Efficient Algorithms and Heuristics for Strong Local Consistencies

Anastasia Paparrizou

Advisor: **Kostas Stergiou**

Department of Informatics and Telecommunications Engineering,
Uninersity of Western Macedonia, Greece

Contents

- Introduction to Table constraints
- GAC algorithms for table constraints
  - Simple Tabular Reduction (STR)
- Strong Local Consistencies
  - New effiecient algorithms for table constraints
- Adaptive propagation
  - Heuristics
- Conclusions and future directions

Table constraints

- Table constraints are constraints given in extension by listing the tuples of values allowed or forbidden by a set of variables.
- They are widely studied in constraint programming (CP) as they are present in many realworld applications
    - design
    - configuration
    - databases
    - preferences' modeling.
- So far, research on table constraints has mainly focused on the development of fast algorithms to enforce generalized arc consistency (GAC).
- GAC algorithms delete inconsistent values from variable domains and achieve the maximum level of filtering when constraints are treated independently.

GAC algorithms for Table constraints

- Classical algorithms iterate over lists of tuples in different ways
  - Bessiere and Régin 1997, Lhomme and Régin 2005, Lecoutre and Szymanek 2006.
- Recent developments, however, suggested maintaining dynamically the list of supports in constraint tables: these are the variants of simple tabular reduction (STR)
  - Ullmann 2007, Lecoutre 2011, Lecoutre, Likitvivatanavong and Yap 2012
- Alternatively, specially-constructed intermediate structures such as tries (Gent et al. 2007) or multi-valued decision diagrams (MDDs) (Cheng and Yap 2010) have been proposed.
- A more recent development of AC5-based algorithms has also been proposed in (Mairy, Van Hentenryck and Deville 2012), but its relevance has been shown on binary/ternary constraints only.
- Among this variety of algorithms, STR2 along with the MDD approach are considered to be the most efficient ones (especially, for large arity constraints).

STR algorithms

- Structures
  - initialization

table[c]

scp(c) = { x, y, z } position [c]

	x	y	z	position [c]
1	0	0	0	1
2	0	2	1	2
3	1	0	0	3
4	1	0	1	4
5	1	1	2	5
6	2	0	0	6

currentLimit[c]: 6

- Algorithm's steps
  - All tuples are checked until `currentLimit[c]` is reached
    - **if** a tuple is valid **then**
      - values are added to `gacValues[x]`, `gacValues[y]`, `gacValues[z]` respectivelly

    else tuple is removed
  - **foreach** variable `x` ∈ `scp(c)`
    - **if** `gacValues[x]`⊂ `dom(x)` **then**

      `dom(x)`←`gacValues[x]`

      if `dom(x)`= Ø **return** FALSE

      add any c_i to Q, s.t. $c_i \neq c \wedge x \in scp(c_i)$

STR algorithms

- Structures
  - initialization
  - propagation
  - backtracking

initialization

table[c]

	scp(c) = { x,	y,	z }	position [c]
1	0	0	0	1
2	0	2	1	2
3	1	0	0	3
4	1	0	1	4
5	1	1	2	5
6	2	0	0	6

currentLimit[c]: 6

propagation

table[c]

	{ x,	y,	z }	position [c]
1	0	0	0	1
2	0	2	1	6
3	1	0	0	3
4	1	0	1	4
5	1	1	2	5
6	2	0	0	2

currentLimit[c]: 5

STR applied after the removal of (z, 1).
(y, 2) no longer has support and will therefore be deleted.

backtracking

table[c]

	{ x,	y,	z }	position [c]
1	0	0	0	1
2	0	2	1	6
3	1	0	0	3
4	1	0	1	5
5	1	1	2	4
6	2	0	0	2

currentLimit[c]: 6

Structures obtained after backtracking

STR algorithms

- Structures
  - initialization
  - propagation
  - backtracking

initialization

table[c]

	scp(c) = { x,	y,	z }	position [c]
1	0	0	0	1
2	0	2	1	2
3	1	0	0	3
4	1	0	1	4
5	1	1	2	5
6	2	0	0	6

currentLimit[c]: 6

propagation

table[c]

