Constructive Interval Disjunction (CID)

Gilles Trombettoni, Gilles Chabert

COPRIN team, University of Nice–Sophia, INRIA Sophia Antipolis, France

CP, September 2007, Providence

Outline

1. Motivations
2. CID filtering
3. A hybrid 3B / CID algorithm
4. CID-based splitting strategy
5. Experiments
6. Conclusion

Constructive Interval Disjunction

- **Constructive disjunction** (by PVH et al. at ECRC) was introduced to handle disjunctions of constraints: application to scheduling, bin packing...
- In finite CSPs, constructive disjunction can be applied to the variable domains ($x = v_1 \vee ... \vee x = v_n$): success on Sudoku
- Question: can **"constructive domain disjunction"** be applied to **numerical CSPs** ?

Constructive Interval Disjunction

- **Constructive disjunction** (by PVH et al. at ECRC) was introduced to handle disjunctions of constraints: application to scheduling, bin packing...
- In finite CSPs, constructive disjunction can be applied to the variable domains ($x = v_1 \vee ... \vee x = v_n$): success on Sudoku
- Question: can **“constructive domain disjunction”** be applied to **numerical CSPs** ?

Constructive Interval Disjunction

- **Constructive disjunction** (by PVH et al. at ECRC) was introduced to handle disjunctions of constraints: application to scheduling, bin packing...
- In finite CSPs, constructive disjunction can be applied to the variable domains ($x = v_1 \vee ... \vee x = v_n$): success on Sudoku
- Question: can **"constructive domain disjunction"** be applied to **numerical CSPs** ?

Contributions

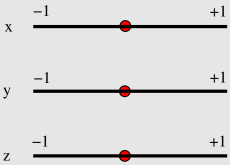
- Study of **shaving** and **constructive disjunction** in numerical CSPs
- Definition of a new partial consistency called **constructive interval disjunction (CID)**
- Design of a filtering operator computing CID
- Design of a hybrid operator 3B/CID
- Design of a new splitting strategy based on CID

Outline

1. Motivations
2. CID filtering
3. A hybrid 3B / CID algorithm
4. CID-based splitting strategy
5. Experiments
6. Conclusion

Example of CID-based filtering

Solving: $x = y, y = -z, z = x$

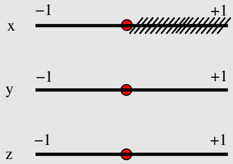


Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

Example of CID-based filtering

Solving: $x = y, y = -z, z = x$

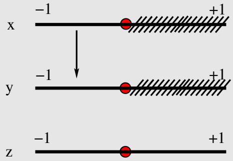


Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

Example of CID-based filtering

Solving: $x = y, y = -z, z = x$

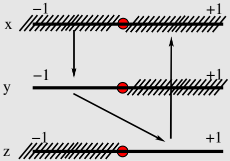


Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

Example of CID-based filtering

Solving: $x = y, y = -z, z = x$



Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

Example of CID-based filtering

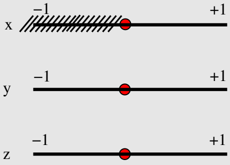
Solving: $x = y, y = -z, z = x$

Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

Example of CID-based filtering

Solving: $x = y, y = -z, z = x$

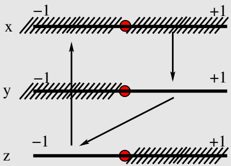


Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

Example of CID-based filtering

Solving: $x = y, y = -z, z = x$



Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

Example of CID-based filtering

Solving: $x = y, y = -z, z = x$

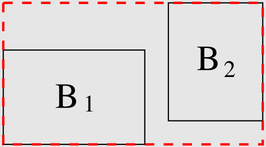
Observations

- The system is already 2B-consistent (Hull-consistent)
- CID (applied on only x) finds the solution
- The 3B-consistency operator implies a slow convergence

CID-consistency

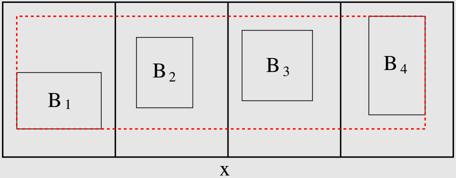
The **Hull** operator (union)

`Hull`(B_1, B_2) is the minimal box including boxes B_1 and B_2.



CID(s)-consistency ... informally

1. Cut the interval of variable x in s slices.
2. Apply a subfiltering operator (`2B` or `Box`) on the sub-boxes.
3. Take the hull of the contracted sub-boxes.



