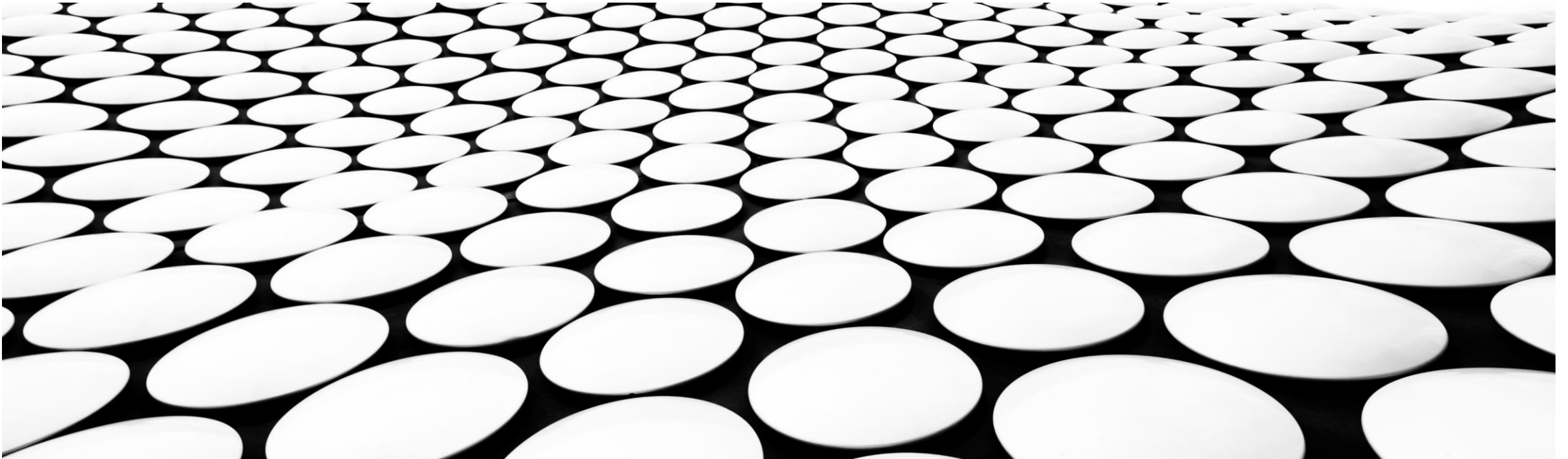


EXPLORING THE SOLUTION SPACE

SATISFIABILITY AND OPTIMIZATION





EXPLORING THE SOLUTION SPACE

- Part 1 – Constraint Propagation
- Part 2 – Simulated Annealing
- Part 3 – Genetic Algorithm

Part 1

Constraint Propagation

EXAMPLE OF A CSP

- Variables:

A, B, C, D, E represent activities

- Domains:

- $D_A = \{1, 2, 3, 4\}$
- $D_B = \{1, 2, 3, 4\}$
- $D_C = \{1, 2, 3, 4\}$
- $D_D = \{1, 2, 3, 4\}$
- $D_E = \{1, 2, 3, 4\}$

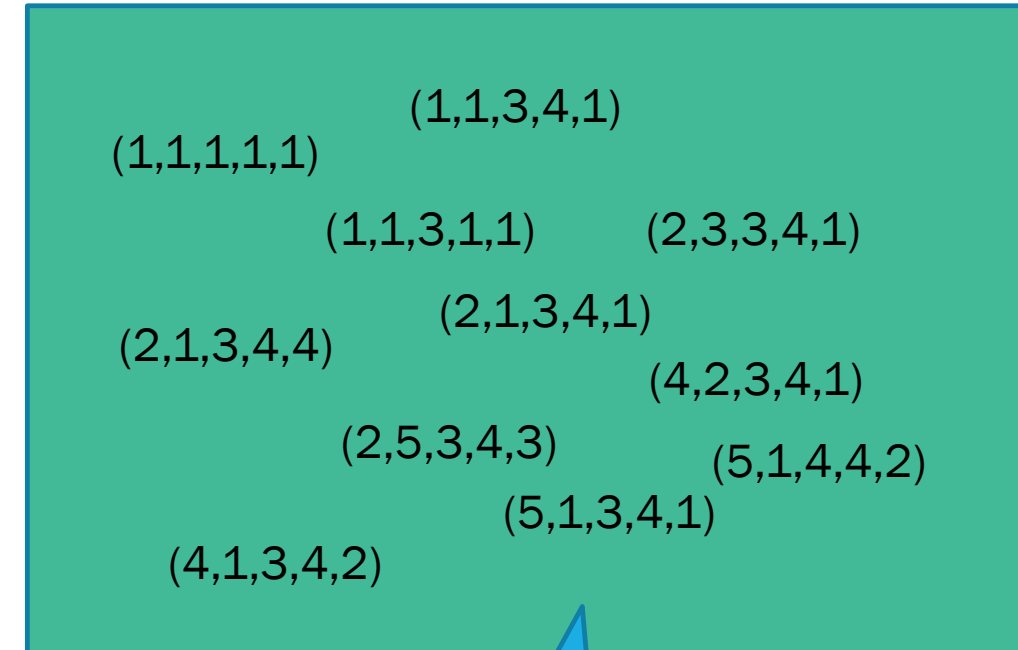
- Hard constraints:

$(B \neq 3)$ and $(C \neq 2)$ and $(A \neq B)$ and $(B \neq C)$ and
 $(C < D)$ and $(A = D)$ and $(E < A)$ and $(E < B)$ and
 $(E < C)$ and $(E < D)$ and $(B \neq D)$

- Soft constraints:

B and C should be as small as possible.

Solution space (A,B,C,D,E)



Find the solution(s)

EXAMPLE OF A CSP

- Variables:

A, B, C, D, E represent activities

- Domains:

- $D_A = \{1, 2, 3, 4\}$
- $D_B = \{1, 2, 3, 4\}$
- $D_C = \{1, 2, 3, 4\}$
- $D_D = \{1, 2, 3, 4\}$
- $D_E = \{1, 2, 3, 4\}$

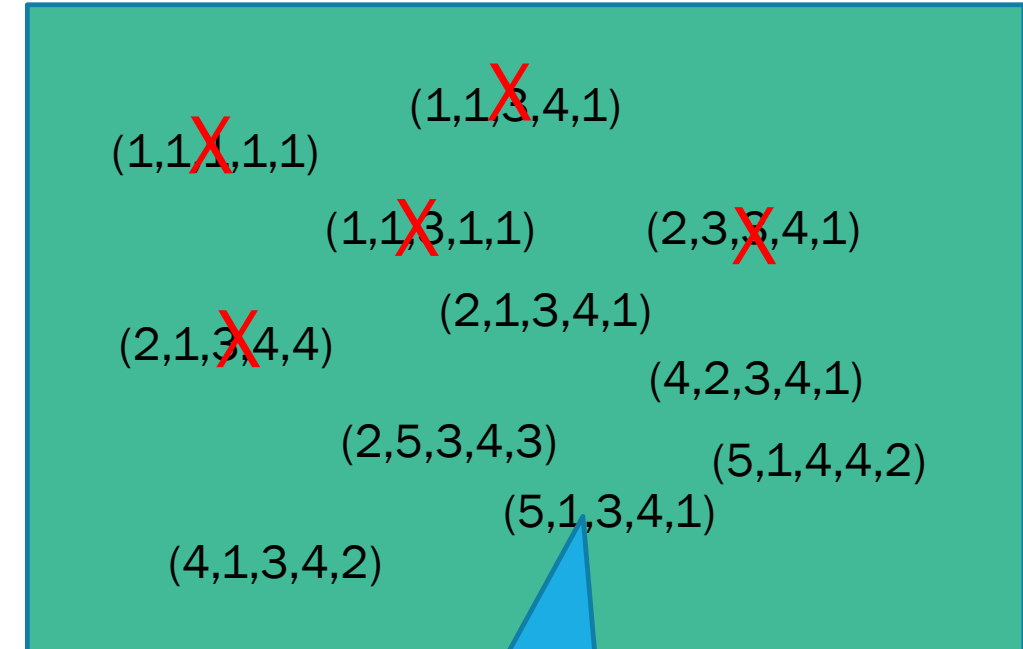
- Hard constraints:

$(B \neq 3)$ and $(C \neq 2)$ and $(A \neq B)$ and $(B \neq C)$ and
 $(C < D)$ and $(A = D)$ and $(E < A)$ and $(E < B)$ and
 $(E < C)$ and $(E < D)$ and $(B \neq D)$

- Soft constraints:

B and C should be as small as possible.

