INSTITUTO SUPERIOR TÉCNICO

Search and Planning

2021/2022 Academic Year 1st Period

1st Exam November 22, 2021 Duration: 2h

- This is a closed book exam.
- **You should resolve 4 out of the 5 groups in the exam**.
- Ensure that your name and number are written on all pages.

EXAM SOLUTION

I. Modeling as a CSP (2+3 = 5/20)

1) A Latin square of order n is defined as an $n \times n$ matrix made out of the integers in $[1..n]$ with the property that each of these n integers occurs exactly once in each row and exactly once in each column of the array. The following matrix is an example of a Latin square of order 5.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \\ 3 & 4 & 5 & 1 & 2 \\ 4 & 5 & 1 & 2 & 3 \\ 5 & 1 & 2 & 3 & 4 \end{bmatrix}$$

Propose a formulation of the problem as a constraint network. Identify variables, domains, and constraints.

Solution:
Variables:
$X = \{x_{ij} | 1 \leq i \leq n, 1 \leq j \leq n\}$ where x_{ij} is the entry in the i-th row and j-th column of the matrix, and n is the order of the Latin square, i.e. the number of rows and columns.
Domains:
All variables share a common domain, namely $D = \{k | 1 \leq k \leq n\}$
Constraints:
$\forall_i$ AllDiff$(x_{i1}, ..., x_{in})$ and $\forall_j$ AllDiff$(x_{1j}, ..., x_{nj})$ to ensure that a value occurs at most once in each row and column, respectively. The at least once constraint derives from the domain.

2) Now consider the problem of orthogonal Latin squares. Let A and B be Latin squares of order n and let the entry on the i^{th} row and the j^{th} column of A and B be denoted as a_{ij} and b_{ij}, respectively, with $i, j = 1, ..., n$. A and B are orthogonal if the n^2 order pairs (a_{ij}, b_{ij}) are all distinct. For example, the following juxtaposed Latin squares are orthogonal.

$$\begin{bmatrix} (3,2) & (2,3) & (1,1) \\ (2,1) & (1,2) & (3,3) \\ (1,3) & (3,1) & (2,2) \end{bmatrix}$$

Propose a formulation of the problem as a constraint network. Identify variables, domains, and constraints.

Solution:
Variables:
$A = \{a_{ij} | 1 \leq i \leq n, 1 \leq j \leq n\}$ where a_{ij} is the entry in the Latin square A,
$B = \{b_{ij} | 1 \leq i \leq n, 1 \leq j \leq n\}$ where b_{ij} is the entry in the Latin square B,
$X = \{(a_{i,j}, b_{ij}) | 1 \leq i \leq n, 1 \leq j \leq n\}$ where a_{ij} and b_{ij} are the entries in the Latin squares A and B, respectively, and n is the order of the Latin squares.
Domains:
All variables in A and B share a common domain, namely $D = \{k | 1 \leq k \leq n\}$.
Consequently, variables in X have domain $D = \{(l,m) | 1 \leq l \leq n, 1 \leq m \leq n\}$.
Constraints:
AllDifferent(X)

II. Inference in CSP (1.5 + 1.5 + 2 = 5/20)

1) Consider a three-variable network z,x,y with $D_z = D_x = \{2,5\}$ and $D_y = \{2,4\}$. There are two constraints: R_{zx} specifying that z evenly divides x, and R_{zy} specifying that z evenly divides y. Apply AC-3 to the network according to the algorithm that is given below.