	{ x,	y,	z }	position [c]
1	0	0	0	1
2	0	2	1	6
3	1	0	0	3
4	1	0	1	5
5	1	1	2	4
6	2	0	0	2

currentLimit[c]: 4

STR applied after the removal of (z, 1). (y, 2) no longer has support and will therefore be deleted.

backtracking

table[c]

	{ x,	y,	z }	position [c]
1	0	0	0	1
2	0	2	1	6
3	1	0	0	3
4	1	0	1	5
5	1	1	2	4
6	2	0	0	2

currentLimit[c]: 6

Structures obtained after backtracking

Strong Local Consistencies

- GAC algorithms process one constraint at a time and thus, they cannot exploit possible intersections that may exist between different constraints.
- On the other hand, existing algorithms for consistencies stronger than GAC that can exploit constraint intersections are generic and thus very expensive.
- A specialized algorithm for table constraints, called `maxRPWC+`, that achieves a consistency stronger than GAC was proposed very recently (Paparrizou and Stergiou 2012).
- This algorithm extends the `GACva` algorithm (Lecoutre and Szymanek 2006) and enforces a domain filtering restriction of PWC, called max Restricted PairWise Consistency (maxRPWC) (Bessiere, Stergiou, and Walsh 2008).

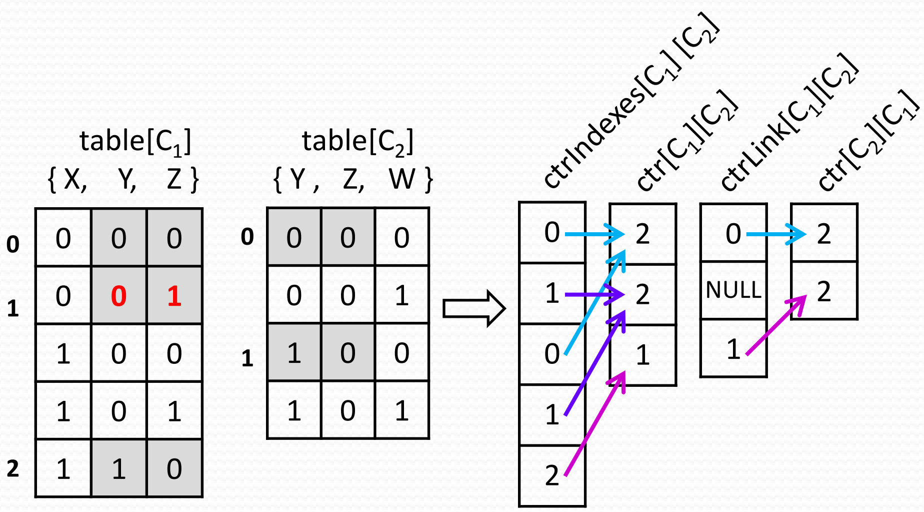
New efficient Algorithms

- One objective of this research is to propose efficient algorithms for strong local consistencies that can be applied on table constraints and can be easily adopted by standard CP solvers.
- Towards this, we propose a new higher-order consistency algorithm for table constraints, called `eSTR*`.
- It is based on simple tabular reduction (`STR`) that is able to efficiently achieve ***F**ull **P**air**W**ise **C**onsistency* (`PWC+GAC`).
- Despite its high space and time requirements to construct its structures, its *worst-case time complexity* is quite close to that of STR algorithms.

  - *The concept of `eSTR*` is to extend any STR-based algorithm to achieve stronger pruning, simply by introducing a set of counters for each intersection between any two constraints c_i and c_j.*

