Stochastic Local Search (SLS)

(Version of 22nd January 2026)

Pierre Flener

Department of Information Technology
Computing Science Division
Uppsala University
Sweden

Course 1DL481:
Algorithms and Data Structures 3 (AD3)

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

Outline

Example: Sudoku: The Problem

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

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

A Sudoku is a 9-by-9 array of integers in the domain $\{1, 2, \ldots, 9\}$.
Some of the elements are provided as hints.
The remaining elements are (decision) variables that are constrained so that the elements in each row, column, and boldfaced 3-by-3 block are distinct.

Example: Sudoku: Initialisation, Probing, and Moving

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

Randomised **initial assignment** of the variables, and thus the first **current assignment**.

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

Just a few variables (= two variables, here) are picked for modification in each candidate move.

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

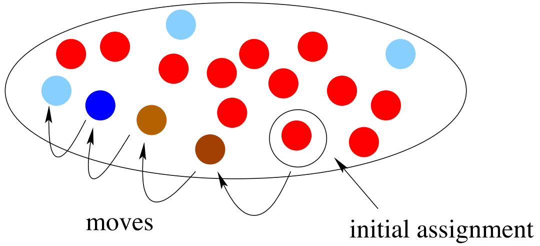
The **neighbour** reached by the **candidate move** that swaps the values of the two picked variables.

Probing reveals that this candidate move decreases by 1 the **cost** (= number of violated constraints, here). If it is **selected** (as it is most cost-decreasing, say) as the **move**, then this neighbour will be the new current assignment.

UPPSALA UNIVERSITET

Intuition: Abandon Backtracking & Optimality Guarantee

- An **initial assignment** gives each decision variable a value in its domain.
- Search proceeds iteratively by moves: each **move** modifies the values of just a few decision variables in the **current assignment**, and is **selected** after **probing** the **cost** decreases of several **candidate moves**, which go to assignments called **neighbours** that form the **neighbourhood**.
- Stop either when a trivially optimal assignment is found, or when an allocated budget is spent, such as time spent or moves made.



Problems

Definition

A problem is a tuple $\langle X, D, C\,[, f\,]\rangle$, where:

- $X = \{x_1, \ldots, x_n\}$ is the set of decision variables;
- D is the domain of all decision variables, without loss of generality (wlog);
- C is the set of constraints, each constraint being a function in $D^n \to \mathbb{B}$;
- the optional $f\colon D^n \to \mathbb{R}$ is the objective function, to be minimised, wlog.

Without f, it is a constraint satisfaction problem (CSP).
With f, it is a constrained optimisation problem (COP).

The variables need not have the same domain (so D is the union of all their domains and disequality constraints are needed) and constraints & objective function need not be over all the variables, but we want a simple notation. Maximising $f(X)$ amounts to minimising $-f(X)$.

Assignments

Consider a problem $\langle X, D, C\,[, f\,]\rangle$, with $|X| = n$:

Definitions

- An assignment $a\colon X \rightarrow D$ maps each decision variable to a domain value.
- The search space is the set $A := X \rightarrow D$ of all assignments, of size $|D|^n$.

For assignment a and function ϕ on D^n, denote $\phi(a(x_1), \ldots, a(x_n))$ by $\phi(a)$.

- A constraint c is satisfied under assignment a iff $c(a) = \textbf{true}$; else we say that c is violated under a.
- An assignment a is feasible iff all the constraints in C are satisfied under a.
- A solution to a CSP $\langle X, D, C\rangle$ is a feasible assignment.
- A minimal solution to a COP $\langle X, D, C, f\rangle$ is a feasible assignment a^* such that $f(a^*) \leq f(a)$ for every feasible assignment a.

Assignments, Moves, and Neighbours

An assignment in $X \to D$ for a problem $\langle X, D, C\,[, f\,]\rangle$ can be seen as a set of $x \mapsto d$ mappings, exactly one per decision variable $x \in X$, with $d \in D$.

Example: For $X = \{x_1, x_2, x_3, x_4\}$ and $D = \{\mathrm{q}, \mathrm{r}, \mathrm{s}, \mathrm{t}, \mathrm{u}\}$,
we have $\{x_1 \mapsto \mathrm{r},\ x_2 \mapsto \mathrm{r},\ x_3 \mapsto \mathrm{u},\ x_4 \mapsto \mathrm{r}\}\,(x_3) = \mathrm{u}$.