AC-3($\mathcal{R}$)
Input: A network of constraints $\mathcal{R}$ = (X, D, C).
Output: $\mathcal{R}'$, which is the largest arc-consistent network equivalent to $\mathcal{R}$.
1. **for** every pair $\{x_i, x_j\}$ that participates in a constraint $R_{ij} \in \mathcal{R}$
2. *queue* ← *queue* ∪ $\{(x_i, x_j), (x_j, x_i)\}$
3. **endfor**
4. **while** *queue* ≠ {}
5. select and delete (x_i, x_j) from *queue*
6. REVISE((x_i), x_j)
7. **if** REVISE((x_i), x_j) causes a change in D_i
8. **then** *queue* ← *queue* ∪ $\{(x_k, x_i), k \neq i, k \neq j\}$
9. **endif**
10. **endwhile**

Solution:

We put (z,x), (x,z), (z,y) and (y,z) onto the queue. Processing the pairs (z,x) and (x,z) does not change the problem, since the domains of z and x are already arc-consistent relative to R_{zx}. When we process (z,y), we delete 5 from D_z, and consequently place (x,z) back on the queue. Processing (y,z) causes no further change, but when (x,z) is revised, 5 will be deleted from D_x. At this point, no further constraints will be inserted into the queue, and the algorithm terminates with the domains $D_z = D_x = \{2\}$ and $D_y = \{2,4\}$.

2) Consider the graph-coloring problem on a four-variable network where the domains contain only two possible colors $D_1 = D_2 = D_3 = D_4 = \{red, blue\}$ and the constraints are $x_1 \neq x_2$, $x_2 \neq x_3$, $x_3 \neq x_4$ and $x_4 \neq x_1$. Explain why the network is currently arc-consistent but not path-consistent. For your convenience, the PC-2 algorithm is given below.

PC-2($\mathcal{R}$)

Input: A network $\mathcal{R} = (X, D, C)$.

Output: $\mathcal{R}'$ a path-consistent network equivalent to $\mathcal{R}$.

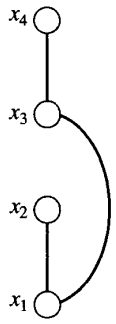
1. $Q \leftarrow \{(i, k, j) \mid 1 \leq i < j \leq n,\ 1 \leq k \leq n,\ k \neq i,\ k \neq j\}$
2. **while** Q is not empty
3. select and delete a 3-tuple (i, k, j) from Q
4. $R_{ij} \leftarrow R_{ij} \cap \pi_{ij}\ (R_{ik} \bowtie D_k \bowtie R_{kj})$ /* (REVISE-3((i, j), k))
5. **if** R_{ij} changed then
6. $Q \leftarrow Q \cup \{(l, i, j)\ (l, j, i) \mid 1 \leq l \leq n,\ l \neq i,\ l \neq j\}$
7. **endwhile**

Solution:

Considering only pairs of variables, we conclude that this network is arc-consistent without reducing any domains. However, considering triplets of variables, we observe that the network is not path-consistency. Although the constraint R_{xy}, which is $x \neq y$, allows the assignment $x = red$ and $y = blue$ (or $x = blue$ and $y = red$), no color for z will be consistent with these assignments. Indeed, since the domain size is 2, we can infer $x \neq y$ and $y \neq z$ imply $x = z$. In practice, PC adds to the network the constraints $x_1 = x_3$ and $x_2 = x_4$ as a result of reducing R_{13} and R_{24} to $\{(red, red), (blue, blue)\}$.

3) Consider that the domains and the constraints of the ordered constraint graph given below are as follows:

D_1 = {red, white, black}
D_2 = {green, white, black}
D_3 = {red, white, blue}
D_4 = {white, blue, black}
$R_{12} : x_1 = x_2$
$R_{13} : x_1 = x_3$
$R_{34} : x_3 = x_4$



Apply directional arc-consistency (DAC) using the ordering $\{x_1, x_2, x_3, x_4\}$. Is the resulting network full arc consistent? Is the resulting network backtrack-free? Justify your answers.

DAC($\mathcal{R}$)

Input: A network $\mathcal{R} = (X, D, C)$, its constraint graph G, and an ordering $d = (x_1, \ldots, x_n)$.

Output: A directional arc-consistent network.

