each feasible team assignment there is one labeling per working team to be scheduled. The results of the instance presented belong to the first feasible team assignment and the labeling of the second working team. Regarding the table, the first two rows represent the results obtained when solving the ETP instance in ILOG and TOY(FDi), resp. The third row represents the speed-up of such these results. Next three rows represent the same information, but for the Golomb instance. All the results are obtained via the solver API function *printInformation()*. The third column represents the running time of the solver (measured in seconds) for solving the labeling primitive. The fourth and fifth columns represent the number of FD variables and constraints posted to the solver at the moment of dealing with the labeling primitive, resp. The sixth and seventh columns represent the number of fails and choice points produced by the solver for finding the optimal solution, resp. As both ETP and Golomb are optimization problems, those fails and choice points are obtained after exploring the whole search tree.

Instance	System	Time	Vars.	Cons.	Back.	Choice
ETP-21	ILOG	43.98	168	120	1,938,432	1,938,432
	TOY(FDi)	50.78	145	175	2,139,003	2,139,003
	Speed-Up ILOG/TOY	0.87	1.16	0.69	0.91	0.91
Golomb-11	ILOG	54.01	66	68	1,484,086	1,484,102
	TOY(FDi)	51.12	55	121	1,484,086	1,484,102
	Speed-Up ILOG/TOY	1.06	1.20	0.56	1.00	1.00

Table 1. Analysis of the primitive labeling for an ETP and a Golomb instance

The results of Table 1 reveal that, for each instance, the labeling primitive spends a different amount of time when being used in TOY(FDi) or in ILOG. Even more, no general rules on the performance of the systems can be inferred, as the labelings of both instances are solved in the order of magnitude of seconds (between 40 and 55 seconds), and whereas ILOG is faster for the ETP labeling, TOY(FDi) is faster for the Golomb one. The following observations can be drawn:

1. For each instance, there is a deviation between the number of FD variables and constraints posted in TOY(FDi) and in ILOG. Whereas ILOG has more variables for both instances,

2. The number of variables and constraints is the key concept affecting the performance of the labeling. By looking to the ETP instance one might think that the less a system fails the best performance it gets, but this does not hold in the Golomb instance, where TOY(FDi) is faster even for the same number of failures and choice points as ILOG. Also, we have ensured that both TOY(FDi) and ILOG maintain the same propagation consistency for each kind of constraint (as the API of ILOG allows to configure it).

The contribution of this technical report is to present a trace of the execution in TOY(FDi) and ILOG of an ETP instance with 7 days to be scheduled and of a Golomb instance with 4 rulers. The idea of selecting such small-sized instances is that they are posting less FD variables and constraints, and thus it is easier to identify all of the them. For each instance and system, the report presents the different function calls performed and the amount of FD variables and constraints associated to each of them (analyzing where do each single variable and constraint come from). As both systems are using the same formulation, algorithm and constraint solver, it is possible to identify the points of the modeling process making the deviation between the number of variables and constraints posted in TOY(FDi) and ILOG. For each of these points, additional remarks are provided, explaining the reasons of such deviation.

The structure of this technical report is the following: First three sections deal with the ETP instance. Section 2 presents the instance to be analyzed, its input parameters and a general overview of its obtained solution and the solving process to achieve it. Sections 3 and 4 present the trace in TOY(FDi) and ILOG (resp.) Next three sections follow the same scheme but for the Golomb instance. Finally, Section 8 presents the obtained conclusions.

2. ETP Instance to be analyzed

The analyzed ETP instance has 7 days. Due to the absences of their team workers, it contains only two feasible team assignments (corresponding to its two solutions). Stage I (*team_assign*) computes each feasible team assignment. This report only analyzes *the first team assignment (first solution)*. Departing from each feasible team assignment, stage II (*tt_split*) creates *tt* and splits it into as many independent sub-problems as working teams there are. Once *tt* is splitted, stage III (*tt_solve*) solves the different sub-problems (*tt[i]*) sequentially. This report only analyzes *the solving of the first sub-problem tt[0]*. In summary, this analysis focuses on the FD variables and constraints of:

- First stage *team_assign* (first solution).
- Second stage *tt_split*.
- Third stage *tt_solve* (only for the first working team *tt[0]*).

Input arguments of the instance

- nd = 7
- nt = 3
- ntw = 4
- er = 3
- ef = 2
- ws = [[20,22,24],[24,24]]
- abs = [(1,1),(2,1),(5,1),(6,1),(7,1),(10,1),(11,1),(12,1),(5,6),(6,6),(7,6),(10,6),(11,6),(12,6)]
- dc = [1,1,1,1,1,2,2]
- T = 1
- P = true
- W = calByWorkers
- SS = fstUnbound

Output result for the instance

sol.1 ≡ Schedule = (Timetabling, EHours)
- Timetabling = [[0, 0, 22, 24, 0, 0, 0, 0, 0, 0, 0, 0, 20],
  [0, 0, 0, 0, 0, 20, 22, 24, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 20, 22, 24, 0],
  [0, 20, 24, 22, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 24, 22, 20, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 24],
  [24, 24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
- EHours = 133

Command for solving the instance

p_tt 7 3 4 3 2 [[20,22,24],[24,24]]
[(1,1),(2,1),(5,1),(6,1),(7,1),(10,1),(11,1),(12,1),(5,6),(6,6),(7,6),(10,6),(11,6),(12,6)] [1,1,1,1,1,2,2]
1 true calByWorkers fstUnbound == Schedule

3. TOY(FDi) trace of the ETP instance

This section describes the FD variables and constraints that TOY(FDi) creates for solving the ETP instance. For the sake of readability, the names of the functions are maintained as they are in the TOY(FD) program, but an identifier is also added as a prefix to univocally distinguish each function call to be analyzed. The IloSolver API method *printInformation()* is used to obtain the number of

FD variables and constraints posted to the solver. Computing the amount of FD variables and constraints per function is simply done by using this method before and after the function call.

