Title: A Disjunctive Decomposition Schema for Constraint Satisfaction

Author: Freuder & Hubbe

Book.: Constraint Programming (Proc. PPCP'93)

Chapter: 17

Year: 1995

Berthe Y. Choueiry (Shu-we-ri)
Ferguson Hall, Room 104
choueiry@cse.unl.edu, Tel: (402)472-5444

Images scanned from paper by Nimit Mehta

Contributions

- One unifying schema for problem decomposition
  valid for old & new algorithms

  $$\text{Usefulness: facilitate} \left\{ \begin{array}{l} \text{description} \\ \text{implementation} \\ \text{comparison} \end{array} \right\} \text{of algorithms}$$

- New decomposition method
  reduces problem size while keeping at least one solution

Assumptions: Binary CSPs, finite domains

Basic idea

- Decomposition: divide and conquer
- Conduct search in alternative spaces (disjunctive)
- Opportunities for parallel and distributed processing

Disjunctive Decomposition Algorithm

Place the initial problem on the Agenda

Until the Agenda is empty:

  Remove a problem P from Agenda

  IF P has only instantiated variables

    THEN EXIT with their values

    ELSE decompose P into a set of subproblems $\{P_i\}$

      Place each non-empty P_i onto the Agenda

EXIT with no solution

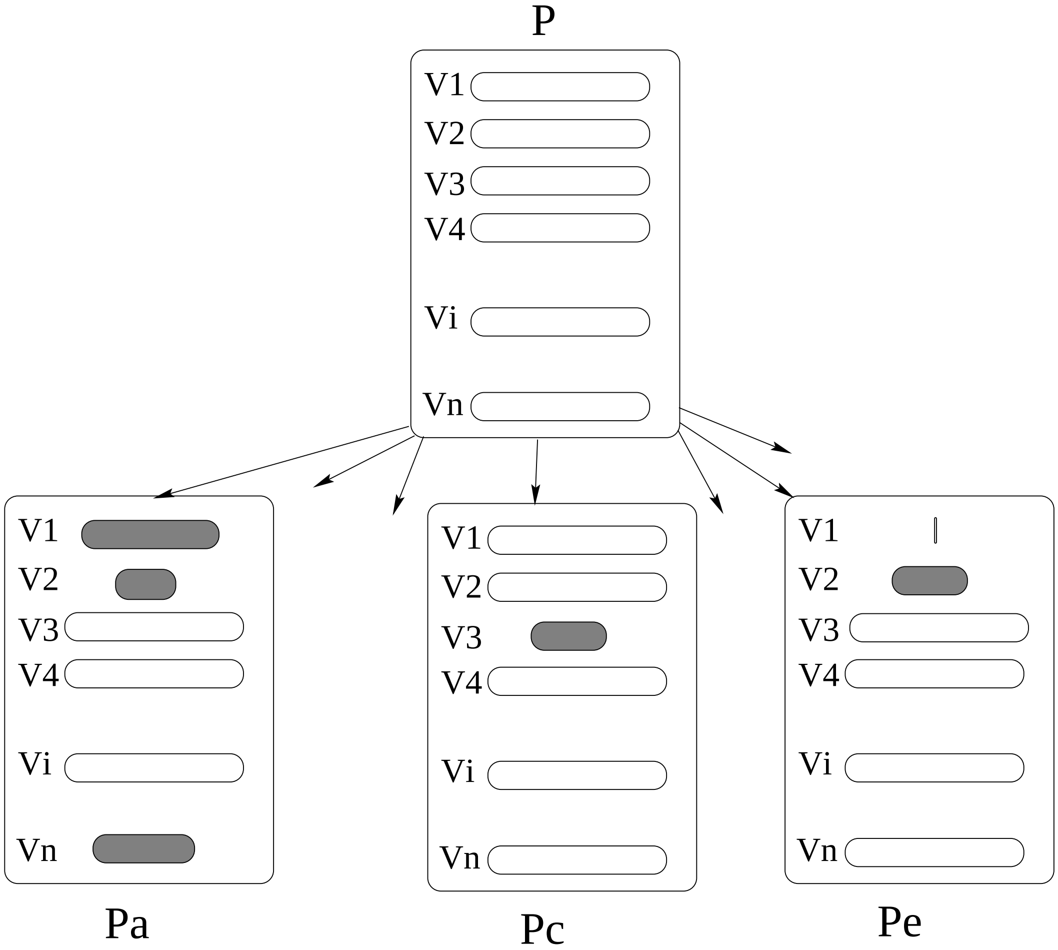
Upon exit, the Cartesian product of the instantiated variable domains is the set of reported solutions

