The data files for problems and the report

For this assignment (latin square completion problem), the components of solution schemes can be organized along two dimensions. Along one dimension (x-axis), we have BT, FC, MAC, etc. Along another dimension (y-axis), we have the variable ordering heuristics like SDF, max-static-degree, max-dynamic-degree, etc.

problem file	solution scheme-1 MAC + SDF		solution scheme-2 FC + DomDeg		solution scheme-3 FC + Brelaz	
	Number of nodes	Number of backtracks				
d-10-01						
d-10-06						
d-10-07						
d-10-08						
d-10-09						
d-15-01						

The data files are provided as text files. For all problems, it is possible to complete a latin square. For each solution scheme, you should mention the number of nodes and the number of backtracks. For example, 4-queens problem and FC required eight nodes and two backtracks. On the other hand, 4-queens problem and MAC required three nodes and one backtrack.

[It is shown in the webpage: https://ktiml.mff.cuni.cz/~bartak/constraints/propagation.html]

Explanation of BT, FC, and MAC

You should visit the following webpages:

(1) https://ktiml.mff.cuni.cz/~bartak/constraints/ordering.html

(2) https://ktiml.mff.cuni.cz/~bartak/constraints/propagation.html

(3) https://ktiml.mff.cuni.cz/~bartak/constraints/consistent.html

From the first webpage, it is necessary to understand the principles behind the variable ordering and the value ordering heuristics. The second and the third webpages explain BT, FC, and MAC algorithm. For implementing MAC, the pseudocode for Algorithm AC-3 (described in third webpage) may be utilized.

Next, an informal description of the search algorithms is provided.

DFS, BT, FC, MAC are search algorithms which can be used for solving a CSP. In a CSP formulation, we have the variables, domains of the variables (or values), and constraints. The constraints can be viewed as relations. Solving a CSP may be viewed as extending a partially constructed solution to a complete solution. During search, we are going to have a set of assigned variables (which forms the partial solution), and a set of unassigned variables.

- **What is the difference between DFS and BT?**

In depth-first search (DFS), we continue to explore (and assign values to one of the unassigned variables) until we reach a leaf node (when there are no more any unassigned variable). At a leaf node, we make the decision if it is a goal node or solution node.

In backtracking search (BT), we may cut-off the search and do not move forward (or downward) if we recognize the partial solution already violates one (or more) constraint(s). We need to satisfy all the constraints to obtain a solution. In BT, at any search node, we can use the set of assigned variables and test the constraints to make the decision if we can backtrack.

To sum up, we learn that reasoning activity can be used to cut down search activity. BT uses more reasoning and less search in comparison with DFS. We are going to find out that FC uses more reasoning than BT, and MAC uses more reasoning than FC.

- **What is the difference between BT and FC?**

In backtracking search (BT), we are using only the set of assigned variables (the partially constructed solution) to make the backtracking decision. In forward checking (FC), we are also going to use some of unassigned variables who are involved in at least one constraint with the assigned variables. It is easy to find them using the constraint graph because these are the neighbor nodes of the assigned variable nodes.

In FC, at a certain search node, assume that we have just made an assignment of a certain value d1 to a certain variable Xi. Also, assume Xk is an unassigned variable which is involved in a constraint C(Xi, Xk) with Xi. We can apply a filtering algorithm to remove or filter out those values from the domain of Xk which are inconsistent with the assignment Xi = d1

For example, X1 and X2 are two variables, each with domains {1, 2, 3, 4, 5}, and there is a constraint

X1 + X2 > 7. Now if an assignment X2 = 4 is made, we can filter out 1, 2, and 3 from the domain of X1.

When an assignment X2 = 4 is made, the domain of X2 has become {4}, and we test all values remaining in the domain of X1 in order to remove those values which are inconsistent. We find that the following tuples <1, 4>, <2, 4>, and <3, 4> do not satisfy the constraint X1 + X2 > 7

We can backtrack if the domain of an unassigned variable becomes empty (after running the filtering algorithm). For example, assume X1 has a domain of {3, 4} and X2 has been assigned 5. There are two constraints C1 and C2.

