Massachusetts Institute of Technology
16.412J/6.834J Cognitive Robotics

Comparison of
Russian Doll Search
and
Bucket Tree Elimination

Problem Set #2
Due: in class Wed, 3/16/05

**Lawrence Bush, Brian Bairstow,
Thomas Coffee and Shuonan Dong**

Table of Contents

Chapter 1

Comparative Performance Analysis

Lawrence Bush, Brian Bairstow, Thomas Coffee and Shuonan Dong

---

Introduction

In this paper we compare the performance of two algorithms on a set of weighted constraint satisfaction problems (WCSP). The two algorithms that we compare are Branch and Bound using Russian Doll Search (RDS) and Bucket Tree Elimination (BTE). The RDS algorithm is a depth first search of the variable assignment tree with significant optimization by construction tight bounds for the search. The BTE algorithm is a variable elimination algorithm. A WCSP is a type of Valued Constraint Satisfaction Problem (VCSP), where there is a cost associated with each possible assignment of variables within a constraint (a tuple), and the goal is to find a complete assignment with minimum cost. The set of WCSPs that we tested our algorithms on include 6 academic problems, 20 randomly generated problems, which systematically vary in constraint density and tightness, 3 radio link frequency assignment problems and 3 circuit fault analysis problems. In addition to timing the algorithms, we will use platform independent metrics to compare them.

In this paper, we provide a background of the two algorithms being compared, an explanation of the experiments that we ran, the results of those experiments, a description of the time and space complexity metrics used to compare the algorithms, and a discussion of our observations about these results.

Algorithm Descriptions

The Russian Doll Search Algorithm

The RDS algorithm is a depth first search of the variable assignment tree with significant optimization by construction tight bounds for the search. The Russian Doll Search runs on increasingly larger sub-problems in order to improve the lower bound on the global valuation of any partial assignment. The paper [1] addressed constraint optimization and the inefficiency of complete tree search methods. It described the problem that Depth First Branch and Bound has with poor quality lower bounds, even when using Forward Checking (note that Forward Checking is efficient on constraint satisfaction problems).

The simplest way to explore a valued constraint satisfaction problem (VCSP) is to do depth-first or breadth-first search over all of the variables to find the assignment with the best value. Of these two methods, depth-first is clearly better, since it saves space and the whole tree must be searched anyway. However, this method quickly blows up in time complexity since the number of nodes in the tree grows exponentially.

Branch and Bound is an improvement on depth-first search, which allows paths to be discontinued when they are known to be suboptimal. It works by giving a lower bound to the value at each node, and having an upper bound based on the value of the best solution found thus far. At each step the algorithm only ventures further into the tree if this lower bound is better than the upper bound. When a complete solution is found with a better value, it becomes the new incumbent upper bound against which the lower bounds are compared. Thus ignoring portions of the tree, which are quickly found to be bad, saves time.

Unfortunately, Branch and Bound is only as good as its estimate for lower bound. The typical lower bound used for VCSP's is just the sum of the constraints of the variables assigned thus far. This bound is not very tight, as it assumes all future constraints will be met perfectly, and is thus often too optimistic.

Russian Doll Search (RDS) is a special method of using Branch and Bound that replaces one search with multiple searches on successively large sub-problems. It starts with the last variable first, and solves the VCSP for that variable alone. Then it adds the second to last variable and solves that sub-problem, and so forth. The benefit gained is that at any point the optimal solution for each smaller sub-problem is known, which can be used in the calculation for the lower bound.

Effectively the lower bound becomes:

$$lb' = lb + a$$

where lb' is the lower bound in RDS, lb is the normal lower bound, and a is the optimal value for the sub-problem containing all the variables lower in the tree. This gives a tighter lower bound, and can potentially be valuable in speeding up the Branch and Bound algorithm.

Bucket Tree Elimination

The Bucket Tree Elimination (BTE) described in [2], is a decomposition algorithm. The BTE algorithm determines the optimal solution to a VCSP through recursive variable elimination: at each step of the algorithm, a variable is removed from the variable set and the VCSP is reduced to an equivalent problem among the other variables. The assignments of eliminated variables in the optimal solution can be determined by back-tracking the elimination procedure.

Eliminating a variable v and constructing an equivalent problem requires two steps. First, the set of constraints C incident on v is combined into a single constraint on the set of all variables V_C incident on C. Second, the combined constraint is projected onto the reduced variable set $V_C \setminus \{v\}$, selecting an optimal

assignment to v for the valuation of each tuple in the new constraint. An optimal solution to the resulting problem corresponds to an optimal solution to the original problem; moreover, the assignment to v can be recovered from the assignments to $V_C \setminus \{v\}$ using the combined constraint generated in the first step.

The resource requirements of the algorithm are strongly influenced by the order in which variables are eliminated. The number of tuples in a constraint is related exponentially to the number of incident variables, hence we wish to minimize the number of relations added between variables when new constraints are generated. Finding such an optimization is generally NP-hard, therefore, we employ a greedy heuristic method (minimum fill) to generate an approximation to the optimal variable ordering: at each step, we eliminate the variable with the least number of previously independent neighbor pairs in the constraint graph.

Results

This section explains how we tested our implementations on a set of WCSP problem. A WCSP is a type of Valued Constraint Satisfaction Problem (VCSP), where there is a cost associated with each possible assignment of variables associated with a constraint. The goal is to find a complete assignment with minimum cost. The set of (32) WCSPs that we tested our algorithms on include 6 academic problems, 20 randomly generated problems, which systematically vary in constraint density and tightness, 3 radio link frequency assignment problems and 3 circuit fault analysis problems.

The RDS implementation was able to solve 5 of the 32 WCSPs within a 600 second time limit. The BTE algorithm was able to solve 3 of the 32 WCSPs before running out of memory.

The results from the problems that were successfully solved are summarized in Table 1. The problems solved are all academic. Neither algorithm is able to solve the real world problems in a timely manner, which is the ultimate objective.

The RDS algorithm was implemented in C++ and compiled under Microsoft Windows 2000, using the Cygwin / GNU g++ compiler using –O3 optimization level. The computer that the tests were run on has an Intel Xeon 3.06 GHz CPU with 1GB of RAM.

The BTE algorithm was implemented in Wolfram Mathematica using kernel version 5.1.0.0 under Microsoft Windows 2000. Computer tests were run using an Intel Pentium 4 2.20 GHz CPU with 1 GB of RAM. Kernel caches were cleared prior to each benchmark test execution.

Comparison of Metrics

Time Complexity

We chose system independent measures of performance in order to compare the time complexity of our algorithms. However, fundamental differences in the algorithms made it challenging to perform direct comparisons between the two.

In the Russian Doll Search, the main computational activity is the evaluation of constraint costs for given variable assignments. At each node in the assignment tree, the total cost of the constraint set is summed then added to the sub-problem optimal cost in order to establish the lower bound for the cost. The evaluation procedure requires comparing constraint tuples to the variable assignment to find the cost for the constraint. Comparing a tuple to the assignment requires the algorithm to go through each variable in the tuple. Consequently, larger tuples can take a longer time. The metrics used for time complexity were nodes visited, tuples evaluated, and variables evaluated.

In the Bucket Tree Elimination algorithm, the main computational activity is the summation of constraint costs to build cost arrays for combined constraints, and the minimization across eliminated variables to project constraints onto variable subsets. When two constraints are combined, the cost of each tuple in the combined constraint is computed as the sum of the costs of the intersecting tuples in the original constraints. When a constraint is projected onto a subset of variables, the cost of each tuple in the projected constraint is computed as the minimum of the costs of the intersecting tuples in the original constraint. The former operation requires integer additions, the latter integer comparisons. We counted these integer operations independently to be used as metrics for time complexity.

Thus while RDS spends most of its time searching for intersecting tuples in the original constraints to evaluate candidate assignments, BTE stores assignment cost values in random access arrays, and spends most of its time combining and projecting constraints to build these cost arrays. Although both involve integer operations, the fundamentally different character of these operations made direct comparisons difficult. However, the above metrics provide a basis for comparing the performance drivers of the two algorithms.

Verification of Metrics

In order to see how well these metrics measure relate to run time, we plotted program execution time versus each of the metrics, for the two algorithms, and checked the linear fit. We used the "least squares" method to calculate a straight line that best fit the data. We then looked at the R-squared value to gauge the strength of this relationship. This gave us insight into the validity of the metrics.

For the BTE algorithm, the two metrics that were suspected to impact run time were the combination sums, which are integer addition operations of cost values among the constraint tensors, and the projection minimums, which are integer comparison operations. We correlated these two metrics with program execution time. Figure 1 shows a strong correlation (R^2=0.9993) for combination sums, and Figure 2 shows very little correlation (R^2=0.1873) for projection minimums. Therefore it is most reasonable to associate the combination sum operations with the time complexity of the algorithm.

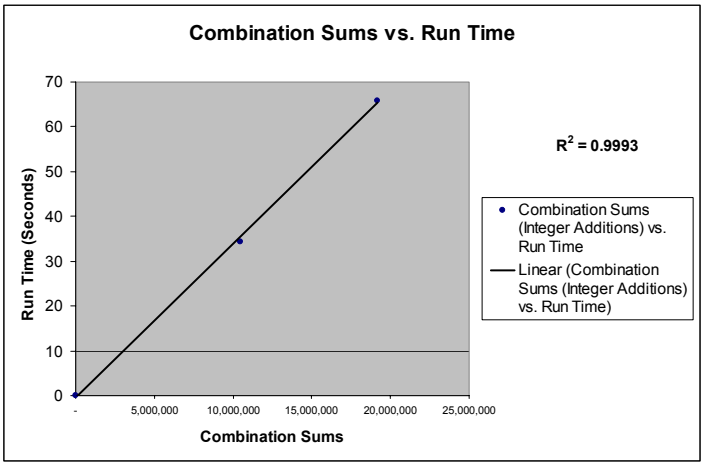


Figure 1. The combination sums metric correlates very well with run time.

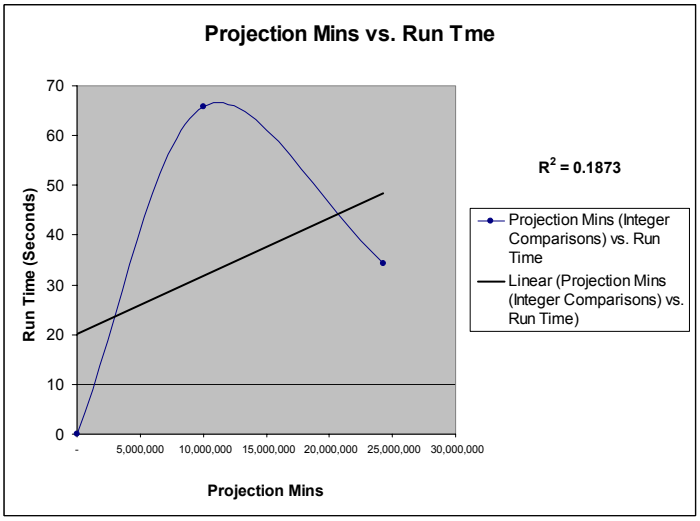


Figure 2. The projection mins metric correlates poorly with run time.

For the RDS algorithm, the simple nodes visited metric correlated the best ($R^2 = 0.99$) of the metrics tested, as shown in Figure 3. Surprisingly, the more complex the metric was, the worse it did. However, all of the metrics had an R^2 above 0.9, which implies that each is a fairly accurate indicator of problem complexity. However, the linear fit is suspect because three of the points are clustered near (0,0), and the other two are clustered very high up.

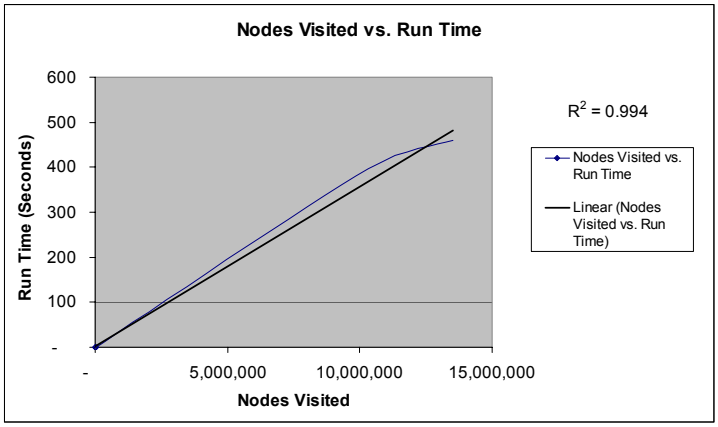


Figure 3a.

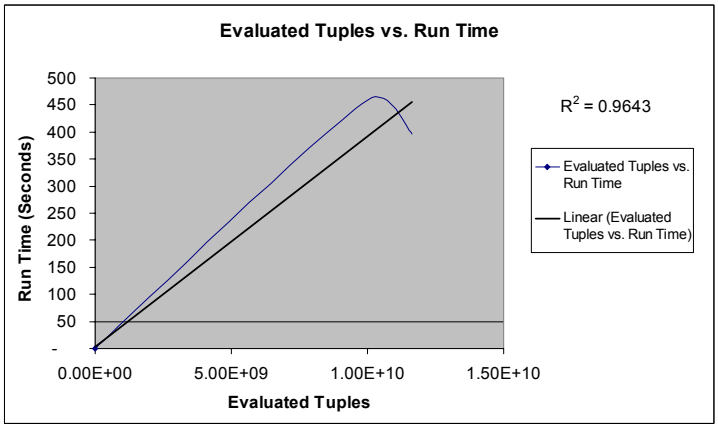


Figure 3b.

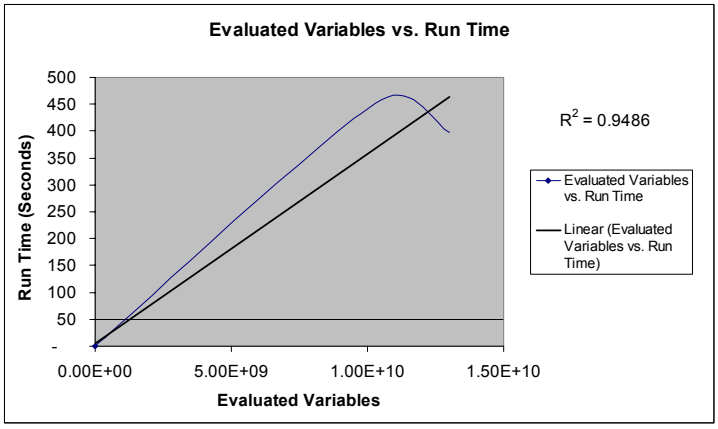


Figure 3c.

Figure 3. Three metrics—nodes visited, evaluated tuples, and evaluated variables—are correlated with run time. Nodes visited shows the best correlation.

Space Complexity

For BTE, the major driver for memory usage is the storage of large constraints generated by recursive combination. Thus space complexity is measured as the maximum number of valuations stored by all constraints at any given step. Figure 4 shows the space complexity of the BTE algorithm as generated by the three benchmark problems that successfully ran. The regression with zero y-intercept is linear, with R^2=0.93. This seems to indicate a linear space complexity, O(kn) with $k \approx 8\times10^4$. Theoretically, the worst case space complexity for BTE should be exponential. Due to the limited data that we have, the space complexity analysis is by no means conclusive. If we had the memory capability on our machine to run more problems, we can complete a more meaningful space complexity analysis.

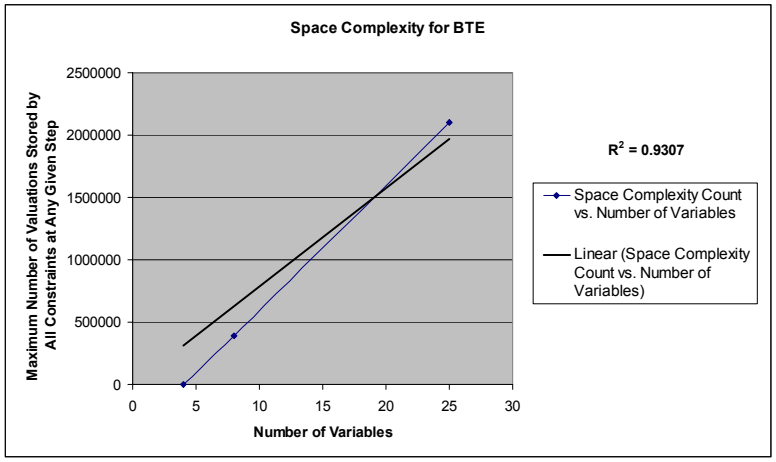


Figure 4. The space complexity for BTE is not very conclusive for the three problems that successfully ran.

Memory is not an issue for the RDS algorithm because it is based on a depth-first search, and only stores the current assignment. The RDS algorithm implementation only keeps 1 copy of each assignment variable in memory at a time, with the exception of temporary variables. Consequently, memory use is linear (O(n)) to the number of problem variables. This is a search algorithm rather than a variable elimination algorithm. As such, only 1 copy of the constraints is kept in memory at a time. Therefore, memory use is also linear to the number of tuples and variables in the constraints.

Comparison of Results

Neither algorithm was able to solve the majority of the problems. The BTE algorithm ran out of memory for all problems except three. The RDS algorithm took longer than ten minutes to reach a solution for all but five problems.

The results from the problems that were successfully solved are summarized in Table 1.

Table 1. Summary of results from BTE and RDS.

	Bucket Tree Elimination				Russian Doll Search			
Problem	Absolute Run Time (Seconds)	Combination Sums (Integer Additions)	Projection Mins (Integer Comparisons)	Space Complexity Count	Absolute Run Time (Seconds)	Nodes Visited	Evaluated Tuples	Evaluated Variables
4 Queens	0.0178	340	516	173	0.0004	16	497	593
8 Queens	65.88	19173960	10025650	392150	0.025	3984	415893	458106
16 Queens	-	-	-	-	459	13531728	9.995E+09	1.05E+10
Zebra	34.36	10438905	24309768	2098552	0.5	35720	438218	6057780
Send	-	-	-	-	397	10340430	1.162E+10	1.3E+10

Run times are shown in wall clock time; however, these comparisons offer limited insight. The two algorithms were run on different machines; moreover, BTE simulations were run using the *Mathematica* kernel on interpreted symbolic expressions, while RDS was compiled from C++ using machine arithmetic. These differences may easily account for the few orders of magnitude observed between the respective results.

System-independent metrics may be roughly compared between the two algorithms since both measure machine integer operations. The number of evaluated tuples in RDS effectively counts individual integer comparisons, as does the projection mins metric in BTE; the combination sums metric in BTE effectively counts integer additions. Both metrics show comparable orders of magnitude to evaluated tuples in RDS for the 4 queens and zebra problems, though the results diverge substantially for 8 queens, as discussed earlier. Essentially, the RDS algorithm is able to effectively exploit the sub-problem bounds in the highly connected queens problems, compared to the BTE algorithm. This asset becomes more pronounced as the problem size grows. The culmination of this is the RDS's ability to solve the 16 queens problem, while the BTE algorithm was unable to do so.

Relative running times provide more insight into algorithm performance and asymptotic complexity. The BTE algorithm required relatively longer run time on the 8 queens problem, as expected given the high connectivity of its constraint graph. For this problem, the algorithm must generate very large cost arrays during the initial stages of the decomposition. Conversely, the RDS algorithm did relatively well on the highly connected queens problems, but not as well (in terms of both running time and platform independent metrics) on the less connected zebra problem.

BTE completed fewer benchmark problems with available resources due to substantially greater memory requirements. While partially attributable to implementation, the BTE algorithm requires fundamentally greater memory allocations to store large combined constraints, particularly in highly connected problems. More efficient techniques for storage and operations with cost arrays may reduce the redundancy inherent in the current implementation and realize significant performance gains.

In general, it would appear that RDS exhibits faster running times and significantly lower memory requirements. Time comparisons are not necessarily valid due to differences in processor speed and implementation. RDS was able to solve a larger number of problems, due to space complexity issues in BTE. Since RDS had very low space complexity, it could have conceivably completed more or all of the problems with a large amount of time. However the time complexity seemed to grow rapidly with problem difficulty, so this was probably not feasible.

References:

[1] G. Verfaillie, M. Lemaitre, and T. Schiex. "Russian Doll Search for Solving Constraint Optimization Problems," Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96), pages 181--187, Portland, OR, USA, 1996.

