A SOLUTION-CLUSTERS BASED APPROACH TO SOLVE HARD CONSTRAINT SATISFACTION PROBLEMS

By

Himanshu Neema

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

In

Computer Science

May, 2008

Nashville, Tennessee

Approved: Date:

_____ _____

_____ _____

To my beloved wife, Reena.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude towards my academic and research advisor Dr. Gabor Karsai for helping me through this academic pursuit. I am grateful to him for providing me with the necessary guidance and supervision, for constantly motivating me to finish this research, and for his humility and kindness to allow me to study at Vanderbilt University and conduct this research.

I also extend my sincere thanks to Dr. Gautam Biswas, my second approver, for his thorough evaluation of this thesis and providing his detailed and invaluable feedback.

I also thank Chris van Buskirk for thoroughly reviewing this thesis and providing very useful comments and advices for improving its content.

I am also thankful to Vanderbilt University for giving me the opportunity to study and pursue my Masters degree part-time while working here.

Finally, but not the least, I offer my sincere thanks to my wife, Reena Neema, for her motivating and cheering support, and for always being there to support my work in every possible way.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Constraint Programming (CP), with a focus on solving optimization problems that involve combinatorial search for satisfying varied constraints, represents a major breakthrough in programming language innovation because they allow problem solving to be abstracted to much higher levels than is possible with traditional programming languages like C, C++, or Java. A *constraint* represents a specific relationship among the program variables – an *invariant* – that must hold in any solution of the given problem. Not only does constraint programming allow direct representation of such relationships, but, as a *declarative* programming approach, it allows complete separation of problem description from problem solving. This clear separation enables one to program a range of problem modeling and problem solving strategies in parallel without having to adjust the other. This is a truly remarkable and essential feature for solving large combinatorial problems where not only the problem typically requires experiments with a large number of problem modeling and search strategies, but the problem itself keeps evolving continually with the application domain. Traditional programming languages do not provide a direct way of representing problem invariant relationships, and they may require huge efforts in maintaining and keeping the program current with the application domain evolution.

Large combinatorial search problems are pervasive in many areas of science and engineering and occur with various levels of complexity. However, most of these

problems have proved to be computationally intractable (NP-Hard) [1] and require enormous amount of computation, despite the availability of efficient problem models and search heuristics (e.g., planning scheduling, sequencing, and resource allocation problems). These problems can be formulated in both the continuous and/or discrete domains. However, most combinatorial search problems are usually cast as Finite Domain (FD) discrete Constraint Satisfaction Problems (CSPs) – which consists of a *finite set of variables* each of which must be assigned a value (or values) from its given *finite domain* of possible values, and a *finite set of constraints* that restrict the set of values that the variables may assume simultaneously (see Chapter II for more). This is because, practically, a finite range of values is more suitable for casting the domains of variables of most industry problems. Finite Domain Constraint Programming is the term used when such a representation is chosen for the problems.

Despite its huge success, there are several limitations of the above approach. The most crucial issue is that of scalability. As the size of the combinatorial problems grow, highly efficient heuristics – that were previously successful – can suddenly become non-productive in finding solutions in a reasonable amount of time. With exponential complexity, solution time for these problems can increase dramatically with increase in the size of the problem. Although a reasonable compromise can usually be found – albeit after extensive trials of various models and heuristics – between the quality of the solution found and the time it takes to find a solution, it can be painstakingly difficult to maintain/adapt this compromise as the problem evolves with time. This is due to the fact that clear separation of problem representation and problem solving – howsoever good it might be for parallel development and re-use –

looses the cause-and-effect relationship between the two. Thus, with problem evolution, with the presence of heterogeneous constraints, the problem characteristics can change drastically with respect to the solving strategy that was developed for it. Owing to these difficulties, it becomes necessary to devise ways that might ease the burden of reinventing the efficient models and search methods.

One way of dealing with it is to use *hybrid* approaches that combine several techniques from Constraint Programming (CP), Artificial Intelligence (AI), and Operations Research (OR). There is already a growing interest in the research community in this direction and several projects are currently investigating such possibilities [4, 5]. A second way is to devise *anytime algorithms* that find a partial solution quickly and work over time to improve the quality of the solution using techniques like local search and greedy search. This guarantees the presence of a solution at any point, although the quality of the solution may not be that high as one might like. Yet another way, and probably more scalable one, is to divide the problems in reasonably small chunks and develop efficient algorithms for solving these individual sub-problems, providing effective communication among the sub-problem solvers, ensuring *global consistency* of the overall solution, and combining individual sub-problem solutions to find a globally consistent solution for the original problem. One such method currently into research is Distributed Constraint Satisfaction [3]. This thesis uses the third method of “divide and conquer” by exploiting and expanding the well-known theory of solution clusters [2].

To date, solution clusters have only been successfully applied for 3-SAT problems [26] due to the availability of techniques for characterizing their runtime

complexity [6]. This thesis develops a theory for applying the theory of solution clusters in the area of Constraint Satisfaction Problems (CSPs).

The rest of the thesis is organized as follows. Chapter II gives a brief introduction to constraint programming and several of its techniques that are used to solve Constraint Satisfaction Problems. It also presents several examples of Constraint Satisfaction Problems to illustrate how a typical search proceeds while searching for a solution. Finally, it highlights some of the difficulties that CP faces and provides the motivation for using distributed coarse-grained search mechanisms. Chapter III provides a background on the theory of solution clusters as developed by Parkes [6] for 3-SAT problems. Chapter IV develops the complete approach of using solution clusters based distributed search to Constraint Satisfaction Problems. Chapter V presents some case studies and results of the experiments performed. Finally, in Chapter VI, the thesis concludes and identifies areas for future work.

CHAPTER II

CONSTRAINT PROGRAMMING

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

– Eugene C. Freuder, Constraints, April 1997

Introduction to Constraint Programming and Constraint Satisfaction Problems

Constraint Programming (CP) is a powerful programming technique to declaratively specify and effectively solve large combinatorial and optimization problems. Although constraint satisfaction problems were being studied in the seventies, it is only in last two decades that this technique gained huge momentum. Since then it has been successfully applied in various application domains like planning, scheduling, DNA sequencing, stock trading analysis, graph coloring, and resource allocation [13].

A *constraint* represents a relationship that must hold – an *invariant* – among the participating variables in any solution of the given problem. Constraints provide a natural way to express problem requirements. A constraint represents the relationship among a set of variables, for example, the relationship between the radius of a circle and its area or the relationship between end time of a preceding task and the start time of a succeeding task sharing the same single capacity resource.

A Constraint Satisfaction Problems (CSP) consists of a *finite set of variables* each of which must be assigned a value (or values) from its given *finite domain* of possible values, and a *finite set of constraints* that restrict the set of values that these variables may assume simultaneously. A *solution* to the CSP consists of an assignment of a value (or values) to each of its variables (called *binding*) such that all constraints of the problem are satisfied. The set of variables and their domains taken together is often referred to as the *constraints store*. Also, *Constraint Satisfaction* differs from *Constraint Solving* where the domains of variables can be infinite and, instead of combinations and search, numerical methods are used.

A constraint can either be *unary*, *binary*, *tertiary*, or *n-ary* (affecting *n* variables) depending on the number of variables it restricts the domains of. A simple example of a unary constraint is $X \neq 10$. This constraint restricts the domain of only one variable, viz., *X*. As soon as the value 10 is removed from *X*'s domain, this constraint can no longer contribute toward restriction of domain in the future. This is referred to as *entailment* of the constraint by the constraint store. Constraints affecting more than two variables can be easily converted to an equivalent set of binary constraints using several new auxiliary variables (called *binarization of constraints*). A CSP containing only unary and binary constraints is called a *binary CSP*. Thus, prior to solving, all CSPs can be, and usually are, converted to binary CSPs for which a lot is already known – viz., efficient search methods, useful search heuristics, several polynomial-time special cases, and special-purposed algorithms [12].

A constraint is called *hard* if it must be satisfied for a solution to be feasible. Whereas constraints that are only desirable to be satisfied are called *soft constraints*.

An *optimization problem* is a computational problem in which the objective is to find the best among all possible solutions of the problem (e.g., minimizing a cost function). Thus, for a CSP containing both hard and soft constraints, if the goal is to satisfy all hard constraints, but only as many soft constraints as possible, then the CSP can be casted as an optimization problem and the best solution can be determined using the classical *branch and bound* techniques [13].

Solving Constraint Satisfaction Problems

To find a solution of a CSP, most constraint satisfaction algorithms use *systematic searches* by trying combinations of values for variables and checking if they are *consistent* with the constraints of the CSP. In contrast to anytime algorithms, these are *complete* algorithms which guarantee that they will find a solution if one exists. However, they do not return until they have found the solution or have determined that no solution to the CSP exists. This section continues below by first introducing a few basic concepts related to constraint solvers which use systematic search methods. Next, the search procedure for constraint solving is discussed in detail, along with several search heuristics that are commonly used for improving the efficiency of constraint solvers.

Instead of *generating-and-testing* for consistency of all possible assignment combinations, which can be enormously large, a *backtracking search* is usually employed where the assignments proceed in a tree structure starting from a *root* variable toward the last *unbound* variable (*leaf* of the tree) while constantly maintaining the consistency of the constraints. When a variable is assigned a value

from its domain, the domains of the connecting variables (via constraints) may get reduced to a currently consistent set. For example, if there is a constraint $X \neq Y$, and a value for X is chosen during search; that value can be immediately removed from Y's domain – which otherwise may lead to the failure of the inequality constraint between X and Y. This is called *domain reduction*. Moreover, domain reduction achieved via one constraint can further affect domains of other variables in other constraints owing to their relationships with the variables whose domains are being currently reduced. This is called *constraint propagation*.

While *node-consistency* [13] ensures that the current domain of a variable is consistent with its unary constraints, *arc-consistency* [13] algorithms utilize constraint propagation and domain reduction to ensure that all binary constraints (called *arcs* of the search tree – a graph) are satisfiable with each of the values in the current domains of variables at both ends of the arc. Several arc-consistency algorithms have been developed to increase the efficiency of constraint satisfaction. The arc-consistency can also be taken to higher levels of consistency that ensure that for a consistent assignment of any *K-1* variables, any K^{th} variable can be assigned at least one value from its domain that is consistent with the constraints of the CSP (*K-consistency*) [13]. Although higher levels of consistency provide stronger constraint propagation, executing them at every node of the search tree requires significant run-time. In general, arc-consistency algorithms represent the best trade-off between the run-time efficiency and constraint propagation achieved. Therefore, in backtracking search algorithms, at each variable node of the *search-tree* – called a *choicepoint* –

arc-consistency algorithms are used for constraint propagation followed by search for the assignment of the next variable in the tree.

The above search procedure employs the strategy of randomly choosing the variables at the choice nodes of the search tree. Also, by default, it chooses the first available value in the domain of a variable as the next assignment to try to find the solution of the CSP. Furthermore, it keeps choosing the next variable and a value from its domain until it finds a variable with an empty domain (*constraint failure*) at which point it backtracks to the previous variable to try a new value from its domain if available. If, at any point, domains of all variables become empty, the CSP is declared as unsolvable. This is just a naïve example of a *distribution strategy* employed for solving a CSP. In general, a distribution strategy can involve use of any scheme for *variable ordering* – to order the variables in the search tree, *value ordering* – to choose a value assignment order among the values in the domains of variables, and *search strategy* – to explore the search tree (e.g., *Depth-First Search* or *Breadth-First Search*). These three mechanisms can be combined in various ways, usually employing domain-specific rules of thumb – called *heuristics*, leading to a vast number of distribution strategies. A reasonable approach for variable ordering is based on the *first-fail principle* which states that it is more efficient to fail early and get rid of inconsistent values from variable domains. This variable ordering technique, referred to as *Minimum Remaining Value (MRV)* heuristic, chooses that variable first which has fewest values left in its domain. Similarly, a reasonable approach for value ordering is to choose that value first which is in the least number of conflicts with connected variables. This value ordering technique aims to increase

the chances of finding a satisfying assignment faster and is called *Least Constraining Value (LCV)* heuristic. As far as a default mechanism for search strategy is concerned, Depth-First Search usually works well for most problems owing to its well-known significantly small memory foot-print. However, in general, finding a good overall distribution strategy often becomes an arduous task depending on the complexity of the CSP being solved.

Over the past several years a huge amount of work [14, 15, 16] has been done to increase the efficiency of constraint satisfaction solvers by finding novel ways to increase the amount of propagation specialized constraints can provide, smartly detecting previously seen conflicts by storing a set of smallest possible *no-good constraints* (inconsistent assignments) [21], and speeding up the search to arrive at solutions faster. However, with the growing complexity of real-world applications (CSPs), optimal solutions are highly difficult to find. Usually, a "*good enough*" solution is considered better if it satisfies most of the hard constraints of the CSP and gets generated in a reasonable amount of time. However, as described above, a large number of factors affect the chosen distribution strategy for solving the CSP and often it becomes highly sensitive to small changes in the problem descriptions. Real-world problems, on the other hand, evolve continuously over time, and it becomes unmanageable to keep refining the distribution strategies as the problem changes. Scalability is a major problem with constraint satisfaction algorithms which often become unusable as the size of the problem grows in various ways. Suddenly an algorithm that worked for several problems can become unreliable to the point that it can no longer find a solution of the CSP in a reasonable amount of time.

Issues with CP and motivation for distributed solving

Owing to these scalability issues and exponential complexity of constraint satisfaction algorithms, and to deal with sensitivity issues of distribution strategies with problem evolution, it has become essential to investigate new ways to address these issues. As previously described, one way is use hybrid approaches by combining techniques from various fields like Operations Research, Constraint Programming, and Artificial Intelligence. Alternatively, a range of anytime algorithms using techniques like Local Search [17, 18, 19], Greedy Local Search [20] may be employed. Recently, however, there is a growing interest in finding ways to divide the problem into smaller sub-problems that can be easily solved separately in an independent or co-operative fashion, possibly on separate machines and potentially distributed over the internet. This approach is called *Distributed Solving* and the problem where the set of variables and constraints is distributed among multiple sub-problem solvers is called a *Distributed-CSP* [3]. Although this approach involves a great amount of communication overhead among the individual solvers working on sub-problems [3, 8], they are increasingly more preferred due their capability to deal with scalability issues of solving the ever growing complex combinatorial problems. This thesis presents a distributed solving approach for CSPs using a solution clusters based approach.

CHAPTER III

THEORY OF SOLUTION CLUSTERS FOR DISTRIBUTED SEARCH

Introduction

A *solution cluster* is a region of CSP variables' assignment space that contains a rich set of solutions to the CSP. Typically, in a solution cluster, a large portion of the variables are already bound to a value and only a few constraints remain that have not yet been entailed. The theory of solution clusters was first developed by Andrew J. Parkes [6] and is based on the observation that solutions of a CSP tend to cluster together in the assignment space [2]. One example is of finding an optimal schedule that minimizes the *makespan* – time elapsed between start of the first task and completion of the last task – for a scheduling problem. Despite the fact that optimal schedules are extremely rare compared to the size of the solution space, it can be easily shown that an optimal schedule usually lies within a solution rich solution cluster. This is due to the fact that for the schedule that minimizes the makespan even though the tasks that are on its critical path cannot be moved around much in time without increasing the makespan, a large number of other tasks that lie outside the critical path can usually be moved in time without affecting the overall makespan of the schedule. This leads to a solution cluster when all of the tasks that lie on the critical path have been bound to a start time.

Clustering at the Phase Transition

It has been shown by Selman and Kirkpatrick [7] that constraint satisfaction problems have marked characteristics with regard to their complexity of solving. Problems that have a large number of solutions are clearly easily solvable due to the large number of satisfying assignments. On the other hand, problems that have a significantly small number of solutions are also easily solved owing to a huge domain reduction achieved due to a large number of inconsistencies discovered much earlier in the search. In fact, it is the problems that possess a smaller number of solutions (but not so small that they become easily solvable) are the hardest among constraint satisfaction problems. This is typically marked by the occurrence of a phase-transition[1] in the problem complexity with variation in the number of constraints or the number of variables. In the paper [2] Parkes studied the complexity phenomenon of *3-SAT* problems represented in Conjunctive Normal Form (CNF) (see [25] for more on CNF, and [26] for more on 3-SAT problems[2]). The outcome of this study confirmed the phase transition curve that relates the difficulty of the problem with the ratio of number of clauses to the number of variables in the problem. Also, it was shown that at the phase transition, the solutions of the 3-SAT problems get clustered into separate regions such that it becomes difficult to jump from one region to the

[1] Phase-transition here refers to an abrupt change in the difficulty level of the problem. This is validated by observing a sharp increase in runtime for solving the problem. The term phase-transition is taken from the field of thermodynamics, where it refers to an abrupt change in one or more physical properties of a thermodynamic system (particularly the heat-capacity) owing to a slight change in the system's environment. One familiar example of phase transition is that of water which abruptly changes its states from ice to liquid water at 0^{o}C and liquid water to gas at 100^{o}C.

[2] In a *3-SAT problem* all variables are Boolean variables with only values TRUE and FALSE. A *literal* is either simply the variable itself (e.g., *x1*), or the variable's negated form (e.g., *~x1*). A *clause* is a disjunction of literals (e.g., *[x1 OR ~x3 OR x5]*), such that for the clause to be TRUE one or more of its constituent literals must be TRUE. A 3-SAT problem consists of a formula formed by a conjunctions of clauses with maximum 3 literals per clause. Thus, for a 3-SAT formula to be TRUE all its constituent clauses must be TRUE. For example, *[(x1 OR ~x2 OR ~x3) AND (x1 OR x2 OR x4)]* is a 3-SAT problem, where *x1*, *x2*, *x3*, and *x4* are all Boolean variables.

other without changing a significantly large number of variable assignments (required by huge backtracking steps). Thus, it was shown that clustering is a direct cause of sharp increase in the runtime complexity of 3-SAT problems at the phase transition. This complexity behavior of satisfiability problems has been published in several publications. A graphical figure illustrating the complexity behavior of satisfiability problems is reproduced below from Bart Selman's website[3].

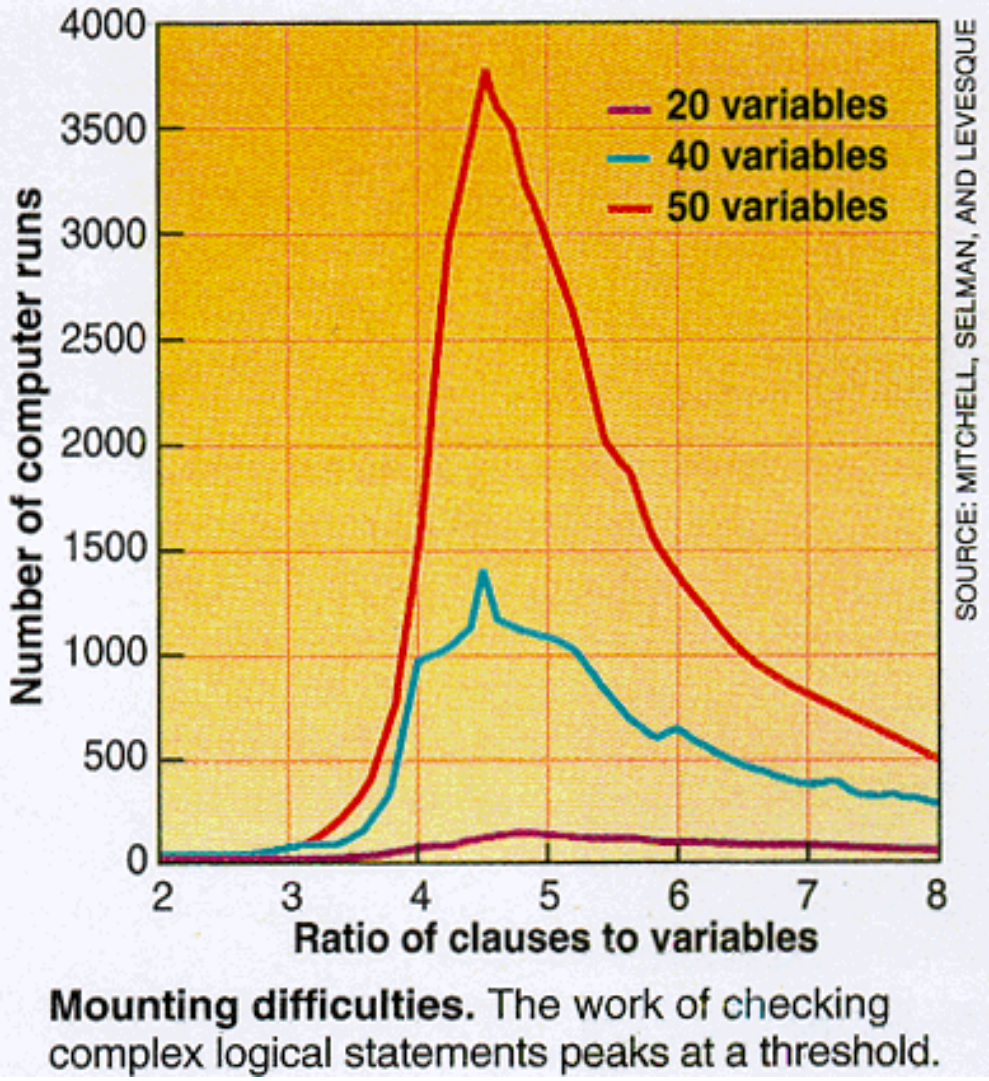

Figure 1: Clustering at the phase transition

[3] http://www.cs.cornell.edu/home/selman/phase2.gif [Credit: Bart Selman]

The above figure shows that typically a phase transition occurs when the ratio of number of clauses to number of variables, increases to ~4.26 – which can be observed in the figure as an abrupt and exponential increase in the runtime of the solvers [10]. This corresponds to the example of finding a task schedule with minimum makespan (given earlier on page 12). In finding the optimal schedule, the tasks that lie on the critical path of the optimal schedule (with minimum makespan) possess the least variability in terms of the start times that can be assigned to them. As such these variables constitute the least flexible variables of the problem (sometimes referred to as *backbone* variables). Interestingly, as soon as these tasks are determined (assigned a start time during the search), the remainder of the search space transitions into a CSP in which the remaining variables usually possess a relatively larger degree of freedom in choosing a start time for them. Thus, the remaining search space usually gets solved much quicker than compared to the time it takes to solve the entire CSP for the optimal schedule. Additionally, when there is not a severe resource contention, the makespan of the resulting optimal schedule does not change while solving the remaining search space. It is noteworthy that depending on the distribution strategy chosen (i.e., variable and value ordering, search strategy), the tasks on the critical path may get bound sooner or much later. Thus, the phase transition can occur at different levels and the corresponding solution cluster found might or might not differ *non-trivially* from other solution clusters.

Exploiting solution clusters for coarse-grained distributed search

Distributed problem solving usually proceeds by dividing the set of variables and constraints among several sub-problem solvers each – trying to solve its sub-problem locally while maintaining global consistency of its assignments by frequently communicating with other solvers. Each solver classifies its set of variables as *public* or *private* depending on whether it participates in constraints of other solvers or not. Thus, while communicating with other solvers only public variables are of interest and the solvers are usually free to choose any *locally consistent* value for their private variables. As such, distributed search usually involves a significantly high communication cost among the individual solvers solving the sub-problems co-operatively while maintaining global consistency of their partial solutions. The huge communication costs arise not only due to the transfer load of the passed information, but mainly because the receiving solver must be interrupted at some point to process the received information and determine whether the current search tree is still consistent not. If not, it might need to restart the search while including the information it has already received from other solvers. This interruption and merging of information creates a huge overhead for the overall distributed search algorithm.

