Docker setup

You can start

```
docker pull registry.gitlab.inria.fr/soliman/inf555/td9
```

now

Local Search & Constraint Satisfaction Problems

Sylvain Soliman

November 21st, 2018

Thanks to P. Flener, L. Michel and P. Van Hentenryck for inspiration

What is “Local Search”

- iterative **optimization** method
- looking for an assignment of variables to values of their domains that minimizes some *cost*
- **local** move from solution to *neighboring* solution
- try to (always or not) decrease the cost of the selected assignment

How does this relate to Constraint Solving?

Compared to what we have seen up to now:

- only **optimization**
- requires defining:
  - **neighborhood** and
  - **selection criterion** (*single* state)
  - **stopping criterion**

$\Rightarrow$ **incomplete** (i.e., not optimal) but **low cost** (time and memory)

Constraint-Based Local Search

A pure CSP can be transformed easily into a LS problem:

Constraint-Based Local Search

A pure CSP can be transformed easily into a LS problem:

Use **constraint violations** as *cost function*

Some constraints can be given an infinite cost, these are *hard* constraints that all states have to satisfy

Other are soft constraints that will guide the search
Allows us to solve *over-constrained* problems

Neighborhood is defined as changing **a single variable assignment**

In that framework redundant constraints play two roles:

Constraint-Based Local Search

A pure CSP can be transformed easily into a LS problem:

Use **constraint violations** as *cost function*

Some constraints can be given an infinite cost, these are *hard* constraints that all states have to satisfy

Other are soft constraints that will guide the search
Allows us to solve *over-constrained* problems

Neighborhood is defined as changing **a single variable assignment**

In that framework redundant constraints play two roles: propagation and

Constraint-Based Local Search

A pure CSP can be transformed easily into a LS problem:

Use **constraint violations** as *cost function*

Some constraints can be given an infinite cost, these are *hard* constraints that all states have to satisfy

Other are soft constraints that will guide the search
Allows us to solve *over-constrained* problems

Neighborhood is defined as changing **a single variable assignment**

In that framework redundant constraints play two roles: propagation and search

Violations

Violations defined on basic constraints

Can be **composed**:

- $V(c_1 \wedge c_2) = V(c_1) + V(c_2)$
- $V(c_1 \vee c_2) = \min(V(c_1), V(c_2))$

- $V(\neg c) = 1 - \min(1, V(c))$

Violations

Violations defined on basic constraints

Can be **composed**:

- $V(c_1 \wedge c_2) = V(c_1) + V(c_2)$
- $V(c_1 \vee c_2) = \min(V(c_1), V(c_2))$

- $V(\neg c) = 1 - \min(1, V(c))$ $\Rightarrow$ not compatible with the above!

For global constraints, one can *decompose* or not

N-Queens as a Local Search problem

```
constraint alldifferent(q);
constraint alldifferent([q[i] + i | i in 1..n]);
constraint alldifferent([q[i] - i | i in 1..n]);
```

Count violations as the **total** number identical pairs in an `alldifferent` constraint

Very dependent on the model! (dual, symmetries, etc.)
But the state space does depend too!

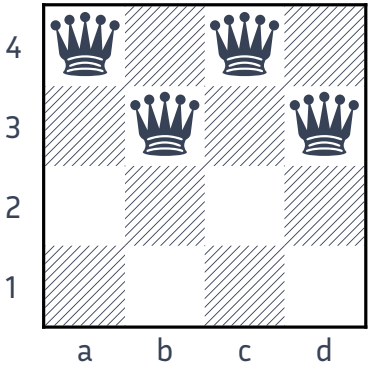
N-Queens as a Local Search problem

```
constraint alldifferent(q);
constraint alldifferent([q[i] + i | i in 1..n]);
constraint alldifferent([q[i] - i | i in 1..n]);
```

Count violations as the **total** number identical pairs in an `alldifferent` constraint

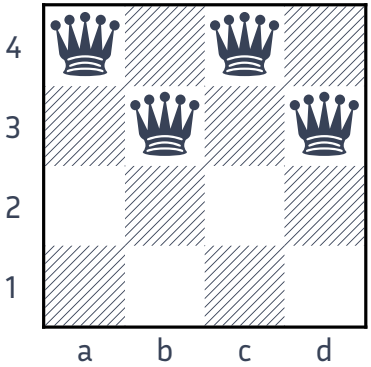
Very dependent on the model! (dual, symmetries, etc.)
But the state space does depend too!

