Dynamic Bundling: Less Effort for More Solutions

B.Y. Choueiry and A.M. Davis
Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
`choueiry|amydavis@cse.unl.edu`

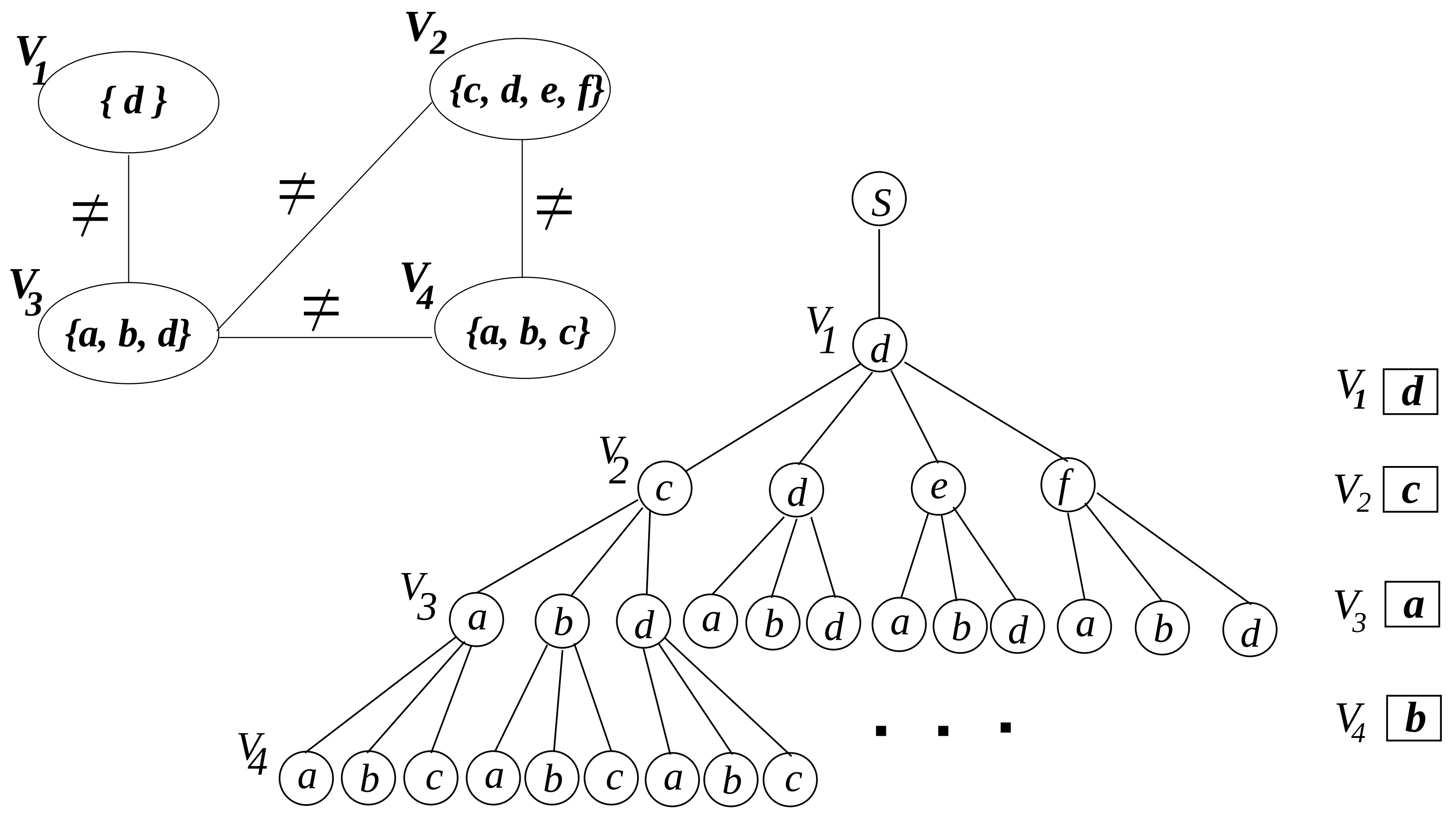
Long term goal:

Build a compact representation of the solution space of decision problems, a landscape over which the user can navigate

Current investigations:

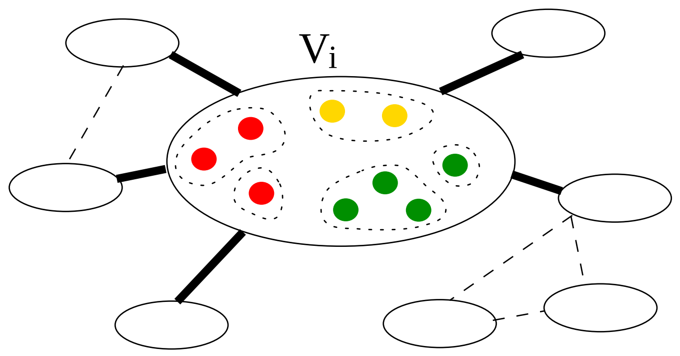
- Focus: Constraint Satisfaction Problems
- Technique: Bundling by interchangeability
- How to do it? Dynamically, repeatedly during backtrack search
- Isn't prohibitively expensive? No, it is actually worthwhile!
- Really? Yes, it reduces the peak of cost at phase transition
- How well does it combine with known BT search heuristics? Very well, does not necessitate aggressive look-ahead
- Lesson: multiple, robust solutions are cheaper than a single solution

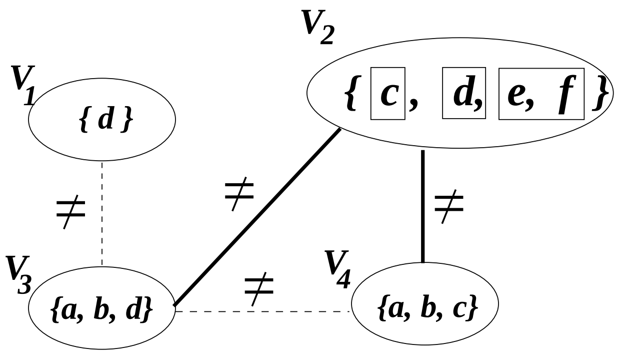
Context: CSP and backtrack search



Neighborhood interchangeability [Freuder 91]