Solution space (A, B, C, D, E)



Can we reduce the domains to not even explore these solutions?

Domain reduction through constraint propagation.

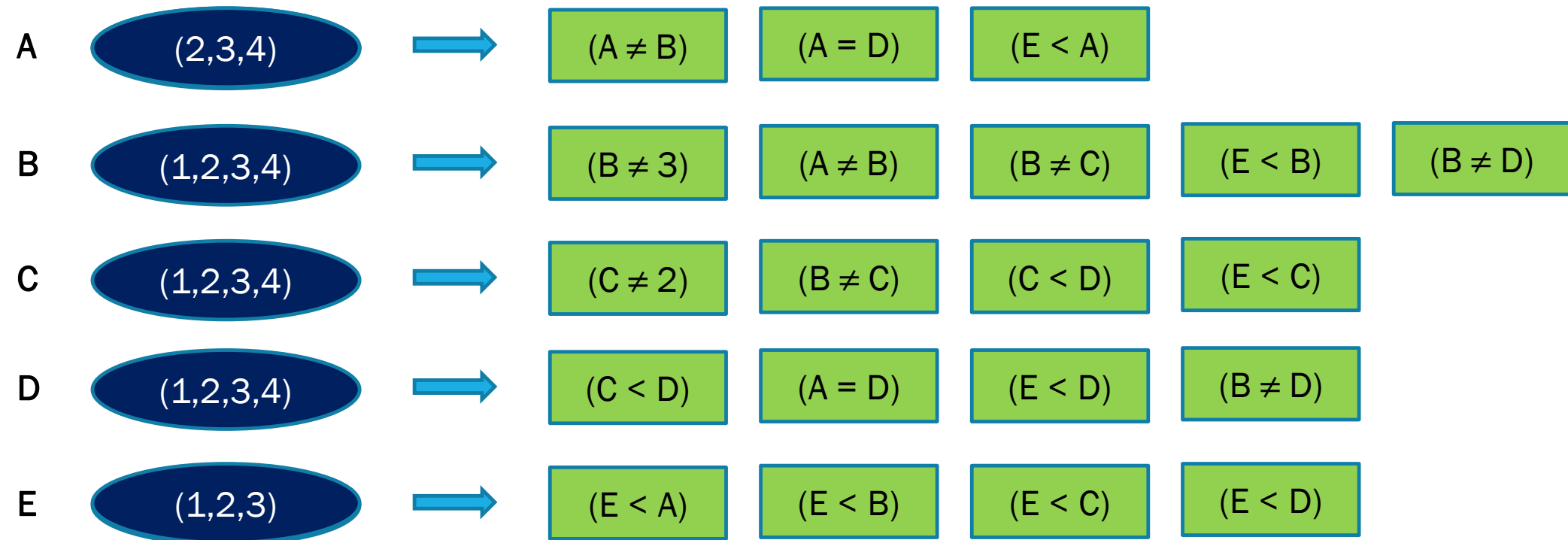
Types of Constraints

- Equality
- Inequality
- Ordering

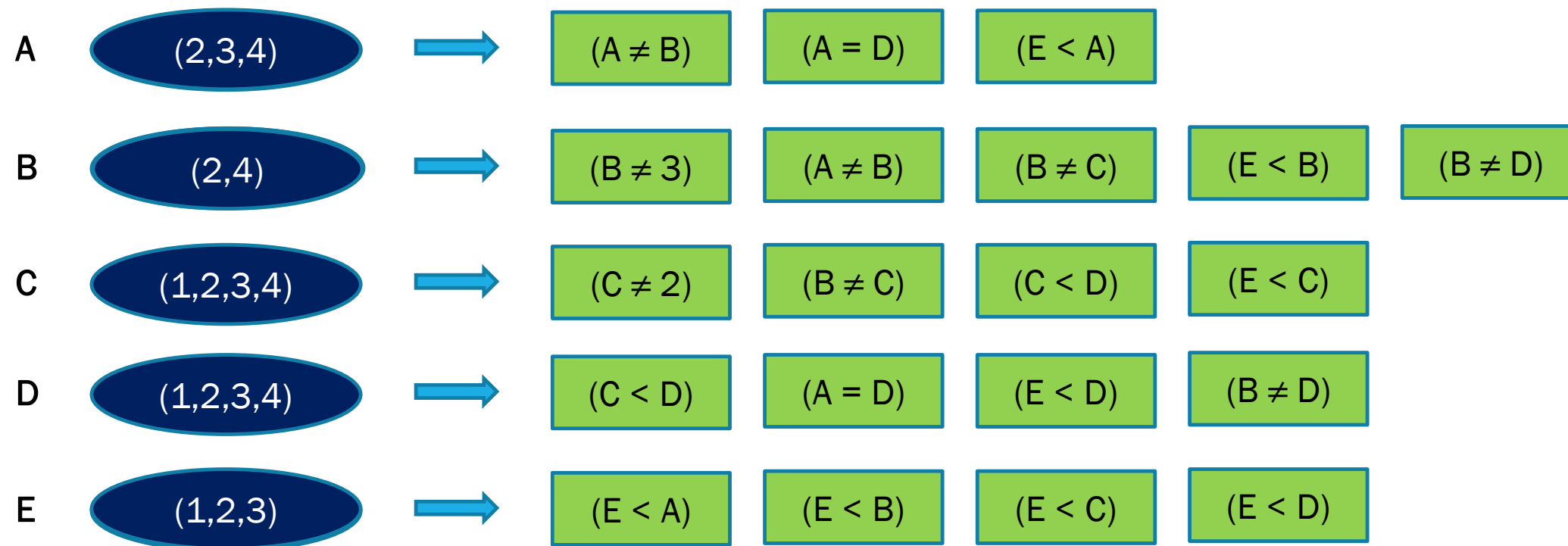
CONSTRAINT PROPAGATION (ONE VERSION OF THE ALGORITHM)

- SolutionExists = True
- Make a list of all variables with their domains
- For each variable, make a list of all the constraints applying to it
- Put all variables in the “to review” list
- Repeat
 - Remove the top variable from the “to review” list
 - For each of its constraints
 - Remove from the domains involved all values which are IMPOSSIBLE according to the constraint
 - If a domain for a variable is being modified, add the variable to the “to review” list (unless it is already present in the list)
 - If a domain becomes empty, set SolutionExists to False
- Until the “to review” list is empty or SolutionExists is False