Extending STR algorithms

- Structures
  - description

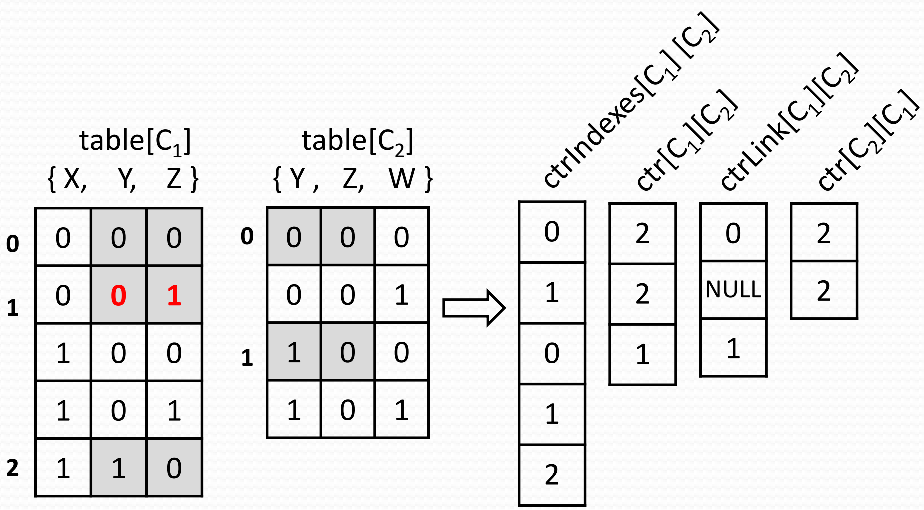


eSTR structures for the intersection of C_1 with C_2 on variables Y and Z. The highlighted values show the first occurrence of the different subtuples for scp(C_1)∩scp(C_2).

- `ctr[c][cᵢ]`. holds the number of valid tuples in `table[c]` that include the subtuple for variables in `scp(c)∩scp(cᵢ)` that appears in at least once in `table[c]`.
- `ctrIndexes[c][cᵢ]` holds the index of the counter in `ctr[c][cᵢ]` that is associated with the subtuple `[scp(c)∩scp(cᵢ)]`.
- `ctrLink[c][cᵢ]` is an array of size `ctr[c][cᵢ].length` that links `ctr[c][cᵢ]` with `ctr[cᵢ][c]`. It holds the index of the counter in `ctr[cᵢ][c]` that is associated with that subtuple. If the subtuple is not included in any tuple of `table[cᵢ]` then `ctrLink[c][cᵢ][j]` is set to NULL.

eSTR algorithm

- Structures
  - initialization



- Algorithm's steps
  - All tuples are checked until `currentLimit[c]` is reached
    - **if** a tuple is valid **AND** *PW-consistent*
      - values are added to `pwValues[x]`, `pwValues[y]`, `pwValues[z]` respectivelly

      else tuple is removed

      counter is *updated*
  - **foreach** variable `x` ∈ `scp(c)`
    - **if** `pwValues[x]`⊂ `dom(x)`

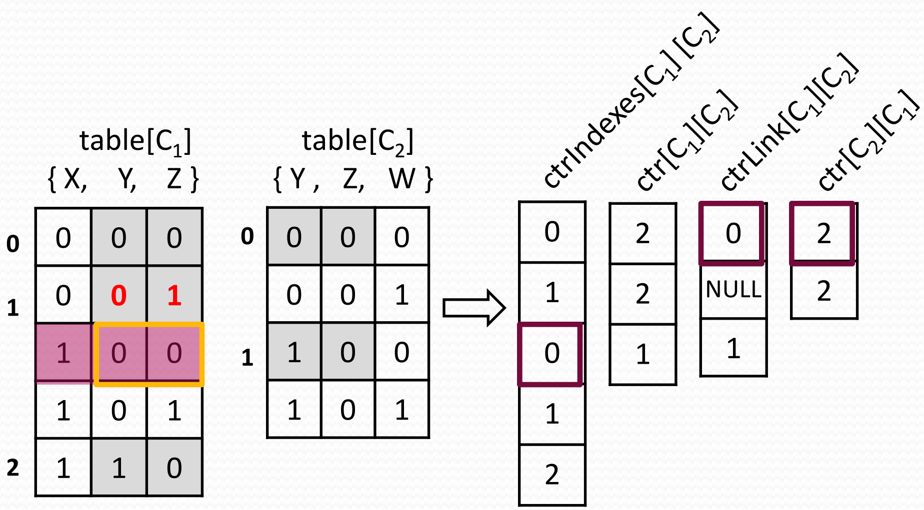
      `dom(x)`←`pwValues[x]`

      if `dom(x)`= Ø **return** FALSE

      add any `ci` to Q, s.t. `ci`≠`c` ∧ `x` ∈`scp(ci)`

eSTR algorithm

- Structures
  - propagation



Function 2 $\texttt{isPWconsistent}(c, \texttt{index})$

```
for each c_i ≠ c s.t. |scp(c_i) ∩ scp(c)| > 1 do
    j ← ctrIndexes[c][c_i][index]
    k ← ctrLink[c][c_i][j]
    if k = NULL OR ctr[c_i][c][k] = 0 then
        return FALSE
return TRUE
```