3.1. *team_assign*

3.1.1. Function calls

15 - team_assignment 7 3 4 3 [[20,22,24],[24,24]]
[(1,1),(2,1),(5,1),(6,1),(7,1),(10,1),(11,1),(12,1),(5,6),(6,6),(7,6),(10,6),(11,6),(12,6)]
[1,1,1,1,1,2,2] == (D, E, OAbs)

3 - gen_d 7 3 == D

8 - gen_i 7 4 [3,2] [1,1,1,1,1,2,2] [D1,D2,D3,D4,D5,D6,D7]
[[2,3,3],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,3,3],[0,0,0]] == true

14 - gen_e 7 3 [D1,D2,D3,D4,D5,D6,D7] [[1,1,1],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,1,1],[0,0,0]] == E

Where the obtained result is:
D = [1,2,3,1,2,3,1]
E = [1,0,0,0,0,1,0]
OAbs = [[1,2,5,6,7,10,11,12],[],[],[],[],[5,6,7,10,11,12],[]]

3.1.2. Table

Id	FUNCTION	VARIABLES	CONSTRAINTS
1	** **domain**	**3**	**3**
2	** **all_different**	**0**	**1**
3	* **gen_d**	**3**	**4**
4	*** domain	7	7
5	** **max_absences**	**7**	**7**
6	*** post_implications	0	15
7	** **d_implications**	**0**	**15**
8	***gen_i**	**7**	**22**
9	** **domain**	**7**	**7**
10	*** sum	0	5
11	** **extra_worker_rest**	**0**	**5**
12	*** post_implications	0	6
13	** **d_implications**	**0**	**6**
14	***gen_e**	**7**	**18**
15	**team_assign**	**17**	**44**

3.1.3 Remarks

- Id. 7. One might expect the posting of 21 constraints, instead of 15. However, as propagation is set to true (*incremental* mode), the domains of D and A can be pruned as the constraints are posted. For example, when D1 is bound to 1, so it does D4 and D7, and the value 1 is pruned of the domain of D2, D3, D5 and D6. Thus, there is no equivalence between each *post_implication* call and the posting of a single constraint. Examples:
* *post_implication D1 (#=) 1 A1 (#=) 2* One constraint is posted.
* *post_implication D1 (#=) 2 A1 (#=) 3* Two constraints are posted (as the domain of A1 is 0..2, the solver also posts D1 /= 2).
* *post_implication D1 (#=) 3 A1 (#=) 3* Two constraints are posted (as the domain of A1 is 0..2, the solver also posts D1 /= 3). Thus, D1 is bound to 1 (also D4 and D7), and value 1 is pruned from D2, D3, D5 and D6.
* post_implication D2 (#=) 1 A2 (#=) 0 No constraints are posted (as the domain of D2 is 2..3, the expression is trivially entailed).

3.2. *tt_split*

3.2.1. Function calls

17 - timetabling_splitting 7 3 4 [[20,22,24],[24,24]]
[[1,2,5,6,7,10,11,12],[],[],[],[],[5,6,7,10,11,12],[]] [1,1,1,1,1,2,2] [1,2,3,1,2,3,1] [1,0,0,0,0,1,0]
== (Table, TT, DC, H)
16 - (#/ 12) (foldl (std_working_hours [[20,22,24],[24,24]]) 0 [1,1,1,1,1,2,2]) == H

Where the obtained result is:
Table = [[0,0,A,B,C],[D,E,F,G,0],[H,I,J,K,0],[L,M,N,O,0],[P,Q,R,S,0],[T,0,0,0,U],[V,W,X,Y,0]]
TT = [[[0,0,A,B,C],[L,M,N,O,0],[V,W,X,Y,0]],[[D,E,F,G,0],[P,Q,R,S,0]],[[H,I,J,K,0],[T,0,0,0,U]]]
DC = [[1,1,2],[1,1],[1,2]]
H = 35

3.2.2. Table

Id	FUNCTION	VARIABLES	CONSTRAINTS
16	*** #/**	**1**	**1**
17	**tt_split**	**1**	**1**

3.2.3. Remarks

No remarks.

3.3. *tt_solve*

3.3.1. Function calls

64 - timetabling_solving 4 [[20,22,24],[24,24]] 2 1 35 calByWorkers fstUnbound
[[0,0,A,B,C],[L,M,N,O,0],[V,W,X,Y,0]] [1,1,2] == EH
27 - zipWith (post_working_slots 4 [[20,22,24],[24,24]]) [[0,0,A,B,C],[L,M,N,O,0],[V,W,X,Y,0]]
[1,1,2] == [true, true, true]
50 - map (tight_slot 1 4 [[0,L,V],[0,M,W],[A,N,X],[B,O,Y],[C,0,0]]) [0,20,22,24] ==
[true,true,true,true]
63 - compute_extra_hours 4 35 2 [[0,L,V],[0,M,W],[A,N,X],[B,O,Y],[C,0,0]] == EH

20 - post_working_slots 4 [[20,22,24],[24,24]] [0,0,A,B,C] 1 == true
23 - post_working_slots 4 [[20,22,24],[24,24]] [L,M,N,O,0] 1 == true
26 - post_working_slots 4 [[20,22,24],[24,24]] [V,W,X,Y,0] 2 == true
18 - domain_valArray [0,0,A,B,C] [0,20,22,24],
19 - distribute [2,1,1,1] [0,20,22,24] [0,0,A,B,C]

21 - domain_valArray [L,M,N,O,0] [0,20,22,24],
22 - distribute [2,1,1,1] [0,20,22,24] [L,M,N,O,0]