Could use the **max** number of equalities instead of their sum
try to guide the search as much as possible



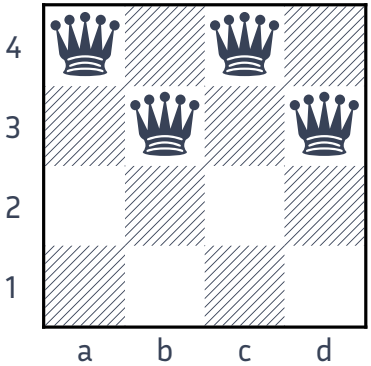
`alldifferent` lists:

$\rightarrow$



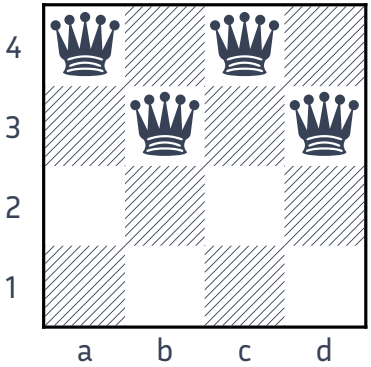
`alldifferent` lists:

$\rightarrow [4, 3, 4, 3] \quad \searrow$



`alldifferent` lists:

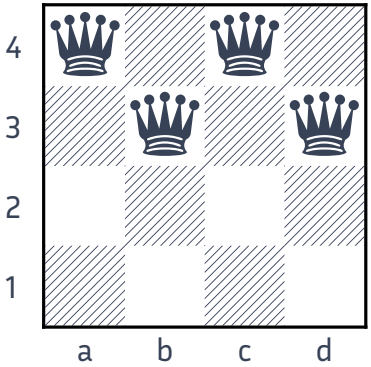
$\rightarrow[4,3,4,3]$ $\searrow[4,4,6,6]$ $\nearrow$



`alldifferent` lists:

$\rightarrow[4,3,4,3]$ $\searrow[4,4,6,6]$ $\nearrow[4,2,2,0]$

Violations:

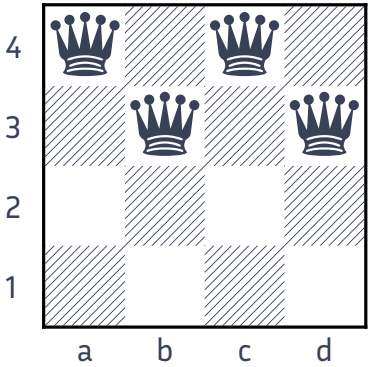


`alldifferent` lists:

$\rightarrow[4,3,4,3]$ $\searrow[4,4,6,6]$ $\nearrow[4,2,2,0]$

Violations:

2+

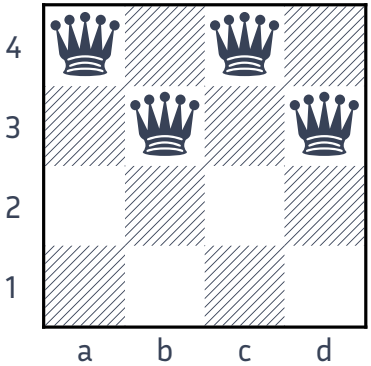


`alldifferent` lists:

$\rightarrow[4,3,4,3]$ $\searrow[4,4,6,6]$ $\nearrow[4,2,2,0]$

Violations:

2+2+



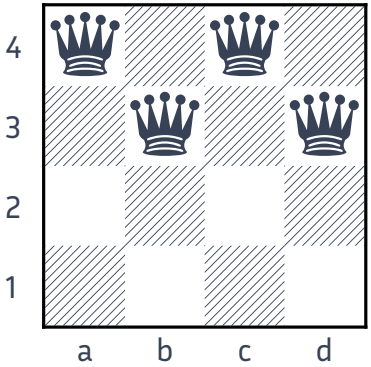
`alldifferent` lists:

$\rightarrow[4,3,4,3]$ $\searrow[4,4,6,6]$ $\nearrow[4,2,2,0]$

Violations:

$2+2+1 = 5$

Neighborhood:



`alldifferent` lists:

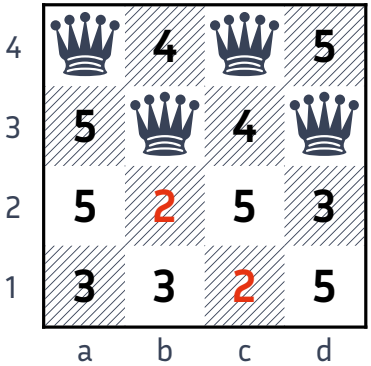
$\rightarrow[4,3,4,3]$ $\searrow[4,4,6,6]$ $\nearrow[4,2,2,0]$

Violations:

$2+2+1 = 5$

Neighborhood:

Move **one queen** in its column
i.e., change the valuation of a single variable



`alldifferent` lists:

$\rightarrow[4,3,4,3]$ $\searrow[4,4,6,6]$ $\nearrow[4,2,2,0]$

Violations:

$2+2+1 = 5$

Neighborhood:

Move **one queen** in its column
i.e., change the valuation of a single variable

Example: Greedy Local Search (aka. Hill-climbing)

Pure **exploitation** (intensification)

Analog for the discrete case of gradient descent

Select the *most improving* neighbor (random if multiple bests)

Stop when no improvement found

Example: Min. Conflict Search (MCS) / Heuristics (MCH)

Original heuristics for CBLS on SAT problems (and still part of GSAT, WalkSAT, etc.)

Implemented by default in the COMET system

Basis of most other heuristics

Select the neighbor (i.e., variable assignment) that minimizes the **number** of violated constraints

Example: Min. Conflict Search (MCS) / Heuristics (MCH)

Original heuristics for CBLS on SAT problems (and still part of GSAT, WalkSAT, etc.)

Implemented by default in the COMET system

Basis of most other heuristics

Select the neighbor (i.e., variable assignment) that minimizes the **number** of violated constraints

IOW, Hill-Climbing with violations saturated at 1

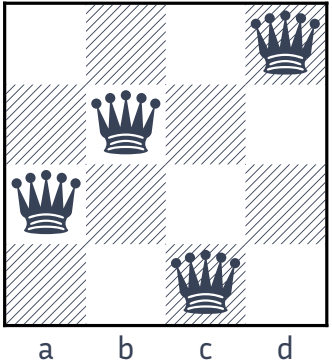
Issues

Local extrema

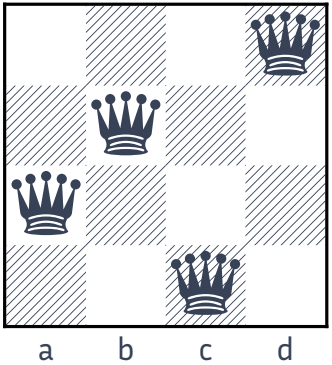
Plateaus

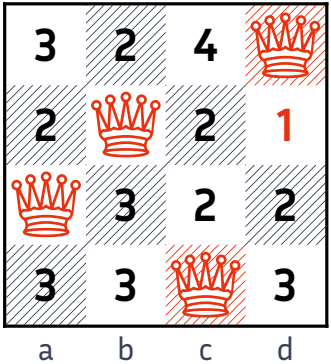
Diagonal ridges (i.e., moves of same cost leading to different extrema)

Big neighborhood (might require two steps: **variable** and then **value** selection)



