Dynamic Bundling: Less Effort for More Solutions

Berthe Y. Choueiry and Amy M. Davis

Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
{choueiry | amydavis }@cse.unl.edu

Abstract. Bundling of the values of variables in a Constraint Satisfaction Problem (CSP) as the search proceeds is an abstraction mechanism that yields a compact representation of the solution space. We have previously established that, in spite of the effort of recomputing the bundles, dynamic bundling is never less effective than static bundling and non-bundling search strategies. Objections were raised that bundling mechanisms (whether static or dynamic) are too costly and not worthwhile when one is not seeking *all* solutions to the CSP. In this paper, we dispel these doubts and empirically show that (1) dynamic bundling remains superior in this context, (2) it does not require a full lookahead strategy, and (3) it dramatically reduces the cost of solving problems at the phase transition while yielding a bundle of multiple, robust solutions.

1 Introduction

Many problems in engineering, computer science, and management are naturally modeled as Constraint Satisfaction Problems (CSPs), which is, in general, **NP**-complete. Backtrack search remains the ultimate mechanism for solving such problems. One important mechanism for enhancing the performance of search is the exploitation of symmetries in the problem or a particular instance of it. From a practical perspective, the exploitation of symmetry can be used both to reduce the size of the search space and, more importantly, to represent the *solution space* in a compact manner by identifying families of qualitatively equivalent solutions, as we argued which is useful in practical applications [6].

In this paper we study the following two issues: (1) the combination of the dynamic computation of symmetries during search with the currently most popular lookahead strategy, and (2) the effect of symmetry detection on the presence and severity of the phase-transition phenomenon believed to be inherent to **NP**-complete problems. Our results are two fold. First, in accordance with [11], we dispel the growing myth that the aggressive lookahead strategy known as Maintaining Arc Consistency (MAC) [17] is always beneficial. Second, we establish that the dynamic detection and exploitation of symmetries during search, which results in multiple robust solutions, does not impede the performance of search but is actually a cost-effective tool for dramatically reducing the cost peak at the

phase transition, possibly the most critical phenomenon challenging the efficient processing of combinatorial problems in practice.

2 Background and motivation

Glaisher [13], Brown et al. [3], Fillmore and Williamson [9], Puget [16] and Ellman [8] proposed to exploit *declared* symmetries among values in the problem to improve the performance of search. The first four papers considered *exact* symmetries only, and the latter proposed to include also necessary and sufficient *approximations* of symmetry relations. Freuder [10] introduced a classification of various types of symmetry, which he called interchangeability. While all prior approaches focused on declared symmetry relations, Freuder proposed an efficient algorithm that *discovers* an exact but local form of interchangeability, neighborhood interchangeability (NI). NI partitions the domain of a given variable into a set of equivalence classes of values. Haselböck [14] simplified NI into a weaker form that we call *neighborhood interchangeability according to one constraint* (NI_C). Further, he showed how to exploit NI_C advantageously during backtrack search by generating solution *bundles*. Every solution bundle is a set of robust solutions [12]: Any value for a variable can be safely replaced by another value for the variable taken from the same domain bundle without altering the assignments of the remaining variables. Since the strategy devised by Haselböck computes domain partitions in a pre-processing step prior to search, we call this strategy *static bundling* and denote it NIC-FC. We proposed [7] a weak form of NI, namely *neighborhood partial interchangeability* (NPI) that can be controlled to compute interchangeability anywhere between, and including, NI and NI_C, see Fig. 1. In [1, 2] we proposed to recompute NPI relations *dynamically* during

Fig. 1. *Three types of neighborhood interchangeability.*

search yielding a new type of interchangeability we called *dynamic neighborhood partial interchangeability* (DNPI). We call this *dynamic bundling* and the search strategy DNPI-FC. While restricting our investigations to forward checking (FC) as a lookahead strategy during search, we established the superiority of dynamic bundling (DNPI-FC) over both static bundling (NIC-FC) and non-bundling search (FC) in terms of the criteria that assess the search effort and the 'compaction' of the solution space. These *theoretical* results hold when looking for all solutions and using a static variable ordering.

3 Our study and expectations

There has been a misconception that the cost of bundling in general and that of dynamic bundling in particular require too much overhead when one is only looking for a *first* solution. Indeed, the following two erroneous impressions prevailed: (1) when looking for a first solution, bundling–whether static or dynamic–is not

worth the effort, and (2) all the more reason, dynamic bundling is an overkill. We showed empirically[1] that the above stated superiority of dynamic bundling holds almost always in practice when looking for one solution and under various ordering heuristics [2]. For lack of space, we cite here only a few of the ones we investigated: (1) *Static ordering* (`SLD`): Variables are ordered statically before search according to the least domain heuristic. (2) *Dynamic variable ordering* (`DLD`): Variables are ordered dynamically during search according to the same heuristic. And (3) *Dynamic variable-value ordering:* In the context of bundling, a value is in fact a bundle. For this purpose, we proposed a new strategy, Least Domain-Max Bundle [2] (`LD-MB`) that chooses, dynamically during search, the variable of smallest domain (as in `DLD`) and, for this variable, the largest bundle in its domain. This heuristic[2] proved to be superior to other heuristics [5].

The surprising performance of dynamic bundling is explained by the fact that, while bundling partial solutions, it also *factors out the no-goods*, thus effectively pruning the search space. The two questions below remained unanswered:

1. How well does dynamic bundling combine with the most aggressive and popular, lookahead strategy MAC?
2. How does dynamic bundling affect the spike of the problem-solving cost at the phase transition identified in [4] and extensively studied in [15]? The cost and behavior of bundling on the cross-over point (i.e., critical area of the order parameter [4]) is the ultimate for its practical utility.

We concentrate on finding the *first* solution (bundle) *where we expect the cost of bundling to hinder seriously the performance of search*. To answer these questions, we conduct the experiments summarized below:

Question	Values reported	Ordering
Dynamic bundling: MAC vs. FC	Ratio of DNPI-MAC to DNPI-FC	`SLD`, `DLD`, `LD-MB`, Fig. 3
Effects of bundling on the phase transition	DNPI-FC, DNPI-MAC NIC-FC, (non-bundling) FC	`SLD`, Fig. 4 `DLD`, Fig. 5 `LD-MB`, Fig. 6

Below, we discuss the context of these two questions and list our expectations. Then, in Section 4, we present our experiments, provide a list of observations that summarize our findings, and finally discuss these results in detail.