24 - domain_valArray [V,W,X,Y,0] [0,24],
25 - distribute [3,2] [0,24] [V,W,X,Y,0]
46 - tight_slot 1 4 [[0,L,V],[0,M,W],[A,N,X],[B,O,Y],[C,0,0]] 0 == true
47 - tight_slot 1 4 [[0,L,V],[0,M,W],[A,N,X],[B,O,Y],[C,0,0]] 20 == true
48 - tight_slot 1 4 [[0,L,V],[0,M,W],[A,N,X],[B,O,Y],[C,0,0]] 22 == true
49 - tight_slot 1 4 [[0,L,V],[0,M,W],[A,N,X],[B,O,Y],[C,0,0]] 24 == true

36 - zipWith (post_count_constraint 0) [0,L,V],[0,M,W],[A,N,X],[B,O,Y] [F1,F2,F3,F4] ==
[true, true, true, true]
39 - post_tight 1 [F1,F2,F3,F4]
42 - post_tight 1 [F2,F3,F4]
45 - post_tight 1 [F3,F4]

29 - post_count_constraint 0 [0,L,V] F1
31 - post_count_constraint 0 [0,M,W] F2
33 - post_count_constraint 0 [A,N,X] F3
35 - post_count_constraint 0 [B,O,Y] F4

28 - count 0 [0,L,V] (#=) F1
30 - count 0 [0,M,W] (#=) F2
32 - count 0 [A,N,X] (#=) F3
34 - count 0 [B,O,Y] (#=) F4

37 - map (F1#- [F2,F3,F4]) == L1,
38 - domain L1 (-1) 1

40 - map (F2#- [F3,F4]) == L2,
41 - domain L2 (-1) 1

43 - map (F3#- [F4]) == L3,
44- domain L3 (-1) 1

59 - map (compute_worker_extra_hours 35) [[0,L,V],[0,M,W],[A,N,X],[B,O,Y]] ==
[EH1, EH2, EH3, EH4]

60 - sum [EH1, EH2, EH3, EH4] (#=) TeamWorkersExtraHours

61 - sum [C,0,0] (#=) ExtraWorkerExtraHours

62 - EHours == TeamWorkersExtraHours #+ (2 #* ExtraWorkerExtraHours)

55 - compute_worker_extra_hours 35 [0,L,V] == EH1
56 - compute_worker_extra_hours 35 [0,M,W] == EH2
57 - compute_worker_extra_hours 35 [A,N,X] == EH3
58 - compute_worker_extra_hours 35 [B,O,Y] == EH4

51 - sum [0,L,V] (#=) HoW,
52 - HoW #- 35 == EWL,
53 - post_implication EWL (#>=) 0 EH1 (#=) EWL,
54 - post_implication EWL (#<) 0 EH1 (#=) 0

Where the obtained result is:
A = 22
B = 24
C = 20
L = 0
M = 20
N = 24
O = 22
V = 24
W = 24
X = 0
Y = 0
EH = 71

3.3.2. Table

Id	FUNCTION	VARIABLES	CONSTRAINTS
18	*** domain_valArray	3	3
19	*** distribute	7	2
20	**** post_working_slots**	**10**	**5**
21	*** domain_valArray	4	4
22	*** distribute	6	2
23	**** post_working_slots**	**10**	**6**
24	*** domain_valArray	4	4
25	*** distribute	4	2
26	**** post_working_slots**	**8**	**6**
27	***** zip_with post_working_slots**	**28**	**17**
28	***** *count*	*3*	*3*
29	***** *post_count_constraint*	*3*	*3*
30	***** *count*	*3*	*3*
31	***** *post_count_constraint*	*3*	*3*
32	***** *count*	*2*	*3*
33	***** *post_count_constraint*	*2*	*3*
34	***** *count*	*2*	*3*
35	***** *post_count_constraint*	*2*	*3*
36	**** zip_with post_count_const**	**10**	**12**
37	***** *map*	*3*	*3*
38	***** *domain*	*0*	*3*
39	**** post_tight**	**3**	**6**
40	***** *map*	*2*	*2*
41	***** *domain*	*0*	*2*
42	**** post_tight**	**2**	**4**
43	***** *map*	*1*	*1*
44	***** *domain*	*0*	*1*
45	**** post_tight**	**1**	**2**
46	**** tight_slot**	**16**	**24**
47	**…. ** tight_slot**	**16**	**24**
48	**… ** tight_slot**	**16**	**24**
49	**… ** tight_slot**	**16**	**24**
50	***** zip_with tight_slot**	**64**	**96**
51	***** *sum*	*2*	*2*
52	***** *#-*	*1*	*2*
53	***** *post_implication*	*1*	*2*
54	***** *post_implication*	*0*	*1*
55	****compute_worker_extra_hours**	**4**	**7**
56	**… ****compute_worker_extra_hours**	**4**	**7**
57	**… ****compute_worker_extra_hours**	**3**	**7**

58	**… ***compute_worker_extra_hours**	**3**	**7**
59	** **map**	**14**	**28**
60	** **sum**	**1**	**2**
61	** **sum**	**3**	**2**
62	**#+ (#*)	**2**	**4**
63	***compute_extra_hours**	**20**	**36**
64	tt_solve	**112**	**149**

3.3.3. Remarks

- Id 19. Distribute creates:

* One variable per element of cards (with only one value domain, the one of cards[i]).
* One variable per zero of vars (with only one value domain, 0).
* One extra variable attached to the IloSolver kernel.
* Two constraints due to the IloDistribute posting.

- Id 28. Count creates:

* One variable for the fresh_var.
* One variable with only one value domain (the number of hours).
* One variable per each zero of the vars of the day.
* One constraint to set the domain of the fresh var to IloIntMin, IloIntMax (inherent to the creation of each new TOY(FDi) variable).
* Two constraints for the count L List Op R. One for making Val = #L in List. One for making Val Op R.