The initial work on solving distributed-CSPs [3] had a very fine-grained distribution, for example, keeping just one variable in the sub-problems. Although keeping distribution fine-grained can be useful in designing simple distributed search algorithms, it is highly unrealistic owing to the huge number of variables in real-world problems. These algorithms are known as Distributed Constraint Satisfaction algorithms and a lot of research [8] is currently being conducted to increase their

efficiency. Also, there has been some work in extending distributed constraint satisfaction to coarser level; however, the huge communication costs associated with distributed search prohibit using these algorithms for generating solutions faster.

Solution clusters, on the other hand, possess a unique characteristic of being able to express a large amount of assignment space with a very compact description (defined in the next chapter). This compact description stems from the fact that a solution cluster is not a collection of solutions but merely a part of the search space with several variables bound in such a way that the remainder of the search space is rich in the number of solutions it contains. Due to this, they can significantly reduce the communication costs among the cooperatively working sub-problem solvers. A single message in the form of a solution cluster can carry an exponential number of sub-problem solutions to other collaborating solvers, thereby reducing the frequent need of messaging among sub-problem solvers. When these large number of solutions are passed in the form of a solution cluster, there is a good potential that some of the solutions may, in fact, be consistent with the current partial solution of the receiving solver.

There is a trade-off between decreasing communication expense by increasing the size of sub-problems and being able to solve the sub-problems themselves in a reasonable amount of time. If the size of the sub-problems is very small (which corresponds to a large number of sub-problem solvers), the smaller sub-problem might be easier to solve, but a lot of communication needs to take place among sub-problem solvers to arrive at a globally consistent solution. Therefore, what is needed is to decrease the amount of communication among sub-problem solvers and still

keep the sub-problem sizes as manageable. As such, using the compact description of solution clusters, while at the same time communicating potentially an exponentially large number of solutions, can significantly reduce the delays in collaboratively arriving at a global solution [9]. As the paper [9] further points out, as the number of solutions passed to a collaborating solver increases, chances are more that one of the solutions passed to it is consistent with its current search tree so that it can avoid backtracking (or at least reduce the amount of backtracking it has to do to become consistent with one of the solutions sent to it). Therefore, solution clusters, by being able to communicate a large number of sub-problem solutions in a single message, can significantly decrease the communication overhead.

CHAPTER IV

APPLYING SOLUTION CLUSTER TO SOLVE A DISTRIBUTED-CSP

The solution clusters theory, as originally developed by Parkes [4], uses the domain of 3-SAT problems [26] to apply its heuristics and algorithms. The main reason for choosing 3-SAT problems was the knowledge of how to characterize their complexity using their ratio of number of clauses to the number of variables (~4.26) – often called as *phase transition point*. While searching for a solution of a 3-SAT problem, as the variables get bound, the remaining problem can be continually checked to determine if it has transitioned into an easily solvable instance. However, as it is not known how to characterize the complexity of CSPs, it is very hard to find solution clusters for them. This is because, in general case, finding solution clusters has already been proven to be NP-hard by Parkes even for 3-SAT problems [4]. As previously mentioned, the theory of solution clusters have already been successfully used to solve hard 3-SAT problems [6]. The goal of this thesis is to build upon the current theory to develop a solution clusters based approach that is applicable to solve hard constraint satisfaction problems (CSPs). Owing to the compact description of the solution clusters and the exponential number of solutions they can contain; it is expected that this results in a significant gain in the efficiency of solving distributed constraint satisfaction problems.

The rest of the chapter is organized as below. First, the assumptions made while developing the theory are stated and the challenges in solving CSPs using a

solution clusters based approach are presented. Next, several preliminary definitions, theorems, and their proofs used in this context are given. Following that, a method for finding clusters for a given CSP is developed. Subsequently, the main algorithms for solving the distributed-CSP using the solution clusters based distributed search are presented. Finally, a runtime complexity analysis of the distributed search procedure is performed and a few guidelines on effectively dividing a CSP into sub-problems are given.

Assumptions

For applying the solution clusters based approach to the distributed search algorithm for solving distributed-CSPs, a few basic assumptions are made. However, most of these assumptions are merely to simplify the development of the theory and are usually true in real-world complex problems. Here is the list of assumptions:

1. The distributed-CSP is complex enough to justify the distributed search using this technique. This is because distributed solving of hard CSPs without the solution clusters approach involves a huge communication cost. Whereas, solution clusters can, despite the cost of finding them in the search space, carry huge amount of information in a highly compact form, thereby providing a significant communication advantage.
2. The distributed-CSP has a large number of solutions, but not large enough that it can be solved trivially. This is because if a CSP contains very few solutions, the constraints of the CSP will perform significant domain reduction during search and the search space will rapidly reduce significantly in size, such that

a solution can usually be found quickly. On the other hand, if a CSP contains a large number of solutions, the problem is under-constrained and therefore the search usually confronts significantly less contradictions, resulting in less backtracking and finding a solution quickly. It is usually the CSPs with not very large and not very small number of solutions that are critically constrained and hard to solve.

3. Each of the sub-problems into which the distributed-CSP is divided into possesses a large number of solutions.
4. The messages passed between two individual solvers, in the form of a solution cluster, contain at least one solution for the sub-problem given to the solver generating the solution cluster.
5. Solution-clusters represent an *under-constrained* constraint problem that is easily and quickly solvable. This is necessary for the efficiency of the distributed search algorithm so that the receiving solvers can respond quickly enough to these incoming solution-cluster messages.

Challenges in applying solution clusters theory to CSPs

To the best of out knowledge, the solution clusters theory has never been applied to solve distributed search for CSPs. There are four major difficulties in applying solution clusters approach to solve constraint satisfaction problems.

Firstly, constraint satisfaction problems are extremely difficult to characterize. For 3-SAT problems, a huge amount of work has been done in studying their behavior and characterizing them in terms of problem parameters. For example, it is

possible to reasonably predict the difficulty of solving a 3-SAT problem – based on the ratio of number of clauses to the number of Boolean variables in the problem – and develop specialized solving procedures that exploit the problem structure [10]. However, characterizing the problem structure of CSPs is extremely hard. In the worst case, CSPs are proven as NP-hard problems; however, it is the average case where the characterization of the CSPs is sought. The main reasons why CSPs are difficult to characterize in terms of their complexity level are:

1. The way CSP problems are formalized is by clearly separating the problem description with problem solving. Although, this is good for parallel development and re-use of modeling and distribution strategies, there occurs quite a big disconnect between problem representation and their solving procedures. In order to estimate the difficulty of a CSP, both the way it has been modeled and the search strategies that will be used for solving needs to be considered. It is well-known that either of these can drastically affect the amount of time in which the entire CSP will eventually get solved.
2. In contrast to 3-SAT problems where each of the constraint is simply a satisfiability clause, CSPs have a marked heterogeneity among the types of constraints that are present. New types of constraints are constantly being developed for increasing the degree and efficiency of constraint propagation. There are usually a number of different ways in which constraints can be defined to represent a domain restriction in the problem.
3. CSPs usually benefit a lot in terms of efficiency of solving them by employing special constraints called "*global constraints*". These are special purpose

procedural constraints which can represent a whole class of constraints in a single propagation routine. One famous example of a global constraint that is present in all major constraint solving libraries is *AllDiff*. This constraint takes a list of variables as input and asserts that each of the variables must be assigned a distinct value from all other variables. Let there be three variables – *X*, *Y*, and *Z* – given to the *AllDiff* constraint, then asserting it will be equivalent to posting several individual inequality constraints, viz., $X \neq Y$, $Y \neq Z$, and $Z \neq X$. Although very useful for constraint propagation efficiency and even characterizable taken as an individual constraint, global constraints pose a big problem when trying to characterize the complexity of a CSP.

4. Moreover, in general, CSPs may also contain any number of non-linear constraints – which by definition are non-trivial to characterize when mixed with several other heterogeneous constraints in the CSP. One simple example of such a constraint is *[(0 <= X <= 90) AND {(X * Sin*^{3}*(X)) < 20}]*. As an individual constraint it is not difficult to find a range of values for *X* where above constraint can be satisfied. However, when the CSP is considered as a whole, where the above constraint could merely be one among several other constraints of the CSP, it becomes non-trivial to combine the effect/characterization of these individual constraints to judge the characterization of the entire set of constraints of the CSP.
5. Furthermore, in contrast to the 3-SAT problem – where each variable is of Boolean type and contains only two values in its domain (viz., TRUE and FALSE) – Finite Domain CSP variables can contain any number of discrete

values in their domain. The larger the domain, the greater the branching factor of the search tree, and hence the longer it can take to investigate all possible assignments.

Secondly, the above difficulty of characterizing CSPs manifests itself into several other problems of not being able to know things like if a current partial solution is completely solvable or if the current partial assignment can be extended into a large number of solutions (i.e., if the current partial assignment is a solution cluster).

Thirdly, in general it is harder, more time-consuming, and more distracting, for CSP solvers to process the received solution clusters. This is mainly due to the need of adjusting the distribution strategy based on new information – which might also require a restart of the search in the receiving solver. Depending on the amount of information contained in the solution cluster for the receiving solver, at times, it might, in fact, save time to start a fresh search instead of backtracking several levels up in a currently deep search tree. However, due to difficulty in characterization of CSPs, it is harder to determine the optimal procedure to incorporate the received solution cluster.

Finally, due to the time-spent in incorporating the solution cluster by the receiving solver, the cluster sending solver experiences an even greater delay in receiving a satisfactory/unsatisfactory response from the receiving solver. The response is needed for sending a new solution cluster (and hopefully non-trivially different – to avoid similar repeating failures) to the receiving solver.

Definitions

This section introduces a few basic definitions, theorems, and their proofs used in developing the theory of using solution clusters for distributed search to solve complex distributed constraint satisfaction problems. It is intended to use a terminology that is as similar as possible to that is known in the 3SAT domain.

Constraint Satisfaction Problem (CSP)

A CSP consists of a finite set of variables $V_1..V_n$, with a corresponding finite domain $D_1..D_n$, and a finite set of constraints C – that restrict the domain of these n finite domain variables. A solution to this CSP consists of an assignment, to each of the variables V_i, of a value from its domain D_i, such that all of the constraints in the set C are satisfied with this assignment.

Distributed Constraint Satisfaction Problem (distributed-CSP)

In a *distributed-CSP*, the set of variables and constraints of the overall CSP is divided among individual solvers working on these sub-problems cooperatively to arrive at a consistent solution for the overall CSP. The variables and constraints can completely belong to a solver or may be shared among more than one solver. However, it is intended that, and useful if, the division of the global CSP among individual solvers be performed in a way that minimizes the number of conflicts over the value assignments of shared variables among the individual solvers.

Full and Partial assignments

A *full* assignment consists of an assignment of a single value to all variables of the CSP. On the other hand, in a *partial* assignment, there is at least one variable of the CSP that is not yet bound to a single value.

Cluster-defining partial assignment (CPA)

This refers to a *partial* assignment to some variables of a CSP such that the remainder of the variables can be assigned quickly in a reasonable amount of time and the current partial assignment contains a large number of solutions (>= 1). A solution cluster can directly be generated from such a cluster-defining partial assignment.

Residual constraints

Any constraint that has not been entailed by the assignments made in a CPA is called a residual constraint – which still needs to be satisfied when further assignments are made.

Solution Cluster

As per previous discussion, a solution cluster is a region of assignment space that is easily solvable and contains a large number of solutions (>= 1). The larger the numbers of solutions in the solution cluster, the better are the chances for it to have fewer contradictions with the solution clusters generated for other sub-problems.

Within the context of distributed-CSPs, this definition of solution cluster is further refined to include only the variables that directly affect other sub-problem solvers.

Pure and Impure CPAs

A CPA in which all of the constraints have been entailed is called a Pure CPA. A Pure CPA contains nothing but solutions to the problems – that is, any assignment of a value to a variable from its current domain remains consistent with the original sub-problem and is part of one of its solutions. This is because there are no constraints left that can be violated by any of the assignments. In contrast, Impure CPAs also contain a set of residual constraints.

Private, Public, and Foreign Variables (*PRV*, *PUV*, *FOV*)

As mentioned above, when a distributed-CSP is divided into sub-problems for individual solvers, each solver is assigned a subset of variables and constraints and these variables can either belong completely to a solver (i.e., not affecting any other solvers – called *private* variables for that solver, and *foreign* variables for the other solver) or they can be shared among the solvers (i.e., their values affect the sub-problems of both the solvers – called *public* variables for the solvers sharing these variables). See the figure below for a pictorial illustration of these different types of variables, constraints in the CSP, and "multiple" solution clusters.

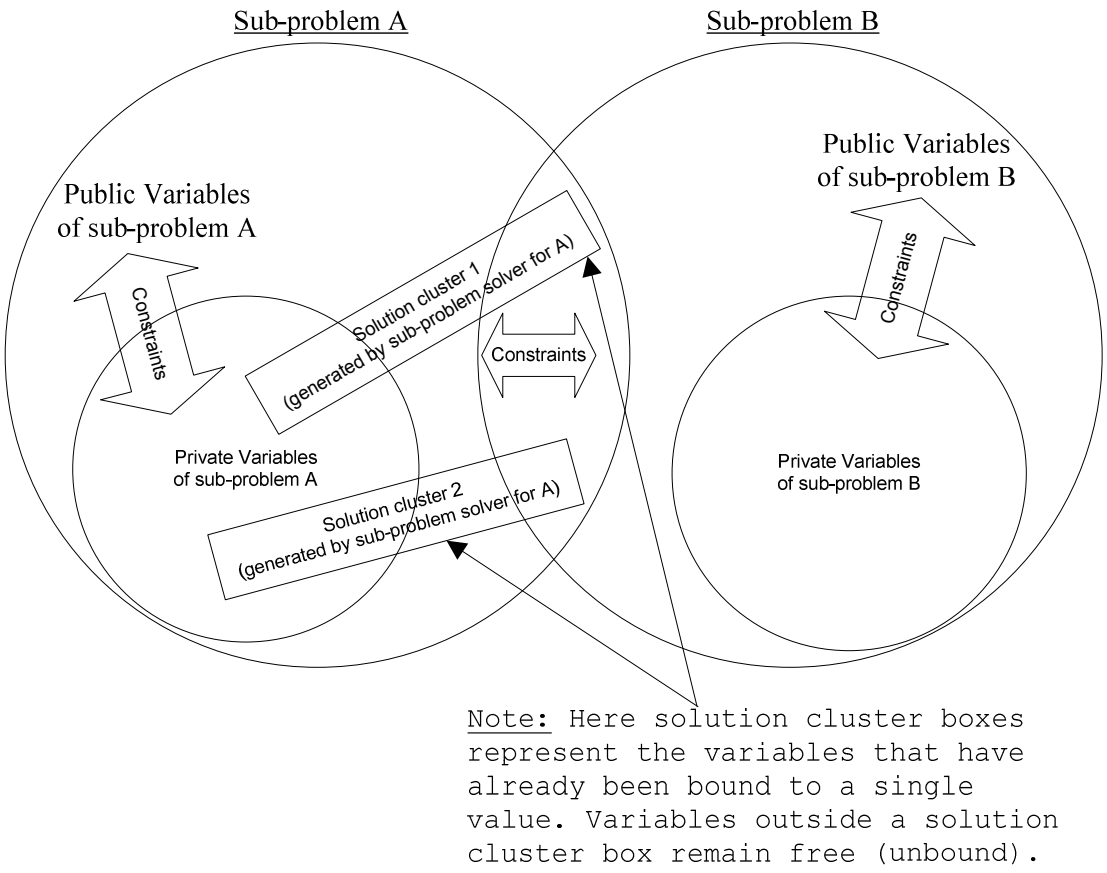

Figure 2: Sub-problem variables, constraints, and multiple solution clusters

Residual Variables

When a cluster-defining partial assignment (CPA) is generated, the variables that have not been bound to a single value are termed as residual variables.

Residual Theory (RT)

Once a CPA is generated with a partial set of variables bound to a single value, its residual variables are relaxed to their *ground domains* (i.e., the domain before any search began) and the constraints of the sub-problems are re-propagated to reduce the domains of these residual variables. This ensures that the domains of all

residual variables are maximal by including every possible solution that may be represented by the CPA. The resulting set of variables and their domains with the constraints that have not been entailed yet is called the residual theory. As a result, any solution of the residual theory is also a solution of the original sub-problem.

Collaborating solvers

When two sub-problems share some of the variables, the solvers working on individual sub-problems need to collaborate with each other in order to ensure global consistency of the solution for the complete CSP. Any pair of such solvers that share some variables are termed as collaborating solvers.

Using variable ownership for the distributed search

As discussed above, among the variables of a sub-problem, only public variables directly participate in some of the constraints of the partial CSP of a collaborating solver. In contrast, private variables do not directly participate in neighbor's constraints. The term "directly" is used because even though the private variables are not present in the constraints of the neighbors, they can still have an affect on the public variables of the same sub-problem via connecting constraints, which in turn can affect the private variables of other solvers. This thesis defines the variable ownership relations such that the updates to the domain of private variables are restricted only to the solver that owns them. Public variables, on the other hand, are jointly owned by solvers for which they are public variables. As such, both solvers are permitted to restrict the domain of public variables. In general, the larger

the number of public variables that are still residual and the larger the domains of public residual variables may be taken to mean a solution cluster with smaller chances of conflict with the collaborating solver. However, when looked at closely, it appears that this is not always true. This is because when public variables of a solver get their domains reduced by propagation and search within a sub-problem solver, it captures the information about conflicts among the public variables themselves and private variables of the same sub-problem solver; but, the final domains of the public variables may still be in conflict with the constraints of other collaborating sub-problem solvers. This is an interesting observation and is useful for generating non-trivially different solution clusters. However, sometimes a less conflicting solution cluster may actually contain smaller number of sub-problem solutions than a solution cluster that is more conflicting with the sub-problem of the collaborating solver.

Checking whether a CPA is solution rich

In order for a collaborating solver to be able to easily solve for a solution cluster passed to it during distributed solving, it is usually helpful if the solution cluster is solution rich (i.e., it contains a large number of solutions). This is necessary to avoid the overhead of time spent in adjusting the current search tree in the receiving solver to incorporate the solution cluster message. Ensuring that a cluster defining partial assignment (CPA) is easily solvable and contains a large number of solutions is a difficult problem and can be solved in the following ways:

1. One can specify a minimum number of solutions that must be present. However, counting the number of solutions is itself a non-trivial and NP-hard

problem. The overhead of time spent in counting the solutions is simply prohibitive.

2. Secondly, one can keep searching and reducing domains until all constraints of the CSP have been entailed. This is also time-consuming and it is possible that at times it may not even be feasible to do so without generating all solutions of the CPA.
3. A reasonably efficient way is to first find a single solution for the sub-problem and generate a solution rich CPA from it. This is the method (described in detail later) chosen in this thesis to generate solution rich CPAs in a reasonably efficient manner.

In general, the efficiency of the distributed search increases if a large number of solutions are present in the solution cluster messages. However, even in the absence of a large number of solutions in solution clusters, the solution clusters based distributed search algorithm often remains useful when compared to solving the distributed-CSP as a standalone problem.

Algorithm for generating a solution rich CPA

Owing to the difficulties of generating a solution-rich CPA from scratch, this thesis resorts to using the third method described above and proposes to generate the solution rich from a known solution of the CSP by unsetting its public variables.

As noted earlier, the goal is to generate a cluster that conflicts as little as possible with the receiving solver. Thus, in general, it is preferable to unset a public variable with the smallest domain (> 1) first as opposed to the general intuition of unsetting a

public variable with the largest domain first. This trivially assumes that the smaller the domain of the public variable, the more controversial that public variable is. However, in the algorithm presented below, after generating a solution, all public variables of the sub-problem are unset, thus making this a non-issue. It is important to note that the resulting CPA is not the same as would have been generated by simply doing a Depth-First Search and setting only the private variables. This is because this algorithm starts from a known solution and unsets public variables (one at a time) to include only the values that result in a solution of the problem.

However, on receiving a solution cluster, the receiving solver may generate a failure at the receiving end. This requires the sender solver to revise the solution cluster such that it becomes *non-trivially* different from any of the clusters that were tried previously. This scenario is handled separately in this thesis. Here is the complete algorithm for generating a solution rich CPA:

1. For the given CSP, find a solution *S* using usual constraint programming techniques. With the assumptions about sub-problem complexity given previously, it can be easily seen that finding the solution *S* for a fraction of original CSP may not be particularly difficult. In fact, while dividing the distributed-CSP among individual solvers, it is assumed that the sub-problems are solvable in a reasonable amount of time.
2. Unset the public variables of the sub-problem from the solution *S* as below:
    a. Choose the next public variable PUV_i to unset, while keeping all other variables assigned as in the solution *S*.

b. Unset PUV_i by resetting its current domain to its ground domain (i.e., the domain it had before starting any search).
c. Propagate all of the constraints of the sub-problem until no more propagation is achievable.
d. Note down the reduced domain of PUV_i as the outgoing domain OD_i to be sent out in the outgoing solution cluster message.
e. Reset the domain of PUV_i back to the value given in the solution *S*.
f. Repeat the above steps until all public variables in the set *PUV* have been processed.

3. Reset the domains of all public variables, PUV_i, to their corresponding outgoing domain OD_i.

The steps above leads to a solution cluster *SC* with the property that the local solver can, without any information about the solution *S*, quickly process it to arrive at a solution. The reasons are twofold. First, the relaxed domains of public variables were generated from a known solution. Secondly, the relaxing of the domain of any variable was limited to contain only the values that were consistent with the constraints of the sub-problem. However, this does not mean that no search will be required while generating a solution *S'* from the cluster *SC*. This is due to the presence of constraints among the public variables.

Once the above solution cluster *SC* is sent as a message to all collaborating solvers, the solver generating the cluster stores *SC* in its internal list of solution clusters that it generated. It also stores the solution *S* used to generate *SC*. It can then

immediately start, in parallel while the collaborating solvers are working on adjusting their search trees for the sent solution cluster message, to generate another solution cluster for the same sub-problem that is non-trivially different than any of the solution clusters generated thus far.

Algorithm for generating a non-trivially different solution cluster

As mentioned earlier, the solution cluster generated by a sub-problem solver may not be acceptable to the receiving solver because of conflicts between the domains of the public variables that were sent in the solution cluster message. In this situation, it is necessary to generate a new solution cluster for the collaborating solver(s) that is *non-trivially* different from the previously generated solution cluster(s). To avoid the cluster-receiving collaborating solver from failing repeatedly to solve for the solution clusters sent to it, it is important that every solution cluster sent to it is non-trivially different from previously sent solution clusters. However, to generate a non-trivially different solution cluster for the same problem with this algorithm, it is necessary to also unset a few *private* variables *PRV* of the solution *S* that was used to generate previous cluster(s). Following this, here is a complete algorithm for generating a solution cluster that it is non-trivially different from all of the previously generated solution clusters (set *SSC*) as below:

1. Let there be a number *k* for the minimum percentage of private variables to unset to get a reasonably non-trivially different solution cluster.
2. Take the previously sent solution cluster and the current set of no-good constraints and propagate constraints to find private variables that get bound

to single value. If more than *(100 – k)* percentage of private variables get determined, then the solution cluster represents a *maximal* solution cluster for *k* unbound private variables (i.e., any other solution cluster will contain a subset of solutions that this solution cluster contains) and is simply returned. Here, on receiving the same solution cluster again, the global CSP search procedure (described later) determines not to backtrack this solution cluster. To avoid this situation from the start, *k* can be chosen accordingly. However, this situation can still arise at a later stage due to the addition of *no-good constraints* [21] to sub-problems (see the overall distributed search algorithm described later for further details). If less than *(100 – k)* percentage of private variables get determined with propagation alone, then unsetting of the needed private variables can be accomplished using the steps described below.