[2] Kask K, Dechter R, Larrosa J. "Unifying Cluster-Tree Decompositions for Automated Reasoning," University of California at Irvine Technical Report, 2003.

Chapter 2

Russian Doll Search Algorithm Analysis

Lawrence Bush, Brian Bairstow

Introduction

In this paper we discuss our implementation and testing of the Russian Doll Search (RDS) algorithm. This is a Branch and Bound (depth first search with bounds) algorithm, which uses the RDS to compute tighter bounds. We examine the performance of the algorithm on a set of weighted constraint satisfaction problems (WCSP). A WCSP is a type of Valued Constraint Satisfaction Problem (VCSP), where there is a cost associated with each possible assignment of variables associated with a constraint. The goal is to find a complete assignment with minimum cost. The set of WCSPs that we tested our algorithms on include 6 academic problems, 20 randomly generated problems, which systematically vary in constraint density and tightness, 3 radio link frequency assignment problems and 3 circuit fault analysis problems.

In addition to timing the algorithms, we used platform independent metrics which allowed us to compare this algorithm to the Bucket Tree Elimination (BTE) algorithm in Chapter 1.

Chapter 2 Outline

The Russian Doll Search Algorithm

Algorithm Description

- **Theory**
- **Method**
- **Pedagogical Example**

Implementation

- **Code Structure**
- **Sample Runs**
- **Optimizations**

Results

Discussion of Results

- **Metrics**

Observations

- **Strengths/Weaknesses**
- **Limitations of Bounding**
- **Possible Extensions and Improvements**

Appendix RDS-A: Contributions

Team 3: Lawrence Bush, Brian Bairstow

Appendix RDS-B: Sample Runs Walk Through (RDS)

Appendix RDS-C: Code Print Out

The Russian Doll Search Algorithm

Algorithm Description:

Background

The following paper provided the basic algorithm we used in our study:

G. Verfaillie, M. Lemaitre, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96), pages 181--187, Portland, OR, USA, 1996. http://citeseer.ist.psu.edu/verfaillie96russian.html

The paper addressed constraint optimization and the inefficiency of complete tree search methods. It described the problem that Depth First Branch and Bound has with poor quality lower bounds, even when using Forward Checking (note that Forward Checking is efficient on constraint satisfaction problems). The paper introduced Russian Doll Search, which runs on increasingly larger sub-problems in order to improve the lower bound on the global valuation of any partial assignment. They found that the algorithm yielded surprisingly good results, and we were interested in verifying those findings.

Theory

The simplest way to explore a valued constraint satisfaction problem (VCSP) is to do depth-first or breadth-first search over all of the variables to find the assignment with the best value. Of these two methods, depth-first is clearly better, since it saves space and the whole tree must be searched anyway. However, this method quickly blows up in time complexity since the number of nodes in the tree grows exponentially.

Branch and Bound is an improvement on depth-first search, which allows paths to be discontinued when they are known to be suboptimal. It works by giving a lower bound to the value at each node, and having an upper bound based on the value of the best solution found thus far. At each step the algorithm only ventures further into the tree if this lower bound is better than the upper bound. When a complete solution is found with a better value, it becomes the new incumbent upper bound against which the lower bounds are compared. Thus ignoring portions of the tree, which are quickly found to be bad, saves time.

Unfortunately, Branch and Bound is only as good as its estimate for lower bound. The typical lower bound used for WCSP's is just the sum of the constraints of the variables assigned thus far. This bound is not very tight, as it assumes all future constraints will be met perfectly, and is thus often too optimistic.

Russian Doll Search (RDS) is a special method of using Branch and Bound that replaces one search with multiple searches on successively large sub-problems. It starts with the last variable first, and solves the VCSP for that variable alone. Then it adds the second to last variable and solves that sub-problem, and so forth. The benefit gained is that at any point the optimal solution for each smaller sub-problem is known, which can be used in the calculation for the lower bound.

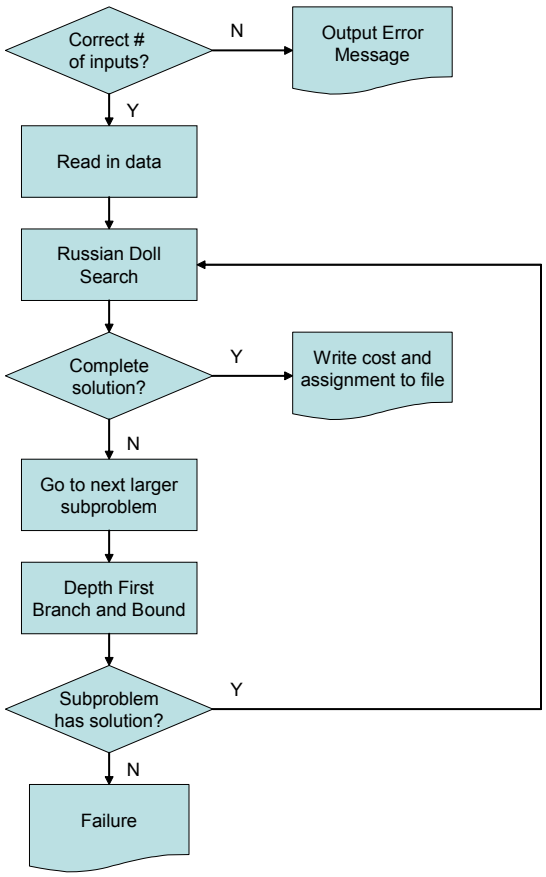
Effectively the lower bound becomes:

$$lb' = lb + a$$

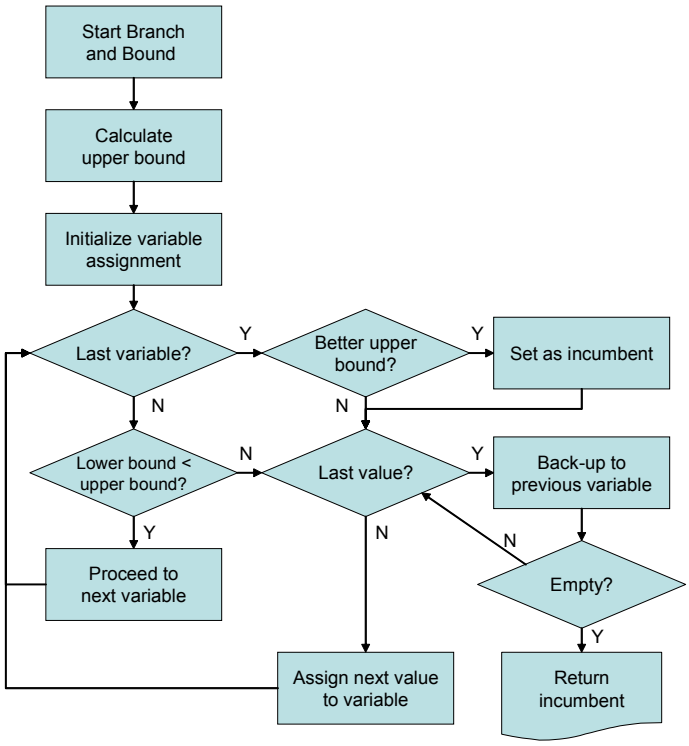
where lb' is the lower bound in RDS, lb is the normal lower bound, and a is the optimal value for the sub-problem containing all the variables lower in the tree. This gives a tighter lower bound, and can potentially be valuable in speeding up the Branch and Bound algorithm.

Method

This section explains the logical flow of the RDS algorithm. The program starts by reading in the WCSP file and extracting the data about constraints, variables, domain values, and so on. It then begins the RDS algorithm with an empty set of variable assignments. It starts by calling Branch and Bound with the smallest sub-problem, and works up to the full problem. Each time RDS calls Branch and Bound, it passes it the optimal assignment of the previous sub-problem and the optimal costs for all of the sub-problems done thus far. In numerical terms, it works from sub-problem n-1 to sub-problem 0. When it has a complete assignment, it outputs the assignment and the corresponding value. If any sub-problem returns a value that is worse than the global upper bound, then it returns failure. This process is shown in the flowchart below.



The Branch and Bound algorithm is complicated, and is, therefore, illustrated in a separate flowchart below:



This is the Branch and Bound algorithm for a sub-problem i. The Branch and Bound function gets the optimal assignment to the previous sub-problem (i+1) and the optimal costs for all completed sub-problems (i+1 to n) as inputs. It calculates the initial upper bound by evaluating all the values for the newest variable combined with the optimal variable assignment of the previous sub-problem. For example, if the previous sub-problem had assignments [0 3 2], and the newest variable has domain size 4, then the new upper bound will be min([0 0 3 2], [1 0 3 2], [2 0 3 2], [3 0 3 2], global upper bound). This is a slightly better method than assuming the worst cases for all of the new constraints, because it can give a tighter upper bound and sometimes find an answer with cost 0.

Then the algorithm initializes the variable assignment. The variable assignment is a list of variable objects, which each contain data for variable number and domain value. The variable assignment is set to contain a single variable with number i and value 0.

The algorithm then checks if all the variables (from i to n-1) have been assigned. If so, then the assignment is complete so it evaluates it to compare to the incumbent. If the value is better, then it is set as the new incumbent. Then it proceeds to the “Last value?” function.

If the variables were not all assigned, then the program compares the current estimate for lower bound (using the optimal value for future sub-problems) to the incumbent upper bound. If the lower bound is better, then the algorithm moves down the tree to the next variable by adding the variable to the assignment list with domain value 0, and then returns to the ”Last variable?” step. If the lower bound is not better, then there is no point in moving down the tree, so it goes to the “Last value?” step.

The “Last value?” function checks if the domain value of the last variable is at its maximum yet. If the value is not at its maximum, then the algorithm increments the domain value of the last variable and returns to the “Last

variable?" step. If it is the last value, then it "backs up" to the previous variable by removing the last variable in the list. If this makes the list empty, then the tree has been explored, and it returns the incumbent as the answer. If the list still has variables in it, then the algorithm returns to the "Last value?" step in order to continue exploring the tree.

In this manner the algorithm moves through the tree. It moves down when the lower bound is promising. When it has a complete assignment it considers setting a new incumbent. When it does not want to move down in the tree, it moves to the right, and when it cannot move to the right it returns back up the tree. If it moves up past the top of the tree, then it is done.

Short Example

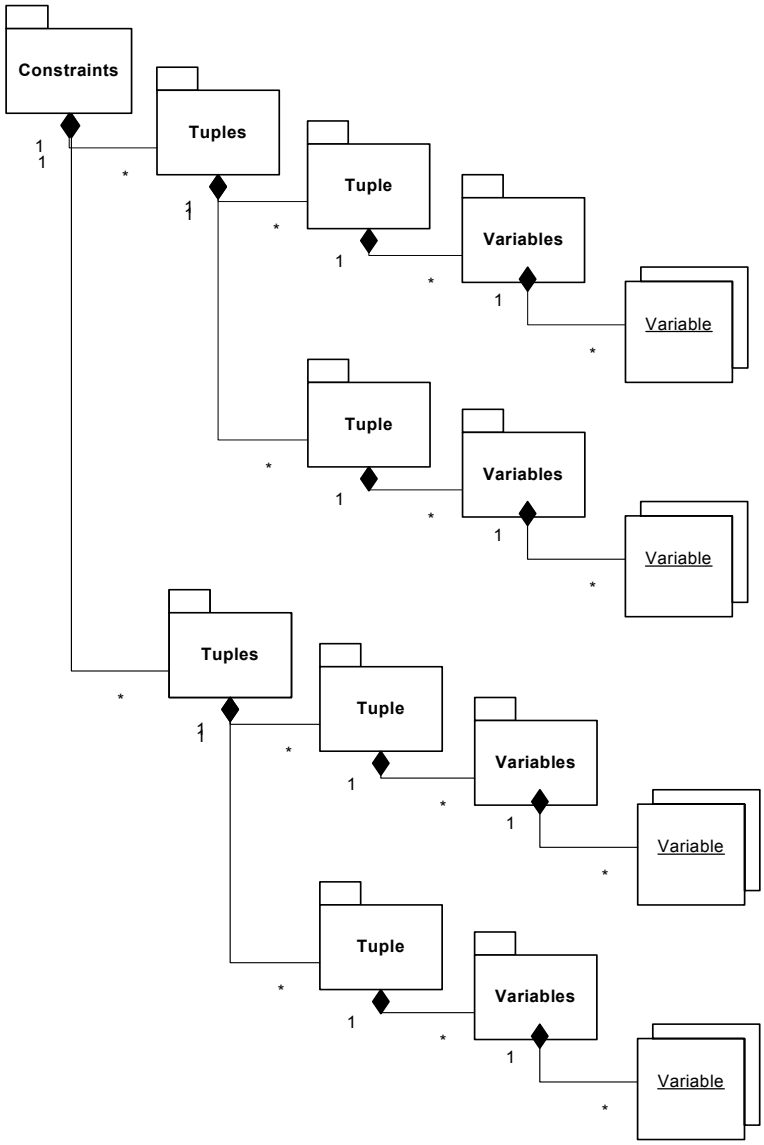
In this section, we will explain how the algorithm would solve a very simple problem, called the "2 queens problem." This problem is how to put 2 queens onto a 2 by 2 chess board such that neither can take each other. To do this, they must not be diagonal, across or down from each other. Each piece is represented as variable 0 and 1, which indicates its, column. The row of each piece is represented as a domain value (0 or 1).

- The RDS starts with the sub-problem of only variable 1.
- Branch and Bound initializes upper bound to the global upper bound (5), and initializes the assignment to [(variable 1, value 0)].
- This is a complete assignment, so it compares it to the global upper bound and finds that it is better, so the new upper bound is 0, and the corresponding assignment is [(variable 1, value 0)].
- Then it moves to the right to [(variable 1, value 1)] and evaluates it, but it is not better than 0, so the incumbent does not change. Since 1 is the last value for variable 1, it backs up to [ ], which is empty so it returns the incumbent. Note that if it was not a full assignment, it still would not progress down the tree because the lower bound = 0 is not better than the upper bound = 0.
- Then RDS starts the sub-problem with variables 0 and 1, which is the full problem.
- Branch and Bound initializes the upper bound to min([0,0],[1,0],global upper bound) = 5, and initializes the assignment to [(variable 0, value 0)].
- This is not a complete assignment, and lower bound = 0 < upper bound = 5, so it moves to the next variable.
- The assignment is now [(variable 0, value 0),(variable 1, value 0)] which is complete, but the value is 5, which does not beat the incumbent. It then increments the value of variable 1 to value 1, so the assignment is [(variable 0, value 0),(variable 1, value 1)]. This still does not beat the incumbent, and value 1 is the maximum domain value for variable 1, so it backs up by removing variable 1 from the list, and incrementing variable 0, so the assignment is [(variable 0, value 1)].
- Once again, lower bound = 0 < upper bound = 5, so it adds variable 1 again to get [(variable 0, value 1),(variable 1, value 0)]. The value of this assignment is 5, which it does not beat the incumbent. It then increments the value of variable 1 to value 1, so the assignment is [(variable 0, value 1),(variable 1, value 1)]. This still does not beat the incumbent, and the value of variable cannot be incremented, so it backs up by removing variable 1 from the list. Now the value of variable 0 cannot be incremented, so variable 0 is removed from the list. At this point the assignment list is empty, so the incumbent is returned.
- Since the incumbent has upper bound greater than or equal to the global upper bound, RDS returns a failure, and the CSP is unsolvable.

Implementation

Code Structure:

This section describes the code structure. We implemented the RDS algorithm in C++ which is an object oriented language. Our implementation included 5 classes: constraints, tuples, tuple, variables and variable. The constraints class holds all of the problem constraints and executes functions on these constraints. The constraints are made up of a set of tuple sets. The class "tuples" is a set of tuples that make up 1 sub-constraint. An object of type "tuples" contains multiple objects of type "tuple." These each in turn contain a set of variables that is contained in a class of type "variables." This class hierarchy is shown below.



The most significant member functions of the constraints class are get_next_value(), get_next_variable() and back_up() which enable traversal of the assignment tree. The following table shows the class interface description:

constraints
-number_of_variables : int -maximum_domain_size : int -number_of_constraint_groups : int -global_upper_bound : double -upper_bound : double -global_lower_bound : double -X : variables -tuples_vector : vector<tuples> -current_assignment : variables -next_variable : int -next_value : int
+constraints(in number_of_variables_in : int, in maximum_domain_size_in : int, in number_of_constraint_groups_in : int, in global_upper_bound_in : int, in X_in : variables) +insert_tuples(in ts_in : tuples) +get_number_of_constraint_groups() : int +get_number_of_variables() : int +operator [](in k : int) : tuples +size() : int +initialize_upper_bound(inout sa : variables, in ca : variables, inout operations : int) : double +get_upper_bound() : double +get_global_upper_bound() : double +print(inout out : ostream) +is_last_value(in ca_in : variables) : bool +is_last_variable(in ca_in : variables) : bool +initialize_assignment(in initial : int) : variables +increment_first_value(in ca_in : variables) : variables +get_next_value(in ca_in : variables) : variables +get_next_variable(in ca_in : variables) : variables +back_up(in ca_in : variables) : variables +get_next_assignment() : variables +evaluate(in ca_in : variables, inout operations : int, in additional_cost : double, in upper_bound : double) : double +evaluate(in ca_in : variables, inout operations : int) : double +bind_next_assignment() : variables +sort_tuples_by_num_non_default_cost()

The most significant member function of the 'tuples' class is evaluate(), which ascribes a value to the current assignment. It iterates through each tuple object that it contains and calls the evaluate() function on each of these. It combines these using a binary operator. The following table shows the class interface description:

tuples
-constraint_arity : int -default_cost : double -number_of_tuples : int -tuple_vector : vector<tuple> -cX_indecies : vector<int>
+tuples() +tuples(in constraint_arity_in : int, in default_cost_in : int, in number_of_tuples_in : int, in cX_indecies_in : vector<int>) +insert(in t_in : tuple) +remove() +get_constraint_arity() : int +get_default_cost() : double +get_number_of_tuples() : int +get_cX_indecies() : vector<int> +operator [](in k : int) : tuple +size() : int +print(inout out : ostream) +constraintdefined(in ca_in : variables) : bool +evaluate(in ca_in : variables) : double

The most significant member function of the tuple class is also evaluate(). The evaluate function is called by a 'tuples' object during evaluation. The tuple::evaluate() member function determines if the current assignment matches it by iterating through the variables passed into it and comparing them to the tuple variable assignments. The following table shows the class interface description:

tuple
-c_vars : variables -non_default_value : double -constraint_arity : int
+tuple() +tuple(in c_vars_in : variables, in non_default_value_in : int) +get_non_default_value() : double +operator [](in k : int) : variable +size() : int +print(inout out : ostream) +evaluate(in ca_in : variables) : double

The variables and variable class together perform the lower level evaluation operations such as determining if the current variable assignments contain and match the tuple assignments. The following 2 tables show the class interface descriptions:

variables
-variable_list : vector<variable>
+variables() +insert(in v : variable) +remove() +operator [](in k : int) : variable +size() : int +get_variable_list() : vector<variable> +print(inout out : ostream) +isIn(in index : int) : bool +isMatched(in a : variable) : bool +matches(in ca_in : variables) : bool +empty() : bool +back() : variable

variable
-var_index : int -domain_size : int -domain_value : int
+variable() +variable(in var_index_in : int, in domain_size_in : int, in domain_value_in : int) +get_domain_value() : int +get_var_index() : int +get_domain_size() : int +increment_domain_value() : bool +max_domain_value() : bool +next_domain_value() : bool

The program is driven by the 'main' function in the 'rds_driver.cpp' file, which parses in the data and calls the RDS algorithm function.

Sample Runs : Walk Through (RDS)

This section includes the output and explanation of 2 sample RDS runs.

- 4 weighted queens problem
- 8 weighted queens problem

4 weighted queens problem
The following is a sample run for the 4 weighted queens problem.