Partitions the domain of a variable into a set equivalence classes

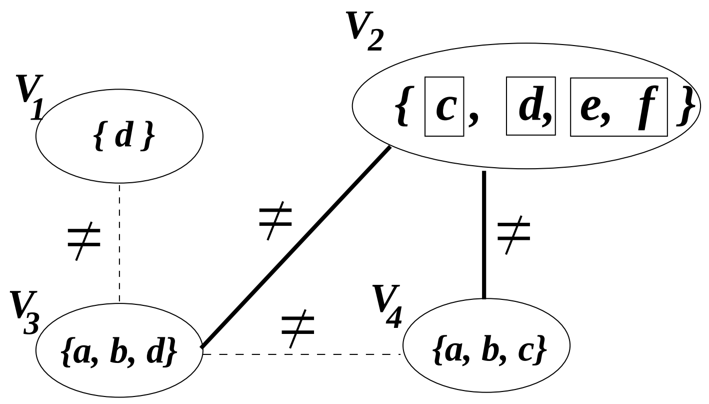


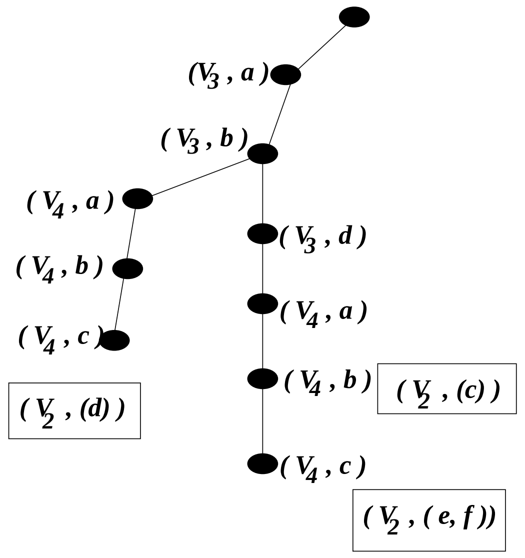


Discrimination tree:

- groups values of V_2 according to what values in the neighborhood of V_2 they are consistent with
- is restricted to `Neigh`$(\{V_2\})$
- time: $O(na^2)$, space: $O(na^2)$

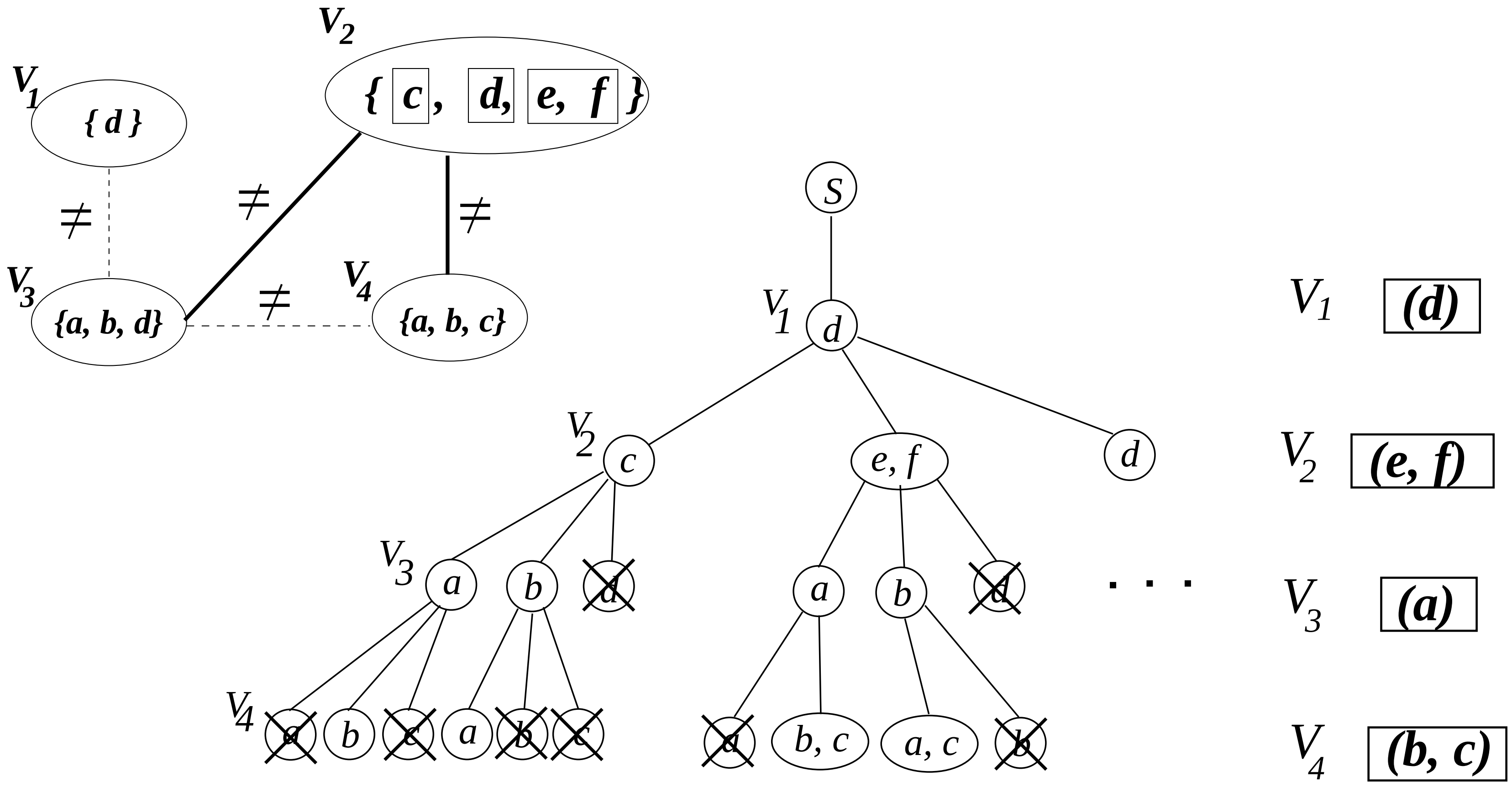
Discrimination tree





- groups values of V_2 according to what values in the neighborhood of V_2 they are consistent with
- is restricted to `Neigh`($\{V_2\}$)
- time: $O(na^2)$, space: $O(na^2)$