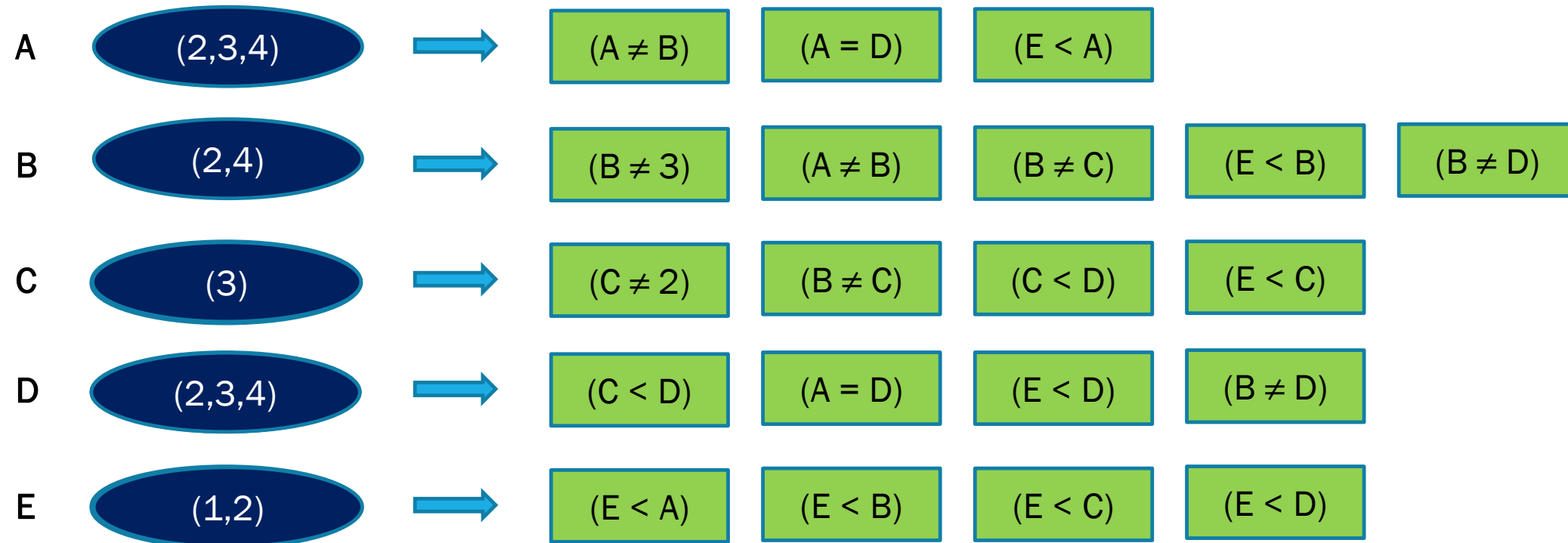
CONSTRAINT PROPAGATION



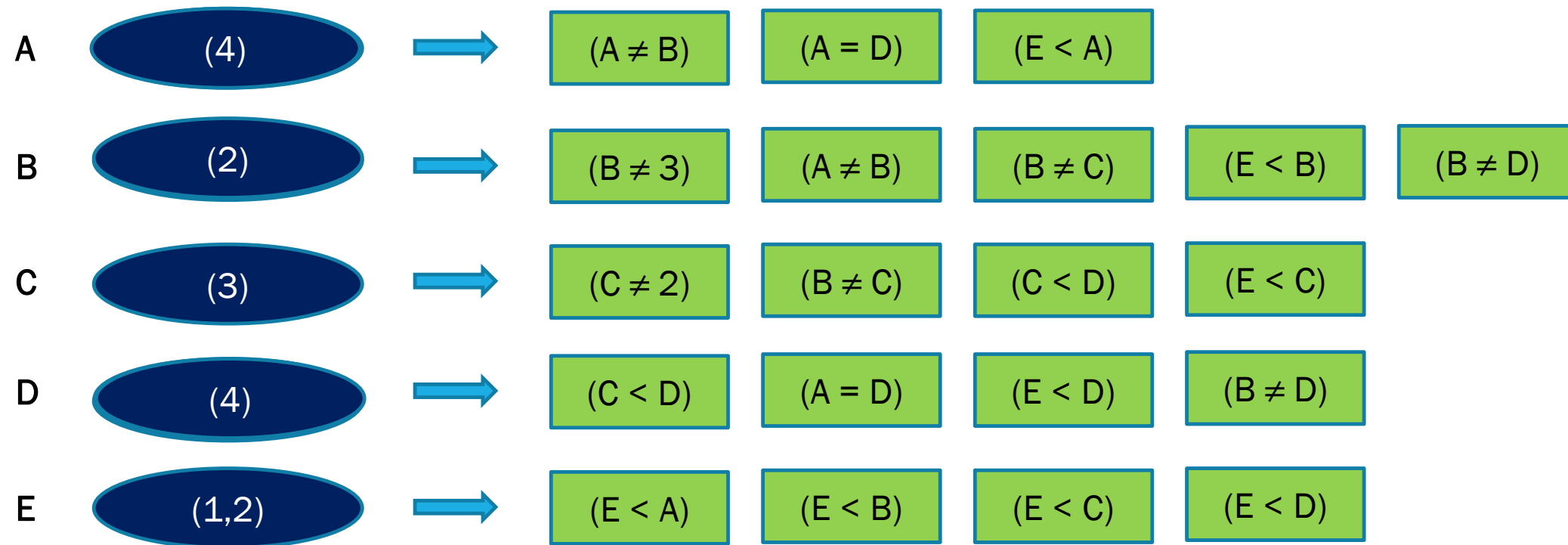
CONSTRAINT PROPAGATION



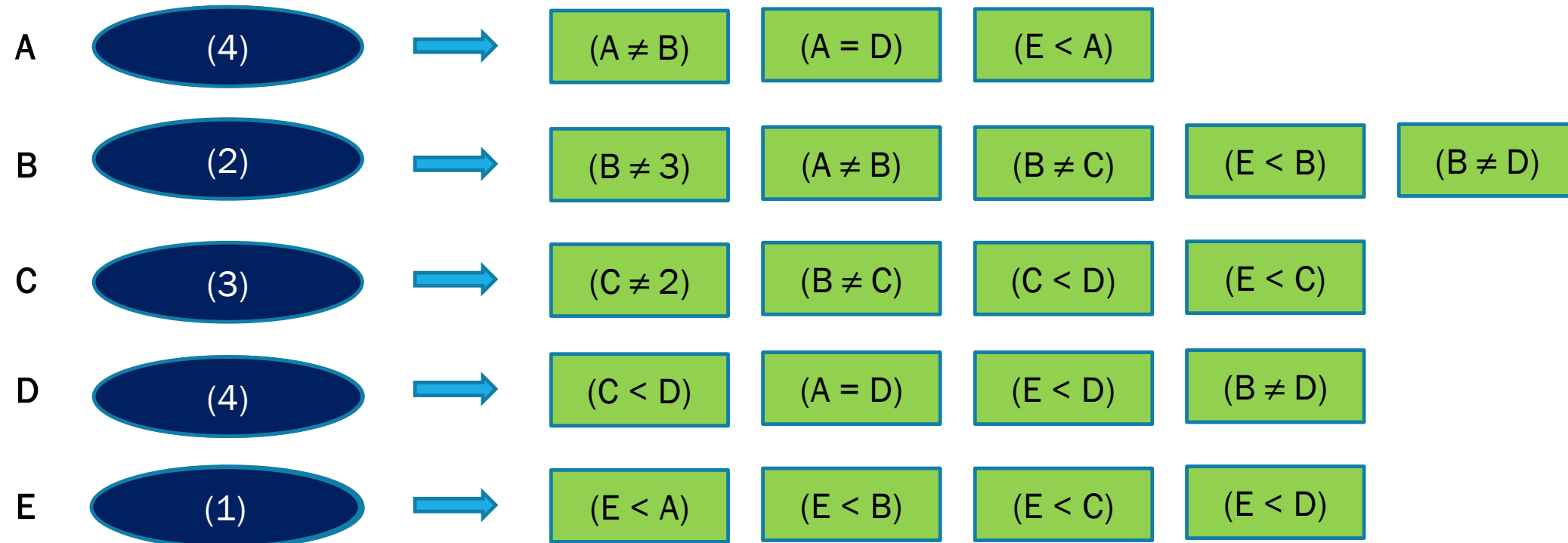
CONSTRAINT PROPAGATION



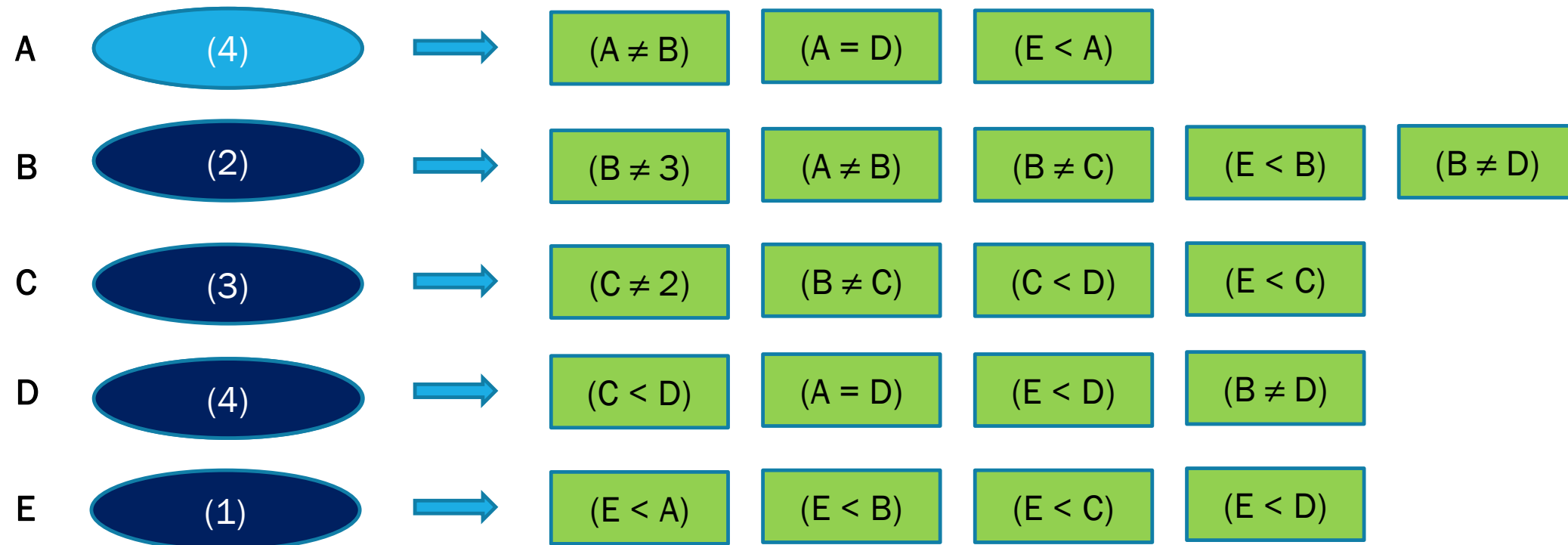
CONSTRAINT PROPAGATION



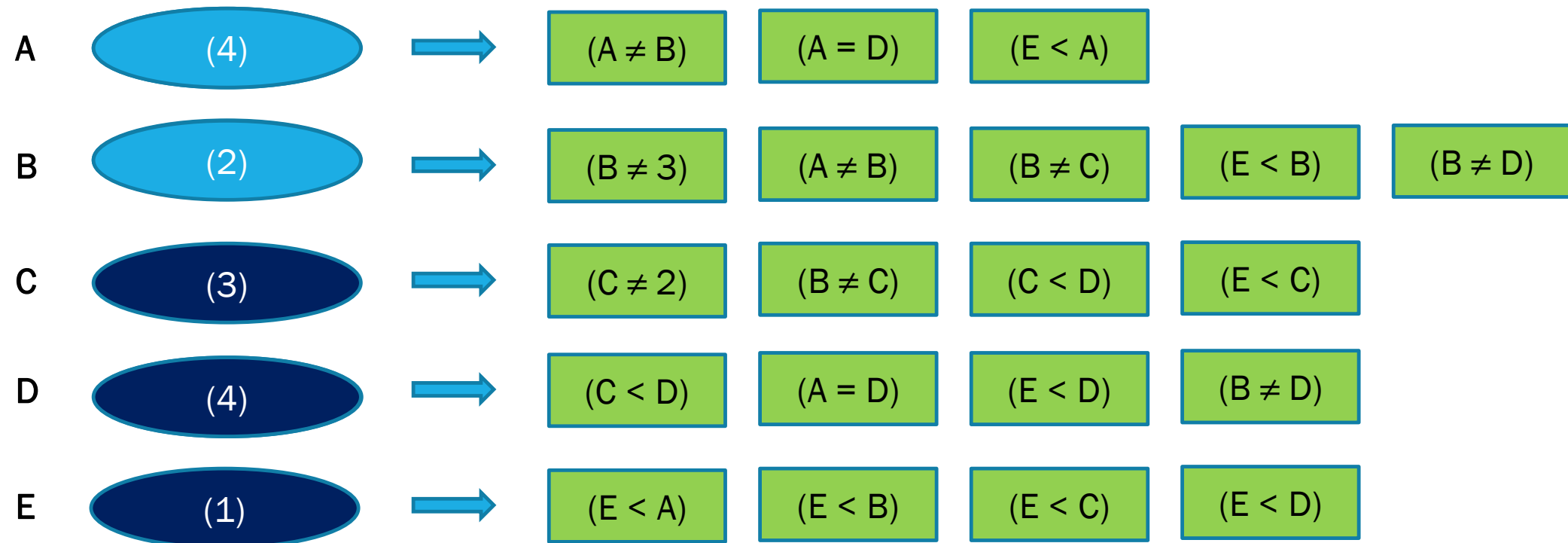
CONSTRAINT PROPAGATION



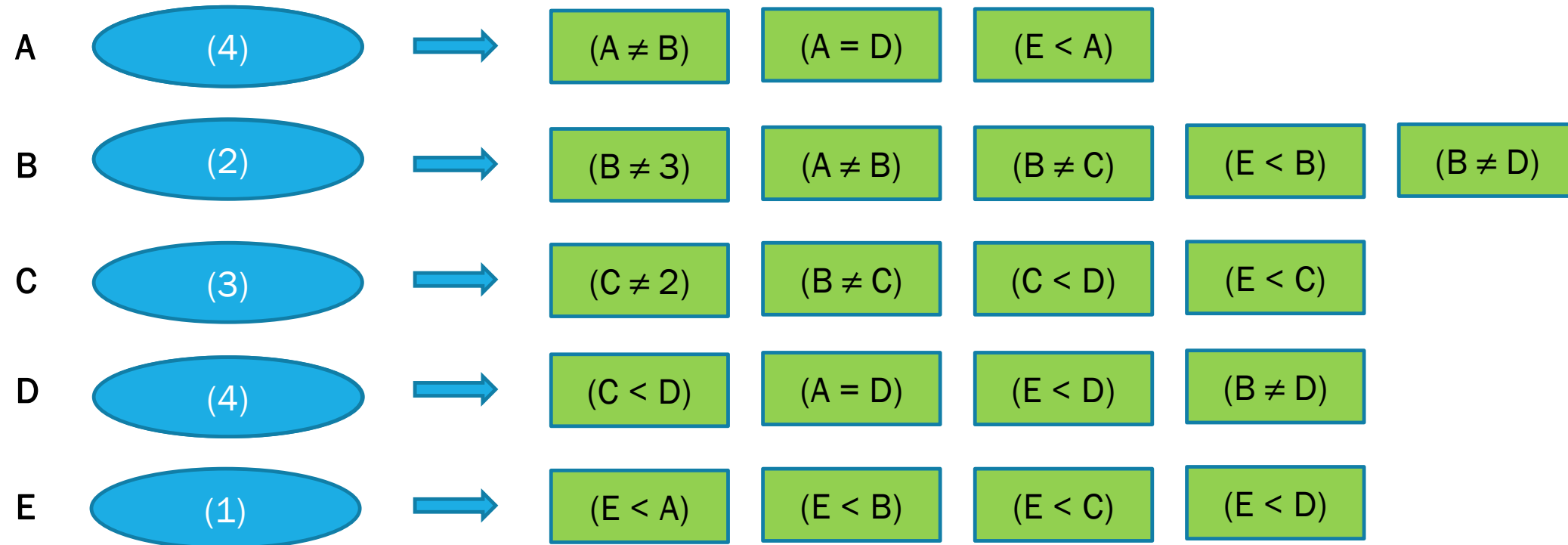
CONSTRAINT PROPAGATION



CONSTRAINT PROPAGATION



CONSTRAINT PROPAGATION



CONSTRAINT PROPAGATION (ONE VERSION OF THE ALGORITHM)

- SolutionExists = True
- Make a list of all variables with their domains
- For each variable, make a list of all the constraints applying to it
- Put all variables in the “to review” list
- Repeat
 - Remove the top variable from the “to review” list
 - For each of its constraints
 - Remove from the domains involved all values which are IMPOSSIBLE according to the constraint
 - If a domain for a variable is being modified, add the variable to the “to review” list (unless it is already present in the list)
 - If a domain becomes empty, set SolutionExists to False
- Until the “to review” list is empty or SolutionExists is False

Result of reducing the domains by constraint propagation

- Reducing each variable to a single value
 - Single solution
- Reducing one of the variables to an empty domain
 - No solution
- Reducing variables to a subset of values
