Encodings, Consistency Algorithms and Dynamic Variable - Value Ordering Heuristics for Multiple Permutation Problems

Tayfun Pay[1] **and James L. Cox**[2]

[1]Computer Science Department
Graduate Center of New York
365 5th Ave. NY NY 10016
tpay@gradcenter.cuny.edu

[2]Computer and Information Science Department
Brooklyn College, CUNY
2900 Bedford Ave. Brooklyn NY 11210
cox@sci.brooklyn.cuny.edu

ABSTRACT

We introduce a technique to model a given multiple permutation problem as a constraint satisfaction problem (CSP). Our modeling eliminates the explicit need for channeling constraints by ensuring that an assignment or domain reduction to a variable in one model immediately updates the domains of all of the effected variables in all of the redundant models of the problem. We develop a forward-checking algorithm and then an arc-consistency (AC) algorithm that is tailored towards our modeling. We also explore various optimization methods for our AC algorithm that reduce propagation redundancy and find local inconsistencies sooner. We show that our CSP modeling coupled with our AC algorithm perform better than other CSP modelings with a generic AC algorithm. We also explore heuristics that integrate dynamic variable and value ordering into an AC algorithm and compare their performance to the state of the art CSP solvers.

Keywords: constraint satisfaction problems, multiple permutation problems, encodings, modelings, constraint propagation algorithms, consistency algorithms, dynamic variable ordering heuristics, dynamic value ordering heuristics, backtracking search, phase-transition

Mathematics Subject Classification: 68T20.
Computing Classification System: I.2.8.

1 Introduction

Many real world applications require the solution of NP-Complete optimization and decision problems. There is therefore much interest in finding new heuristics and techniques for improving the performance of solution software for these applications. Some recent examples include the following work in (Martin, del Toro, Haber and Dorronsoro, 2009), (Precup, Tomescu, Rădac, Petriu, Preitl and Dragoş, 2012) and (Ramírez-Ortegón, Märgner, Cuevas and Rojas, 2013). In this paper, we consider NP-complete multiple permutation problems. Some well-known problems of this type are Quasigroup Completion Problem, also referred to as Latin Squares (Colbourn, 1984), and Sudoku (Yato and Seta, 2003). Multiple permutation problems have been getting a lot of attention in the literature in the recent years (Walsh, 2001; Kautz, Ruan, Achlioptas, Gomes, Selman and Stickel, 2001; Gomes and Shmoys, 2002b; Dotu, Del Val and Cebrian, 2003; Hnich, Smith and Walsh, 2004; Ansotegui, Bejar, Fernandez, Gomez and Mateu, 2006) due to having a highly structured nature and being a challenging combinatorial search problem. Furthermore, it has been shown that many real-world problems in planning (Raza and Vidyarthi, 2008), rostering (Niño, 2010) and scheduling (Solos, Tassopoulos and Beligiannis, 2016) take the form of multiple permutation problems.Therefore, it has been suggested in (Gomes and Shmoys, 2002a; Ansotegui, Del Val, I., Fernandez and Manya, 2004; Ansotegui, Bejar, Fernandez, Gomez and Mateu, 2011) that they can be used as a benchmark problem to test various enhancement methods for solving constraint satisfaction problems (CSP) with backtracking-search. Then it is hoped that the insight gained from studying them can be applied to other hard structured as well as unstructured problems. Our enhancement methods and results can be summarized as follows:

1) We introduce the concept of *natural combined model* for encoding a multiple permutation problem as a CSP. A multiple permutation problem is formulated as a 3D Boolean matrix in the *natural combined model*, where the redundant-models coexists simultaneously and $\neq$ constraints are enforced on the pairs of variables of each redundant-model independently. The 3D Boolean matrix encoding collapses the dual variables of the redundant models such that an assignment to any variable immediately implies assignments to the respective variables in all of the possible redundant models of the problem. Additionally, a domain reduction of any variable immediately implies a domain reduction of the respective variables in all of the possible redundant models of the problem. Notice that the domains of both the primal and the dual variables are represented by 1D slices in our 3D Boolean matrix. Our *natural combined model* can be viewed as a hybrid in between the *minimal combined model* that was introduced in (Smith, 2001) that encodes the problem as a CSP and enforces *channeling constraints* among the variables of the redundant models of the problem; and the 3D model that was introduced in (Kautz et al., 2001) that encodes the problem as a Boolean satisfiability problem and enforces *at least* and *at most* constraints among the Boolean variables. We theoretically show that the constraints enforced by the *natural combined model* are as tight as the model that enforces the channeling constrains between primal and all dual variables with respect to arc-consistency.

2) Whenever redundant modeling was employed to formulate a given CSP in (Cheng, Lee and Wu, 1996; Cheng, Choi, Ho-Man Lee and Wu, 1999; Dotu et al., 2003; Ansotegui et al., 2004; Ansotegui et al., 2006), *channeling constraints* were used to connect as well as to propagate the constraints among the various redundant models of the problem. It has been shown in (Walsh, 2001; Smith, 2001; Hnich et al., 2004) that employing *channeling constraints* is superior to employing $\neq$ constraints between pairs of variables of a single model. Nevertheless, it has been theoretically shown in (Walsh, 2001; Hnich et al., 2004) that *channeling constraints* enforce a weaker form of local-consistency than <u>alldifferent</u> constraints with a generalised

arc-consistency (GAC) algorithm, but it has been empirically shown in (Smith, 2001; Hnich et al., 2004) that they are still competitive. It has been also reported in (Choi, Lee and Stuckey, 2003a; Choi, Lee and Stuckey, 2003b; Choi, Lee and Stuckey, 2008) that problems modeled with *channeling constraints* create propagation redundancy. However, there has been no case where a constraint propagation algorithm was specifically designed and developed for a given CSP modeling to reduce or eliminate propagation redundancy. Therefore, we first design and develop a forward-checking (FC) algorithm for the *natural combined model* and then we extend this notion to an arc-consistency (AC) algorithm. Then we create four additional renditions of our AC algorithm, where each implementation employs an additional optimization method to reduce propagation redundancy and/or find local-inconsistencies sooner. Then we show by our empirical results that these incremental modifications are justified since we achieve an overall performance improvement at the phase-transition point with each such optimization.
3) We develop the notion of integrating dynamic-variable and value ordering heuristics into an AC algorithm, which was first proposed in (Horsch and Havens, 2000) with a probabilistic heuristic. These dynamic-variable and value ordering heuristics employ two different counting assertions within our most enhanced AC algorithm. Our heuristics resemble impact based search of (Refalo, 2004) and activity based search of (Michel and Van Hentenryck, 2011), but they tremendously differ in the way they pick the next variable, order the assignment of values as well as the way they break-ties. After our initial experiments, we also created a heuristic that runs both of them concurrently in parallel.
4) We programmed our CSP modeling, algorithms and heuristics in C++ and ran numerous simulations with various problem sets that consisted of Sudoku puzzles. We tested our encoding method and five renditions of our AC algorithm with only the Smallest Domain (SD) heuristic, which is sometimes referred to as the Fail-First (FF) heuristic (Haralick and Elliot, 1980). We observed that as more optimization methods were introduced into our AC algorithm, more instances were solved at the phase-transition point within the given time limit. Next we wished to compare the performance of our natural combined model and its tailored RAC algorithm against other modelings using a generic arc consistency algorithm and employing the FF heuristic. We chose to do these in PICAT (Zhou, Kjellerstrand and Fruhman, 2015) since it is a very easy programming language to learn and it comes with a high quality builtin constraint solver, which also has an option to employ the FF heuristic. We used the programming language PICAT to encode these same problems with the previously studied CSP modelings in (Walsh, 2001; Dotu et al., 2003; Ansotegui et al., 2004; Hnich et al., 2004). These CSP modelings coupled with a generic AC algorithm performed worse than our approaches. Then we tested the performance of our integrated dynamic variable and value ordering heuristics on the same problem set, where our methods were able to solve many more instances at the phase-transition point. However, during all of these simulations, including the one with PICAT, we observed that the second phase-transition, as studied in (Hogg and Williams, 1994; Gent and Walsh, 1994; Mitchell and Levesque, 1996; Selman and Kirkpatrick, 1976; Gomes, Selman, Crato and Kautz, 2000), is algorithm dependent for multiple permutation problems, as was already shown to be the case for some other NP-Complete problems in (Coarfa, Demopoulos,