3. Assign a score to each of the private variables PRV_i of the sub-problem that takes into account:
    a. The degree to which PRV_i is constrained. If it gets a smaller domain after unsetting it from a solution, it is considered more constrained and so more important for unsetting to avoid conflicts with the receiving solver.
    b. The number of times PRV_i has been unset in previous solution clusters $NPRV_i$. In order to generate a non-trivially different solution cluster and avoid repeated failures in the receiving solver, it is useful to unset new private variables.

Admittedly, above is simply one *general* scheme to choose a score for selecting the next private variable to unset and that there can be different schemes that may be more appropriate for different problem domains. One example of using the above scoring scheme is shown in the equation below.

$\forall$ i: $0 \leq i <$ Number of private variables

Score for i^{th} private variable PRV_i, $PRVS_i$, is given by:

$$PRVS_i = f_1 * \left[\sum_{j=1}^{SizeOf(SSC)} SizeOf\left(OD_{ij}\right) \right] + f_2 * [NPRV_i]$$

where,

SSC = set of previously computed solution clusters,

$NPRV_i$ = Number of times PRV_i was unset in earlier solution clusters,

OD_{ij} = PRV_i's domain in j^{th} previous solution cluster, and

f_1, f_2 = Tuning factors (determined experimentally)

Here, for each of the solution cluster and its corresponding solution from which the cluster was generated, the outgoing domain OD_{ij} of a private variable PRV_i is determined by following the steps in the algorithm for generating solution clusters. This scoring scheme intends to give more weight to private variables that either result in smaller domains after unsetting or were not unset in previous solution clusters. Also, the factors f_1 and f_2 are determined through experiments and may differ significantly for different CSPs/domains.

4. Using above scores, order the private variables in the order of increasing scores. The smaller the score, the better it is, to unset the private variable for

generating a non-trivially different solution cluster (while still being less conflicting with the receiving sub-problem solver).

5. Take any of the previously computed solution S_{prev}. Here a more complicated procedure for selecting a better solution is possible (e.g., comparing the size of the domains of the variables after unsetting each variable and re-propagating the constraints, and choosing a solution with greater average consistent domain size). However, for simplicity, any of the previously computed solutions can be chosen.
6. Choose the next private variable to unset according to the order generated in step 2.
7. Reset the domain of the chosen private variable to its ground domain (i.e., the domain it had before the start of any search)
8. Repeat steps 4 and 5 until *k* percentage of private variables have been unset. Note that the percentage *k* depends on the complexity of the sub-problem and can be much different for different CSPs/domains. The appropriate value of *k* needs to be determined for different classes of CSPs in the problem domain with a range of experiments. Additionally, the percentage *k* can be different for different sub-problems depending on the size and complexity of individual sub-problems and the number of private variables that already get bound by constraint propagation alone in the sub-problem. Also, in order to be able to generate multiple non-trivially different solution clusters, it is necessary that *k* percentage of private variables is greater than the number of private variables that get bound by constraint propagation alone in the sub-problem.

9. Unset all public variables to their ground domain and search for a new solution. As the CPA generated in step 6 originated from a known solution, it is certain that it contains at least one solution and possibly several others. Thus, all public variables are then unset to their ground domains and start a search from the resulting CPA to arrive at a new solution *S'*.
10. In order to generate a solution cluster from *S'*, steps described in the previous algorithm are used.

The overall distributed search algorithm

Once the procedure for generating a solution cluster (and generating other *non-trivially* different solution clusters) is clear, a distributed search scheme can be constructed that exploits these solution clusters as messages passed among collaborating solvers. However, as previously mentioned, it is assumed that each of the sub-problems divided among individual solvers are easily solvable and contain a large number of solutions. For more assumptions, please refer back to the earlier section.

The central idea of the proposed distributed search algorithm is to use solution clusters for passing more meaningful messages among collaborating solvers. However, a collaborating solver can still fail to find a solution with the given solution cluster. Therefore, the cluster sending node then needs to send a new cluster that is non-trivially different than the previous cluster to avoid repeated failures. To effectively propagate the solution cluster messages among collaborating solvers, a

central coordinator is employed that orchestrates the negotiations among the collaborating solvers as illustrated pictorially in the diagram below.

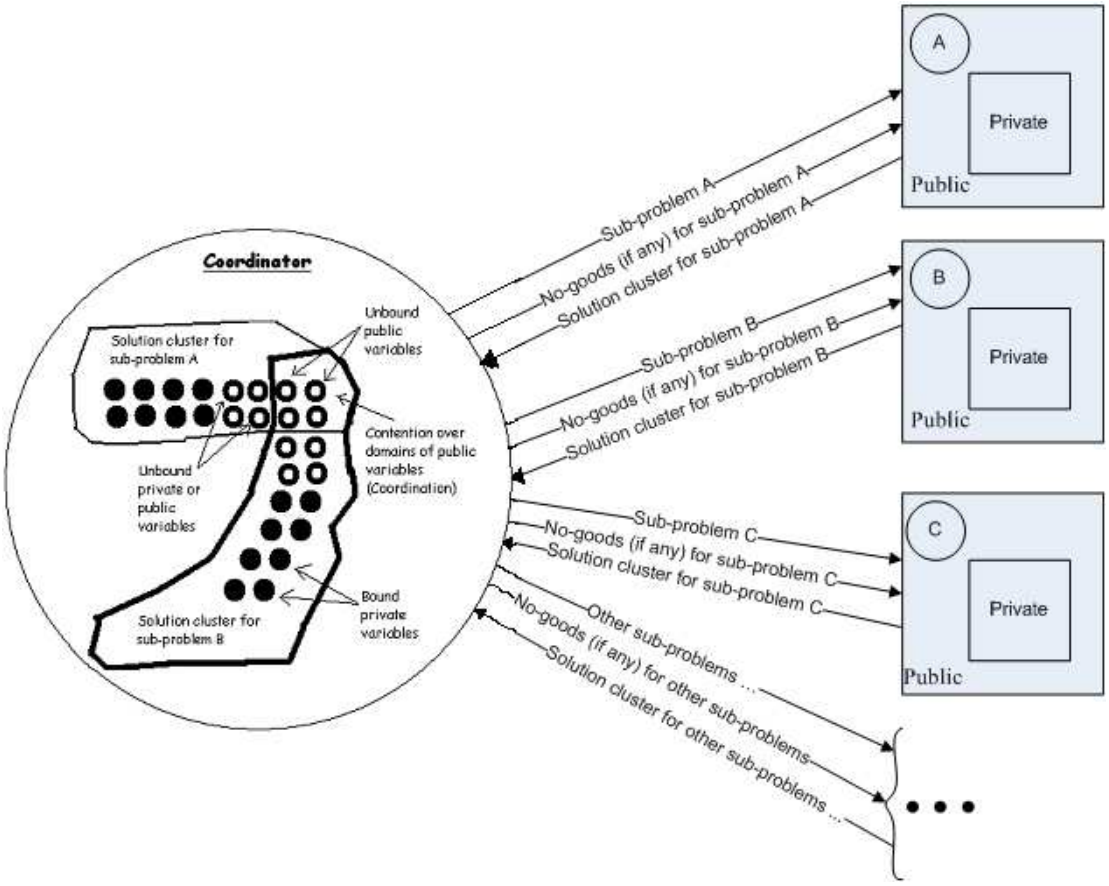

Figure 3: Distributed search

As the above diagram shows, the distributed-CSP is divided into several small sub-problems {*A, B, C, D, …*} to be solved by individual sub-problem solvers. The public and private variables of these solvers are also shown clearly. The way to divide a global CSP into smaller sub-problems is not clearly defined; however, this thesis does provide some guidelines later on useful ways of doing this with the general idea being to minimize the conflicts among the sub-problems to avoid a prolonged search for finding a globally consistent solution. As can be seen in the diagram above, in this algorithm, the individual sub-solvers do not negotiate directly with each other.

Rather, a central coordinator orchestrates the negotiation among individual sub-problem solvers and resolving conflicts that arise while solving the global CSP in a distributed manner. The arrows between a sub-problem solver and the coordinator denote the message traffic between them. Also, the diagram illustrates pictorially the coordination process that takes place inside the coordinator for resolving conflicts among the solution clusters sent by multiple sub-problems solvers. Below are the detailed steps of the overall distributed search procedure:

- Step 1: The coordinator divides the overall CSP into several small sub-problems such that the expected conflicts between the sub-problems are minimal. Guidelines on some useful ways for this division are given later. However, the general idea is to divide problem such that there are less number of public variables shared among the sub-problems. This decreases contradictions among solution clusters for the sub-problem and therefore reduces coordination time.
- Step 2: The coordinator then assigns sub-problems to be solved to collaborating solvers. Depending on the number of computing resources available as well as the specifications of those machines, multiple sub-problems can be solved on the same machine at the same time. If sufficient resources are available, all of the sub-problems can be sent out for solving. Otherwise, sub-problems are sent for solving as the resources become available. In this case, the order in which the coordinator assigns the sub-problems to the collaborating solver is determined based on simple heuristics like those employed by a normal constraint solver for ordering the variables in

the depth-first search. For example, a simple heuristic could be choosing that sub-problem first which contains a smaller number of public variables. Here, the intuition is that choosing a sub-problem with smaller number of public variables first corresponds to the first-fail principle of constraint programming. However, admittedly, this needs further research and experimentation to develop more specific and effective heuristics. The coordinator manages this process of assigning sub-problems to collaborating solvers.

- Step 3: The individual solvers start working on the sub-problems as soon as they receive them. They go through the steps described earlier to generate a solution cluster message for the coordinator without any knowledge about the global CSP. If a solving time limit is set for the sub-problem solver, it can be run in anytime mode to generate the best possible solution cluster in the allotted time. At an extreme, if the size of the sub-problem is small enough, and it does not take too long to generate all solutions of the sub-problem, the sub-problem solver can also be asked to package all of the sub-problem solutions into the solution cluster it generates. The sub-problem solvers also record both the solution used to generate the solution cluster and the solution cluster itself. This helps it to generate a non-trivially different solution cluster in the future, when needed.
- Step 4: As the coordinator receives solution clusters for sub-problems from the collaborating solvers, it keeps collecting and unifying them. The coordinator performs a backtracking depth-first search using each solution

cluster received as the nodes of the search tree. This is similar to depth-first search for a solution by a normal constraint solver where the variables of the CSP form the nodes of the search tree. The difference being that in the coordinator the solution clusters for the sub-problems are the nodes of the search tree. Also, no matter whether the coordinator used a sub-problem ordering or not while assigning the sub-problems to collaborating solvers (see step 2 above), it always forms a depth-first search like tree for the solution clusters it receives from sub-problem solvers. As it receives solution clusters, it unifies them into a larger solution cluster that is consistent with all of the sub-problem solution clusters it has received thus far. The simplest procedure of combining the solution clusters might involve simply taking the union of domains for all of the variables and running the constraint propagation for all of the constraints involved in the two sub-problems so as to avoid inconsistent assignments so far. It is possible in this step that the coordinator may not be able to resolve every conflict among the solution clusters it has received. In this situation, it is necessary for the coordinator to backtrack the solution cluster and send the sub-problem again to the collaborating solver for generating a non-trivially different solution cluster. The backtracking procedure is given in step 5 below.

- Step 5: As mentioned above, if the coordinator cannot resolve conflict among the sub-problem solution clusters (i.e., while unifying the domains of the public variables as given by the solution clusters, the domain of certain variables became empty), it is necessary for the coordinator to backtrack and

try a new solution cluster for one of the sub-problems it has considered so far. This is similar to backtracking in the depth-first search of a normal constraint solver where when the domain of a variable becomes empty, the search procedure needs to backtrack up in the search tree to try different value of the parent node. However, in this algorithm, the coordinator, orchestrating the distributed search, does not necessarily backtracks the solution cluster which was sent the latest to the coordinator. Instead, it employs a domain-specific search heuristic to determine the solution cluster to backtrack. One example of such a search heuristic would be to choose that solution cluster first to backtrack that contains the least number of public variables. Again as in step 2 above, this corresponds to the first-fail principle of constraint programming. When a solution-cluster is backtracked, the coordinator notes the combination of current assignments – to the public variables in the solution cluster being backtracked – as *no-good constraints* [21]. In order to prevent the collaborating solver from repeating the mistake, the coordinator updates the sub-problem with these no-good constraints before resending for generating a non-trivially different solution cluster. Additionally, when a sub-problem solver cannot send a non-trivially different solution cluster owing to it being a maximal solution cluster (as described earlier in the steps to generate a non-trivially different solution cluster), a different solution cluster must be chosen for being backtracked.

- Step 6: In this way, the coordinator keeps proceeding until all sub-problems have been solved and a globally consistent solution cluster for the global CSP

has been found. Next, the coordinator performs a local search for the global CSP to find a globally consistent solution for the overall problem. In case, the partial assignment in the solution cluster for the global CSP results in a failure (i.e., no satisfying assignment for all variables of the global CSP could be found), the coordinator repeats backtracking as given above in step 5.

- Step 7: The coordinator keeps repeating the above steps to perform the backtracking search (with solution clusters as nodes of the search tree) until either it is determined that none of the solutions of a sub-problem can lead to a globally consistent solution cluster for the global CSP or what is usually the case, a globally consistent solution for the global CSP is found. In case no globally consistent solution is found, the coordinator returns with a failure and it is determined that the global CSP is not satisfiable.

The main advantage of this approach comes with the fact that as the size of the problem increases, the search time to solve it as one big problem with any constraint satisfaction solver typically increases exponentially. However, when the problem is broken into smaller sub-problems, and the solutions of sub-problems are coordinated using the solution clusters approach, the solution time can be dramatically reduced. This is because not only do the solution clusters allow for effective (in little conflict with each-other and with the solution of the overall problem) negotiation among sub-problem solutions, but also express a large amount of information in a compact form thereby reducing the communication penalties during coordination. Furthermore, using a distributed setup of computing resources

and parallel searching in sub-problem solvers, the overall time for finding the solution of the problem can be significantly low. The tricky part is to come up with an algorithm that can enable this procedure in a systematic manner. This thesis provides one such algorithm. Also, as will be shown by means of case studies in the later section, the feasibility of this approach toward reducing the solution time is clearly proved.

Analysis of the distributed search procedure

Following the assumptions made earlier about the CSP, particularly that the overall CSP is hard to solve as a standalone problem, for the purpose of simplifying the analysis, it is further assumed that the runtime complexity of the CSP increases exponentially with the increase in the problem size. Let the original problem be divided into *N* equal sub-problems each of size *1/N* times the size of the original CSP. Also, let T_{sub} be the maximal time required for solving a sub-problem, then with the assumption of exponential increase in runtime with problem size, the time required for the original problem standalone can be approximated to $(T_{sub})^N$.

Now, Amdahl's law [27] for speed-up with parallel processing, states that the speed-up attained by parallelizing part of the program computations is given by:

$$SpeedUp = \frac{1}{(1-P) + \frac{P}{S}}$$

where,

P = fraction of the program computation that can be parallelized, and

S = the number of parallel processors used

This comes from the fact that *(1 – P)* part of computation still runs on a single processor taking *(1 – P)* times the time to perform the entire computation on the single processor, and *P* part of the computation can be performed in parallel on *S* processors. Following the same reasoning, and noting that the sub-problem solvers themselves run on a single computing resource, the speed-up attained by dividing the problem into *N* sub-problems, XT_N, is given by:

$$XT_N = \frac{(T_{sub})^N}{N * T_{sub}} = \frac{(T_{sub})^{N-1}}{N}$$

However, above analysis assumes that no contradiction among the solution clusters of sub-problems occurs and as such no time is spent in coordinating and backtracking in the coordinator. In order to consider the time spent during coordination of the distributed search, the runtime of depth-first search (like in case of a normal constraint solver) must be determined first. The time taken by a depth-first search is given by $O\left(b^d\right)$, where *b* is the maximal branching factor of the search tree and *d* is the depth of the tree. Above, assumes that maximal path length of the search tree is not greater than *d* – which is true in case of distributed search where the depth of the distributed search tree equals the number of sub-problems (viz., *N*). Notice that if the global CSP is solved as a standalone problem, the depth *d* can be exceedingly deep and correspondingly the runtime can be prohibitively huge. For the distributed search algorithm, the maximal branching factor, in the worst-case, corresponds to the number of non-trivially different solution clusters that a sub-problem solver can generate. This not only depends on the factor *k* which is percentage of private variables unset in a sub-problem, but also on the number of assignments possible for

the chosen private variables. The number of possible assignments to private variables in turn depends on the maximal branching factor of the normal constraint solver while choosing possible values for the chosen private variables from their domains. Thus, the branching factor b_{dist} of the distributed search, in the worst-case, is given by:

$$\forall \text{ i: } 0 \leq \text{i} < \text{N}$$

$$b_{dist} = \max\left[\left({}^{NumPRVS_i}C_{k_i}\right) * (bMax_i)^{(1-k_i/100)}\right], \textit{ where,}$$

b_{dist} = maximal branching factor of distributed search

N = Number of sub-problems,

$NumPRVs_i$ = Number of private variables in the i^{th} sub-problem,

$bMax_i$ = maximal branching factor of regular constraint-based search in the i^{th} sub-problem, and

k_i = Percentage of private variables to unset in i^{th} sub-problem

However, as there are *N* sub-problems, which corresponds to the depth of the coordinated distributed search, the total time spent in coordination, in the worst-case, is given by $O\left(b_{dist}{}^{N}\right)$, where b_{dist} is the maximal branching factor as given by the above equation. Furthermore, at every coordination step during the distributed search, the major time-consuming part, ignoring the constant time minor calculations, deals with searching a solution cluster for a sub-problem. Therefore, in the worst-case, the run-time complexity, T_{dist}, of the distributed search procedure is given by:

$$T_{dist} = O\left(b_{dist}{}^{N} * T_{sub}\right)$$

However, one must realize that this is simply a worst-case scenario. In practice, for the average case, the algorithm performs far superior than the above time-bounds. Also, exploiting parallel computing resources, the distributed search procedure can significantly reduce the runtime to find the solution of the overall CSP – which was simply too time-consuming to be solved as a standalone problem.

Guidelines for efficient division of a distributed-CSP

Now, after devising the procedure to solve a distributed-CSP using solution cluster based approach, it is important to take a step back to develop strategies for efficiently dividing the original distributed-CSP among individual sub-problems. The goal is obviously to minimize the possibility of future conflicts among the sub-problem solvers during the distributed search. One simple approach is to minimize the number of public variables shared among the sub-problems. Whereas other complex domain-specific approaches could also look at maximizing the utilization of the computing resources available for the coordinator and determining the optimal break-up by means of experiments. However, often this division is solely directed by the security concerns to divide the problem around security barriers and/or organizational boundaries. Though, in general, it is advisable, and may turn out to be highly useful for distributed search efficiency, that the division of the original CSP is not carried out based on any trivial logic or randomly. Rather, a systematic approach of characterizing the given CSP in terms of the connectivity among its variables through its constraints and determining the location of clusters in an equivalent constraint graph must be used. However, note that characterizing a CSP in terms of its

runtime complexity was already described earlier as an extremely hard problem. This is an area of research in itself and can require considerable amount of experiments to come up with a set strategy that works in a particular application domain.

CHAPTER V

EXPERIMENTAL RESULTS

This chapter provides a sample proof-of-concept example code in Oz language [28] that demonstrates the feasibility of using a solution clusters based approach for solving distributed-CSPs. The chapter continues by taking a graph coloring instance in the famous DIMACS [11] standard ASCII format, divided into two parts, forming a solution cluster using these two sub-problems, and showing that the time taken to solve the problem using the solution cluster information is smaller than the time it takes to solve the overall problem standalone. In both the cases, same search strategy and constraints are used with the only difference being that in case of solving with solution clusters based approach, additional information of the solution cluster is used. Below, first a well-known SEND+MORE=MONEY example from the constraint programming domain is presented, then the example of a standard DIMACS graph coloring challenge problem is considered, and finally experimental results are provided for several programmatically generated graphs.

SEND+MORE=MONEY example

Before delving into applying the solution clusters approach to graph coloring instances, this section first presents a small example to illustrate the concept. The famous SEND+MORE=MONEY example from the constraint programming domain is chosen. Here instead of solving the problem as a single problem, it is divided into

two small parts *p1* and *p2*. The problem is divided in such a way that the two sub-problems are not independent of each other as illustrated below:

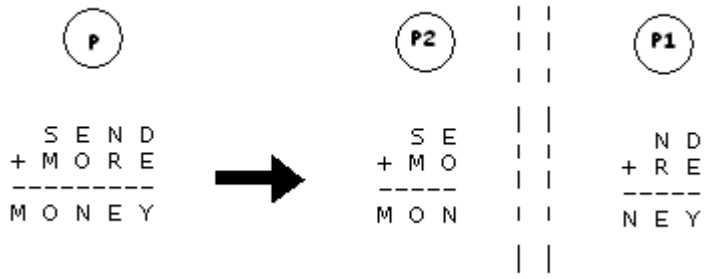

Figure 4: Dividing SEND+MORE=MONEY problem *P* into two sub-problems *P1* and *P2*

In above example the variables shared between sub-problems *p1* and *p2* are *N* and *E*. Thus, the values chosen while solving any of the sub-problems may be in conflict for a solution of the other sub-problem. However, both sub-problems *p1* and *p2* involved smaller number of variables and constraints than the complete problem. In general, it is simpler to solve a smaller part of the problem than the overall problem as a whole. For illustrating the sub-problem solving and merging useful information among parts of the problem, a simple Oz program is provided (see Appendix F). In this program, the sub-problems *p1* and *p2* are solved separately for all solutions and then *p2* is solved again using the information generated in the solution cluster for sub-problem *p1*.