3.1 Lookahead strategies

Sabin and Freuder [17] introduced a procedure to Maintain Arc Consistency (MAC), and advised to use this full-lookahead strategy instead of the popular partial-lookahead strategy known as forward checking (FC). While FC propagates the effects of a variable assignment to connected future variables, MAC propagates these effects over the entire remaining (future) constraint network. This stronger filtering of MAC has the potential to be particularly advantageous

[1] Under a wide variety of testing conditions and for problems in which we finely controlled the amount of interchangeability embedded in a given problem instance.
[2] Note that for finding all solutions `LD-MB` collapses to `DLD`.

when coupled with dynamic bundling. We call this new search strategy DNPI-MAC. A similar study was independently conducted by Silaghi et al. in [18], who coupled MAC with the Cross Product Representation (CPR)[3]. They consider two different implementations of MAC, and show that they are comparable. They test *only* CPR and *only* MAC omitting (1) whether or not MAC is beneficial and (2) a comparison of CPR with static and non-bundling search strategies. We seek to quantify the value of adding MAC to dynamic bundling.

We first study dynamic bundling in combination with MAC and with FC under various ordering heuristics. In Section 3.2 we further compare them (i.e., DNPI-MAC and DNPI-FC) to non-bundling and static bundling strategies with the goal of studying their behavior at the phase transition. We anticipate that the integration of DNPI and MAC will fulfill the expectations discussed below.

Expectation 1. *Since MAC performs a stronger pruning than FC, DNPI-MAC should not visit more nodes than DNPI-FC does.*

Indeed any value that is pruned by MAC and not pruned by FC is an additional node that FC examines. Further, it is guaranteed to fail and it results in extra useless work for the FC search strategy. We suspect that Expectation 1 could stand as a theorem *under static ordering*. It is supported by strong empirical evidence in Section 4.2, Fig. 3 and Observation 2, which relate average values over a pool of 6040 random problems. However, a careful examination of the results uncovered a *single* exception that we have not yet resolved.

Expectation 2. *DNPI-MAC should generate larger bundles than DNPI-FC and every bundle of DNPI-FC should be a subset of a bundle of DNPI-MAC under the same static variable ordering. The following expression should hold:*

$$\texttt{SB}(\textit{FC}) \geq \texttt{SB}(\textit{NIC-FC}) \geq \texttt{SB}(\textit{DNPI-FC}) \geq \texttt{SB}(\textit{DNPI-MAC}) \qquad (1)$$

where SB *is the number of solution bundles found. When finding only the first solution, Equation (1) suggests a statement about the First Bundle Size (*FBS*) (when* SB *is small, bundles are large). Thus we anticipate the following:*

$$\texttt{FBS}(\textit{FC}) \leq \texttt{FBS}(\textit{NIC-FC}) \leq \texttt{FBS}(\textit{DNPI-FC}) \leq \texttt{FBS}(\textit{DNPI-MAC}) \qquad (2)$$

Expectation 3. *Because of the above, we are tempted to infer that DNPI-MAC should be computationally cheaper than DNPI-FC and perform better bundling.*

3.2 Phase transition

Cheeseman et al. [4] presented empirical evidence of the existence of a phase transition phenomenon in the cost of solving **NP**-complete problems when varying an order parameter. In particular, they showed that the location of the phase transition and its steepness increase with the size of the problem[4], thus yielding

[3] Silaghi et al. erroneously claim that CPR was proposed as DNPI. We showed independently that CPR and DNPI yield the same bundling while DNPI visits fewer nodes. The difference is polynomially bounded, as suggested by a reviewer.

[4] Problems in **P** do not in general contain a phase transition, or if they do, the cost at the transition bounded and is not affected by an increase of problem size [4].

a new characterization of this important class of problems. We now examine the effect of bundling (statically and dynamically) on the phase transition. Because problems at the cross-over point are acknowledged to be probabilistically the most difficult to solve, determining the performance of our algorithms in this region is important. So far, we have not found a scenario where dynamic bundling is any hindrance to the performance of search, both for all solutions and for one solution. It is further aided by dynamic variable ordering, but *not* by MAC as discussed in Observations 2 and 3. In these experiments with the phase transition, we prepare the most adverse situation that we know of for our algorithms. However, given the ability of dynamic bundling in pruning 'bundled' no-goods, we anticipate the following:

Expectation 4. *Dynamic bundling will significantly reduce the steepness of the phase transition, that is, the problems in the phase transition will be easier to solve with dynamic bundling.*

4 Experiments

In this section, we examine the above expectations through empirical tests. We used the random generator for binary CSPs described in [2]. This generator allows us to control the level of interchangeability embedded in an instance of a CSP by controlling the number of equivalence classes of values induced by every constraint on the domain of one of the variables in its scope. We call this number the induced domain fragmentation `IDF`. Fig. 2 shows a constraint C with an `IDF`=3. For each measurement point, we generated 20 instances with

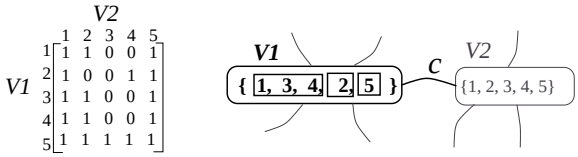

Fig. 2. *Left:* Constraint as a binary matrix. *Right:* Domain of V_1 partitioned by C.

the following characteristics: number of variables $n = 20$; domain size $a = 10$; induced domain fragmentation `IDF` $= [2, a]$ by a step of 1; tightness (ratio of number of allowed tuples over that of all possible tuples) of any constraint $t = [0.15, 0.85]$ with a step of 0.05; and constraint probability (ratio of the number of constraints in the CSP over that of all possible constraints) p= 0.5 and 1.0.

In order to reduce the duration of our experiments to a reasonable value, we chose to make *all* problems arc-consistent (AC-3) before search begins. This is done uniformly in all experiments and for all strategies and does not affect the quality of our conclusions. We compute, at each data point, both the average and the median values of the following *evaluation criteria*: number of nodes visited (`NV`); number of constraint checks (`CC`); CPU time in msec with a clock resolution of 10 msec; and size of the first bundle found (`FBS`). The horizontal axis denotes various tightness values t= $0.15 \cdot 0.4$. Problems beyond $t = 0.4$ were determined unsolvable by the preprocessing step and thus required no search. We notice

that the average and median curves almost always have the same shapes (except for one case discussed below). Our experiments yield a number of observations summarized in Section 4.1 and discussed in detail in Sections 4.2 and 4.3.

4.1 List of observations

We first state a general observation (Observation 1), then observations regarding the effect of lookahead strategies (Observations 2 to 5) and finally observations about the effect on the phase transition (Observations 6 to 16).

Observation 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 2. *DNPI-MAC always visits fewer nodes* `(NV)` *than DNPI-FC, in confirmation of Expectation 1.*[5]

Observation 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.*

When constraint checks, and not nodes visited, dominate computation cost, DNPI-FC performs better than DNPI-MAC. Thus, contrary to Expectation 3, the advantage in fewer nodes visited does not translate into saved time, yielding:

Observation 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 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 6. *The magnitude and steepness of the phase transition increases proportionally with p, in accordance with the experiments reported in* [15].