1. **for** $i = n$ to 1 by -1 **do**
2. **for** each $j < i$ such that $R_{ji} \in \mathcal{R}$, do
3. $D_j \leftarrow D_j \cap \pi_j (R_{ji} \bowtie D_i)$, (this is REVISE($(x_j)$, x_i)).
4. **endfor**

Solution:

The algorithm processes the variables in the reverse order along the ordered graph. Starting with x_4, DAC first revises x_3 relative to x_4, deleting *red* from D_3 (since *red* of x_3 has no match in D_4), yielding $D_3 = \{white, blue\}$. Since x_4 has only this one constraint, processing proceeds with x_3, and the constraint R_{13} is tested to ensure that x_1 is arc-consistent relative to x_3. As a result, *red* and *black* are eliminated from D_1 since they have no equals in the updated domain of D_3. When x_2 is next processed, nothing changes because x_1, with its current domain $D_1 = \{white\}$, is already arc-consistent relative to x_2. The final resulting domains are D_1 = {white}, D_2 = {green, white, black}, D_3 = {white, blue} and D_4 = {white, blue, black}, yielding a directional arc-consistent network relative to the given ordering.

The resulting network is not full arc-consistent because variable x_3 is not arc-consistent relative to x_1 at the light of R_{31}, given that blue in D_3 has no match in D_1. However, the network is backtrack-free because if we try to assign values in the forward direction of the ordering d = (x_1, x_2, x_3, x_4) a consistent assignment is made to every variable ($x_1 = x_2 = x_3 = x_4$ =white) and no dead-end is encountered.

III. Search in CSP (2 + 3 = 5/20)

1) Consider the following figure with squares labeled with letters. Suppose we want to assign an integer from 1 to 8 to each square such that the number in square B must be half the number of square A (using integer division), and the number of square C must be smaller than the number of square B.

A	B	C

This problem may be formalized as a CSP as follows:
Variables and Domains:

$$\{A, B, C\}$$

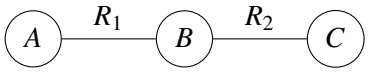
$$D_A = D_B = D_C = 1..8$$

Constraints:

$$R_1 : B = \left\lfloor \frac{A}{2} \right\rfloor$$

$$R_2 : C < B$$

Graph:



Perform backtracking search. The variables and values must be chosen using an alphabetical/numerical ascending order.

Solution:

1 $A = 1$
2 $B = \not{1}, \not{2}, \not{3}, \not{4}, \not{5}, \not{6}, \not{7}, \not{8}$

3 $A = 2$
4 $B = 1$
5 $C = \not{1}, \not{2}, \not{3}, \not{4}, \not{5}, \not{6}, \not{7}, \not{8}$
6 $B = \not{2}, \not{3}, \not{4}, \not{5}, \not{6}, \not{7}, \not{8}$

7 $A = 3$
8 $B = 1$
9 $C = \not{1}, \not{2}, \not{3}, \not{4}, \not{5}, \not{6}, \not{7}, \not{8}$
10 $B = \not{2}, \not{3}, \not{4}, \not{5}, \not{6}, \not{7}, \not{8}$

11 $A = 4$
12 $B = \not{1}, 2$
13 $C = 1$ ✓

2) Consider again the following CSP problem:

Variables and Domains:

$$\{A, B, C\}$$

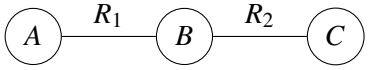
$$D_A = D_B = D_C = 1..8$$

Constraints:

$$R_1 : B = \left\lfloor \frac{A}{2} \right\rfloor$$

$$R_2 : C < B$$

Graph:



a) Perform backtracking search combined with forward checking.

b) **Briefly** discuss the advantages of backjumping when compared with backtracking.