 - Several solutions



IN SUMMARY

- Constraint Propagation algorithm
- Example of propagation with constraints of inequality, equality and ordering
- Types of results (no solution, single solution, multiple solutions)

Part 2

Simulated Annealing

Search space reduced

- Generate and Test
 - If the search space is small enough and we look for an optimal solution

Search space still large

- Greedy search
 - Relying on a strategy based on knowledge of the problem
- Randomized search
 - Random restart
 - Random step
 - Random modification

Unfortunately can lead to a local maxima/minima

By trying again and again... more of the search space is explored and we might find the global maxima/minima

RANDOM MODIFICATION

Generic algorithm:

- Start with an initial (random / greedy) solution
- Repeat for N iterations:
 - Make local changes
 - If better (according to cost function):
 - Keep change
 - Else
 - Keep change (according to some probability)

Variations exists on how to calculate/adjust such probability

The algorithm also keeps track of the best solution so far

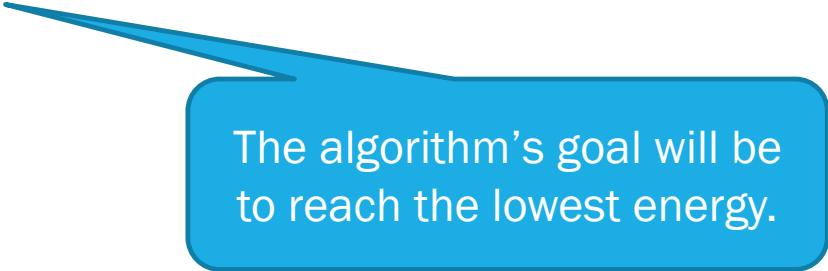
Researchers look for inspiration in the « real world »

Simulated annealing: a well-known random modification algorithm

Simulated Annealing is inspired by the process of restructuring an internal configuration of a solid which is annealed (e.g. crystallization process).