Observation 7. *Although dynamic bundling does not completely eliminate the phase transition, it dramatically reduces it.*

Observation 8. `DLD` *orderings are generally less expensive than* `SLD` *orderings for all search strategies and yield larger bundles.*

Observation 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 10. `LD-MB` *orderings are generally less expensive than* `SLD` *orderings for all search strategies and yield larger bundles.*

[5] This observation holds for the *average* values reported in our graphs of Fig. 3, however we detected a single anomaly as mentioned in Section 3.1.

Observation 11. *The bundle sizes of all bundling strategies are comparable, thus their respective advantages are better compared using other criteria.*

Observation 12. *DNPI-MAC is effective in reducing the nodes visited* (NV) *at the phase transition.*

Observation 13. *DNPI-MAC does not significantly reduce the overall cost at the phase transition.*

Observation 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.*[6]

Observation 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.*[7]

Observation 16. *In dynamic orderings, DNPI-FC is a clear 'champion' among all strategies with regard to cost (i.e., constraint checks and CPU time).*

4.2 Data analysis: MAC vs. FC

As stated above, we report the results for finding one solution bundle. Fig. 3 shows the *ratio* of the values of the evaluation criteria for MAC versus FC under dynamic bundling and for the three ordering heuristics SLD (⧫), DLD (□), LD-MB (×). For values above 1, the value of DNPI-MAC is higher than the value of FC, and vice versa. Below we discuss each row:

Nodes visited (row 1): The value of the ratio is consistently below 1 across ordering strategies, IDF values, and p values. This indicates that MAC always visits fewer nodes than FC and supports Observation 2. Note that this effect becomes more pronounced as the constraint probability increases from 0.5 (left column) to 1.0 (right column) and as tightness increases (shown by the downward slope of the lines).

Constraint checks (row 2): The non-null values (except for a few black diamonds that we discuss below) are all above 1, indicating that FC is superior to MAC (Observation 3). Now let's look at the few cases where DNPI-MAC outperforms DNPI-FC (ratio < 1). This sometimes happens, almost exclusively under the static ordering SLD (⧫). It happens for: (1) p=0.5, t is large, and for all values of IDF, and (2) p=1.0 and IDF is small. This provides an interesting result, because DNPI-MAC is sensitive to IDF where DNPI-FC is insensitive (specifically, when constraint probability is high). As IDF increases, we see that DNPI-MAC loses the edge it had when more interchangeability was present. Note also that when using dynamic variable ordering such as DLD (□) or LD-MB (×), DNPI-FC is a clear winner over DNPI-MAC. When we use dynamic variable ordering, DNPI-MAC checks from 3 to 13 times as many constraints as

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

[7] 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.

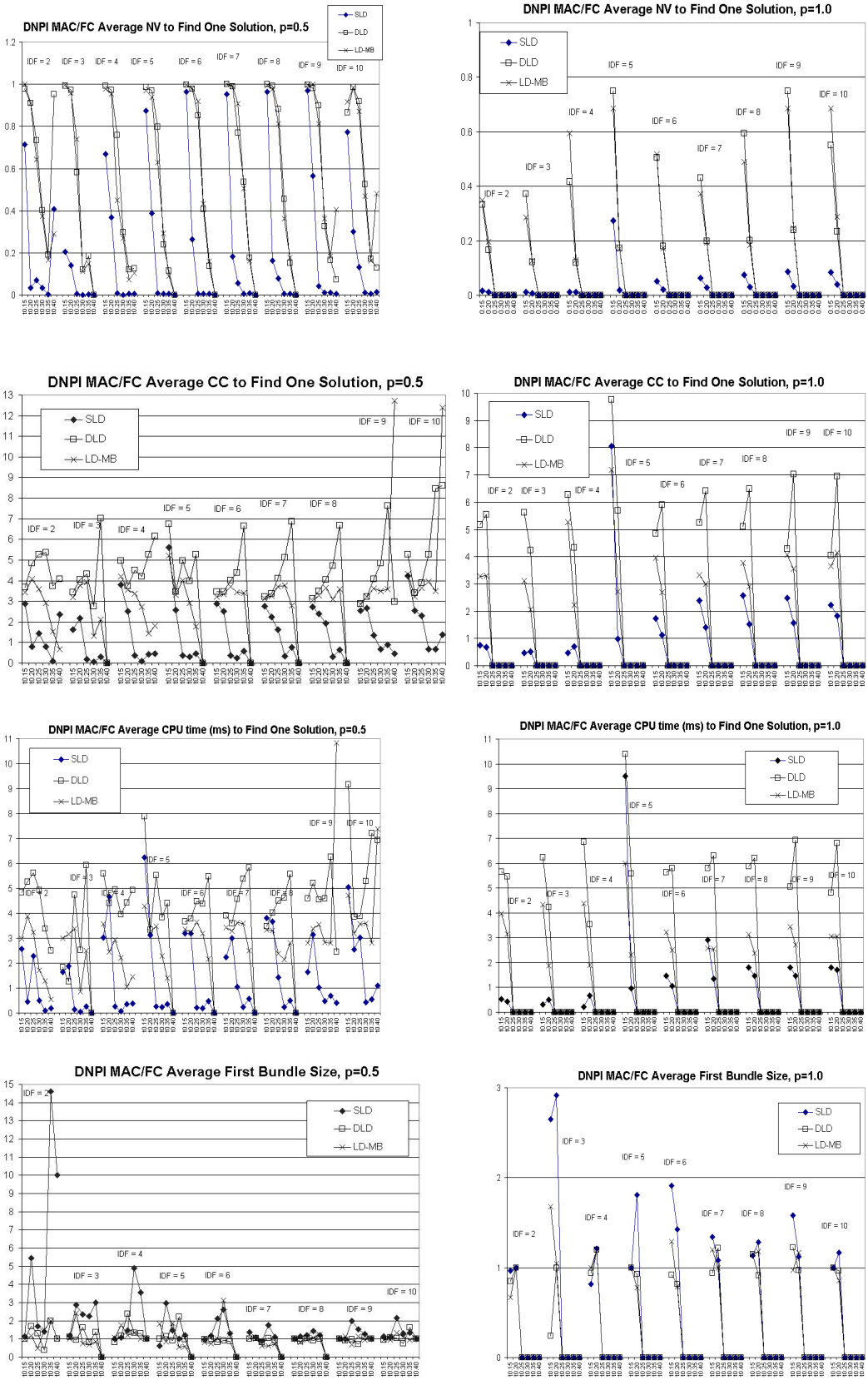

Fig. 3. *Comparing MAC and FC in the context of DNPI.*

DNPI-FC. This supports Observation 3 and is a *clear indication of an expense in MAC that is not compensated by dynamic bundling.*

CPU time (row 3): The advantage of DNPI-FC over DNPI-MAC noted in the constraint checks is reinforced by the CPU time data. Thus the use of DNPI-MAC (except for a few cases, mostly in `SLD`) is *detrimental* to the performance of search despite the savings of nodes visited (Observation 4).