San Miguel Aguierre, Subramanian and Vardi, 2000; San Miguel Aguirre and Vardi, 2001). We also observed that our parallel heuristic was able to solve many more instances at the phase-transition point, and also mitigated the heavy-tail region, which had been shown in (Hulubei and O'Sullivan, 2005; Hulubei and O'Sullivan, 2006) to be amenable to exploitation. Finally, we ran simulations with a set of balanced problems from (Ansotegui et al., 2006; Mateu, 2009; Ansotegui et al., 2011), where it was shown that it gets harder to solve a given problem when the punched hole pattern becomes more balanced among the permutations. This specific problem set was previously simulated using MAC-LAH of (Ansotegui et al., 2004) and Minion of (Gent, Jefferson and Miguel, 2006), where these CSP solvers employ a different variable ordering heuristic and a different filtering algorithm, both from each other and from ours; they both employ the same encoding as studied in (Dotu et al., 2003; Ansotegui et al., 2004). We showed that both of our integrated dynamic variable and value ordering heuristics performed much better than both of these methods with only 10% of the allocated time limit that they were given in (Ansotegui et al., 2006; Mateu, 2009; Ansotegui et al., 2011).

2 Natural combined model

A given constraint satisfaction problem (CSP) has a set of variables V_i with their respective domains D_{V_i} and a set of constraints, C_j among these variables. The aim of a constraint solver is to find a solution S that assigns a value to each of these variables from their respective domains such that all of the given constraints are satisfied. Any given CSP can be modeled at least two ways by switching the variables to be the domains and domains to be the variables. Either of these redundant models can be employed independently to find a solution using a constraint solver. However, what is most often done is to employ *channeling constraints* between these redundant models and forget about the constraints between the variables of each redundant model. This concept was formally defined in (Cheng et al., 1996).
It is best to illustrate the *channeling constraints* through an example. Let's examine how the Quasigroup Completion Problem (QCP) was redundantly modeled with *channeling constraints* in (Dotu et al., 2003), where this modeling was also adopted in (Ansotegui et al., 2004). As defined in (Dotu et al., 2003), the primal variables are the set $X = \{x_{i,j} | 1 \leq i \leq n, 1 \leq j \leq n\}$ where $x_{i,j}$ is the cell in the QCP, and the domains are $D = \{v | 1 \leq v \leq n\}$. The primal variables are divided among columns and rows, where column j is $\{x_{i,j} | 1 \leq i \leq n\}$ and row i is $\{x_{i,j} | 1 \leq j \leq n\}$. There exist $\neq$ constraints between primal variables within each column and row. The column dual variables are $C = \{c_{v,i} | 1 \leq i \leq n, 1 \leq v \leq n\}$ and their domains are $D_c = \{j | 1 \leq j \leq n\}$, which represents the rows in column i where value v can occur. There exist $\neq$ constraints between column variables within each column that enforce that two values cannot occur in the same row within that column. The row dual variables and constraints can be defined similarly. Then the following two *channeling constraints* between primal and dual variables were proposed in (Dotu et al., 2003) to connect the three redundant models of the QCP. The column *channeling constraints*, $x_{i,j} = v \leftrightarrow c_{v,i} = j$, link the primal variables with the column dual variables. The row *channeling constraints*, $x_{i,j} = v \leftrightarrow r_{v,j} = i$, link the primal variables with the row dual variables. They called this model the *bi-channeling* model, which is

very much alike the *minimal combined* model proposed in (Smith, 2001), and they showed that these *channeling constraints* are sufficient to completely define the problem, that is without the $\neq$ constraints between the variables of each redundant model.
They also introduced a third *channeling constraint* in (Dotu et al., 2003), that links the column dual variables with the row dual variables, $c_{v,i} = j \leftrightarrow r_{v,j} = i$. They called the model that used all of the three aforementioned *channeling constraints* the *triangular channeling* model. They employed a generalised arc-consistency algorithm to maintain arc-consistency on the "binary" *channeling constraints*, because that is the only arc-consistency algorithm that is built into the constraint solver CPlan. Thus the constraint solver that they relied on is definitely not tailored towards the modeling choice. This issue was discussed in section 6 of (Dotu et al., 2003), where they noted that using forward-checking with the *triangular channeling* model achieved more pruning than using the generalised arc-consistency algorithm with the *bi-channeling* model. They imprecisely indicated that the built in generalised arc-consistency algorithm creates propagation redundancy and this in return hindered the performance.
On the other hand, in (Walsh, 2001; Hnich et al., 2004) they modeled the Quasigroup Completion Problem with the three different CSP models, the $\neq$ constraints between primal variables, *channeling constraints* between primal and dual variables, and the alldifferent constraints among primal variables. They used the constraint solver ILOG and showed that sometimes employing the *channeling constraints* and maintaining arc-consistency is as competitive as employing the alldifferent constraint and maintaining generalised arc-consistency. However, it is also indicated that maintaining arc-consistency on the *channeling constraints* potentially creates propagation redundancy.
The problem of propagation redundancy in redundantly modeled problems with *channeling constraints* have been discussed in (Choi et al., 2003a; Choi et al., 2003b; Choi et al., 2008), where they explained how propagation redundancy occurs when *channeling constraints* are employed. This can be briefly summarized in the sense of (Choi et al., 2008) as follows :

> "*In redundant modeling, each model is logically redundant with respect to the other model plus the channeling constraints. In general, the propagators defined for two viewpoints act in different ways and discover information at different stages in the search. ..., we show two possibilities in which propagation caused by some constraints in one model can be made redundant by: (a) propagation induced from constraints in the other model through channels and (b) propagation of the channels themselves.*"

They also showed in (Choi et al., 2008) that reducing propagation redundant constraints can speed up search when *channeling constraints* are employed. However, the methods they proposed are abstract and do not really go to the root of the problem, that is the modeling choice and the interaction of this modeling with the given constraint propagation algorithm. In other words, even though the effectiveness of the modeling choice is dependent on the constraint propagation algorithm for multiple permutation problems encoded with *channeling constraints*, no modeling was specifically formulated for a given constraint propagation algorithm. Conversely, no constraint propagation algorithm was explicitly designed and developed for multiple permutation problems encoded with *channeling constraints*. Therefore, we propose what we call the *natural combined model* to formulate a given multiple permutation problem as a CSP. Then in the next section, we design and develop a redundantly modeled forward-checking algorithm and a redundantly modeled arc-consistency algorithm that are custom-made for the *natural combined model*. In this way, we can take full advantage of our formulation of the problem so we do not have to deal with the setbacks of using a generic constraint propagation algorithm that is not necessarily tailored towards our modeling and eventually creates propagation redundancy. We will next define the *natural combined model* in terms of Sudoku, since it has been empirically shown to be the hardest multiple permutation problem to solve in (Ansotegui et al., 2006; Mateu, 2009; Ansotegui et al., 2011).