The solid is heated up to melting point, so that its particles are randomly distributed.

The material is then slowly cooled down, so that the particles reorganize in order to reach a low energy state.



The algorithm's goal will be to reach the lowest energy.

Algorithm 2: Simulated Annealing Optimizer

```
 $T \leftarrow T_{max}$   
 $\mathbf{x} \leftarrow$  generate the initial candidate solution  
 $E \leftarrow E(\mathbf{x})$  compute the energy of the initial solution  
while  $(T > T_{min})$  and  $(E > E_{th})$  do  
     $\mathbf{x}_{new} \leftarrow$  generate a new candidate solution  
     $E_{new} \leftarrow$  compute the energy of the new candidate  $\mathbf{x}_{new}$   
     $\Delta E \leftarrow E_{new} - E$   
    if Accept  $(\Delta E, T)$  then  
         $\mathbf{x} \leftarrow \mathbf{x}_{new}$   
         $E \leftarrow E_{new}$   
    end  
     $T \leftarrow \frac{T}{\alpha}$  cool the temperature  
end  
return  $\mathbf{x}$ 
```

E_{th} is a minimum energy required
(could be 0)

Rather use $r \cdot T$ where r is called
« cooling factor » and is smaller
than 1

Algorithm 1: Acceptance Function

Data: T , ΔE - the temperature and the energy variation between the new candidate solution and the current one.

Result: Boolean value that indicates if the new solution is accepted or rejected.

```
if ( $\Delta E < 0$ ) then
|   return True;
else
|    $r \leftarrow$  generate a random value in the range  $[0, 1)$ 
|   if ( $r < \exp(-\Delta E/T)$ ) then
|   |   return True
|   else
|   |   return False
|   end
end
```

IMPACT OF TEMPERATURE ON ACCEPTANCE PROBABILITY

Example with current solution = 0,1 fitness (Energy)

New solution is 0,3 – worst solution, we will decide if we keep it

Depending on the temperature (T), the table shows the probability of accepting a solution.

Cooling factor (r) is applied to T.

```

r ← generate a random value in the range [0, 1)
if (r < exp(-ΔE/T)) then
|   return True
else
|   return False
  
```

T, r=0,8	exp(-0,2/T)
1	0,81873075
0,8	0,77880078
0,64	0,73161563
0,512	0,67663385
0,4096	0,61368025
0,32768	0,54315988
0,262144	0,46629376
0,2097152	0,38532262
0,16777216	0,30358523
0,13421773	0,22534649

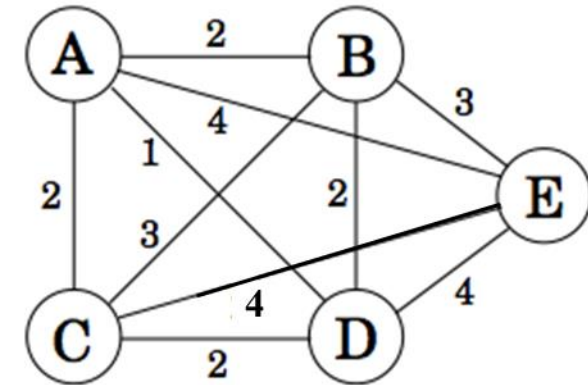
TSP PROBLEM

First solution...

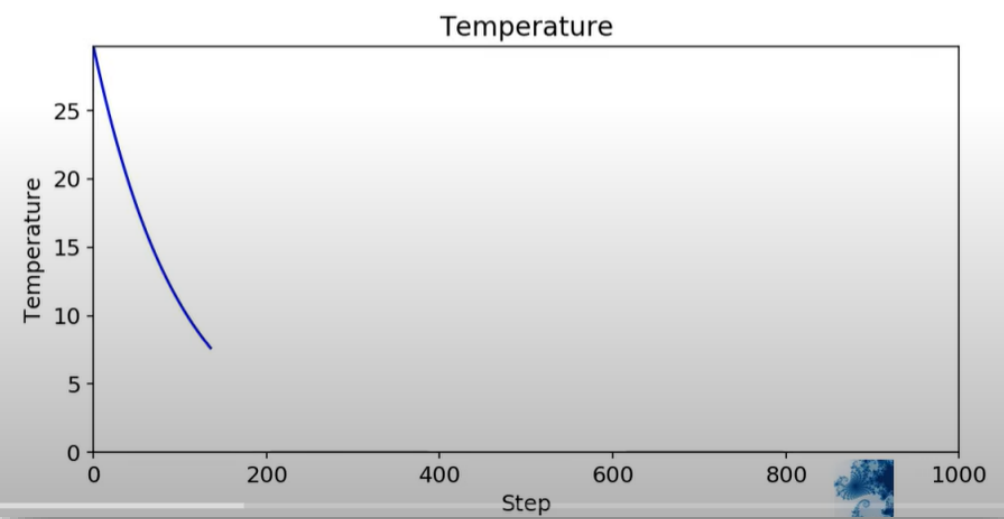
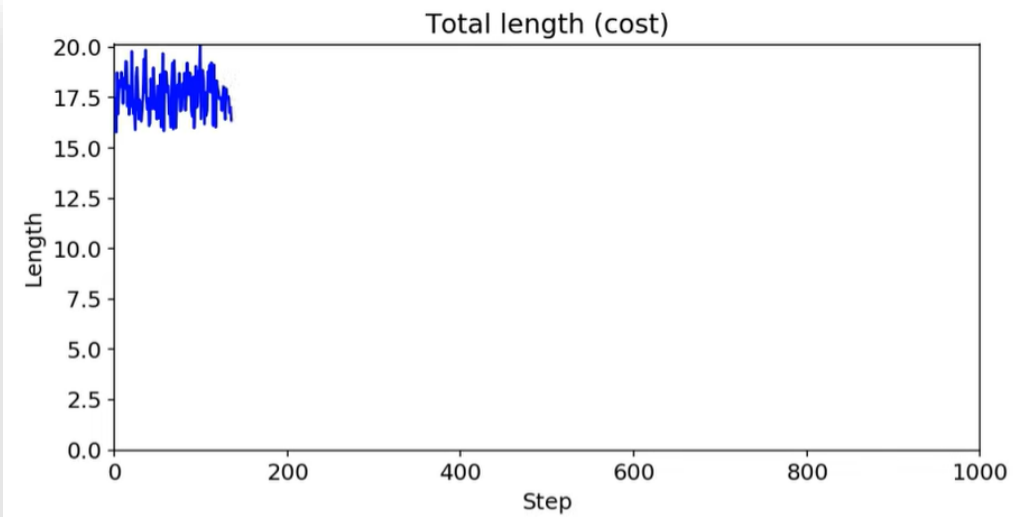
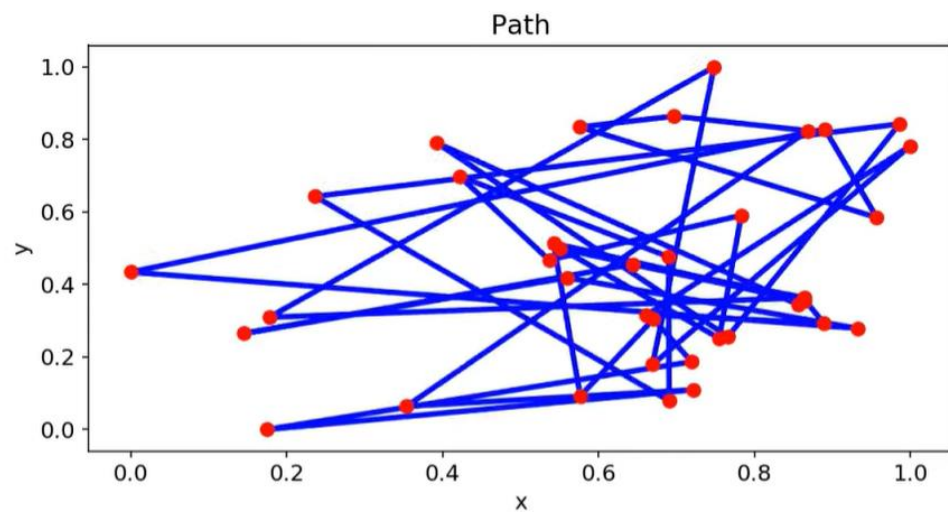
T1	T2	T3	T4	T5	retour	Coût
D	A	C	B	E	D	
	1	2	3	3	4	13