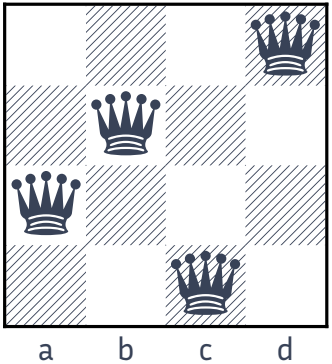
Violation cost =



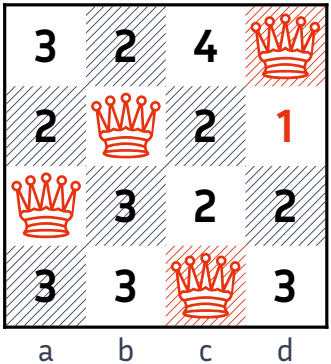


Violation cost = 1

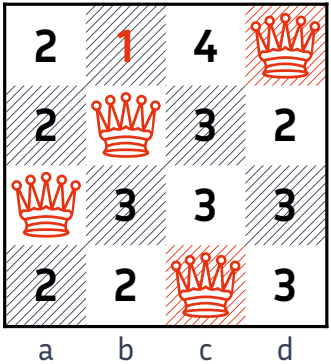
Line-wise costs



Violation cost = 1



Line-wise costs



Column-wise costs

	a	b	c	d	e	f	g	h
8	4	♕	2	3	3	2	4	3
7	4	4	3	3	3	3	3	♕
6	1	2	♕	2	2	3	2	2
5	3	3	3	5	4	3	♕	3
4	3	3	3	♕	5	4	3	4
3	1	2	3	2	3	♕	3	1
2	♕	3	3	4	3	4	4	3
1	3	4	3	3	♕	2	3	4

Line-wise costs

Real local optimum

Necessary to get through a *worse* solution to get to a global optimum

Example: Random walk

Pure **exploration** (diversification)

Select *a random* neighbor

Remember the best solution found

Stop after a given number of iterations

Being smarter

$\Rightarrow$ combine a way to escape local extrema with some Hill-climbing

Some examples:

- diagonal moves (see *Practical work session*)

Being smarter

$\Rightarrow$ combine a way to escape local extrema with some Hill-climbing

Some examples:

- diagonal moves (see *Practical work session*)
- Simulated annealing (*idem*)

Being smarter

$\Rightarrow$ combine a way to escape local extrema with some Hill-climbing

Some examples:

- diagonal moves (see *Practical work session*)
- Simulated annealing (*idem*)
- Tabu search

Being smarter

$\Rightarrow$ combine a way to escape local extrema with some Hill-climbing

Some examples:

- diagonal moves (see *Practical work session*)
- Simulated annealing (*idem*)
- Tabu search
- random **restarts** (underrated!)

Being smarter

$\Rightarrow$ combine a way to escape local extrema with some Hill-climbing

Some examples:

- diagonal moves (see *Practical work session*)
- Simulated annealing (*idem*)
- Tabu search
- random **restarts** (underrated!)

What if *restarts were actually done simultaneously*?
population-based approaches

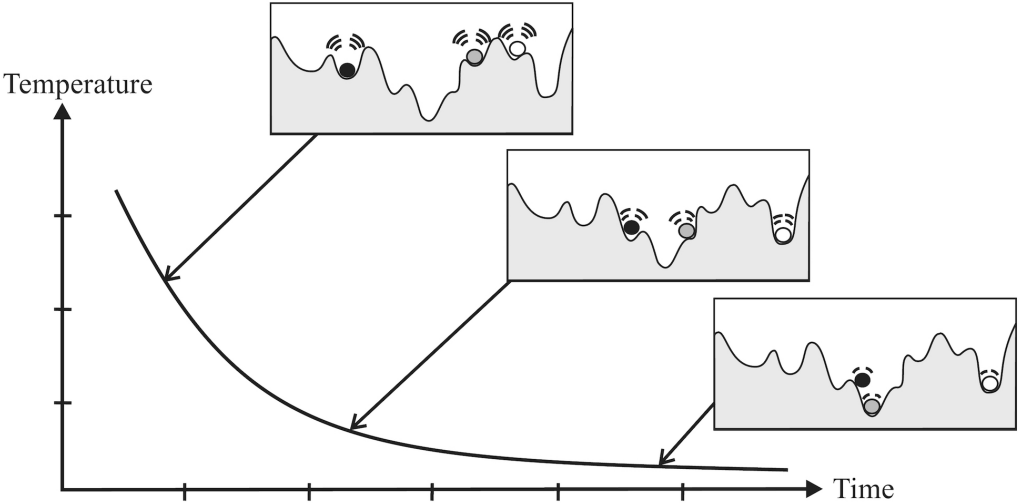
Simulated annealing

Inspiration from the physics' world (Metropolis-Hastings algorithm for a sampling states of a thermodynamic system, 1953)

Allow some *exploration* while the **temperature** of the system is high

Decrease temperature with time (i.e., iterations)

Focus on *exploitation* when the system cools down



Practically

At each step:

- select a **random neighbor** (no guidance at all...)
- compare its cost with the current cost
- accept it or not depending on the temperature but always accept improving moves
- stop if the move was rejected and the temperature too low (return the best solution found)

Parameters: initial/stopping temperature, cooling regime, acceptance condition

Parameters

Initial temperature:

Parameters

Initial temperature: allow any move (**random-walk**)

Ending temperature:

Parameters

Initial temperature: allow any move (**random-walk**)

Ending temperature: would reject most non-improving moves (**hill-climbing**)

Cooling regime:

Parameters

Initial temperature: allow any move (**random-walk**)

Ending temperature: would reject most non-improving moves (**hill-climbing**)

Cooling regime: observed to have almost *no impact*
usually $T_{t+1} = (1 - c)T_t$ with a small cooling-rate, e.g., $c = 0.01$

Acceptance condition:

Parameters

Initial temperature: allow any move (**random-walk**)

Ending temperature: would reject most non-improving moves (**hill-climbing**)

Cooling regime: observed to have almost *no impact*
usually $T_{t+1} = (1 - c)T_t$ with a small cooling-rate, e.g., $c = 0.01$

Acceptance condition: mostly coming from physics
consensus: $\exp(\Delta/T) > r$, Δ is current cost minus new cost,
r is a random variable in $[0, 1)$

Results

Very often used as a basic LS algorithm

Decent results on the Traveling Salesman Problem
not for finding an optimal solution but a *good* solution

On the N-Queens problem...

Results

Very often used as a basic LS algorithm

Decent results on the Traveling Salesman Problem
not for finding an optimal solution but a *good* solution

On the N-Queens problem...
we'll see during the *practical work session*

Not always easy to fine-tune temperature

Tabu search

Created by F. Glover in 1986

At its core: Hill-Climbing with some kind of memory forbidding moves

At each step select the **best neighbor** that is **not tabu**

Stop after a given number of iterations

Tabu moves force diversification

Tabu search

Created by F. Glover in 1986

At its core: Hill-Climbing with some kind of memory forbidding moves

At each step select the **best neighbor** that is **not tabu**

Stop after a given number of iterations

Tabu moves force diversification in a more *guided* way than random-walks

Types of memory

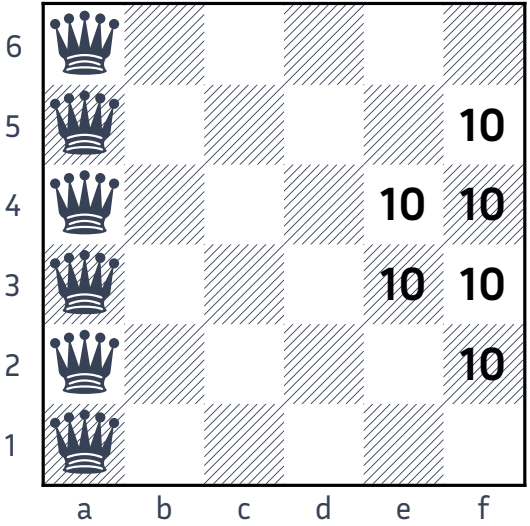
The *Tabu-list* is usually a list of recently visited states, to avoid cycles (classical issue with random diagonal moves)

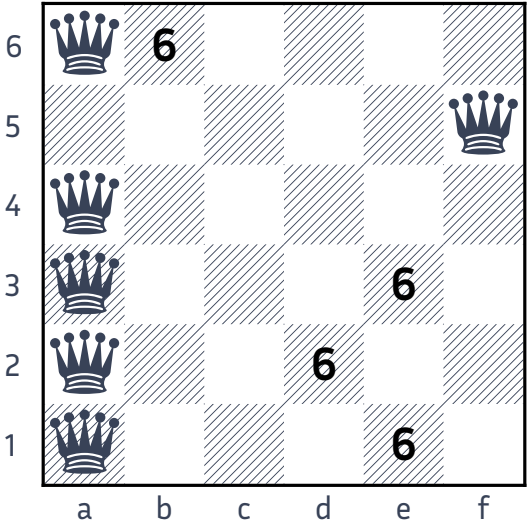
Its length is a **sensitive parameter**

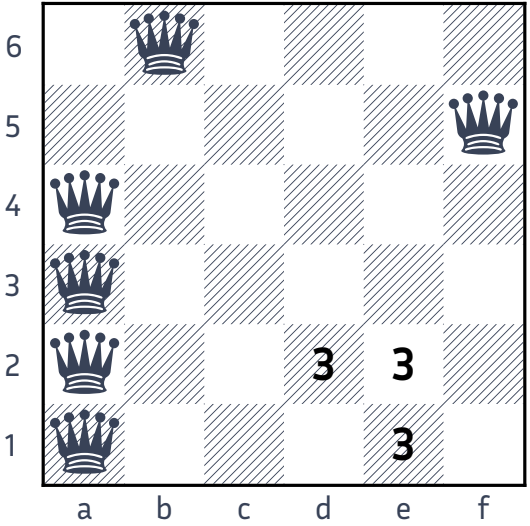
Sometimes not a full state but a *feature* is stored