Branch and Bound never meaningfully runs for this problem because optimal solutions for each sub-problem are found while building upper bounds. Notice that at each step the answer to the sub-problem simply builds on the previous sub-problem assignment.

```
--------------------------------------------------
Initial Variable Assignment:
Variable Index: 0, Domain Size: 4, Domain Value: -1
Variable Index: 1, Domain Size: 4, Domain Value: -1
Variable Index: 2, Domain Size: 4, Domain Value: -1
Variable Index: 3, Domain Size: 4, Domain Value: -1

Sub-problem Optimum:   0
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 4, Domain Value: 1

Sub-problem Optimum:   0
--------------------------------------------------
Print all Variables:
Variable Index: 2, Domain Size: 4, Domain Value: 3
Variable Index: 3, Domain Size: 4, Domain Value: 1

Sub-problem Optimum:   0
--------------------------------------------------
Print all Variables:
Variable Index: 1, Domain Size: 4, Domain Value: 0
Variable Index: 2, Domain Size: 4, Domain Value: 3
Variable Index: 3, Domain Size: 4, Domain Value: 1

Sub-problem Optimum:   0
--------------------------------------------------
Print all Variables:
Variable Index: 0, Domain Size: 4, Domain Value: 2
Variable Index: 1, Domain Size: 4, Domain Value: 0
Variable Index: 2, Domain Size: 4, Domain Value: 3
Variable Index: 3, Domain Size: 4, Domain Value: 1

Final Value:    0
--------------------------------------------------
Final Variable Assignments:
Variable Index: 0, Domain Size: 4, Domain Value: 2
Variable Index: 1, Domain Size: 4, Domain Value: 0
Variable Index: 2, Domain Size: 4, Domain Value: 3
Variable Index: 3, Domain Size: 4, Domain Value: 1
```

8 weighted queens problem

We also describe a sample run for the 8 weighted queens problem. However, due to its length, it is included in Appendix RDS-B.

Optimizations

After implementing the RDS algorithm we attempted to improve upon it. First we will explain 3 optimizations that improved the performance of the algorithm. Then we will discuss a few attempts that show promise, but did not improve the performance of the algorithm.

3 optimizations that improved the algorithm:

Initial upper bound optimization:

The key idea of the RDS algorithm is to compute a tighter lower bound. The paper also indicates that a tighter initial upper bound can be computed. However, the exact way to do this is defined quite loosely.

An initial upper bound is computed for each sub-problem. In order to improve the upper bound computation we evaluate all the values for the newest variable combined with the optimal variable assignment of the previous sub-problem. For example, if the previous sub-problem had assignments [0 3 2], and the newest variable has domain size 4, then the new upper bound will be min([0 0 3 2], [1 0 3 2], [2 0 3 2], [3 0 3 2], global upper bound). This is a slightly better method than assuming the worst cases for all of the new constraints, because it can give a tighter upper bound and sometimes find an answer with cost 0. Note that if this upper bound is equal to the upper bound of the previous sub-problem, then we can immediately return the newly found solution. Otherwise, we proceed with the BNB algorithm using this tighter upper bound.

We found that this optimization cuts the run time of the 8 weighted queens problem by two-thirds.

Evaluate only until we pass the upper bound:

A second optimization that we performed is to only evaluate an assignment until we pass the current upper bound. As we evaluate an assignment, we must go through each tuple set that applies. Once we have exceeded the current upper bound, further evaluation of tuple sets is unnecessary because we already know the variable assignment is suboptimal. Therefore, we pass in the future value (of the previous sub-problem) and the current upper bound. If the cost of the variables assigned so far (from the top of the assignment tree to the current level) plus the future cost exceeds the current upper bound at any point in the evaluation procedure, then we can end the evaluation. The BNB algorithm immediately discards this assignment because it is poorer than the current best-known assignment.

Sort constraints (shortest first):

Our third optimization builds upon our second by sorting the constraints (shortest first). The complete constraint set is made up of a set of tuple sets. These tuple sets have a default value and non-default values. The tuple sets will contain n tuples. If we sort these sets so that the smaller sets (fewer tuples) are first, we will avoid evaluating some of the longer sets. Our second optimization terminates the evaluation when we exceed the upper bound. This strategy is improved further by sorting the tuple sets.

We should also point out that we tried sorting the tuples by their cost range. The cost range is the difference between their maximum and minimum cost. The idea here is that the tuples with a large cost range would cause the evaluation (probabilistically) to exceed the upper bound more quickly. This would cause the evaluation to terminate more quickly. However, the previous tuple sorting strategy proved better.

3 optimizations that show promise, but did not demonstrate improvement:

Sort the variables by a metric:

One optimization strategy that we experimented with is to sort the order of the variables (using a variety of metrics).

We know that variable ordering significantly affects the performance. We proved this by adapting the implementation to allow any variable ordering. In this new implementation we do not use the variable index to keep track of the variable order, since the variable index is independent of the ordering. We then swapped the variable order and ran the algorithm. The algorithm ran much faster one-way versus the other.

To exploit this fact, we implemented a few ordering strategies that seemed promising. One promising idea is to use an ordering that creates a lower bound that is closer to the true optimal solution. Another idea is to put variables at the top of the tree that have a wide value range. This could cause the variables at the top to contribute heavily to the cost and consequently terminate that branch early. Another strategy is to use some connectedness value to put the well-connected variables at the bottom. This could create a higher upper bound. It should also result in less connected variables at the top, which are less likely to combine poorly with the sub-problems. Another misguided strategy is to put the well-connected variables at the top; this could make the sub-problem bound more accurate because it is less related to the next problem. This is akin to the maximum occurrence minimum size clause (MOMS) variable ordering strategy commonly employed in the DPLL algorithm.

None of these strategies worked. Part of the problem is that the newer implementation that does not use the indices has a bit more C++ object related overhead. We believe that a sorting mechanism that is able to clearly separate less connected variable sets into groups would improve the algorithm.

Store the previous variable value and iterate around them:

A second strategy that we experimented with is to keep the previous sub-problem assignment and iterate through the values in round-robin fashion. Our initial implementation re-initializes all values to 0 and then traverses them from 0 to n-1. A novel notion is that the previous assignment has some relationship to the best assignment of the next problem with one additional variable. Therefore, we stored the values and used them as the initial value for the next problem. This strategy seemed to work very well in the lower sub-problems. However, the strategy broke down on the problems that included most or all of the variables. Since that is where most of the work is done, this strategy reaped little or no gain.

Start with a zero upper bound and iterate upward:

A third strategy that we employed is to set the global upper bound to 0 and run RDS. If RDS fails, then set the global upper bound to 1, etcetera, until a solution is found. This idea seems promising, particularly if the solution is near 0. However, if the solution is not near 0, this appears to be a poor strategy. However, this issue can be overcome by starting at ½ the known upper bound. If the algorithm fails, then we increase the upper bound by ¼ the known upper bound. This would result in $O(\log_2(N))$ RDS calls.

However, this is a moot point. The problems that we are searching have so many sub-problems with solutions (assignments) that are grossly sub-optimal. In other words, even if we set the global upper bound to 0, when the optimal solution is 10, we may still search a large portion of the tree because many branches and sub-problems have very low evaluations. This point is explained below in the Limitation of Bounding section.

Results

This section explains how we compiled, ran and timed our implementation on a set of WCSP problem. A WCSP is a type of Valued Constraint Satisfaction Problem (VCSP), where there is a cost associated with each possible assignment of variables associated with a constraint. The goal is to find a complete assignment with minimum cost. The set of (32) WCSPs that we tested our algorithms on include 6 academic problems, 20 randomly generated problems, which systematically vary in constraint density and tightness, 3 radio link frequency assignment problems and 3 circuit fault analysis problems.

The following is a list of the results of the RDS algorithm runs on the benchmark problems:

Problem Name	Number of Variables	Run Time (Seconds)	Evaluated Tuples	Evaluated Variables	Nodes Visited	Optimal Value
4wqueens	4	0.0004	497	593	16	0
8wqueens	8	0.0250	415,893	458,106	3,984	2
16wqueens	16	459.0000	9,994,872,751	10,537,435,275	13,531,728	2
zebra	25	0.5000	4,438,218	6,057,780	35,720	0
send	11	397.0000	11,620,072,119	13,022,229,607	10,340,430	0
donald		-	-	-	-	-
CELAR6-SUB0		-	-	-	-	-
CELAR6-SUB1-24		-	-	-	-	-
CELAR6-SUB2		-	-	-	-	-
ssa0432-003		-	-	-	-	-
ssa2670-130		-	-	-	-	-
ssa2670-141		-	-	-	-	-
vcsp25_10_21_85_1		-	-	-	-	-
vcsp25_10_21_85_2		-	-	-	-	-
vcsp25_10_21_85_3		-	-	-	-	-
vcsp25_10_21_85_4		-	-	-	-	-
vcsp25_10_21_85_5		-	-	-	-	-
vcsp25_10_25_87_1		-	-	-	-	-
vcsp25_10_25_87_2		-	-	-	-	-
vcsp25_10_25_87_3		-	-	-	-	-
vcsp25_10_25_87_4		-	-	-	-	-
vcsp25_10_25_87_5		-	-	-	-	-
vcsp30_10_25_48_1		-	-	-	-	-
vcsp30_10_25_48_2		-	-	-	-	-
vcsp30_10_25_48_3		-	-	-	-	-
vcsp30_10_25_48_4		-	-	-	-	-
vcsp30_10_25_48_5		-	-	-	-	-
vcsp40_10_13_60_1		-	-	-	-	-
vcsp40_10_13_60_2		-	-	-	-	-
vcsp40_10_13_60_3		-	-	-	-	-
vcsp40_10_13_60_4		-	-	-	-	-
vcsp40_10_13_60_5		-	-	-	-	-

The algorithm was implemented in C++ and compiled under Microsoft Windows 2000, using the Cygwin / GNU g++ compiler using –O3 optimization level. The computer that the tests were run on has and Intel Xeon 3.06 GHz CPU with 1GB of RAM.

The runs were timed in 2 different ways, depending on their duration. If a run was over 1 second, then it was we used the time( ) function which returns a type 'time_t' which is accurate to the second. If a run was under 1 second, then we used the 'timer.h' class provided by David Musser and referenced in his book "STL Tutorial and Reference Guide, Second Edition: C++ Programming with the Standard Template Library," Addison-Wesley, Reading, MA, 2001. This class runs the algorithm N times on the problem until the time exceeds 1 second. It then computes the time as 1 / N. This way, we were able to see a difference in the easier problems.

Discussion of Results

Our implementation was able to solve only 5 of the 32 WCSPs within a 600 second time limit. While this does not seem very good, it is useful to consider the number of nodes that would exist in each problem's assignment tree. If the algorithm were not bounded, it would have to traverse all of these nodes in order to find the optimal (complete) solution. The first 3 of these problems (weighted queens) are very similar. The differences are how they are weighted and the number of queens. It is interesting to note that as the problem gets more complex, the bounding mechanism becomes more valuable. With that said, the problems shown below are all academic. The algorithm is still not able to solve the real world problems in a timely manner, which is the ultimate objective.

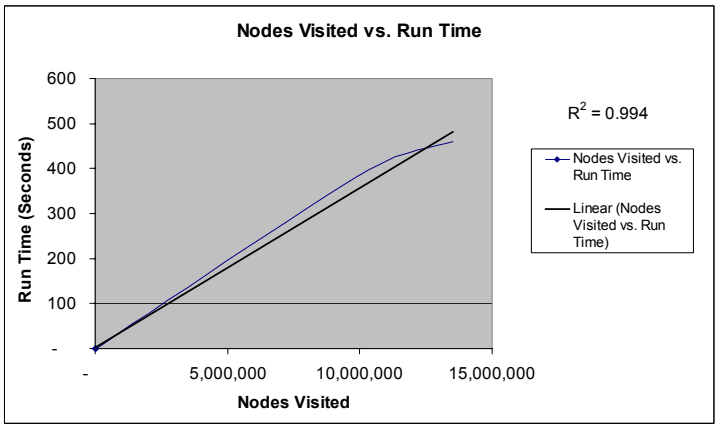
Problem Name	Number of Variables	Run Time (Seconds)	Total Nodes	Nodes per Second
4wqueens	4	0.0004	256	613,096
8wqueens	8	0.0250	16,777,216	671,102,062
16wqueens	16	459.0000	18,446,744,073,709,600,000	40,188,984,910,042,600
zebra	25	0.5000	167,814,181	335,629,033
send	11	397.0000	648,000,000	1,632,242

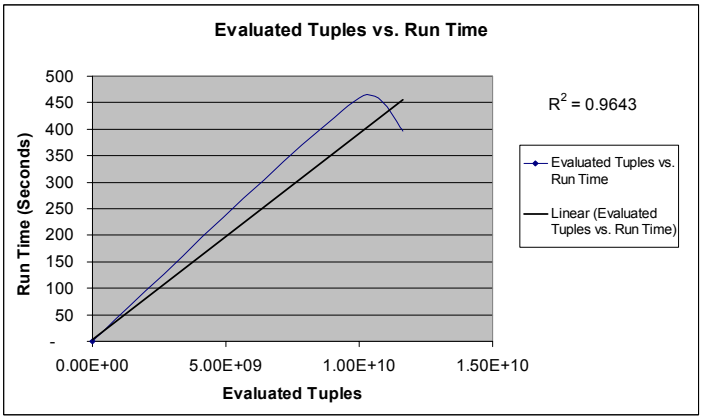
Metrics:

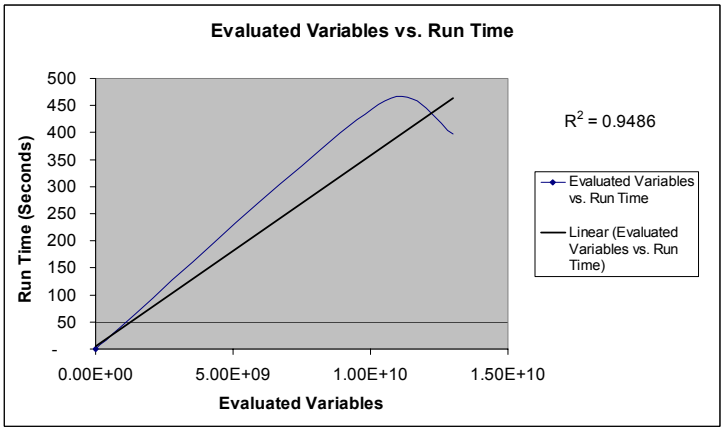
Memory was not an issue for our algorithm. The RDS algorithm implementation only keeps 1 copy of each variable in memory at a time, with the exception of temporary variables. Therefore, memory use is linear to the number of variables. This is a search algorithm rather than a variable elimination algorithm. Therefore, only 1 copy of the constraints is kept in memory at a time. Therefore, memory use is also linear to the number of tuples and variables in the constraints.

The time complexity was our main concern. The algorithm searches a tree, so a larger tree will take longer to search. This suggests number of nodes as a metric. The main time consumer in our algorithm is evaluating the constraints. Constraint evaluation requires comparing constraint tuples to our variable assignment to find the cost for the constraint. Comparing a tuple to our assignment requires us to go through each variable in the tuple; so larger tuples can take a longer time. We implemented three system-independent metrics to track time complexity. One is simply the number of nodes visited in the search tree. The second is the number of tuples evaluated. The third is the number of variables evaluated.

To get some insights about the validity of the metrics (how these metrics measure relate to run time), we plotted program execution time versus each of the metrics, for the two algorithms, and checked the linear fit. We used the "least squares" method to calculate a straight line that best fit the data. We then looked at the R-squared value to gauge the strength of this relationship. These plots are below.







The simple nodes visited metric did the best with R^2 = 0.99. Surprisingly, the more complex the metric was, the worse it did. Each metric had an R^2 above 0.9, which implies that each metric is fairly accurate. However, the linear fit might be suspect because three of the points are clustered near (0,0), and the other two are clustered very high up.

Observations:

Strengths/Weaknesses

One of the strengths of Russian Doll Search is that it has very low space complexity since it is a depth-first search. Most of the memory usage comes from reading in the constraint problem data, and after that the problem is linear (O(n)) with the number of variables.

The Russian Doll Search seems to run very quickly on loosely constrained problems, because it quickly reaches the bottom and gets a strong upper bound to make the rest of the search run quickly. However, in difficult problems the time blows up at some point. The sub-problems go very quickly until a sub-problem is reached in which it is difficult to find a new incumbent. Thus in its worst case, each iteration runs like depth-first search and is O(n^2).

Limitations of Bounding

The RDS does not work well when the optimal solution is X when there are N variables assigned, but there are many variable assignments that valuate to lower than X when fewer than N variables are assigned. For example, suppose we have an assignment tree of depth N, where every node in the first N-1 levels evaluates to 0, but every node at depth N evaluates to 1, with the exception of the right most node. In this case, we would have to traverse every node in the tree, regardless of how well we set our upper and lower bound. This is an extreme case, however, it clearly shows the limitations of the RDS algorithm.

Possible Extensions and Improvements

The RDS algorithm could be extended to find all solutions instead of just one. It would not be hard to implement, but it would reduce the effectiveness of Branch and Bound because it would have to search when lb <= ub, not just when lb < ub.

The program could also be extended to find the next-best solution. To find the next-best solution up until the point the best answer is found, you just need to have a second incumbent that stores the previous incumbent when the incumbent changes. However, to be complete in finding the second-best solution the tree would basically need to be searched again while ignoring all the best solutions.

Summary

In this paper we discussed our implementation and testing of the Russian Doll Search (RDS) algorithm. This included details regarding the algorithm, implementation, timing and platform independent metrics. The algorithm performed will on academic problems. While this is a big improvement over the Branch and Bound algorithm, it is not sufficient to solve real world problems. With that said, we look forward to comparing our results to the Decomposition and Dynamic Programming (DDP) algorithm.

Appendix RDS-A: Contributions

Team 3: Lawrence Bush, Brian Bairstow

Both team members made significant contributions to the project. Where one team member may have written the code, the other team member often helped clarifying the algorithm, optimizing, and error checking the code. Both team members also contributed significantly to the write-up.

Lawrence Bush:

Code:
- Wrote initial code framework.
- Wrote the I/O code.
- Wrote initial Branch and Bound Code.
- Optimized the C++ implementation.
- Optimized the algorithm logic.
- Ran the tests.

Paper:
- Wrote the framework of the paper.
- Wrote optimization discussion.
- Wrote C++ implementation description.
- Wrote the Optimizations section.
- Wrote the results and results discussion section.
- Created the metrics graphs.
- Contributed to the observations section.
- Contributed to writing, editing, formatting and concept review.
- Wrote the introduction and summary.

Brian Bairstow:

Code:
- Wrote initial Russian Doll Search Code.
- Implemented the evaluation functionality.

Paper:
- Wrote the Russian Doll Search description.
- Contributed to the observations section.
- Contributed to writing, editing and concept review.
- Wrote the walk through description.
- Annotated the sample runs/walk through.
- Wrote the metrics section.
- Contributed to the observations section.

Appendix RDS-B: Sample Run : Walk Through (RDS)

This section includes the output and explanation of 1 sample RDS run.

- 8 weighted queens problem

8 weighted queens problem

The following is a sample run for the 8 weighted queens problem.

For all the sub-problems shown, the future sub-problems have optimal cost 0. Thus you can notice that the "Evaluates to:" and "Current lb:" lines are always equal. If the future cost was x, the "Current lb:" line would be x larger than the "Evaluates to:" line.

Sub-problem 5 is fully annotated below; the other sub-problems have brief notes.
The outputs of Sub-problems 0 and 1 have been cut from here because those take up 800 pages.