- Id 38. Domain does not create new variables, as they are previously created by the functional notation.

- Id 47, 48, 49. We have skipped a detailed explanation for the three other computations of tight_slot, as they are exactly the same as the one we have already described from 28 to 46.

- Id 51. sum List Op R creates:

* One variable for R
* One variable for each zero var of List
* One variable Val for sum List == Val
* One constraint for R
* One constraint for Val
* One constraint for Val Op R

- Id 53 and 54. In Id 53 one variable for EH. In Id 54 EH the variable is already created.

4. ILOG trace of the ETP instance

This section describes the FD variables and constraints that ILOG creates for solving the ETP instance. For the sake of readability, we maintain the names of the functions as they are in the ILOG program, but we also add an identifier as a prefix to univocally distinguish each function call to be analyzed. The IloSolver API method *printInformation()* is used to obtain the number of FD variables and constraints posted to the solver. Computing the amount of FD variables and constraints per function is simply done by using this method before and after the function call.

4.1. *team_assign*

4.1.1. Function calls

15 - team_assignment(env, model, solver, true, 7, 3, 4, 3, <<20,22,24>,<24,24>>, <<1,1>,<2,1>,<5,1>,<6,1>,<7,1>,<10,1>,<11,1>,<12,1>,<5,6>,<6,6>,<7,6>,<10,6>,<11,6>,<12,6>>, <1,1,1,1,1,2,2>, d, e, o_abs);

3 - gen_d(env, model, solver, true, 7, 3, d);
8 - gen_i(env, model, solver, true, 7, 3, 4, <1,1,1,1,1,2,2>, d, <3,2>,
<<2,3,3>,<0,0,0>,<0,0,0>,<0,0,0>,<0,0,0>,<0,3,3>,<0,0,0>>);

14 - gen_e(env, model, solver, true, 7, 3, 3, d, e,
<<1,1,1>,<0,0,0>,<0,0,0>,<0,0,0>,<0,0,0>,<0,1,1>,<0,0,0>>);

Where the obtained result is:
d = <1,2,3,1,2,3,1>
e = <1,0,0,0,0,1,0>
o_abs = <<1,2,5,6,7,10,11,12>,<>,<>, <>, <>, <5,6,7,10,11,12>,<>>

4.1.2. Table

FUNCTION	ILOG VARIABLES	ILOG CONSTRAINTS	TOY(FDi) VARIABLES	TOY(FDi) CONSTRAINTS
gen_d	4 (+1)	2 (-2)	3	4
gen_i	7 (+0)	21 (-1)	7	22
gen_e	12 (+5)	26 (+8)	7	18
team_assign	**23 (+6)**	**49 (+5)**	**17**	**44**

4.1.3. Remarks

Explanations are provided (in italics) for each deviation on the amount of FD variables and constraints that TOY(FDi) and ILOG need.

gen_d

- **TOY(FDi).** First, 3 variables and their 3 associated daemon constraints are needed for representing [D1,D2,D3]. Second, one constraint for all_different.
- **ILOG.** In this case, the IloIntVarArray d is created in the main function, with the constructor *IloIntVarArray d(env, nd, 1, nt);* The array is initially associated to env, and thus the d variables are no going to be taken into account by solver until explicit constraints are posted over them on model. As the instance contains seven days, the following constraints will be posted:
  * d[0] == d[3] (leading to 1 variable and 0 constraints)
  * d[0] == d[6] (leading to 1 variable and 1 constraint)
  * d[1] == d[4] (leading to 1 variable and 0 constraints)
  * d[2] == d[5] (leading to 1 variable and 0 constraints)

- *Explanation to this deviation between TOY(FDi) and ILOG. Neither d[0] and d[3] belong to* model *before posting the constraint d[0] == d[3]. In this context, we can say they are created in the environment* env*, but not associated to* model*. The posting of this constraint is interpreted by* model *and* solver *as the creation of a new variable v1, to which both d[0] and d[3] point to. Further, when d[0] == d[6] is going to be posted, v2 is created, but a constraint must relate v1 and v2 in solver. That is why we have 4 variables and 1 constraint. The second constraint of gen_d belongs to the all_different constraint between d[0], d[1] and d[2].*

gen_i

- **TOY(FDi).** First, creating the variable list A implies 7 variables and their 7 associated daemon constraints. Second, the implications between D and A imply 0 variables and 15 constraints (not 21, as it was explained in Section 3).
- **ILOG.** First, creating the IloIntVarArray a in env implies no variables and constraints. Then, pruning the upper bound of each a[i] makes the creation of a new v[i], to which a[i] points to. Thus, solver creates 7 new variables but 0 constraints. Second, implication implies 0 variables but the posting of the 21 constraints.
- *Explanation to this deviation between TOY(FDi) and ILOG. ILOG always deals with decision variables. However, in TOY(FDi) the internal implementation of post_implication must deal with logical variables (associated to unbound decision variables) and integers (associated to bound decision variables). Thus, the number of constraints TOY(FDi) post to solver depend of the concrete arguments of the post_implication call.*

gen_e

- **TOY(FDi).** First, creating the variable list E supposes 7 variables and their 7 associated daemon constraints. Second, the posting of the sum constraints suppose 0 variables and 5 constraints. Third, the implications between D and E suppose 0 variables and 6 constraints (not 21, as it was explained in Section 3).
- **ILOG.** First, creating the IloIntVarArrya e in env implies no variables and constraints. Second, the posting of the sum constraints supposes 5 constraints but 12 new variables. Third, the implications between D and E suppose 0 variables 21 constraints.

- *Explanation to this deviation between TOY(FDi) and ILOG. Seven variables belong to* e *(created as soon each* e_i *is related in a sum constraint). The other five variables are created for each sum constraint* v ← sum(e_i, e_{i+1}, e_{i+2}). Also, a constraint pruning v to be lower or equal to 1 is posted.