Search with bundling



Advantages of search with bundling

- Reduces search space
- Creates bundled solutions
  - Compact representation of the solution space
  - Each bundle is a set of robust solutions

$$V_1 \quad \boxed{(d)}$$

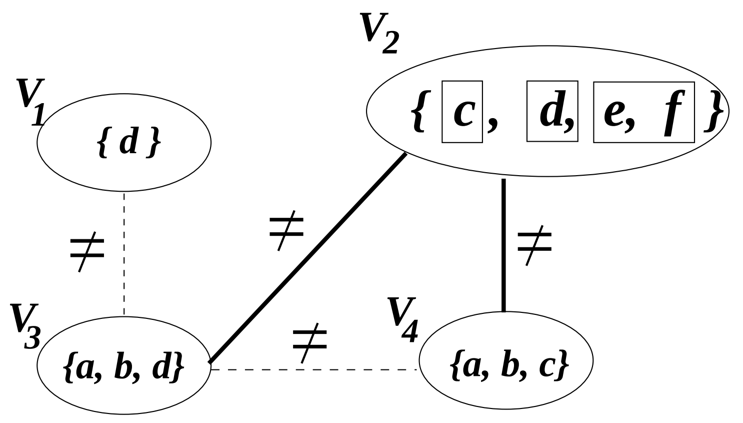
$$V_2 \quad \boxed{(e, f)}$$

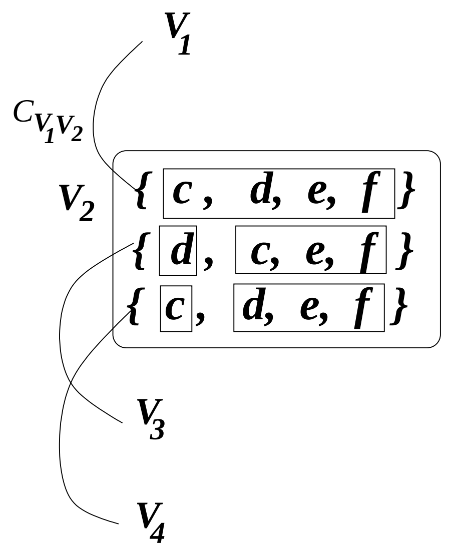
$$V_3 \quad \boxed{(a)}$$

$$V_4 \quad \boxed{(b, c)}$$

Static bundling

NI [Benson & Freuder 92] NI$_C$ [Haselböck 93]

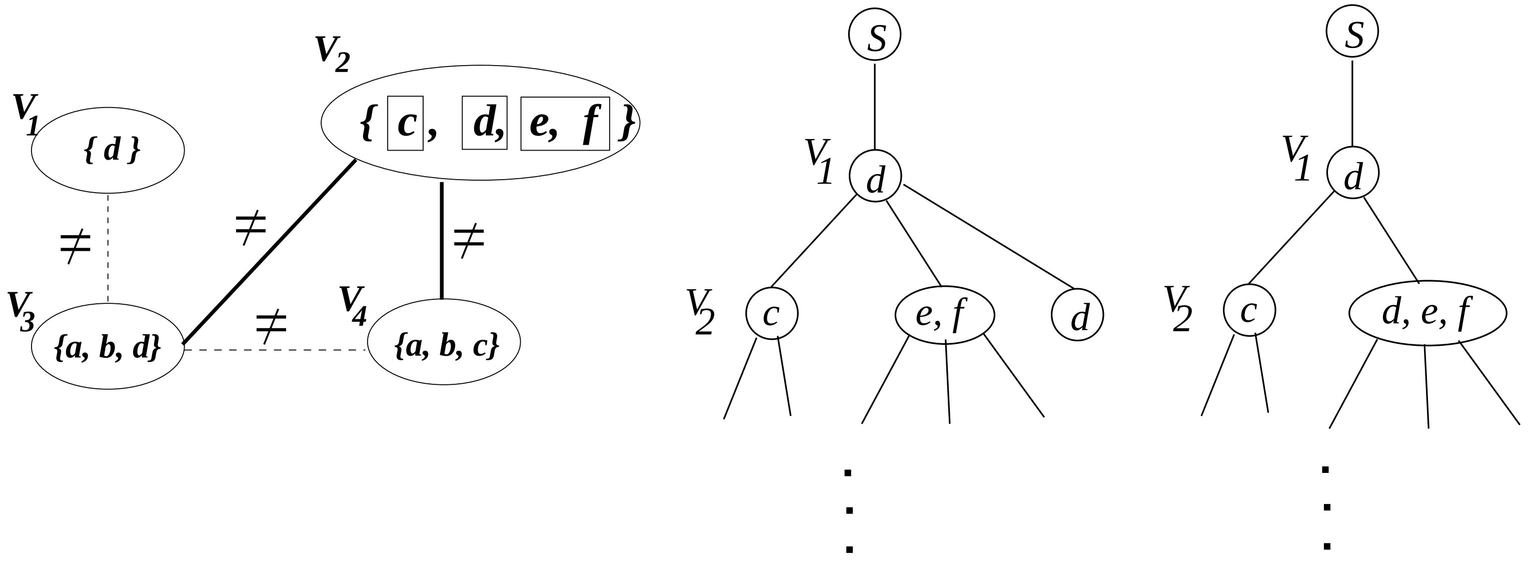




– Before search, find interchangeable values

– Store interchangeable values for duration of search

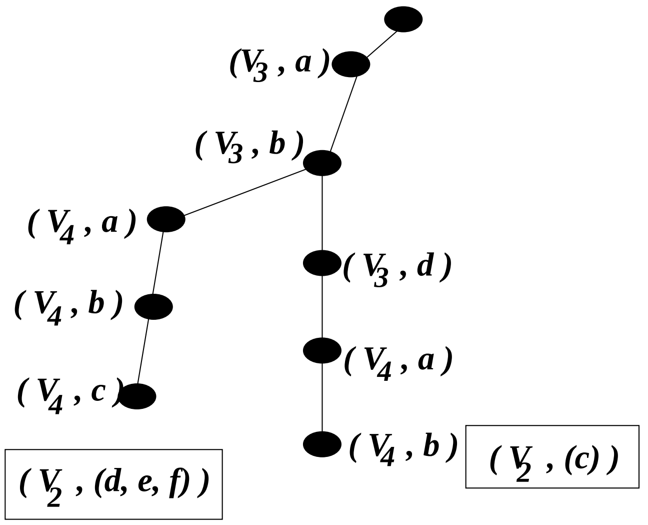
Dynamic bundling, during search



- As search proceeds, 'commitments' are made and more values become interchangeable [Freuder 91]