The figure below shows the output generated while running the program. In this figure, it can be seen that a total of 6 solutions were found for sub-problem *p2* and 290 solutions were found for sub-problem *p1* without any communication from each other. However, in this example, when the information generated from the

solutions of sub-problem *p1* was utilized in solving sub-problem *p2* again, the number of solutions still consistent with this information reduced from 6 to 1. This clearly illustrates that dividing the problem into smaller parts is not only feasible, but may result in speed-up in solving the overall problem.

```
''
'CREATING GLOBALS'
global(ene:_{0#10} nne:_{0#10})
''
''
'CALLING P1 IN A SEPARATE THREAD'
''
'CALLING P2 IN A SEPARATE THREAD'
'P2: Inside P2... adding constraints'
'P1: Inside P1... adding constraints'
'P1: Before distribution, current values = '#sol(d:_{0#9} e:_{0#9} n:_{0#9} r:_{0#9} y:_{0#9})
'P2: Returned from P2'
[sol(e:2 m:1 n:3 o:0 s:9) sol(e:3 m:1 n:4 o:0 s:9) sol(e:4 m:1 n:5 o:0 s:9) sol(e:5 m:1 n:6 o:0 s:9)
 sol(e:6 m:1 n:7 o:0 s:9) sol(e:7 m:1 n:8 o:0 s:9)]
'P2: Number of solutions = '#6
''
'P1: Returned from P1'
sol(d:1 e:2 n:4 r:8 y:3)|sol(d:1 e:2 n:5 r:7 y:3)|sol(d:1 e:2 n:7 r:5 y:3)|sol(d:1 e:2 n:8 r:4 y:3)|
sol(d:1 e:3 n:5 r:8 y:4)|sol(d:1 e:3 n:6 r:7 y:4)|sol(d:1 e:3 n:7 r:6 y:4)|sol(d:1 e:3 n:8 r:5 y:4)|
sol(d:1 e:4 n:6 r:8 y:5)|sol(d:1 e:4 n:8 r:6 y:5)|,,,|,,,
'P1: Number of solutions = '#290
''
'NOW REDUCING DOMAINS OF GLOBAL VARIABLES'
'Printing globals again'
global(ene:_{2#7} nne:_{3#8})
''
'NOW SOLVING P2 AGAIN WITH CROSS DOMAIN INFO'
'P2: Inside P2... adding constraints'
'Returned from P2'
[sol(e:2 m:1 n:3 o:0 s:9)]
```

Figure 5: Output of solving SEND+MORE=MONEY problem in parts

DIMACS graph coloring challenge problem: a case study

For demonstrating the feasibility of solution clusters based approach, this thesis takes a graph coloring example *queen5_5.col* in the DIMACS standard ASCII format provided on Michael Trick's graph coloring instances webpage[4]. However, to increase the number of solutions of the problem, first 100 edges of the graph were removed. The resulting graph has a total of 360 solutions. When solved as a standalone problem the time taken was 172 milliseconds on a PC with Windows XP Pro SP2, an Intel® Pentium® 4 Processor 630 with HT Technology (3GHz, 800FSB),

[4] http://mat.gsia.cmu.edu/COLOR/instances/queen5_5.col

and 2GB RAM. On the other hand, when solved using the information generated using solution clusters approach on the same PC, the time taken got reduced to 156 milliseconds resulting in a savings of 16 milliseconds. This savings may look smaller at first; however, it clearly shows both the applicability and feasibility of using solution clusters approach. The complete procedure to use the automatically generated Oz program from a given graph coloring problem in DIMACS standard ASCII format is described below. This Oz program has the code necessary to solve the problem as standalone or using the solution clusters approach. When compiled into an executable, the .EXE can be run with an option '-c' to use the solution clusters approach. In the solution clusters approach, the two sub-problems are invoked on separate Oz threads[5] and once both the threads return with solution clusters, they are combined and then the overall problem is solved using this information. Note that it is trivial to change the program so that these sub-problems can run in a distributed[6] manner on multiple machines and in parallel [22, 23, 24]. For example, if a problem *p* is divided into two sub-problems, s*p1* and s*p2*, to be solved in parallel, then the worst case execution time, *wcet*, for the overall problem is given by $wcet\{sp1 \parallel sp2\} = \max\{wcet(sp1), wcet(sp2)\}$, which can be substantially less than $wcet(p)$. Additionally, the overall problem can potentially be divided into more than two sub-problems to increase the speed depending upon the computing hardware resources available (e.g., in a grid fashion) [22, 23, 24]. Although this thesis clearly demonstrates the feasibility and effectiveness of using the solution clusters approach,

[5] http://www.mozart-oz.org/documentation/tutorial/node8.html#chapter.concurrency
[6] http://www.mozart-oz.org/documentation/dstutorial/index.html

more work needs to be done to develop a *framework* for solving problems using solution clusters approach as described in the future work section.

The C++ code that takes a graph coloring problem in the DIMACS standard ASCII format as input, and generates an Oz program is provided in Appendix A. The DIMACS format graph coloring problem used is given in Appendix B. The generated Oz program using above C++ code for the above graph coloring problem is given in Appendix C. The output of running the program as a standalone problem is given in Appendix D, and with the solution clusters based approach respectively is given in Appendix E.

Experimental runs on several generated graphs

Several graphs were generated in the DIMACS standard ASCII format using the very_nauty[7] graph theory software [29]. Appendix G provides a sample C source code listing that was used to generate these graphs in the DIMACS format using the above library. The complexity of these graphs was gradually increased by increasing the number of nodes (and correspondingly the number of edges) in the graph. All of these generated graphs are listed in Appendix H. Next, the same Oz code generating software was used to generate Oz programs that include code to solve the problem both as standalone and with the solution clusters approach. For the sake of brevity, the generated Oz programs for all of these graphs are not listed. However, the same can be easily generated using out Oz code generation C++ program given in Appendix A.

[7] http://keithbriggs.info/very_nauty.html

The table below shows the speed-ups attained when running the generated Oz code for the graphs (generated using the very_nauty graph library [29]) given in Appendix H:

Table 1: Experimental runs on several generated graphs

Graph ID (as given in Appendix H)	**#Nodes/ #Edges**	**Runtime when solved as a standalone problem T_{prev} (milliseconds)**	**Runtime when solved using solution clusters based approach T_{new} (milliseconds)**	**Time saved ($T_{saved} = T_{prev} - T_{new}$) (milliseconds)**
Graph-A	6/15	79	78	1
Graph-B	7/21	830	828	2
Graph-C	8/28	9031	8984	47
Graph-D	9/36	108672	105954	2718
Graph-E	10/45	1253578	1248892	4686

These experiments were performed on a PC with Windows XP Pro SP2, an Intel® Pentium® 4 Processor 630 with HT Technology (3GHz, 800FSB), and 2GB RAM. Also, as described earlier in the challenge problem experiment, the generated Oz program divides the CSP into two overlapping sub-problems and then generates the solution cluster for one of the sub-problems and uses that information in solving the second sub-problem. CSP problems are known to show exponential increase in solver runtime with a linear increase in the problem size. This is also seen in the experimental results above. Particularly, on running the solver on the above generated graphs, it can be seen that increasing the number of nodes and edges to 10 and 45 respectively in Graph-E from 6 and 15 respectively in Graph-A causes the solver's

runtime to increase from 79 milliseconds to 1253578 milliseconds. However, even though the original CSP was divided into only two sub-problems, the runtime savings with the use of solution clusters approach also increased exponentially. This is particularly important considering the facts that neither a multi-core processor nor multiple computers were used and that the Oz program generated did not make use of distributed constraint programming in Oz [30] – which supports any number of communicating processes executing in parallel. Note that although separate Oz threads[8] were used for running each of the two sub-problems, they are simply dataflow style concurrency abstractions in the Oz language and they do not take full advantage of multi-core processors. In practice, however, significantly huge runtime savings can be realized by dividing the large CSP into several smaller sub-problems (as detailed in the algorithms presented in this thesis) and by taking full advantage of multi-core processors and even multiple computers.

[8] http://www.mozart-oz.org/documentation/tutorial/node8.html#chapter.concurrency

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

In this thesis, a solution clusters based approach was presented to speed-up solving of large combinatorial problems. After developing the overall solution clusters based approach, this thesis presented algorithms to apply this approach to solve hard Constraint Satisfaction Problems (CSPs) by dividing them into simpler sub-problems and solving each of them in a distributed manner, provided Oz program of constraint-based solver for several examples, and demonstrated both the applicability and feasibility of this approach to significantly reduce solving time. The main contributions of this work are listed below:

1. Background research for application of solution clusters based approach for solving hard Constraint Satisfaction Problems (CSPs). This involved:
    a. Surveying the state-of-the-art constraint programming techniques used in solving CSPs.
    b. Surveying the solution clusters based approach as developed for solving 3-SAT problems.
    c. Identifying challenges in applying this technique to the domain of Constraint Satisfaction Problems.
    d. Determining equivalent set of entities in the CSP domain for being able to apply the solution clusters based approach for solving hard

CSPs in a distributed and efficient manner (by casting an equivalent *distributed-CSP* for the original problem).

2. Development of a complete theory for applying a solution clusters based approach to solve hard Constraint Satisfaction Problems. This involved developing algorithms for each major step of the theory as below:
    a. Algorithm for generating a "solution rich" Cluster defining Partial Assignment (CPA): Here, the apparent difficulties in applying this to Constraint Satisfaction Problems were highlighted and an approach that works in this domain was developed.
    b. Algorithm for generating non-trivially different solution clusters: This is a significantly hard problem in general, and in the CSP domain in particular, for which this thesis developed an approach that can be effectively used for generating non-trivially different solution clusters when backtracking occurs while coordinating solution clusters from various sub-problem solvers.
    c. The thesis presented the detailed distributed search and coordination algorithm that is used for solving the hard CSPs into smaller sub-problems and solved as *distributed-CSP* using a central coordinator orchestrating the distributed search.
    d. Developed general guidelines for breaking the original CSP into smaller sub-problems in a manner that is efficient for applying the solution clusters approach to solve them.

3. A detailed runtime complexity analysis was given for the solution clusters based distributed search algorithm proposed in thesis.
4. This thesis also presented full C++ source code to read a graph coloring problem in the DIMACS standard ASCII format and generate a corresponding Oz program that is capable of solving the problem both as standalone and with the solution clusters approach. The experiments conducted in the thesis used the above generated Oz code to solve several graph coloring problems. The experimental results showed both the feasibility and efficiency of using the solution clusters approach in solving hard CSPs.

In the course of the thesis, some progress was also made toward writing a generic framework for constraint-based problem solving using the solution clusters approach. However, developing the complete framework was out of the scope of this thesis and remains as part of the future work. However, it is envisioned that, given the proven feasibility of this approach and the detailed theory developed in this thesis, it is not difficult to come up with an extensive framework to solve significantly hard Constraint Satisfaction Problems using this approach. Moreover, as described in the overall distributed search algorithm, the framework could also exploit multiple computing machines for solving sub-problems in parallel to significantly reduce the overall runtime.

APPENDIX A

C++ CODE TO GENERATE THE OZ PROGRAM FOR A GRAPH COLORING PROBLEM GIVEN IN DIMACS STANDARD ASCII FORMAT

File: genbin.h

```
#ifndef GENBIN
#define GENBIN

#include <stdio.h>
#include <iostream>

/* If you change MAX_NR_VERTICES, change MAX_NR_VERTICESdiv8 to be
the 1/8th of it */

#define MAX_NR_VERTICES       5000
#define MAX_NR_VERTICESdiv8   625
#define MAX_NR_EDGES 15000

#define BOOL      char

class EDGE {
public:
      int from;
      int to;
};

int Nr_vert, Nr_edges;
EDGE allEdges[MAX_NR_EDGES];
BOOL Bitmap[MAX_NR_VERTICES][MAX_NR_VERTICESdiv8];

char masks[ 8 ] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80
};

#define MAX_PREAMBLE 10000
static char Preamble[MAX_PREAMBLE];

int MAX_COLORS_REQD = 1;

#endif
```

File: asc2oz.cpp

```
/* Converts a DIMACS graph coloring problem to Oz input functor */

#include "genbin.h"

int get_params();
/* ======================================================= */
```

```
void read_graph_DIMACS_ascii(char* file)
{

      int c, oc;
      char * pp = Preamble;
      int i,j;
      FILE *fp;
      int edgesRead = 0;

      if ( (fp=fopen(file,"r"))==NULL ) {
            printf("ERROR: Cannot open infile\n");
            exit(10);
      }

      for(oc = '\0' ;(c = fgetc(fp)) != EOF && (oc != '\n' || c != 
'e')
            ; oc = *pp++ = c);

      ungetc(c, fp);
      *pp = '\0';
      get_params();

//    allEdges = new EDGE[Nr_edges];

      // printf("allEdges size = %d\n", Nr_edges);

      while ((c = fgetc(fp)) != EOF){
            switch (c)
              {
                  case 'e':
                    if (!fscanf(fp, "%d %d", &i, &j))
                        { printf("ERROR: corrupted inputfile\n");
exit(10); }

                    if (i > j)  {
                          printf("\nAdding edge no. %d",
(edgesRead+1));
                        allEdges[edgesRead].from = i-1;
                        allEdges[edgesRead++].to = j-1;
                    } else {
                          printf("\nAdding edge no. %d",
(edgesRead+1));
                        allEdges[edgesRead].from = j-1;
                        allEdges[edgesRead++].to = i-1;
                    }

                    break;

                  case '\n':

                  default:
                    break;
              }
      }

      fclose(fp);
```

```
}

int get_params()
                        /* getting Nr_vert and Nr_edge from the
preamble string,
                                        containing Dimacs format "p ???
num num" */
{
      char c, *tmp;
      char * pp = Preamble;
      int stop = 0;
      tmp = (char *)calloc(100, sizeof(char));

      Nr_vert = Nr_edges = 0;

      while (!stop && (c = *pp++) != '\0'){
            switch (c)
              {
                  case 'c':
                    while ((c = *pp++) != '\n' && c != '\0');
                    break;

                  case 'p':
                    sscanf(pp, "%s %d %d\n", tmp, &Nr_vert,
&Nr_edges);
                    stop = 1;
                    break;

                  default:
                    break;
              }
      }

      free(tmp);

      if (Nr_vert == 0 || Nr_edges == 0)
        return 0;  /* error */
      else {
            printf("No. of vertices = %d\n", Nr_vert);
            printf("No. of edges = %d\n", Nr_edges);
        return 1;
      }

}

void write_graph_DIMACS_bin(char* file)
{

      int i;
      FILE *fp;

      if ( (fp=fopen(file,"w"))==NULL )
        { printf("ERROR: Cannot open outfile\n"); exit(10); }

      i=0;
      printf("\nNow printing %d edges", Nr_edges);
      while(i<Nr_edges) {
```

```
            printf("\nEdge %d from: %d to: %d", i, allEdges[i].from,
allEdges[i].to);
            i++;
      }
      i=0;

      /* Functor Header */
      fprintf(fp, "functor\n");
      fprintf(fp, "import\n");
      fprintf(fp, "   Search\n");
      fprintf(fp, "   FD\n");
      fprintf(fp, "   System\n");
      fprintf(fp, "   Application\n");
      fprintf(fp, "   Space\n");
      fprintf(fp, "   Browser\n");
      fprintf(fp, "   Explorer\n");
      fprintf(fp, "   Property\n");
      fprintf(fp, "prepare\n");
      fprintf(fp, "   ArgSpec = record(useclusters(single
char:&c))\n");
      fprintf(fp, "define\n");

      /* Declare misc. variables */
      fprintf(fp, "\n%% Misc. variables\n");
      fprintf(fp, "Args\n");
      fprintf(fp, "P1Sols\n");
      fprintf(fp, "P2Sols\n");
      fprintf(fp, "ThreadCheck1\n");
      fprintf(fp, "ThreadCheck2\n");
      fprintf(fp, "Solutions\n");
      fprintf(fp, "Depth = {NewCell 0}\n");

      /* Write generic functions (not dependent on P1 and P2) */
      /* Function to remove unsatisfying values (in domain of a var)
from domain of a variable */
      fprintf(fp, "\n%% Removes all values from $Var that are NOT in
domain of $NewValueVar\n");
      fprintf(fp, "proc {RomoveValuesNotInNewValueVar NewValueVar
Var}\n");
      fprintf(fp, "   ListCurDom ListNewDom ListFilteredDom\n");
      fprintf(fp, "in\n");
      fprintf(fp, "   ListCurDom = {FD.reflect.domList Var}\n");
      fprintf(fp, "   ListNewDom = {FD.reflect.domList
NewValueVar}\n");
      fprintf(fp, "   ListFilteredDom = {List.filter ListCurDom fun
{$ Val} {List.member Val ListNewDom} end}\n");
      fprintf(fp, "   {List.forAll ListCurDom proc {$ X} if
{List.member X ListFilteredDom} == false then Var \\=: X end
end}\n");
      fprintf(fp, "end\n");

      /* Function to remove unsatisfying values (in a list) from
domain of a variable */
      fprintf(fp, "\n%% Removes all values from $Var that are NOT in
value list $NewValueList\n");
      fprintf(fp, "proc {RemoveValuesNotInNewList NewValueList
Var}\n");
```

```
      fprintf(fp, "   ListCurDom ListFilteredDom\n");
      fprintf(fp, "in\n");
      fprintf(fp, "   ListCurDom = {FD.reflect.domList Var}\n");
      fprintf(fp, "   ListFilteredDom = {List.filter ListCurDom fun
{$ Val} {List.member Val NewValueList} end}\n");
      fprintf(fp, "   {List.forAll ListCurDom proc {$ X} if
{List.member X ListFilteredDom} == false then Var \\=: X end
end}\n");
      fprintf(fp, "end\n");

      /* Function to increment value of cell $Depth by 1 */
      fprintf(fp, "\n%% Increments value of cell $Depth by 1\n");
      fprintf(fp, "proc {IncreaseDepth}\n");
      fprintf(fp, "   CurVal NewVal\n");
      fprintf(fp, "in\n");
      fprintf(fp, "   {Exchange Depth CurVal NewVal}\n");
      fprintf(fp, "   NewVal = CurVal + 1\n");
      fprintf(fp, "end\n");

      /* Function to decrement value of cell $Depth by 1 */
      fprintf(fp, "\n%% Decrements value of cell $Depth by 1\n");
      fprintf(fp, "proc {DecreaseDepth}\n");
      fprintf(fp, "   CurVal NewVal\n");
      fprintf(fp, "in\n");
      fprintf(fp, "   {Exchange Depth CurVal NewVal}\n");
      fprintf(fp, "   NewVal = CurVal - 1\n");
      fprintf(fp, "end\n");

      /* Depth First Search (DFS) sub-routine */
      fprintf(fp, "\n%% Depth First Search (DFS) sub-routine\n");
      fprintf(fp, "fun {DFE S}\n");
      fprintf(fp, "   case {Space.ask S} of\n");
      fprintf(fp, "   failed then\n");
      fprintf(fp, "      nil\n");
      fprintf(fp, "   [] succeeded then\n");
      fprintf(fp, "      [S]\n");
      fprintf(fp, "   [] alternatives(2) then\n");
      fprintf(fp, "      {IncreaseDepth}\n");
      fprintf(fp, "      C = {Space.clone S} in\n");
      fprintf(fp, "      {Space.commit S 1}\n");
      fprintf(fp, "      case {DFE S} of\n");
      fprintf(fp, "      nil then\n");
      fprintf(fp, "         {Space.commit C 2} {DFE C}\n");
      fprintf(fp, "      [] [T] then\n");
      fprintf(fp, "         [T]\n");
      fprintf(fp, "      end\n");
      fprintf(fp, "   end\n");
      fprintf(fp, "end\n");

      /* Main Depth First Search (DFS) routine */
      fprintf(fp, "\n%% Main Depth First Search (DFS) routine\n");
      fprintf(fp, "%% Given {Script Sol}, returns solution [Sol] or
nil\n");
      fprintf(fp, "fun {DFS Script}\n");
      fprintf(fp, "   case {DFE {Space.new Script}} of\n");
      fprintf(fp, "   nil then\n");
      fprintf(fp, "      nil\n");
```

```
      fprintf(fp, "   [] [S] then\n");
      fprintf(fp, "      [{Space.merge S}]\n");
      fprintf(fp, "   end\n");
      fprintf(fp, "end\n");

      /* Function to return X no. of solutions */
      fprintf(fp, "\n%% Computes a given no. of solutions (if
any)\n");
      fprintf(fp, "fun {GetXSols SearchObj NumMoreSols
PrevSolList}\n");
      fprintf(fp, "   if NumMoreSols =< 0 then\n");
      fprintf(fp, "      PrevSolList\n");
      fprintf(fp, "   else\n");
      fprintf(fp, "      NewSol\n");
      fprintf(fp, "   in\n");
      fprintf(fp, "      NewSol = {SearchObj next($)}\n");
      fprintf(fp, "      if NewSol \\= nil then\n");
      fprintf(fp, "         {GetXSols SearchObj NumMoreSols-1
NewSol|PrevSolList}\n");
      fprintf(fp, "      else\n");
      fprintf(fp, "         PrevSolList\n");
      fprintf(fp, "      end\n");
      fprintf(fp, "   end\n");
      fprintf(fp, "end\n");

      /* Removes duplicates from a given list */
      fprintf(fp, "\n%% Removes duplicate elements from the
list\n");
      fprintf(fp, "fun {RemoveDups AList}\n");
      fprintf(fp, "   fun {RemoveDupsInternal Xs}\n");
      fprintf(fp, "      case Xs\n");
      fprintf(fp, "      of nil then\n");
      fprintf(fp, "         nil\n");
      fprintf(fp, "      [] H|T then\n");
      fprintf(fp, "         {List.foldL T\n");
      fprintf(fp, "            fun {$ Acc X}\n");
      fprintf(fp, "               if Acc.1==X then\n");
      fprintf(fp, "                  Acc\n");
      fprintf(fp, "               else\n");
      fprintf(fp, "                  X|Acc\n");
      fprintf(fp, "               end\n");
      fprintf(fp, "            end\n");
      fprintf(fp, "         [H]}\n");
      fprintf(fp, "      end\n");
      fprintf(fp, "   end\n");
      fprintf(fp, "in\n");
      fprintf(fp, "   {RemoveDupsInternal {List.sort AList
Value.'>'}}\n");
      fprintf(fp, "end\n");

      /* Write procedure for first half of problem */
      fprintf(fp, "\n%% First half of the problem");
      int numHalf = (Nr_vert * 7)/10;
      fprintf(fp, "\nproc {P1 S1}");
      i=0;
      while(i<numHalf) {
            fprintf(fp, "\n   ");
```

```
            for(int count=0; count<10 && i<numHalf; count++,i++) {
                  fprintf(fp, "Y%d ", i);
            }
      }
      fprintf(fp, "\nin");
      fprintf(fp, "\n\n   S1 = sol(");
      i=0;
      while(i<numHalf) {
            fprintf(fp, "\n      ");
            for(int count=0; count<5 && i<numHalf; count++,i++) {
                  fprintf(fp, "y%d:Y%d ", i, i);
            }
      }
      fprintf(fp, "\n   )");
      fprintf(fp, "\n\n   S1 ::: 0#%d", MAX_COLORS_REQD-1);
      fprintf(fp, "\n\n   %% ApplyConstraints\n");
      for(i=0; i<Nr_edges; i++) {
            if(allEdges[i].from < numHalf && allEdges[i].to <
numHalf) {
                  fprintf(fp, "   {FD.distinct [Y%d Y%d]}\n",
allEdges[i].from, allEdges[i].to);
            }
      }
      fprintf(fp, "\n\n   %% Distribute over the half variables");
      fprintf(fp, "\n   {FD.distribute min S1}");
      fprintf(fp, "\nend\n");

      /* Write procedure for second half of problem */
      fprintf(fp, "\n%% Second half of the problem");
      fprintf(fp, "\nproc {P2 S2}");
      i=(Nr_vert * 3)/10;
      int startingIdx = i;
      while(i<Nr_vert) {
            fprintf(fp, "\n   ");
            for(int count=0; count<10 && i<Nr_vert; count++,i++) {
                  fprintf(fp, "Z%d ", i);
            }
      }
      fprintf(fp, "\nin");
      fprintf(fp, "\n\n   S2 = sol(");
      i=startingIdx;
      while(i<Nr_vert) {
            fprintf(fp, "\n      ");
            for(int count=0; count<5 && i<Nr_vert; count++,i++) {
                  fprintf(fp, "z%d:Z%d ", i, i);
            }
      }
      fprintf(fp, "\n   )");
      fprintf(fp, "\n\n   S2 ::: 0#%d", MAX_COLORS_REQD-1);
      fprintf(fp, "\n\n   %% ApplyConstraints\n");
      for(i=0; i<Nr_edges; i++) {
            if(allEdges[i].from >= startingIdx && allEdges[i].to >=
startingIdx) {
                  fprintf(fp, "   {FD.distinct [Z%d Z%d]}\n",
allEdges[i].from, allEdges[i].to);
            }
      }
```

```
      fprintf(fp, "\n\n   %% Distribute over the half variables");
      fprintf(fp, "\n   {FD.distribute min S2}");
      fprintf(fp, "\nend\n");

      /* Function to call procedure P1 */
      fprintf(fp, "\n%% Procedure to call P1 in a separate
thread\n");
      fprintf(fp, "proc {CallP1InASeparateThread}\n");
      fprintf(fp, "   MySearchObj\n");
      fprintf(fp, "in\n");
      fprintf(fp, "   thread\n");
      fprintf(fp, "      {System.show ''}\n");
      fprintf(fp, "      {System.show 'CALLING P1 IN A SEPARATE
THREAD'}\n");
      fprintf(fp, "      MySearchObj = {New Search.object script(P1
rcd:30)}\n");
      // fprintf(fp, "      P1Sols = {GetXSols MySearchObj 100
nil}\n");
      fprintf(fp, "      P1Sols = {Search.all P1 30 _}\n");
      fprintf(fp, "      {System.show 'P1: Returned from P1'}\n");
      fprintf(fp, "      {System.show P1Sols}\n");
      fprintf(fp, "      {System.show 'P1: Number of solutions =
'#{List.length P1Sols}}\n");
      fprintf(fp, "      {System.show ''}\n");
      fprintf(fp, "      ThreadCheck1 = unit\n");
      fprintf(fp, "   end\n");
      fprintf(fp, "end\n");

      /* Function to call procedure P2 */
      fprintf(fp, "\n%% Procedure to call P2 in a separate
thread\n");
      fprintf(fp, "proc {CallP2InASeparateThread}\n");
      fprintf(fp, "   MySearchObj\n");
      fprintf(fp, "in\n");
      fprintf(fp, "   thread\n");
      fprintf(fp, "      {System.show ''}\n");
      fprintf(fp, "      {System.show 'CALLING P2 IN A SEPARATE
THREAD'}\n");
      fprintf(fp, "      MySearchObj = {New Search.object script(P2
rcd:30)}\n");
      // fprintf(fp, "      P2Sols = {GetXSols MySearchObj 100
nil}\n");
      fprintf(fp, "      P2Sols = {Search.all P2 30 _}\n");
      fprintf(fp, "      {System.show 'P2: Returned from P2'}\n");
      fprintf(fp, "      {System.show P2Sols}\n");
      fprintf(fp, "      {System.show 'P2: Number of solutions =
'#{List.length P2Sols}}\n");
      fprintf(fp, "      {System.show ''}\n");
      fprintf(fp, "      ThreadCheck2 = unit\n");
      fprintf(fp, "   end\n");
      fprintf(fp, "end\n");

      /* The key procedure to reduce global domains based on
solution clusters */
      fprintf(fp, "\n%% Procedure to reduce global domains based on
solution clusters\n");
      fprintf(fp, "proc {ReduceGlobalDomains Vars}\n");
```

```
	i=0;
	while(i<numHalf) {
		fprintf(fp, "   X%dValidValsP1\n", i);
		i++;
	}
	fprintf(fp, "\n");
	i=startingIdx;
	while(i<Nr_vert) {
		fprintf(fp, "   X%dValidValsP2\n", i);
		i++;
	}
	fprintf(fp, "in\n");
	i=0;
	while(i<numHalf) {
		// fprintf(fp, "   X%dValidValsP1 = {RemoveDups
{List.map P1Sols fun{$ AP1Sol} {List.nth AP1Sol 1}.y%d end}}\n", i,
i);
		fprintf(fp, "   X%dValidValsP1 = {RemoveDups {List.map
P1Sols fun{$ AP1Sol} AP1Sol.y%d end}}\n", i, i);
		i++;
	}
	fprintf(fp, "\n");
	i=startingIdx;
	while(i<Nr_vert) {
		// fprintf(fp, "   X%dValidValsP2 = {RemoveDups
{List.map P2Sols fun{$ AP2Sol} {List.nth AP2Sol 1}.z%d end}}\n", i,
i);
		fprintf(fp, "   X%dValidValsP2 = {RemoveDups {List.map
P2Sols fun{$ AP2Sol} AP2Sol.z%d end}}\n", i, i);
		i++;
	}
	fprintf(fp, "\n");
	i=0;
	while(i<numHalf) {
		fprintf(fp, "   {RemoveValuesNotInNewList X%dValidValsP1
Vars.x%d}\n", i, i);
		i++;
	}
	fprintf(fp, "\n");
	i=startingIdx;
	while(i<Nr_vert) {
		fprintf(fp, "   {RemoveValuesNotInNewList X%dValidValsP2
Vars.x%d}\n", i, i);
		i++;
	}
	fprintf(fp, "end\n");

	/* Function to solve problem without clustering */
	fprintf(fp, "\n%% Solves the problem without using
clustering\n");
	fprintf(fp, "proc {PFullWithoutClustering S}\n");
	for(i=0; i<Nr_vert; i++) {
		fprintf(fp, "   X%d = {FD.int 0#%d}\n", i,
MAX_COLORS_REQD-1);
	}
	fprintf(fp, "in\n");
	fprintf(fp, "   S = allVars(\n");
```

```
	i=0;
	while(i<Nr_vert) {
		fprintf(fp, "	");
		for(int count=0; count<5 && i<Nr_vert; count++,i++) {
			fprintf(fp, "x%d:X%d ", i, i);
		}
		fprintf(fp, "\n");
	}
	fprintf(fp, "   )\n\n");
	fprintf(fp, "   %% Applying constraints\n");
	for(i=0; i<Nr_edges; i++) {
		fprintf(fp, "   {FD.distinct [X%d X%d]}\n",
allEdges[i].from, allEdges[i].to);
	}
	fprintf(fp, "\n   {System.show 'Before search: time = '}\n");
	fprintf(fp, "   {System.show {Property.get
'time.total'}}\n\n");
	fprintf(fp, "   {FD.distribute min S}\n");
	fprintf(fp, "end\n");

	/* Function to solve problem using solution clusters */
	fprintf(fp, "\n%% Solves the problem using solution
clusters\n");
	fprintf(fp, "proc {PFullUsingSolutionClusters S}\n");
	for(i=0; i<Nr_vert; i++) {
		fprintf(fp, "   X%d = {FD.int 0#%d}\n", i,
MAX_COLORS_REQD-1);
	}
	fprintf(fp, "in\n");
	fprintf(fp, "   S = allVars(\n");
	i=0;
	while(i<Nr_vert) {
		fprintf(fp, "	");
		for(int count=0; count<5 && i<Nr_vert; count++,i++) {
			fprintf(fp, "x%d:X%d ", i, i);
		}
		fprintf(fp, "\n");
	}
	fprintf(fp, "   )\n\n");
	fprintf(fp, "   %% Applying constraints\n");
	for(i=0; i<Nr_edges; i++) {
		fprintf(fp, "   {FD.distinct [X%d X%d]}\n",
allEdges[i].from, allEdges[i].to);
	}
	fprintf(fp, "   {System.show 'Now reducing domains in
PFullUsingSolutionClusters'}\n");
	fprintf(fp, "   {ReduceGlobalDomains S}\n\n");
	fprintf(fp, "   {System.show 'Before search: time = '}\n");
	fprintf(fp, "   {System.show {Property.get
'time.total'}}\n\n");
	fprintf(fp, "   {FD.distribute min S}\n");
	fprintf(fp, "end\n");

	/* Generate the main body of the functor */
	fprintf(fp, "\nin\n\n");
	fprintf(fp, "   Args = {Application.getCmdArgs ArgSpec}\n\n");
```

```
      fprintf(fp, "   if {Value.hasFeature Args useclusters}
then\n");
      fprintf(fp, "      {CallP1InASeparateThread}\n");
      fprintf(fp, "      {CallP2InASeparateThread}\n");
      fprintf(fp, "      {Wait ThreadCheck1}\n");
      fprintf(fp, "      {Wait ThreadCheck2}\n");
      fprintf(fp, "      Solutions = {Search.all
PFullUsingSolutionClusters 30 _}\n");
      fprintf(fp, "   else\n");
      fprintf(fp, "      Solutions = {Search.all
PFullWithoutClustering 30 _}\n");
      fprintf(fp, "   end\n\n");
      fprintf(fp, "   {System.show 'After search: time = '}\n");
      fprintf(fp, "   {System.show {Property.get
'time.total'}}\n\n");
      fprintf(fp, "   {System.showInfo \"\\nFound \"#{Length
Solutions}#\" solution(s)\"}\n");
      fprintf(fp, "   {ForAll Solutions\n");
      fprintf(fp, "      proc {$ ASolution}\n");
      fprintf(fp, "         {System.showInfo \"\\nNext
solution\"}\n");
      fprintf(fp, "         {System.show ASolution}\n");
      fprintf(fp, "      end\n");
      fprintf(fp, "   }\n");
      fprintf(fp, "   {System.showInfo \"\\nNo. of solutions printed
above = \"#{Length Solutions}#\" solution(s)\"}\n");
      fprintf(fp, "   {Delay 1000}\n");
      fprintf(fp, "   {Application.exit 0}\n");

      fprintf(fp, "\nend\n");
      fclose(fp);
}
void main(int argc,
     char **argv)
{
      int i;
      char name[255];
      if ( argc > 4 || argc < 3)
        {printf("Usage: %s infile [outfile]
[MAX_COLORS_REQUIRED]\n",argv[0]);  exit(10); }

      MAX_COLORS_REQD = atoi(argv[3]);

      read_graph_DIMACS_ascii( argv[1] );
      printf("After reading graph file, numEdges = %d", Nr_edges);

      if (argc == 2)
        sprintf(name, "%s.b", argv[1]);
      else
        sprintf(name, "%s", argv[2]);

      write_graph_DIMACS_bin( name );
}
```