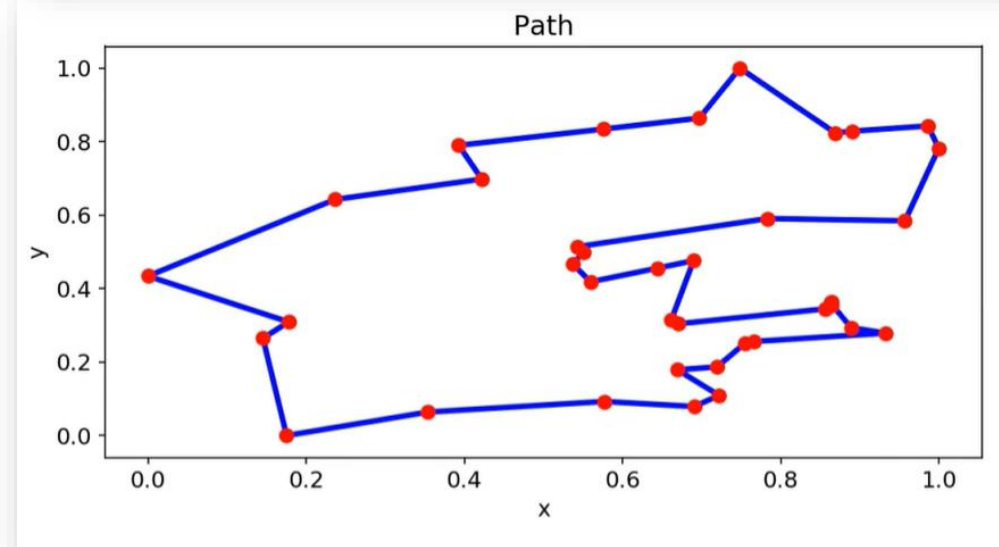
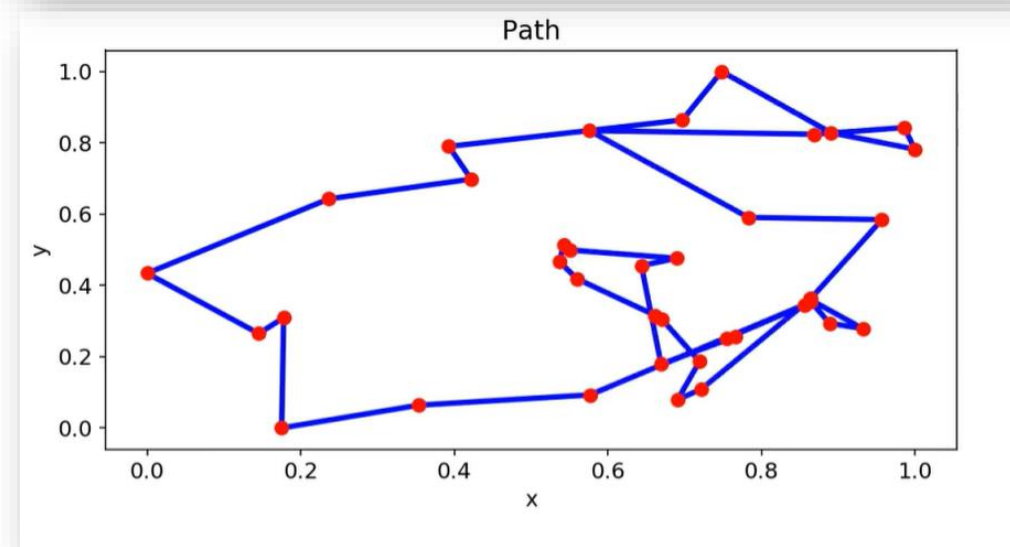
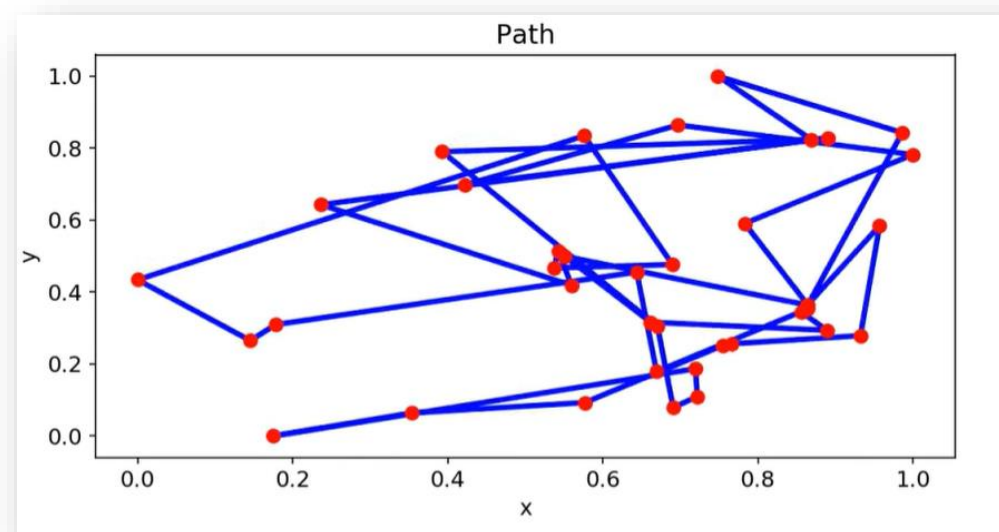
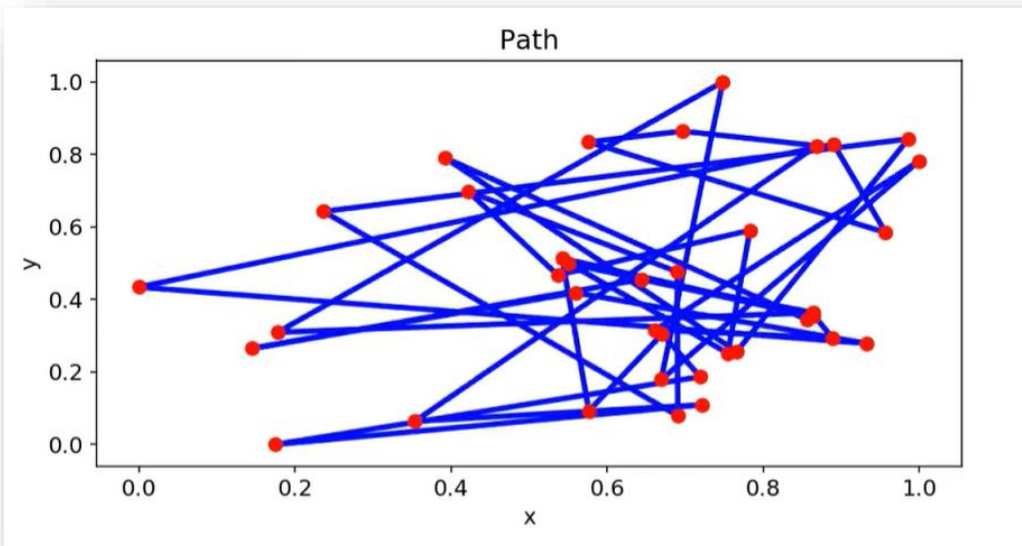
We exchange cities (modification)...