C1: X1 + X2 < 7 and C2: X1 + X2 >3

In this case, after running the filtering algorithm, the domain of X1 becomes empty and the decision to backtrack is made.

To sum up, we become familiar with consistency checking and filtering algorithms. They are going to be used for reasoning activity in FC. Also, they allow us to make the backtracking decision earlier than BT.

- **Explaining MAC with the 4-queens problem**

In solving a CSP, we keep partial solutions and extend them to construct a complete solution. Also, solving a CSP may be viewed as solving sub-problems or smaller problems, and then joining the solutions together to obtain the complete solution.

Before explaining MAC, we need to learn the definitions of node consistency and arc consistency. Also, we need to consider the constraint graph.

If we consider that the sub-problems are limited to a single node, then we need to ensure unary constraints related to a particular node is satisfied. In this case, the filtering algorithm removes a value from the domain of a variable, if the value is not consistent with the unary constraint. The node representing a variable V in constraint graph is node consistent if for every value x in the current domain of V, each unary constraint on V is satisfied.

If there is a binary constraint C (Vi, Vj), we have an edge (Vi, Vj) in the constraint graph. This edge is actually two arcs: (1) one arc from Vi to Vj and (2) another arc from Vj to Vi

The Arc (Vi,Vj) is arc consistent if for every value x the current domain of Vi there is some value y in the domain of Vj such that Vi=x and Vj=y is permitted by the binary constraint between Vi and Vj. Here, the sub-problem is limited to an arc. After ensuring the arc consistency of the Arc (Vi, Vj), it is guaranteed that every value in the domain of Vi can participate in at least one valid solution (or be a part of one valid solution) of this sub-problem.

A binary constraint C (x1, x2) is arc consistent if for all values d1 ∈ D(x1) there exists a value d2 ∈ D(x2) such that (d1, d2) ∈ C, and for all values d2 ∈ D(x2) there exists a value d1 ∈ D(x1) such that (d1, d2) ∈ C. Here, the sub-problem is limited to a binary constraint. After ensuring the arc consistency of the binary constraint C (x1, x2), it is guaranteed that for each variable, every value in the domain can be a part of at least one valid tuple.

Finally, A constraint C (x1, . . . , xm) (m > 1) is hyper-arc consistent if for all i ∈ {1, . . . , m} and all values di ∈ D(xi), there exist values dj ∈ D(xj) for all j ∈ {1, . . . , m} − i such that (d1, . . . , dm) ∈ C. Here, the sub-problem is limited to a single constraint which can be binary or ternary or any constraint with arity > 1. Arc consistency is equal to hyper-arc consistency applied to binary constraints. hyper-arc consistency is also referred to as generalized arc-consistency (GAC).

Now, we are going to describe how to solve the 4-queens problem with MAC. It is an informal description of AC-3 algorithm.

Let us assume the CSP formulation of the 4-queens has four variables $q1$, $q2$, $q3$, $q4$. The variable qi refers to the queen placed in column i. Each variable has a domain {$r1$, $r2$, $r3$, $r4$}. If we assign $q1 = r4$, it means the first queen is placed on row 4.