Solution:

a)

1	$A = 1$	$D(B) = \emptyset$
2	$A = 2$	$D(B) = \{1\}$
3	$B = 1$	$D(C) = \emptyset$
4	$A = 3$	$D(B) = \{1\}$
5	$B = 1$	$D(C) = \emptyset$
6	$A = 4$	$D(B) = \{2\}$
7	$B = 2$	$D(C) = \{1\}$
8	$C = 1$ ✓	

b) Backjumping attempts to counteract backatracking's propensity for thrashing, or repeatedly rediscovering the same inconsistencies and partial successes during search. In backjumping, by analyzing the reasons for a dead-end, irrelevant backtrack points can often be avoided so that the algorithm jumps back directly to the source of failure, instead of just to the immediately preceding variable in the ordering.

IV. Plan Space Planning (2 + 3 = 5/20)

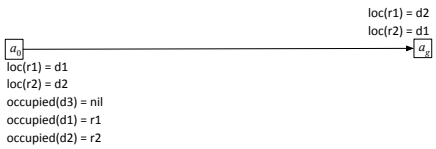
1) Consider a simple planning problem in which there are two robots and three loading docks, that is, B = Robots ∪ Docks, where Robots = {r1, r2} and Docks = {d1, d2, d3}. There are no rigid relations. There is one action template,

move(r, d, d')
pre: loc(r) = d, occupied(d') = nil
eff: loc(r) ← d', occupied(d) = nil, occupied(d') = r

where r ∈ Robots and d,d' ∈ Docks. The initial state and the goal are:

s_0 ={loc(r1)=d1, loc(r2)=d2, occupied(d1) = r1, occupied(d2) = r2, occupied(d3) = nil};
g = {loc(r1) = d2, loc(r2) = d1}.

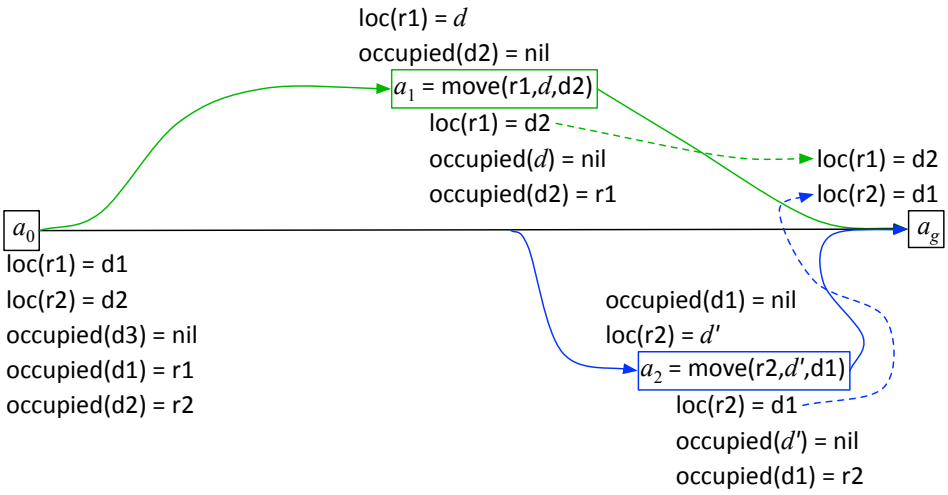
In the context of Plan-Space Search (PSP), the figure below is the initial partial plan containing dummy actions a_0 and a_g that represent s_0 and g.



a) Identify the two flaws in the initial plan.
b) Resolve the two flaws adding to the figure the appropriate actions and causal links.

Solution:

The two flaws are the two open goals: loc(r1) = d2 and loc(r2) = d1.