Question: how does the decomposition operate?

Decomposition

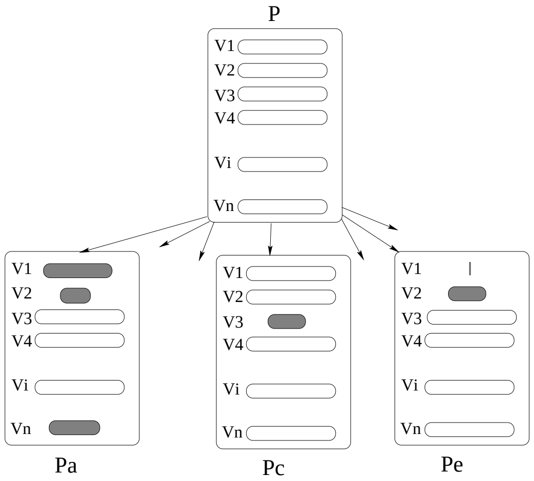
Variable domains in P_i are subset of those in P

Constraints are updates accordingly



- Problem size: number of possibilities ...remember?
- $D_{V_i} = 1 \Leftrightarrow$ instantiated variable
- $D_{V_i} = 0$

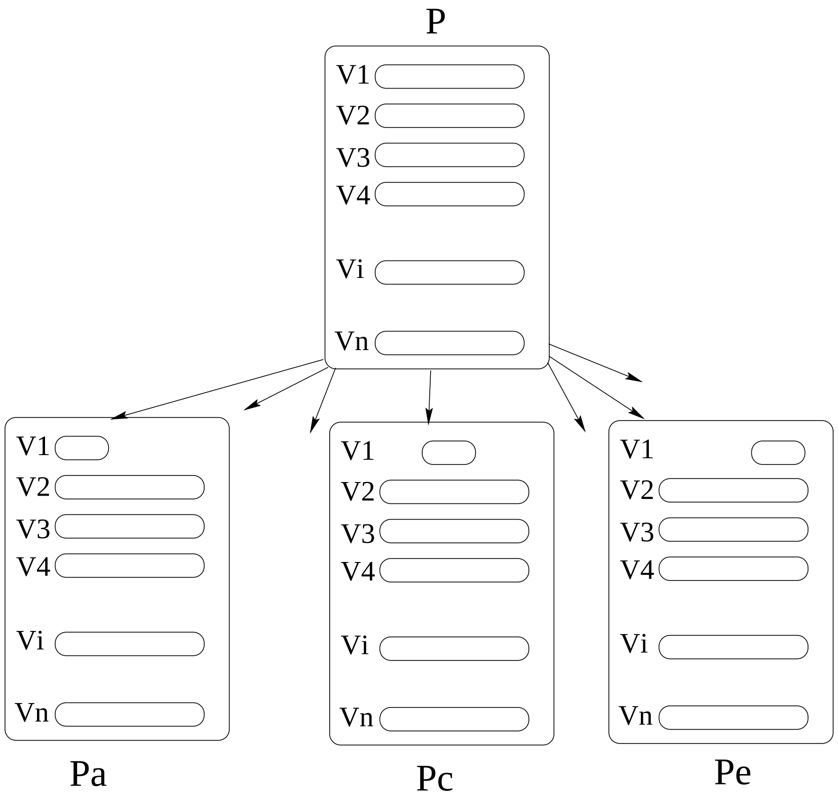
Required properties



- Consistency: pairs of instantiated variables are consistent
- Simplification: $\forall\ P_i$, Size(P_i) $<$ Size(P)
- Semi-completeness: If P has solution $\rightarrow \exists\ P_i$ with a solution