Size of first bundle (row 4): Finally, we look at the bottom row of Fig. 3 to check whether or not the additional propagation effort of MAC benefits the size of the bundle found (Observation 5). We see that, and in accordance with Expectation 2, DNPI-MAC does generate slightly larger bundles than DNPI-FC. For p=0.5, the bundle comparisons huddle mostly just above 1. This means that the bundle sizes are comparable, with DNPI-MAC generally producing bundles that are just a little bit larger. We note two extreme behaviors:

1. At `IDF`=2, t=0.40, the bundle produced by DNPI-MAC is fifteen times larger than that of DNPI-FC (⧫). Additionally, this much larger bundle took less time to find. This demonstrates and justifies the advantage of DNPI-MAC. Note however that this extent of divergence between DNPI-MAC and DNPI-FC does not hold when `IDF`$>$ 2. Indeed, for p=1.0, the bundles of DNPI-MAC are never more than three times larger than those of DNPI-FC, but are frequently smaller (especially under dynamic variable ordering).
2. At `IDF`=4 and t=0.15, the bundle produced by DNPI-MAC is smaller than that of DNPI-FC (⧫). This is the only major opposition to our Expectation 2. (There are other small exceptions, where DNPI-FC produces a bundle that is 1 or 2 solutions larger than DNPI-MAC's bundle. We traced all these exceptions to the non-deterministic value ordering.) We traced this violation of Expectation 2 to a single problem where DNPI-MAC found a bundle of size 84 and DNPI-FC found a bundle of size 168. This happens for the single problem mentioned above, which we cannot yet justify.

Conclusions of the data analysis: The cost of DNPI-MAC is neither systematically nor predictably worthwhile, except possibly under `SLD` ordering. This tendency becomes even stronger under dynamic orderings `DLD` and `LD-MB`.

4.3 Data analysis: Dynamic bundling at the phase transition

These charts are arranged in Fig. 4, 5, and 6 for the ordering heuristics `SLD`, `DLD`, and `LD-MB`, respectively. Each graph shows four search strategies, namely non-bundling FC (△), static-bundling NIC-FC (×), dynamic bundling with forward checking DNPI-FC (⧫), and dynamic bundling with Maintaining Arc Consistency DNPI-MAC (□). Each figure has two columns: left for p=0.5 and right for p=1.0. We first state some global observations over the experiments as a group and across ordering heuristics. Then we examine the data for each of the three ordering heuristics in this order: `SLD`, `DLD`, then `LD-MB`. In each graph, we pay particular attention to the relative behavior of the search strategies at the phase transition, demonstrated here by the presence of a 'spike' in nodes visited, constraint checks and CPU time.

Global observations: The comparison of the left and right charts in Fig. 4, 5, and 6, confirms the findings in [15] on how the slope and importance of the phase transition augment with the constraint probability (Observation 6). The examination of all 24 graphs at the phase transition peak confirms that dynamic bundling dramatically reduces its magnitude (Observation 7). Finally, the comparison of graphs for CPU time and bundling power, interpreted as first bundle size across ordering strategies, shows that `DLD` is consistently an excellent ordering unless one is specifically seeking larger bundles at the expense of a slight increase in cost in which case `LD-MB` is justified (Observations 8, 9, and 10).

4.3.1 DNPI at phase transition: static ordering

Recall from Observation 4 and Section 4.2 that DNPI-MAC performs best under `SLD` ordering. Under dynamic orderings (`DLD` and `LD-MB`), it is non-competitive.

Nodes visited (`NV`) with `SLD` (row 1): Fig. 4 shows that strategies based on dynamic bundling (□ and ⧫) expand in general fewer nodes than strategies based on non-bundling (△) or static bundling (×) (Observation 7). A careful examination shows that the phase transition is indeed present for DNPI-FC (to some extent) and for NIC-FC and FC (to a large extent) but seemingly absent in DNPI-MAC (Observation 12). Recall that MAC almost always visits fewer nodes than DNPI-FC, which is guaranteed to visit fewer nodes than the others *when finding all solutions*. We see here that MAC expands by far the fewest nodes in the phase transition. It seems to almost not have a phase transition at all. A closer inspection of the data shows that a phase transition is present, even in DNPI-MAC, but is 3 to 4 *orders of magnitude* smaller than the competing methods. DNPI seems to benefit very much from the pairing with MAC in `SLD`. However, this saving on the number of nodes visited does not extend to the CPU time as noted in Observations 1 and discussed below.

Constraint checks (`CC`) with `SLD` (row 2): At the left column (p=0.5), we notice that, in general, either FC (△) or NIC-FC (×) performs the most constraint checks. In particular, the performance of static bundling (×) is quite disappointing, see `IDF`= 7, 9 for p=0.5 (Observation 15). Moreover, DNPI-MAC performs the fewest constraint checks at every phase transition except when `IDF`=10. This illustrates an advantage that justifies the use of MAC. Further, dynamic bundling remains the winning strategy as stated in Observation 7.

Recall that the 'traditional' fear of dynamic bundling is the excessive number of constraint checks it requires to compute the interchangeability sets. We show here that, in the most critical region, it is the *non-bundling and static bundling strategies that are actually requiring the most constraint checks*. We are now confident to recommend the use of dynamic bundling in search not only to output several interchangeable solutions (which is useful in practice and the main goal of bundling) but moreover to reduce the severity of the phase transition. This result becomes even more significant under dynamic orderings (see Fig. 5 and 6). In the right column (p=1.0), this tendency is less pronounced and all strategies seem to perform fairly similarly: none of them consistently performing fewer or more constraint checks than any other. Nevertheless, the behavior of