APPENDIX B

THE GRAPH COLORING INSTANCE USED QUEEN5_5.COL9 (MINUS FIRST 100 EDGES)

```
c FILE: queen5_5.col
c Translated from Stanford GraphBase File: queen5_5.sgb
c Stanford GraphBase ID: gunion(board(5,5,0,0,-1,0,0),board(5,5,0,0,-2,0,0),0,0)
p edge 25 220
e 9 15
e 9 13
e 9 17
e 9 21
e 9 10
e 9 14
e 9 19
e 9 24
e 9 8
e 9 7
e 9 6
e 9 5
e 9 4
e 9 3
e 10 14
e 10 18
e 10 22
e 10 15
e 10 20
e 10 25
e 10 9
e 10 8
e 10 7
e 10 6
e 10 5
e 10 4
e 11 17
e 11 23
e 11 12
e 11 13
e 11 14
e 11 15
e 11 16
e 11 21
```

[9] http://mat.gsia.cmu.edu/COLOR/instances/queen5_5.col

e 11 7
e 11 6
e 11 3
e 11 1
e 12 18
e 12 24
e 12 16
e 12 13
e 12 14
e 12 15
e 12 17
e 12 22
e 12 11
e 12 8
e 12 7
e 12 6
e 12 4
e 12 2
e 13 19
e 13 25
e 13 17
e 13 21
e 13 14
e 13 15
e 13 18
e 13 23
e 13 12
e 13 11
e 13 9
e 13 8
e 13 7
e 13 5
e 13 3
e 13 1
e 14 20
e 14 18
e 14 22
e 14 15
e 14 19
e 14 24
e 14 13
e 14 12
e 14 11
e 14 10
e 14 9
e 14 8

e 14 4
e 14 2
e 15 19
e 15 23
e 15 20
e 15 25
e 15 14
e 15 13
e 15 12
e 15 11
e 15 10
e 15 9
e 15 5
e 15 3
e 16 22
e 16 17
e 16 18
e 16 19
e 16 20
e 16 21
e 16 12
e 16 11
e 16 8
e 16 6
e 16 4
e 16 1
e 17 23
e 17 21
e 17 18
e 17 19
e 17 20
e 17 22
e 17 16
e 17 13
e 17 12
e 17 11
e 17 9
e 17 7
e 17 5
e 17 2
e 18 24
e 18 22
e 18 19
e 18 20
e 18 23
e 18 17

e 18 16
e 18 14
e 18 13
e 18 12
e 18 10
e 18 8
e 18 6
e 18 3
e 19 25
e 19 23
e 19 20
e 19 24
e 19 18
e 19 17
e 19 16
e 19 15
e 19 14
e 19 13
e 19 9
e 19 7
e 19 4
e 19 1
e 20 24
e 20 25
e 20 19
e 20 18
e 20 17
e 20 16
e 20 15
e 20 14
e 20 10
e 20 8
e 20 5
e 20 2
e 21 22
e 21 23
e 21 24
e 21 25
e 21 17
e 21 16
e 21 13
e 21 11
e 21 9
e 21 6
e 21 5
e 21 1

e 22 23
e 22 24
e 22 25
e 22 21
e 22 18
e 22 17
e 22 16
e 22 14
e 22 12
e 22 10
e 22 7
e 22 2
e 23 24
e 23 25
e 23 22
e 23 21
e 23 19
e 23 18
e 23 17
e 23 15
e 23 13
e 23 11
e 23 8
e 23 3
e 24 25
e 24 23
e 24 22
e 24 21
e 24 20
e 24 19
e 24 18
e 24 14
e 24 12
e 24 9
e 24 6
e 24 4
e 25 24
e 25 23
e 25 22
e 25 21
e 25 20
e 25 19
e 25 15
e 25 13
e 25 10
e 25 7