A first CID operator: `CID(s,w)`

Algorithm `CID` *(s: number of slices, w: precision, in-out* $P = (X, C, B)$*: an NCSP, F: subfiltering operator and its parameters)*

```
    repeat
        LoopCID (X, s, P, F)
    until StopCriterion(w, P)
end.
Procedure LoopCID (X, s, in-out P, F)
    for every variable xi ∈ X do
        VarCID (xi, s, P, F)
    end
end.
Procedure VarCID (xi, s, (X, C, in-out B), F)
    B′ ← empty box
    for j ← 1 to s do
        sliceBox ← SubBox (j, s, xi, B) /* the j^th sub-box of B on xi */
        sliceBox′ ← F(X, C, sliceBox) /* perform a partial consistency */
        B′ ← Hull(B′, sliceBox′) /* Union with previous subboxes */
    end
    B ← B′
end.
```

A CID-based solving scheme

Experimental feedback

- Reaching a fixed-point (based on w) is not fruitful $\Rightarrow$ $w = \infty$ is no more a user-defined parameter.
- The right number s of slices $\in [2, 8]$ (default value = 4).
- Tuning the number n' of variables that are varcided between two bisections may be fruitful.

An efficient CID-based solving scheme

Loop until small boxes (solutions) are obtained:

- CID filtering : `CID(s, w-hc4, n')`
- Interval Newton
- Bisection

Outline

1 Motivations

2 CID filtering

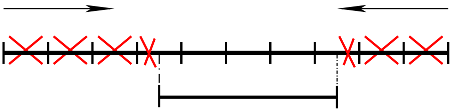
3 A hybrid 3B / CID algorithm

4 CID-based splitting strategy

5 Experiments

6 Conclusion

3B-consistency



CID and 3B (by Olivier Lhomme) have common points and differences.

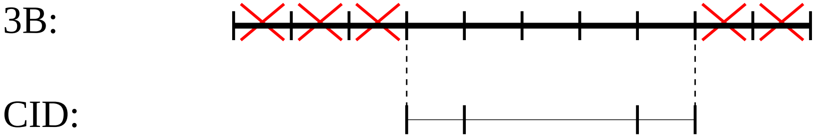
- When the subfiltering process provides an empty box, `3B` and `CID` yield the same result.
- Otherwise, `CID` is stronger: `CID` performs a union between sub-boxes whereas `3B` does simply nothing.
-
  - Constructive disjunction applied to a single variable (i.e., `VarCID`) reduces the box in **all** dimensions.
  - Shaving a variable reduces the box in **one** dimension.

3BCID: a hybrid 3B/CID filtering algorithm

Principle:

1. Three parameters: a maximum number of slices s_{3B} (for the 3B part), a number of slices s_{CID} (for the CID part), a width `w-hc4` (2B/Box).
2. For every interval, `VarShavingCID` first performs a `VarShaving` and then a `VarCID` with $s_{CID} + 2$ slices.

Example: `VarShavingCID` (s_{3B}=10, s_{CID}=`1, w-hc4`)

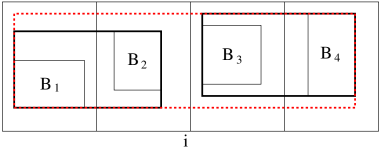


Outline

1. Motivations
2. CID filtering
3. A hybrid 3B / CID algorithm
4. CID-based splitting strategy
5. Experiments
6. Conclusion

A new splitting strategy

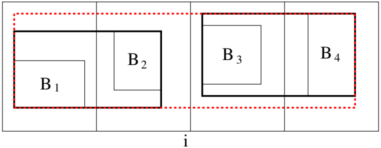
- A `VarCID` operation **learns for free** a relevant information called **ratioBis**.
- $$\texttt{ratioBis} = \frac{\text{Size}(B_i^l) + \text{Size}(B_i^r)}{2 \times \text{Size}(NewBox)}$$



- New CID-based splitting strategy: after a call to `LoopCID`, one selects the variable with the lowest ratioBis.
- `ratioBis` computes the size lost by the `Hull` operation, as compared to a bisection.
  `ratioBis` computes the size lost for avoiding a combinatorial explosion.
- Drawback: The evaluation of the lost size is exact only for the last variable $\Rightarrow$ ratioBis is not "up-to-date" for the first varcided variables.