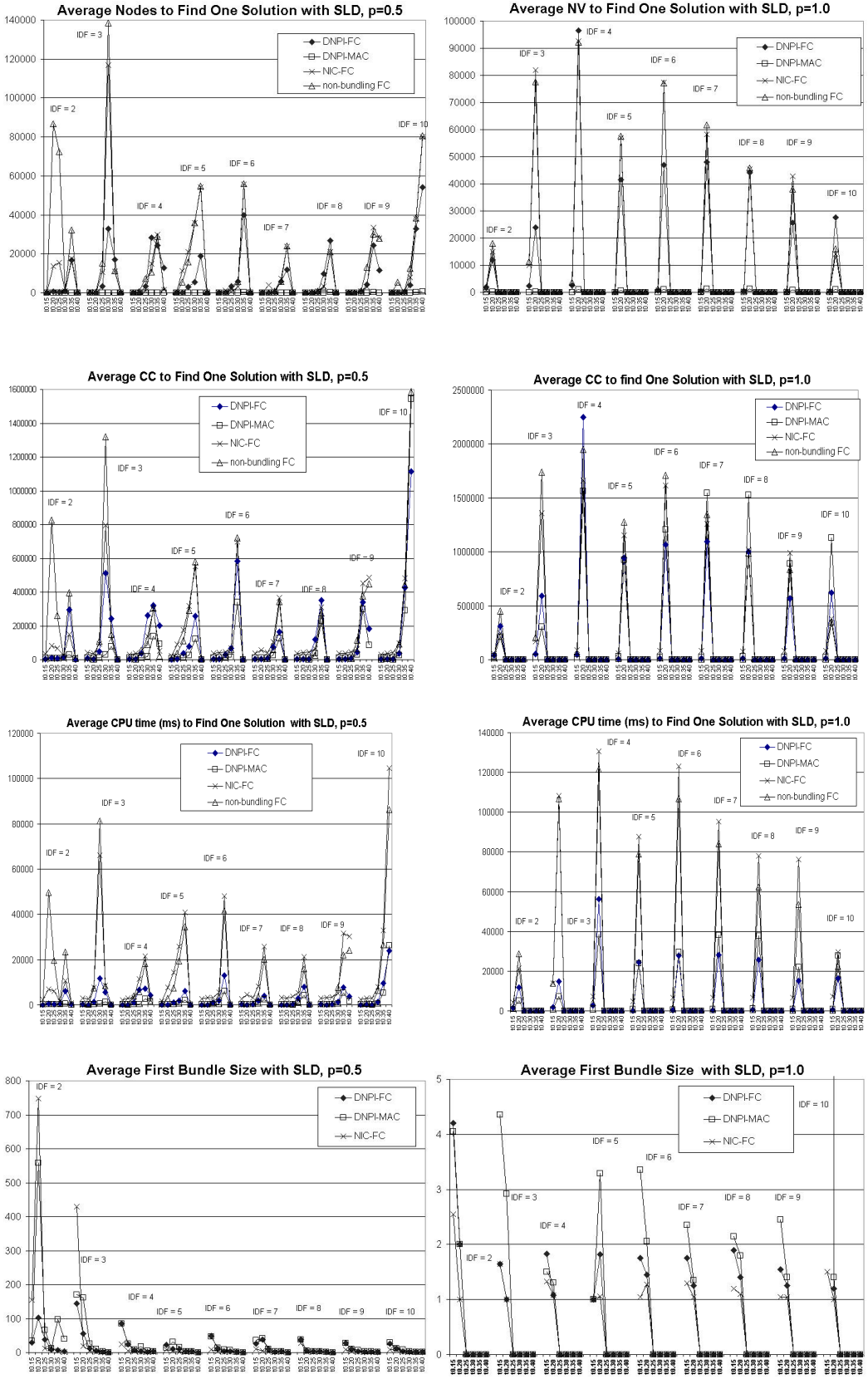

Fig. 4. *Bundling with static variable ordering.*

dynamic bundling never deteriorates the performance of search enough to make it impractical.

CPU time with SLD *(row 3):* The graphs for CPU time bear quite a resemblance with that of constraint checks (Observations 1). However, the distance between the two dynamic bundling strategies and the two others is more clearly visible, in favor of dynamic bundling. Indeed, both dynamic bundling strategies DNPI-FC (⧫) and DNPI-MAC (□) are well below the other strategies in both graphs. This is likely thanks to the significant reduction in the number of nodes visited by these strategies and again supports the use of dynamic bundling to reduce the steepness of the phase transition. Both Observations 7 and 14 are supported here. In the left chart (p=0.5), DNPI-MAC (□) usually consumes the least CPU time. In the right chart (p=1.0), DNPI-MAC consumes more time than DNPI-FC for high IDF values. This is consistent with the behavior that we observed for constraint checks.

*First bundle size (*FBS*) with* SLD *(row 4):* We do not report the First Bundle Size FBS for non-bundling FC, since the solution size is either 1 (when a solution exists) or 0 (when the problem is unsolvable). In general, we find that the sizes of the first bundle found are comparable across strategies (Observation 11) with a few exceptions addressed below. For p=0.5 (left), we see that NIC-FC surprisingly performs the best bundling for low values of IDF (i.e., 2 and 3). When IDF increases to and beyond 4, dynamic bundling regains its advantage: DNPI-FC and DNPI-MAC compete for the larger bundle in most cases. However, for p=1.0 (bottom graph), DNPI-MAC nearly always performs the best bundling. Notice that these bundles are quite small, less than 5 solutions are contained in each. One exception is worth mentioning here at the right chart when IDF=10 and $t = 0.15$: the data point here is off the chart indicating an exceedingly large first bundle. This is effect is traced to a *single* instance of the 20 values averaged here and can traced to a weird, but correct, random problem in which 173 out of 190 constraints are identical.

Conclusions relative to SLD*:* Under static variable ordering, dynamic bundling, especially when coupled with MAC in DNPI-MAC, drastically reduced the phase transition for a CSP, making the most difficult instances easier to solve.

4.3.2 DNPI at phase transition: dynamic variable ordering

In general, search strategies with dynamic variable orderings (i.e., DLD and LD-MB) almost always perform better than statically ordered search strategies [2]. In this section we examine DLD and in Section 4.3.3 we will examine LD-MB. The results are presented in Fig. 5. The comparison of Fig. 5 and 4 shows that in general DLD indeed performs better that SLD (the CPU time scale decreases almost tenfold). Moreover, we see that dynamic variable ordering heuristics have a stronger effect on some strategies than others. Specifically, it seems to hurt DNPI-MAC while helping the other strategies. This justifies Observation 4.