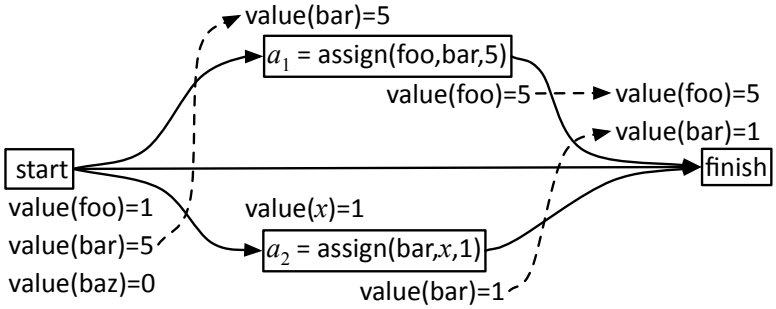
2) Consider a state-variable version of the problem of swapping the values of two variables. The set of objects is B = Variables ∪ Numbers, where Variables = {foo, bar, baz}, and Numbers = {0, 1, 2, 3, 4, 5}. There is one action template:

assign(x_1, x_2, n)
pre: value(x_2) = n
eff: value(x_1) ← n

where Range(x_1) = Range(x_2) = Variables, and Range(n) = Numbers. Consider that the initial state and goal are:

s_0 ={value(foo)=1,value(bar)=5,value(baz)=0};
g = {value(foo) = 5, value(bar) = 1}.

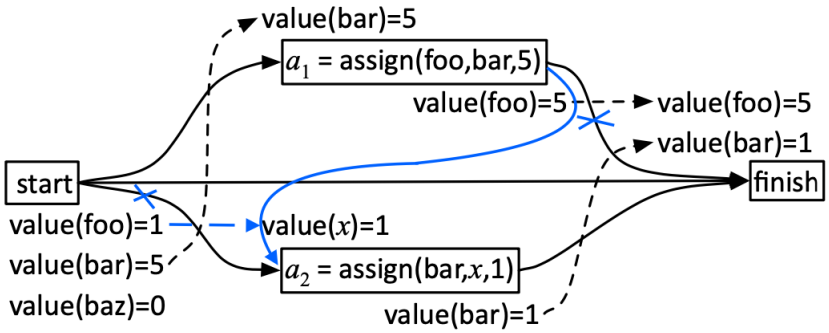
This figure shows a partial plan for this problem instance.



Identify the existing flaws. Update the figure, adding resolvents for each flaw.

Solution:

There is one open goal: value(x)=1. There is one threat: action a_2 has effect value(bar)=1 and so threatens the causal link that supports the pre-condition value(bar)=5 of action a_1.
The open goal can be resolved with a causal link from start to a_2 and substitution x=foo. The threat can be resolved requiring a_2 to come after a_1. (Note that the resulting plan has flaws and a new action will have to be added to the final plan.)

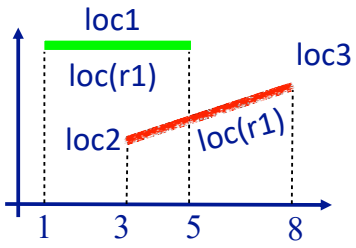


V. Temporal Planning (3 + 2 = 5/20)

1) Consider the following timeline
$(\mathcal{T}, \mathcal{C})$ = ({[t_1,t_2] loc(r1) = loc1, [t_3,t_4] loc(r) : (l, l')}, {$t_1 < t_2$, $t_3 < t_4$})

a) Note that the temporal assertion [t_3,t_4] loc(r) : (l, l') has no causal support, i.e. what causes r to be at l at time t_3? Provide **three** different ways of resolving this flaw.

b) Note that $\mathcal{T}$ contains a pair of possibly-conflicting temporal assertions, as illustrated by the figure. Provide **five** different sets of separation constraints to resolve this flaw.