- Compute interchangeability sets considering constraints with future variables (NPI) using the joint discrimination tree (JDT) [Choueiry & Noubir 98]

Dynamic bundling and forward checking



- Recompute NPI sets, dynamic NPI (DNPI)
  Since the JDT requires the same constraint checks as forward checking (FC), use the JDT for FC [Beckwith & Choueiry 02]

- As a result, at each step during search, we have new larger partitions (bundles) with no more effort than necessary for FC

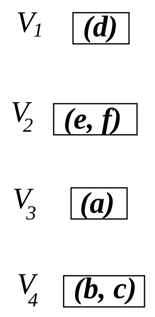
Dynamic bundling, too expensive to be practical?

Evaluation criteria

– Nodes Visited `NV`

– Constraints Checked `CC`

– Size of Solution Bundles `SB`



- Theoretical guarantees, *all solutions, static orderings*
  - No more expensive than forward checking (`NV`, `CC`)
  - No more expensive than static bundling (`NV`)
  - Produces larger bundles than FC and static bundling (`SB`)
- Empirical tests, also measured CPU Time.

Results hold, even when no bundling is possible (e.g., puzzles)

Theoretical comparisons

Number of Nodes Visited

$$\boxed{\begin{tikzcd}\text{FC} \arrow[r, dash, "\geqslant"] & \text{NI}_C \arrow[r, dash, "\geqslant"] & \text{DNPI}\end{tikzcd}}$$

Number of Constraints Checks

$$\boxed{\begin{tikzcd} & \text{NI}_C & \\ \text{FC} \arrow[rr, dash, "\geqslant"] & & \text{DNPI}\end{tikzcd}}$$

Size of Solution Bundles

$$\boxed{\begin{tikzcd}\text{FC} \arrow[r, dash, "\leqslant"] & \text{NI}_C \arrow[r, dash, "\leqslant"] & \text{DNPI}\end{tikzcd}}$$

Objections: Skepticism about dynamic bundling

Dynamic bundling may be prohibitively expensive, when

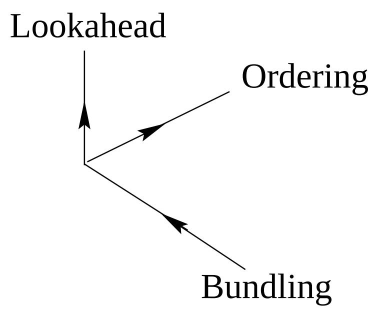
- searching for ONE solution
- using aggressive full lookahead during search (MAC)
- solving difficult problems

We provide empirical evidence of the contrary, dynamic bundling

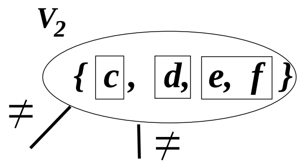
- remains superior under the above conditions
- does not require full lookahead
- reduces the cost peak at phase transition *totally unexpected*

Testing conditions

- Ordering: static `SLD`, dynamic variable `DLD`, dynamic variable-value `LD-MB`



- Lookahead: forward checking (FC), maintaining arc-consistency (MAC)
- Bundling: None, NI_C, DNPI
- (Toy problems, puzzles,) random problems, varying
  — tightness: loose $\cdots$ tight problems
  — density: sparse $\cdots$ dense problems
  — interchangeability level: Induced Domain Fragmentation `IDF`