*Nodes visited (*NV*) with* DLD *(row 1):* Similar to what we saw for SLD, DNPI-MAC (□) with DLD clearly visits fewer nodes than any other search strategy (Observation 12). Further, we see that the other three strategies DNPI-FC (⧫),

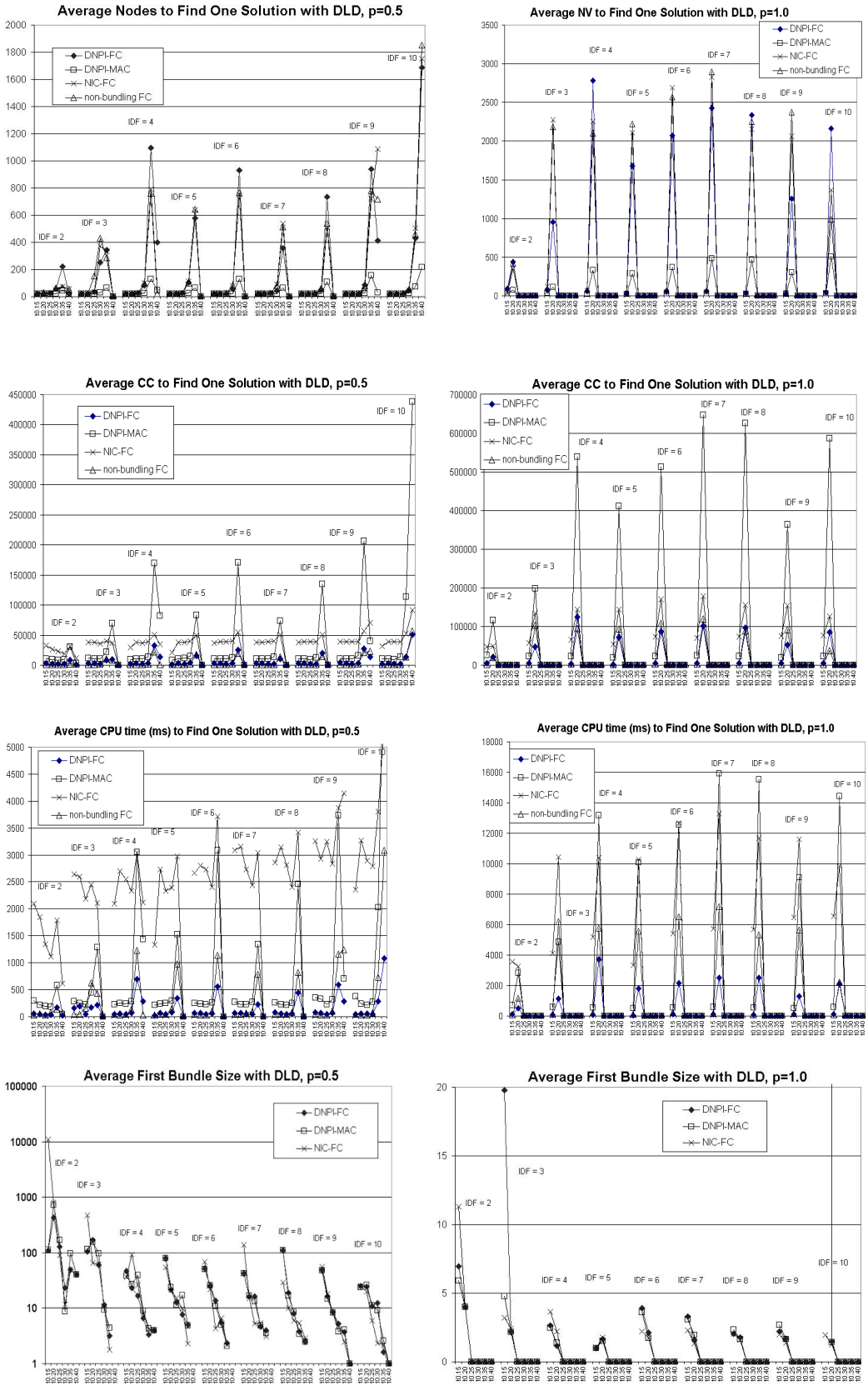

Fig. 5. *Bundling with dynamic variable ordering.*

NIC-FC (×) and FC (△) compete for visiting the most nodes. DNPI-FC performs the worst most frequently as shown in (p=0.5 at IDF = 2, 4, 5, 8 and 9) and (p=1.0 at IDF = 2, 4, 8 and 10). Fortunately, poor performance in nodes visited does not otherwise affect the other performance of DNPI-FC, which remains a champion (Observation 16).

*Constraint checks (*CC*) with* DLD *(row 2):* The data here discredits NIC-FC and DNPI-MAC and demonstrates that DNPI-FC is a champion under dynamic ordering. DNPI-MAC performs quite poorly with dynamic ordering, beginning with DLD in Fig. 5 and carrying over LD-MB in Fig. 6. In all cases, DNPI-MAC performs the most constraint checks at the phase transition (Observation 4). In all cases, DNPI-MAC performs the most constraint checks at the phase transition (Observation 4). *Therefore, we safely conclude that the large amount of checks performed by DNPI-MAC is due to the addition of MAC and not to dynamic bundling.* Further, we see that DNPI-FC (⧫) is quite effective. It clearly and significantly reduces the phase transition (Observation 7) and, in general, outperforms the other methods (Observation 16). Interestingly, the strongest competitor to DNPI-FC is FC (△) itself. Note however that FC gives one solution while DNPI-FC gives several robust ones. Neither NIC-FC (Observation 15) nor DNPI-MAC (Observation 4) is worthwhile: they *increase* the phase transition rather than decrease it. This justifies our argument in favor of dynamic bundling and confirms our doubts about the appropriateness of MAC in dynamic orderings.

CPU time with DLD *(row 3):* The charts here amplify the effects discussed above. The disadvantage of static bundling becomes apparent in the left chart, for p=0.5 (Observation 15). Though the phase transition is less steep (i.e., the spike is almost concealed), the overall cost of performing NIC-FC (×) search is unnecessarily high. This is due to the overhead of finding all NI_C interchangeabilities before beginning search. We can also see that DNPI-MAC (□) continues to perform poorly (Observation 4). DNPI-MAC is consistently more costly than DNPI-FC (⧫) and FC (△) even when not at the phase transition. When p=1.0, it takes more CPU time than even NIC-FC. Once again, DNPI-FC performs the best overall (Observation 16). In the right graph (p=1.0), we see that DNPI-FC (⧫) reduces the phase transition, and performs best at every phase transition (Observations 7 and 16).

*First bundle size (*FBS*) with* DLD *(row 4):* Finally, we see that though DNPI-MAC (□) puts forth much effort, it does not even produce the best bundles. In reality, NIC-FC (×) performed unexpectedly good bundling, especially where p=0.5. DNPI-FC (⧫) also bundles very well; it is even with DNPI-MAC in much of the left charts, and slightly better for most of the right charts (Observation 11).

Conclusions relative to DLD*:* Under dynamic variable ordering, DNPI-FC continues to perform better than non-bundling FC as bundling effectively reduces the phase transition. Further, the addition of MAC to DNPI is disastrous with a DLD ordering: it *increases* the amplitude of the spike of the phase transition (Observation 4). Similarly, NIC-FC behaves worse than FC overall under this ordering (though it finds large bundles).