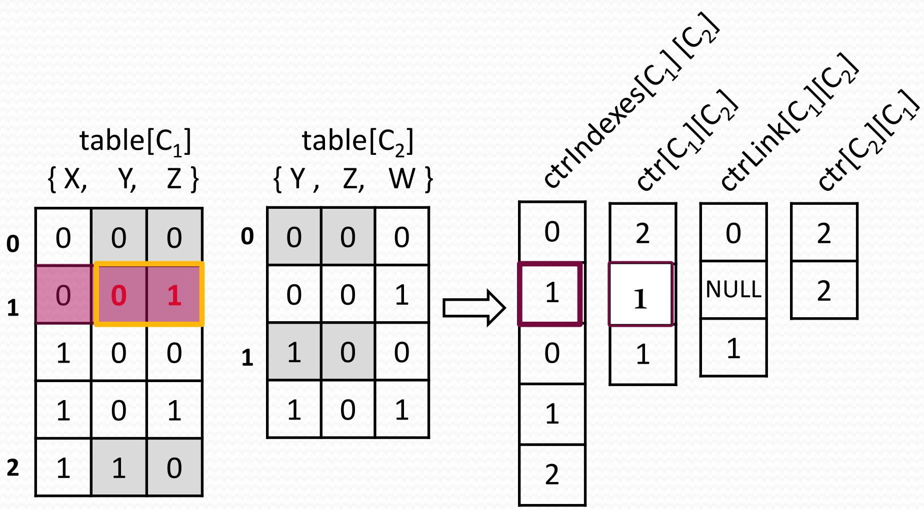
Function 3 $\texttt{updateCtr}(c, \texttt{index})$

```
for each c_i ≠ c s.t. |var(c_i) ∩ var(c)| > 1 do
    j ← ctrIndexes[c][c_i][index];
    ctr[c][c_i][j] ← ctr[c][c_i][j] − 1
    if ctr[c][c_i][j] = 0 then
        add c_i to Q
```

eSTR checks if the tuple (1, 0, 0) of C_1 is PW-consistent

eSTR algorithm

- Structures
  - propagation



eSTR removes tuple (0, 0, 1) of C_1 and updates its counters

Function 2 isPWconsistent(c, index)

1: **for each** $c_i \neq c$ s.t. $|scp(c_i) \cap scp(c)| > 1$ **do**
2: $\quad j \leftarrow$ ctrIndexes$[c][c_i]$[index]
3: $\quad k \leftarrow$ ctrLink$[c][c_i][j]$
4: $\quad$ **if** k = NULL OR ctr$[c_i][c][k] = 0$ **then**
5: $\quad\quad$ **return** FALSE
6: **return** TRUE

Function 3 updateCtr(c, index)

1: **for each** $c_i \neq c$ s.t. $|var(c_i) \cap var(c)| > 1$ **do**
2: $\quad j \leftarrow$ ctrIndexes$[c][c_i]$[index];
3: $\quad$ ctr$[c][c_i][j] \leftarrow$ ctr$[c][c_i][j] - 1$
4: $\quad$ **if** ctr$[c][c_i][j] = 0$ **then**
5: $\quad\quad$ add c_i to Q