Definition 2.1. Sudoku puzzle
A given Sudoku puzzle of order k is played on a $(k * k)$ by $(k * k)$ grid that consists of $(k * k)^2$ cells. A solution to a given Sudoku puzzle assigns a single number to each cell so that numbers 1 through k^2 appears exactly once on each of the k^2 boxes, k^2 columns and k^2 rows. The set of cells in a Sudoku puzzle is defined as $X = \{x_{i,j} | 1 \leq i \leq n, 1 \leq j \leq n\}$, where $n = (k * k)$ and their domains are $D = \{1 \leq v \leq n\}$. These primal variables are dividied among boxes, columns, and rows, where column j is $\{x_{i,j} | 1 \leq i \leq n\}$ and row i is $\{x_{i,j} | 1 \leq j \leq n\}$. Then the more complex boxes are indexed by p and q for $\{1 \leq p \leq k\}$ and $\{1 \leq q \leq k\}$, and box$_{p,q}$ consists of $\{x_{i,j} | \lceil i/k \rceil = p, \lceil j/k \rceil = q\}$. There exist a $\neq$ constraint on each pair of variables within each box, column and row.

The dual models for Sudoku is defined similarly to the QCP with the same column and row channelling constraints with the addition of box dual variables.

Definition 2.2. Natural Combined Model for Sudoku
In the *natural combined model*, the Sudoku puzzle is formulated on a 3D $\mathcal{S}(k * k) * (k * k) * (k * k)$ Boolean matrix, where $n = (k * k)$. The primal variable $x_{i,j}$ is represented by the slice $\{\mathcal{S}(i, j, v) | 1 \leq v \leq n\}$. The initial domain of $x_{i,j}$ is denoted $D(x_{i,j})$ and is $\{v | \mathcal{S}(i, j, v) = True, , 1 \leq v \leq n\}$. The column dual variable $c_{v,j}$ is represented by the slice $\{\mathcal{S}(i, j, v) | 1 \leq i \leq n\}$. The initial domain of $c_{v,j}$ is denoted $D(c_{v,j})$ and is $\{i | \mathcal{S}(i, j, v) = True, 1 \leq i \leq n\}$. The row dual variable $r_{v,i}$ is represented by the slice $\{\mathcal{S}(i, j, v) | 1 \leq j \leq n\}$. The initial domain of $r_{v,j}$ is denoted $D(r_{v,j})$ and is $\{j | \mathcal{S}(i, j, v) = True, 1 \leq j \leq n\}$. The box dual variable $b_{v,p,q}$ where $\lceil i/k \rceil = p, \lceil j/k \rceil = q$ is represented by the slice $\{\mathcal{S}(i, j, v) | ((p - 1) * k + 1) \leq i \leq p * k, ((q - 1) * k + 1) \leq j \leq q * k\}$. The initial domain of $b_{v,p,q}$ is denoted $D(b_{v,p,q})$ and, is $\{i, j | \mathcal{S}(i, j, v) = True, ((p - 1) * k + 1) \leq i \leq p * k, ((q - 1) * k + 1) \leq j \leq q * k\}$. We enforce $\neq$ constraints on pairs of primal variables within each box, column and row. We also enforce $\neq$ constraints on pairs of box, column and row dual variables of each value.

In the *natural combined model*, there is no need to maintain a constraint graph, as dependencies between variables are given by the structure of the array. This is also true for the QCP minus the box dual variables. Each entry $\mathcal{S}(i, j, v)$ encodes four pieces of information: whether v is in the domain of $x_{i,j}$, whether i is in the domain of $c_{v,j}$, whether j is in the domain of $r_{v,i}$, and whether i, j is in the domain of the corresponding $b_{v,p,q}$. Thus, if we remove a value from the domain of a primary variable $x_{i,j}$ by setting the appropriate entry of $\mathcal{S}$ to zero, this automatically removes the corresponding place from the domains of the respective box, column and row dual variables. Moreover, we will observe in the next section that this representation will greatly simply our filtering algorithms.

The following two theorems show that the constraints enforced by the *natural combined model* are as tight as the model that enforces the channeling constrains between primal and all dual variables with respect to arc-consistency. The first theorem shows that our model suffers no loss of power by its lack of explicit channeling constraints. For one direction we observe that the act of enforcing $\neq$ on the set of primal variables in a particular column (or box, or row) has the dual effect of removing both values from the domains of those primal variables, as well as removing corresponding places from the domains of the column dual variables. That is, if we remove a value v from the domain of a cell in row i within a specific column j, place i is removed from the domain of the dual column variable for value v and column j. The other direction is symmetric. The beauty of this simple encoding is that we get the channeling constraints for free.

Theorem 2.1. *The constraints enforced by the natural combined model are equivalent to the channeling constraints between primal and all dual variables.*

Proof. Without loss of generality we shall consider the column *channeling constraint*, $x_{i',j'} = v' \leftrightarrow c_{v',j'} = i'$. For the first direction, assume that $x_{i',j'} = v'$. Enforcing the $\neq$ constraints on the pairs of primal variables minimally removes the value v' from the domains of all primal variables in column j', that is, $\forall i \neq i', \mathcal{S}(i, j', v') = False$. From our encoding this means that $D(c_{v',j'})$ now only contains i' and so $c_{v',j'} = i'$. For the other direction, assume that that $c_{v',j'} = i'$. Again enforcing the $\neq$ constraints on the pairs column dual variables means minimally that this place is removed from the other column dual variables for column j' and thus $\forall v \neq v', \mathcal{S}(i', j', v) = False$. From our encoding this means that $D(x_{i',j'})$ now only contains v' and so $x_{i',j'} = v'$. The proofs for box and row dual variables are similar. □

This next theorem shows that if the number of unnassigned variables within a particular box, column, or row is greater than the cardinality of the union of their domains, then the state is inconsistent and we will detect this fact. This is a limited form of Hall's theorem, which is a more general result related to the general problem of bipartite matching. The full form says that a state can be extended to a solution if and only if the set of unassigned variables has smaller cardinality than the union of their domains.

Theorem 2.2. *Limited form of Hall's theorem is enforced by the Natural Combined Model of Sudoku*

Proof. Let the set $BCR \subseteq \{y_1, ..., y_m\}$ be the unassigned primal variables in the same box, column or row and that $|BCR| > | \cup_{y_l \in BCR} D_l|$. Also assume without loss of generality that these variables are from column i. Then by the pigeonhole principle there exists an unassigned value v that is also $v \notin \cup_{y'_l \in BCR} D'_l$ then all of the entries from $\mathcal{S}(i, 1, v)$ through $\mathcal{S}(i, n, v)$ are zero. However, this local-inconsistency will be detected when enforcing $\neq$ constraints between pairs of column dual variables. Conversely, let the set $BCR \subseteq \{y_1, ..., y_m\}$ be the unassigned box dual, column dual or row dual variables and that $|BCR| > | \cup_{y_l \in BCR} D_l|$. Also assume without loss of generality that these are from the same row j. Then by the pigeonhole principle there exists an unassigned value i that is also $i \notin \cup_{y'_l \in BCR} D'_l$ then all of the entries from $\mathcal{S}(i, j, 1)$ through $\mathcal{S}(i, j, n)$ are zero. However, this local-inconsistency will be detected when enforcing $\neq$ constraints between pairs of primal variable. □

3 Redundantly Modeled Forward-Checking and Arc-Consistency Algorithms

There has been various forward-checking algorithms proposed in the literature (Haralick and Elliot, 1980) as well as numerous arc-consistency algorithms (Mackworth, 1977; Mohr and Henderson, 1986). Even though, these algorithms have general applications, they were not designed and developed for an explicit CSP modeling. The aim of this section is to introduce our own consistency algorithms that are specifically designed and developed for our *natural combined model*. Then in the subsequent sections, we will compare the performance of our *natural combined model* coupled with our consistency algorithms against the performance of other modelings coupled with generic consistency algorithms. Once again, we will exclusively study Sudoku, since it has been empirically shown to be the hardest multiple permutation problem to solve in (Ansotegui et al., 2006; Mateu, 2009; Ansotegui et al., 2011).