//comments are marked with double slashes

```
Final Assignment:

Value:    2
---------------------------------------------------
Print all Variables:
Variable Index: 0, Domain Size: 8, Domain Value: 1
Variable Index: 1, Domain Size: 8, Domain Value: 4
Variable Index: 2, Domain Size: 8, Domain Value: 6
Variable Index: 3, Domain Size: 8, Domain Value: 3
Variable Index: 4, Domain Size: 8, Domain Value: 0
Variable Index: 5, Domain Size: 8, Domain Value: 7
Variable Index: 6, Domain Size: 8, Domain Value: 5
Variable Index: 7, Domain Size: 8, Domain Value: 2


Sample Run:

---------------------------------------------------
Print all Variables:
Variable Index: 0, Domain Size: 8, Domain Value: -1
Variable Index: 1, Domain Size: 8, Domain Value: -1
Variable Index: 2, Domain Size: 8, Domain Value: -1
Variable Index: 3, Domain Size: 8, Domain Value: -1
Variable Index: 4, Domain Size: 8, Domain Value: -1
Variable Index: 5, Domain Size: 8, Domain Value: -1
Variable Index: 6, Domain Size: 8, Domain Value: -1
Variable Index: 7, Domain Size: 8, Domain Value: -1
Sub-problem Optimum:   0
---------------------------------------------------
Print all Variables:
Variable Index: 7, Domain Size: 8, Domain Value: 1

//Above is the solution to sub-problem 7.
//A solution for sub-problem 7 is found immediately when
//finding the upper bound, so Branch and Bound is not
//actually run.

Sub-problem Optimum:   0
---------------------------------------------------
Print all Variables:
Variable Index: 6, Domain Size: 8, Domain Value: 6
Variable Index: 7, Domain Size: 8, Domain Value: 1

//Above is the solution to sub-problem 6.
```

```
//A solution for sub-problem 6 is found immediately when
//finding the upper bound, so Branch and Bound is not
//actually run.

//In-depth example of solving sub-problem 5
//Start at variable 5, value 0
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 0
Evaluates to:    1
Current lb:      1
Current ub:      1
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 1
Evaluates to:    1
Current lb:      1
Current ub:      1
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 2
Evaluates to:    1
Current lb:      1
Current ub:      1
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 3
Evaluates to:    1
Current lb:      1
Current ub:      1
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 4
Evaluates to:    1
Current lb:      1
Current ub:      1
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Evaluates to:    0
Current lb:      0
Current ub:      1
-----------------------------------------------------
//lower bound is less than upper bound, so move to next variable
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 0
Evaluates to:    1
Current lb:      1
Current ub:      1
-----------------------------------------------------
//lower bound is not lower than upper, bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 1
Evaluates to:    1
Current lb:      1
Current ub:      1
```

```
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 2
Evaluates to:    1
Current lb:      1
Current ub:      1
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 3
Evaluates to:    1
Current lb:      1
Current ub:      1
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 4
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 5
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 6
Evaluates to:    9
Current lb:      9
Current ub:      1
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Evaluates to:    0
Current lb:      0
Current ub:      1
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 0
Evaluates to:    1
Current lb:      1
Current ub:      1
--------------------------------------------------
//lower bound is lower than upper bound, so move to next variable
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
```

```
Variable Index: 7, Domain Size: 8, Domain Value: 1
Evaluates to:    0
Current lb:      0
Current ub:      1
-----------------------------------------------------
//complete assignment, and lower bound is less than upper bound,
//so new incumbent
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 2
Evaluates to:    0
Current lb:      0
Current ub:      0
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 3
Evaluates to:    1
Current lb:      1
Current ub:      0
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 4
Evaluates to:    1
Current lb:      1
Current ub:      0
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 5
Evaluates to:    1
Current lb:      1
Current ub:      0
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 6
Evaluates to:    1
Current lb:      1
Current ub:      0
-----------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
-----------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 7
Evaluates to:    1
Current lb:      1
Current ub:      0
-----------------------------------------------------
//variables 6 and 7 are both at the maximum domain value, so back up
//twice
-----------------------------------------------------
Print all Variables:
```

```
Variable Index: 5, Domain Size: 8, Domain Value: 6
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
//lower bound is not lower than upper bound, so move to next value
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 7
Evaluates to:    0
Current lb:      0
Current ub:      0
--------------------------------------------------
//variable 5 is at the maximum domain value, so back up.
--------------------------------------------------
Print all Variables:
//now the variable list is empty, so the sub-problem is complete
//and it returns the following answer:
Sub-problem Optimum:   0
--------------------------------------------------
Print all Variables:
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 1

//Above is the solution to sub-problem 5.
//This optimal cost and variable assignment is used in future
//sub-problems.

Sub-problem Optimum:   0
--------------------------------------------------
Print all Variables:
Variable Index: 4, Domain Size: 8, Domain Value: 0
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 1

//Above is the solution to sub-problem 4.
//A solution for sub-problem 4 is found immediately when
//finding the upper bound, so Branch and Bound is not
//actually run

--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 0
Evaluates to:    1
Current lb:      1
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Evaluates to:    0
Current lb:      0
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 0
Evaluates to:    9
Current lb:      9
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 1
Evaluates to:    10
Current lb:      10
```

```
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 2
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Evaluates to:    0
Current lb:      0
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 0
Evaluates to:    1
Current lb:      1
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 1
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 2
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 3
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 4
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Evaluates to:    0
Current lb:      0
```

```
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 0
Evaluates to:    1
Current lb:      1
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 1
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 2
Evaluates to:    1
Current lb:      1
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 3
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 4
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 5
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 6
Evaluates to:    9
Current lb:      9
```

```
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Evaluates to:    0
Current lb:      0
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 0
Evaluates to:    10
Current lb:      10
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 1
Evaluates to:    9
Current lb:      9
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 2
Evaluates to:    0
Current lb:      0
Current ub:      1
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 3
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 4
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
```

```
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 5
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 6
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 7
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 6
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 7
Evaluates to:    0
Current lb:      0
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 4
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 5
Evaluates to:    1
Current lb:      1
Current ub:      0
--------------------------------------------------
--------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 6
Evaluates to:    1
Current lb:      1
```

```
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 7
Evaluates to:    1
Current lb:      1
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 2
Evaluates to:    1
Current lb:      1
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 3
Evaluates to:    0
Current lb:      0
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 4
Evaluates to:    1
Current lb:      1
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 5
Evaluates to:    1
Current lb:      1
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 6
Evaluates to:    1
Current lb:      1
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 7
Evaluates to:    1
Current lb:      1
Current ub:      0
-------------------------------------------------
-------------------------------------------------
Print all Variables:
Sub-problem Optimum:   0
-------------------------------------------------
Print all Variables:
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 2

Sub-problem Optimum:   0
-------------------------------------------------
Print all Variables:
Variable Index: 2, Domain Size: 8, Domain Value: 4
Variable Index: 3, Domain Size: 8, Domain Value: 1
Variable Index: 4, Domain Size: 8, Domain Value: 3
Variable Index: 5, Domain Size: 8, Domain Value: 5
```

```
Variable Index: 6, Domain Size: 8, Domain Value: 7
Variable Index: 7, Domain Size: 8, Domain Value: 2
```
//Above is the solution to sub-problem 3.

//Sub-problems 0 and 1 take up 800 pages, so they have
//been cut out.

Final Solution:

```
Value:    2
--------------------------------------------------
Print all Variables:
Variable Index: 0, Domain Size: 8, Domain Value: 1
Variable Index: 1, Domain Size: 8, Domain Value: 4
Variable Index: 2, Domain Size: 8, Domain Value: 6
Variable Index: 3, Domain Size: 8, Domain Value: 3
Variable Index: 4, Domain Size: 8, Domain Value: 0
Variable Index: 5, Domain Size: 8, Domain Value: 7
Variable Index: 6, Domain Size: 8, Domain Value: 5
Variable Index: 7, Domain Size: 8, Domain Value: 2
```

Appendix RDS-C: Code Print Out

Instructions:

The algorithm was implemented in C++ and compiled under Microsoft Windows 2000, using the Cygwin / GNU g++ compiler using -O3 optimization level.
The computer that the tests were run on has and Intel Xeon 3.06 GHz CPU with 1GB of RAM.

The program can be compiled and run using the included Makefile.

To run the program, put this folder in a place that you can navigate to through Cygwin.
Run Cygwin.
Navigate to the folder (rdsv17o2).
Type make.