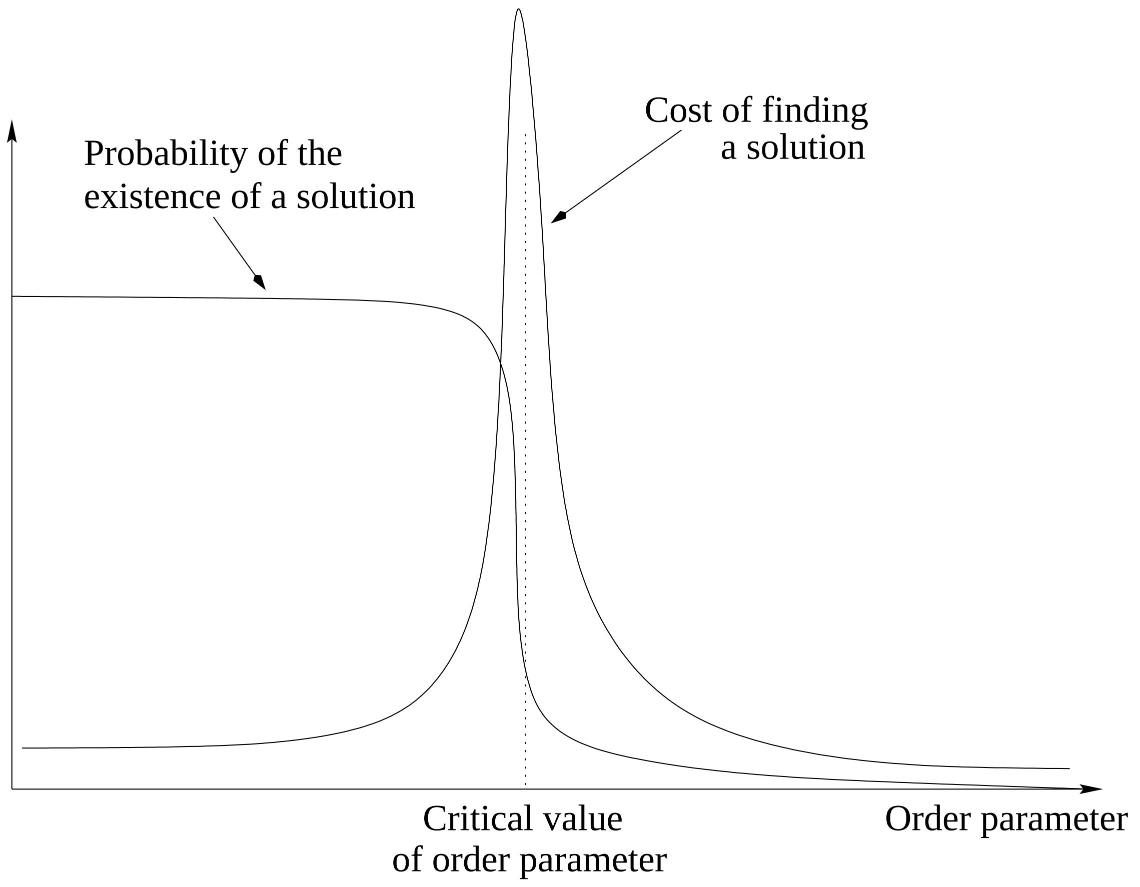


[Davis et al. 2001]

- Nodes visited, constraint checks, CPU time, size of bundle

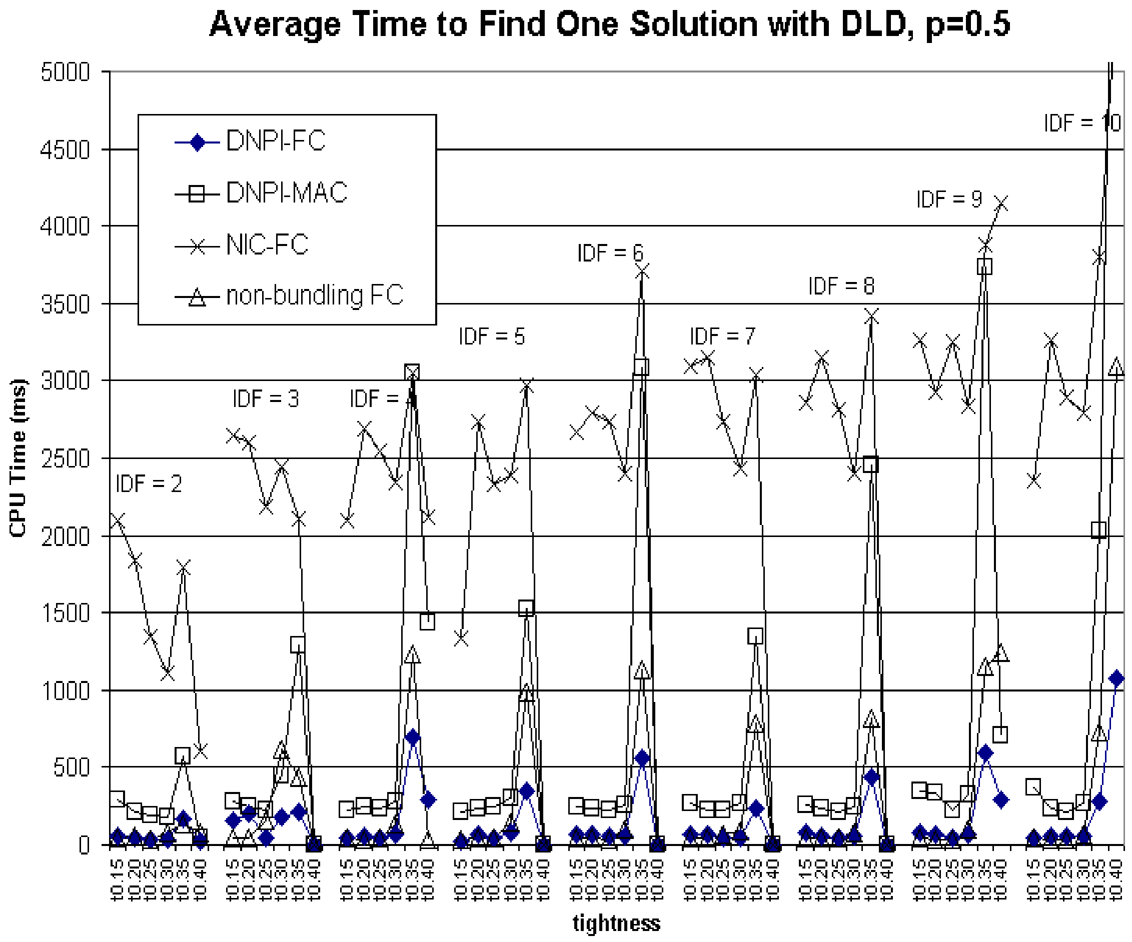
Phase Transition

[Cheeseman *et al.* 1991]