Definition 3.1. Redundantly Modeled Forward-Checking Algorithm
Once a value v is invoked to be instantiated at location $x_{i,j}$

1. Remove other values in the domain of the primal variable $x_{i,j}$ and immediately check to see that these values were not the last places that they could appear in that Box, Column or Row, otherwise raise an inconsistency flag.

2. Go through the Box, Column and Row of the primal variable $x_{i,j}$ and remove value v where ever it appears.

   A. Then immediately check to see that value v was not the last value in the domain of the given primal variable, otherwise raise an inconsistency flag.

   B. If we are not in the same Box as the instantiated primal variable $x_{i,j}$, then also check to see that this was not the last location where value v could appear in that Box, otherwise raise an inconsistency flag.

   C. If we are not in the same Column as the instantiated primal variable $x_{i,j}$, then also check to see that this was not the last location where value v could appear in that Column, otherwise raise an inconsistency flag.

   D. If we are not in the same Row as the instantiated primal variable $x_{i,j}$, then also check to see that this was not the last location where value v could appear in that Row, otherwise raise an inconsistency flag.

Let's examine how we cut down on propagation redundancy with our redundantly modeled forward checking (RFC) algorithm. As we can see in Figure1, the value 7 is instantiated at primal variable with orange edges ($x_{5,5}$). At this point, step 1 of the RFC algorithm is executed, where values other than 7 in the domain of the primal variable are removed and immediately

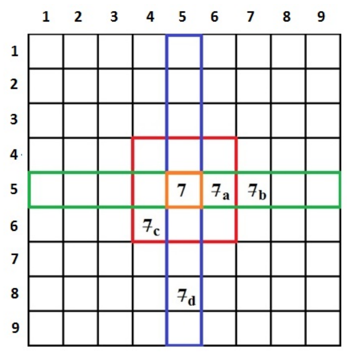

Figure 1: Running Example of Forward Checking Algorithm

a check is performed to see that this primal variable was not the last places that they could appear in the red box ($x_{4,4}$, $x_{4,5}$, $x_{4,6}$, $x_{5,4}$, $x_{5,5}$, $x_{5,6}$, $x_{6,4}$, $x_{6,5}$, $x_{6,6}$), blue column ($x_{5,1}$ ... $x_{5,9}$) and green row ($x_{1,5}$ ... $x_{9,5}$). Then we go through all of the primal variables in the red box, blue column and green row and remove 7 from their domains. Once we are at primal variable labeled 'a' in Figure1, we remove 7 from the domain of that primal variable, check to see that 7 was not the last value in the domain of that primal variable, which is step 2.A of the RFC algorithm and then check to see that this location was not the last place 7 could appear in that column, which is step 2.C of the RFC algorithm. We do not perform steps 2.B and 2.D of the RFC algorithm, because we are in the same box and row as the instantiated primal variable. This process works similarly when we are at primal variables labeled 'b', 'c' and 'd' in Figure1. At primal variable labeled 'b' in Figure1, the RFC algorithm performs steps 2.A, 2.B, 2.C, but not 2.D. At primal variable labeled 'c' in Figure1, the RFC algorithm performs steps 2.A, 2.C and 2.D but not 2.B. At primal variable labeled 'd' in Figure1, the RFC algorithm performs steps 2.A, 2.B, 2.D and not 2.C.

The generic arc-consistency algorithms that are built into the given constraint solvers have been shown to create propagation redundancy when enforcing *channeling constraints*. For example, when one employs the *channeling constraints* between primal variables and box, column and row dual variables. Then a generic arc-consistency algorithm, such as AC3 (Mackworth, 1977), enforce these constraints the following way. If a primal variable is fixed to 7 then the three *channeling constraints* that connect this cell and value pair to its dual variables will be placed on the queue. However, in our *natural combined model*, those dual variables are immediately updated and propagations to these *channeling constraints* are not necessary as we observed with our RFC algorithm. We extend the RFC algorithm into an arc-consistency algorithm.

Definition 3.2. Redundantly Modeled Arc-Consistency Algorithm ($RAC-1$)

1. Go through the variables of $each$ redundant model consecutively.
2. If the domain of the given variable of some redundant model is singleton, then instantiate that variable with that value and then call the revised RFC algorithm to prune the search space and to find possible local-inconsistencies.

Execute 1 and then 2 until no value has been instantiated in one full pass of all of the variables of each redundant model or a local-inconsistency is discovered.

When a choice point is reached during backtracking-search, the primal variable is instantiated and then the RFC algorithm is invoked. If the RFC algorithm does not find a local-inconsistency, then the redundantly modeled arc-consistency algorithm is invoked. Thus, we use four different revised versions of the RFC algorithm in the redundantly modeled arc-consistency (RAC) algorithm to do pruning. For example, if the primal variable's domain is singleton, then we do not have to examine it's domain. Conversely, if the box dual variable's domain is singleton, then we do not have to go through the primal variables in that box to remove that value. Similarly for the column dual and row dual variables. In essence, once again we are cutting down on propagation redundancy. However, we are doing a lot of unnecessary singleton domain checks by going through the variables of each redundant model consecutively until no value has been instantiated in one full pass of all of the variables of each redundant model. A better option is to employ a queue.

Definition 3.3. Redundantly Modeled Arc-Consistency Algorithm ($RAC-2$)
We create a queue Q of primal variables to be instantiated by going through the variables of $each$ redundant model and inserting into the queue Q the primal "variable" index and the"value" of the respective dual variables with a singleton domain as well as primal variables with a singleton domain.

1. Remove a value-location pair from the queue Q.
2. Instantiate that primal variable with the given value and then call the revised RFC algorithm that besides checking for local-inconsistencies, it also checks to see if the domain of any variable (primal or dual) became singleton and then inserts the respective primal variables index and the value into the queue Q.

Execute until queue Q is empty or a local-inconsistency is discovered.

Although we eliminate a lot of unnecessary singleton domain checks with the RAC-2 algorithm, we only perform instantiations using the primal variables. As a result, we do not know the reason why that value was invoked to be instantiated at that location. The information about which variable's domain (primal or one of the dual) became singleton is lost. This evidently leads to propagation redundancy. For example, if it was the box dual variable that invoked the instantiation to that location then we unnecessarily go through that box to check to see if we can remove the instantiated value from the domains of those primal variables. A similar scenario happens if it was the column or row dual variables that invoked the instantiation to that location.

Definition 3.4. Redundantly Modeled Arc-Consistency Algorithm ($RAC-3$)
We create an array L that keeps track of which variable is to be instantiated next. We also create two integer matrices, where the first integer matrix called V keeps track of the value that is to be instantiated at location i, j and the second integer matrix called M keeps track of the model that invoked the instantiation.
We initialize the array L of primal variables to be instantiated by going through the variables of $each$ redundant model and inserting into the array L the location of the primal variable (encoded

as $\lfloor i/k \rfloor * k + \lfloor j/k \rfloor$) associated with the respective dual variables with a singleton domain or simply the primal variable with a singleton domain. Whenever we insert an encoded location into the array L, we also insert the value to be instantiated at that location into the matrix V and insert the type of the model that invoked the instantiation into the matrix M.

1. Decode the primal variable location at index p in array L
2. Instantiate the primal variable at location i, j with the value at the same location in matrix V. Then call the appropriate revised RFC algorithm according to the information for location i, j in matrix M. The revised RFC algorithm that besides checking for local-inconsistencies, it also checks to see if the domain of any variable (primal or dual) became singleton and inserts the encoded location into the array L and updates the matrices V and M accordingly.

Execute until all of the variables in array L have been instantiated or a local-inconsistency is discovered.