A new splitting strategy

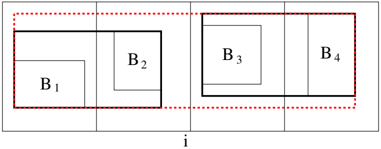
- A `VarCID` operation **learns for free** a relevant information called **ratioBis**.
- $$\texttt{ratioBis} = \frac{\mathrm{Size}(B_i^l) + \mathrm{Size}(B_i^r)}{2 \times \mathrm{Size}(\mathit{NewBox})}$$



- New CID-based splitting strategy: after a call to `LoopCID`, one selects the variable with the lowest ratioBis.
- `ratioBis` computes the size lost by the `Hull` operation, as compared to a bisection.
  `ratioBis` computes the size lost for avoiding a combinatorial explosion.
- Drawback: The evaluation of the lost size is exact only for the last variable $\Rightarrow$ ratioBis is not “up-to-date” for the first varcided variables.

A new splitting strategy

- A `VarCID` operation **learns for free** a relevant information called **ratioBis**.
- $$\texttt{ratioBis} = \frac{\mathrm{Size}(B_i^l) + \mathrm{Size}(B_i^r)}{2 \times \mathrm{Size}(NewBox)}$$



- New CID-based splitting strategy: after a call to `LoopCID`, one selects the variable with the lowest ratioBis.
- `ratioBis` computes the size lost by the `Hull` operation, as compared to a bisection.
  `ratioBis` computes the size lost for avoiding a combinatorial explosion.
- Drawback: The evaluation of the lost size is exact only for the last variable $\Rightarrow$ ratioBis is not “up-to-date” for the first varcided variables.

Outline

1 Motivations

2 CID filtering

3 A hybrid 3B / CID algorithm

4 CID-based splitting strategy

5 Experiments

6 Conclusion

Benchmarks and interval-based solver

Twenty benchmarks of medium difficulty:

- Five sparse (geometric) systems: `Hourglass`, `Tetra`, `Tangent`, `Ponts`, `Mechanism`
- Fifteen benchmarks found in COCONUT and/or the COPRIN team Web pages

Implementation on a new interval-based solver

- Solver in `C++` developed by Gilles Chabert
- Precision: $1e-08$ ($5e-06$ for `Mechanism`)
- Local filtering: `2B` (`2B+Box` for `Yamamura8`)
- Tests performed on a `Pentium IV 2.66 Ghz`

[CID+Newton+bisection] vs [2B+Newton+bisection]

Nom	*n*	**#s**	whc4	2B/Box	CID	CID(n′)	CID(w)	CID()	*s*
Broy.	32	2	15%	758 2e+07	**0.12** 46	0.28 44	0.19 65	0.45 50	4
Hourg.	29	8	5%	24 1e+05	**0.44** 109	0.44 109	0.52 80	0.52 80	4
Tetra	30	256	0.02%	401 1e+06	**10.1** 2116	11.6 1558	11.7 1690	14.5 1320	4
Tang.	28	128	15%	32 1e+05	**3.7** 692	3.7 692	4.9 447	5.1 450	4
React.	20	38	5%	156 1e+06	**15.6** 2588	16.4 2803	16.7 2381	17.7 2156	4
Trig.1	30	1	20%	371(3.4) 5025	**0.12** 1	0.15 3	0.14 2	0.14 3	8
Disc.	27	1	0.01%	5.2 1741	**0.62** 2	1.08 3	0.84 12	2.13 99	8
I5	10	30	2%	692 3e+06	**126** 23105	147 60800	150 20874	157 32309	6
Trans.	12	1	10%	179 1e+06	**66** 11008	79.4 31426	91.4 16333	91.4 16333	8
Ponts	30	128	5%	10.8 34994	**2.7** 388	2.9 338	2.9 380	3.1 304	4

[CID+Newton+bisection] vs [2B+Newton+bisection]

Nom	n	#s	whc4	2B/Box	CID	CID(n′)	CID(w)	CID()	s
Yamam8	8	7	1%	13 1032	**7.5** 104	9.5 60	9.5 60	9.5 60	4
Design	9	1	10%	395 3e+06	**275** 2e+05	278 2e+05	313 76633	313 76633	5
D1	12	16	5%	4.1 35670	**1.7** 464	1.7 464	1.7 464	1.7 464	4
Mechan.	98	448	0.5%	TO(111) 24538	**43.1** 3419	45.2 3300	46.6 2100	47.8 2420	4
Hayes	8	1	0.01%	155 3e+05	**75.8** 1e+05	77 1e+05	111 81750	147 58234	4
Kin1	6	16	10%	84 70368	**76.8** 6892	76.8 6892	83.5 4837	87.4 4100	4
Eco9	8	16	10%	26 2e+05	**18** 55657	19.4 46902	26.6 10064	26.6 10064	3
Belli.	9	8	10%	**80** 7e+05	94.4 1e+05	94.4 1e+05	106 45377	106 45377	4
Trig.2	9	1	20%	61.8	**50.4**	65.14	62.4	68.4	6
Trig.2	5	1	20%	**3.0** 13614	3.8 10221	4.6 4631	6.2 2293	6.6 1887	2
Capras.	4	18	30%	**2.6**	2.73	3.0	4.7	5.1	2