4.2. *tt_split*

4.2.1. Function calls

17 - hours = timetabling_splitting(7, 3, 4, <<20,22,24>,<24,24>>, <<1,2,5,6,7,10,11,12>,<>,<>, <>, <>, <5,6,7,10,11,12>,<>>, <1,1,1,1,1,2,2>, true, <1,2,3,1,2,3,1>, <1,0,0,0,0,1,0>, env, solver, mdl, slv, tt, dC);

16 - hours = gen_hours(3, 4, <<20,22,24>,<24,24>>, <1,1,1,1,1,2,2>);

4.2.2. Table

FUNCTION	ILOG VARIABLES	ILOG CONSTRAINTS	TOY(FDi) VARIABLES	TOY(FDi) CONSTRAINTS
put_zeros	10 [4,2,4] (+10)	0 [0,0,0] (+0)	0	0
#/	0 (-1)	0 (-1)	1	1
tt_split	**10 (+9)**	**0 (-1)**	**1**	**1**

4.2.3. Remarks

Explanations are provided (in italics) for each deviation on the amount of FD variables and constraints that TOY(FDi) and ILOG need.

put_zeros

- **TOY(FDi).** The tasks of creating tt and splitting it into the different working teams are managed by the H solver. Thus, no FD variables and constraints are created.
- **ILOG.** One might expect a new FD variable for each element of tt. However, the variables are created under env, and they are not associated to model until a constraint is posted on them. *The 10 variables thus represent those ones that are bound to zero. However, no constraints are needed, in a similar situation to the one described in the function gen_d of Section 4.1.3).*

4.3. *tt_solve*

We are going to consider the solving of the first team tt[0], which initially is:
<<0,0,A,B,C>, <L,M,N,O,0>,<V,W,X,Y,0>>

4.3.1. Function calls

27 - zipWith_post_working_slots(env, model, solver, <<20,22,24>,<,3,2>>,

<<0,0,A,B,C>,<L,M,N,O,0>,<V,W,X,Y,0>>, <1,1,2>, true, 4);

- transpose(4, *<<0,0,A,B,C>,<L,M,N,O,0>,<V,W,X,Y,0>>*,
<<0,L,V>,<0,M,W>,<A,N,X>,<B,O,Y>,<C,0,0>>, env, model, solver, true);

46 - tight_slot (env, model, solver, 1, 4, <<0,L,V>,<0,M,W>,<A,N,X>,<B,O,Y>,<C,0,0>>, 0, 3, true);
47 - tight_slot (env, model, solver, 1, 4, <<0,L,V>,<0,M,W>,<A,N,X>,<B,O,Y>,<C,0,0>>, 20, 3, true);
48 - tight_slot (env, model, solver, 1, 4, <<0,L,V>,<0,M,W>,<A,N,X>,<B,O,Y>,<C,0,0>>, 22, 3, true);
49 - tight_slot (env, model, solver, 1, 4, <<0,L,V>,<0,M,W>,<A,N,X>,<B,O,Y>,<C,0,0>>, 24, 3, true);

63 - compute_extra_hours(env, model, solver, true, 4, 35, 2,
<<0,L,V>,<0,M,W>,<A,N,X>,<B,O,Y>,<C,0,0>>, extra_hours);

4.3.2. Table

Id	FUNCTION	ILOG VARIABLES	ILOG CONSTRAINTS	TOY(FDi) VARIABLES	TOY(FDi) CONSTRAINTS
18	*domain_valArray	3 (+0)	0 (-3)	3	3
19	*distribute	5 (-2)	2 (+0)	7	2
20	**post_working_slots**	**8 (-2)**	**2 (-3)**	**10**	**5**
21	*domain_valArray	4 (+0)	0 (-4)	4	4
22	*distribute	5 (-1)	2 (+0)	6	2
23	**post_working_slots**	**9 (-1)**	**2 (-4)**	**10**	**6**
24	*domain_valArray	4 (+0)	0 (-4)	4	4
25	*distribute	3 (-1)	2 (+0)	4	2
26	**post_working_slots**	**7 (-1)**	**2 (-4)**	**8**	**6**
27	**zip_with post_working_slots**	**24 (-4)**	**6 (-11)**	**28**	**17**
	transpose	**15 (+15)**	**15 (+15)**	**0**	**0**
36	** zip_with post_count_const	8	8	10	12
45	** post_tight	6	6	6	12
46	* tight_slot	14 (-2)	14 (-10)	16	24
47	…. * tight_slot	14 (-2)	14 (-10)	16	24
48	… * tight_slot	14 (-2)	14 (-10)	16	24
49	… * tight_slot	14 (-2)	14 (-10)	16	24
50	zip_with tight_slot	56 (-8)	56 (-40)	64	96
55	*compute_worker_extra_hours	2 (-2)	3 (-4)	4	7
56	*compute_worker_extra_hours	2 (-2)	3 (-4)	4	7
57	*compute_worker_extra_hours	2 (-1)	3 (-4)	3	7

58	*compute_worker_extra_hours	2 (-1)	3 (-4)	3	7
59	**compute_extra_hours**	**8 (-6)**	**12 (-16)**	**14**	**28**
60	* sum	1 (+0)	1 (-1)	1	2
61	* sum	1 (-2)	1 (-1)	3	2
62	* #+ (#*)	1 (-1)	1 (-3)	2	4
63	*** compute_extra_hours**	**11 (-9)**	**15 (-21)**	**20**	**36**
64	tt_solve	**106**	**92**	112	149

4.3.3. Remarks

zip_with post_working_slots

The total amount of FD variables and constraints (Id 27) comes from the sum of the variables and constraints of each of the three days (Id 20, Id 23 and Id 26, resp.) For example, the amount of FD variables and constraints for the first day <0,0,A,B,C> comes from posting the initial domain (Id 18) and the distribute constraint (Id 19).