Although RAC-3 eliminates some propagation redundancy, it does not eliminate propagation redundancy caused by repetitious instantiations. For example, location i, j might get invoked to be instantiated with value v because that is the last value left in that primal variable's domain. Then location i, j might get invoked to be instantiated with value v because location i, j is the last place where value v can appear in that box as well. Then the RAC-3 algorithm will attempt to instantiate value v to location i, j twice, but it will think that both of these instantiations were caused by the box dual variable. As a result, it will perform a lot of unnecessary domain checks. Additionally, if location i, j is also the last place where value v can appear in that column as well as in that row, then nothing needs to be checked at all! There are 15 different combined ways an instantiation can be invoked when you consider the permutation of the primal, box dual, column dual and row dual variables.

Definition 3.5. Redundantly Modeled Arc-Consistency Algorithm ($RAC-4$)
Append the following text at the end of bullet point "2." in RAC-3: The indices of matrix M are checked before written over to account for repetitious instantiations. If there was already a value at the respective index in matrix M, that value is combined with the new value.

We should note that there are four revised versions of the RFC algorithm in the RAC-3 algorithm whereas there are fifteen revised versions of the RFC algorithm in the RAC-4 algorithm. Although RAC-4 eliminates propagation redundancy caused by repetitious instantiations, it does not account for the fact that these instantiations might have invoked different values to be instantiated at that location. For example, let's assume that the primal variable at location i, j has the domain $\{4, 5, 6\}$. However, location i, j is the only place 4 could appear in that column and location i, j is also the only place 5 could appear in that row. Thus, which ever one is the last invoked instantiation, that value would be present in matrix V. Even though, the algorithm will eventually catch this local-inconsistency, catching it as soon as possible is much better because we might unnecessarily end up instantiating a lot of other variables in between these two inconsistent instantiations.

Definition 3.6. Redundantly Modeled Arc-Consistency Algorithm ($RAC - 5$)
Append the following text at the end of bullet point "2." in RAC-4: We also check to see that the value in matrix V is the same for repetitious instantiations, where otherwise we raise an inconsistency flag.

The worst-case time complexity of RAC-1 is **O**($m * v^2 * d$), whereas the worst-case time complexities of RAC-2 through RAC-5 are **O**($m * v^{3/2} * d$). In terms of Sudoku puzzles of order k, $m = 4$, $v = (k * k)^2$ and $d = k * k$. Then the worst-case time complexity of RAC-1 becomes **O**(k^{10}) = **O**($v^{5/2}$) and the worst-case time complexities of RAC-2 through RAC-5 become **O**(k^8) = **O**(v^2) .

4 Redundantly modeled arc-consistency with integrated dynamic variable-value ordering heuristic

In the previous two sections, we introduced a new modeling technique and new consistency algorithms that are tailored towards our modeling. The next step in building a constraint solver is to incorporate variable and value ordering heuristics to guide the search. What we propose is something that has only been explored once before in (Horsch and Havens, 2000), which is to integrate a dynamic variable ordering heuristic into a consistency algorithm. The heuristic in (Horsch and Havens, 2000) is probabilistic in nature, whereas the ones we propose employ greedy counting assertions. Furthermore, we also integrate a dynamic value ordering into this heuristic, so that the selection of the variable also depends on the values in its domain, which is novel. We will use the most sophisticated version of our filtering algorithm, which is RAC-5.

Definition 4.1. Total Number of Fixed Variables with Redundantly Modeled Arc-Consistency ($TNFV \oplus RAC$)

1. Calculate the smallest domain size for the primal variables.
2. Then for each primal variable with the smallest domain size:

   Instantiate each of the possible values in its domain.

   For each such instantiation execute RAC-5 and calculate the total number of additional fixed variables obtained
3. Select the variable that gives the most number of additional fixed variables and order the assignment of values from the value that produces the most number of fixed variables to the value that produces the least number of fixed variables

   In case of a tie, select the variable for which the next value in its domain returns a higher value.
4. If for any variable, all of its instantiations after executing RAC-5 results in a local-inconsistency then raise an inconsistency flag.

Definition 4.2. Total Number of Domain Reductions with Redundantly Modeled Arc-Consistency $(TNDR \oplus RAC)$
We change the phrase "most number of fixed variables " in $(TNFV \oplus RAC)$ with the phrase"most number of cumulative domain reductions"

We noticed an interesting behavior in our initial experiments, where each algorithm would solve quite a few instances that the other was not able to solve at the phase-transition point as well as the subsequent region. Therefore, we developed a parallel version that runs both of them concurrently.

Definition 4.3. Total Number of (Domain Reductions OR Fixed Variables) with Redundantly Modeled Arc-Consistency $(TN(DR||FV) \oplus RAC)$

1. Execute $(TNFV \oplus RAC)$ on thread 1.
2. Execute $(TNDR \oplus RAC)$ on thread 2.

Several related search heuristics were proposed in the literature, such as impact based search (Refalo, 2004) and activity based search (Michel and Van Hentenryck, 2011). Impact based search attempts to estimate the product of the domain sizes following a possible assignment, called the impact, and attempts to choose the variable with the greatest impact for assignment, where the smaller the product of domains the greater the impact. It arbitrarily breaks ties and then selects a value with the least impact. In contrast, in our total number of fixed variable heuristic, we calculate the total number of fixed values (domain of size one) obtained after constraint propagation by the RAC algorithm for each possible assignment for each unassigned variable. Moreover, after the variable is chosen, its values are ordered such that the ones that yield the most number of fixed cells are chosen first. Furthermore, if two variables yield the same number of fixed variables, then the second value in their domain is used to break the ties. On the other hand, the activity based search keeps track of the number of times a variable has been updated on the current search path by constraint propagation. This is not the same as the total number of values that were removed from its domain; any number of domain removals during single constraint propagation counts as a single update. A variable is then chosen by the ratio of its update history and its domain size. Furthermore, the ties are broken arbitrarily and no particular value heuristic is specified. By contrast, it is not necessary to maintain a history in our total number of domain reduction heuristic. In this heuristic, for each possible variable and value pair, we calculate the total number of actual values removed from all of the domains by constraint propagation using the RAC algorithm. A variable is chosen with the assignment that yields the most number of domain reductions and the values are ordered accordingly. A tie-breaker is also employed as in the total number of fixed variables heuristic.

5 Experimental setup

We implemented our encoding method, the consistency algorithms and dynamic variable-value ordering heuristics in C++. Then we generated the test instances by following the methodology that was outlined in (Lewis, 2007) for Sudoku puzzles, which is like the ones that were used

in (Cazenave, 2006; Whitlock and Lambert, 2010; Whitlock and Mayorov, 2011; Chaimowicz and Machado, 2011; Simonis, 2005). We first randomly filled 5%-25% percent of the empty puzzle board without invalidating the puzzle and then randomly chose one of our algorithms to complete the puzzle. We generated 200 fully solved Sudoku puzzles of order 5, which is a 25 by 25 Sudoku puzzle. Then we removed a cell from a given fully solved puzzle with probability p, where $p = 0$ implies a fully solved puzzle and $p = 1$ implies an empty one. We did this for all of the 200 solved puzzles we generated for each probability p, from $p = 0.05$ through $p = 0.95$ with 0.05 increments. We used the Mersenne Twister Pseudo-Random generator mt19937 in C$++$ for generating our random numbers, that were uniformly distributed between 0 and 1. A total of 3800 puzzles were generated for Sudoku puzzles of order 5. None of these puzzles are guaranteed to have a unique solution and thus they are not necessarily logically solvable. We set the time limit to 120 seconds for Sudoku puzzles of order 5, since this was the time limit employed in (Pay, 2015) and close to the time limit used in (Cazenave, 2006).
We kept track of four different parameters at each probability p: 1) Percentage of puzzles solved 2) Average time 3) Standard Deviation Error for time 4) Average number of explored nodes. If no solution was found within the given time limit, then the given time limit and the calculated values for each parameter up to that time limit were used in the calculations. We used a computer that possesses an Intel Dual Core (Four Threads) I3-3227U CPU at 1.90GHZ x64 based processor with 4GB of RAM to run our simulations.
We did not perform any studies on Sudoku puzzles of order 3, 4 or 3D Sudoku puzzles of order 3 as they did in (Cazenave, 2006; Lewis, 2007; Whitlock and Lambert, 2010; Whitlock and Mayorov, 2011; Chaimowicz and Machado, 2011) because no phase-transition was found for these puzzles in (Pay, 2015) when employing the $(TN(DR||FV) \oplus RAC)$ heuristic.