Observations

- Drastic reduction in the number of required bisections $\Rightarrow$ good filtering power of `CID`
- Impressive gains in performance obtained by `CID` on the benchmarks on the top of the table.
- `CID(n')` allows a continuum between pure 2B/Box and CID.
- Due to a combinatorial consideration, `CID` should not be used for systems with a small number of variables.

CID vs 3B vs 3BCID

Nom	n	2B/Box	CID	3B	3BCID($s_{cid} = 1$)	3BCID($s_{cid} = 2$)
BroydenTri	32	758	0.23	0.22	**0.18**	0.19
Hourglass	29	24	0.45	0.73	**0.43**	0.50
Tetra	30	401	**13.6**	20.7	17.1	18.8
Tangent	28	32	4.13	8.67	**3.18**	4.13
Reactors	20	156	18.2	24.2	**15.5**	16.9
Trigexp1	30	3.4	**0.10**	0.26	0.12	0.11
Discrete25	27	5.2	1.37	2.19	1.26	**1.13**
I5	10	692	139	144	**115**	123
Transistor	12	179	71.5	77.9	49.3	**46.9**
Ponts	30	10.8	**3.07**	5.75	4.19	4.43
Yamamura8	8	13	**9.0**	9.1	10.3	10.7
Design	9	395	300	403	**228**	256
D1	12	4.1	1.78	2.99	**1.64**	1.76
Mechanism	98	111	**79**	185	176	173
Hayes	8	155	**99**	188	102	110
Kinematics1	6	84	**76.1**	136	76.6	81.4
Eco9	8	26	**19.3**	40.1	27.0	30.3
Bellido	9	**80**	95	143	**93**	102
Trigexp2-9	9	61.8	52.2	74.5	**39.9**	45.1
Caprasse	4	**2.6**	**3.1**	9.38	4.84	5.35

Observations

Main observations

- Protocol: 7-9 values of parameters have been tried for every algorithm; `w-hc4 = 5%`
- `3BCID` and `CID` always outperform `3B`.
- `3BCID` is competitive with `CID`.

$\Rightarrow$ `3BCID` with $s_{cid} = 1$ is a promising variant of `3B`!

Comparison between splitting strategies

Filtering	CID	CID	CID
Bisection	Round-robin	Largest Int.	CID-based
BroydenTri	0.21	0.18	**0.17**
Hourglass	0.52	0.51	**0.37**
Tetra	**12.1**	28.2	16.4
Tangent	**3.7**	21.7	5.2
Reactors	17.0	13.2	**12.7**
Trigexp1	0.15	0.19	**0.14**
Discrete25	**0.84**	1.49	1.06
I5	**151**	421	179
Transistor	93	**36**	41
Ponts	2.92	5.51	**2.31**
Yamamura8	9.5	6.9	**5.1**
Design	318	334	**178**
D1	**1.72**	2.96	2.50
Hayes	**115**	564	318
Kinematics1	83	70	**63**
Eco9	26.7	31.4	**26.1**
Bellido	107	102	**99**
Trigexp2-9	62	55	**53**
Caprasse	5.16	5.43	**5.04**

Outline

1 Motivations

2 CID filtering

3 A hybrid 3B / CID algorithm

4 CID-based splitting strategy

5 Experiments

6 Conclusion

Conclusion

Recommandations

- Splitting is a **combinatorial** way to refute parts of the search space (a tree search).
  CID-consistency and 3B-consistency operators are **polynomial** ways to refute parts of the search space.
- `CID(n')` has the potential to subsume other filtering operators.
- `3BCID` with $s_{cid} = 1$ is a promising variant of `3B`.

Future works

- Comparison of `CID` and `3BCID` with `weak3B`.
- Deeper study of the CID-based splitting strategy.
- Design of **adaptive** versions of `CID`.