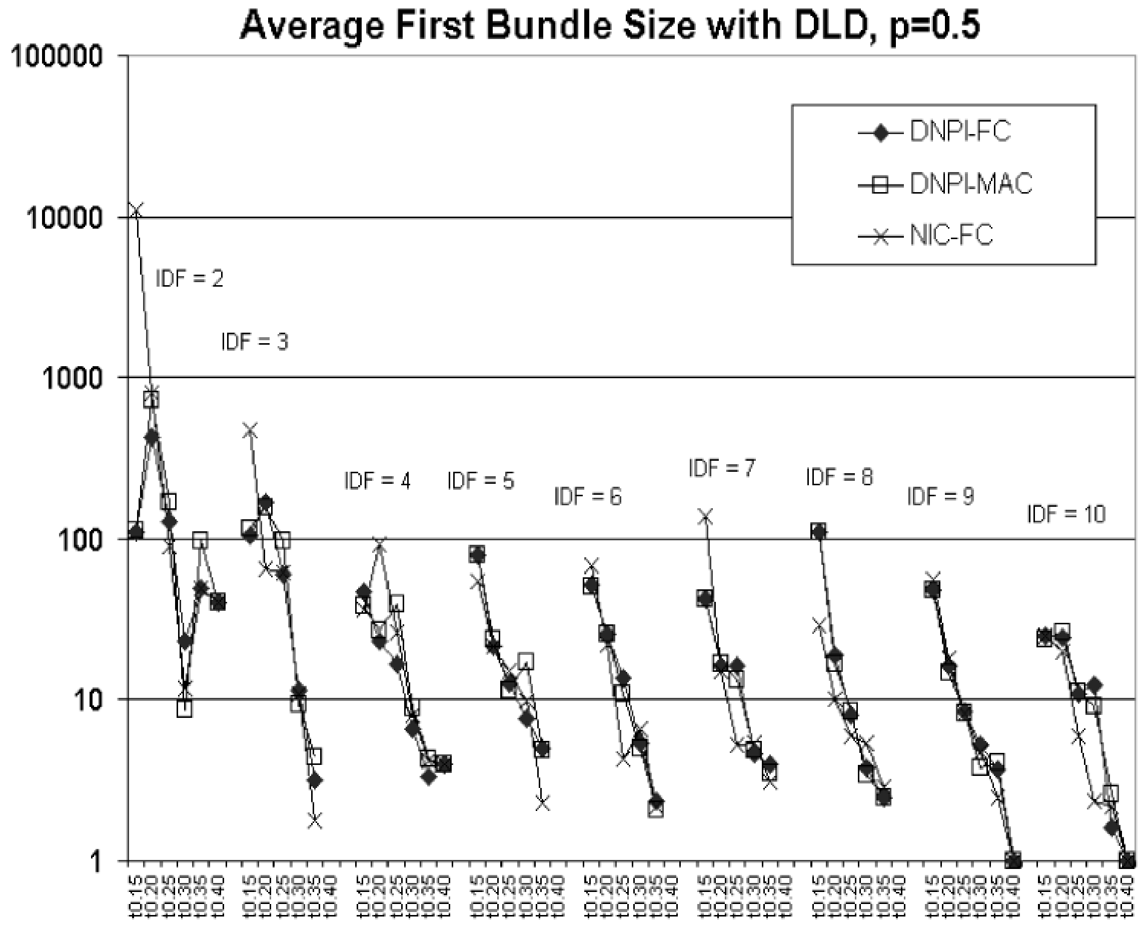
Novelty: reduce phase transition while finding multiple robust solutions

Evidence: time (varying: tightness, `IDF`)



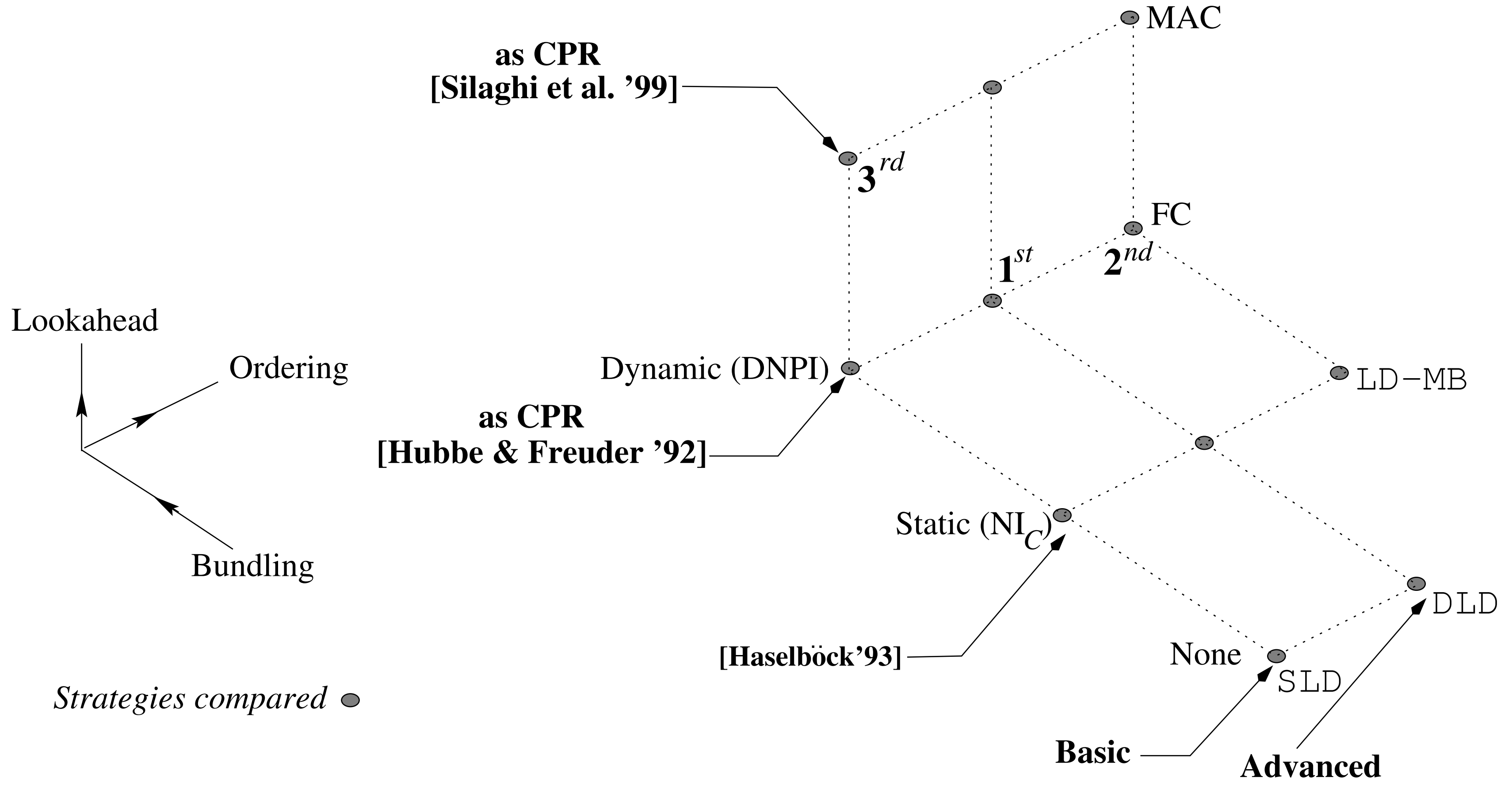
Novelty: reduce phase transition while . . .