This will compile the program and run all of the benchmark examples.

```
#Larry Bush and Brian Bairstow
#Russian Doll Search Make File
#16.412J/6.834J COGNITIVE ROBOTICS
#Prof. Brian Williams

runtests:
    g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
    ./rds_driver.exe ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/4wqueens.wcsp ./out/4wqueens.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/8wqueens.wcsp ./out/8wqueens.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out
    .txt
    ./rds_driver.exe ./benchmarks/send.wcsp ./out/send.wcsp_out.txt ./out/collective_out.
    txt
    ./rds_driver.exe ./benchmarks/16wqueens.wcsp ./out/16wqueens.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB0.wcsp ./out/CELAR6-SUB0.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB1-24.wcsp ./out/CELAR6-SUB1-24.wcsp_out.txt ./
    out/collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB2.wcsp ./out/CELAR6-SUB2.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/donald.wcsp ./out/donald.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa0432-003.wcsp ./out/ssa0432-003.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa2670-130.wcsp ./out/ssa2670-130.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa2670-141.wcsp ./out/ssa2670-141.wcsp_out.txt ./out/
    collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_1.wcsp ./out/vcsp25_10_21_85_1.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_2.wcsp ./out/vcsp25_10_21_85_2.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_3.wcsp ./out/vcsp25_10_21_85_3.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_4.wcsp ./out/vcsp25_10_21_85_4.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_5.wcsp ./out/vcsp25_10_21_85_5.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_1.wcsp ./out/vcsp25_10_25_87_1.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_2.wcsp ./out/vcsp25_10_25_87_2.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_3.wcsp ./out/vcsp25_10_25_87_3.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_4.wcsp ./out/vcsp25_10_25_87_4.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_5.wcsp ./out/vcsp25_10_25_87_5.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_1.wcsp ./out/vcsp30_10_25_48_1.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_2.wcsp ./out/vcsp30_10_25_48_2.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_3.wcsp ./out/vcsp30_10_25_48_3.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_4.wcsp ./out/vcsp30_10_25_48_4.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_5.wcsp ./out/vcsp30_10_25_48_5.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_1.wcsp ./out/vcsp40_10_13_60_1.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_2.wcsp ./out/vcsp40_10_13_60_2.wcsp_out.
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_3.wcsp ./out/vcsp40_10_13_60_3.wcsp_out.
```

```
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_4.wcsp ./out/vcsp40_10_13_60_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_5.wcsp ./out/vcsp40_10_13_60_5.wcsp_out.↙
	txt ./out/collective_out.txt


rds_compile_and_run:
	g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
	./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out↙
	.txt

clear:
	./rds_driver.exe ./out/collective_out.txt

test:
	g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
	./rds_driver.exe ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/4wqueens.wcsp ./out/4wqueens.wcsp_out.txt ./out/        ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/8wqueens.wcsp ./out/8wqueens.wcsp_out.txt ./out/        ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out↙
	.txt


runtests2:
	g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
	./rds_driver.exe ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/send.wcsp ./out/send.wcsp_out.txt ./out/collective_out. ↙
	txt
	./rds_driver.exe ./benchmarks/4wqueens.wcsp ./out/4wqueens.wcsp_out.txt ./out/        ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/8wqueens.wcsp ./out/8wqueens.wcsp_out.txt ./out/        ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out↙
	.txt
	./rds_driver.exe ./benchmarks/16wqueens.wcsp ./out/16wqueens.wcsp_out.txt ./out/      ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/CELAR6-SUB0.wcsp ./out/CELAR6-SUB0.wcsp_out.txt ./out/  ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/CELAR6-SUB1-24.wcsp ./out/CELAR6-SUB1-24.wcsp_out.txt ./↙
	out/collective_out.txt
	./rds_driver.exe ./benchmarks/CELAR6-SUB2.wcsp ./out/CELAR6-SUB2.wcsp_out.txt ./out/  ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/donald.wcsp ./out/donald.wcsp_out.txt ./out/            ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/send.wcsp ./out/send.wcsp_out.txt ./out/collective_out. ↙
	txt
	./rds_driver.exe ./benchmarks/ssa0432-003.wcsp ./out/ssa0432-003.wcsp_out.txt ./out/  ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/ssa2670-130.wcsp ./out/ssa2670-130.wcsp_out.txt ./out/  ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/ssa2670-141.wcsp ./out/ssa2670-141.wcsp_out.txt ./out/  ↙
	collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_21_85_1.wcsp ./out/vcsp25_10_21_85_1.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_21_85_2.wcsp ./out/vcsp25_10_21_85_2.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_21_85_3.wcsp ./out/vcsp25_10_21_85_3.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_21_85_4.wcsp ./out/vcsp25_10_21_85_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_21_85_5.wcsp ./out/vcsp25_10_21_85_5.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_1.wcsp ./out/vcsp25_10_25_87_1.wcsp_out.↙
	txt ./out/collective_out.txt
```

```
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_2.wcsp ./out/vcsp25_10_25_87_2.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_3.wcsp ./out/vcsp25_10_25_87_3.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_4.wcsp ./out/vcsp25_10_25_87_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_5.wcsp ./out/vcsp25_10_25_87_5.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_1.wcsp ./out/vcsp30_10_25_48_1.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_2.wcsp ./out/vcsp30_10_25_48_2.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_3.wcsp ./out/vcsp30_10_25_48_3.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_4.wcsp ./out/vcsp30_10_25_48_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_5.wcsp ./out/vcsp30_10_25_48_5.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_1.wcsp ./out/vcsp40_10_13_60_1.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_2.wcsp ./out/vcsp40_10_13_60_2.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_3.wcsp ./out/vcsp40_10_13_60_3.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_4.wcsp ./out/vcsp40_10_13_60_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_5.wcsp ./out/vcsp40_10_13_60_5.wcsp_out.↙
	txt ./out/collective_out.txt
```

```
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//
// _____________________________________________
//
// _____________________________________________
//  Russian Doll Search - Main program.

#include<iostream>
#include<iomanip>
#include<string>
#include<fstream>
#include<vector>
#include<string>
#include<sstream>
#include "variable.h"
#include "variables.h"
#include "tuple.h"
#include "tuples.h"
#include "constraints.h"

//uncomment the next line to see the full walk through output
//#define DEBUG_OUTPUT_PROBLEM_FILE
//timer code
//#include "long_timer.h"
#include "timer.h"


using namespace std;


variables bnb(constraints C, std::ostream & out, int initial, vector<double> & future_cost
    , variables sa, vector<unsigned long long int> & operations, time_t & time_limit,int &
     variable_counter) {


    if(time_limit < time(0)) { return variables(); }

    variables ca = C.initialize_assignment(initial);

    //  double old_sa_eval = C.evaluate( sa,operations );// the following line is quicker,
     but this one would work also
    double old_sa_eval = future_cost[initial+1];
    C.initialize_upper_bound(sa, ca, operations);
    double new_sa_eval = C.evaluate(sa,operations );
    if(new_sa_eval==old_sa_eval) {
        future_cost[initial] = new_sa_eval;
        return sa;
    }

    double ub=C.get_upper_bound();
    double lb=0;
    double eval;

    while(1) {
        if(time_limit < time(0)) { return variables(); }

#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        ca.print(out);
```

```
#endif

        if(ca.empty()){
            future_cost[initial] = ub;
            return sa;
        }
        eval = C.evaluate(ca,operations, future_cost[initial+ca.size()], ub  );
        //      eval = C.evaluate(ca,operations);// the above line is faster, but this one
     would work also
        lb = eval + future_cost[initial+ca.size()];
#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        out << "Evaluates to:    " << eval << endl;
        out << "Current lb:      " << lb << endl;
        out << "Current ub:      " << ub << endl;
        out << "-----------------------------------------------------"<<endl;
#endif

        // terminal case
        // change the ub if applicable
        // even if it is ok, go to the right/up
        if(C.is_last_variable(ca)){// last variable

            // if last variable, and eval < ub, then set ub = eval (ub is current best
    assignment cost)
            if(lb<ub){
                ub=lb;
                sa = ca;
            }

            if(C.is_last_value(ca)){// last value
                ca = C.back_up(ca);
                operations[2]+=1;

            } else {// not last value
                // if last variable and if not last value, try next value
                ca = C.get_next_value(ca);
                operations[2]+=1;
            }

            ////////////////////////////////////////
        } else {// not last variable yet

            // if eval >= ub && not last value, then try next value or go up(if last
    value)
            // if eval < ub, then try next variable

            if(lb<ub){//good, then try next variable
                ca = C.get_next_variable(ca);
                operations[2]+=1;
            } else {// if eval >= ub && not last value, then try next value or go up(if
    last value)
                if(C.is_last_value(ca)){// last value
                    ca = C.back_up(ca);
                    operations[2]+=1;
                } else {// not last value // not last variable
                    //  if not last value, try next value
                    ca = C.get_next_value(ca);
                    operations[2]+=1;
                }


            }



        }
```

```
    }

}


////////////////////////////////////////////////////////////////////////////////

double rds(constraints C, ostream & out, vector<unsigned long long int> & operations,
    time_t & time_limit,int & variable_counter)
{
    int n = C.get_number_of_variables();
    vector<double> future_costs(n+1, -1);
    variables sa;  //subproblem assignment

    future_costs[n] = 0;

    for(int i = n-1; i >= 0; i--)
    {
        sa = bnb(C, out, i, future_costs, sa, operations, time_limit, variable_counter);
        if(time_limit < time(0)) { return -1; }

#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        out << "Subproblem Optimum:   " << future_costs[i] << endl;
        sa.print(out);
        out << endl ;
#endif

        if(sa.empty()) {
            return -2;
        }

        if(future_costs[i] >= C.get_global_upper_bound())
        {
#ifdef DEBUG_OUTPUT_PROBLEM_FILE
            out << endl << "Failure";
            //if for any subproblem that cost is worse than the global upper bound
            // then it is going to fail on any macro problem, so return flag
#endif
            return -2;//flag
        }
    }
    //#ifdef DEBUG_OUTPUT_PROBLEM_FILE
    out << endl << endl << "Value:    " << future_costs[0] << endl;
    sa.print(out);
    //#endif

    return future_costs[0];

}


////////////////////////////////////////////////////////////////////////////////


// This function converts doubles to strings.

string ltos(unsigned long long int d) {
    ostringstream ost;
    ost<<d;
    return ost.str();
}
string dtos(double d) {
    ostringstream ost;
    ost<<d;
    return ost.str();
}
int main(int argc, char *argv[] ) { //  start of main, with input of file argument and the
```

```
     number of arguments

    vector<unsigned long long int> operations_old;
    vector<unsigned long long int> operations;

    int time_duration = 599;

    if (argc == 2 ) {  //  if there are 2 arguments
        //open out file
        ofstream collective_out;
        collective_out.open(argv[1],ios::out);//write / clear

        if (collective_out == NULL) {// error of out file could not be opened
            cerr << "Error, could not clear the collective outfile" << endl;
            exit(0);
        }

        // 0 is tuple    count
        // 1 is variable count
        // 3 is nodes visited
        collective_out <<"Problem_Name  "<<"  Number of Variables"<<"  Number of Nodes"<< ↙
    "Run_Time(seconds)"<<"  Evaluated Tuple"<<"  Evaluated Variables"<<"  Nodes Visited"< ↙
    <"  Optimal_Value"<<endl;
        collective_out.close();//close output file
        cout<< "Cleared the file : " << argv[1] << endl << endl ;
        cout <<"Problem_Name  "<<"Run_Time(seconds)"<<"  Operation_Counts"<<endl;
        exit(0);

    }

    if (argc != 4 ) {  //  if there are not 4 arguments
        // Error message and command line argument instructions
        cout<< "Command line arguments:\n" << "example:\n" << "problem_in_file_name       ↙
    problem_out_file_name collective_out.txt\n";
        exit(0);
    } else {
        cout<< "There are 4 Command line arguments: "<< argv[0] << " and " << argv[1] << "↙
     and " << argv[2] << " and " << argv[3] << ".\n" << endl;

    }

    //open out file
    ofstream out;
    out.open(argv[2],ios::out);
    if (out == NULL) {// error of out file could not be opened
        cerr << "Error, could not open the file file" << endl;
        exit(0);
    }

    //open out file
    ofstream collective_out;
    collective_out.open(argv[3],ios::app);//append
    if (collective_out == NULL) {// error of out file could not be opened
        cerr << "Error, could not open the collective outfile" << endl;
        exit(0);
    }


    string input_file_name = argv[1];
    cout << "Input Filename = "<<input_file_name<<"\n";

    ifstream in;
    in.open(argv[1],ios::in);
    if (in == NULL) {// error message if in file could not be opened
        cerr << "Error, could not open the output file" << endl;
        exit(0);
    }
```

```
    // Instantiate variables for file input
    string problem_name;
    int number_of_variables;
    int maximum_domain_size;
    int number_of_constraint_groups;
    int global_upper_bound;

    // Read in the problem header:
    in>>problem_name>>number_of_variables>>maximum_domain_size>>
    number_of_constraint_groups>>global_upper_bound;
    // Output the problem header:
    cout<<problem_name<<" "<<number_of_variables<<" "<<maximum_domain_size<<" "<
    <number_of_constraint_groups<<" "<<global_upper_bound<<endl;

    variables X; // X is a set of variables

    // Read in the domain sizes and initialize the variables.
    for(int count_var_index = 0; count_var_index < number_of_variables ; count_var_index+
    +){
        int next_variable_domain_size;
        in>>next_variable_domain_size;
        //      cout<<next_variable_domain_size<<endl;

        int domain_value = -1;

        //  variable next_variable(count_var_index, next_variable_domain_size,
    domain_value);
        X.insert(   variable(count_var_index, next_variable_domain_size, domain_value) );
        //      cout<<next_variable<<endl;

    }

    X.print(out);

    constraints C(number_of_variables, maximum_domain_size,number_of_constraint_groups,
    global_upper_bound, X);

    //  vector<constraint> constraint_vector;

    int constraint_arity;
    int next_variables_in_constraint;
    int default_cost;
    int number_of_tuples;
    int next_tup_value;
    int non_default_value;

    // Read in each constraint group:
    for(int count = 0; count < number_of_constraint_groups ; count++){

        ////////////////////////////////////////////////////////////////////////
        // Read constraint group header line (num vars, each var, default cost, num
    tuples).

        // Read number of variables in the constraint (constraint arity).
        in>>constraint_arity;
        // Read the indecies of variables in the constraint.
        // Note: we have to remember this order, so we can apply the right value
    assingment to the right variable.
        // This should be an iterable set of variables.
        vector<int> cX_indecies; // cX is the subset of variable indecies in the
    constraint.
        for(int count_constraint_arity = 0; count_constraint_arity < constraint_arity ;
    count_constraint_arity++){
            in>>next_variables_in_constraint;
            cX_indecies.push_back(next_variables_in_constraint);
        }
```

```
        // Read the Default cost value.
        in>>default_cost;
        // Read the Number of tuples (variable set assignments) without the default cost.
        in>>number_of_tuples;

        // create the constraint
        //constraint temp_constraint(constraint_arity, default_cost, number_of_tuples,
    cX_indecies);

        // create the tuple set
        tuples temp_tuples(constraint_arity, default_cost, number_of_tuples, cX_indecies);
        //  cout<<number_of_tuples_without_default_cost_value<<endl;
        //
        ////////////////////////////////////////////////////////////////////////////////
    //

        ////////////////////////////////////////////////////////////////////////////////
    //
        // Read in each constraint tuple (variable set assignments without the default
    cost).
        for(int count_number_of_tuples = 0; count_number_of_tuples < number_of_tuples ;
    count_number_of_tuples++){
            variables c_vars; // a set of variables in the sub-constraint

            // Read each the tuple:
            // Here we should iterate through the variable set.
            // Then create a variable / value "pairing" which goes into a value object.
            // The value objects are the combined into a constraint object, which also
    contains the default value.

            for(vector<int>::iterator count_icx = cX_indecies.begin(); count_icx !=
    cX_indecies.end() ; count_icx++ ){
                //          for(int count_number_of_variables_in_constraint= 0;
    count_number_of_variables_in_constraint < number_of_variables_in_constraint ;
    count_number_of_variables_in_constraint++){

                // read the n tuple assignments
                in>>next_tup_value;

                c_vars.insert(  variable(*count_icx, X[*count_icx].get_domain_size(),
    next_tup_value) );
            }

            // read the non-default cost.
            in>>non_default_value;
            // Create a new value.
            tuple temp_tuple = tuple(c_vars,non_default_value);
            temp_tuples.insert(temp_tuple);
        }
        C.insert_tuples(temp_tuples);
    }// end read each constraint group

    C.sort_tuples_by_num_non_default_cost();



    in.close();//close input file

    ////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
    /////                                          /////
    /////               Time Trials                /////
    /////                                          /////
    ////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
    /////         To seconds trials first          /////
    ////////////////////////////////////////////////////
```

```
    srand(time(0));//timer
    time_t initial, final;
    initial = time(0);
    time_t time_limit = initial + time_duration;
    int memory_used = 0;
    int variable_counter = 0;
    //operations=0;
    operations.clear();
    operations.push_back(0);//
    operations.push_back(0);//
    operations.push_back(0);//

    ////////////////////////
    ///     run rds      ///
    ////////////////////////
    double optimal_value = rds(C, out,operations, time_limit, memory_used);
    cout<<"hi"<<endl;
    final = time(0);
    double elapsed_seconds = difftime(final,initial);

    if(elapsed_seconds> 1){

        ////////////////////
        //  output lines  //
        ////////////////////
        // parse the problem name
        string problem_name = argv[1];
        string::size_type ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find(".");
        problem_name.replace(ssl,ssl+4,"");
        ////////////////////////////////////

        string optimal_value_string=dtos(optimal_value);
        if(optimal_value == -1){ optimal_value_string = "time expired"; }
        if(optimal_value == -2){ optimal_value_string = "failure";}



        collective_out <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<    C.    ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos         ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-       ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        cout <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<  C.                ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos         ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-       ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        ////////////////////


    } else {
        ////////////////////////////////////////////////////
        ////////////////////////////////////////////////////
        ////////////////////////////////////////////////////
        // if it is a short time frame, then use this more precise method:
        // timer declarations
        unsigned int reps;
        unsigned long N;
        N=10;
        reps = 10;
        srand(time(0));//timer
```

```
        timer tim;
        // Compute the baseline time for N
        tim.start_baseline(reps);
        do {
            ////////////////////////////////
            // Baseline Operations Here
            // (anything that isn't really the algorithm
            ////////////////////////////////
            // nothing //
            // End of Baseline Operations
            ////////////////////////////////
        } while (tim.check());

        tim.report(false);

        time_t time_limit = time(0) + time_duration;
        tim.start(reps, N);
        do {

            ////////////////////////////////
            // Baseline Operations Here
            // (anything that isn't really the algorithm
            ////////////////////////////////

            // End of Baseline Operations
            ////////////////////////////////
            // Main Timed Operation
            // this is the actual timing
            //      for(int count_rds=1;count_rds<20000;count_rds++) {
            //operations=0;
            // 0 is tuple    count
            // 1 is variable count
            // 3 is nodes visited
            operations.clear();
            operations.push_back(0);//
            operations.push_back(0);//
            operations.push_back(0);//
            double optimal_value = rds(C, out,operations, time_limit, variable_counter);
            ////////////////////////////////////
        }
        while (tim.check());

        tim.report(false);

        cout<<"longer that 3 seconds"<<endl;
        elapsed_seconds = tim.report();
        tim.report(false);

        ////////////////////
        //  output lines  //
        ////////////////////
        string problem_name = argv[1];
        string::size_type ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find(".");
        problem_name.replace(ssl,ssl+4,"");

        string elapsed_seconds_string=dtos(elapsed_seconds);
        string optimal_value_string=dtos(optimal_value);
        if(optimal_value== -1){ optimal_value_string = "time expired";}
        if(optimal_value== -2){ optimal_value_string = "time expired";}

        collective_out <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<    C.    ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos         ↙
```

```
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) <
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        cout <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<  C.
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) <
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;

        //sa.print(collective_out);//you can print out the final assignment again here if
    you wish.
        ////////////////////


    }

    out.close();//close input file

    return 0;

}
```

```
//  File:  constraints.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//
// ________________________________________
//
//

#ifndef _constraints_h_
#define _constraints_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
#include "variables.h"
#include "tuples.h"
using namespace std;

// constraints class
//
//
class constraints {


public:
    //constraints() {}  // default constructor // not needed or wanted
    // Constructor - assigns all variable attributes
    constraints(int number_of_variables_in, int maximum_domain_size_in, int
    number_of_constraint_groups_in, int  global_upper_bound_in, variables  X_in)
        : number_of_variables(number_of_variables_in), maximum_domain_size
    (maximum_domain_size_in),number_of_constraint_groups(number_of_constraint_groups_in),
    global_upper_bound(global_upper_bound_in), X(X_in)
    {
        global_lower_bound = 0;
    }

    void insert_tuples(tuples ts_in){ // insert a variable object into the container
        tuples_vector.push_back(ts_in);
    }


    int get_number_of_constraint_groups()   const {return number_of_constraint_groups;}
    int get_number_of_variables()   const {return number_of_variables;}

    // assessor operator, returns player k
    tuples operator[](int k) const
    {
        return tuples_vector[k];
    }

    int get_number_of_nodes() {
        vector<variable> vl = X.get_variable_list();

        int number_of_nodes=1;
        for(vector<variable>::iterator i = vl.begin(); i != vl.end(); i++)
        {
            number_of_nodes *= i->get_domain_size();// metric counter
        }
        return number_of_nodes;
```

```
    }

    int size() { return tuples_vector.size(); } // returns the number of tuple in the container

    double initialize_upper_bound(variables & sa, variables ca, vector<unsigned long long int> & operations) {

        upper_bound = 999999999;
        double temp;
        variables tempvar;
        int i;

        for(i = 0; i < sa.size(); i++)
        {
            ca.insert(sa[i]);
        }

        for(i = 0; i < ca[0].get_domain_size(); i++)
        {
            temp = evaluate(ca,operations);
            if(temp < upper_bound)
            {
                upper_bound = temp;
                tempvar = ca;
            }
            if(i < ca[0].get_domain_size()-1)
                ca = increment_first_value(ca);
            operations[2]+=1;
        }

        sa = tempvar;
        //if(upper_bound==sa_eval) {upper_bound -= 1;   cout<<"ub == sa_eval"<<endl;} // obsolete not
        return upper_bound;
    }

    double get_upper_bound() {
        return upper_bound;
    }
    double get_global_upper_bound() {
        return global_upper_bound;
    }



    // print tuple container
    void print(std::ostream & out) {

        out << "-----------------------------------------------------"<<endl;
        out << "Tuple Sets Statistics: \n";
        out << "-----------------------------------------------------"<<endl;
        out << "Number of Variables:  "<< number_of_variables << endl;
        out << "Domain Sizes:         "<< maximum_domain_size << endl;
        out << "Number of Tuple Sets: "<< number_of_constraint_groups << endl;
        out << "Global Upper Bound:   "<< global_upper_bound << endl;
        out << "-----------------------------------------------------"<<endl;
        out << "Print all Tuple Sets: \n";
        out << "-----------------------------------------------------"<<endl;

        for( int i = 0 ; i < size() ; i++ ) {
            tuples_vector[i].print(out);
        }
    }

    bool is_last_value(variables & ca_in)
    {
```

```
        if(ca_in.back().get_domain_value()==(ca_in.back().get_domain_size()-1)){
            return true;
        } else {
            return false;
        }
    }


    bool is_last_variable(variables & ca_in)
    {
        if(ca_in.back().get_var_index()==X.back().get_var_index()){
            return true;
        } else {
            return false;
        }
    }

    variables initialize_assignment(int initial ){
        current_assignment = variables();
        next_variable = initial;
        next_value = 0;
        current_assignment.insert(variable(next_variable,X[next_variable].get_domain_size ↙
    (),next_value ));
        return current_assignment;
    }

    variables increment_first_value(variables ca_in)
    {
        int n = ca_in.size();
        vector<variable> temp(n);

        for(int i = n - 1; i > 0; i--)
        {
            temp[i] = ca_in[i];
            ca_in.remove();
        }
        ca_in = get_next_value(ca_in);

        for(int count_n = 1; count_n < n; count_n ++)
        {
            ca_in.insert(temp[count_n]);
        }
        return ca_in;
    }

    variables get_next_value(variables & ca_in){

        variable temp = ca_in.back();
        ca_in.remove();
        if(!temp.increment_domain_value()) {
            cout<<"Error, domain exceeded!\n";
        }
        ca_in.insert(temp);

        return ca_in;

        return ca_in;
    }


    variables get_next_variable(variables ca_in){

        // insert the next variable if I can
        ca_in.insert(variable(X[ca_in.back().get_var_index()+1].get_var_index(),X[ca_in.  ↙
    back().get_var_index()+1].get_domain_size(),0));
        return ca_in;
    }
```

```
variables back_up(variables & ca_in){

    if(ca_in.size()==0){return ca_in;}
    ca_in.remove();
    if(ca_in.size()==0){return ca_in;}

    if(ca_in.back().increment_domain_value()) {
        ca_in = get_next_value(ca_in);
    } else {
        ca_in = back_up(ca_in);
    }
    return ca_in;
}


variables get_next_assignment(){

    // insert the next variable if I can
    if(next_variable+1 != number_of_variables){
        next_variable++;
        current_assignment.insert(variable(next_variable,X[next_variable].
get_domain_size(),0));
        return current_assignment;
    }
    // else increment the value
    while( 1 ){

        variable temp = current_assignment.back();
        current_assignment.remove();
        next_variable--;

        if(!temp.next_domain_value()){
            current_assignment.insert(temp);

            next_variable++;
            return current_assignment;
        }
    }
    return current_assignment;
}


double evaluate(variables & ca_in,vector<unsigned long long int> & operations, double
& additional_cost, double & upper_bound){

    //operations += ca_in.size();// obsolete counter

    double return_value = 0;
    for( int i = 0 ; i < size() ; i++ ) {//size is the number of tuple sets

        return_value += tuples_vector[i].evaluate(ca_in, operations);
        if( (return_value + additional_cost) > upper_bound) { return return_value; }
    }
    return return_value;
}


double evaluate(variables ca_in,vector<unsigned long long int> & operations){

    //operations += ca_in.size();// obsolete counter

    double return_value = 0;
    for( int i = 0 ; i < size() ; i++ ) {
        return_value += tuples_vector[i].evaluate(ca_in, operations);
    }
```

```
        return return_value;
    }

    variables bind_next_assignment(){
        // this function goes to the next value, rather than going deeper
        // increment the value
        while( 1 ){

            variable temp = current_assignment.back();
            current_assignment.remove();
            next_variable--;

            if(!temp.next_domain_value()){
                current_assignment.insert(temp);

                next_variable++;
                return current_assignment;
            }
        }
        return current_assignment;
    }
    // constraint sorts
    void sort_tuples_by_num_non_default_cost(){
        sort(tuples_vector.begin(),tuples_vector.end(),less_size());
    }
    class less_size {
    public:
        bool operator()(tuples x, tuples y) const { return (x.get_number_of_tuples() < y. ↙
    get_number_of_tuples()); }
    };


private:

    int number_of_variables;
    int maximum_domain_size;
    int number_of_constraint_groups;
    double global_upper_bound;
    double upper_bound;
    double global_lower_bound;
    variables X;
    vector<tuples> tuples_vector;


    variables current_assignment;
    int next_variable;
    int next_value;


};

#endif
```

```
//  File:  tuples.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//
// ________________________________________
//
//  tuples.h -  Contains a set of tuples and algorithm pertaining to them.
//              Stores a vector of tuples.
//

#ifndef _tuples_h_
#define _tuples_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
using namespace std;

// tuples class
//
//
class tuples {


public:
    tuples() {}  // default constructor
    // Constructor - assigns all variable attributes
    tuples(int constraint_arity_in, int default_cost_in, int number_of_tuples_in, vector ↙
    <int> cX_indecies_in)
        : constraint_arity(constraint_arity_in), default_cost(default_cost_in),          ↙
    number_of_tuples(number_of_tuples_in), cX_indecies(cX_indecies_in)
    {
    }

    void insert(tuple t_in){ // insert a variable object into the container
        tuple_vector.push_back(t_in);
    }

    //remove a tuple from the container
    void remove(){
        tuple_vector.pop_back();
    }

    int get_constraint_arity()  const {return constraint_arity;}
    double get_default_cost()   const {return default_cost;}
    int get_number_of_tuples()  const {return number_of_tuples;}
    vector<int> get_cX_indecies()   const {return cX_indecies;}

    // assessor operator, returns player k
    tuple operator[](int k) const
    {
        return tuple_vector[k];
    }


    int size() { return tuple_vector.size(); } // returns the number of tuple in the      ↙
    container

    // print tuple container
```

```
    void print(std::ostream & out) {
        out << "-------------------------------------------------------"<<endl;
        out << "-----                Next Tuple Set              -----"<<endl;
        out << "-------------------------------------------------------"<<endl;
        out << "Tuple Set Arity:          "<<constraint_arity<<endl;
        out << "Tuple Set Variables:      ";
        for(vector<int>::iterator iter = cX_indecies.begin();iter != cX_indecies.end();
    iter++){ out << *iter << "  ";}
        out << endl;
        out << "Tuple Set Default Value: "<<default_cost<<endl;
        out << "Tuple Set Size:           "<<number_of_tuples<<endl;

        out << "-------------------------------------------------------"<<endl;

        out << "Print each Tuple in this Tuple Set: \n";
        out << "-------------------------------------------------------"<<endl;
        for( int i = 0 ; i < size() ; i++ ) {
            tuple_vector[i].print(out);
            out << "-------------------------------------------------------"<<endl;
        }
    }

    bool constraintdefined(variables & ca_in)
        //This returns true if all the variables necessary for the constraint have been
    defined.
    {
        for(int i = 0; i < constraint_arity; i++)
        {
            if(!ca_in.isIn(cX_indecies[i]))
                return false;
        }
        return true;
    }


    double evaluate(variables & ca_in, vector<unsigned long long int> & operations)
    {
        double temp;

        if(constraintdefined(ca_in)){


            for( int i = 0 ; i < size() ; i++ )//Size is the number of tuple objects.
            {

                // 0 is tuple    count
                // 1 is variable count
                // 3 is nodes visited
                operations[0] += 1;// for each tuple evaluated
                temp = tuple_vector[i].evaluate(ca_in, operations);
                //operations[1] += tuple_vector[i].size();// count each variable in each
    tuple set evaluated
                if(temp != -1)
                {
                    return temp;
                }
            }
            return default_cost;
        }
        return 0;   //This must be the neutral value
    }

private:
    int constraint_arity;
    double default_cost;
    int number_of_tuples;
    vector<tuple> tuple_vector;
```

```
    vector<int> cX_indecies;
};

#endif
```

```
//  File:  tuple.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//
// ________________________________________
//
//
//  tuple.h -   Contains a tuple class.
//          Stores a tuple read in from file.
//          A tuple is an assignment to a set of variables.
//          More precisely, it is an ordered set of values
//          assigned to the ordered set of variables.
//
//          In this class, there is a set of vaiables that
//          have an assignment.
//

#ifndef _tuple_h_
#define _tuple_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
using namespace std;

// tuple class
//
//
class tuple {


public:
    tuple() {}  // default constructor
    // Constructor - assigns all variable attributes
    tuple( variables c_vars_in, int non_default_value_in )
        : c_vars(c_vars_in), non_default_value(non_default_value_in)
    {
    }




    double get_non_default_value()  const {return non_default_value;}

    // assessor operator, returns player k
    variable operator[](int k) const
    {
        return c_vars[k];
    }


    int size() { return c_vars.size(); } // returns the number of variable in variables

    // print draft list container
    void print(std::ostream & out) {
        out << "Non-Default Tuple Cost:  " << non_default_value << endl;
        out << "-------------------------------------------------"<<endl;
        c_vars.print(out);
```

```
    }


    double evaluate(variables & ca_in, vector<unsigned long long int> & operations){

        if(c_vars.matches(ca_in, operations)){

            return non_default_value;

        } else {
            return -1;
        }
    }


private:
    variables c_vars;
    double non_default_value;
    int constraint_arity;

};



#endif
```

```
//  File:  variables.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//
// ________________________________________
//

//  variables.h -   Contains a variables class.
//          Stores a vector of variables read in from file.
//          Performs functions on the vector.
//          This is a wrapper for a vector.
//          The vector stores the list of all of the variables to pick from.
//

#ifndef _variables_h_
#define _variables_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
#include <iostream>
#include <fstream>
using namespace std;


class variables {


public:
    variables() {}  // default constructor

    void insert(variable v){ // insert a variable object into the container
        variable_list.push_back(v);
    }

    // remove a variable object from the container
    void remove(){
        variable_list.pop_back();
    }

    // assessor operator, returns player k
    variable operator[](int k) const
    {
        return variable_list[k];
    }


    int size() { return variable_list.size(); } // returns the number of variables in the ↙
    container

    vector<variable> get_variable_list() { return variable_list; } // returns the list of ↙
    variables

    // print container
    void print(std::ostream & out) {

        out << "-----------------------------------------------------"<< endl;
        out << "Print all Variables: \n";
        for( int i = 0 ; i < size() ; i++ ) {
```

```
            out  <<"Variable Index: " << variable_list[i].get_var_index() << ", Domain
    Size: " << variable_list[i].get_domain_size() << ", Domain Value: " << variable_list
    [i].get_domain_value() <<  endl;
        }
    }


    bool isIn(int & index)
    {
        for(int i = 0; i < size(); i++)
        {
            if(index == variable_list[i].get_var_index())
                return true;
        }
        return false;
    }

    bool isMatched(variable & a)
    {
        for(vector<variable>::iterator i = variable_list.begin(); i != variable_list.end()
    ; i++)
        {
            if(a.get_var_index() == i->get_var_index() && a.get_domain_value() == i->
    get_domain_value())
                return true;
        }

        return false;
    }


    bool matches(variables & ca_in, vector<unsigned long long int> & operations)
        //This returns true if the variable assignments (ca_in) contains the tuple
    assignments.(c_vars)
    {
        for(vector<variable>::iterator i = variable_list.begin(); i != variable_list.end()
    ; i++)
        {
            // 0 is tuple    count
            // 1 is variable count
            // 2 is nodes visited
            operations[1]+=1;
            if(!ca_in.isMatched(*i)) {

                return false;
            }
        }

        return true;
    }


    bool empty() { return variable_list.empty(); }

    variable back() {
        return variable_list.back();
    }


private:
    vector<variable> variable_list; // private vector data member

};
```

```
#endif
```

```
//
//  File:  variable.h
//
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//
// ________________________________________
//
//  variable.h -    Contains a variable class.
//          variable object stores variable information,
//          and performs formated output and other
//          functions on variable data.

#ifndef _variable_h_
#define _variable_h_
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
//
class variable {

    friend std::ostream& operator << ( std::ostream& outstr,
        const variable& v );

public:
    variable(){};  // default constructor --

    // Constructor - assigns all variable attributes
    variable( int var_index_in, int domain_size_in, int domain_value_in  )
        : var_index(var_index_in),domain_size(domain_size_in), domain_value
    (domain_value_in)
    {

    }

    //*********************************************************
    //*********************************************************
    // Get functions to retrieve a statistic about the variable.
    //*********************************************************
    //*********************************************************

    int get_domain_value()  const {return domain_value;} // returns variable value
    int get_var_index()     const {return var_index;} // returns variable value
    int get_domain_size()   const {return domain_size;} // returns variable value

    bool increment_domain_value()    {
        domain_value++;
        if(domain_value >= domain_size) {
            domain_value--;
            return false; // check for going to far
        } else {
            return true;
        }
    }


    bool max_domain_value() const {return domain_value==domain_size;} // returns variable
    value
```

```
    bool next_domain_value(){
        domain_value++;
        return max_domain_value();
    } // returns variable value


private:
    //*********************************
    //*********************************
    //      Private Data Members
    //*********************************
    //*********************************
    int var_index;
    int domain_size;
    int domain_value;

};

//*********************************************************
// Overloaded non-member << operator
//*********************************************************
std::ostream &
operator << ( std::ostream & outstr, const variable & v )
{
    outstr <<"Variable Index: " << v.get_var_index() << ", Domain Size: " << v.
    get_domain_size() << ", Domain Value: " << v.get_domain_value() <<  endl;
    return outstr;
}


#endif
```