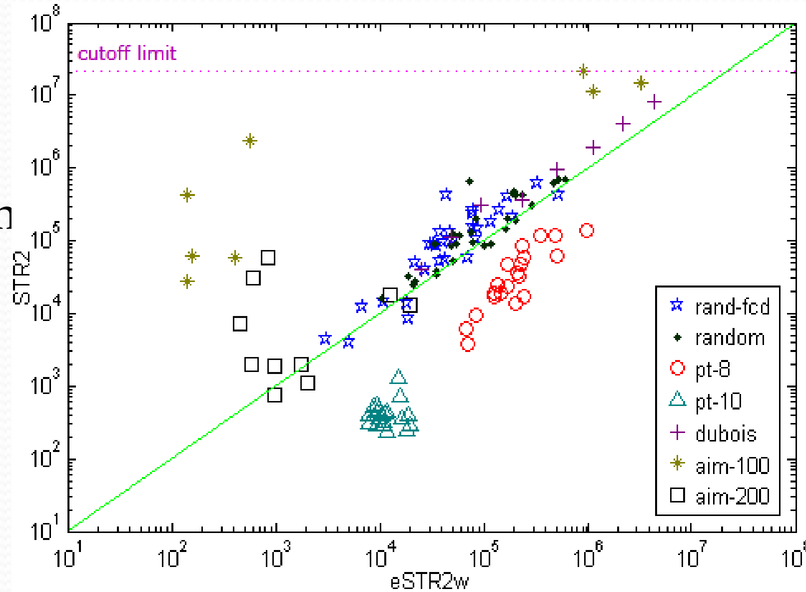
A weak version of **eSTR**, denoted by **eSTRw** can be obtained by discarding lines 4–5 of Function 3 (i.e., the update of Q is ignored when a PW-support is lost).

Theoretical results

- Algorithm eSTR applied to a CN P enforces Full PairWise Consistency on P.
- PWC+GAC and PWC+maxRPWC are equivalent.
- The consistency level achieved by Algorithm `eSTRw` is incomparable to maxRPWC and PWC.
- The ***worst-case time complexity*** of one call to `eSTR` is `O(rd+max(r,g)t)` where `r` denotes the arity of the constraint, `t` the size of its table and `g` the number of intersecting constraints.
  - The worst-case time complexity of `STR` is `O(rd+rt)` (Lecoutre 2011).
- The ***worst-case space complexity*** of `eSTR` for handling one constraint is `O(n+max(r,g)t)`.
  - The worst-case space complexity of `STR` is `O(n+rt)` per constraint (Lecoutre 2011). Each additional `eSTR` structure is `O(t)` per intersecting constraint, giving `O(gt)`.

eSTR2w vs. STR2

- Points above the diagonal are solved faster by `eSTR2w`. The majority of the instances are above and belong to *Random*, *Random-forced* and *Dubois*.
- On *Aim* classes `eSTR2w` can outperform `STR2` by several orders of magnitude on some instances.
- They are particularly expensive on classes of problems which include intersections on large sets of variables, as is the case with the *Positive-table* and *BDD* instances.



Adaptive Propagation

- Since GAC may still be superior in many problems we also suggest ways to interleave GAC with stronger consistency algorithms.
- One such way is to apply heuristics that can dynamically select between GAC and a stronger propagator during search.
- We describe and evaluate simple, fully automated heuristics that monitor the effects of propagation and are applicable on constraints of any arity.
- Experimental results demonstrate that the proposed heuristics for adaptive propagation result in a more robust solver.