6 Experimental analysis

Since our objective is to compare the performance of the various renditions of the same algorithm, we only employed the smallest domain (SD) heuristic, where the variable with the smallest domain size was lexicographically picked and the values in its domain were lexicographically ordered.

6.1 Performance of the RAC algorithms with the Natural Combined Model

	p = 0.5	p = 0.55	p = 0.6	p = 0.65	p = 0.7	p = 0.75	p = 0.8	p = 0.85	p = 0.9	p = 0.95
SD+RAC-1	77.5%	31.5%	54%	79%	84.5%	91%	88%	96.5%	99.5%	99.5%
SD+RAC-2	78%	32%	54%	79%	84.5%	91%	87%	96.5%	99.5%	99.5%
SD+RAC-3	79%	33%	54.5%	79.5%	85%	91%	87.5%	96.5%	99.5%	99.5%
SD+RAC-4	79.5%	33%	54.5%	79.5%	85%	91%	87.5%	96.5%	99.5%	99.5%
SD+RAC-5	80%	34%	55.5%	79.5%	85%	91%	87%	96.5%	99.5%	99.5%

Table 1: Percentage of solved instances for RAC-1 through RAC-5 at each probability p from 0.5 through 0.95.

All renditions of our RAC algorithm were able to solve all of the instances up to and including probability $p = 0.45$. We observe from Table 1 that the percentage of solved instances

decreased for all of the renditions at probability $p = 0.5$ and abruptly dropped at probability $p = 0.55$, which is the phase-transition point. The RAC-5 rendition was able to solve the most number of instances at the phase-transition point. The mean times and standard deviation error (sde) at the phase-transitions point were as follows: RAC-1 had a mean time of 94 seconds with a sde of 3.2, RAC-2 had a mean time of 92 seconds with a sde of 3.3, RAC-3 had a mean time of 90 seconds with a sde of 3.3, RAC-4 had a mean time of 89 seconds with a sde of 3.4 and RAC-5 had a mean time of 84 seconds with a sde of 3.4. This is better than any of the methods proposed in (Cazenave, 2006; Lewis, 2007; Chaimowicz and Machado, 2011), where they cannot solve any instances at the phase-transition point.
All of the renditions of our RAC algorithm were able to solve more and more instances as the probability increases, where the RAC-5 rendition out performed all of them up to and including probability $p = 0.7$. Then at probability $p = 0.75$, all of the renditions solved the same number of instances. However, an interesting phenomenon occurs at probability $p = 0.8$, where the RAC-1 rendition solved more instances than all of the other renditions. This is also the point where we observe a small bump, a second phase-transition point, where the percentage of solved instances is less for all of the renditions compared to percentage of solved instances at the adjacent probabilities of $p = 0.75$ and $p = 0.85$.

	p = 0.5	p = 0.55	p = 0.6	p = 0.65	p = 0.7	p = 0.75	p = 0.8	p = 0.85	p = 0.9	p = 0.95
SD+RAC-1	20090	58613	49011	26477	18514	15045	13451	4866	1218	1195
SD+RAC-2	20971	61122	50986	26354	18278	14996	12401	4257	996	975
SD+RAC-3	22603	65736	51671	26494	19085	15096	13182	4487	1123	1103
SD+RAC-4	23864	68987	57101	27765	19515	15235	12918	4498	1097	1094
SD+RAC-5	26212	76369	58808	28911	19952	15909	12771	4552	1085	1090

Table 2: Mean of explored nodes for RAC-1 through RAC-5 at each probability p from 0.5 through 0.95.

We observe from Table 2 that more and more nodes are being explored on average at probability $p = 0.55$ within the given time limit of 120 seconds as we go from RAC-1 through RAC-5 renditions of our algorithm. As the probability p increases, the mean of explored nodes starts to converge among all of the renditions, where RAC-5 still explores more nodes on average up to and including probability $p = 0.75$. Then starting at probability $p = 0.8$, the RAC-1 rendition has a higher mean of explored nodes than all of the other renditions.
The RAC-5 rendition outperformed all of the other renditions at the phase-transition point. In other words, the RAC-5 rendition solved more instances and explored more nodes when an instance was not solved within the given time limit. This clearly indicates that each optimization technique we applied was able to speed up the constraint propagation at probability $p = 0.55$. We should also point out that when a given instance was solved by all or some subset of the renditions of our RAC algorithm at any given probability, it was solved with the same number of explored nodes and instantiations.
On the other hand, the reason why the performance among the renditions of our RAC algorithm converge going from probability $p = 0.55$ to probability $p = 0.75$ and then flip for the better for RAC-1 rendition at probability $p = 0.8$ is not so clear. This behavior can perhaps be explained by the fact that as the probability p increases, the initial puzzle boards start to become more

and more sparsely populated. This in return causes all of the RAC algorithms to instantiate a lot of nodes after each choice-point. Consequently, there is a greater amount of time in between possible local-inconsistencies. As a result, the benefit of keeping a list of nodes to be instantiated slowly diminishes as the probability increases. In other words, probability $p = 0.8$ is the point where keeping a list of nodes to be instantiated outweighs it's benefits and going through all of the variables of each redundant model consecutively until there are no more updates performs better.

It is worth noting that probability $p = 0.8$ is also the location of the second phase-transition. It has been observed in (Hogg and Williams, 1994; Gent and Walsh, 1994; Mitchell and Levesque, 1996; Selman and Kirkpatrick, 1976; Gomes et al., 2000) that very hard NP-complete problems have a second peak of hardness, which is referred to as the double phase-transition phenomenon. It has been also shown in (Coarfa et al., 2000; San Miguel Aguirre and Vardi, 2001) that the location of the second peak of hardness is algorithm dependent for the same NP-complete problem. We employed the same fundamental algorithm, but with different implementations that introduced some optimization techniques. The second peak in hardness for all of these renditions were at probability $p = 0.8$. However, the fact that RAC-1 performed the worst at the first phase-transition point, but out performed the other renditions at the second phase-transition point may perhaps be traced back to the algorithm dependent nature of this region.

6.2 Performance of the previously studied CSP modelings

	p = 0.45	p = 0.5	p = 0.55	p = 0.6	p = 0.65	p = 0.7	p = 0.75	p = 0.8	p = 0.85	p = 0.9	p = 0.95
SD+RFC	98%	25.5%	0%	0%	4.5%	14%	34.5%	48%	73.5%	86%	96%
$\neq$	93.5%	29%	0%	0.5%	1%	4%	15.5%	35%	54.5%	83.5%	87.5%
SD+RAC-1	100%	77.5%	31.5%	54%	79%	84.5%	91%	88%	96.5%	99.5%	99.5%
BiChan	100%	76%	23.5%	40%	53%	70%	81%	90%	92.5%	99%	95.5%
TriChan	100%	74.5%	22%	38.5%	52%	68%	81%	89.5%	92.5%	99%	95%

Table 3: Percentage of solved instances with three different encodings with CP solver in PICAT and our encoding with SD+RFC and SD+RAC-1 algorithm at each probability p from 0.45 through 0.95.