- **Id 18.**
  - **TOY(FDi).** Each variable initialization implies a new variable and its associated daemon constraint.
  - **ILOG.** It is only created a new variable, but no constraint (see gen_d in Section 4.1.3).
- **Id 19.**
  - **TOY(FDi).** It is created: (i) One variable per each element of the array cards. (ii) One dummy variable per each variable of the day previously bound to zero. (In our case, the first two variables of <0,0,A,B,C>. (iii) The IloDistribute makes the solver create one extra variable. (iv) The constraint IloDistribute makes the solver create two constraints.
  - **ILOG.** It saves the variables of (ii).

transpose

- **TOY(FDi).** The execution of transpose [[0,0,A,B,C], [L,M,N,O,0],[V,W,X,Y,0]] == *[[0,L,V],[0,M,W],[A,N,X],[B,O,Y],[C,0,0]]* is entirely managed by the H solver, saving the creation of the 12 new FD variables and constraints.
- **ILOG.** A new IloIntVarArray must be created for each worker, containing as many variables as working days there are. Also, a constraint relating each new variable with its mate variable in tt must be posted. This supposes the creation of 15 new FD variables and constraints.

tight_slot

The total amount of FD variables and constraints (Id 46) comes from the sum of the variables and constraints of Id 36 and Id 45.

- **Id 36.**
  - **TOY(FDi).** There are four count constraints, of type count 0 [0,L,V] == F. For each of them it is created: (i) One variable for F. (ii) One daemon constraint

- **ILOG.** It saves the variables of (iii) and the constraints of (ii).

- **Id 45.**
  - **TOY(FDi).** There are six #- constraints, subtracting each pair of [F1,F2,F3,F4]. Thus, it is created: (i) Six variable for the subtractions. (ii) Six daemon constraints associated to the new variables. (iii) Six constraints for the subtactions.
  - **ILOG.** It saves the constraints of (ii).

compute_extra_hours

The total amount of FD variables and constraints (Id 63) comes from the amount of extra hours of each worker (Id 55, Id 56, Id 57 and Id 58, resp.) plus the extra hours of the extra worker (Id 61). The sum of both relating them to the extra factor is done in 63.

- **Id 55.**
  - **TOY(FDi).** It is created: (i) One dummy variable per each variable of the sum previously bound to zero. (ii) One variable for HoW. (iii) One variable for EHL. (iv) One variable for EHours. (v) Three daemon constraints for HoW, EHL and EHours. (vi) One constraint relating the sum to HoW. (vii) One constraint for the subtraction. (viii) Two constraints for the two post_implications.
  - **ILOG.** It saves the variable of (i). It saves the variable HoW (ii) by using a complex expression relating the IloSum(…) – 35 == EHL (this kind of expressions are not supported in TOY(FDi)). Also, it saves the constraint of (vi). Finally, it saves the three daemon constraints of (v).

5. Golomb Instance to be analyzed

The analyzed Golomb instance has 4 rulers and follows incremental propagation. As this problem is simpler than the ETP, this report analyzes the whole execution.

Input arguments of the instance

- N = 4
- P = true

Output result for the instance

sol.1 ≡ M = [0, 1, 4, 6]

Command for solving the instance

golomb 4 true == M

6. TOY(FDi) trace of the Golomb instance

This section describes the FD variables and constraints that TOY(FDi) creates for solving the Golomb instance. For the sake of readability, the names of the functions are maintained as they are in the TOY(FD) program, but an identifier is also added as a prefix to univocally distinguish each function call to be analyzed. The IloSolver API method *printInformation()* is used to obtain the number of FD variables and constraints posted to the solver. Computing the amount of FD variables and constraints per function is simply done by using this method before and after the function call.

6.1. Function calls

1. M == [0 | take 3 gen_v_list] (M ≡ [0, A, B, C])

2. domain M 0 7

3. 0 #< A
4. A #< B
5. B #< C
6. order M

7. A #- 0 == K1
8. B #- 0 == K2
9. C #- 0 == K3
11. K1 #>= 1
12. K2 #>= 3
13. K3 #>= 6

15. B #- A == K4
16. C #- A == K5
18. K4 #>= 1
19. K5 #>= 3

21. C #- B == K6
23. K6 #>= 1

10 - map (#- 0) [A,B,C] == DM1 (DM1 ≡ [K1, K2, K3])
14 - lb DM1 1 2

17. map (#- A) [B,C] == DM2 (DM2 ≡ [K4, K5])
20. lb DM2 1 2

22. map (#- B) [C] == DM3 (DM3 ≡ [K6])
24. lb DM3 1 2

25. gen_dif M == D (gen_dif [0, A, B, C] == [K1, K2, K3, K4, K5, K6])

26. all_different D (all_different [K1, K2, K3, K4, K5, K6])

27. (head D) #< (last D) (K1 #< K6)

28. golomb 4 true == M (golomb 4 true == [0, A, B, C]

Where the obtained result is:
M = [0, 1, 4, 6]
K = [1, 4, 6, 3, 5, 2]

6.2. Table

Id	FUNCTION	VARIABLES	CONSTRAINTS
1	*** ==**	**0**	**0**
2	*** domain**	**3**	**3**
3	** #<	0	0
4	** #<	0	1
5	** #<	0	1
6	*** order**	**0**	**2**
7	*** #-	0	1
8	*** #-	0	1
9	*** #-	0	1
10	**** map (#-)**	**0**	**3**
11	*** #>=	0	0
12	*** #>=	0	0
13	*** #>=	0	0
14	**** lb**	**0**	**0**
15	** #-	1	2
16	** #-	1	2
17	**** map (#-)**	**2**	**4**
18	*** #>=	0	0
19	*** #>=	0	0
20	**** lb**	**0**	**0**
21	** #-	1	2
22	**** map (#-)**	**1**	**2**
23	*** #>=	0	0
24	**** lb**	**0**	**0**
25	*** gen_diff**	**3**	**9**
26	*** all_different**	**0**	**1**
27	*** #<**	**0**	**1**
28	**golomb**	**6**	**16**