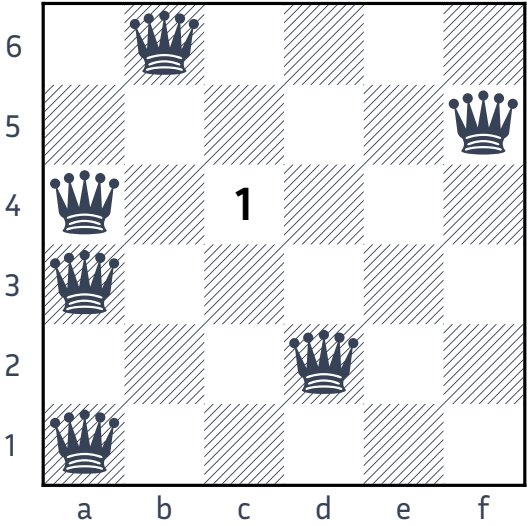
in that case it might be necessary to overcome *tabu* when a better candidate is found

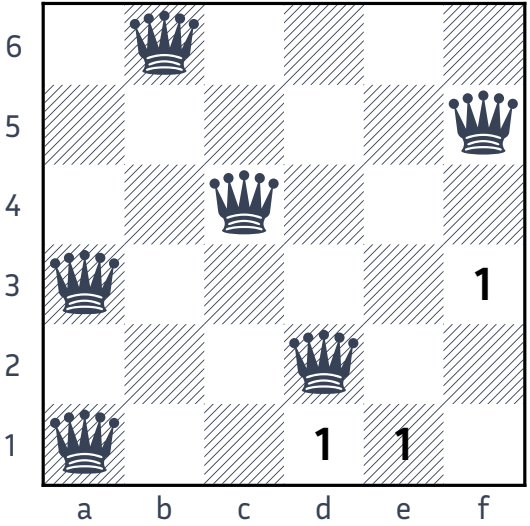
Intermediate-term memory (intensification) and Long-term memory (diversification) rules can be added