We consider two kinds of move to a neighbour of an assignment a:

- Upon the assign move $x := d$, the assign neighbour $a_{x:=d}$ denotes a where the decision variable $x \in X$ is reassigned the value $d \in D$: $a \setminus \{x \mapsto a(x)\} \cup \{x \mapsto d\}$. We assume $d \neq a(x)$ in the sequel.
- Upon the swap move $x :=: y$, the swap neighbour $a_{x:=:y}$ denotes a where the decision variables $x, y \in X$ have their values swapped: $a \setminus \{x \mapsto a(x),\ y \mapsto a(y)\} \cup \{x \mapsto a(y),\ y \mapsto a(x)\}$.

An assignment a is often implemented by X being one or more arrays of *programming* variables that undergo destructive updates: replace all $a(x)$ by x.

Notation:

- 1-based indexing: The indexing of all array and set elements starts from 1.
- Arg min operator: Let $\arg\min_{x \in X} f(x)$ denote the *set* of arguments in X on which function f takes its minimum value: $\{x \in X \mid \forall s \in X : f(x) \leq f(s)\}$.
- Iverson bracket:
  We define $[\phi] = 1$ if and only if formula ϕ evaluates to **true**, else $[\phi] = 0$.
- Lexicographic ordering: Let $\langle x_1, x_2 \rangle <_{\text{lex}} \langle y_1, y_2 \rangle$ denote that $\langle x_1, x_2 \rangle$ is lexicographically smaller than $\langle y_1, y_2 \rangle$: either $x_1 < y_1$ or $x_1 = y_1 \wedge x_2 < y_2$.
  This generalises to tuples of any length and any totally ordered types.
  Ex: $\langle 1, 2, 34, 5, 678 \rangle <_{\text{lex}} \langle 1, 2, 36, 45, 78 \rangle$ as $34 < 36$, though $678 \not< 78$.
- Range: Let $\ell \mathinner{\ldotp\ldotp} u$ denote the integer set $\{\ell,\ \ell+1,\ \ldots,\ u-1,\ u\}$, where $\ell \in \mathbb{N}$ and $u \in \mathbb{N}$. Note that $\ell \mathinner{\ldotp\ldotp} u = \varnothing$ when $\ell > u$.

Credits: Much of what follows is based on the book *Constraint-Based Local Search* by Pascal Van Hentenryck and Laurent Michel (see the Bibliography).

Soft Constraints and Their Violations

We can soften a constraint *c* into a soft constraint, meaning it may be violated during search: it has a function $\text{VIOLATION}_c \colon A \to \mathbb{N}$ that, for an assignment *a*, returns its violation, which must be zero iff *c* is satisfied under *a*, else a positive value that can for example be how many moves from *a* it takes to satisfy *c*.

Example ($\leq$)

- At least one variable must be reassigned in order to satisfy a violated inequality constraint:
  $\text{VIOLATION}_{x\leq y}(a) \triangleq \textbf{if } a(x) \leq a(y) \textbf{ then } 0 \textbf{ else } 1$
- Sometimes, we want to penalise a violated inequality constraint by more than the number of variables that must be reassigned in order to satisfy it:
  $\text{VIOLATION}_{x\leq y}(a) \triangleq \textbf{if } a(x) \leq a(y) \textbf{ then } 0 \textbf{ else } a(x) - a(y)$

We will see 3 ways of hardening a problem constraint into a hard constraint, meaning it is inviolable during search: on slide 22, slide 34, and slide 35.

The constraint AllDifferent(Y) requires all the elements of the array $Y[1\,..\,p]$ of p decision variables over the domain D to take distinct values, with $|D| \geq p$. It corresponds to $\frac{p\cdot(p-1)}{2}$ disequalities: $\forall i,j \in 1\,..\,p$ **where** $i < j : Y[i] \neq Y[j]$.