e 25 5
e 25 1

APPENDIX C

GENERATED OZ PROGRAM

```
functor
import
   Search
   FD
   System
   Application
   Space
   Browser
   Explorer
   Property
prepare
   ArgSpec = record(useclusters(single char:&c))
define

% Misc. variables
Args
P1Sols
P2Sols
ThreadCheck1
ThreadCheck2
Solutions
Depth = {NewCell 0}

% Removes all values from $Var that are NOT in domain of
$NewValueVar
proc {RomoveValuesNotInNewValueVar NewValueVar Var}
   ListCurDom ListNewDom ListFilteredDom
in
   ListCurDom = {FD.reflect.domList Var}
   ListNewDom = {FD.reflect.domList NewValueVar}
   ListFilteredDom = {List.filter ListCurDom fun {$ Val}
{List.member Val ListNewDom} end}
   {List.forAll ListCurDom proc {$ X} if {List.member X
ListFilteredDom} == false then Var \=: X end end}
end

% Removes all values from $Var that are NOT in value list
$NewValueList
proc {RemoveValuesNotInNewList NewValueList Var}
   ListCurDom ListFilteredDom
in
   ListCurDom = {FD.reflect.domList Var}
   ListFilteredDom = {List.filter ListCurDom fun {$ Val}
{List.member Val NewValueList} end}
   {List.forAll ListCurDom proc {$ X} if {List.member X
ListFilteredDom} == false then Var \=: X end end}
end
```

```
% Increments value of cell $Depth by 1
proc {IncreaseDepth}
   CurVal NewVal
in
   {Exchange Depth CurVal NewVal}
   NewVal = CurVal + 1
end

% Decrements value of cell $Depth by 1
proc {DecreaseDepth}
   CurVal NewVal
in
   {Exchange Depth CurVal NewVal}
   NewVal = CurVal - 1
end

% Depth First Search (DFS) sub-routine
fun {DFE S}
   case {Space.ask S} of
   failed then
      nil
   [] succeeded then
      [S]
   [] alternatives(2) then
      {IncreaseDepth}
      C = {Space.clone S} in
      {Space.commit S 1}
      case {DFE S} of
      nil then
         {Space.commit C 2} {DFE C}
      [] [T] then
         [T]
      end
   end
end

% Main Depth First Search (DFS) routine
% Given {Script Sol}, returns solution [Sol] or nil
fun {DFS Script}
   case {DFE {Space.new Script}} of
   nil then
      nil
   [] [S] then
      [{Space.merge S}]
   end
end

% Computes a given no. of solutions (if any)
fun {GetXSols SearchObj NumMoreSols PrevSolList}
   if NumMoreSols =< 0 then
      PrevSolList
   else
      NewSol
   in
      NewSol = {SearchObj next($)}
      if NewSol \= nil then
```

```
         {GetXSols SearchObj NumMoreSols-1 NewSol|PrevSolList}
      else
         PrevSolList
      end
   end
end

% Removes duplicate elements from the list
fun {RemoveDups AList}
   fun {RemoveDupsInternal Xs}
      case Xs
      of nil then
         nil
      [] H|T then
         {List.foldL T
            fun {$ Acc X}
               if Acc.1==X then
                  Acc
               else
                  X|Acc
               end
            end
         [H]}
      end
   end
in
   {RemoveDupsInternal {List.sort AList Value.'>'}}
end

% First half of the problem
proc {P1 S1}
   Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8 Y9
   Y10 Y11 Y12 Y13 Y14 Y15 Y16
in

   S1 = sol(
      y0:Y0 y1:Y1 y2:Y2 y3:Y3 y4:Y4
      y5:Y5 y6:Y6 y7:Y7 y8:Y8 y9:Y9
      y10:Y10 y11:Y11 y12:Y12 y13:Y13 y14:Y14
      y15:Y15 y16:Y16
   )

   S1 ::: 0#4

   % ApplyConstraints
   {FD.distinct [Y14 Y8]}
   {FD.distinct [Y12 Y8]}
   {FD.distinct [Y16 Y8]}
   {FD.distinct [Y9 Y8]}
   {FD.distinct [Y13 Y8]}
   {FD.distinct [Y8 Y7]}
   {FD.distinct [Y8 Y6]}
   {FD.distinct [Y8 Y5]}
   {FD.distinct [Y8 Y4]}
   {FD.distinct [Y8 Y3]}
   {FD.distinct [Y8 Y2]}
   {FD.distinct [Y13 Y9]}
```

```
{FD.distinct [Y14 Y9]}
{FD.distinct [Y9 Y8]}
{FD.distinct [Y9 Y7]}
{FD.distinct [Y9 Y6]}
{FD.distinct [Y9 Y5]}
{FD.distinct [Y9 Y4]}
{FD.distinct [Y9 Y3]}
{FD.distinct [Y16 Y10]}
{FD.distinct [Y11 Y10]}
{FD.distinct [Y12 Y10]}
{FD.distinct [Y13 Y10]}
{FD.distinct [Y14 Y10]}
{FD.distinct [Y15 Y10]}
{FD.distinct [Y10 Y6]}
{FD.distinct [Y10 Y5]}
{FD.distinct [Y10 Y2]}
{FD.distinct [Y10 Y0]}
{FD.distinct [Y15 Y11]}
{FD.distinct [Y12 Y11]}
{FD.distinct [Y13 Y11]}
{FD.distinct [Y14 Y11]}
{FD.distinct [Y16 Y11]}
{FD.distinct [Y11 Y10]}
{FD.distinct [Y11 Y7]}
{FD.distinct [Y11 Y6]}
{FD.distinct [Y11 Y5]}
{FD.distinct [Y11 Y3]}
{FD.distinct [Y11 Y1]}
{FD.distinct [Y16 Y12]}
{FD.distinct [Y13 Y12]}
{FD.distinct [Y14 Y12]}
{FD.distinct [Y12 Y11]}
{FD.distinct [Y12 Y10]}
{FD.distinct [Y12 Y8]}
{FD.distinct [Y12 Y7]}
{FD.distinct [Y12 Y6]}
{FD.distinct [Y12 Y4]}
{FD.distinct [Y12 Y2]}
{FD.distinct [Y12 Y0]}
{FD.distinct [Y14 Y13]}
{FD.distinct [Y13 Y12]}
{FD.distinct [Y13 Y11]}
{FD.distinct [Y13 Y10]}
{FD.distinct [Y13 Y9]}
{FD.distinct [Y13 Y8]}
{FD.distinct [Y13 Y7]}
{FD.distinct [Y13 Y3]}
{FD.distinct [Y13 Y1]}
{FD.distinct [Y14 Y13]}
{FD.distinct [Y14 Y12]}
{FD.distinct [Y14 Y11]}
{FD.distinct [Y14 Y10]}
{FD.distinct [Y14 Y9]}
{FD.distinct [Y14 Y8]}
{FD.distinct [Y14 Y4]}
{FD.distinct [Y14 Y2]}
{FD.distinct [Y16 Y15]}
```

```
   {FD.distinct [Y15 Y11]}
   {FD.distinct [Y15 Y10]}
   {FD.distinct [Y15 Y7]}
   {FD.distinct [Y15 Y5]}
   {FD.distinct [Y15 Y3]}
   {FD.distinct [Y15 Y0]}
   {FD.distinct [Y16 Y15]}
   {FD.distinct [Y16 Y12]}
   {FD.distinct [Y16 Y11]}
   {FD.distinct [Y16 Y10]}
   {FD.distinct [Y16 Y8]}
   {FD.distinct [Y16 Y6]}
   {FD.distinct [Y16 Y4]}
   {FD.distinct [Y16 Y1]}


   % Distribute over the half variables
   {FD.distribute min S1}
end

% Second half of the problem
proc {P2 S2}
   Z7 Z8 Z9 Z10 Z11 Z12 Z13 Z14 Z15 Z16
   Z17 Z18 Z19 Z20 Z21 Z22 Z23 Z24
in

   S2 = sol(
      z7:Z7 z8:Z8 z9:Z9 z10:Z10 z11:Z11
      z12:Z12 z13:Z13 z14:Z14 z15:Z15 z16:Z16
      z17:Z17 z18:Z18 z19:Z19 z20:Z20 z21:Z21
      z22:Z22 z23:Z23 z24:Z24
   )

   S2 ::: 0#4

   % ApplyConstraints
   {FD.distinct [Z14 Z8]}
   {FD.distinct [Z12 Z8]}
   {FD.distinct [Z16 Z8]}
   {FD.distinct [Z20 Z8]}
   {FD.distinct [Z9 Z8]}
   {FD.distinct [Z13 Z8]}
   {FD.distinct [Z18 Z8]}
   {FD.distinct [Z23 Z8]}
   {FD.distinct [Z8 Z7]}
   {FD.distinct [Z13 Z9]}
   {FD.distinct [Z17 Z9]}
   {FD.distinct [Z21 Z9]}
   {FD.distinct [Z14 Z9]}
   {FD.distinct [Z19 Z9]}
   {FD.distinct [Z24 Z9]}
   {FD.distinct [Z9 Z8]}
   {FD.distinct [Z9 Z7]}
   {FD.distinct [Z16 Z10]}
   {FD.distinct [Z22 Z10]}
   {FD.distinct [Z11 Z10]}
   {FD.distinct [Z12 Z10]}
```

```
{FD.distinct [Z13 Z10]}
{FD.distinct [Z14 Z10]}
{FD.distinct [Z15 Z10]}
{FD.distinct [Z20 Z10]}
{FD.distinct [Z17 Z11]}
{FD.distinct [Z23 Z11]}
{FD.distinct [Z15 Z11]}
{FD.distinct [Z12 Z11]}
{FD.distinct [Z13 Z11]}
{FD.distinct [Z14 Z11]}
{FD.distinct [Z16 Z11]}
{FD.distinct [Z21 Z11]}
{FD.distinct [Z11 Z10]}
{FD.distinct [Z11 Z7]}
{FD.distinct [Z18 Z12]}
{FD.distinct [Z24 Z12]}
{FD.distinct [Z16 Z12]}
{FD.distinct [Z20 Z12]}
{FD.distinct [Z13 Z12]}
{FD.distinct [Z14 Z12]}
{FD.distinct [Z17 Z12]}
{FD.distinct [Z22 Z12]}
{FD.distinct [Z12 Z11]}
{FD.distinct [Z12 Z10]}
{FD.distinct [Z12 Z8]}
{FD.distinct [Z12 Z7]}
{FD.distinct [Z19 Z13]}
{FD.distinct [Z17 Z13]}
{FD.distinct [Z21 Z13]}
{FD.distinct [Z14 Z13]}
{FD.distinct [Z18 Z13]}
{FD.distinct [Z23 Z13]}
{FD.distinct [Z13 Z12]}
{FD.distinct [Z13 Z11]}
{FD.distinct [Z13 Z10]}
{FD.distinct [Z13 Z9]}
{FD.distinct [Z13 Z8]}
{FD.distinct [Z13 Z7]}
{FD.distinct [Z18 Z14]}
{FD.distinct [Z22 Z14]}
{FD.distinct [Z19 Z14]}
{FD.distinct [Z24 Z14]}
{FD.distinct [Z14 Z13]}
{FD.distinct [Z14 Z12]}
{FD.distinct [Z14 Z11]}
{FD.distinct [Z14 Z10]}
{FD.distinct [Z14 Z9]}
{FD.distinct [Z14 Z8]}
{FD.distinct [Z21 Z15]}
{FD.distinct [Z16 Z15]}
{FD.distinct [Z17 Z15]}
{FD.distinct [Z18 Z15]}
{FD.distinct [Z19 Z15]}
{FD.distinct [Z20 Z15]}
{FD.distinct [Z15 Z11]}
{FD.distinct [Z15 Z10]}
{FD.distinct [Z15 Z7]}
```

```
{FD.distinct [Z22 Z16]}
{FD.distinct [Z20 Z16]}
{FD.distinct [Z17 Z16]}
{FD.distinct [Z18 Z16]}
{FD.distinct [Z19 Z16]}
{FD.distinct [Z21 Z16]}
{FD.distinct [Z16 Z15]}
{FD.distinct [Z16 Z12]}
{FD.distinct [Z16 Z11]}
{FD.distinct [Z16 Z10]}
{FD.distinct [Z16 Z8]}
{FD.distinct [Z23 Z17]}
{FD.distinct [Z21 Z17]}
{FD.distinct [Z18 Z17]}
{FD.distinct [Z19 Z17]}
{FD.distinct [Z22 Z17]}
{FD.distinct [Z17 Z16]}
{FD.distinct [Z17 Z15]}
{FD.distinct [Z17 Z13]}
{FD.distinct [Z17 Z12]}
{FD.distinct [Z17 Z11]}
{FD.distinct [Z17 Z9]}
{FD.distinct [Z17 Z7]}
{FD.distinct [Z24 Z18]}
{FD.distinct [Z22 Z18]}
{FD.distinct [Z19 Z18]}
{FD.distinct [Z23 Z18]}
{FD.distinct [Z18 Z17]}
{FD.distinct [Z18 Z16]}
{FD.distinct [Z18 Z15]}
{FD.distinct [Z18 Z14]}
{FD.distinct [Z18 Z13]}
{FD.distinct [Z18 Z12]}
{FD.distinct [Z18 Z8]}
{FD.distinct [Z23 Z19]}
{FD.distinct [Z24 Z19]}
{FD.distinct [Z19 Z18]}
{FD.distinct [Z19 Z17]}
{FD.distinct [Z19 Z16]}
{FD.distinct [Z19 Z15]}
{FD.distinct [Z19 Z14]}
{FD.distinct [Z19 Z13]}
{FD.distinct [Z19 Z9]}
{FD.distinct [Z19 Z7]}
{FD.distinct [Z21 Z20]}
{FD.distinct [Z22 Z20]}
{FD.distinct [Z23 Z20]}
{FD.distinct [Z24 Z20]}
{FD.distinct [Z20 Z16]}
{FD.distinct [Z20 Z15]}
{FD.distinct [Z20 Z12]}
{FD.distinct [Z20 Z10]}
{FD.distinct [Z20 Z8]}
{FD.distinct [Z22 Z21]}
{FD.distinct [Z23 Z21]}
{FD.distinct [Z24 Z21]}
{FD.distinct [Z21 Z20]}
```

```
   {FD.distinct [Z21 Z17]}
   {FD.distinct [Z21 Z16]}
   {FD.distinct [Z21 Z15]}
   {FD.distinct [Z21 Z13]}
   {FD.distinct [Z21 Z11]}
   {FD.distinct [Z21 Z9]}
   {FD.distinct [Z23 Z22]}
   {FD.distinct [Z24 Z22]}
   {FD.distinct [Z22 Z21]}
   {FD.distinct [Z22 Z20]}
   {FD.distinct [Z22 Z18]}
   {FD.distinct [Z22 Z17]}
   {FD.distinct [Z22 Z16]}
   {FD.distinct [Z22 Z14]}
   {FD.distinct [Z22 Z12]}
   {FD.distinct [Z22 Z10]}
   {FD.distinct [Z22 Z7]}
   {FD.distinct [Z24 Z23]}
   {FD.distinct [Z23 Z22]}
   {FD.distinct [Z23 Z21]}
   {FD.distinct [Z23 Z20]}
   {FD.distinct [Z23 Z19]}
   {FD.distinct [Z23 Z18]}
   {FD.distinct [Z23 Z17]}
   {FD.distinct [Z23 Z13]}
   {FD.distinct [Z23 Z11]}
   {FD.distinct [Z23 Z8]}
   {FD.distinct [Z24 Z23]}
   {FD.distinct [Z24 Z22]}
   {FD.distinct [Z24 Z21]}
   {FD.distinct [Z24 Z20]}
   {FD.distinct [Z24 Z19]}
   {FD.distinct [Z24 Z18]}
   {FD.distinct [Z24 Z14]}
   {FD.distinct [Z24 Z12]}
   {FD.distinct [Z24 Z9]}


   % Distribute over the half variables
   {FD.distribute min S2}
end

% Procedure to call P1 in a separate thread
proc {CallP1InASeparateThread}
   MySearchObj
in
   thread
      {System.show ''}
      {System.show 'CALLING P1 IN A SEPARATE THREAD'}
      MySearchObj = {New Search.object script(P1 rcd:30)}
      P1Sols = {Search.all P1 30 _}
      {System.show 'P1: Returned from P1'}
      {System.show P1Sols}
      {System.show 'P1: Number of solutions = '#{List.length
P1Sols}}
      {System.show ''}
      ThreadCheck1 = unit
```

```
   end
end

% Procedure to call P2 in a separate thread
proc {CallP2InASeparateThread}
   MySearchObj
in
   thread
      {System.show ''}
      {System.show 'CALLING P2 IN A SEPARATE THREAD'}
      MySearchObj = {New Search.object script(P2 rcd:30)}
      P2Sols = {Search.all P2 30 _}
      {System.show 'P2: Returned from P2'}
      {System.show P2Sols}
      {System.show 'P2: Number of solutions = '#{List.length
P2Sols}}
      {System.show ''}
      ThreadCheck2 = unit
   end
end

% Procedure to reduce global domains based on solution clusters
proc {ReduceGlobalDomains Vars}
   X0ValidValsP1
   X1ValidValsP1
   X2ValidValsP1
   X3ValidValsP1
   X4ValidValsP1
   X5ValidValsP1
   X6ValidValsP1
   X7ValidValsP1
   X8ValidValsP1
   X9ValidValsP1
   X10ValidValsP1
   X11ValidValsP1
   X12ValidValsP1
   X13ValidValsP1
   X14ValidValsP1
   X15ValidValsP1
   X16ValidValsP1

   X7ValidValsP2
   X8ValidValsP2
   X9ValidValsP2
   X10ValidValsP2
   X11ValidValsP2
   X12ValidValsP2
   X13ValidValsP2
   X14ValidValsP2
   X15ValidValsP2
   X16ValidValsP2
   X17ValidValsP2
   X18ValidValsP2
   X19ValidValsP2
   X20ValidValsP2
   X21ValidValsP2
   X22ValidValsP2
```

```
   X23ValidValsP2
   X24ValidValsP2
in
   X0ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y0 end}}
   X1ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y1 end}}
   X2ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y2 end}}
   X3ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y3 end}}
   X4ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y4 end}}
   X5ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y5 end}}
   X6ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y6 end}}
   X7ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y7 end}}
   X8ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y8 end}}
   X9ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y9 end}}
   X10ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y10 end}}
   X11ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y11 end}}
   X12ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y12 end}}
   X13ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y13 end}}
   X14ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y14 end}}
   X15ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y15 end}}
   X16ValidValsP1 = {RemoveDups {List.map P1Sols fun{$ AP1Sol}
AP1Sol.y16 end}}

   X7ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z7 end}}
   X8ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z8 end}}
   X9ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z9 end}}
   X10ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z10 end}}
   X11ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z11 end}}
   X12ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z12 end}}
   X13ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z13 end}}
   X14ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z14 end}}
   X15ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z15 end}}
```

```
   X16ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z16 end}}
   X17ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z17 end}}
   X18ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z18 end}}
   X19ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z19 end}}
   X20ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z20 end}}
   X21ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z21 end}}
   X22ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z22 end}}
   X23ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z23 end}}
   X24ValidValsP2 = {RemoveDups {List.map P2Sols fun{$ AP2Sol}
AP2Sol.z24 end}}

   {RemoveValuesNotInNewList X0ValidValsP1 Vars.x0}
   {RemoveValuesNotInNewList X1ValidValsP1 Vars.x1}
   {RemoveValuesNotInNewList X2ValidValsP1 Vars.x2}
   {RemoveValuesNotInNewList X3ValidValsP1 Vars.x3}
   {RemoveValuesNotInNewList X4ValidValsP1 Vars.x4}
   {RemoveValuesNotInNewList X5ValidValsP1 Vars.x5}
   {RemoveValuesNotInNewList X6ValidValsP1 Vars.x6}
   {RemoveValuesNotInNewList X7ValidValsP1 Vars.x7}
   {RemoveValuesNotInNewList X8ValidValsP1 Vars.x8}
   {RemoveValuesNotInNewList X9ValidValsP1 Vars.x9}
   {RemoveValuesNotInNewList X10ValidValsP1 Vars.x10}
   {RemoveValuesNotInNewList X11ValidValsP1 Vars.x11}
   {RemoveValuesNotInNewList X12ValidValsP1 Vars.x12}
   {RemoveValuesNotInNewList X13ValidValsP1 Vars.x13}
   {RemoveValuesNotInNewList X14ValidValsP1 Vars.x14}
   {RemoveValuesNotInNewList X15ValidValsP1 Vars.x15}
   {RemoveValuesNotInNewList X16ValidValsP1 Vars.x16}

   {RemoveValuesNotInNewList X7ValidValsP2 Vars.x7}
   {RemoveValuesNotInNewList X8ValidValsP2 Vars.x8}
   {RemoveValuesNotInNewList X9ValidValsP2 Vars.x9}
   {RemoveValuesNotInNewList X10ValidValsP2 Vars.x10}
   {RemoveValuesNotInNewList X11ValidValsP2 Vars.x11}
   {RemoveValuesNotInNewList X12ValidValsP2 Vars.x12}
   {RemoveValuesNotInNewList X13ValidValsP2 Vars.x13}
   {RemoveValuesNotInNewList X14ValidValsP2 Vars.x14}
   {RemoveValuesNotInNewList X15ValidValsP2 Vars.x15}
   {RemoveValuesNotInNewList X16ValidValsP2 Vars.x16}
   {RemoveValuesNotInNewList X17ValidValsP2 Vars.x17}
   {RemoveValuesNotInNewList X18ValidValsP2 Vars.x18}
   {RemoveValuesNotInNewList X19ValidValsP2 Vars.x19}
   {RemoveValuesNotInNewList X20ValidValsP2 Vars.x20}
   {RemoveValuesNotInNewList X21ValidValsP2 Vars.x21}
   {RemoveValuesNotInNewList X22ValidValsP2 Vars.x22}
   {RemoveValuesNotInNewList X23ValidValsP2 Vars.x23}
   {RemoveValuesNotInNewList X24ValidValsP2 Vars.x24}
end
```

```
% Solves the problem without using clustering
proc {PFullWithoutClustering S}
   X0 = {FD.int 0#4}
   X1 = {FD.int 0#4}
   X2 = {FD.int 0#4}
   X3 = {FD.int 0#4}
   X4 = {FD.int 0#4}
   X5 = {FD.int 0#4}
   X6 = {FD.int 0#4}
   X7 = {FD.int 0#4}
   X8 = {FD.int 0#4}
   X9 = {FD.int 0#4}
   X10 = {FD.int 0#4}
   X11 = {FD.int 0#4}
   X12 = {FD.int 0#4}
   X13 = {FD.int 0#4}
   X14 = {FD.int 0#4}
   X15 = {FD.int 0#4}
   X16 = {FD.int 0#4}
   X17 = {FD.int 0#4}
   X18 = {FD.int 0#4}
   X19 = {FD.int 0#4}
   X20 = {FD.int 0#4}
   X21 = {FD.int 0#4}
   X22 = {FD.int 0#4}
   X23 = {FD.int 0#4}
   X24 = {FD.int 0#4}
in
   S = allVars(
      x0:X0 x1:X1 x2:X2 x3:X3 x4:X4
      x5:X5 x6:X6 x7:X7 x8:X8 x9:X9
      x10:X10 x11:X11 x12:X12 x13:X13 x14:X14
      x15:X15 x16:X16 x17:X17 x18:X18 x19:X19
      x20:X20 x21:X21 x22:X22 x23:X23 x24:X24
   )

   % Applying constraints
   {FD.distinct [X14 X8]}
   {FD.distinct [X12 X8]}
   {FD.distinct [X16 X8]}
   {FD.distinct [X20 X8]}
   {FD.distinct [X9 X8]}
   {FD.distinct [X13 X8]}
   {FD.distinct [X18 X8]}
   {FD.distinct [X23 X8]}
   {FD.distinct [X8 X7]}
   {FD.distinct [X8 X6]}
   {FD.distinct [X8 X5]}
   {FD.distinct [X8 X4]}
   {FD.distinct [X8 X3]}
   {FD.distinct [X8 X2]}
   {FD.distinct [X13 X9]}
   {FD.distinct [X17 X9]}
   {FD.distinct [X21 X9]}
   {FD.distinct [X14 X9]}
   {FD.distinct [X19 X9]}
   {FD.distinct [X24 X9]}
```

```
{FD.distinct [X9 X8]}
{FD.distinct [X9 X7]}
{FD.distinct [X9 X6]}
{FD.distinct [X9 X5]}
{FD.distinct [X9 X4]}
{FD.distinct [X9 X3]}
{FD.distinct [X16 X10]}
{FD.distinct [X22 X10]}
{FD.distinct [X11 X10]}
{FD.distinct [X12 X10]}
{FD.distinct [X13 X10]}
{FD.distinct [X14 X10]}
{FD.distinct [X15 X10]}
{FD.distinct [X20 X10]}
{FD.distinct [X10 X6]}
{FD.distinct [X10 X5]}
{FD.distinct [X10 X2]}
{FD.distinct [X10 X0]}
{FD.distinct [X17 X11]}
{FD.distinct [X23 X11]}
{FD.distinct [X15 X11]}
{FD.distinct [X12 X11]}
{FD.distinct [X13 X11]}
{FD.distinct [X14 X11]}
{FD.distinct [X16 X11]}
{FD.distinct [X21 X11]}
{FD.distinct [X11 X10]}
{FD.distinct [X11 X7]}
{FD.distinct [X11 X6]}
{FD.distinct [X11 X5]}
{FD.distinct [X11 X3]}
{FD.distinct [X11 X1]}
{FD.distinct [X18 X12]}
{FD.distinct [X24 X12]}
{FD.distinct [X16 X12]}
{FD.distinct [X20 X12]}
{FD.distinct [X13 X12]}
{FD.distinct [X14 X12]}
{FD.distinct [X17 X12]}
{FD.distinct [X22 X12]}
{FD.distinct [X12 X11]}
{FD.distinct [X12 X10]}
{FD.distinct [X12 X8]}
{FD.distinct [X12 X7]}
{FD.distinct [X12 X6]}
{FD.distinct [X12 X4]}
{FD.distinct [X12 X2]}
{FD.distinct [X12 X0]}
{FD.distinct [X19 X13]}
{FD.distinct [X17 X13]}
{FD.distinct [X21 X13]}
{FD.distinct [X14 X13]}
{FD.distinct [X18 X13]}
{FD.distinct [X23 X13]}
{FD.distinct [X13 X12]}
{FD.distinct [X13 X11]}
{FD.distinct [X13 X10]}
```

```
{FD.distinct [X13 X9]}
{FD.distinct [X13 X8]}
{FD.distinct [X13 X7]}
{FD.distinct [X13 X3]}
{FD.distinct [X13 X1]}
{FD.distinct [X18 X14]}
{FD.distinct [X22 X14]}
{FD.distinct [X19 X14]}
{FD.distinct [X24 X14]}
{FD.distinct [X14 X13]}
{FD.distinct [X14 X12]}
{FD.distinct [X14 X11]}
{FD.distinct [X14 X10]}
{FD.distinct [X14 X9]}
{FD.distinct [X14 X8]}
{FD.distinct [X14 X4]}
{FD.distinct [X14 X2]}
{FD.distinct [X21 X15]}
{FD.distinct [X16 X15]}
{FD.distinct [X17 X15]}
{FD.distinct [X18 X15]}
{FD.distinct [X19 X15]}
{FD.distinct [X20 X15]}
{FD.distinct [X15 X11]}
{FD.distinct [X15 X10]}
{FD.distinct [X15 X7]}
{FD.distinct [X15 X5]}
{FD.distinct [X15 X3]}
{FD.distinct [X15 X0]}
{FD.distinct [X22 X16]}
{FD.distinct [X20 X16]}
{FD.distinct [X17 X16]}
{FD.distinct [X18 X16]}
{FD.distinct [X19 X16]}
{FD.distinct [X21 X16]}
{FD.distinct [X16 X15]}
{FD.distinct [X16 X12]}
{FD.distinct [X16 X11]}
{FD.distinct [X16 X10]}
{FD.distinct [X16 X8]}
{FD.distinct [X16 X6]}
{FD.distinct [X16 X4]}
{FD.distinct [X16 X1]}
{FD.distinct [X23 X17]}
{FD.distinct [X21 X17]}
{FD.distinct [X18 X17]}
{FD.distinct [X19 X17]}
{FD.distinct [X22 X17]}
{FD.distinct [X17 X16]}
{FD.distinct [X17 X15]}
{FD.distinct [X17 X13]}
{FD.distinct [X17 X12]}
{FD.distinct [X17 X11]}
{FD.distinct [X17 X9]}
{FD.distinct [X17 X7]}
{FD.distinct [X17 X5]}
{FD.distinct [X17 X2]}
```

```
{FD.distinct [X24 X18]}
{FD.distinct [X22 X18]}
{FD.distinct [X19 X18]}
{FD.distinct [X23 X18]}
{FD.distinct [X18 X17]}
{FD.distinct [X18 X16]}
{FD.distinct [X18 X15]}
{FD.distinct [X18 X14]}
{FD.distinct [X18 X13]}
{FD.distinct [X18 X12]}
{FD.distinct [X18 X8]}
{FD.distinct [X18 X6]}
{FD.distinct [X18 X3]}
{FD.distinct [X18 X0]}
{FD.distinct [X23 X19]}
{FD.distinct [X24 X19]}
{FD.distinct [X19 X18]}
{FD.distinct [X19 X17]}
{FD.distinct [X19 X16]}
{FD.distinct [X19 X15]}
{FD.distinct [X19 X14]}
{FD.distinct [X19 X13]}
{FD.distinct [X19 X9]}
{FD.distinct [X19 X7]}
{FD.distinct [X19 X4]}
{FD.distinct [X19 X1]}
{FD.distinct [X21 X20]}
{FD.distinct [X22 X20]}
{FD.distinct [X23 X20]}
{FD.distinct [X24 X20]}
{FD.distinct [X20 X16]}
{FD.distinct [X20 X15]}
{FD.distinct [X20 X12]}
{FD.distinct [X20 X10]}
{FD.distinct [X20 X8]}
{FD.distinct [X20 X5]}
{FD.distinct [X20 X4]}
{FD.distinct [X20 X0]}
{FD.distinct [X22 X21]}
{FD.distinct [X23 X21]}
{FD.distinct [X24 X21]}
{FD.distinct [X21 X20]}
{FD.distinct [X21 X17]}
{FD.distinct [X21 X16]}
{FD.distinct [X21 X15]}
{FD.distinct [X21 X13]}
{FD.distinct [X21 X11]}
{FD.distinct [X21 X9]}
{FD.distinct [X21 X6]}
{FD.distinct [X21 X1]}
{FD.distinct [X23 X22]}
{FD.distinct [X24 X22]}
{FD.distinct [X22 X21]}
{FD.distinct [X22 X20]}
{FD.distinct [X22 X18]}
{FD.distinct [X22 X17]}
{FD.distinct [X22 X16]}
```

```
   {FD.distinct [X22 X14]}
   {FD.distinct [X22 X12]}
   {FD.distinct [X22 X10]}
   {FD.distinct [X22 X7]}
   {FD.distinct [X22 X2]}
   {FD.distinct [X24 X23]}
   {FD.distinct [X23 X22]}
   {FD.distinct [X23 X21]}
   {FD.distinct [X23 X20]}
   {FD.distinct [X23 X19]}
   {FD.distinct [X23 X18]}
   {FD.distinct [X23 X17]}
   {FD.distinct [X23 X13]}
   {FD.distinct [X23 X11]}
   {FD.distinct [X23 X8]}
   {FD.distinct [X23 X5]}
   {FD.distinct [X23 X3]}
   {FD.distinct [X24 X23]}
   {FD.distinct [X24 X22]}
   {FD.distinct [X24 X21]}
   {FD.distinct [X24 X20]}
   {FD.distinct [X24 X19]}
   {FD.distinct [X24 X18]}
   {FD.distinct [X24 X14]}
   {FD.distinct [X24 X12]}
   {FD.distinct [X24 X9]}
   {FD.distinct [X24 X6]}
   {FD.distinct [X24 X4]}
   {FD.distinct [X24 X0]}

   {System.show 'Before search: time = '}
   {System.show {Property.get 'time.total'}}

   {FD.distribute min S}
end

% Solves the problem using solution clusters
proc {PFullUsingSolutionClusters S}
   X0 = {FD.int 0#4}
   X1 = {FD.int 0#4}
   X2 = {FD.int 0#4}
   X3 = {FD.int 0#4}
   X4 = {FD.int 0#4}
   X5 = {FD.int 0#4}
   X6 = {FD.int 0#4}
   X7 = {FD.int 0#4}
   X8 = {FD.int 0#4}
   X9 = {FD.int 0#4}
   X10 = {FD.int 0#4}
   X11 = {FD.int 0#4}
   X12 = {FD.int 0#4}
   X13 = {FD.int 0#4}
   X14 = {FD.int 0#4}
   X15 = {FD.int 0#4}
   X16 = {FD.int 0#4}
   X17 = {FD.int 0#4}
   X18 = {FD.int 0#4}
```

```
   X19 = {FD.int 0#4}
   X20 = {FD.int 0#4}
   X21 = {FD.int 0#4}
   X22 = {FD.int 0#4}
   X23 = {FD.int 0#4}
   X24 = {FD.int 0#4}
in
   S = allVars(
      x0:X0 x1:X1 x2:X2 x3:X3 x4:X4
      x5:X5 x6:X6 x7:X7 x8:X8 x9:X9
      x10:X10 x11:X11 x12:X12 x13:X13 x14:X14
      x15:X15 x16:X16 x17:X17 x18:X18 x19:X19
      x20:X20 x21:X21 x22:X22 x23:X23 x24:X24
   )

   % Applying constraints
   {FD.distinct [X14 X8]}
   {FD.distinct [X12 X8]}
   {FD.distinct [X16 X8]}
   {FD.distinct [X20 X8]}
   {FD.distinct [X9 X8]}
   {FD.distinct [X13 X8]}
   {FD.distinct [X18 X8]}
   {FD.distinct [X23 X8]}
   {FD.distinct [X8 X7]}
   {FD.distinct [X8 X6]}
   {FD.distinct [X8 X5]}
   {FD.distinct [X8 X4]}
   {FD.distinct [X8 X3]}
   {FD.distinct [X8 X2]}
   {FD.distinct [X13 X9]}
   {FD.distinct [X17 X9]}
   {FD.distinct [X21 X9]}
   {FD.distinct [X14 X9]}
   {FD.distinct [X19 X9]}
   {FD.distinct [X24 X9]}
   {FD.distinct [X9 X8]}
   {FD.distinct [X9 X7]}
   {FD.distinct [X9 X6]}
   {FD.distinct [X9 X5]}
   {FD.distinct [X9 X4]}
   {FD.distinct [X9 X3]}
   {FD.distinct [X16 X10]}
   {FD.distinct [X22 X10]}
   {FD.distinct [X11 X10]}
   {FD.distinct [X12 X10]}
   {FD.distinct [X13 X10]}
   {FD.distinct [X14 X10]}
   {FD.distinct [X15 X10]}
   {FD.distinct [X20 X10]}
   {FD.distinct [X10 X6]}
   {FD.distinct [X10 X5]}
   {FD.distinct [X10 X2]}
   {FD.distinct [X10 X0]}
   {FD.distinct [X17 X11]}
   {FD.distinct [X23 X11]}
   {FD.distinct [X15 X11]}
```

```
{FD.distinct [X12 X11]}
{FD.distinct [X13 X11]}
{FD.distinct [X14 X11]}
{FD.distinct [X16 X11]}
{FD.distinct [X21 X11]}
{FD.distinct [X11 X10]}
{FD.distinct [X11 X7]}
{FD.distinct [X11 X6]}
{FD.distinct [X11 X5]}
{FD.distinct [X11 X3]}
{FD.distinct [X11 X1]}
{FD.distinct [X18 X12]}
{FD.distinct [X24 X12]}
{FD.distinct [X16 X12]}
{FD.distinct [X20 X12]}
{FD.distinct [X13 X12]}
{FD.distinct [X14 X12]}
{FD.distinct [X17 X12]}
{FD.distinct [X22 X12]}
{FD.distinct [X12 X11]}
{FD.distinct [X12 X10]}
{FD.distinct [X12 X8]}
{FD.distinct [X12 X7]}
{FD.distinct [X12 X6]}
{FD.distinct [X12 X4]}
{FD.distinct [X12 X2]}
{FD.distinct [X12 X0]}
{FD.distinct [X19 X13]}
{FD.distinct [X17 X13]}
{FD.distinct [X21 X13]}
{FD.distinct [X14 X13]}
{FD.distinct [X18 X13]}
{FD.distinct [X23 X13]}
{FD.distinct [X13 X12]}
{FD.distinct [X13 X11]}
{FD.distinct [X13 X10]}
{FD.distinct [X13 X9]}
{FD.distinct [X13 X8]}
{FD.distinct [X13 X7]}
{FD.distinct [X13 X3]}
{FD.distinct [X13 X1]}
{FD.distinct [X18 X14]}
{FD.distinct [X22 X14]}
{FD.distinct [X19 X14]}
{FD.distinct [X24 X14]}
{FD.distinct [X14 X13]}
{FD.distinct [X14 X12]}
{FD.distinct [X14 X11]}
{FD.distinct [X14 X10]}
{FD.distinct [X14 X9]}
{FD.distinct [X14 X8]}
{FD.distinct [X14 X4]}
{FD.distinct [X14 X2]}
{FD.distinct [X21 X15]}
{FD.distinct [X16 X15]}
{FD.distinct [X17 X15]}
{FD.distinct [X18 X15]}
```

```
{FD.distinct [X19 X15]}
{FD.distinct [X20 X15]}
{FD.distinct [X15 X11]}
{FD.distinct [X15 X10]}
{FD.distinct [X15 X7]}
{FD.distinct [X15 X5]}
{FD.distinct [X15 X3]}
{FD.distinct [X15 X0]}
{FD.distinct [X22 X16]}
{FD.distinct [X20 X16]}
{FD.distinct [X17 X16]}
{FD.distinct [X18 X16]}
{FD.distinct [X19 X16]}
{FD.distinct [X21 X16]}
{FD.distinct [X16 X15]}
{FD.distinct [X16 X12]}
{FD.distinct [X16 X11]}
{FD.distinct [X16 X10]}
{FD.distinct [X16 X8]}
{FD.distinct [X16 X6]}
{FD.distinct [X16 X4]}
{FD.distinct [X16 X1]}
{FD.distinct [X23 X17]}
{FD.distinct [X21 X17]}
{FD.distinct [X18 X17]}
{FD.distinct [X19 X17]}
{FD.distinct [X22 X17]}
{FD.distinct [X17 X16]}
{FD.distinct [X17 X15]}
{FD.distinct [X17 X13]}
{FD.distinct [X17 X12]}
{FD.distinct [X17 X11]}
{FD.distinct [X17 X9]}
{FD.distinct [X17 X7]}
{FD.distinct [X17 X5]}
{FD.distinct [X17 X2]}
{FD.distinct [X24 X18]}
{FD.distinct [X22 X18]}
{FD.distinct [X19 X18]}
{FD.distinct [X23 X18]}
{FD.distinct [X18 X17]}
{FD.distinct [X18 X16]}
{FD.distinct [X18 X15]}
{FD.distinct [X18 X14]}
{FD.distinct [X18 X13]}
{FD.distinct [X18 X12]}
{FD.distinct [X18 X8]}
{FD.distinct [X18 X6]}
{FD.distinct [X18 X3]}
{FD.distinct [X18 X0]}
{FD.distinct [X23 X19]}
{FD.distinct [X24 X19]}
{FD.distinct [X19 X18]}
{FD.distinct [X19 X17]}
{FD.distinct [X19 X16]}
{FD.distinct [X19 X15]}
{FD.distinct [X19 X14]}
```

```
{FD.distinct [X19 X13]}
{FD.distinct [X19 X9]}
{FD.distinct [X19 X7]}
{FD.distinct [X19 X4]}
{FD.distinct [X19 X1]}
{FD.distinct [X21 X20]}
{FD.distinct [X22 X20]}
{FD.distinct [X23 X20]}
{FD.distinct [X24 X20]}
{FD.distinct [X20 X16]}
{FD.distinct [X20 X15]}
{FD.distinct [X20 X12]}
{FD.distinct [X20 X10]}
{FD.distinct [X20 X8]}
{FD.distinct [X20 X5]}
{FD.distinct [X20 X4]}
{FD.distinct [X20 X0]}
{FD.distinct [X22 X21]}
{FD.distinct [X23 X21]}
{FD.distinct [X24 X21]}
{FD.distinct [X21 X20]}
{FD.distinct [X21 X17]}
{FD.distinct [X21 X16]}
{FD.distinct [X21 X15]}
{FD.distinct [X21 X13]}
{FD.distinct [X21 X11]}
{FD.distinct [X21 X9]}
{FD.distinct [X21 X6]}
{FD.distinct [X21 X1]}
{FD.distinct [X23 X22]}
{FD.distinct [X24 X22]}
{FD.distinct [X22 X21]}
{FD.distinct [X22 X20]}
{FD.distinct [X22 X18]}
{FD.distinct [X22 X17]}
{FD.distinct [X22 X16]}
{FD.distinct [X22 X14]}
{FD.distinct [X22 X12]}
{FD.distinct [X22 X10]}
{FD.distinct [X22 X7]}
{FD.distinct [X22 X2]}
{FD.distinct [X24 X23]}
{FD.distinct [X23 X22]}
{FD.distinct [X23 X21]}
{FD.distinct [X23 X20]}
{FD.distinct [X23 X19]}
{FD.distinct [X23 X18]}
{FD.distinct [X23 X17]}
{FD.distinct [X23 X13]}
{FD.distinct [X23 X11]}
{FD.distinct [X23 X8]}
{FD.distinct [X23 X5]}
{FD.distinct [X23 X3]}
{FD.distinct [X24 X23]}
{FD.distinct [X24 X22]}
{FD.distinct [X24 X21]}
{FD.distinct [X24 X20]}
```

```
   {FD.distinct [X24 X19]}
   {FD.distinct [X24 X18]}
   {FD.distinct [X24 X14]}
   {FD.distinct [X24 X12]}
   {FD.distinct [X24 X9]}
   {FD.distinct [X24 X6]}
   {FD.distinct [X24 X4]}
   {FD.distinct [X24 X0]}
   {System.show 'Now reducing domains in
PFullUsingSolutionClusters'}
   {ReduceGlobalDomains S}

   {System.show 'Before search: time = '}
   {System.show {Property.get 'time.total'}}

   {FD.distribute min S}
end

in

   Args = {Application.getCmdArgs ArgSpec}

   if {Value.hasFeature Args useclusters} then
      {CallP1InASeparateThread}
      {CallP2InASeparateThread}
      {Wait ThreadCheck1}
      {Wait ThreadCheck2}
      Solutions = {Search.all PFullUsingSolutionClusters 30 _}
   else
      Solutions = {Search.all PFullWithoutClustering 30 _}
   end

   {System.show 'After search: time = '}
   {System.show {Property.get 'time.total'}}

   {System.showInfo "\nFound "#{Length Solutions}#" solution(s)"}
   {ForAll Solutions
      proc {$ ASolution}
         {System.showInfo "\nNext solution"}
         {System.show ASolution}
      end
   }
   {System.showInfo "\nNo. of solutions printed above = "#{Length
Solutions}#" solution(s)"}
   {Delay 1000}
   {Application.exit 0}

end
```