We encoded the same problems from the previous subsection with the three previously studied CSP modelings of multiple permutation problems in (Walsh, 2001; Dotu et al., 2003; Ansotegui et al., 2004; Hnich et al., 2004; Ansotegui et al., 2006; Mateu, 2009; Ansotegui et al., 2011). We chose to do this in PICAT (Zhou et al., 2015) because it is an easy programming language to learn and it also comes with a built in CSP solver called CP. This CSP solver comes with the exact tools that we need, a constraint propagation algorithm that maintains at most arc-consistency and an option to use the SD heuristic, which they call it the FF heuristic. The first encoding consists of setting alldifferent constraints on the box, column and row dual variables, which corresponds to the $\neq$ between pair of primal variables. (We should note that alldifferent constraint in PICAT maintains at most AC and not GAC.) Then we created the necessary channeling constraints for the *bi-channeling* model and then the *triangular channeling* model, which we previously discussed in section 2. We again set the time limit to 120 seconds per each instance.

We observe from Table 3 that the $\neq$ model with a generic AC algorithm performs all most comparable to our *natural combined* model coupled with our RFC algorithm, where the only times the $\neq$ model performs better are at probabilities $p = 0.5$ and $p = 0.6$. Neither of them can solve any instances at the phase-transition point within the allocated time. Then our *natural combined model* with our RAC-1 algorithm performs better at the phase-transition point than the *bi-channeling* and the *triangular channeling* models with a generic AC algorithm. The mean times and standard deviation error (sde) at the phase-transitions point were as follows: the *bi-channeling* model had a mean time of 100 seconds and sde of 3.6, the *triangular channeling* model had a mean time of 106 seconds and sde of 3.5 and RAC-1 had a mean time of 94 seconds with a sde of 3.2. It is interesting to see that the *triangular channeling* model performed worse than the *bi-channeling* model, but not too surprising as previously indicated in section 2. Furthermore, we also observe a second peak of hardness at probability $p = 0.95$ for the *bi-channeling* and the *triangular channeling* models, which is consistent with our earlier observations.

6.3 Performance of the integrated dynamic variable and value ordering heuristics

	p = 0.5	p = 0.55	p = 0.6	p = 0.65	p = 0.7	p = 0.75	p = 0.8	p = 0.85	p = 0.9	p = 0.95
TNDR	92.5%	60.5%	82%	77.5%	89%	88.5%	95%	96%	99%	100%
TNFV	93%	63.5%	81%	84.5%	81.5%	93.5%	97%	97%	98.5%	100%
TNDR||FV	98%	71.5%	92%	98%	98.5%	98.5%	99.5%	98.5%	100%	100%

Table 4: Percentage of solved instances for TNDR, TNFV and TNDR||FV at each probability p from 0.5 through 0.95.

We simulated the same problem set from the previous two subsections with our integrated dynamic variable and value ordering heuristics (as described in Definitions 4.1 and 4.2) , where we once again set the time limit to 120 seconds per each instance. We observe from Table 4 that depicts the percentage of solved instances that the best performing heuristic is clearly the parallel one. More specifically, the $(TN(DR||FV) \oplus RAC)$ heuristic solved 71.5% of the instances at the phase-transition point with a mean time of 51 seconds and a sde of 3.5, compared to 60.5% of the instances with a mean time of 67 seconds and a sde of 3.8 for the $((TNDR) \oplus RAC)$ heuristic and 63.5% of the instances with a mean time of 64 seconds and a sde of 3.7 for the $((TNFV) \oplus RAC)$ heuristic. We also observe a second peak in hardness at different locations for the $((TNDR) \oplus RAC)$ and the $((TNFV) \oplus RAC)$ heuristics, which are at probabilities $p = 0.65$ and $p = 0.7$ respectively. It has been shown in (Hulubei and O'Sullivan, 2005; Hulubei and O'Sullivan, 2006) that this region could be mitigated. Indeed, we observe that the $(TN(DR||FV) \oplus RAC)$ heuristic can solve all most all of the instances from probability $p = 0.65$ and onwards.

6.4 Performance of the integrated dynamic variable and value ordering heuristics with very hard instances

The puzzles we generated in the previous section and employed in the previous subsections have their holes punched randomly anywhere on the board according to the given probability.

However, it has been shown in (Ansotegui et al., 2006; Ansotegui et al., 2011; Mateu, 2009) that as the punched holes are more balanced among the different permutations of the problem, then it gets harder to solve the given problem. They introduced three different methods to punch holes in a balanced way as follows: the first method balances the punched holes in the columns and rows, the second method also balances the punched holes in the box and the third method also balances the punched holes in the columns and rows within the box. We obtained a set of 200 set of Sudoku puzzles that are of the second kind with 346 punched holes ($p = 0.5536$) from the authors of these papers. This same problem set was simulated on MAC-LAH of (Ansotegui et al., 2004) and Minion of (Gent et al., 2006) using the *bi-channeling* model, where each instance was allowed to run for 10,000 seconds. It was reported in (Ansotegui et al., 2006; Mateu, 2009; Ansotegui et al., 2011) that MAC-LAH was able to solve 42% of the instances whereas Minion was not able to solve any of the instances within the given time limit of 10,000 seconds per instance.
Instead of replicating these simulations, we cut down the allowed time for each instance to run from 10,000 seconds to 1,000 seconds. The results of our simulations with our integrated dynamic variable and value ordering heuristics were as follows: The $(TN(DR||FV) \oplus RAC)$ heuristic solved 55.5% of the instances with a mean time of 594 seconds and sde of 29.9. The $((TNDR) \oplus RAC)$ heuristic solved 43% of the instances with a mean time of 702 seconds and sde of 28.4. The $((TNFV) \oplus RAC)$ heuristic solved 48.5% of the instances with a mean time of 668 seconds and sde of 27.9.

7 Conclusion

We introduced a novel way to model a given multiple permutation problem as a CSP. Then we proved that the constraints enforced by the *natural combined model* are as tight as the channeling constraints between primal and all dual variables. Then we developed a custom made FC algorithm for our modeling and then extended this to an AC algorithm. We empirically showed that our combined methods perform better than the previous CSP modelings with *channeling constraints* that employs a generic AC algorithm. We also showed that adding various enhancement methods to our AC algorithm ($RAC-1$ through $RAC-5$) that either reduced propagation redundancy and/or found local-inconsistencies sooner improved the performance at the phase-transition point, despite the overhead that such additional filtering incurs. However, it's effect at the heavy tail region varied because of the algorithm dependent nature of this region. Finally, we introduced two dynamic variable and value ordering heuristics that are integrated into our AC algorithm. We showed that these heuristics gave a boost to our AC algorithm compared to the plain vanilla heuristics. Furthermore, their performance were better than the state-of the art CSP solvers when simulated with very hard instances.
The encodings, consistency algorithms and the dynamic variable-value ordering heuristics that we introduced are not solely for multiple permutation problems. They can be adopted for other permutation problems, where we also include any constraint satisfaction problems in this set since they can be easily modeled at least two ways. Furthermore, the integrated dynamic variable-value ordering heuristics that we introduced can be combined with any type

of consistency algorithm and can be employed by any CSP solver.
We hope to continue our research in two areas. First branch being coming up with more powerful consistency algorithms than the AC algorithm given our *natural combined model*. And the other branch being coming up with dynamic variable - value ordering heuristics that introduce different as well as multiple tie-breakers.

References

Ansotegui, C., Bejar, R., Fernandez, C., Gomez, C. and Mateu, C. 2006. The impact of balancing on problem hardness in highly structured domain., *AAAI Conference on Artificial Intelligence - 06.* pp. 121–129.

Ansotegui, C., Bejar, R., Fernandez, C., Gomez, C. and Mateu, C. 2011. Generating highly balanced sudoku problems as hard problems., *Journal of Heuristics.* **17**: 589–614.

Ansotegui, C., Del Val, A., I., D., Fernandez, C. and Manya, F. 2004. Modeling choices in quasigroup completion: Sat vs csp., *AAAI Conference on Artificial Intelligence - 04.* pp. 173–42.