Examples ($\neq$ and AllDifferent)

1. $\textsc{Violation}_{x \neq y}(a) \triangleq$ **if** $a(x) \neq a(y)$ **then** 0 **else** 1
2. Let $Occ[d]$ be the number of occurrences in array Y under a of value d.
   Example: For the values $[\mathrm{r},\mathrm{r},\mathrm{u},\mathrm{r}]$ of array Y of $p = 4$ decision variables over the domain $\{\mathrm{q},\mathrm{r},\mathrm{s},\mathrm{t},\mathrm{u}\}$, we have $Occ = [0,3,0,0,1]$. So the violation of AllDifferent($[\mathrm{r},\mathrm{r},\mathrm{u},\mathrm{r}]$) could be $0 + (3-1) + 0 + 0 + (1-1) = 2$, because at least 2 decision variables (of the 3 ones assigned ‘r’) must be reassigned in order to satisfy the constraint.

   With Occ, we can compute in $\mathcal{O}(|D|)$ time, independently of $p = |Y|$:
   $\textsc{Violation}_{\text{AllDifferent}(Y)}(a) \triangleq \sum_{d \in D} \max(Occ[d] - 1,\ 0)$

   The total violation of the corresponding $\frac{p\cdot(p-1)}{2} = 6$ disequalities is 3 (under choice 1 above): this is needlessly high and takes $\mathcal{O}(p^2)$ time.

Probing a Move or Neighbour in $\mathcal{O}(1)$ Time?

Example (AllDifferent)

- **Example:** Let c be the soft constraint AllDifferent($[y_1, y_2, y_3, y_4]$), with $p = 4$ decision variables over the domain $D = \{\mathrm{q}, \mathrm{r}, \mathrm{s}, \mathrm{t}, \mathrm{u}\}$. If assignment a gives AllDifferent([r, r, u, r]), then $Occ = [0, 3, 0, 0, 1]$. Upon the move $y_1 := \mathrm{s}$ we would see $\textsc{Violation}_c(a)$ decrease by 1 (since there would be one less of the at least two occurrences of ‘r’) and increase by 0 (as there was not at least one ‘s’ under a). The net decrease is $1 - 0 = 1$.
- Upon $x := d$ the violation decreases by $[Occ[a(x)] \geq 2] - [Occ[d] \geq 1]$: evaluating this expression takes $\mathcal{O}(1)$ time, beating $\mathcal{O}(|D|)$ from scratch (an algorithm for which we will see on slide 15).

Probing a Move or Neighbour under *One* Soft Constraint

A soft constraint c also has a probe function $\text{DECREASE}_c : A \times A \to \mathbb{Z}$ that, for a neighbour n of an assignment a, returns the decrease of the violation of c if moving from a to n. An improving move or neighbour has a positive decrease. A worsening move or neighbour has a negative decrease (that is an increase).

Example (AllDifferent, using its array *Occ*)

Let a be an assignment and let v be $\text{VIOLATION}_{\text{AllDifferent}(Y)}(a)$:

- $\text{DECREASE}_{\text{AllDifferent}(Y)}(a, a_{x:=d}) \triangleq$
  $\begin{cases} v - \text{VIOLATION}_{\text{AllDifferent}(Y)}(a_{x:=d}) & \text{from scratch, in } \mathcal{O}(|D|) \text{ time (slide 15)} \\ [Occ[a(x)] \geq 2] - [Occ[d] \geq 1] & \text{differentially, in } \mathcal{O}(1) \text{ time} \end{cases}$
- $\text{DECREASE}_{\text{AllDifferent}(Y)}(a, a_{x:=:y}) \triangleq 0$, in $\mathcal{O}(1)$ time.

We cannot always design a differential DECREASE function that takes $\mathcal{O}(1)$ time or is asymptotically strictly faster than a from-scratch function.

Probing a Move or Neighbour under *One* Soft Constraint

If a soft constraint c has a function $\textsc{Violation}_c$ that has its own auxiliary data structures (for efficiency reasons, by trading space for time), then probing a move or neighbour via $\textsc{Decrease}_c$ may need to update those data structures.

Example (AllDifferent, using its array *Occ*)

- Probing an assign move $x := d$ from scratch, with violation v under a:

```
$Occ[a(x)]{-}{-};\ Occ[d]{+}{+}$                          // tentatively make the move
$v' := \textsc{Violation}_{\text{AllDifferent}(Y)}(a)$      // $Occ$ is queried, not $a$: recall slide 12
$Occ[a(x)]{+}{+};\ Occ[d]{-}{-}$                          // undo the tentative move
**return** $v - v'$                                         // return the decrease of the violation
```

- Probing an assign move differentially does not need to update *Occ*.
- Probing a swap move (from scratch or differentially) is meaningless.