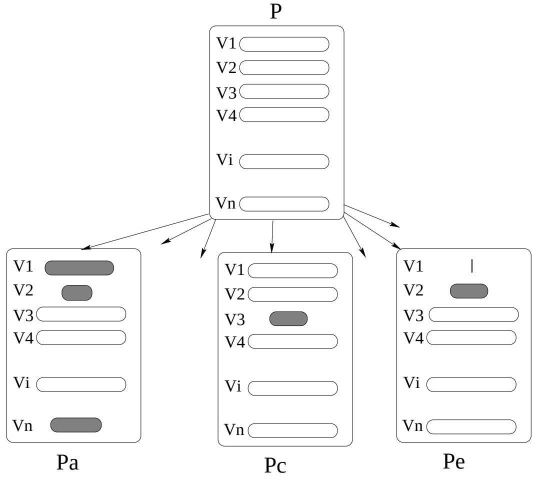
Proposition: The disjunctive decomposition schema terminates and is complete.

Disjunctive decomposition schema is a general paradigm



- domain splitting
- search
- ...

Optional properties



- Completeness:
  Any solution to P will be a solution to at least one P_i
- Non-redundancy:
  Any possibility for P appears in at most one P_i
- Reducibility: $\sum \text{Size}(P_i)$ may be $< \text{Size}(P)$

Completeness guarantees getting all solutions (not required in general)

Two questions are now important

- How is the decomposition performed?
- How is the agenda organized?

  Stack? $\begin{cases} \text{what subproblem is placed next on the agenda?} \\ \text{in what order?} \end{cases}$

$\longrightarrow$ families of (search) algorithms!

If agenda is a stack

$\Rightarrow$ depth-first search

If agenda is managed according to a heuristic

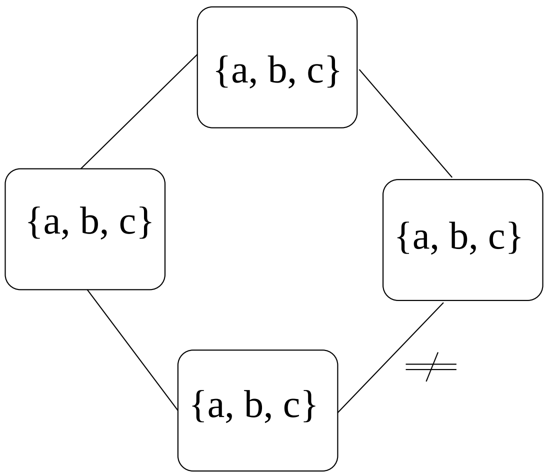
$\Rightarrow$ heuristic ordered search

etc..

$$\text{Study: } \begin{cases} \text{backtrack search} \\ \text{forward checking} \\ \text{network consistency} \\ \text{backtrack with Cartesian product representation} \\ \text{inferred disjunctive constraints} \end{cases}$$

$\rightarrow$ consistent, simplifying, semi-complete

$\rightarrow$ complete? non-redundant? reducing?