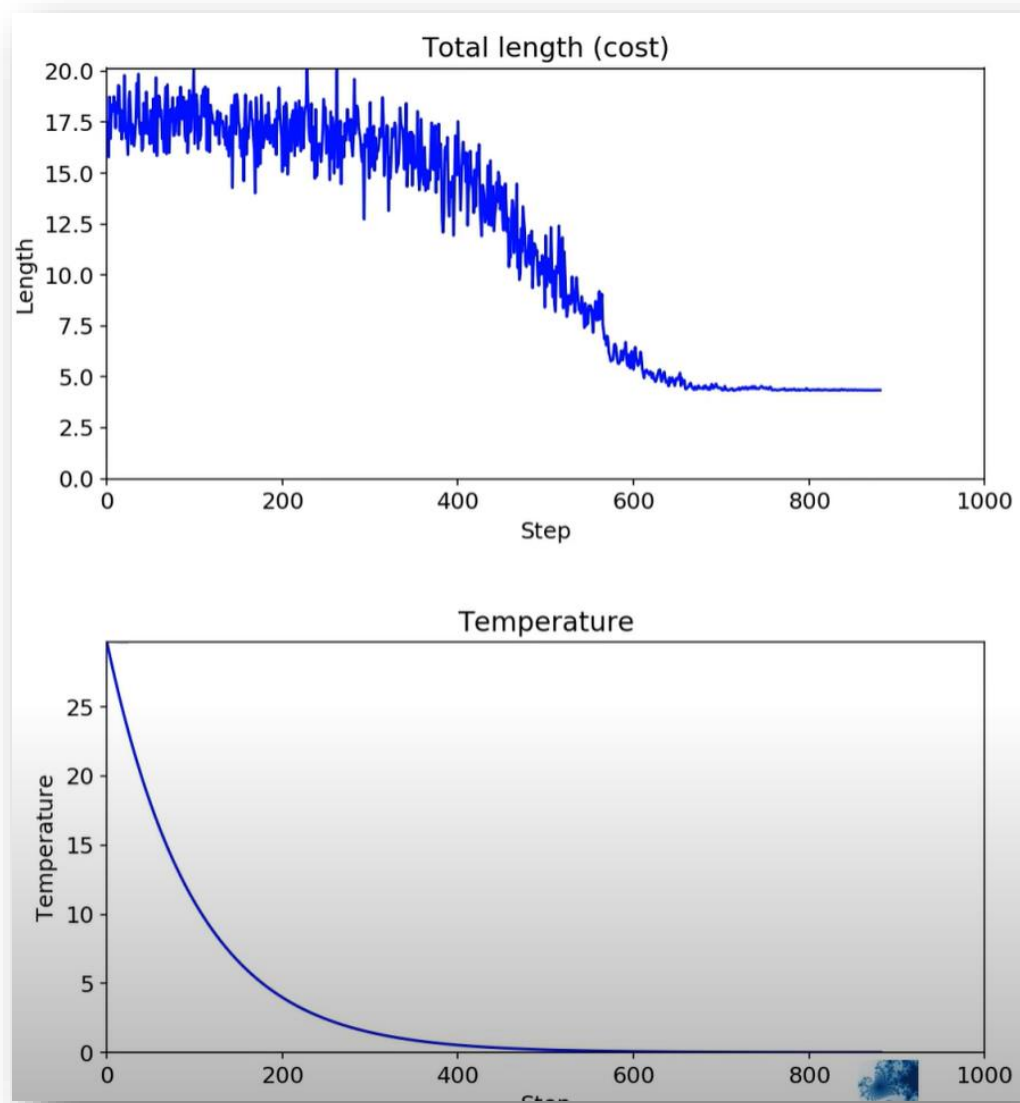
T1	T2	T3	T4	T5	retour	Coût
D	A	C	E	B	D	
	1	2	4	3	2	12

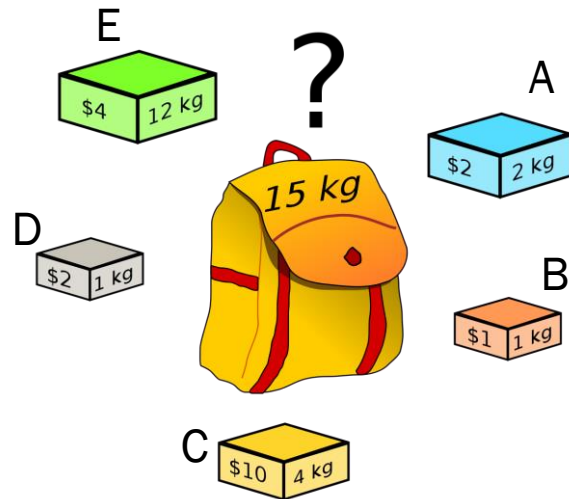


Finding a « neighbour » solution, apply modification









Finding a « neighbour » solution, apply modification

(A,B,C,D,E)
(1,0,1,1,0) – Value (Energy) = 14\$



(1,0,1,0,0) – Value (Energy) = 12\$

Attention: here you maximize... adapt the simulated annealing

Algorithm 2: Simulated Annealing Optimizer

```
 $T \leftarrow T_{max}$ 
 $\mathbf{x} \leftarrow$  generate the initial candidate solution
 $E \leftarrow E(\mathbf{x})$  compute the energy of the initial solution
while  $(T > T_{min})$  and  $(E > E_{th})$  do
     $\mathbf{x}_{new} \leftarrow$  generate a new candidate solution
     $E_{new} \leftarrow$  compute the energy of the new candidate  $\mathbf{x}_{new}$ 
     $\Delta E \leftarrow E_{new} - E$ 
    if Accept  $(\Delta E, T)$  then
         $\mathbf{x} \leftarrow \mathbf{x}_{new}$ 
         $E \leftarrow E_{new}$ 
    end
     $T \leftarrow \frac{T}{\alpha}$  cool the temperature
end
return  $\mathbf{x}$ 
```



IN SUMMARY

- Introduction to Simulated Annealing

You will further explore
during Assignment 1

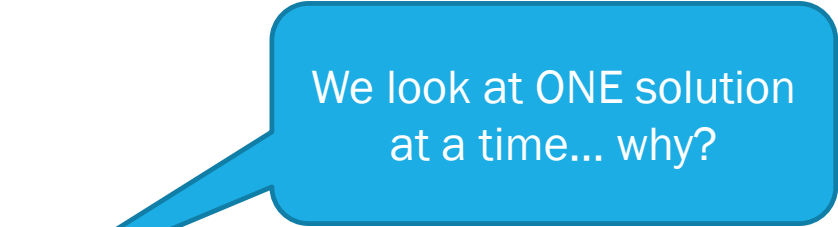
Part 3

Genetic Algorithm

RANDOM MODIFICATION

Generic algorithm:

- Start with an initial (random / greedy) solution
- Repeat for N iterations:
 - Make local changes
 - If better (according to cost function):
 - Keep change
 - Else
 - Keep change (according to some probability)



We look at ONE solution
at a time... why?

POPULATION-BASED ALGORITHM

Simple idea...

Instead of exploring one solution at a time, **explore M solutions in parallel**.
The solutions are the *individuals*, and the set of M solutions is the *population*.



EXAMPLES OF POPULATION-BASED ALGORITHMS

Algorithms inspired from social behavior or nature's behavior

- Social behavior in birds
- Rain algorithm
- Bee colony
- Ant colony
- Genetic algorithm

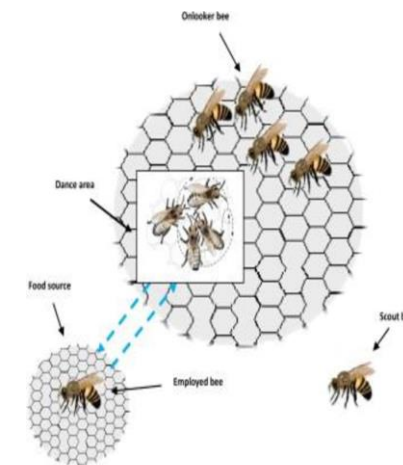


Figure 1: A Typical Bee Colony Model

Population based algorithm

- Generate a population $X(t)$ of M individuals
- Repeat
 - Select a subset of individuals according to their “fitness”
 - Perform variations on these individuals to generate new individuals who will form population $X(t + 1)$
- Until stopping condition

This selection can be done in different ways. Idea of “tournament” (tournament selection).

These variations often involve several individuals (combinations, direction by mass effect)

- (1) Max Nb iterations
- (2) No better solution within K iterations
- (3) One solution better than threshold on cost function

Let's focus on one popular algorithm:

Genetic algorithm

Algorithm inspired from genetic, including crossovers and mutations of ADN occurring during reproduction. The best adapted (according to a cost function) “parent” population will hopefully create even more adapted “children” individuals.

Cross-over

Exchange portions from both parents

Mutation

Modify an element of an individual

TSP

Initial
population

Parents

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

9	8	7	6	5	4	3	2	1
---	---	---	---	---	---	---	---	---

Offspring

					6	7	8	
--	--	--	--	--	---	---	---	--

9	5	4	3	2	6	7	8	1
---	---	---	---	---	---	---	---	---

Crossover

1-point crossover – single split
2-point crossover – 2 splits

Mutation

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

1	2	8	4	5	6	7	3	9
---	---	---	---	---	---	---	---	---

Performed after crossover

Population based algorithm – Genetic algorithm

- Generate a population $X(t)$ of M individuals
- Repeat
 - Select a subset of individuals according to their “fitness”
 - Perform variations on these individuals to generate new individuals who will form population $X(t + 1)$
- Until stopping condition

This selection can be done in different ways. Idea of “tournament” (tournament selection).