APPENDIX D

OUTPUT OF SOLVING
WITHOUT USING SOLUTIONS CLUSTERS APPROACH

These shows that the time taken to solve = 219 – 47 = 172 milliseconds.

```
O:\Personal\MS\Thesis\translator>Out.exe
'Before search: time = '
47
'After search: time = '
219

Found 360 solution(s)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:3 x12:2 x13:4 x14:0 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:4 x12:2 x13:3 x14:0 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:2 x12:3 x13:4 x14:0 x15:4 x16:0 x17:1 ,,,)

Next solution
```

allVars(x0:0 x1:1 x10:1 x11:4 x12:3 x13:2 x14:0 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:2 x12:4 x13:3 x14:0 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:3 x12:4 x13:2 x14:0 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:3 x12:1 x13:4 x14:0 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:4 x12:1 x13:3 x14:0 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:1 x12:3 x13:4 x14:0 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:4 x12:3 x13:1 x14:0 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:1 x12:4 x13:3 x14:0 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:3 x12:4 x13:1 x14:0 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:3 x11:0 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:2 x12:1 x13:4 x14:0 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:3 x11:0 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:4 x12:1 x13:2 x14:0 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:3 x11:0 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:1 x12:2 x13:4 x14:0 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:4 x12:2 x13:1 x14:0 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:1 x12:4 x13:2 x14:0 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:3 x11:0 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:2 x12:4 x13:1 x14:0 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution

allVars(x0:0 x1:2 x10:4 x11:0 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:2 x12:1 x13:3 x14:0 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:3 x12:1 x13:2 x14:0 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:4 x11:0 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:1 x12:2 x13:3 x14:0 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:3 x12:2 x13:1 x14:0 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:4 x11:0 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:1 x12:3 x13:2 x14:0 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:2 x12:3 x13:1 x14:0 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:4 x11:0 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:0 x11:1 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:3 x12:2 x13:4 x14:1 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:4 x12:2 x13:3 x14:1 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:0 x11:1 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:2 x12:3 x13:4 x14:1 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:4 x12:3 x13:2 x14:1 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:0 x11:1 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:2 x12:4 x13:3 x14:1 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:3 x12:4 x13:2 x14:1 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:0 x11:1 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:3 x12:0 x13:4 x14:1 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:2 x11:1 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:4 x12:0 x13:3 x14:1 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:0 x12:3 x13:4 x14:1 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:4 x12:3 x13:0 x14:1 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution

allVars(x0:1 x1:4 x10:2 x11:1 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:0 x12:4 x13:3 x14:1 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:1 x1:4 x10:2 x11:1 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:3 x12:4 x13:0 x14:1 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:2 x11:1 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:2 x12:0 x13:4 x14:1 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:4 x12:0 x13:2 x14:1 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:0 x12:2 x13:4 x14:1 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:4 x12:2 x13:0 x14:1 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:0 x12:4 x13:2 x14:1 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:2 x12:4 x13:0 x14:1 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:2 x12:0 x13:3 x14:1 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:3 x12:0 x13:2 x14:1 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:0 x12:2 x13:3 x14:1 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:3 x12:2 x13:0 x14:1 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:0 x12:3 x13:2 x14:1 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:2 x12:3 x13:0 x14:1 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:2 x1:1 x10:0 x11:2 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:3 x12:1 x13:4 x14:2 x15:4 x16:2 x17:0 ,,,)

Next solution

allVars(x0:2 x1:1 x10:0 x11:2 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:4 x12:1 x13:3 x14:2 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:0 x11:2 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:1 x12:3 x13:4 x14:2 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:4 x12:3 x13:1 x14:2 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:1 x12:4 x13:3 x14:2 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:0 x11:2 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:3 x12:4 x13:1 x14:2 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:3 x12:0 x13:4 x14:2 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:4 x12:0 x13:3 x14:2 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:0 x12:3 x13:4 x14:2 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:4 x12:3 x13:0 x14:2 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:0 x12:4 x13:3 x14:2 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:3 x12:4 x13:0 x14:2 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:3 x11:2 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:1 x12:0 x13:4 x14:2 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:3 x11:2 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:4 x12:0 x13:1 x14:2 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:3 x11:2 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:0 x12:1 x13:4 x14:2 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:4 x12:1 x13:0 x14:2 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:0 x12:4 x13:1 x14:2 x15:1 x16:2 x17:3 ,,,)

Next solution

allVars(x0:2 x1:0 x10:3 x11:2 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:1 x12:4 x13:0 x14:2 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:1 x12:0 x13:3 x14:2 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:3 x12:0 x13:1 x14:2 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:0 x12:1 x13:3 x14:2 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:3 x12:1 x13:0 x14:2 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:0 x12:3 x13:1 x14:2 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:1 x12:3 x13:0 x14:2 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:0 x11:3 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:2 x12:1 x13:4 x14:3 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:4 x12:1 x13:2 x14:3 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:0 x11:3 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:1 x12:2 x13:4 x14:3 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:4 x12:2 x13:1 x14:3 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:0 x11:3 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:1 x12:4 x13:2 x14:3 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:2 x12:4 x13:1 x14:3 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:0 x11:3 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:2 x12:0 x13:4 x14:3 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution

allVars(x0:3 x1:1 x10:1 x11:4 x12:0 x13:2 x14:3 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:0 x12:2 x13:4 x14:3 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:4 x12:2 x13:0 x14:3 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:0 x12:4 x13:2 x14:3 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:2 x12:4 x13:0 x14:3 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:1 x12:0 x13:4 x14:3 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:4 x12:0 x13:1 x14:3 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:0 x12:1 x13:4 x14:3 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:4 x12:1 x13:0 x14:3 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:0 x12:4 x13:1 x14:3 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:1 x12:4 x13:0 x14:3 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:1 x12:0 x13:2 x14:3 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:2 x12:0 x13:1 x14:3 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:0 x12:1 x13:2 x14:3 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:2 x12:1 x13:0 x14:3 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:0 x12:2 x13:1 x14:3 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution

allVars(x0:3 x1:4 x10:4 x11:1 x12:2 x13:0 x14:3 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:2 x12:1 x13:3 x14:4 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:3 x12:1 x13:2 x14:4 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:1 x12:2 x13:3 x14:4 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:3 x12:2 x13:1 x14:4 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:1 x12:3 x13:2 x14:4 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:2 x12:3 x13:1 x14:4 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:2 x12:0 x13:3 x14:4 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:3 x12:0 x13:2 x14:4 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:0 x12:2 x13:3 x14:4 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:3 x12:2 x13:0 x14:4 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:0 x12:3 x13:2 x14:4 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:2 x12:3 x13:0 x14:4 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:2 x11:4 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:1 x12:0 x13:3 x14:4 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:3 x12:0 x13:1 x14:4 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution

allVars(x0:4 x1:1 x10:2 x11:4 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:0 x12:1 x13:3 x14:4 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:3 x12:1 x13:0 x14:4 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:1 x10:2 x11:4 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:0 x12:3 x13:1 x14:4 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:1 x12:3 x13:0 x14:4 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:1 x10:2 x11:4 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:1 x12:0 x13:2 x14:4 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:2 x12:0 x13:1 x14:4 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:0 x12:1 x13:2 x14:4 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:2 x12:1 x13:0 x14:4 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:0 x12:2 x13:1 x14:4 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:1 x12:2 x13:0 x14:4 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

No. of solutions printed above = 360 solution(s)

APPENDIX E

OUTPUT OF SOLVING WITH THE SOLUTIONS CLUSTERS APPROACH

These shows that the time taken to solve = 19016 – 18860 = 156 milliseconds.

O:\Personal\MS\Thesis\translator>Out.exe -c
"
'CALLING P1 IN A SEPARATE THREAD'
"
'CALLING P2 IN A SEPARATE THREAD'
'P2: Returned from P2'
sol(z10:0 z11:1 z12:2 z13:3 z14:4 z15:2 z16:3 z17:4 z18:0 z19:1 ,,,)|sol(z10:0 z11:1 z12:2 z13:3 z1
4:4 z15:3 z16:4 z17:0 z18:1 z19:2 ,,,)|sol(z10:0 z11:1 z12:2 z13:4 z14:3 z15:4 z16:3 z17:0 z18:1 z1
9:2 ,,,)|sol(z10:0 z11:1 z12:2 z13:4 z14:3 z15:2 z16:4 z17:3 z18:0 z19:1 ,,,)|sol(z10:0 z11:1 z12:3
z13:2 z14:4 z15:3 z16:2 z17:4 z18:0 z19:1 ,,,)|sol(z10:0 z11:1 z12:3 z13:2 z14:4 z15:2 z16:4 z17:0
z18:1 z19:3 ,,,)|sol(z10:0 z11:1 z12:3 z13:4 z14:2 z15:4 z16:2 z17:0 z18:1 z19:3 ,,,)|sol(z10:0 z1
1:1 z12:3 z13:4 z14:2 z15:3 z16:4 z17:2 z18:0 z19:1 ,,,)|sol(z10:0 z11:1 z12:4 z13:2 z14:3 z15:4 z1
6:2 z17:3 z18:0 z19:1 ,,,)|sol(z10:0 z11:1 z12:4 z13:2 z14:3 z15:2 z16:3 z17:0 z18:1 z19:4 ,,,)|,,,
|,,,
'P2: Number of solutions = '#240
"
'P1: Returned from P1'
sol(y0:0 y1:1 y10:1 y11:0 y12:2 y13:3 y14:4 y15:2 y16:3 y2:0 ,,,)|sol(y0:0 y1:2 y10:1 y11:0 y12:2 y
13:3 y14:4 y15:2 y16:3 y2:0 ,,,)|sol(y0:0 y1:4 y10:1 y11:0 y12:2 y13:3 y14:4 y15:2 y16:3 y2:0 ,,,)|
sol(y0:0 y1:1 y10:1 y11:0 y12:2 y13:3 y14:4 y15:2 y16:3 y2:0 ,,,)|sol(y0:0 y1:2 y10:1 y11:0 y12:2 y
13:3 y14:4 y15:2 y16:3 y2:0 ,,,)|sol(y0:0 y1:4 y10:1 y11:0 y12:2 y13:3 y14:4 y15:2 y16:3 y2:0 ,,,)|
sol(y0:0 y1:1 y10:1 y11:0 y12:2 y13:3 y14:4 y15:2 y16:3 y2:3 ,,,)|sol(y0:0 y1:2 y10:1 y11:0 y12:2 y
13:3 y14:4 y15:2 y16:3 y2:3 ,,,)|sol(y0:0 y1:4 y10:1 y11:0 y12:2 y13:3 y14:4 y15:2 y16:3 y2:3 ,,,)|

sol(y0:0 y1:1 y10:1 y11:0 y12:2 y13:3 y14:4 y15:2 y16:3 y2:3 ,,,)|,,,|,,,
'P1: Number of solutions = '#46800
"
'Now reducing domains in PFullUsingSolutionClusters'
'Before search: time = '
18860
'After search: time = '
19016

Found 360 solution(s)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:3 x12:2 x13:4 x14:0 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:4 x12:2 x13:3 x14:0 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:2 x12:3 x13:4 x14:0 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:4 x12:3 x13:2 x14:0 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:2 x12:4 x13:3 x14:0 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:1 x11:0 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:1 x11:3 x12:4 x13:2 x14:0 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:1 x11:0 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:1 x11:0 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:3 x12:1 x13:4 x14:0 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:4 x12:1 x13:3 x14:0 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:1 x12:3 x13:4 x14:0 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:4 x12:3 x13:1 x14:0 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:1 x12:4 x13:3 x14:0 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:2 x11:0 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:2 x11:3 x12:4 x13:1 x14:0 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:2 x11:0 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:2 x11:0 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:3 x11:0 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:2 x12:1 x13:4 x14:0 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:3 x11:0 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:4 x12:1 x13:2 x14:0 x15:2 x16:0 x17:3 ,,,)

Next solution

allVars(x0:0 x1:1 x10:3 x11:0 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:1 x12:2 x13:4 x14:0 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:4 x12:2 x13:1 x14:0 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:1 x12:4 x13:2 x14:0 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:1 x10:3 x11:0 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:3 x11:2 x12:4 x13:1 x14:0 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:0 x1:2 x10:3 x11:0 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:0 x1:4 x10:3 x11:0 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:0 x1:2 x10:4 x11:0 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:2 x12:1 x13:3 x14:0 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:3 x12:1 x13:2 x14:0 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:2 x10:4 x11:0 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:1 x12:2 x13:3 x14:0 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:3 x12:2 x13:1 x14:0 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:4 x11:0 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:1 x12:3 x13:2 x14:0 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:1 x10:4 x11:0 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:0 x1:4 x10:4 x11:2 x12:3 x13:1 x14:0 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:0 x1:2 x10:4 x11:0 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:0 x1:3 x10:4 x11:0 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:0 x11:1 x12:2 x13:4 x14:3 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:3 x12:2 x13:4 x14:1 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:2 x13:3 x14:4 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:4 x12:2 x13:3 x14:1 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:0 x11:1 x12:3 x13:4 x14:2 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:2 x12:3 x13:4 x14:1 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:4 x12:3 x13:2 x14:1 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:0 x11:1 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:3 x13:2 x14:4 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:2 x12:4 x13:3 x14:1 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:0 x11:1 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:4 x13:3 x14:2 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:0 x11:3 x12:4 x13:2 x14:1 x15:2 x16:1 x17:0 ,,,)