Probing a Move / Neighbour under *All* Soft Constraints

Let S be the set of all soft constraints in a formulation of problem $\langle X, D, C\,[, f\,]\rangle$.

The total violation of all the soft constraints under assignment a is:

$$\textsc{Violation}_S(a) \triangleq \sum_{c \in S} \textsc{Violation}_c(a)$$

Probing a neighbour n of an assignment a, that is probing a move from a to n, is measuring the total decrease of the total violation if moving from a to n:

$$\textsc{Decrease}_S(a, n) \triangleq \sum_{c \in S} \textsc{Decrease}_c(a, n)$$

We may selectively apply these concepts to any strict subset, called a system, of S: we then talk of its system violation and system decrease.

Initialisation

Let S be the set of all soft constraints in a formulation of problem $\langle X, D, C\,[, f\,]\rangle$.

Initialisation returns an assignment that becomes the first current assignment, say a, for which we also must initialise the current total violation of S:

1. Initialise the auxiliary data structures owned by all the soft constraints.

   Example: For each soft constraint AllDifferent(Y) initialise its *Occ* array:

   ```
   for $d$ in $D$ do $Occ[d] := 0$
   for $i := 1$ to $|Y|$ do $Occ[a(Y[i])]{+}{+}$
   ```

   Time complexity: $\mathcal{O}(|D| + |Y|)$.
2. $v := \textsc{Violation}_S(a)$

where v is a *programming* variable, denoting the current total violation.

Making a Move

Let S be the set of all soft constraints in a formulation of problem $\langle X, D, C\,[, f\,]\rangle$.

Making a move from the current assignment a, of total violation v, to a selected probed neighbour n of a requires performing the following destructive updates:

1. $v := v - \textsc{Decrease}_S(a, n)$
2. Update the auxiliary data structures owned by all the soft constraints.

   Example: Upon $x := d$, we need to perform $Occ[a(x)]{-}{-}$ and $Occ[d\,]{+}{+}$ for the array *Occ* of each soft constraint AllDifferent(Y) where x is in Y.
3. $a := n$

Decision Variables and their Violations

A soft constraint c may also have a function $\textsc{Violation}_c^x\colon A \to \mathbb{N}$ that, for an assignment a, returns the variable violation of decision variable x of c, which must be zero iff $\textsc{Violation}_c(a)$ cannot be decreased by any move on x, else a positive value that can for example be how much $\textsc{Violation}_c(a)$ can be decreased by some move on x, which is then called a violating variable.

Example (AllDifferent)

- $\textsc{Violation}^{Y[i]}_{\text{AllDifferent}(Y)}(a) \triangleq [Occ[a(Y[i])] \geq 2]$
  where $Occ[d]$ is the number of occurrences in array Y under a of value d.
- **Example:** Let c be AllDifferent(Y) and let a give AllDifferent([r, r, u, r]): the only violating variables are $Y[1]$, $Y[2]$, and $Y[4]$.

We can define the concept of total variable violation of x and use $\textsc{Decrease}_c$ in order to probe the (total) variable violation decrease of x upon a move on x.

The functions for the soft constraints can be queried in order to guide search:

- The constraint violation functions $\textsc{Violation}_c$ can be queried to find promising soft constraint(s) on whose decision variables to probe a move.
- The variable violation functions $\textsc{Violation}_c^x$ can be queried to find promising decision variable(s) on which to probe a move.
- The probing functions $\textsc{Decrease}_c$ can be queried to select a move in a good direction for the violations of some decision variable(s) of a soft constraint c, or for the violation of c itself, or for the violation of several (if not all) soft constraints.

These functions must be implemented for the highest time efficiency, possibly with the help of auxiliary data structures (by trading space for time), as they will be queried in the innermost loop,
that is during the probing of the neighbourhood at each search iteration.