Cazenave, T. 2006. A search based sudoku solver., *Labo IA Dept. Informatique Universite Paris* p. 93526.

Chaimowicz, L. and Machado, M. 2011. Combining metaheuristics and csp algorithms to solve sudoku., *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on* pp. 124–131.

Cheng, B., Choi, K., Ho-Man Lee, J. and Wu, J. 1999. Increasing constraint propagation by redundant modeling: An experience report., *Constraints.* pp. 167–192.

Cheng, B., Lee, J. M. and Wu, J. 1996. Speeding up constraint propagation by redundant modeling., *The International Conference on Principles and Practice of Constraint Programming - 96.* pp. 91–103.

Choi, C. W., Lee, J. and Stuckey, P. 2003a. Propagation redundancy for permutation channels., *International Joint Conference on Artificial Intelligence - 03.* pp. 1370–81.

Choi, C. W., Lee, J. and Stuckey, P. 2003b. Propagation redundancy in redundant modeling., *The International Conference on Principles and Practice of Constraint Programming - 03.* pp. 229–243.

Choi, C. W., Lee, J. and Stuckey, P. 2008. Removing propagation redundant constraints in redundant modeling., *ACM Transactions on Computational Logic.* **8**.

Coarfa, C., Demopoulos, D., San Miguel Aguierre, A., Subramanian, D. and Vardi, Y. M. 2000. Random 3-sat: The plot thickens., *The International Conference on Principles and Practice of Constraint Programming - 00.* pp. 143–159.

Colbourn, C. 1984. The complexity of completing partial latin squares., *Discrete Applied Mathematics.* **8**: 25–30.

Dotu, I., Del Val, A. and Cebrian, M. 2003. Redundant modeling for the quasigroup completion problem., *The International Conference on Principles and Practice of Constraint Programming -03.* pp. 288–302.

Gent, I. P., Jefferson, C. and Miguel, I. 2006. Minion: A fast scalable constraint solver., *European Conference on Artificial Intelligence - 06.* pp. 98–102.

Gent, P. I. and Walsh, T. 1994. Easy problems are sometimes hard., *Artificial Intelligence.* **70**: 335–345.

Gomes, C., Selman, B., Crato, N. and Kautz, H. 2000. Heavy-tailed phenomena in satisfiability and csp., *Journal of Automated Reasoning.* **24**: 67–100.

Gomes, C. and Shmoys, D. 2002a. Completing quasigroups or latin squares: A structured graph coloring problem., *Computational Symposium on Graph Coloring and Extensions.* .

Gomes, C. and Shmoys, D. 2002b. The promise of lp to boost csp techniques for combinatorial problems, *International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming - 02.* pp. 291–305.

Haralick, R. M. and Elliot, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems., *Artificial Intelligence.* **14**: 263–313.

Hnich, B., Smith, B. and Walsh, T. 2004. Dual modelling of permutation and injection problems., *Journal of Artificial Intelligence Research.* **21**: 357–91.

Hogg, T. and Williams, C. 1994. The hardest constraint problems: A double phase-transition., *Artificial Intelligence.* **69**: 359–377.

Horsch, M. C. and Havens, W. S. 2000. Probabilistic arc consistency: A connection between constraint reasoning and probabilistic reasoning., *Uncertainty in Artificial Intelligence Proceedings.* pp. 282–90.

Hulubei, T. and O'Sullivan, B. 2005. Search heuristics and heavy-tailed behavior., *The International Conference on Principles and Practice of Constraint Programming - 05.* pp. 328–57.

Hulubei, T. and O'Sullivan, B. 2006. The impact of search heuristics and heavy-tailed behavior., *Constraints.* **11**: 159–78.

Kautz, H. A., Ruan, Y., Achlioptas, D., Gomes, C. P., Selman, B. and Stickel, M. 2001. Balance and filtering in structured satisfiable problems., *International Joint Conference on Artificial Intelligence - 01.* pp. 351–8.

Lewis, R. 2007. Metaheuristics can solve sudoku puzzles., *Journal of Heuristics.* **13**: 387–401.

Mackworth, A. 1977. Consistency in networks of relations., *Artificial Intelligence.* **8**: 99–118.

Martin, D., del Toro, R., Haber, R. and Dorronsoro, J. 2009. Optimal tuning of a networked linear controller using a multi-objective genetic algorithm and its application to one complex electromechanical process, *International Journal of Innovative Computing, Information and Control* **5**(10): 3405–3414.

Mateu, C. 2009. Csp problems as algorithmic benchmarks: measures, methods and models., *PhD Dissertation, Universitat de Lleida.* .

Michel, L. and Van Hentenryck, P. 2011. Activity based search for black-box constraint programming solvers., *arXiv preprint arXiv:1105.6314* .

Mitchell, D. G. and Levesque, H. J. 1996. Some pitfalls for experimenters with random sat., *Artificial Intelligence.* **81**: 111–125.

Mohr, R. and Henderson, T. 1986. Arc and path consistency revisited., *Artificial Intelligence.* **28**: 225–233.

Niño, V. P. M. 2010. Nurse rostering problems, *International Journal of Artificial Intelligence* **5**(A10): 109–116.

Pay, T. 2015. Some enhancement methods for backtracking-search in solving multiple permutation problems, *PhD Dissertation, Graduate Center of New York, CUNY.* .

Precup, R.-E., Tomescu, M. L., Rădac, M.-B., Petriu, E. M., Preitl, S. and Dragoş, C.-A. 2012. Iterative performance improvement of fuzzy control systems for three tank systems, *Expert Systems with Applications* **39**(9): 8288–8299.

Ramírez-Ortegón, M. A., Märgner, V., Cuevas, E. and Rojas, R. 2013. An optimization for binarization methods by removing binary artifacts, *Pattern Recognition Letters* **34**(11): 1299–1306.

Raza, Z. and Vidyarthi, D. P. 2008. A computational grid scheduling model to minimize turnaround using modified ga, *International Journal of Artificial Intelligence* **3**(A09): 86–106.

Refalo, P. 2004. Impact-based search strategies for constraint programming., *The International Conference on Principles and Practice of Constraint Programming - 04.* pp. 556–71.

San Miguel Aguirre, A. and Vardi, M. Y. 2001. Random 3-sat and bdds: The plot thickens further., *The International Conference on Principles and Practice of Constraint Programming - 01.* pp. 121–36.

Selman, B. and Kirkpatrick, S. 1976. Critical behavior in computational cost of satisfiability testing., *Artificial Intelligence.* **81**: 273–295.

Simonis, H. 2005. Sudoku as a constraint problem., *Workshop on Modelling and Reformulating Constraint Satisfaction Problems.* pp. 13–27.

Smith, B. 2001. Dual models of permutation problems., *The International Conference on Principles and Practice of Constraint Programming - 01.* pp. 615–619.

Solos, I. P., Tassopoulos, I. X. and Beligiannis, G. N. 2016. Optimizing shift scheduling for tank trucks using an effective stochastic variable neighbourhood approach, *International Journal of Artificial Intelligence* **14**(1): 1–26.

Walsh, T. 2001. Permutation problems and channeling constraints., *International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - 01.* .

Whitlock, P. and Lambert, T. 2010. Generalizing sudoku to three dimensions., *Monte Carlo Methods and Applications.* **16**: 251–63.

Whitlock, P. and Mayorov, M. 2011. Parallelization of algorithms for solving a 3d sudkou, *Monte Carlo Methods and Applications.* pp. 145–153.

Yato, T. and Seta, T. 2003. Complexity and completeness of finding another solution and its application to puzzles, *IEICE transactions on fundamentals of electronics, communications and computer sciences* **86**: 1052–1060.

Zhou, N.-F., Kjellerstrand, H. and Fruhman, J. 2015. *Constraint Solving and Planning with Picat*, Springer.