Fully Automated Heuristics

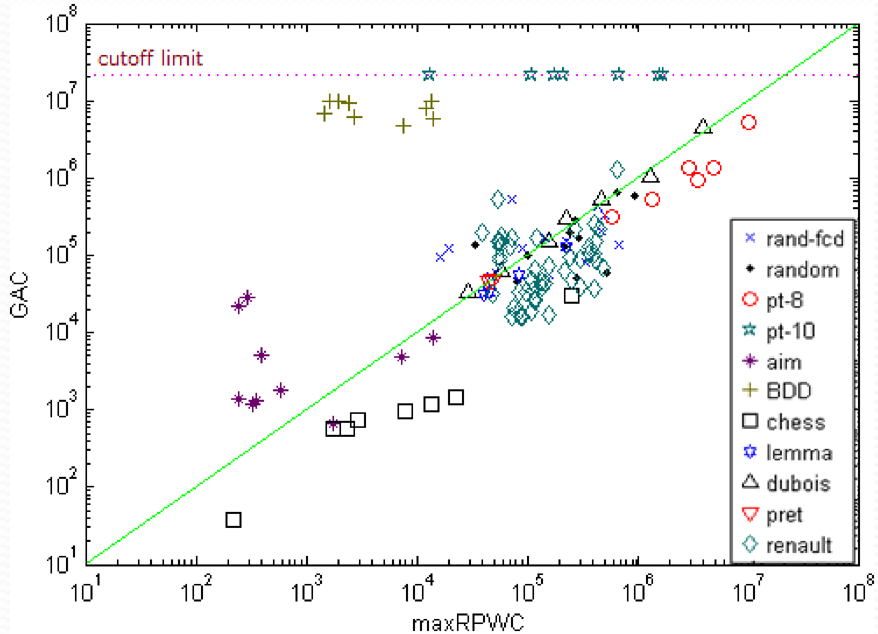
- objective: the exploitation of the filtering power offered by strong propagation methods without incurring severe CPU time penalties or requiring user involvement.
- concept: switching between a weak (W) and a strong (S) propagator for individual constraints during search when a propagation event occurs.
  - The $\mathbf{H}_{\mathbf{dwo}}$ (resp. $\mathbf{H}_{\mathbf{del}}$) heuristic applies a standard propagator on a constraint (e.g. domain consistency) until the constraint causes a domain wipeout - DWO (resp. at least one value deletion). Then, in the immediately following revision of the constraint, a stronger local consistency (e.g. SAC) is applied. (Stergiou 2008)
- ***Refinements*** of $\mathbf{H}_{\mathbf{dwo}}$ and $\mathbf{H}_{\mathbf{del}}$
  - $\mathbf{H}^{\mathbf{v}}_{\mathbf{dwo}}$ (resp. $\mathbf{H}^{\mathbf{v}}_{\mathbf{del}}$) restricts the application of the strong propagator on variables that suffered a propagation event (DWO or value deletion) in the immediately preceding constraint revision as opposed to all variables in the constraint's scope.

AC3 schema with $\mathbf{H}_{\mathbf{dwo}}$

```
Q←C
while Q ≠ Ø do
    pick and delete c from Q
    rev[c]++
    if rev[c]-dwo[c]=1 then
        apply S
    else apply W
    if dom(x)=Ø {∀ x ∊ scp(c)} then
        dwo[c]=rev[c]
        return FAIL
return SUCCESS
```

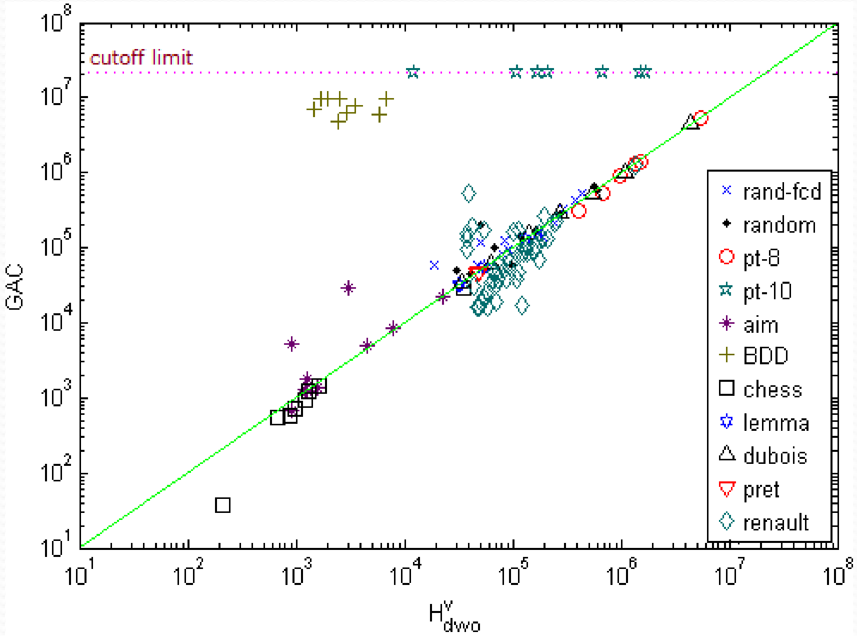
Experiments

- We have considered `GACva` as the standard propagator **W**, given that it is the most commonly used local consistency.
- As the **S** propagator we have considered two strong local consistencies, `maxRPWC` and `SAC`, since we are interested in non-binary problems.
- This figure clearly demonstrates the performance gap between GAC and maxRPWC.
  - GAC is faster on the majority of the instances, often by large margins.
  - Since it is a weaker consistency level, it sometimes thrashes, while the stronger maxRPWC does not.
  - These results justify the need for a robust method that can achieve a balance between the two.