Evidence: first bundle size (varying: tightness, `IDF`)



Novelty: … while finding multiple robust solutions

Ranking of strategies



Conclusions

1. **Lookahead:**
   MAC visits fewer nodes than FC, but requires in general more `CC` and more CPU time, especially in dynamic orderings (serious inconvenient)
   $\longrightarrow$ Unless using `SLD`, which one rarely does, MAC is not worth the effort and FC should be used

2. **Phase transition:**
   Dynamic bundling is uniformly worthwhile and reduces the spike at the cross-over point
   $\longrightarrow$ Less effort for more solutions, even for the hardest instances

Additional slides

Solving a CSP

Basic method
— Backtrack search, exponential

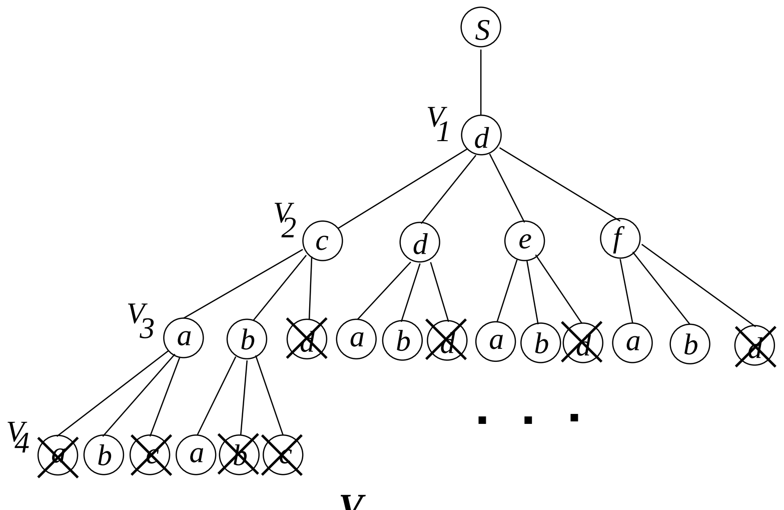
Most effective improvements
— Lookahead filtering
— Dynamic variable-value ordering

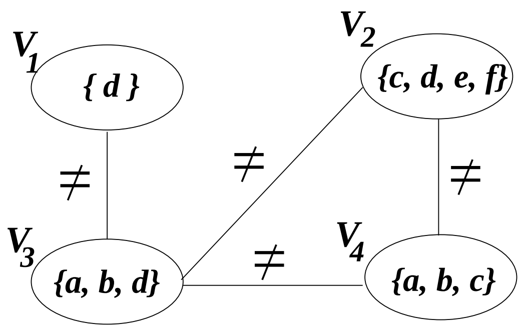
Exploit structure (problem specific)
— Real-life problems have a non-random structure
— Example: topology of graph, semantics of constraints
— **Our choice: symmetry**

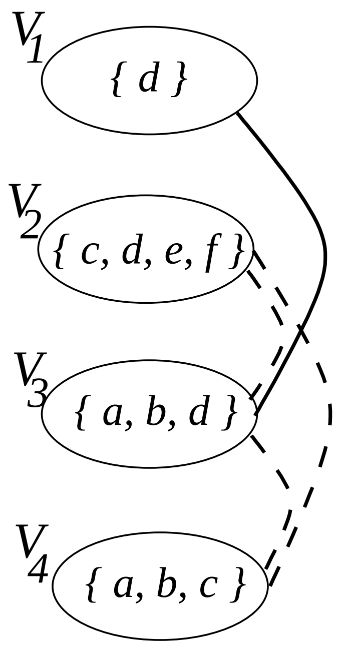
Lookahead Filtering

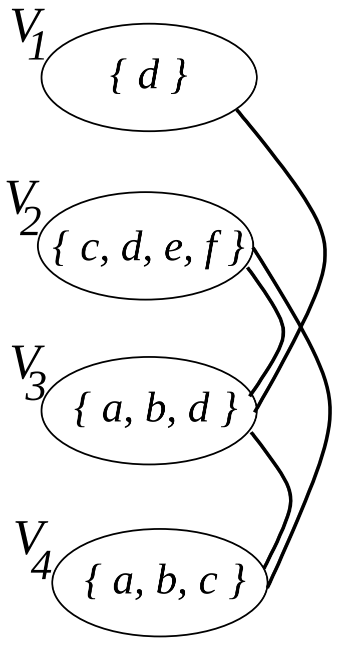
- Partial lookahead: Forward checking (FC)
- Full lookahead: Maintaining arc consistency (MAC)

[Sabin and Freuder, 1994]









Observation 0.1 *The curves for constraint checks* (`CC`) *and CPU time are often similar in shape and differ from that for* `NV`, *suggesting that constraint checks dominate the computational cost in our implementation.*

Observation 0.2 *DNPI-MAC always visits fewer nodes* (`NV`) *than DNPI-FC.*[a]

Observation 0.3 *DNPI-MAC in general requires more constraint checks* (`CC`) *than DNPI-FC. This effect always holds under dynamic orderings* (`DLD` *and* `LD-MB`) *where DNPI-MAC performs particularly poorly.*

[a]This observation holds for the *average* values reported in our graphs, however we detected a single anomaly as mentioned.

When constraint checks, and not nodes visited, dominate computation cost, DNPI-FC performs better than DNPI-MAC. Thus, the advantage in fewer nodes visited does not translate into saved time, yielding:

Observation 0.4 *Either because of its high cost in CPU time (which, in our implementation, seems to reflect more the effort spent on checking constraints than that spent on visiting nodes), or because the advantages of DNPI-MAC in terms of* `NV` *does not balance out the loss for constraint checks, DNPI-MAC is more costly than DNPI-FC. This tendency is aggravated under dynamic orderings where the performance of MAC further deteriorates.*

Observation 0.5 *The solution bundle found by DNPI-MAC is in general not significantly larger than that found by FC and does not justify the additional computational cost.*

Observation 0.6 *The magnitude and steepness of the phase transition increases proportionally with p, in accordance with the experiments reported in* [Phase Transitions and Complexity AIJ96].