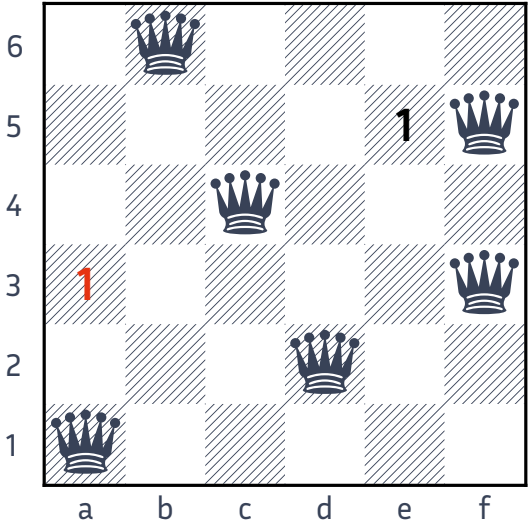

6
5
4
3
2
1
10
10
10
10
10
10
a
b
c
d
e
f

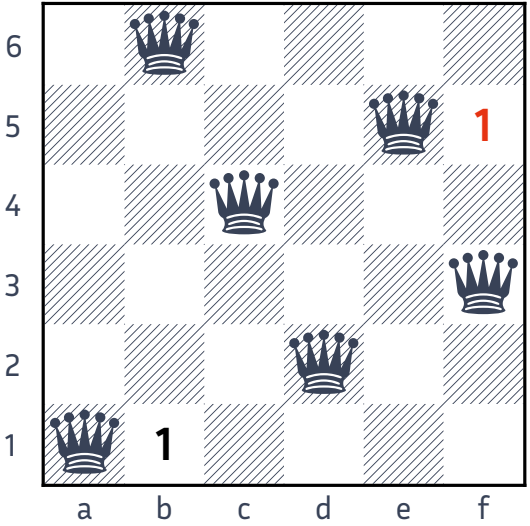

6
5
4
3
2
1
a
b
c
d
e
f
6
6
6
6

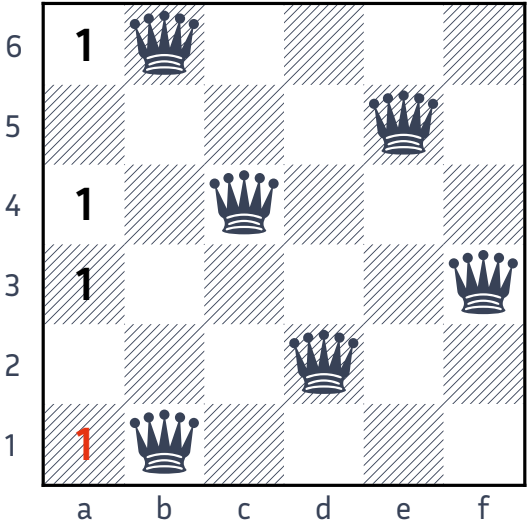

6
5
4
3
2
1
3
3
3
a
b
c
d
e
f

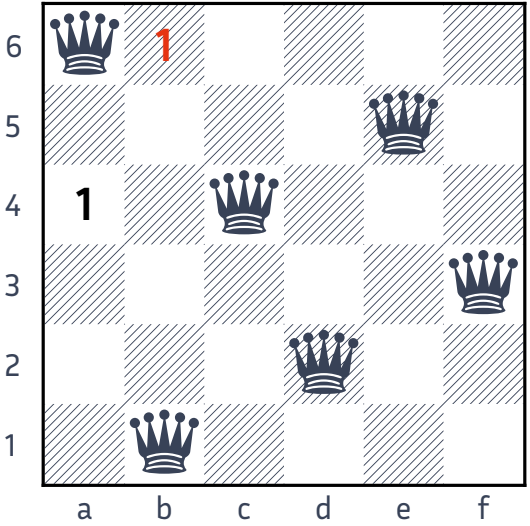

6
5
4
3
2
1
1
a
b
c
d
e
f

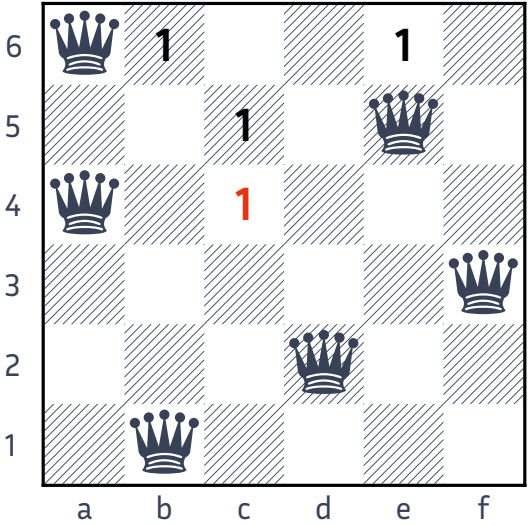

6
5
4
3
2
1
1
1
1
a
b
c
d
e
f

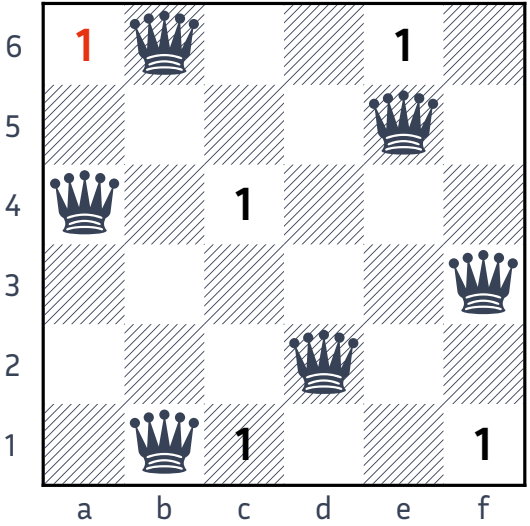

6
5
4
3
2
1
1
1
a
b
c
d
e
f

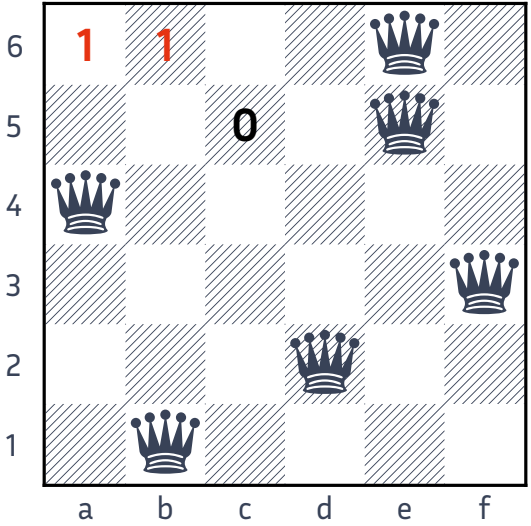

6
5
4
3
2
1
1
1
a
b
c
d
e
f

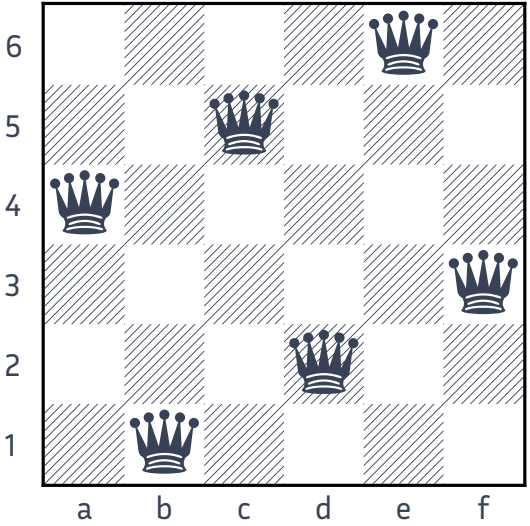

6
5
4
3
2
1
1
1
1
1
a
b
c
d
e
f


6
5
4
3
2
1
1
1
a
b
c
d
e
f


6
5
4
3
2
1
a
b
c
d
e
f
1
1
1
1


6
5
4
3
2
1
1
1
1
1
1
a
b
c
d
e
f


6
5
4
3
2
1
1
1
0
a
b
c
d
e
f


6
5
4
3
2
1
a
b
c
d
e
f

Results

Same as SA but better/worse

More *guided*, but more parameters

Good results on TSP

Even trickier to fine-tune (especially complex Tabu structures/rules)

Population-based approaches

Many versions: Genetic algorithms, Particle Swarm Optimization, Ant-Colony Optimization, etc.

Main idea: instead of restarts, use the information of parallel runs *while* they are running

Balance between exploration/exploitation, diversification/intensification remains hard

Conclusion

Incomplete but **efficient** method, even with *simple algorithms* (e.g., Hill-climbing with restarts and diagonal moves)

Used for **hard** problems for which a complete search is not tractable (e.g., Ant Colony on graph problems by C. Solnon)