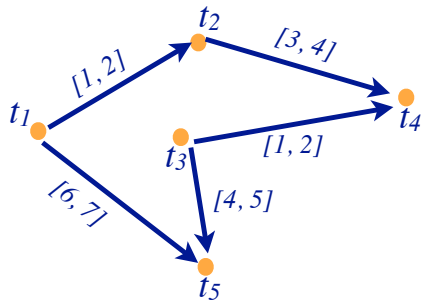
Solution:

a) Three resolvents:
(i) add constraints r = r1, l = loc1, $t_2 = t_3$,
(ii) add a new persistence assertion r = r1, l = loc1, [t_2,t_3] loc(r1) = loc1,
(iii) add an action or task r = r1, [t_2,t_3] move(r1,l).

b) Five separation constraints:
(i) r $\neq$ r1
(ii) $t_2 < t_3$
(iii) $t_4 < t_1$
(iv) $t_2 = t_3$, r = r1 , l = loc1 (also provides causal support for [t_3,t_4] loc(r) : (l, l'))
(v) $t_4 = t_1$, r = r1 , l' = loc1 (also provides causal support for [t_1,t_2] loc(r1) = loc1)

2) Run algorithm Path-Consistency (PC) on the following temporal network. Show that it adds the constraints r_{13} = ~~[1, 3]~~ [2,3], r_{24} = ~~[1, 2]~~ [3,4] and r_{45} = [2, 3].

PC($\mathcal{V}, \mathcal{E}$)
  for $k = 1, \ldots, n$ do
    for each pair i, j such that $1 \leq i < j \leq n, i \neq k$, and $j \neq k$ do
      $r_{ij} \leftarrow r_{ij} \cap [r_{ik} \bullet r_{kj}]$
      if $r_{ij} = \varnothing$ then return *inconsistent*