Backtrack

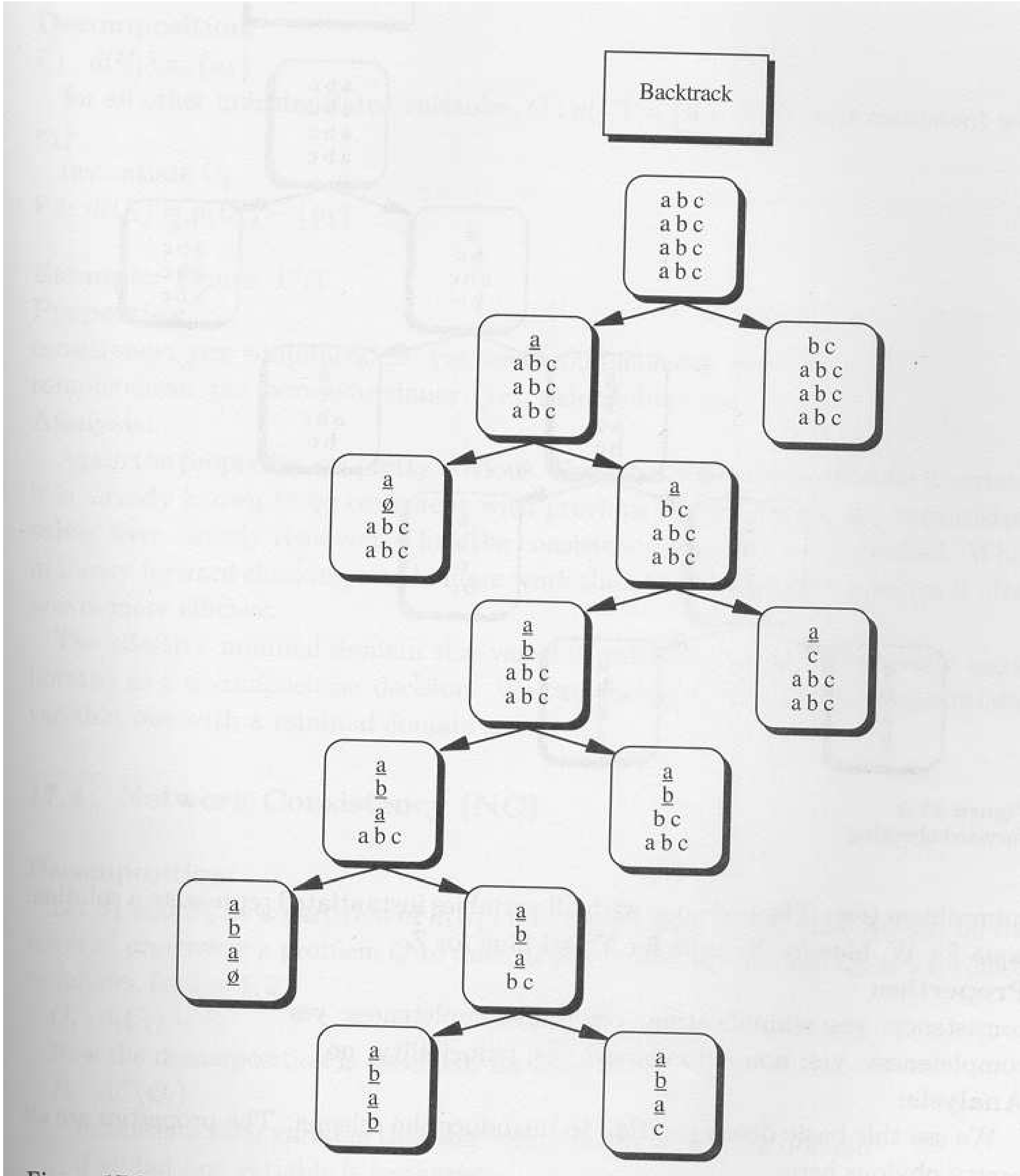


Figure 17.2
Backtrack.

→ consistent? simplifying? semi-complete? complete? non-redundant? reducing?