Next solution

allVars(x0:1 x1:3 x10:0 x11:1 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:0 x11:1 x12:4 x13:2 x14:3 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:3 x12:0 x13:4 x14:1 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:2 x11:1 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:4 x12:0 x13:3 x14:1 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:0 x12:3 x13:4 x14:1 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:4 x12:3 x13:0 x14:1 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:2 x11:1 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:0 x12:4 x13:3 x14:1 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:2 x11:1 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:1 x1:4 x10:2 x11:1 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:2 x11:3 x12:4 x13:0 x14:1 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:2 x11:1 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:2 x11:1 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:2 x12:0 x13:4 x14:1 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:4 x12:0 x13:2 x14:1 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:0 x12:2 x13:4 x14:1 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:4 x12:2 x13:0 x14:1 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:0 x12:4 x13:2 x14:1 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:0 x10:3 x11:1 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:3 x11:2 x12:4 x13:0 x14:1 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:1 x1:2 x10:3 x11:1 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:1 x1:4 x10:3 x11:1 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:2 x12:0 x13:3 x14:1 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:3 x12:0 x13:2 x14:1 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution

allVars(x0:1 x1:4 x10:4 x11:0 x12:2 x13:3 x14:1 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:3 x12:2 x13:0 x14:1 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:0 x12:3 x13:2 x14:1 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:0 x10:4 x11:1 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:1 x1:4 x10:4 x11:2 x12:3 x13:0 x14:1 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:1 x1:2 x10:4 x11:1 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:1 x1:3 x10:4 x11:1 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:2 x1:1 x10:0 x11:2 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:1 x13:4 x14:3 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:3 x12:1 x13:4 x14:2 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:0 x11:2 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:1 x13:3 x14:4 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:4 x12:1 x13:3 x14:2 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:0 x11:2 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:3 x13:4 x14:1 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:1 x12:3 x13:4 x14:2 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:4 x12:3 x13:1 x14:2 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:3 x13:1 x14:4 x15:3 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:1 x12:4 x13:3 x14:2 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:0 x11:2 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:4 x13:3 x14:1 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:0 x11:3 x12:4 x13:1 x14:2 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:0 x11:2 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:0 x11:2 x12:4 x13:1 x14:3 x15:4 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:0 x13:4 x14:3 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:3 x12:0 x13:4 x14:2 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:0 x13:3 x14:4 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:4 x12:0 x13:3 x14:2 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:3 x13:4 x14:0 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:0 x12:3 x13:4 x14:2 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:4 x12:3 x13:0 x14:2 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:3 x13:0 x14:4 x15:3 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:0 x12:4 x13:3 x14:2 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:1 x11:2 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:4 x13:3 x14:0 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:1 x11:3 x12:4 x13:0 x14:2 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:1 x11:2 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:1 x11:2 x12:4 x13:0 x14:3 x15:4 x16:0 x17:3 ,,,)

Next solution

allVars(x0:2 x1:0 x10:3 x11:2 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:1 x12:0 x13:4 x14:2 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:3 x11:2 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:4 x12:0 x13:1 x14:2 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:3 x11:2 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:0 x12:1 x13:4 x14:2 x15:4 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:4 x12:1 x13:0 x14:2 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:0 x12:4 x13:1 x14:2 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:0 x10:3 x11:2 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:3 x11:1 x12:4 x13:0 x14:2 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:2 x1:1 x10:3 x11:2 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:2 x1:4 x10:3 x11:2 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:1 x12:0 x13:3 x14:2 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:3 x12:0 x13:1 x14:2 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:0 x12:1 x13:3 x14:2 x15:3 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:3 x12:1 x13:0 x14:2 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:0 x12:3 x13:1 x14:2 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:0 x10:4 x11:2 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:2 x1:4 x10:4 x11:1 x12:3 x13:0 x14:2 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:2 x1:1 x10:4 x11:2 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:2 x1:3 x10:4 x11:2 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:0 x11:3 x12:1 x13:4 x14:2 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:2 x12:1 x13:4 x14:3 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:1 x13:2 x14:4 x15:1 x16:2 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:4 x12:1 x13:2 x14:3 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:0 x11:3 x12:2 x13:4 x14:1 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:1 x12:2 x13:4 x14:3 x15:4 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:4 x12:2 x13:1 x14:3 x15:1 x16:3 x17:0 ,,,)

Next solution

allVars(x0:3 x1:2 x10:0 x11:3 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:2 x13:1 x14:4 x15:2 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:1 x12:4 x13:2 x14:3 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:0 x11:3 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:4 x13:2 x14:1 x15:4 x16:2 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:0 x11:2 x12:4 x13:1 x14:3 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:0 x11:3 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:0 x11:3 x12:4 x13:1 x14:2 x15:4 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:0 x13:4 x14:2 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:2 x12:0 x13:4 x14:3 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:0 x13:2 x14:4 x15:0 x16:2 x17:4 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:4 x12:0 x13:2 x14:3 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:2 x13:4 x14:0 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:0 x12:2 x13:4 x14:3 x15:4 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:4 x12:2 x13:0 x14:3 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:2 x13:0 x14:4 x15:2 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:0 x12:4 x13:2 x14:3 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:1 x11:3 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:4 x13:2 x14:0 x15:4 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:1 x11:2 x12:4 x13:0 x14:3 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:1 x11:3 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:1 x11:3 x12:4 x13:0 x14:2 x15:4 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:0 x13:4 x14:1 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:1 x12:0 x13:4 x14:3 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:0 x13:1 x14:4 x15:0 x16:1 x17:4 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:4 x12:0 x13:1 x14:3 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:1 x13:4 x14:0 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:0 x12:1 x13:4 x14:3 x15:4 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:4 x12:1 x13:0 x14:3 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:1 x13:0 x14:4 x15:1 x16:0 x17:4 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:0 x12:4 x13:1 x14:3 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:0 x10:2 x11:3 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:4 x13:1 x14:0 x15:4 x16:1 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:2 x11:1 x12:4 x13:0 x14:3 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:3 x1:1 x10:2 x11:3 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:4 x10:2 x11:3 x12:4 x13:0 x14:1 x15:4 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution

allVars(x0:3 x1:4 x10:4 x11:1 x12:0 x13:2 x14:3 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:2 x12:0 x13:1 x14:3 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:0 x12:1 x13:2 x14:3 x15:2 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:2 x12:1 x13:0 x14:3 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:0 x12:2 x13:1 x14:3 x15:1 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:0 x10:4 x11:3 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:3 x1:4 x10:4 x11:1 x12:2 x13:0 x14:3 x15:0 x16:3 x17:4 ,,,)

Next solution
allVars(x0:3 x1:1 x10:4 x11:3 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:3 x1:2 x10:4 x11:3 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:1 x13:3 x14:2 x15:1 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:2 x12:1 x13:3 x14:4 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:1 x13:2 x14:3 x15:1 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:3 x12:1 x13:2 x14:4 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:2 x13:3 x14:1 x15:2 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:1 x12:2 x13:3 x14:4 x15:3 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:3 x12:2 x13:1 x14:4 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:2 x13:1 x14:3 x15:2 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:1 x12:3 x13:2 x14:4 x15:2 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:0 x11:4 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:3 x13:2 x14:1 x15:3 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:0 x11:2 x12:3 x13:1 x14:4 x15:1 x16:4 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:0 x11:4 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:0 x11:4 x12:3 x13:1 x14:2 x15:3 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:0 x13:3 x14:2 x15:0 x16:3 x17:2 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:2 x12:0 x13:3 x14:4 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:0 x13:2 x14:3 x15:0 x16:2 x17:3 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:3 x12:0 x13:2 x14:4 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:2 x13:3 x14:0 x15:2 x16:3 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:0 x12:2 x13:3 x14:4 x15:3 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:3 x12:2 x13:0 x14:4 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:2 x13:0 x14:3 x15:2 x16:0 x17:3 ,,,)

Next solution

allVars(x0:4 x1:1 x10:1 x11:0 x12:3 x13:2 x14:4 x15:2 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:1 x11:4 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:3 x13:2 x14:0 x15:3 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:1 x11:2 x12:3 x13:0 x14:4 x15:0 x16:4 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:1 x11:4 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:1 x11:4 x12:3 x13:0 x14:2 x15:3 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:2 x11:4 x12:0 x13:3 x14:1 x15:0 x16:3 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:1 x12:0 x13:3 x14:4 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:0 x13:1 x14:3 x15:0 x16:1 x17:3 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:3 x12:0 x13:1 x14:4 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:2 x11:4 x12:1 x13:3 x14:0 x15:1 x16:3 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:0 x12:1 x13:3 x14:4 x15:3 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:3 x12:1 x13:0 x14:4 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:1 x10:2 x11:4 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:1 x13:0 x14:3 x15:1 x16:0 x17:3 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:0 x12:3 x13:1 x14:4 x15:1 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:0 x10:2 x11:4 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:3 x13:1 x14:0 x15:3 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:2 x11:1 x12:3 x13:0 x14:4 x15:0 x16:4 x17:2 ,,,)

Next solution
allVars(x0:4 x1:1 x10:2 x11:4 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:3 x10:2 x11:4 x12:3 x13:0 x14:1 x15:3 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:0 x13:2 x14:1 x15:0 x16:2 x17:1 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:1 x12:0 x13:2 x14:4 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:0 x13:1 x14:2 x15:0 x16:1 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:2 x12:0 x13:1 x14:4 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:1 x13:2 x14:0 x15:1 x16:2 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:0 x12:1 x13:2 x14:4 x15:2 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:2 x12:1 x13:0 x14:4 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:1 x13:0 x14:2 x15:1 x16:0 x17:2 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:0 x12:2 x13:1 x14:4 x15:1 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:0 x10:3 x11:4 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:2 x13:1 x14:0 x15:2 x16:1 x17:0 ,,,)

Next solution
allVars(x0:4 x1:3 x10:3 x11:1 x12:2 x13:0 x14:4 x15:0 x16:4 x17:3 ,,,)

Next solution
allVars(x0:4 x1:1 x10:3 x11:4 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

Next solution
allVars(x0:4 x1:2 x10:3 x11:4 x12:2 x13:0 x14:1 x15:2 x16:0 x17:1 ,,,)

No. of solutions printed above = 360 solution(s)

APPENDIX F

SAMPLE OZ PROGRAM FOR SOLVING SEND+MORE=MONEY PROBLEM IN PARTS

```
functor
import
      Search
      FD
      System
      Application
      Space
      Browser
      Explorer
define
      P1 P2Sol1
      P2 %Sol
      Ene Nne
      GlobalVars
      X1
      X2
      Sol
      Depth = {NewCell 0}

      proc {RomoveValuesNotInNewValueVar NewValueVar Var}
            ListCurDom ListNewDom ListFilteredDom
      in
            ListCurDom = {FD.reflect.domList Var}
            ListNewDom = {FD.reflect.domList NewValueVar}
            ListFilteredDom = {List.filter ListCurDom fun {$ Val}
{List.member Val ListNewDom} end}
            {List.forAll ListCurDom proc {$ X} if {List.member X
ListFilteredDom} == false then Var \=: X end end}
      end

      proc {RemoveValuesNotInNewList NewValueList Var}
            ListCurDom ListFilteredDom
      in
            ListCurDom = {FD.reflect.domList Var}
            ListFilteredDom = {List.filter ListCurDom fun {$ Val}
{List.member Val NewValueList} end}
            {List.forAll ListCurDom proc {$ X} if {List.member X
ListFilteredDom} == false then Var \=: X end end}
      end

      proc {P1 S1}
            N D R E Y
      in
            {System.show 'P1: Inside P1... adding constraints'}
            S1 = sol(n:N d:D r:R e:E y:Y)
            S1 ::: 0#9
            (D + E =: Y) + (D + E =: 10 + Y) =: 1
```

```
            (N + R =: E) + (N + R =: 10 + E) + (N + R + 1 =: E) + (N
+ R + 1 =: 10 + E) =: 1
            (D + 10*N + E + 10*R =: Y + 10*E) + (D + 10*N + E + 10*R
=: Y + 10*E + 100) =: 1
            {FD.distinct S1}

            if {FD.reflect.size Ene} < 10 then {System.show 'P1:
filtering domain of E'} {RomoveValuesNotInNewValueVar Ene E} end
            if {FD.reflect.size Nne} < 10 then {System.show 'P1:
filtering domain of N'} {RomoveValuesNotInNewValueVar Nne N} end

            {System.show 'P1: Before distribution, current values =
'#S1}
            {FD.distribute split S1}
      end

      proc {P2 S2}
            S E M O N
      in
            {System.show 'P2: Inside P2... adding constraints'}
            S2 = sol(s:S e:E m:M o:O n:N)
            S2 ::: 0#9
            S \=: 0
            M \=: 0
            (E + O =: N) + (E + O =: 10 + N) + (E + O + 1 =: N) + (E
+ O + 1 =: 10 + N) =: 1
            (S + M =: O) + (S + M =: 10 + O) + (S + M + 1 =: O) + (S
+ M + 1 =: 10 + O) =: 1
            (E + 10*S + O + 10*M =: N + O*10 + M*100) + (1 + E +
10*S + O + 10*M =: N + O*10 + M*100) =: 1
            {FD.distinct S2}
            {FD.distribute split S2}
      end

      proc {InitGlobals}
            {System.show ''}
            {System.show 'CREATING GLOBALS'}
            GlobalVars = global(nne:Nne ene:Ene)
            GlobalVars ::: 0#10
            {System.show GlobalVars}
            {System.show ''}
      end

      proc {CallP1}
            Sol
      in
            {System.show ''}
            {System.show 'CALLING P1'}
            Sol = {Search.all P1 30 _}
            {System.show 'P1: Returned from P1'}
            {System.show Sol}
            {System.show 'P1: Number of solutions = '#{List.length
Sol}}
            {System.show ''}
            X1 = unit
      end
```

```
      proc {CallP1InASeparateThread}
            Sol
      in
            thread
                  {System.show ''}
                  {System.show 'CALLING P1 IN A SEPARATE THREAD'}
                  Sol = {Search.all P1 30 _}
                  {System.show 'P1: Returned from P1'}
                  {System.show Sol}
                  {System.show 'P1: Number of solutions =
'#{List.length Sol}}
                  {System.show ''}
                  X1 = unit
            end
      end

      proc {CallP2}
            Sol
      in
            {System.show ''}
            {System.show 'CALLING P2'}
            Sol = {Search.all P2 30 _}
            {System.show 'P2: Returned from P2'}
            {System.show Sol}
            {System.show 'P2: Number of solutions = '#{List.length
Sol}}
            {System.show ''}
            P2Sol1 = Sol
            X2 = X1
      end

      proc {CallP2InASeparateThread}
            Sol
      in
            thread
                  {System.show ''}
                  {System.show 'CALLING P2 IN A SEPARATE THREAD'}
                  Sol = {Search.all P2 30 _}
                  {System.show 'P2: Returned from P2'}
                  {System.show Sol}
                  {System.show 'P2: Number of solutions =
'#{List.length Sol}}
                  {System.show ''}
                  P2Sol1 = Sol
                  X2 = X1
            end
      end

      proc {ReduceGlobalDomains}
            NewEneVals NewNneVals
      in
            {System.show 'NOW REDUCING DOMAINS OF GLOBAL VARIABLES'}
            NewEneVals = {List.map P2Sol1 fun {$ ASol} ASol.e end}
            {RemoveValuesNotInNewList NewEneVals Ene}
            NewNneVals = {List.map P2Sol1 fun {$ ASol} ASol.n end}
            {RemoveValuesNotInNewList NewNneVals Nne}
            {System.show 'Printing globals again'}
```

```
            {System.show GlobalVars}
            {System.show ''}
      end

      proc {IncreaseDepth}
            CurVal NewVal
      in
            {Exchange Depth CurVal NewVal}
            NewVal = CurVal + 1
      end

      proc {DecreaseDepth}
            CurVal NewVal
      in
            {Exchange Depth CurVal NewVal}
            NewVal = CurVal - 1
      end

      fun {DFE S}
            case {Space.ask S} of
            failed then
                  nil
            [] succeeded then
                  [S]
            [] alternatives(2) then
                  {IncreaseDepth}
                  C = {Space.clone S} in
                  {Space.commit S 1}
                  case {DFE S} of
                  nil then
                        {Space.commit C 2} {DFE C}
                  [] [T] then
                        [T]
                  end
            end
      end

      % Given {Script Sol}, returns solution [Sol] or nil:
      fun {DFS Script}
            case {DFE {Space.new Script}} of
            nil then
                  nil
            [] [S] then
                  [{Space.merge S}]
            end
      end

in

      {InitGlobals}

      %{CallP1}
      {CallP1InASeparateThread}

      %{CallP2}
      {CallP2InASeparateThread}
```

```
      {Wait X2}

      {ReduceGlobalDomains}


      {System.show 'NOW SOLVING P2 AGAIN WITH CROSS DOMAIN INFO'}
      Sol = {DFS P2}
      {System.show 'Returned from P2'}
      {System.show Sol}
      {System.show 'No. of times alternatives were sorted to = '}
      {System.show @Depth}

      %{Explorer.all P2}
      {Delay 5000}
      {Application.exit 0}
end
```

APPENDIX G

SAMPLE C CODE FOR GRAPH GENERATION

Note: This uses the very_nauty graph theory software [29].

```
#include "vn_graph.h"

int main() {
  graph_t g = graph_new(7);
  graph_gnp(g, 49.5);
  graph_show(g);
  printf("chi=%d\n",graph_chromatic_number(g,0));
  graph_to_dimacs(g, "Generated-Graph.col");
  graph_clear(g);
  return 0;
}
```

APPENDIX H

GENERATED SAMPLE GRAPHS (USING C CODE OF APPENDIX G)

Graph-A:

```
c written by graph_to_dimacs
p edge 6 15
e 1 2
e 1 3
e 1 4
e 1 5
e 1 6
e 2 3
e 2 4
e 2 5
e 2 6
e 3 4
e 3 5
e 3 6
e 4 5
e 4 6
e 5 6
```

Graph-B:

```
c written by graph_to_dimacs
p edge 7 21
e 1 2
e 1 3
e 1 4
e 1 5
e 1 6
e 1 7
e 2 3
e 2 4
e 2 5
e 2 6
e 2 7
e 3 4
e 3 5
e 3 6
e 3 7
e 4 5
e 4 6
e 4 7
e 5 6
e 5 7
e 6 7
```

Graph-C:

```
c written by graph_to_dimacs
p edge 8 28
e 1 2
e 1 3
```

```
e 1 4
e 1 5
e 1 6
e 1 7
e 1 8
e 2 3
e 2 4
e 2 5
e 2 6
e 2 7
e 2 8
e 3 4
e 3 5
e 3 6
e 3 7
e 3 8
e 4 5
e 4 6
e 4 7
e 4 8
e 5 6
e 5 7
e 5 8
e 6 7
e 6 8
e 7 8
```

Graph-D:

```
c written by graph_to_dimacs
p edge 9 36
e 1 2
e 1 3
e 1 4
e 1 5
e 1 6
e 1 7
e 1 8
e 1 9
e 2 3
e 2 4
e 2 5
e 2 6
e 2 7
e 2 8
e 2 9
e 3 4
e 3 5
e 3 6
e 3 7
e 3 8
e 3 9
e 4 5
e 4 6
e 4 7
e 4 8
e 4 9
e 5 6
e 5 7
```

```
e 5 8
e 5 9
e 6 7
e 6 8
e 6 9
e 7 8
e 7 9
e 8 9
```

Graph-E:

```
c written by graph_to_dimacs
p edge 10 45
e 1 2
e 1 3
e 1 4
e 1 5
e 1 6
e 1 7
e 1 8
e 1 9
e 1 10
e 2 3
e 2 4
e 2 5
e 2 6
e 2 7
e 2 8
e 2 9
e 2 10
e 3 4
e 3 5
e 3 6
e 3 7
e 3 8
e 3 9
e 3 10
e 4 5
e 4 6
e 4 7
e 4 8
e 4 9
e 4 10
e 5 6
e 5 7
e 5 8
e 5 9
e 5 10
e 6 7
e 6 8
e 6 9
e 6 10
e 7 8
e 7 9
e 7 10
e 8 9
e 8 10
e 9 10
```

REFERENCES

[1]: M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness. W.H. Freeman and Company, 1979.

[2]: A. Parkes. Clustering at the phase transition. In *Proc. AAAI-97*, pages 340--345, 1997.

[3]: Yokoo, M.; Durfee, E. H.; Ishida, T. and Kuwabara, K. 1998. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Transactions on Knowledge and DATA Engineering* 10 (5).

[4]: Hajian, M., Sakkout, H.El, Wallace, M., Richards, E.: Towards a Closer Integration of Finite Domain Propagation and Simplex-Based Algorithms. *Proc. of the AI Maths Conf.* Florida (1995).

[5]: P. Baptiste, C. Le Pape, and W. Nuijten. Incorporating efficient Operations Research algorithms in constraint-based scheduling. In *1st Joint Workshop on Artificial Intelligence and Operational Research*, 1995.

[6]: A. J. Parkes. Exploiting solution clusters for coarse-grained distributed search. In *Proceedings of the workshop on "Distributed Constraint Reasoning"*, held at "Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)", August 2001.

[7]: S. Kirkpatrick and B. Selman, "Critical Behavior in the satisfiability of random Boolean expressions," *Science 264*, pp 1297--1301, 1994.

[8]: Yokoo, M. and Hirayama, K. Algorithms for Distributed Constraint Satisfaction. *A Review of Autonomous Agents and Multi-Agent Systems. 2000*.

[9]: D. W. Etherington and A. J. Parkes. Exploiting Solution-Space Structure. *Technical Report, CIRL*.

[10]: Crawford, J. M., and Auton, L., 1996. Experimental results on the crossover point in radom 3-SAT. *Artificial Intelligence 81*:31-57.

[11]: Center for Discrete Mathematics & Technical Computer Science. http://dimacs.rutgers.edu/.

[12]: F. Bacchus. P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 326--333, 1998.

[13] Marriot K., and Stuckey P.: Programming with Constraints – An Introduction, The MIT Press, 1998.

[14] Harvey W., and Stuckey P. Improving linear constraint propagation by changing constraint representation. *Constraints*, 8(2):173–207, 2003.

[15] Schulte C., and Stuckey P. When do bounds and domain propagation lead to the same search space? *Transactions on Programming Languages and Systems*, 27(3):388–425, May 2005.

[16] Schulte C., and Stuckey P. Efficient Constraint Propagation Engines. *Transactions of Programming Languages and Systems*, 2007.

[17] Walser, J.P. (1997). Solving Linear Pseudo-Boolean Constraints with Local Search. In *Eleventh Conference on Artificial Intelligence*, pp. 269–274. AAAI Press.

[18] A. J. Parkes and J. P. Walser. Tuning local search for satisfiability testing. In *Proceedings of AAAI-96*, pages 356-362, 1996.

[19] Bohlin, M. Constraint satisfaction by local search. *Tech. Rep*. T2002- 07, SICS, 2002.

[20] Selman, B., H. Levesque and D. Mitchell. (1992). A New Method for Solving Hard Satisfiability Problems. In *Tenth National Conference on Artificial Intelligence*, pp. 440–446. AAAI Press.

[21] T. Schiex & G. Verfaillie (1993) No-good recording for static and dynamic constraint satisfaction problems.

[22] Van Roy, P. 1999. On the separation of concerns in distributed programming: Application to distribution structure and fault tolerance in Mozart. In *International Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications (PDSIA 99)*. Tohoku University, Sendai, Japan.

[23] Smolka, G., Schulte, C., and Van Roy, P. 1995. PERDIO--Persistent and distributed programming in Oz. *BHBF project proposal*. Available at http://www.ps.uni-sb.de.

[24] P. Van Roy et al., "Mobile Objects in Distributed Oz", *ACM Transactions on Programming Languages and Systems*, Vol. 19, No. 5, pp. 804-851, Sept. 1997.

[25] Conjunctive Normal Form (CNF): http://en.wikipedia.org/wiki/Conjunctive_normal_form.

[26] 3-SAT: http://en.wikipedia.org/wiki/3SAT.

[27] Amdahl’s law: http://en.wikipedia.org/wiki/Amdahl's_law.

[28] The mOZart programming system: http://www.mozart-oz.org/.

[29] Keith Briggs' graph theory software – very_nauty v1.1: http://keithbriggs.info/very_nauty.html.

[30] Distributed Programming in Mozart - A Tutorial Introduction: http://www.mozart-oz.org/documentation/dstutorial/index.html