GAC vs. H^{v}_{dwo}

- This figure clearly demonstrates the benefits of the adaptive heuristics.
- Although the majority of the instances is still below the diagonal they are much closer to it, indicating small differences between the two methods.
- These are instances where the application of `maxRPWC+` does not offer any notable reductions in search tree size.
- On the other hand, there are still instances where `GACva` thrashes while H^{v}_{dwo}, following `maxRPWC+`, does not.



Conclusions

- We have introduced a new higher-order consistency algorithm for table constraints that enforces FPWC.
- It is based on an original combination of two techniques that have proved their worth: simple tabular reduction and tuple counting.
- Moreover, we have shown that adaptive propagation schemes can exploit efficiently the advantages offered by strong propagators in a fully automated way.
- The presented work can pave the way for the design and implementation of even more efficient higher-order methods for table constraints.
- Also, it can perhaps help initiate a wider study on specialized higher-order consistency algorithms for global constraints.
- We believe that strong local consistencies can pay off, provided that we have efficient methods to apply them.

Publications

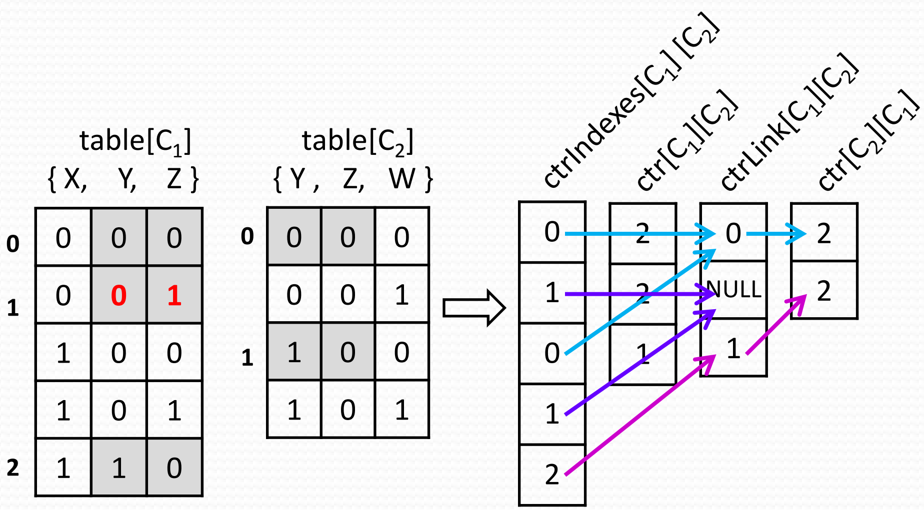
- Christophe Lecoutre, Anastasia Paparrizou, Kostas Stergiou, "*Extending STR to a Higher-Order Consistency*", ***AAAI-13***, Bellevue, Washington (To appear).
- Anastasia Paparrizou, Kostas Stergiou, "*Evaluating Simple Fully Automated Heuristics for Adaptive Constraint Propagation*", **ICTAI-12**, pp. 880-885, Athens, Greece.
- Anastasia Paparrizou, Kostas Stergiou, "*An Efficient Higher-Order Consistency Algorithm for Table Constraints*", **AAAI-12**, pp. 535-541, Toronto, Ontario, Canada.
- Anastasia Paparrizou, Kostas Stergiou, "*Extending Generalized Arc Consistency*", **SETN 2012**, LNCS (LNAI), Vo. 7297, pp. 174-181, Lamia, Greece.
- Thanasis Balafoutis, Anastasia Paparrizou, Kostas Stergiou, Toby Walsh, "*New Algorithms for max Restricted Path Consistency*", **Constraints**, Vo 16, No 4, pp. 372-406, Springer (2011).
- Thanasis Balafoutis, Anastasia Paparrizou, Kostas Stergiou, Toby Walsh, "*Improving the performance of maxRPC*", **CP 2010**, LNCS, Vo 6308, pp. 69-83, St Andrews, Scotland.