Chapter 3

Bucket Tree Elimination Algorithm Analysis

Thomas Coffee and Shuonan Dong

Introduction

Valued constraint satisfaction problems (VCSPs) generalize traditional "hard" constraint satisfaction problems by assigning weighted costs to variable assignment options within each constraint. Hence so-called "soft" CSPs provide a more flexible approach to real-world problem solving.

Like traditional CSPs, VCSPs are posed in terms of a set of variables with finite allowable domains. However, the constraints in VCSPs are not sets of allowable assignments to variable subsets, but mappings from variable subset assignments to values, often described as "costs." The optimal solution is defined in terms of a combination operator specifying the total valuation of multiple constraints.

Because VCSPs deal with multivariate optimization rather than simple boolean propagation, they generally require greater computational resources for solution than traditional CSPs. A number of algorithmic approaches have been developed to handle this class of problems. In this paper, we describe and analyze a decomposition algorithm known as Bucket Tree Elimination (BTE) described by Kask et al. (2003).

1. The BTE Algorithm

The BTE algorithm determines the optimal solution to a VCSP through recursive variable elimination: at each step of the algorithm, a variable is removed from the variable set and the VCSP is reduced to an equivalent problem among the other variables. The assignments of eliminated variables in the optimal solution can be determined by backtracking the elimination procedure.

Eliminating a variable v and constructing an equivalent problem requires two steps. First, the set of constraints C incident on v is combined into a single constraint on the set of all variables V_C incident on C. Second, the combined constraint is projected onto the reduced variable set $V_C \setminus \{v\}$, selecting an optimal assignment to v for the valuation of each tuple in the new constraint. An optimal solution to the resulting problem corresponds to an optimal solution to the original problem; moreover, the assignment to v can be recovered from the assignments to $V_C \setminus \{v\}$ using the combined constraint generated in the first step.

The resource requirements of the algorithm are strongly influenced by the order in which variables are eliminated. The number of tuples in a constraint is related exponentially to the number of incident variables, hence we wish to minimize the number of relations added between variables when new constraints are generated, as illustrated in Figure 1. Finding such an optimization is generally NP-hard, so we employ a greedy heuristic method to generate an approximation to the optimal variable ordering: at each step, we eliminate the variable with the least number of previously independent neighbor pairs in the constraint graph. This so-called "minimum fill" heuristic has demonstrated considerable success in practice.

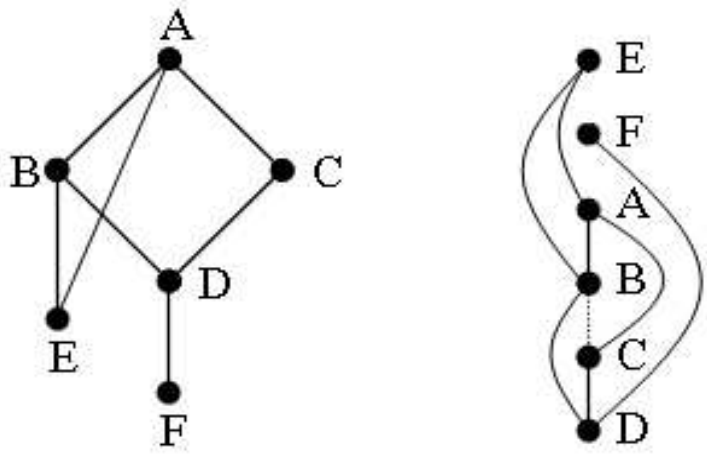


Figure 1: The size of generated constraints depends strongly on the variable elimination order. The image at the right shows the optimal variable elimination order $\{E, F, A, B, C, D\}$ for the constraint graph at left among variables $\{A, B, C, D, E, F\}$. The dotted line $B - - C$ represents a new constraint added by the elimination of variable A, which neighbors both.

The BTE algorithm may be summarized as follows:

Algorithm 1. Bucket Tree Elimination.

(1) Set $i = 1$
(2) Let v_i be the variable in the current VCSP with the least number of independent neighbor pairs
(3) Let C_i be the set of constraints incident on v_i
(4) Combine the constraints in C_i to generate a new constraint c_i among the variables V_{C_i} incident on C_i
(5) Replace the variables V by the reduced set $V \setminus \{v_i\}$
(6) Replace the constraints C_i by the projection of c_i onto the variables $V_{C_i} \setminus \{v_i\}$
(7) If the variable set is nonempty, set $i = i + 1$ and repeat from step (2)
(8) For $j = i, i - 1, \ldots, 2, 1$
assign v_j its optimal value according to the current assignment along with c_j
(9) Return the optimal assignment to the variables v_j

The execution of the algorithm generates large data structures, and will be illustrated by example along with the discussion of implementation in Section 2.

2. Implementation and Examples

We now illustrate the BTE algorithm and our implementation using an instance of the Queens Problem: place n queens on an $n \times n$ chess board such that no two queens threaten one another. The VCSP below describes a soft constraint version of the Queens problem for $n == 4$.

Constraint Representation

Each constraint is represented as a list of k variables incident on the constraint, and a k-dimensional array containing the cost valuations associated with the possible assignments to the variables. The indices of the i^{th} dimension of the array correspond to the domain values of the i^{th} variable. The constraints for the Four Queens Problem are shown in Figure 2.

$$(1) \quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$(2) \quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$(3) \quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$(4) \quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 5 & 0 & 0 & 5 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 5 & 0 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & \mathbf{5} & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

Figure 2: Constraints in the Four Queens Problem are represented by a list of incident variables and a corresponding array of valuations for assignments of the variables. The highlighted value corresponds to the assignment $\{3 \to 2, 4 \to 3\}$, which has a cost valuation of 5 under the constraint incident on $\{3, 4\}$.

Constraint Combination

Constraints on multiple sets of variables are combined using a binary operator that evaluates each tuple in the combined constraint according to the sum of the values of the corresponding tuples in the constituent constraints. Figure 3 shows two of the constraints in the Four Queens Problem and their combination into a new constraint among all incident variables.

$$\begin{pmatrix}1\\3\end{pmatrix} \quad \begin{pmatrix}5&0&\mathbf{5}&0\\0&5&0&5\\5&0&5&0\\0&5&0&5\end{pmatrix}$$

$$\begin{pmatrix}2\\3\end{pmatrix} \quad \begin{pmatrix}5&5&0&0\\5&5&5&0\\0&5&5&5\\0&0&\mathbf{5}&5\end{pmatrix} \longrightarrow \quad \begin{pmatrix}3\\1\\2\end{pmatrix}$$

$$\begin{pmatrix}
\begin{pmatrix}10\\10\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\0\\0\end{pmatrix} & \begin{pmatrix}10\\10\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\0\\0\end{pmatrix} \\
\begin{pmatrix}5\\5\\5\\0\end{pmatrix} & \begin{pmatrix}10\\10\\10\\5\end{pmatrix} & \begin{pmatrix}5\\5\\5\\0\end{pmatrix} & \begin{pmatrix}10\\10\\10\\5\end{pmatrix} \\
\begin{pmatrix}5\\10\\10\\\mathbf{10}\end{pmatrix} & \begin{pmatrix}0\\5\\5\\5\end{pmatrix} & \begin{pmatrix}5\\10\\10\\10\end{pmatrix} & \begin{pmatrix}0\\5\\5\\5\end{pmatrix} \\
\begin{pmatrix}0\\0\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\10\\10\end{pmatrix} & \begin{pmatrix}0\\0\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\10\\10\end{pmatrix}
\end{pmatrix}$$

Figure 3: Two constraints in the Four Queens problem are combined into a single larger constraint by computing the sum of the values of corresponding tuples in the original constraints. The assignment $\{3 \to 3, 1 \to 1, 2 \to 4\}$ highlighted above has a valuation equal to the sum of the valuations of $\{1 \to 1, 3 \to 3\}$ and $\{2 \to 4, 3 \to 3\}$ in the parent constraints.

Constraint Projection

A constraint on a set of variables is projected onto a subset of these variables by assigning to each tuple the minimum valuation among the corresponding tuples in the original constraint. Figure 4 shows the projection of the combined constraint generated above onto a subset of its variables.

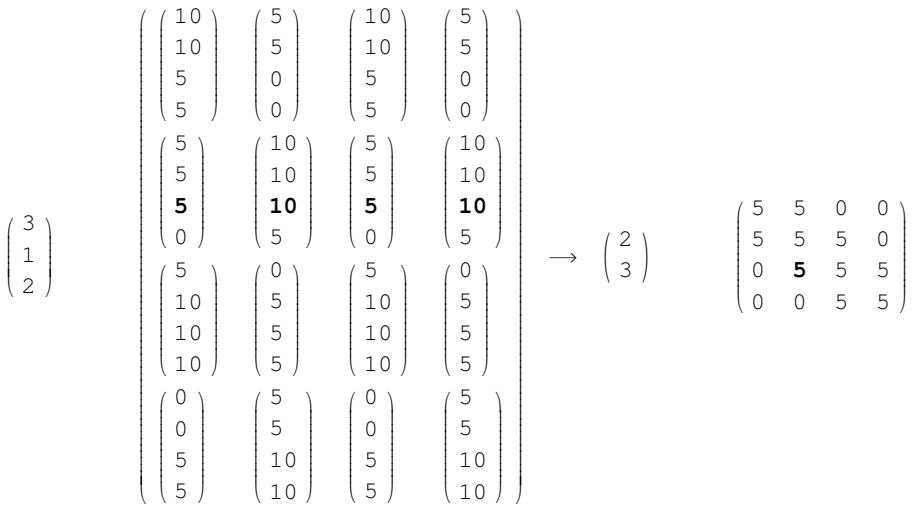


Figure 4: A constraint is projected onto a subset of its variables by finding the minimum value among the corresponding tuples in the original constraint. The valuation of the assignment $\{2 \to 3, 3 \to 2\}$ highlighted above in the projected constraint is the minimum of the valuations of the assignments $\{3 \to 2, 1 \to \_, 2 \to 3\}$ in the parent constraint.

Variable Ordering

Our current implementation of the algorithm performs variable ordering separately using a minimal representation of the constraint graph, though this is not strictly necessary. The Four Queens Problem is strongly connected, and therefore imposes no preferences on variable ordering, though this is not the case in general.

Variable Assignment

After the bucket tree is constructed, the constraints generated by the combination steps are used to propagate the variable assignment backward along the elimination sequence. Figure 5 shows the constraints used to generate the assignment for the Four Queens Problem.

$$
\begin{pmatrix}1\\2\\3\\4\end{pmatrix}
\left(\begin{array}{cccc}
\begin{pmatrix}15&10&10&15\\10&5&5&10\\15&10&10&15\\10&\mathbf{5}&5&10\end{pmatrix} &
\begin{pmatrix}15&10&10&15\\10&5&5&10\\15&10&10&15\\10&5&5&10\end{pmatrix} &
\begin{pmatrix}10&5&5&10\\5&0&0&5\\10&5&5&10\\5&0&0&5\end{pmatrix} &
\begin{pmatrix}10&5&5&10\\5&0&0&5\\10&5&5&10\\5&0&0&5\end{pmatrix}\\
\begin{pmatrix}6&11&6&6\\11&16&11&11\\6&11&6&6\\11&\mathbf{16}&11&11\end{pmatrix} &
\begin{pmatrix}6&11&6&6\\11&16&11&11\\6&11&6&6\\11&16&11&11\end{pmatrix} &
\begin{pmatrix}6&11&6&6\\11&16&11&11\\6&11&6&6\\11&16&11&11\end{pmatrix} &
\begin{pmatrix}1&6&1&1\\6&11&6&6\\1&6&1&1\\6&11&6&6\end{pmatrix}\\
\begin{pmatrix}5&5&10&5\\0&0&5&0\\5&5&10&5\\0&\mathbf{0}&5&0\end{pmatrix} &
\begin{pmatrix}10&10&15&10\\5&5&10&5\\10&10&15&10\\5&5&10&5\end{pmatrix} &
\begin{pmatrix}10&10&15&10\\5&5&10&5\\10&10&15&10\\5&5&10&5\end{pmatrix} &
\begin{pmatrix}10&10&15&10\\5&5&10&5\\10&10&15&10\\5&5&10&5\end{pmatrix}\\
\begin{pmatrix}6&1&1&6\\11&6&6&11\\6&1&1&6\\11&\mathbf{6}&6&11\end{pmatrix} &
\begin{pmatrix}6&1&1&6\\11&6&6&11\\6&1&1&6\\11&6&6&11\end{pmatrix} &
\begin{pmatrix}11&6&6&11\\16&11&11&16\\11&6&6&11\\16&11&11&16\end{pmatrix} &
\begin{pmatrix}11&6&6&11\\16&11&11&16\\11&6&6&11\\16&11&11&16\end{pmatrix}
\end{array}\right)
$$

$$
\begin{pmatrix}2\\3\\4\end{pmatrix}
\left(\begin{array}{cccc}
\begin{pmatrix}15\\6\\11\\10\end{pmatrix} &
\begin{pmatrix}10\\5\\15\\5\end{pmatrix} &
\begin{pmatrix}10\\1\\6\\5\end{pmatrix} &
\begin{pmatrix}5\\\mathbf{0}\\10\\0\end{pmatrix}\\
\begin{pmatrix}12\\12\\7\\17\end{pmatrix} &
\begin{pmatrix}11\\16\\11\\16\end{pmatrix} &
\begin{pmatrix}12\\12\\7\\17\end{pmatrix} &
\begin{pmatrix}6\\\mathbf{11}\\6\\11\end{pmatrix}\\
\begin{pmatrix}12\\6\\11\\7\end{pmatrix} &
\begin{pmatrix}16\\6\\11\\11\end{pmatrix} &
\begin{pmatrix}17\\11\\16\\12\end{pmatrix} &
\begin{pmatrix}16\\\mathbf{6}\\11\\11\end{pmatrix}\\
\begin{pmatrix}1\\10\\1\\6\end{pmatrix} &
\begin{pmatrix}5\\5\\0\\10\end{pmatrix} &
\begin{pmatrix}6\\15\\6\\11\end{pmatrix} &
\begin{pmatrix}10\\\mathbf{10}\\5\\15\end{pmatrix}
\end{array}\right)
$$

$$
\begin{pmatrix}3\\4\end{pmatrix}
\begin{pmatrix}6&\mathbf{11}&1&6\\11&\mathbf{11}&6&6\\7&\mathbf{7}&12&11\\5&\mathbf{0}&10&5\end{pmatrix}
$$

$$
(4)\quad
\begin{pmatrix}\mathbf{6}\\\mathbf{0}\\\mathbf{2}\\\mathbf{5}\end{pmatrix}
$$

Figure 5: The succession of combined constraints shown here is used to assign the variables {1, 2, 3, 4} in reverse order. The assignment is updated beginning from the last constraint, and propagated upward to derive a complete assignment. First, the valuations of {4 → _} by the constraint on {4} yield the partial assignment {4 → 2}. Then the valuations of {3 → _, 4 → 2} by the constraint on {3, 4} yield the partial assignment {3 → 4, 4 → 2}. The process continues to produce the assignment {1 → 3, 2 → 1, 3 → 4, 4 → 2}.

3. Performance Benchmarks

We characterized the performance of the algorithm according to three metrics, two dealing with time complexity (computation) and one with space complexity (memory):

(1) Combination operations: the fundamental operation in the combination function is the summing of valuations on the Cartesian product of intersecting tuples. We counted the number of individual valuation sums performed during the execution.

(2) Projection operations: the fundamental operation in projection is finding the minimum valuation among sets of consistent tuples. We counted the number of individual valuations considered by the minimization function.

(3) Constraint storage: the major driver for memory usage is the storage of large constraints generated by recursive combination. We counted the maximum number of valuations stored by all constraints stored at any given step.

We tested the algorithm on a number of standard benchmark problems, as shown in Table 1. Memory usage was the primary limitation in our current implementation, which we hope to improve in the future by using sparse array methods for constraint storage and processing. We have shown the values of our three performance benchmarks for the examples completed.

Table 1. Results of Benchmark Problem Testing

Problem	*Termination*	*Combination Sums*	*Projection Mins*	*Max Valuations Stored*
zebra	{1 → 1, 2 → 3, 3 → 5, 4 → 4, 5 → 2, 6 → 1, 7 → 5, 8 → 3, 9 → 2, 10 → 4, 11 → 1, 12 → 3, 13 → 2, 14 → 4, 15 → 5, 16 → 5, 17 →2, 18 → 1, 19 → 4, 20 → 3, 21 → 4, 22 → 3, 23 → 5, 24 → 1, 25 → 2}	10,438,905	10,025,650	392,150
4wqueens	{1 → 3, 2 → 1, 3 → 4, 4 → 2}	340	516	173
8wqueens	{1 → 2, 2 → 5, 3 → 7, 4 → 4, 5 → 1, 6 → 8, 7 → 6, 8 → 3}	19,173,960	24,309,768	2,098,552
16wqueens	out of memory	–	–	–
send	out of memory	–	–	–
ssa0432-003	out of memory	–	–	–
vcsp40_10_13_60_1	out of memory	–	–	–
vcsp25_10_21_85_1	out of memory	–	–	–
vcsp25_10_25_87_1	out of memory	–	–	–
vcsp30_10_25_48_1	out of memory	–	–	–

Note: variables and assignments are indexed from one rather than zero in this representation.