Step-1

Assign the value r1 to q1.

q1 = {r1}

q2 = {r3, r4} /* make the arc (q2, q1) arc consistent by removing value r1, r2 */

q3 = {r2, r4} /* make the arc (q3, q1) arc consistent by removing value r1, r3 */

q4 = {r2, r3} /* make the arc (q4, q1) arc consistent by removing value r1, r4 */

Task list

check the arc (q3, q2) and arc (q4, q2) for arc consistency /* this is because domain of q2 has changed*/

check the arc (q2, q3) and arc (q4, q3) for arc consistency /* this is because domain of q3 has changed*/

check the arc (q2, q4) and arc (q3, q4) for arc consistency /* this is because domain of q4 has changed*/

let us check the arc q3->q2

we can not remove r2 from the domain of q3 because [q3 = r2 and q2 = r4] is valid

we can remove r4 from the domain of q3 because both [q3 = r4 and q2 = r3] and [q3 = r4 and q2 = r4] are not valid.

so, q1 = {r1}, q2 = {r3, r4}, q3 = {r2}, q4 = {r2, r3}

Now, let us check the arc q4->q2

we can not remove r2 from the domain of q4 because [q4 = r2 and q2 = r3] is valid.

we can not remove r3 from the domain of q4 because [q4 = r3 and q2 = r4] is valid.

so, q1 = {r1}, q2 = {r3, r4}, q3 = {r2}, q4 = {r2, r3}

Now, let us check the arc q2->q3

we can remove r3 from the domain of q2 because [q2 = r3 and q3 = r2] is not valid.

we can not remove r4 from the domain of q2 because [q2 = r4 and q3 = r2] is valid.

so, q1 = {r1}, q2 = {r4}, q3 = {r2}, q4 = {r2, r3}

Now let us check the arc q4->q3

we can remove r2 from the domain of q4 because [q4 = r2 and q3 = r2] is not valid.

we can remove r3 from the domain of q4 because [q4 = r3 and q3 = r2] is not valid.

so, q1 = {r1}, q2 = {r4}, q3 = {r2}, q4 = {}

The decision to backtrack is made because the domain of q4 is empty.

Step-2

Assign the value r2 to q1.

q1 = {r2}

q2 = {r4} /* make the arc (q2, q1) arc consistent by removing value r2, r1, r3 */

q3 = {r1, r3} /* make the arc (q3, q1) arc consistent by removing value r2, r4 */

q4 = {r1, r3, r4} /* make the arc (q4, q1) arc consistent by removing value r2 */

Task list

check the arc (q3, q2) and arc (q4, q2) for arc consistency /* this is because domain of q2 has changed*/

check the arc (q2, q3) and arc (q4, q3) for arc consistency /* this is because domain of q3 has changed*/

check the arc (q2, q4) and arc (q3, q4) for arc consistency /* this is because domain of q4 has changed*/

Let us check the arc q3->q2

we can not remove r1 from the domain of q3 because [q3 = r1 and q2 = r4] is valid

we can remove r3 from the domain of q3 because [q3 = r3 and q2 = r4] is not valid

So, q1 = {r2}, q2 = {r4}, q3 = {r1}, q4 = {r1, r3, r4}

Let us check the arc q4->q2

we can not remove r1 from the domain of q3 because [q4 = r1 and q2 = r4] is valid

we can not remove r3 from the domain of q3 because [q4 = r3 and q2 = r4] is valid

we can remove r4 from the domain of q4 because [q4 = r4 and q2 = r4] is not valid

So, q1 = {r2}, q2 = {r4}, q3 = {r1}, q4 = {r1, r3}

Let us check the arc q2->q3

we can not remove r4 from the domain of q2 because [q2 = r4 and q3 = r1] is valid

So, q1 = {r2}, q2 = {r4}, q3 = {r1}, q4 = {r1, r3}

Let us check the arc q4->q3

we can remove r1 from the domain of q4 because [q4 = r1 and q3 = r1] is not valid

we can not remove r3 from the domain of q4 because [q4 = r3 and q3 = r1] is valid

So, q1 = {r2}, q2 = {r4}, q3 = {r1}, q4 = {r3}

We have found a solution.

You should visit the webpage https://ktiml.mff.cuni.cz/~bartak/constraints/consistent.html and you should work out Algorithm AC-1 by hand step by step. It is easier to understand MAC algorithm by AC-1, but AC-3 is more efficient.