4.3.3 DNPI at phase transition: dynamic variable-value ordering

The remaining results are shown in Fig. 6. Notice the absence of FC (non-bundling) search on these graphs. Since LD-MB is a strategy specific to *bundling*, a non-bundling search strategy such as FC makes no sense in this context. Therefore the comparisons drawn here are between the different bundling strategies. Recall that LD-MB is merely DLD with a bundle ordering enforced since it assigns to the variable chosen the largest bundle in the partition of its domain. Because of its similarity to DLD, it often generally performs as well as DLD, but produces a larger first bundle. Revisiting this strategy, we see that its effect on the phase transition (when combined with bundling) is also very similar to DLD.

Nodes visited (NV) with LD-MB (row 1): As for the other orderings, DNPI-MAC (□) visits fewer nodes than any other strategy for both values of p (Observation 12). As with DLD in Section 4.3.2, we see that DNPI-FC (⧫) often visits the most nodes. This may serve as a notice that, when looking for only one solution, if it is expensive to expand nodes but cheap to check constraints (here it is the opposite), then DNPI-MAC may be an appropriate choice, as highlighted in Observation 4.

Constraint checks (CC) with LD-MB (row 2): Here we see DNPI-MAC (□) checks significantly more constraints than either DNPI-FC (⧫) or NIC-FC (×) at the phase transition (Observation 4). However, notice that at most points, especially for a low p (left graph), NIC-FC (×) performs many more constraint checks than the others, almost concealing the phase transition (Observation 15). This again is due to the overhead of static interchangeability computation. This disadvantage is less visible with a high p, where NIC-FC performs more constraint checks for very loose problems ($t = 0.15$), but reduces the phase transition more effectively than either DNPI-FC or DNPI-MAC. The comparison of the second rows of Figs. 5 and 6 shows the following. NIC-FC (×) performs about the same number of constraint checks for LD-MB ordering than for DLD ordering (i.e., between 100,000 and 200,000 when p=1.0), but DNPI-FC (⧫) and DNPI-MAC (□) both perform more constraint checks in LD-MB than in DLD (Observation 9). This is a disadvantage of the combination of LD-MB with dynamic bundling. LD-MB requires more backtracking, because the largest bundle in a variable is often a bundle of no-goods, and will trigger backtracking. This will require a re-computation of interchangeability, and becomes more costly in general. Even when looking for only one solution, it seems that DLD is best suited to dynamic bundling. Further, the comparison of second rows in Figs. 4 and 6 confirms an obvious expectation of LD-MB ordering being less expensive and yielding better bundles than static variable ordering SLD (Observation 10).

CPU time with DLD (row 3): The trends we noted in the nodes visited and constraints checked for LD-MB consistently extend to the CPU time consumed. We see that both NIC-FC (×) and DNPI-MAC (□) have generally poor performance, requiring more time that DNPI-FC (⧫). This confirms Observations 15, 4, and 16. Also, the comparison of CPU time (third row) with the first two ordering heuristics (i.e., SLD in Fig. 4 and DLD in Fig. 5) confirm Observations 10 and 9, respectively.