4. Results and Discussion

Despite the small number of successful tests, we can draw some worthwhile insights from the above examples. Foremost, we may observe the strong dependence of algorithm performance on the connectedness of the constraint graph: the queens problems exhibit a denser structure, and while zebra involves a much larger variable space than 8wqueens, the latter requires more operations and substantially more memory. The difference in memory usage profiles over the elimination history of the algorithm is shown in Figure 6. Because storage is driven by the maximum number of variables related in a constraint, dense constraint graphs necessarily lead to large combined constraints near the beginning of an execution, whereas sparser constraint graphs tend to result in peaks midway through the decomposition.

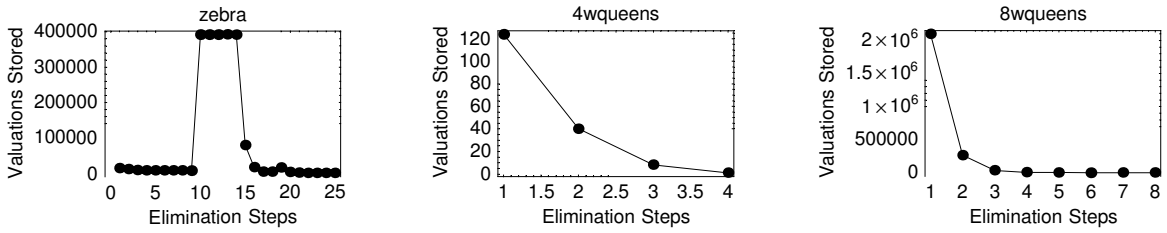


Figure 6: Memory consumption profiles of the three benchmark problems completed reveal the different behaviors corresponding to different constraint graph structures.

The bucket tree elimination method can be easily extended to produce multiple good solutions by taking different branches in the backward propagation, though finding the best suboptimal solutions could require significant additional computational effort. However, the most favorable applications of the algorithm seem to lie in sparse problems with a closely clustered optimal solution set.

We hope to improve our implementation of the BTE algorithm by taking advantage of sparse array data representations, which can be used to exploit the "default cost" attribute common to many tuples within a given constraint. In fact, this was the original motivation for our data representation, but implementation issues left this capability undeveloped as of the time of this writing. With such an implementation, the "sizes" of constraints (that is, number of non-default-valued tuples) would begin to play a significant role in the resources required to execute the algorithm.

5. Contributions and Acknowledgements

The project was largely an integrated effort in design, implementation and analysis. Thomas Coffee led the development of the *Mathematica* implementation of BTE, while Shuonan Dong focused on analysis and documentation of the algorithm and results.

6. References

Kask K, Dechter R, Larrosa J. Unifying Cluster-Tree Decompositions for Automated Reasoning. University of California at Irvine Technical Report, 2003.

Appendix BTE-A: Implementation Code

1. Utilities

```
Off[General::spell1]

Disp[x_] := (Print[x]; x)

Disp[x_, f_] := (Print[f[x]]; x)

DispConstraints[cc_] := Disp[cc, TableForm@Map[MatrixForm, #, {2}] &]

Pairs[x_] := Subsets[x, {2}]

MapGroup[expr_, f_] :=
 Last@Reap[MapThread[Sow, Part[List @@@ {expr, List /@ #}, All, Ordering[#]] &[f /@ expr]]]

MapGroup[expr_, {f_, m_List}] :=
 With[{r = Range@Length[m]}, With[{s = Sequence @@ Flatten[Transpose@{m, r}, 1]},
   Flatten[#, 1] & /@ Last@Reap[Sow[#, List@Switch[f[#], s]] & /@ expr, r]]]

SparseArrayUnion[arrays_] := With[{rules = ArrayRules /@ arrays},
  SparseArray[Join @@ Most /@ rules, Dimensions@First[arrays], Max[Last /@ Last /@ rules]]]
```

2. Example Problem

```
SetDirectory["C:/Documents and Settings/Administrator/Desktop/Documents/aca
   Academic/16412 Cognitive Robotics/aca-16412-ref-ProblemSet02Benchmarks"]
```

```
C:\Documents and Settings\Administrator\Desktop\Documents\aca
  Academic\16412 Cognitive Robotics\aca-16412-ref-ProblemSet02Benchmarks
```

```
p = q = 0; r = {};

data = Import[".\academics\4wqueens.wcsp", "Table"];

nvars = data[[1, 2]]
```

```
4
```

```
domains = data[[2]]
```

```
{4, 4, 4, 4}
```

```
constraints =
  Normal@{First@First[#], SparseArrayUnion[Last /@ #]} & /@ MapGroup[First@Last@Reap[
       Drop[data, 2] //. {{_, v__, d_, l_}, c__} :> (Sow@{{v} + 1, SparseArray[Most[#] + 1 →
                Last[#] & /@ Take[{c}, l], domains[[{v} + 1]], d]}; Drop[{c}, l])], First];

DispConstraints[constraints];
```

$$(1)\quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$(2)\quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$(3)\quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$(4)\quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}\quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix}\quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 4 \end{pmatrix}\quad \begin{pmatrix} 5 & 0 & 0 & 5 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 5 & 0 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix}\quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 4 \end{pmatrix}\quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 4 \end{pmatrix}\quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

3. Algorithm Implementation

```
Hyperedges[cc_] := First /@ cc
```

```
hyperedges = Hyperedges[constraints]
```

```
{{1}, {2}, {3}, {4}, {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}}
```

```
Edges[hh_] := Union @@ Pairs /@ hh
```

```
edges = Edges[hyperedges]
```

```
{{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}}
```

```
Neighbors[n_, ee_] := Flatten /@ Last@
   Reap[Sow @@@ Flatten[{ee, Reverse /@ ee}, 1], Range[n], DeleteCases[Union[#2], #] &]
```

```
neighbors = Neighbors[nvars, edges]
```

```
{{2, 3, 4}, {1, 3, 4}, {1, 2, 4}, {1, 2, 3}}
```

```
FillNodeOrdering[bb_] :=
 Ordering[Length@Select[Pairs[#], ! MemberQ[Part[bb, #], #2] & @@ # &] & /@ bb]
```

```
FillNodeOrdering[neighbors]
```

```
{1, 2, 3, 4}
```

```
NodeOrder[n_, ee_, bb_] := Module[{v, edges = ee, neighbors = bb},
  Flatten@Last@Reap@Do[
      v = Sow@Take[FillNodeOrdering[neighbors], {i}];
      edges = Union[DeleteCases[edges, {v, _} | {_, v}], Pairs[neighbors[[v]]]];
      neighbors = Neighbors[n, edges];
      , {i, n}]
 ]
```

```
NodeOrder[nvars, edges, neighbors]
```

```
{1, 2, 3, 4}
```

```
ProjectConstraint[c_, y_] := Module[{vars, inter, domain, codomain},
  vars = First[c];
  inter = Intersection[vars, y];
  domain = Flatten[Position[vars, #] & /@ inter];
  codomain = Complement[Range@Length[vars], domain];
  {inter, Map[Min@(p += Length[#]; #) &@Flatten[#] &,
    Transpose[Last[c], Ordering@Join[domain, codomain]], {Length[domain]}]}]
```

```
DispConstraints@{constraints[[6]]};
```

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} \qquad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

```
DispConstraints@{ProjectConstraint[constraints[[6]], {1}]};
```

(1)
$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

```
DispConstraints@{ProjectConstraint[constraints[[6]], {3}]};
```

(3)
$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

```
DispConstraints@{ProjectConstraint[{{2, 3, 4},
```

$$\begin{pmatrix} \begin{pmatrix} 5 \\ 1 \\ 1 \\ 5 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 5 \\ 0 \end{pmatrix} & \begin{pmatrix} 5 \\ 1 \\ 1 \\ 5 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 5 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 6 \\ 1 \\ 1 \\ 6 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \end{pmatrix} & \begin{pmatrix} 6 \\ 1 \\ 1 \\ 6 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \end{pmatrix} \\ \begin{pmatrix} 6 \\ 5 \\ 5 \\ 6 \end{pmatrix} & \begin{pmatrix} 5 \\ 0 \\ 0 \\ 5 \end{pmatrix} & \begin{pmatrix} 6 \\ 5 \\ 5 \\ 6 \end{pmatrix} & \begin{pmatrix} 5 \\ 0 \\ 0 \\ 5 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 5 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 5 \\ 0 \\ 0 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 5 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 5 \\ 0 \\ 0 \\ 5 \end{pmatrix} \end{pmatrix}$$

```
}, {3, 4}]};
```

$$\begin{pmatrix} 3 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
CombineConstraints[cc : {_}] := First[cc]
```

```
CombineConstraints[cc : {_, _}] := Module[{vars, inter, indices, coindices},
  vars = First /@ cc;
  inter = Intersection @@ vars;
  indices = Flatten /@ Outer[Position, vars, inter, 1];
  coindices = MapThread[Delete[Range@Length[#], List /@ #2] &, {vars, indices}];
  {Flatten@{inter, MapThread[Part, {vars, coindices}]},
   (q += Times @@ Dimensions[#]; #) & @ MapThread[Function[{x, y}, Map[# + y &, x, {-1}]],
     MapThread[Transpose[Last[#], Ordering@Join[##2]] &,
      {cc, indices, coindices}], Length[inter]]}
 ]
```

```
CombineConstraints[cc : {__}] := Fold[CombineConstraints[{##}] &, First[cc], Rest[cc]]
```

```
DispConstraints[constraints[[{6}]]];
```

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

```
DispConstraints@{CombineConstraints[constraints[[{6}]]]};
```

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

```
DispConstraints[constraints[[{6, 8}]]];
```

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

```
DispConstraints@{CombineConstraints[constraints[[{6, 8}]]]};
```

$$\begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix} 10 \\ 10 \\ 5 \\ 5 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 10 \\ 10 \\ 5 \\ 5 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \\ 0 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 5 \\ 5 \\ 5 \\ 0 \end{pmatrix} & \begin{pmatrix} 10 \\ 10 \\ 10 \\ 5 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \\ 5 \\ 0 \end{pmatrix} & \begin{pmatrix} 10 \\ 10 \\ 10 \\ 5 \end{pmatrix} \\ \begin{pmatrix} 5 \\ 10 \\ 10 \\ 10 \end{pmatrix} & \begin{pmatrix} 0 \\ 5 \\ 5 \\ 5 \end{pmatrix} & \begin{pmatrix} 5 \\ 10 \\ 10 \\ 10 \end{pmatrix} & \begin{pmatrix} 0 \\ 5 \\ 5 \\ 5 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ 5 \\ 5 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \\ 10 \\ 10 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 5 \\ 5 \end{pmatrix} & \begin{pmatrix} 5 \\ 5 \\ 10 \\ 10 \end{pmatrix} \end{pmatrix}$$

```
DispConstraints[constraints[[{3, 6, 8}]]];
```

$$\begin{pmatrix} 3 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

```
DispConstraints@{CombineConstraints[constraints[[{3, 6, 8}]]]};
```

$$\begin{pmatrix}3\\1\\2\end{pmatrix}\quad\left(\begin{array}{cccc}\begin{pmatrix}10\\10\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\0\\0\end{pmatrix} & \begin{pmatrix}10\\10\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\0\\0\end{pmatrix}\\ \begin{pmatrix}6\\6\\6\\1\end{pmatrix} & \begin{pmatrix}11\\11\\11\\6\end{pmatrix} & \begin{pmatrix}6\\6\\6\\1\end{pmatrix} & \begin{pmatrix}11\\11\\11\\6\end{pmatrix}\\ \begin{pmatrix}6\\11\\11\\11\end{pmatrix} & \begin{pmatrix}1\\6\\6\\6\end{pmatrix} & \begin{pmatrix}6\\11\\11\\11\end{pmatrix} & \begin{pmatrix}1\\6\\6\\6\end{pmatrix}\\ \begin{pmatrix}0\\0\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\10\\10\end{pmatrix} & \begin{pmatrix}0\\0\\5\\5\end{pmatrix} & \begin{pmatrix}5\\5\\10\\10\end{pmatrix}\end{array}\right)$$

```
BucketEliminate[n_, cc_] :=
 Module[{constraints = cc, edges, neighbors, order},
  First@Last@Reap[
     vertices = Range[n];
     edges = Edges@Hyperedges[constraints];
     neighbors = Neighbors[n, edges];
     order = Disp@NodeOrder[n, edges, neighbors];
     Fold[
      Function[{c, v}, (AppendTo [r, Total[Times @@@ Dimensions /@ Last /@ #]]; #) &@
        (Append[#, ProjectConstraint[Sow@Disp[CombineConstraints[#2], First],
             vertices = Complement[vertices, {v}]]] &@@ MapGroup[c,
           {MemberQ[First[#], v] &, {False, True}}])], constraints, order];
    ]]
```

```
buckets = BucketEliminate[nvars, constraints]; buckets // ColumnForm
```

```
{1, 2, 3, 4}

{1, 2, 3, 4}

{2, 3, 4}

{3, 4}

{4}

{{1, 2, 3, 4}, {{{{15, 10, 10, 15}, {10, 5, 5, 10}, {15, 10, 10, 15}, {10, 5, 5, 10}}, {{15, 10
{{2, 3, 4}, {{{15, 6, 11, 10}, {10, 5, 15, 5}, {10, 1, 6, 5}, {5, 0, 10, 0}}, {{12, 12, 7, 17},
{{3, 4}, {{6, 11, 1, 6}, {11, 11, 6, 6}, {7, 7, 12, 11}, {5, 0, 10, 5}}}
{{4}, {6, 0, 2, 5}}
```

```
BucketExpand[n_, kk_] := Module[{assign, new},
  assign = Table[All, {n}];
  Scan[Function[c,
    new = First@Select[First[c], assign[[#]] === All &, 1];
    assign[[new]] =
     First@Ordering[Part[Last[c], Sequence @@ Part[assign, First[c]]], 1];
   ], kk];
  MapThread[Rule, {Range[n], assign}]
 ]
```

```
BucketExpand[nvars, Reverse[buckets]]
```

```
{1 → 3, 2 → 1, 3 → 4, 4 → 2}
```

```
SolveWCSP[n_, cc_] := BucketExpand[n, Reverse@BucketEliminate[n, cc]]
```

```
SolveWCSP[path_] :=
 Module[{data = Import[path, "Table"], nvars, constraints, domains},
  p = q = 0; r = {};
  nvars = data[[1, 2]];
  domains = data[[2]];
  constraints = Normal@{First@First[#], SparseArrayUnion[Last /@ #]} & /@
    MapGroup[First@Last@Reap[Drop[data, 2] //. {{_, v__, d_, l_}, c__} :>
          (Sow@{{v} + 1, SparseArray[Most[#] + 1 → Last[#] & /@ Take[{c}, l],
              domains[[{v} + 1]], d]}; Drop[{c}, l])], First];
  DispConstraints[constraints];
  SolveWCSP[nvars, constraints]
 ]
```

4. Algorithm Testing

For each example, the output shows the original constraints, followed by a sequence of variable subsets corresponding to the new combined constraints generated during at each step of the execution.

```
paths = FileNames["*.wcsp", ".", ∞]; paths // ColumnForm
```

```
.\academics\16wqueens.wcsp
.\academics\4wqueens.wcsp
.\academics\8wqueens.wcsp
.\academics\donald.wcsp
.\academics\send.wcsp
.\academics\zebra.wcsp
.\celar\CELAR6-SUB0.wcsp
.\celar\CELAR6-SUB1-24.wcsp
.\celar\CELAR6-SUB2.wcsp
.\dimacs\ssa0432-003.wcsp
.\dimacs\ssa2670-130.wcsp
.\dimacs\ssa2670-141.wcsp
.\random\denseloose\vcsp30_10_25_48_1.wcsp
.\random\denseloose\vcsp30_10_25_48_2.wcsp
.\random\denseloose\vcsp30_10_25_48_3.wcsp
.\random\denseloose\vcsp30_10_25_48_4.wcsp
.\random\denseloose\vcsp30_10_25_48_5.wcsp
.\random\densetight\vcsp25_10_25_87_1.wcsp
.\random\densetight\vcsp25_10_25_87_2.wcsp
.\random\densetight\vcsp25_10_25_87_3.wcsp
.\random\densetight\vcsp25_10_25_87_4.wcsp
.\random\densetight\vcsp25_10_25_87_5.wcsp
.\random\sparseloose\vcsp40_10_13_60_1.wcsp
.\random\sparseloose\vcsp40_10_13_60_2.wcsp
.\random\sparseloose\vcsp40_10_13_60_3.wcsp
.\random\sparseloose\vcsp40_10_13_60_4.wcsp
.\random\sparseloose\vcsp40_10_13_60_5.wcsp
.\random\sparsetight\vcsp25_10_21_85_1.wcsp
.\random\sparsetight\vcsp25_10_21_85_2.wcsp
.\random\sparsetight\vcsp25_10_21_85_3.wcsp
.\random\sparsetight\vcsp25_10_21_85_4.wcsp
.\random\sparsetight\vcsp25_10_21_85_5.wcsp
```