Solution:

```
k: 1 i: 2 j: 3 || Rij /\ [Rik * Rkj]: [-inf, inf] || Rij: [-inf, inf] || Rik * Rkj: [-inf, inf] || Rik: [-2, -1]    || Rkj: [-inf, inf]
k: 1 i: 2 j: 4 || Rij /\ [Rik * Rkj]: [3, 4]      || Rij: [3, 4]      || Rik * Rkj: [-inf, inf] || Rik: [-2, -1]    || Rkj: [-inf, inf]
k: 1 i: 2 j: 5 || Rij /\ [Rik * Rkj]: [4, 6]      || Rij: [-inf, inf] || Rik * Rkj: [4, 6]      || Rik: [-2, -1]    || Rkj: [6, 7]
k: 1 i: 3 j: 4 || Rij /\ [Rik * Rkj]: [1, 2]      || Rij: [1, 2]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [-inf, inf]
k: 1 i: 3 j: 5 || Rij /\ [Rik * Rkj]: [4, 5]      || Rij: [4, 5]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [6, 7]
k: 1 i: 4 j: 5 || Rij /\ [Rik * Rkj]: [-inf, inf] || Rij: [-inf, inf] || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [6, 7]
k: 2 i: 1 j: 3 || Rij /\ [Rik * Rkj]: [-inf, inf] || Rij: [-inf, inf] || Rik * Rkj: [-inf, inf] || Rik: [1, 2]      || Rkj: [-inf, inf]
k: 2 i: 1 j: 4 || Rij /\ [Rik * Rkj]: [4, 6]      || Rij: [-inf, inf] || Rik * Rkj: [4, 6]      || Rik: [1, 2]      || Rkj: [3, 4]
k: 2 i: 1 j: 5 || Rij /\ [Rik * Rkj]: [6, 7]      || Rij: [6, 7]      || Rik * Rkj: [5, 8]      || Rik: [1, 2]      || Rkj: [4, 6]
k: 2 i: 3 j: 4 || Rij /\ [Rik * Rkj]: [1, 2]      || Rij: [1, 2]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [3, 4]
k: 2 i: 3 j: 5 || Rij /\ [Rik * Rkj]: [4, 5]      || Rij: [4, 5]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [4, 6]
k: 2 i: 4 j: 5 || Rij /\ [Rik * Rkj]: [0, 3]      || Rij: [-inf, inf] || Rik * Rkj: [0, 3]      || Rik: [-4, -3]    || Rkj: [4, 6]
k: 3 i: 1 j: 2 || Rij /\ [Rik * Rkj]: [1, 2]      || Rij: [1, 2]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [-inf, inf]
k: 3 i: 1 j: 4 || Rij /\ [Rik * Rkj]: [4, 6]      || Rij: [4, 6]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [1, 2]
k: 3 i: 1 j: 5 || Rij /\ [Rik * Rkj]: [6, 7]      || Rij: [6, 7]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [4, 5]
k: 3 i: 2 j: 4 || Rij /\ [Rik * Rkj]: [3, 4]      || Rij: [3, 4]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [1, 2]
k: 3 i: 2 j: 5 || Rij /\ [Rik * Rkj]: [4, 6]      || Rij: [4, 6]      || Rik * Rkj: [-inf, inf] || Rik: [-inf, inf] || Rkj: [4, 5]
k: 3 i: 4 j: 5 || Rij /\ [Rik * Rkj]: [2, 3]      || Rij: [0, 3]      || Rik * Rkj: [2, 4]      || Rik: [-2, -1]    || Rkj: [4, 5]
k: 4 i: 1 j: 2 || Rij /\ [Rik * Rkj]: [1, 2]      || Rij: [1, 2]      || Rik * Rkj: [0, 3]      || Rik: [4, 6]      || Rkj: [-4, -3]
k: 4 i: 1 j: 3 || Rij /\ [Rik * Rkj]: [2, 5]      || Rij: [-inf, inf] || Rik * Rkj: [2, 5]      || Rik: [4, 6]      || Rkj: [-2, -1]
k: 4 i: 1 j: 5 || Rij /\ [Rik * Rkj]: [6, 7]      || Rij: [6, 7]      || Rik * Rkj: [6, 9]      || Rik: [4, 6]      || Rkj: [2, 3]
k: 4 i: 2 j: 3 || Rij /\ [Rik * Rkj]: [1, 3]      || Rij: [-inf, inf] || Rik * Rkj: [1, 3]      || Rik: [3, 4]      || Rkj: [-2, -1]
k: 4 i: 2 j: 5 || Rij /\ [Rik * Rkj]: [5, 6]      || Rij: [4, 6]      || Rik * Rkj: [5, 7]      || Rik: [3, 4]      || Rkj: [2, 3]
k: 4 i: 3 j: 5 || Rij /\ [Rik * Rkj]: [4, 5]      || Rij: [4, 5]      || Rik * Rkj: [3, 5]      || Rik: [1, 2]      || Rkj: [2, 3]
k: 5 i: 1 j: 2 || Rij /\ [Rik * Rkj]: [1, 2]      || Rij: [1, 2]      || Rik * Rkj: [0, 2]      || Rik: [6, 7]      || Rkj: [-6, -5]
k: 5 i: 1 j: 3 || Rij /\ [Rik * Rkj]: [2, 3]      || Rij: [2, 5]      || Rik * Rkj: [1, 3]      || Rik: [6, 7]      || Rkj: [-5, -4]
k: 5 i: 1 j: 4 || Rij /\ [Rik * Rkj]: [4, 5]      || Rij: [4, 6]      || Rik * Rkj: [3, 5]      || Rik: [6, 7]      || Rkj: [-3, -2]
k: 5 i: 2 j: 3 || Rij /\ [Rik * Rkj]: [1, 2]      || Rij: [1, 3]      || Rik * Rkj: [0, 2]      || Rik: [5, 6]      || Rkj: [-5, -4]
k: 5 i: 2 j: 4 || Rij /\ [Rik * Rkj]: [3, 4]      || Rij: [3, 4]      || Rik * Rkj: [2, 4]      || Rik: [5, 6]      || Rkj: [-3, -2]
k: 5 i: 3 j: 4 || Rij /\ [Rik * Rkj]: [1, 2]      || Rij: [1, 2]      || Rik * Rkj: [1, 3]      || Rik: [4, 5]      || Rkj: [-3, -2]
```