Observation 0.7 *Although dynamic bundling does not completely eliminate the phase transition, it dramatically reduces it.*

Observation 0.8 `DLD` *orderings are generally less expensive than* `SLD` *orderings for all search strategies and yield larger bundles.*

Observation 0.9 `DLD` *orderings are also generally less expensive than* `LD-MB` *for dynamic bundling but similar for static bundling. However,* `LD-MB` *orderings produce larger bundles.*

Observation 0.10 `LD-MB` *orderings are generally less expensive than* `SLD` *orderings for all search strategies and yield larger bundles.*

Observation 0.11 *The bundle sizes of all bundling strategies are comparable, thus their respective advantages are better compared using other criteria.*

Observation 0.12 *DNPI-MAC is effective in reducing the nodes visited* `(NV)` *at the phase transition.*

Observation 0.13 *DNPI-MAC does not significantly reduce the overall cost at the phase transition.*

Observation 0.14 *In static orderings, the reduction of the phase transition due to the use of MAC seems to be more significant than that due to the use of dynamic bundling.*[a]

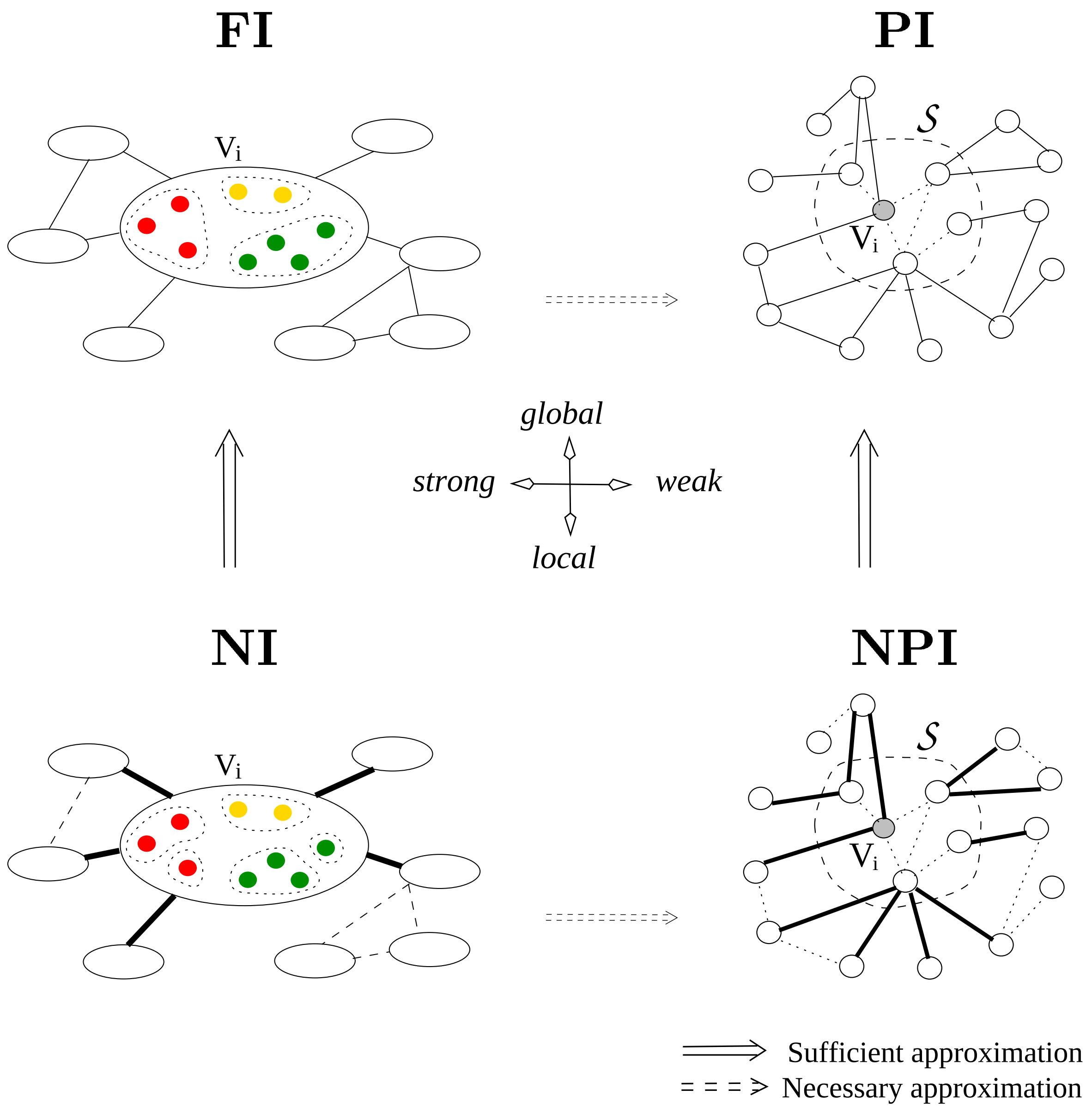
Observation 0.15 *Static bundling (NI_C) is expensive in general and we identify no argument to justify using it in practice. Further, under dynamic orderings, its high cost extends beyond the critical area of the phase transition to the point of almost concealing the spike.*[b]

Observation 0.16 *In dynamic orderings, DNPI-FC is a clear 'champion' among all strategies with regard to cost (i.e., constraint checks and CPU time).*

[a]We stress that this effect is reversed in dynamic orderings.

[b]The high cost of NI_C in the zone of 'easy' of problems is linked to the overhead of pre-computing interchangeability prior to search while many solutions exist.

Control parameters $\begin{cases} \text{choice of } \mathcal{S} \\ \text{neighborhood of } \mathcal{S} \end{cases}$



FI, PI (likely) intractable & PI is cheaper

NI, NPI polynomial & NPI is cheaper