Forward checking

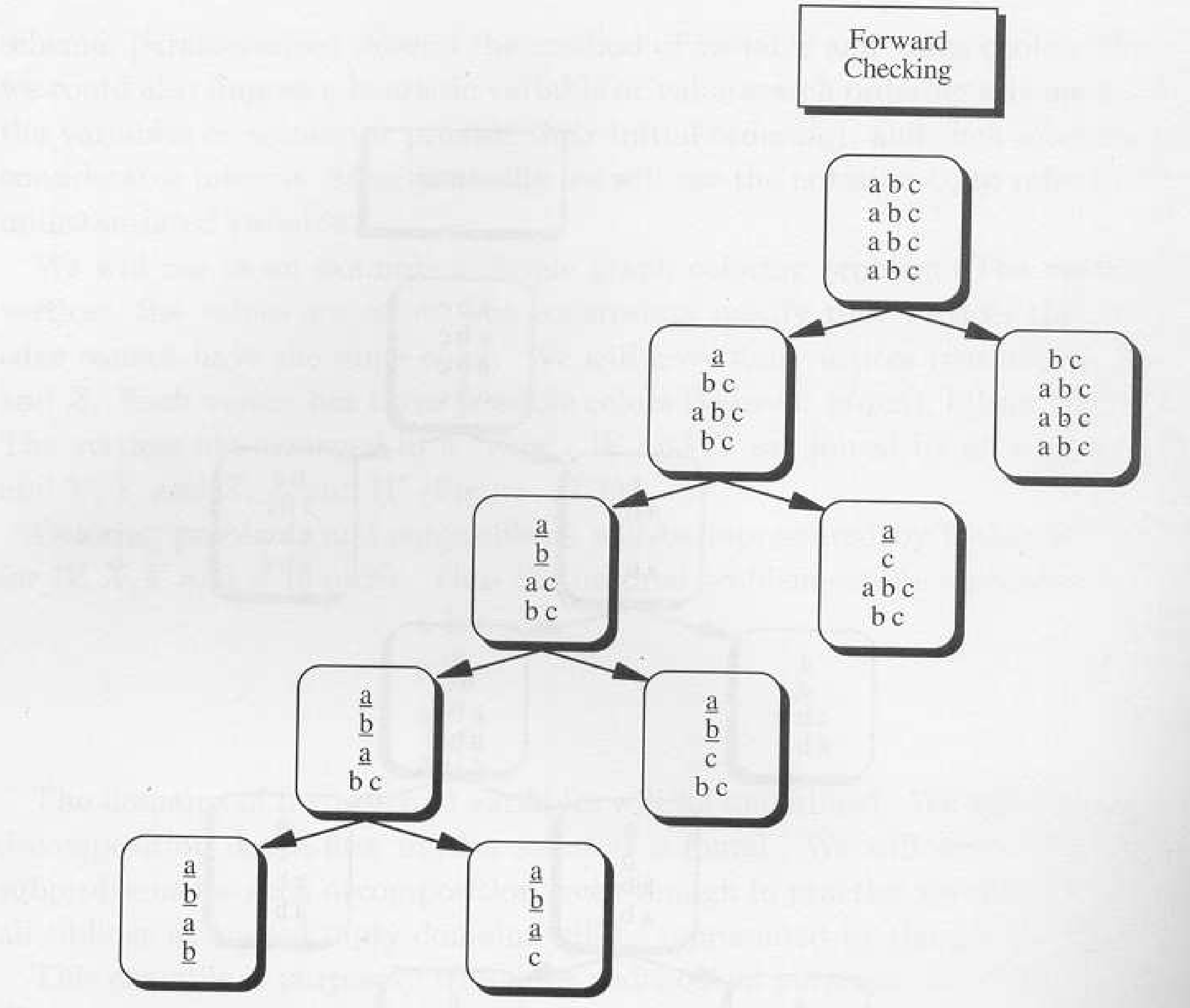


Figure 17.3
Forward checking.

→ consistent? simplifying? semi-complete? complete? non-redundant? reducing?