Extending STR algorithms

- The central idea of `eSTR*` is to store the number of times that each subtuple appears in the intersection of any two constraints.
  - For each constraint c, we introduce a set of counters for each (non trivial) intersection between c and another constraint c_i .
- Assuming that S is the set of variables that are common to both c and c_i, at any time each counter in this ***set holds the number of valid tuples in c's table*** that include a specific combination of values for S .
- In this way, once a tuple $\tau \in$table(c) has been verified as valid, we can check if it has a PW-support in table(c_i) simply by observing the value of the corresponding counter (i.e., the counter for subtuple [scp(c)∩scp(c_i)]).
- If this counter is greater than 0 then τ has a PW-support in table(ci).
- Importantly, this check is done in **constant time**.

Extending STR algorithms

- Structures
  - description



eSTR structures for the intersection of C_1 with C_2 on variables Y and Z. The highlighted values show the first occurrence of the different subtuples for $scp(C_1) \cap scp(C_2)$.

- `ctr[c][cᵢ]`. holds the number of valid tuples in `table[c]` that include the subtuple for variables in `scp(c)∩scp(cᵢ)` that appears in at least once in `table[c]`.
- `ctrIndexes[c][cᵢ]` holds the index of the counter in `ctr[c][cᵢ]` that is associated with the subtuple `[scp(c)∩scp(cᵢ)]`.
- `ctrLink[c][cᵢ]` is an array of size `ctr[c][cᵢ].length` that links `ctr[c][cᵢ]` with `ctr[cᵢ][c]`. It holds the index of the counter in `ctr[cᵢ][c]` that is associated with that subtuple. If the subtuple is not included in any tuple of `table[cᵢ]` then `ctrLink[c][cᵢ][j]` is set to NULL.

Indicative instances…

- Comparing eSTR2 to STR2 it seems that there are problem classes where it can be considerably more efficient (*Random, Random-forced* and *Dubois*).
- eSTR2 can outperform STR2 by several orders of magnitude on some instances of Aim classes.
- The new algorithms are over one order of magnitude faster than STR2 on *Positive table-10* instances which are proven unsatisfiable without search.
- The extra filtering of eSTR2 does pay off on some classes as node counts are significantly reduced (*Aim*) while on other classes it does not (*Random*).
- On the other hand, STR2 is better than the proposed algorithm on *Positive table* problems and of course *BDD*, where eSTR2 and eSTR2w exhausted the available memory.
- Finally, comparing our algorithms to maxRPWC+ it is clear that they are superior as they are faster on all the tested classes (except *BDD*).

Instance		STR2	maxRPWC+	eSTR2^w	eSTR2
rand-3-20-20 60 -632-fcd-8	t	130	102	**37**	66
	n	128,221	33,924	27,490	27,272
rand-3-20-20 60 -632-fcd-26	t	430	183	**43**	80
	n	534,012	38,556	26,531	26,489
rand-3-20-20-60 -632-19	t	450	536	**187**	220
	n	462,920	129,618	121,199	120,795
rand-3-20-20-60 -632-26	t	670	295	**74**	137
	n	827,513	64,665	45,268	45,426
rand-8-20-5-18 -800-7	t	**17**	753	30	26
	n	17,257	3,430	1,001	626
rand-8-20-5-18 -800-11	t	**19**	1,568	52	55
	n	67,803	7,920	3,299	1,279
rand-10-20-10-5 -10000-1	t	0.4	208	**0.02**	**0.02**
	n	1,110	0	0	0
rand-10-20-10-5 -10000-6	t	0.4	1,687	**0.03**	**0.02**
	n	1,110	0	0	0
bdd-21-133-18-78-6	t	30	1.5	-	-
	n	20,582	0	-	-
dubois-22	t	315	734	**96**	182
	n	129,062,226	41,538,898	41,538,898	40,037,032
dubois-27	t	8,404	28,358	**4,448**	8,492
	n	4,206,712,146	1,651,070,290	1,651,070,290	1.808,444,072
aim-100-1-6-sat-2	t	423	0.16	**0.02**	**0.02**
	n	29,181,742	100	100	100
aim-100-2-0-sat-3	t	2,447	0.3	0.14	**0.05**
	n	177,832,989	111	111	100
aim-200-2-0-sat-1	t	57	0.7	0.6	**0.1**
	n	2,272,993	1,782	9,847	200
aim-200-2-0-sat-4	t	30	0.7	0.4	**0.2**
	n	987,160	1,965	4,276	499