6.3. Remarks

- Id 1. The array M = [0, A, B, C] is created, with A, B and C new fresh variables.
- Id 2. An initial FD domain 0…7 is set to A, B and C. A daemon constraint is posted on each of them, and thus the solver takes the three vars into account, as well as their three associated daemon constraints.
- Id 6. The order function only considers two constraints, as the constraint 0 #< A is considered by the solver as a trivial pruning of the domain of the A variable, and thus it is not considered as a proper constraint to be taken into account.
- Id 10. The constraints A #- 0 == K1, B #- 0 == K2 and C #- 0 == K3 are considering three constraints but no new variables, as K1, K2 and K3 are directly pointed to A, B and C, resp.
- Id 17. In contrast, the constraints B #- A == K4 is considering the new variable K4, and two constraints, the subtraction constraints and the daemon constraint associated to the new variable K4. The same holds for C #- A == K5.
- Id 14. The constraint K1 #> 1 is considered as trivial prunning of the variable lower bound, and thus it is not considered as a proper constraint to be taken into account. The same holds for K2 #>= 3 and K3 #>= 6, and thus *lb* implies the posting of no new variables or constraints.

7. ILOG trace of the Golomb instance

This section describes the FD variables and constraints that ILOG creates for solving the Golomb instance. For the sake of readability, the names of the functions are maintained as they are in the TOY(FD) program, but an identifier is also added as a prefix to univocally distinguish each function call to be analyzed. The IloSolver API method *printInformation()* is used to obtain the number of FD variables and constraints posted to the solver. Computing the amount of FD variables and constraints per function is simply done by using this method before and after the function call.

7.1. Function calls

```
1 and 2.  IloIntVarArray m(env, 4, 0, 7);
          IloConstraint ct1 = m[0] == 0;
          IloIntVarArray k(env, 6, 1, 7);

3. IloConstraint ct = m[0] < m[1];
4. IloConstraint ct = m[1] < m[2];
5. IloConstraint ct = m[2] < m[3];
6. order m

7. IloConstraint ct = k[0] == m[1] - m[0];
8. IloConstraint ct = k[1] == m[2] - m[0];
9. IloConstraint ct = k[2] == m[3] - m[0];
11. IloConstraint ct = k[0] #>= 1
12. IloConstraint ct = k[1] #>= 3
```

13. IloConstraint ct = k[2] #>= 6

15. IloConstraint ct = k[3] == m[2] - m[1];
16. IloConstraint ct = k[4] == m[3] - m[1];
18. IloConstraint ct = k[3] #>= 1
19. IloConstraint ct = k[4] #>= 3

21. IloConstraint ct = k[5] == m[3] - m[2];
23. IloConstraint ct = k[5] #>= 1

10 - map (#- 0) [A,B,C] == DM1 (DM1 ≡ [K1, K2, K3])
14 - lb DM1 1 2

17. map (#- A) [B,C] == DM2 (DM2 ≡ [K4, K5])
20. lb DM2 1 2

22. map (#- B) [C] == DM3 (DM3 ≡ [K6])
24. lb DM3 1 2

25. gen_dif M == K (gen_dif [0, A, B, C] == [K1, K2, K3, K4, K5, K6])

26. IloConstraint ct = IloAllDiff(env, k); (all_different [K1, K2, K3, K4, K5, K6])

27. IloConstraint ct = k[0] < k[5]; (K1 #< K6)

28. golomb 4 true == M (golomb 4 true == [0, A, B, C]

Where the obtained result is:
M = <0, 1, 4, 6>
K = <1, 4, 6, 3, 5, 2>

7.2. Table

Id	FUNCTION	ILOG VARIABLES	ILOG CONSTRAINTS	TOY(FDi) VARIABLES	TOY(FDi) CONSTRAINTS
1	*** ==**	**0**	**0**	**0**	**0**
2	*** domain**	**1 (-2)**	**0 (-3)**	**3**	**3**
3	** #<	1 (+1)	1 (+1)	0	0
4	** #<	1 (+1)	1 (+0)	0	1
5	** #<	1 (+1)	1 (+0)	0	1
6	*** order**	**3 (+3)**	**3 (+1)**	**0**	**2**
7	*** #-	1 (+1)	1 (+0)	0	1
8	*** #-	1 (+1)	1 (+0)	0	1
9	*** #-	1 (+1)	1 (+0)	0	1

10	**** map (#-)**	**3 (+3)**	**3 (+0)**	**0**	**3**
11	*** #>=	0 (+0)	0 (+0)	0	0
12	*** #>=	0 (+0)	0 (+0)	0	0
13	*** #>=	0 (+0)	0 (+0)	0	0
14	**** lb**	**0 (+0)**	**0 (+0)**	**0**	**0**
15	** #-	1 (+0)	1 (-1)	1	2
16	** #-	1 (+0)	1 (-1)	1	2
17	**** map (#-)**	**2 (+0)**	**2 (-2)**	**2**	**4**
18	*** #>=	0 (+0)	0 (+0)	0	0
19	*** #>=	0 (+0)	0 (+0)	0	0
20	**** lb**	**0 (+0)**	**0 (+0)**	**0**	**0**
21	** #-	1 (+0)	1 (-1)	1	2
22	**** map (#-)**	**1 (+0)**	**1 (-1)**	**1**	**2**
23	*** #>=	0 (+0)	0 (+0)	0	0
24	**** lb**	**0 (+0)**	**0 (+0)**	**0**	**0**
25	*** gen_diff**	**6 (+3)**	**6 (-3)**	**3**	**9**
26	*** all_different**	**0 (+0)**	**1 (+0)**	**0**	**1**
27	*** #<**	**0 (+0)**	**1 (+0)**	**0**	**1**
28	**golomb**	**10 (+4)**	**12 (-4)**	**6**	**16**

7.3. Remarks

Explanations are provided for each deviation on the amount of FD variables and constraints that TOY(FDi) and ILOG need.