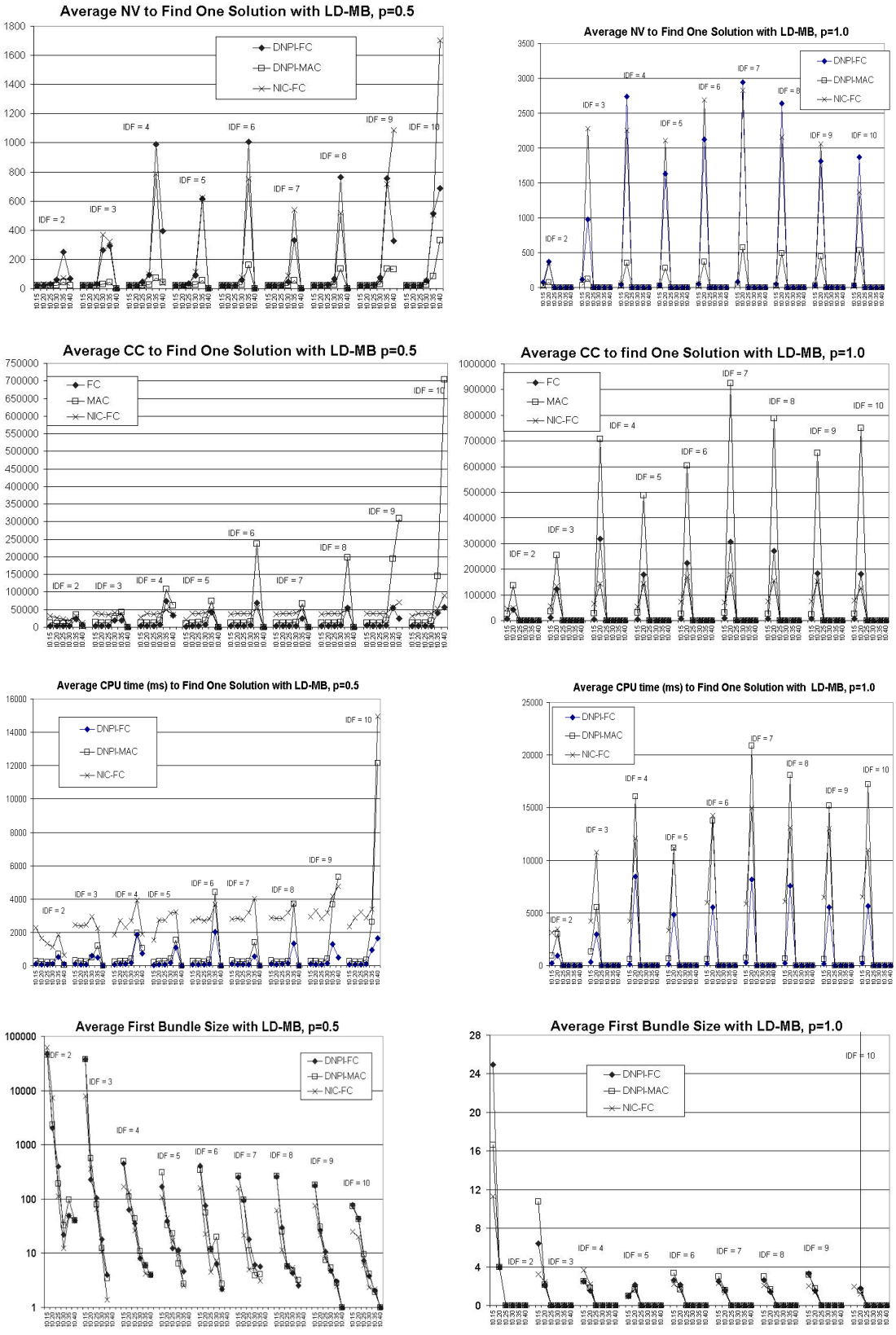

Fig. 6. *Bundling with dynamic variable-value ordering.*

*First bundle size (*FBS*) with* LD-MB *(row 4):* We see that, based on the size of the first bundle, no particular bundling strategy can be declared a *clear* winner. In general, dynamic bundling is a little stronger than static bundling, with two exceptions (p=0.5, IDF=2 and p=1.0, IDF=4). As far as lookahead strategies are concerned, DNPI-MAC and DNPI-FC are comparable and quite competitive with respect to their bundling capabilities, thus justifying again the power of the dynamic bundling and the superfluity of MAC (Observation 11). The comparison across ordering heuristics show that LD-MB yields better bundles than both SLD (Observation 10) and DLD (Observation 9)

Conclusions relative to LD-MB*:* It appears clearly that dynamic bundling *without* MAC, that is DNPI-FC, effectively reduces the phase transition and produces large bundles across all variable ordering heuristics. Further, we note that, in the phase transition, DLD seems to be the most effective ordering.

5 Conclusions

Regarding the lookahead strategy to use with dynamic bundling, we establish that, although MAC visits fewer nodes than FC, it requires in general more constraint checks and more CPU time. This is especially true in dynamic variable ordering (i.e., DLD and LD-MB), where the cost of MAC becomes a serious impediment. In conclusion, unless we are using SLD, DNPI-MAC is not worth the effort and DNPI-FC should be used instead.

Regarding the phase transition, we prove that dynamic bundling is uniformly worthwhile by reducing the spike at the cross-over point. This is especially true for DNPI-FC. DNPI-MAC is a good search strategy to reduce the phase transition if static ordering must be used, but if dynamic ordering is permitted, DNPI-FC is much more effective. As a conclusion, the phase transition is best reduced by DNPI-FC with DLD variable ordering, which has never before been implemented. These strategies, and their relative rank (1st, 2nd and 3rd) are summarized in Figure 7.

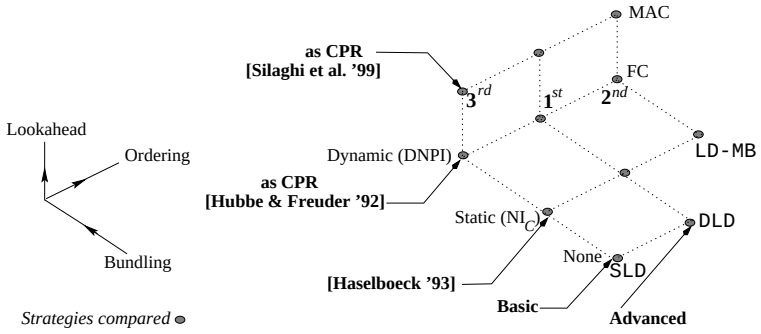

Fig. 7. *A summary of the strategies implemented and tested and the best ranking ones. All strategies not otherwise marked were proposed by us.*

References

1. Amy M. Beckwith and Berthe Y. Choueiry. On the Dynamic Detection of Interchangeability in Finite Constraint Satisfaction Problems. In T. Walsh, editor, 7^{th} *International Conference on Principle and Practice of Constraint Programming (CP'01)*, LNCS Vol. 2239, page 760, Paphos, Cyprus, 2001. Springer Verlag.
2. Amy M. Beckwith, Berthe Y. Choueiry, and Hui Zou. How the Level of Interchangeability Embedded in a Finite Constraint Satisfaction Problem Affects the Performance of Search. In *14th Australian Joint Conference on Artificial Intelligence. LNAI Vol. 2256*, pages 50–61, Adelaide, Australia, 2001. Springer Verlag.
3. Cynthia A. Brown, Larry Finkelstein, and Paul W. Purdom, Jr. Backtrack Searching in the Presence of Symmetry. In T. Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 99–110. Springer-Verlag, 1988.
4. Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proc. of the 12 th IJCAI*, pages 331–337, Sidney, Australia, 1991.
5. Berthe Y. Choueiry and Amy M. Beckwith. On Finding the First Solution Bundle in Finite Constraint Satisfaction Problems. Technical Report CSL-01-03. `consystlab.unl.edu/CSL-01-04.ps`, University of Nebraska-Lincoln, 2001.
6. Berthe Y. Choueiry, Boi Faltings, and Rainer Weigel. Abstraction by Interchangeability in Resource Allocation. In *Proc. of the 14 th IJCAI*, pages 1694–1701, Montréal, Québec, Canada, 1995.
7. Berthe Y. Choueiry and Guevara Noubir. On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems. Technical Report KSL-98-24, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, Stanford, CA, 1998. Preliminary version in *Proc. of AAAI'98*.
8. Thomas Ellman. Abstraction via Approximate Symmetry. In *Proc. of the 13 th IJCAI*, pages 916–921, Chambéry, France, 1993.
9. Jay P. Fillmore and S.G. Williamson. On Backtracking: A Combinatorial Description of the Algorithm. *SIAM Journal on Computing*, 3 (1):41–55, 1974.
10. Eugene C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proc. of AAAI-91*, pages 227–233, Anaheim, CA, 1991.
11. Ian P. Gent and Patrick Prosser. Inside MAC and FC. Technical Report APES-20-2000, APES Research Group, 2000.
12. Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. Supermodels and Robustness. In *Proc. of AAAI-98*, pages 334–339, Madison, Wisconsin, 1998.
13. J.W.L. Glaisher. On the Problem of the Eight Queens. *Philosophical Magazine, series 4*, 48:457–467, 1874.
14. Alois Haselböck. Exploiting Interchangeabilities in Constraint Satisfaction Problems. In *Proc. of the 13 th IJCAI*, pages 282–287, Chambéry, France, 1993.
15. Tad Hogg, Bernardo A. Hubermann, and Colin P. Williams, editors. *Special Volume on Frontiers in Problem Solving: Phase Transitions and Complexity*, volume 81 (1-2). Elsevier Science, 1996.
16. Jean-Francois Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. In *ISMIS'93*, pages 350–361, 1993.
17. Daniel Sabin and Eugene C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of the 11 th ECAI*, pages 125–129, Amsterdam, The Netherlands, 1994.
18. M-C. Silaghi, D. Sam-Haroud, and B. Faltings. Ways of Maintaining Arc Consistency in Search using the Cartesian Representation. In *Proc. of ERCIM'99*, LNAI, Paphos, Cyprus, 1999. Springer Verlag.