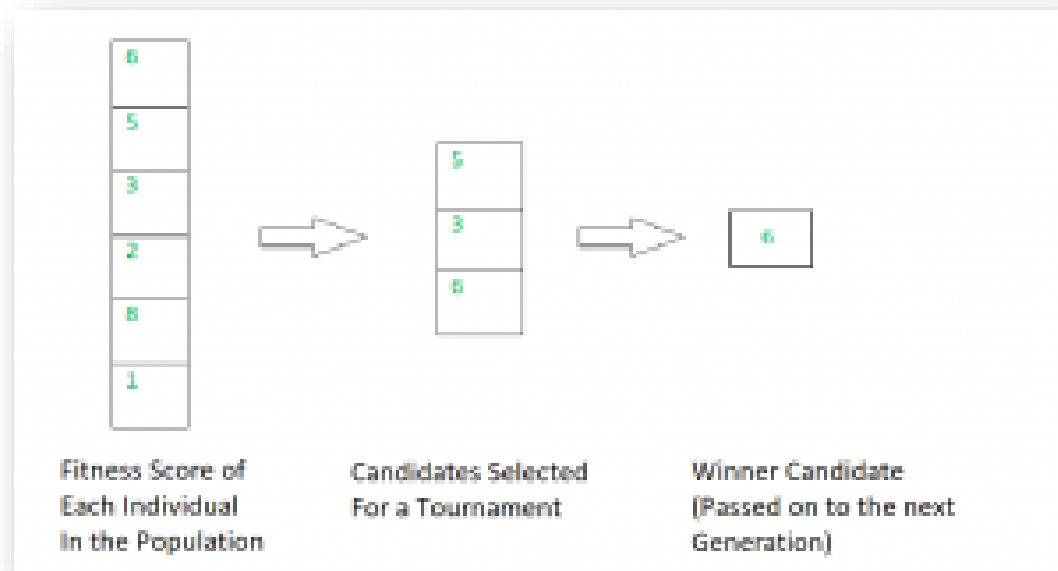
These variations are the crossovers and mutations. Probabilities (e.g. 0.7 crossovers, 0.1 mutations) are used.

- (1) Max Nb iterations
- (2) No better solution within K iterations
- (3) One solution better than threshold on cost function

Tournament-Based Selection

Algorithm --

1. Select k individuals from the population and perform a tournament amongst them
2. Select the best individual from the k individuals
3. Repeat process 1 and 2 until you have the desired amount of population



Variations:

- With or without replacement
- Deterministic or probabilistic
- Size of K

Tournament-Based Selection

```
choose k (the tournament size) individuals from the population at random
choose the best individual from the tournament with probability p
choose the second best individual with probability  $p \cdot (1-p)$ 
choose the third best individual with probability  $p \cdot (1-p)^2$ 
and so on
```

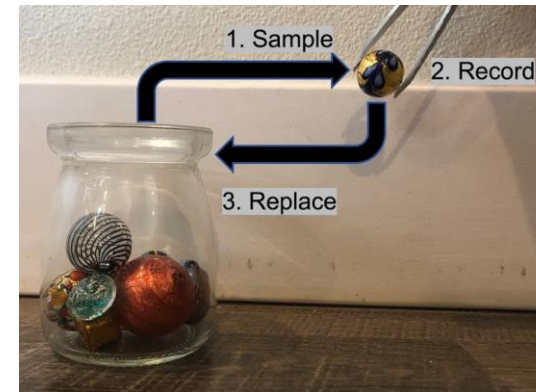
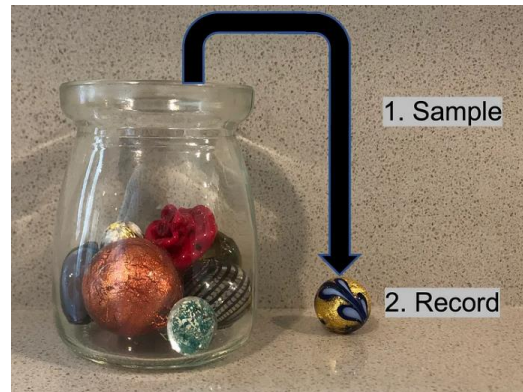
Deterministic: ($p = 1$) so the fittest is taken in each tournament

Probabilistic: for example $p=0.8$, then for one tournament

- 80% probability to take best
- 16% probability to take second best
- 3.2% probability to take 3rd best
- ...

Tournament-Based Selection

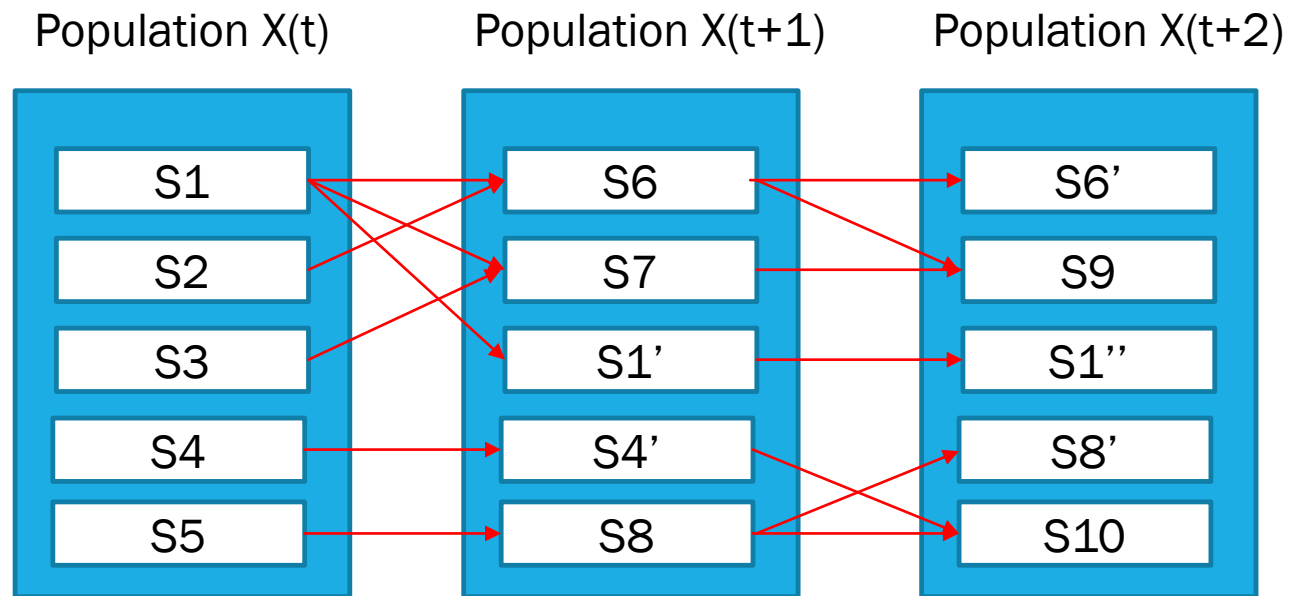
With or without replacement



Population based algorithm – Genetic algorithm

- Generate a population $X(t)$ of M individuals
- Repeat
 - Select a subset of individuals according to their “fitness”
 - Perform variations on these individuals to generate new individuals who will form population $X(t + 1)$
- Until stopping condition

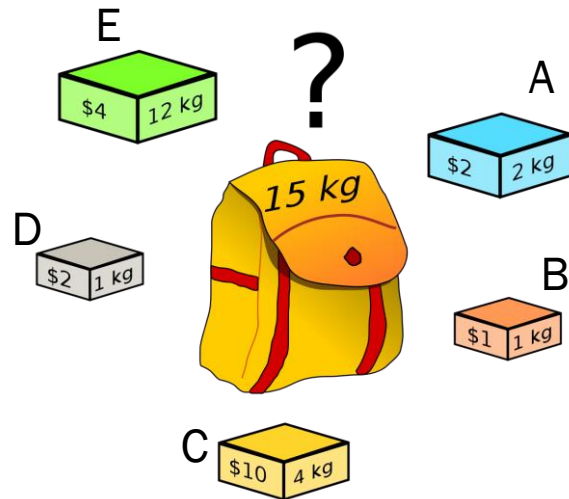
GENETIC ALGORITHM



Cross-overs
(S1/S2, S1/S3)
Mutations (S5)

Cross-overs
(S6/S7, S4'/S8)
Mutations (S6, S8)

KNAPSACK PROBLEM



Variables: (A,B,C,D,E)

S1 : (1,1,0 // 0,1)

S2 : (1,0,1 // 1,0)



Crossover (e.g. rate 0.7)

S1': (1,1,0,0,1)

S2': (1,0,1,1,0)



Mutation
(e.g. rate 0.1 only S1'
modified)

S1': (1,0,0,0,1)

S2': (1,0,1,1,0)



IN SUMMARY

- Introduction to Population-Based Algorithms
- Genetic Algorithm

You will further explore
during Assignment 1



EXPLORING THE SOLUTION SPACE

- Part 1 – Constraint Propagation
- Part 2 – Simulated Annealing
- Part 3 – Genetic Algorithm