Id 2. domain

- **TOY(FDi).** First, 3 variables and their 3 associated daemon constraints are needed for representing A, B and C in the array M = [0, A, B, C].
- **ILOG.** In this case, the IloIntVarArray m is created with the constructor *IloIntVarArray m(env, 4, 0, 7);* The array is initially associated to env, not to model. Thus, the m variables are no going to be taken into account by solver until explicit constraints are posted over them on model. However the constraint m[0] == 0 leads to 1 variable and no constraints, as the constraint is considered by the solver as a trivial pruning of the domain of the m[0] variable, and thus it is not considered as a proper constraint to be taken into account.

Id 6. order

- **TOY(FDi).** Two constraints are considered, instead of three, as the first position of M was previously unified to 0 by the H solver.
- **ILOG.** The posting of new constraints involving m[1], m[2] and m[3] makes the solver take them now into account. Also, as m[0] is a decision variable, the constraint m[0] < m[1] is also taken into account by the solver.

Id 10. map (#- 0)

- **TOY(FDi).** Looking at the constraints A #- 0 == K1, the solver does not consider K1 as a

- **ILOG.** In the case of ILOG, as m[0] is still a decision variable, the constraint m[1] – m[0] == k[0] considers k[0] as a new variable.

Id 17. map (#- A)

- **TOY(FDi).** Looking at the constraints B #- A == K4, the solver considers: (i) K4 as a new variable, (ii) the subtraction constraint and (iii) the daemon constraint associated to K3.
- **ILOG.** In the case of ILOG, it is saving (iii).

8. Conclusions

TOY(FDi) and ILOG are solving the same instances of the ETP and Golomb problems, with the same formulation, algorithm and constraint library. They are using the same API function for the labeling primitive, with the same variable set and the same variable and value selection strategy. However, solving the labeling takes a different amount of time in TOY(FDi) and ILOG, and, depending on the concrete instance to be solved, either TOY(FDi) or ILOG is the fastest one. This technical report has proved that the internal implementation of TOY(FDi) (which interfaces a CP solver to a CFLP system and coordinates the solver from the system engine) requests a different amount of FD variables and constraints to model each instance w.r.t. the ones needed when modeling it directly in ILOG. As the labeling involves a different amount of variables and constraints, it makes sense that it requires a different computational effort for solving it.

On the one hand, this technical report has presented that there are some cases in which the TOY(FDi) implementation requests extra FD variables or constraints w.r.t. the ones required when modeling directly in ILOG:

- A daemon constraint is associated to the creation of each new FD variable. This daemon constraint is mandatory in the TOY(FDi) implementation, as it triggers the unification of the system mate logic variable associated to the solver FD decision variable. With these bindings, the TOY(FDi) solutions can output the results the FD solver has computed.
- The API of the ILOG constraints IloSum, IloDistribute and IloIfThen request an array of decision variables as argument, independently of how many of these variables are bound or unbound. However, this requirement requests the use of extra variables in the TOY(FDi) implementation. Let us consider the following TOY(FDi) goal: TOY(FDi) > domain [X, Y] 1 2, X #> 1, sum [X,Y] (#=) Z

  By using incremental propagation, when the goal reaches the last constraint, X is bound to 2, so the system evaluates *sum [2, Y] (#=) Z*. Whereas Y is used to find out its mate decision variable contained in the solver, we only know that there must be a decision variable bound to 2. But, unfortunately, we do not know which one is it. So, we must create an extra dummy variable with the single value domain 2, and use it to fulfil the requirement of the IloSum constraint of using only decision variables (and not integers) as arguments.
- The use of complex expressions are not supported in TOY(FDi), and thus extra variables are required to represent the same information. For example, the expression X #+ 2 #* Y requests first to evaluate 2 #* Y, generating an extra variable Z. Then, the sum X #+ Z is

On the other hand, this technical report has presented that there are some cases in which the TOY(FDi) implementation saves some FD variables and constraints w.r.t. the ones required when modeling directly in ILOG:

- The use of the H solver can save the creation of some FD variables, by unifying them before assigning an initial FD domain. In the ETP instance, *transpose* (see Section 4.3.3) allows to explore the variables of the team ordered by days or by workers. Whereas in TOY(FDi) *transpose* is managed on the H solver, in ILOG it is necessary to create new FD variables and link the two mate representations with a constraint.
- The internal management of IloIfThen in the TOY(FDi) implementation checks if there are any trivially entailed constraint, saving its posting to the solver. This process is transparent to the user, which can save constraints by doing nothing (see Section 4.1.3. to see how the implementation of TOY(FDi) saves constraints in the function *post_implications*).

References

- Castiñeiras, I. and Sáenz-Pérez, F. 2011. A CFLP Approach for Modeling and Solving a Real Life Employee Timetabling Problem. In COPLAS'11, 63-71.

- Castiñeiras, I. and Sáenz-Pérez, F. 2012. Improving the Performance of FD Constraint Solving in a CFLP System. In FLOPS'12, 88-103. LNCS 7294. Springer.

- P. Galinier, B. Jaumard, R. M. and Pesant, G. 2007. A Constraint-Based Approach to the Golomb Ruler Problem. http://www.crt.umontreal.ca/~quosseca/pdf/41-golomb.pdf.

- Schulte, C., Tack, G., and Lagerkvist, M. Z. 2010. Modeling and Programming with Gecode. http://www.gecode.org/doc-latest/MPG.pdf