```
SolveWCSP[".\\academics\\zebra.wcsp"]
```

$$(1)\quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$(22)\quad \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 17 \end{pmatrix}\quad \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 20 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 7 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 4 \\ 14 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 5 \\ 25 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 6 \\ 13 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 6 \\ 18 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

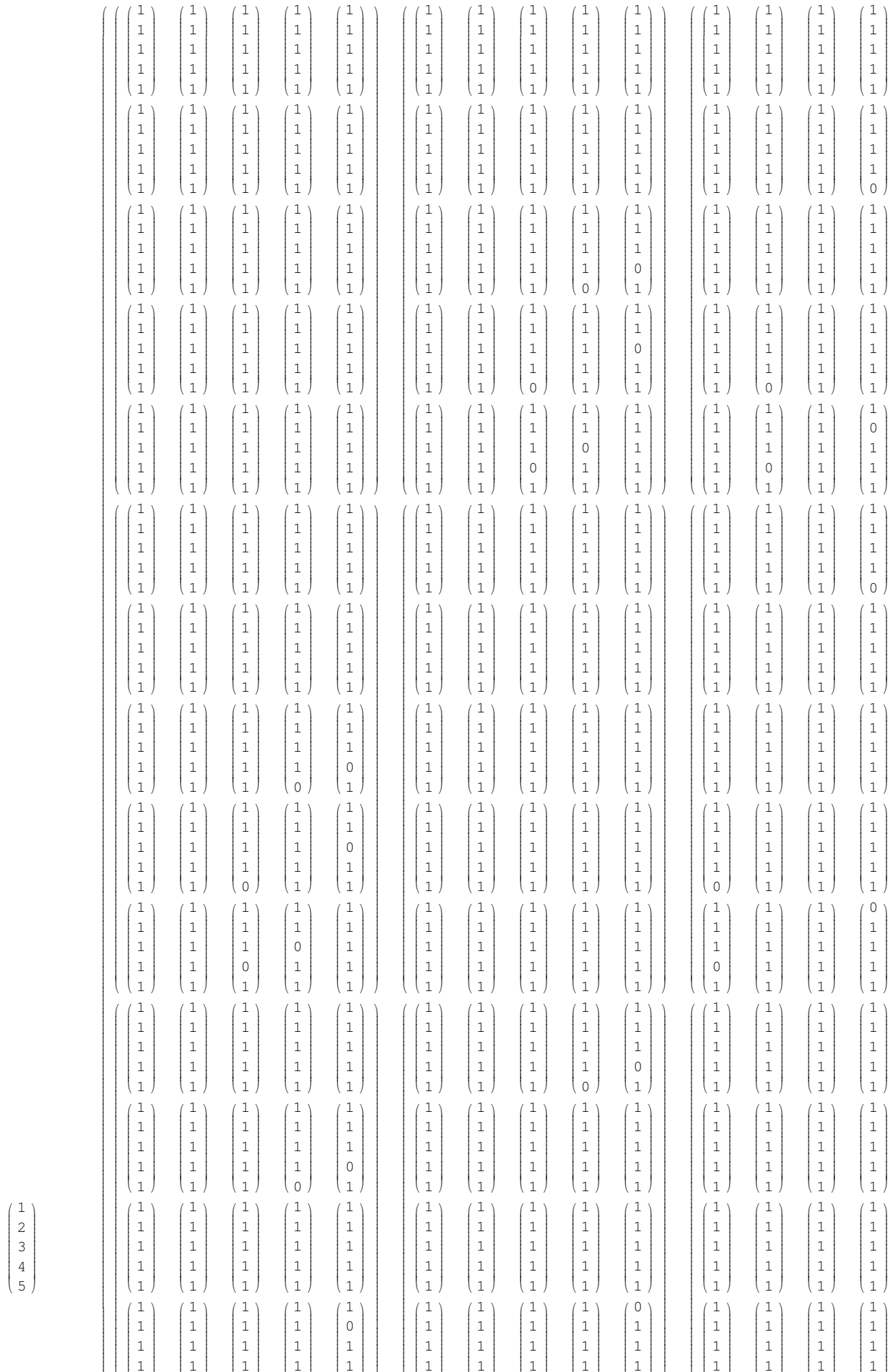
$$\begin{pmatrix} 8 \\ 12 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

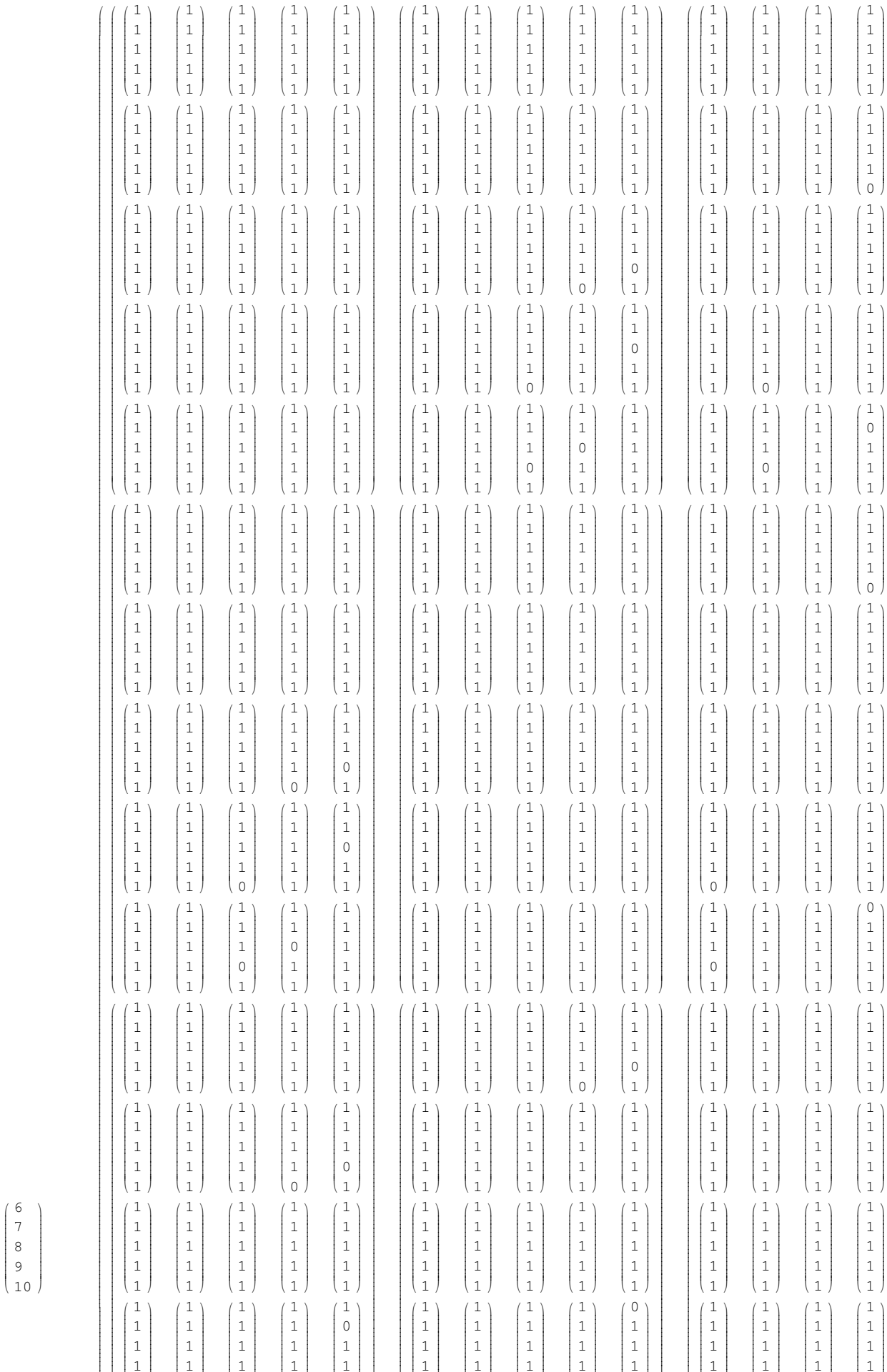
$$\begin{pmatrix} 9 \\ 11 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

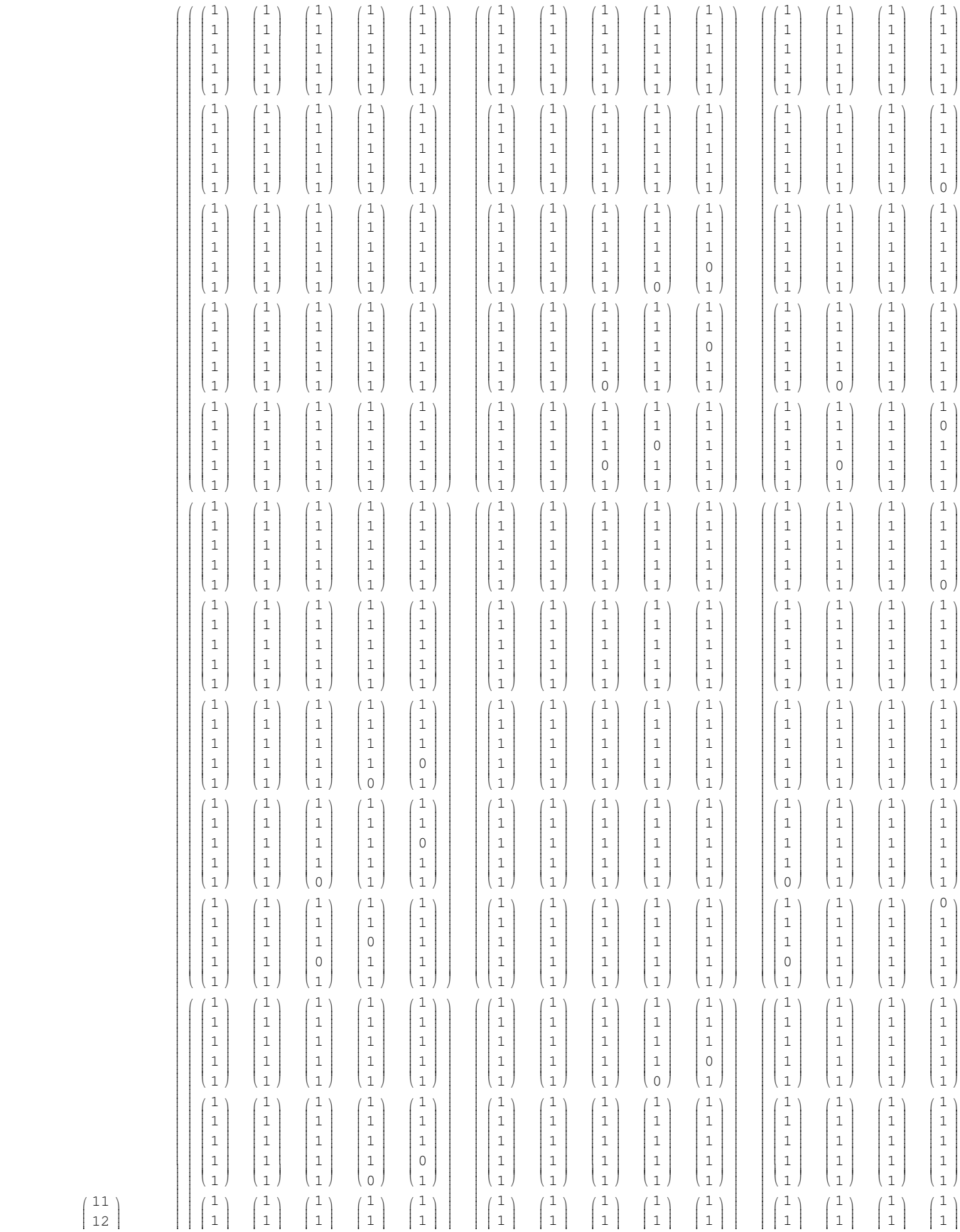
$$\begin{pmatrix} 10 \\ 21 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 16 \\ 19 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 23 \\ 16 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$







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

{11, 12, 13, 14, 15}

{16, 19, 17, 18, 20}

{22, 21, 23, 24, 25}

{21, 23, 24, 25}

{1, 17, 2, 3, 4, 5}

{2, 20, 3, 4, 5, 17}

{3, 7, 4, 5, 17, 20}

{4, 14, 5, 7, 17, 20}

{5, 25, 7, 14, 17, 20}

{7, 6, 8, 9, 10, 14, 17, 20, 25}

{8, 12, 6, 9, 10, 14, 17, 20, 25}

{9, 11, 6, 10, 12, 14, 17, 20, 25}

{10, 21, 6, 11, 12, 14, 17, 20, 25}

{11, 12, 14, 13, 6, 17, 20, 21, 25}

{6, 12, 13, 14, 17, 20, 21, 25}

{6, 13, 14, 17, 20, 21, 25}

{6, 14, 17, 20, 21, 25}

{16, 23, 17, 18, 20}

{17, 20, 6, 21, 25, 18, 23}

{6, 18, 20, 21, 23, 25}

{6, 20, 21, 23, 25}

{21, 23, 25, 6}

{6, 23, 25}

{6, 25}

{6}

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

{p, q, r}

```
{10438905, 10025650,
 {13435, 10910, 8405, 7905, 7875, 7850, 7825, 7800, 7775, 392150, 392125, 392100,
  392075, 391450, 78950, 16425, 3925, 3900, 15775, 3250, 750, 125, 25, 5, 1}}
```

SolveWCSP[".\\academics\\4wqueens.wcsp"]

$$(1)\quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$(2)\quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$(3)\quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$(4)\quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}\quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix}\quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 4 \end{pmatrix}\quad \begin{pmatrix} 5 & 0 & 0 & 5 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 5 & 0 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix}\quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 4 \end{pmatrix}\quad \begin{pmatrix} 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 4 \end{pmatrix}\quad \begin{pmatrix} 5 & 5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 \\ 0 & 0 & 5 & 5 \end{pmatrix}$$

{1, 2, 3, 4}

{1, 2, 3, 4}

{2, 3, 4}

{3, 4}

{4}

{1 → 3, 2 → 1, 3 → 4, 4 → 2}

{p, q, r}

{340, 516, {124, 40, 8, 1}}

SolveWCSP[".\\academics\\8wqueens.wcsp"]

$$(1)\quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$(2)\quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$(3)\quad \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

$$(4)\quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$(5)\quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$(6)\quad \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$(7)\quad \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

(8)
$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \begin{pmatrix} 9 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 9 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 9 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 9 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 9 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 9 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 9 & 0 & 0 & 0 & 0 \\ 9 & 0 & 9 & 0 & 9 & 0 & 0 & 0 \\ 0 & 9 & 0 & 9 & 0 & 9 & 0 & 0 \\ 0 & 0 & 9 & 0 & 9 & 0 & 9 & 0 \\ 0 & 0 & 0 & 9 & 0 & 9 & 0 & 9 \\ 0 & 0 & 0 & 0 & 9 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 9 & 0 & 0 \\ 9 & 0 & 0 & 9 & 0 & 0 & 9 & 0 \\ 0 & 9 & 0 & 0 & 9 & 0 & 0 & 9 \\ 0 & 0 & 9 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 5 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 9 \\ 9 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 6 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 7 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 8 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 9 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 9 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 9 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 9 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 9 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 9 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 9 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 4 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 9 & 0 & 0 & 0 & 0 \\ 9 & 0 & 9 & 0 & 9 & 0 & 0 & 0 \\ 0 & 9 & 0 & 9 & 0 & 9 & 0 & 0 \\ 0 & 0 & 9 & 0 & 9 & 0 & 9 & 0 \\ 0 & 0 & 0 & 9 & 0 & 9 & 0 & 9 \\ 0 & 0 & 0 & 0 & 9 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 5 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 9 & 0 & 0 \\ 9 & 0 & 0 & 9 & 0 & 0 & 9 & 0 \\ 0 & 9 & 0 & 0 & 9 & 0 & 0 & 9 \\ 0 & 0 & 9 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 6 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 9 \\ 9 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 7 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 8 \end{pmatrix} \quad \begin{pmatrix} 9 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\binom{3}{4}\quad\begin{pmatrix}9&9&0&0&0&0&0&0\\9&9&9&0&0&0&0&0\\0&9&9&9&0&0&0&0\\0&0&9&9&9&0&0&0\\0&0&0&9&9&9&0&0\\0&0&0&0&9&9&9&0\\0&0&0&0&0&9&9&9\\0&0&0&0&0&0&9&9\end{pmatrix}$$

$$\binom{3}{5}\quad\begin{pmatrix}9&0&9&0&0&0&0&0\\0&9&0&9&0&0&0&0\\9&0&9&0&9&0&0&0\\0&9&0&9&0&9&0&0\\0&0&9&0&9&0&9&0\\0&0&0&9&0&9&0&9\\0&0&0&0&9&0&9&0\\0&0&0&0&0&9&0&9\end{pmatrix}$$

$$\binom{3}{6}\quad\begin{pmatrix}9&0&0&9&0&0&0&0\\0&9&0&0&9&0&0&0\\0&0&9&0&0&9&0&0\\9&0&0&9&0&0&9&0\\0&9&0&0&9&0&0&9\\0&0&9&0&0&9&0&0\\0&0&0&9&0&0&9&0\\0&0&0&0&9&0&0&9\end{pmatrix}$$

$$\binom{3}{7}\quad\begin{pmatrix}9&0&0&0&9&0&0&0\\0&9&0&0&0&9&0&0\\0&0&9&0&0&0&9&0\\0&0&0&9&0&0&0&9\\9&0&0&0&9&0&0&0\\0&9&0&0&0&9&0&0\\0&0&9&0&0&0&9&0\\0&0&0&9&0&0&0&9\end{pmatrix}$$

$$\binom{3}{8}\quad\begin{pmatrix}9&0&0&0&0&9&0&0\\0&9&0&0&0&0&9&0\\0&0&9&0&0&0&0&9\\0&0&0&9&0&0&0&0\\0&0&0&0&9&0&0&0\\9&0&0&0&0&9&0&0\\0&9&0&0&0&0&9&0\\0&0&9&0&0&0&0&9\end{pmatrix}$$

$$\binom{4}{5}\quad\begin{pmatrix}9&9&0&0&0&0&0&0\\9&9&9&0&0&0&0&0\\0&9&9&9&0&0&0&0\\0&0&9&9&9&0&0&0\\0&0&0&9&9&9&0&0\\0&0&0&0&9&9&9&0\\0&0&0&0&0&9&9&9\\0&0&0&0&0&0&9&9\end{pmatrix}$$

$$\binom{4}{6}\quad\begin{pmatrix}9&0&9&0&0&0&0&0\\0&9&0&9&0&0&0&0\\9&0&9&0&9&0&0&0\\0&9&0&9&0&9&0&0\\0&0&9&0&9&0&9&0\\0&0&0&9&0&9&0&9\\0&0&0&0&9&0&9&0\\0&0&0&0&0&9&0&9\end{pmatrix}$$

$$\binom{4}{7}\quad\begin{pmatrix}9&0&0&9&0&0&0&0\\0&9&0&0&9&0&0&0\\0&0&9&0&0&9&0&0\\9&0&0&9&0&0&9&0\\0&9&0&0&9&0&0&9\\0&0&9&0&0&9&0&0\\0&0&0&9&0&0&9&0\\0&0&0&0&9&0&0&9\end{pmatrix}$$

$$\binom{4}{8}\quad\begin{pmatrix}9&0&0&0&9&0&0&0\\0&9&0&0&0&9&0&0\\0&0&9&0&0&0&9&0\\0&0&0&9&0&0&0&9\\9&0&0&0&9&0&0&0\\0&9&0&0&0&9&0&0\\0&0&9&0&0&0&9&0\\0&0&0&9&0&0&0&9\end{pmatrix}$$

$$\binom{5}{6}\quad\begin{pmatrix}9&9&0&0&0&0&0&0\\9&9&9&0&0&0&0&0\\0&9&9&9&0&0&0&0\\0&0&9&9&9&0&0&0\\0&0&0&9&9&9&0&0\\0&0&0&0&9&9&9&0\\0&0&0&0&0&9&9&9\\0&0&0&0&0&0&9&9\end{pmatrix}$$

$$\binom{5}{7}\quad\begin{pmatrix}9&0&9&0&0&0&0&0\\0&9&0&9&0&0&0&0\\9&0&9&0&9&0&0&0\\0&9&0&9&0&9&0&0\\0&0&9&0&9&0&9&0\\0&0&0&9&0&9&0&9\\0&0&0&0&9&0&9&0\\0&0&0&0&0&9&0&9\end{pmatrix}$$

$$\binom{5}{8}\quad\begin{pmatrix}9&0&0&9&0&0&0&0\\0&9&0&0&9&0&0&0\\0&0&9&0&0&9&0&0\\9&0&0&9&0&0&9&0\\0&9&0&0&9&0&0&9\\0&0&9&0&0&9&0&0\\0&0&0&9&0&0&9&0\\0&0&0&0&9&0&0&9\end{pmatrix}$$

$$\binom{6}{7}\quad\begin{pmatrix}9&9&0&0&0&0&0&0\\9&9&9&0&0&0&0&0\\0&9&9&9&0&0&0&0\\0&0&9&9&9&0&0&0\\0&0&0&9&9&9&0&0\\0&0&0&0&9&9&9&0\\0&0&0&0&0&9&9&9\\0&0&0&0&0&0&9&9\end{pmatrix}$$

$$\binom{6}{8}\quad\begin{pmatrix}9&0&9&0&0&0&0&0\\0&9&0&9&0&0&0&0\\9&0&9&0&9&0&0&0\\0&9&0&9&0&9&0&0\\0&0&9&0&9&0&9&0\\0&0&0&9&0&9&0&9\\0&0&0&0&9&0&9&0\\0&0&0&0&0&9&0&9\end{pmatrix}$$

$$\binom{7}{8}\quad\begin{pmatrix}9&9&0&0&0&0&0&0\\9&9&9&0&0&0&0&0\\0&9&9&9&0&0&0&0\\0&0&9&9&9&0&0&0\\0&0&0&9&9&9&0&0\\0&0&0&0&9&9&9&0\end{pmatrix}$$

```
{1, 2, 3, 4, 5, 6, 7, 8}

{1, 2, 3, 4, 5, 6, 7, 8}

{2, 3, 4, 5, 6, 7, 8}

{3, 4, 5, 6, 7, 8}

{4, 5, 6, 7, 8}

{5, 6, 7, 8}

{6, 7, 8}

{7, 8}

{8}

{1 → 2, 2 → 5, 3 → 7, 4 → 4, 5 → 1, 6 → 8, 7 → 6, 8 → 3}
```

{p, q, r}

```
{19173960, 24309768, {2098552, 263152, 33448, 4512, 728, 144, 16, 1}}
```

SolveWCSP[".\\academics\\16wqueens.wcsp"]

SolveWCSP[".\\academics\\send.wcsp"]

SolveWCSP[".\\dimacs\\ssa0432-003.wcsp"]

SolveWCSP[".\\random\\sparseloose\\vcsp40_10_13_60_1.wcsp"]

SolveWCSP[".\\random\\sparsetight\\vcsp25_10_21_85_1.wcsp"]

SolveWCSP[".\\random\\densetight\\vcsp25_10_25_87_1.wcsp"]

SolveWCSP[".\\random\\denseloose\\vcsp30_10_25_48_1.wcsp"]