Network consistency

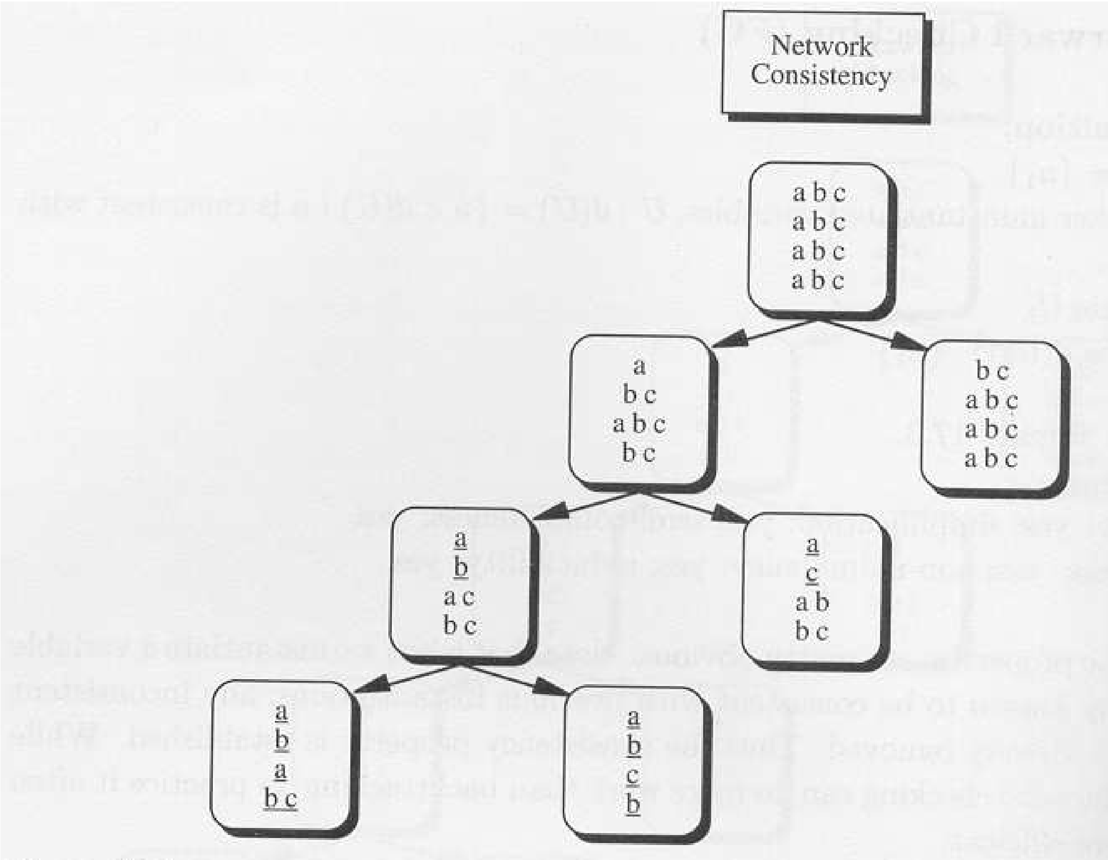


Figure 17.4
Network consistency.

→ consistent? simplifying? semi-complete? complete? non-redundant? reducing?

Backtrack + Cartesian Product Representation (I)