Or for problems that are **over-constrained** (and can be expressed as optimization) (e.g., MaxSAT)

Can be made generic for CSPs once violations are defined

Remains often tricky to parametrize
Does **never prove optimality**

Previously on INF555...

1. Introduction to **CSPs** and to the **modeling** language MiniZinc
2. Boolean satisfiability, **SAT** solvers
3. Polynomial complexity classes in SAT, phase transitions in random k-SAT
4. Constraint propagation and **domain filtering** algorithms
5. Search and **heuristics**
6. Global constraints
7. Symmetries
8. **Arithmetic** Constraints
9. Constraint-Based Local Search

Take-home message

Holy grail: *Programming = Modeling*

Take-home message

Holy grail: *Programming = Modeling*

$\Longrightarrow$ actually not so far...

Take-home message

Holy grail: *Programming = Modeling*

$\Longrightarrow$ actually not so far...

many available solvers efficient on many hard problems

even if none can be efficient on all (unless $P = NP$)

Keep CP in mind and you'll see **problems you can solve everywhere!**

Take-home message

Holy grail: *Programming = Modeling*

$\Longrightarrow$ actually not so far...

many available solvers efficient on many hard problems

even if none can be efficient on all (unless $P = NP$)

Keep CP in mind and you'll see **problems you can solve everywhere!**

But be careful:
"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail." — A. Maslow

Docker setup

You can start

```
docker pull registry.gitlab.inria.fr/soliman/inf555/td9
```

now