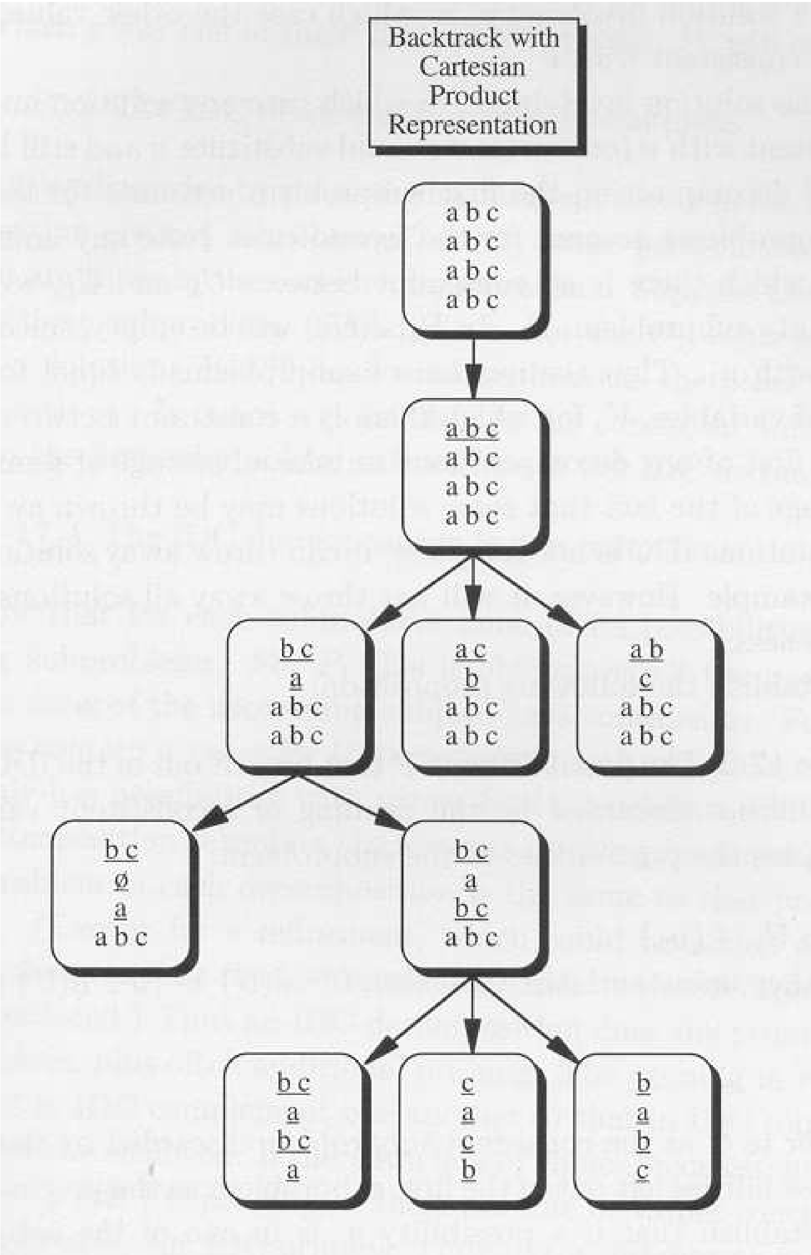
For a variable V_i

Let $\{S_i\}$ be a partition of D_{V_i}
such that two values are in the same subset
iff and only if they are consistent with exactly the same values
in the domains of each of the instantiated variables.

For each S_i, generate a subproblem P_i,
$D_{V_{\text{past}}} \longleftarrow x \in D_{V_{\text{past}}}$ consistent with $y \in S_i$
$D_{V_i} \longleftarrow S_i$

$\rightarrow$ consistent? simplifying? semi-complete? complete?
non-redundant? reducing?

Backtrack with Cartesian Product Representation (II)



→ consistent? simplifying? semi-complete? complete? non-redundant? reducing?

Inferred Disjunctive Constraint (IDC)

For a variable V_i

For P_1:

$V_i \leftarrow u$

$\forall\, V_{\text{future}}, D_{V_{\text{future}}} \leftarrow v \in D_{V_{\text{future}}}$ consistent with $V_i \leftarrow u$

Instantiate V_i (V_i becomes 'past')

if there is one uninstantiated variable remains, instantiate it

For $P_j (j = i+1 \ldots m), m$ #uninstantiated, future variables V_j:

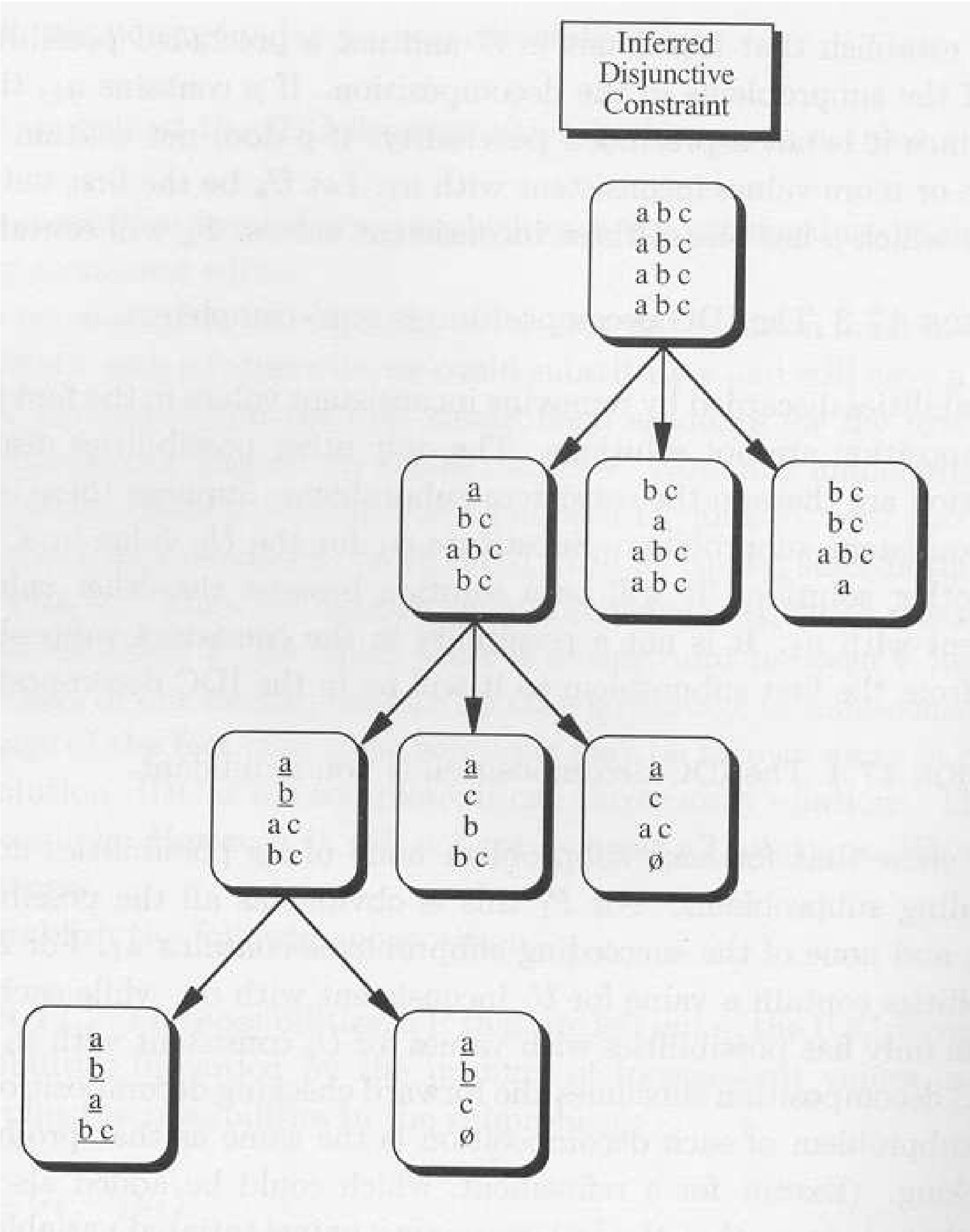
$V_i \leftarrow D_{V_i} - \{u\}$

$1 < k < j : D_{V_{\text{k}}} \leftarrow v \in D_{V_{\text{k}}}$ consistent with $V_i \leftarrow u$

$k = j : D_{V_j} \leftarrow v \in D_{V_j}$ <u>inconsistent</u> with $V_i \leftarrow u$

$\rightarrow$ consistent? simplifying? semi-complete? complete?
non-redundant? reducing?

Inferred Disjunctive Constraint (IDC) (II)



→ consistent? simplifying? semi-complete? complete? non-redundant? reducing?

Conclusions

- Decomposition as a tool for algorithm analysis + synthesis
- General schema allows us to compare decomposition algorithms
- Search in metalevel problem space
- Amenable to distributed and parallel processing
- Future: Mix and match decomposition algorithms