Cost of an Assignment (in general: $\neq$ its objective value)

Let the function $\textsc{Cost}\colon A \to \mathbb{R}^k$ give the cost of an assignment, with $k \in \{1, 2\}$:

Examples (where S is the set of all soft constraints)

- For a CSP: $\textsc{Cost}(a) = \textsc{Violation}_S(a)$
- For a COP: $\textsc{Cost}(a) = \alpha \cdot \textsc{Violation}_S(a) + \beta \cdot f(a)$
- For a COP: $\textsc{Cost}(a) = \langle \textsc{Violation}_S(a), f(a) \rangle$
- For a COP with $S = \varnothing$: $\textsc{Cost}(a) = f(a)$
- For a COP with $S = \varnothing$: $\textsc{Cost}(a) = \langle f(a), \tau(a) \rangle$
  where τ is a tiebreaker between assignments of equal objective value (giving a lower score to an assignment that seems fewer moves from one with lower objective value), and lexicographically minimise $\textsc{Cost}(a)$.

for problem-specific and constraint-specific functions $\textsc{Violation}_S\colon A \to \mathbb{N}$, and for problem-specific hyperparameters α and β in $\mathbb{R}$.

One-Way Constraints (aka Invariants)

Example

The equality constraint $p + q + r = s$ functionally defines s from p, q, and r, so no moves on s should be probed because s can be uniquely determined upon each probe or move on at least one of p, q, and r, but not vice-versa.

Consider a constraint c that functionally defines a decision variable y from its other decision variables $\vec{x}$. We can make c hard during search by reformulating it as a one-way constraint (aka invariant) as follows:

1. Rewrite c into the form $y \leftsquigarrow \rho(\vec{x})$ for some function ρ.
2. Do *not* consider y when designing the neighbourhood.
3. Propagate: transitively update the value of y for each probe and move.

If VIOLATION and DECREASE for the soft constraints are only used in invariants, then invariant propagation does *everything*! **Example:** The *programming* variable w is $\text{VIOLATION}_{x \leq y}(a)$ via $w \leftsquigarrow$ **if** $a(x) \leq a(y)$ **then** 0 **else** 1.

Outline

UPPSALA UNIVERSITET

Outline

- Initialise (in polynomial time) the current assignment, using randomisation.
- Iteratively move to a selected admissible probed neighbour assignment.
- Aim for an assignment that minimises COST according to a strict total order $\prec$ (such as $<$ on numbers, and $<_{\text{lex}}$ on tuples of numbers).

Generic Heuristic

```
$a := \textsc{InitialAssignment}(X, D)$                    // $a$ is the current assignment
$a^* := a$                                                 // $a^*$ is the so far best assignment
while $\textsc{Cost}(a^*)$ is not trivially minimal and $\textsc{Budget}(X, D)$ is not spent do
    $a := \textsc{Select}(\textsc{Admissible}(\textsc{Neighbours}(a), a), a)$
    if $\textsc{Cost}(a) \prec \textsc{Cost}(a^*)$ then $a^* := a$
return $a^*$
```

where (we may need a metaheuristic to escape local minima of $\textsc{Cost}(a)$):

- $\textsc{InitialAssignment}(X, D)$ returns an assignment, using randomisation;
- $\textsc{Budget}(X, D)$ returns the possibly instance-specific budget, such as the time to be spent or the number of moves to be made;
- $\textsc{Neighbours}(a)$ returns the neighbourhood of a, as a stream;
- $\textsc{Admissible}(N, a)$ filters the neighbourhood N with respect to a;
- $\textsc{Select}(N, a)$ selects an element of set N with respect to a.

Examples (NEIGHBOURS)

$$\text{AssignAnyVar}(a) = \{a_{x:=d} \mid x \in X \wedge d \in D \setminus \{a(x)\}\}$$

$$\text{RandomAssign}(a) = \{a_{x:=d} \mid x = \text{random}(X) \wedge d = \text{random}(D \setminus \{a(x)\})\}$$

$$\text{AssignViolatingVar}(a) = \{a_{v:=d} \mid v = \text{random}(\{x \in X \mid \textsc{Violation}^x(a) > 0\}) \wedge d \in D \setminus \{a(v)\}\}$$

$$\text{AssignMostViolatingVar}(a) = \{a_{v:=d} \mid v = \text{random}(\arg\max_{x \in X} \textsc{Violation}^x(a)) \wedge d \in D \setminus \{a(v)\}\}$$

$$\text{SwapAnyVars}(a) = \{a_{x:=:y} \mid x, y \in X \wedge x \neq y\}$$

… and many more, including compositions of some of the above …

Examples (ADMISSIBLE, where S is the set of all soft constraints)

$$\text{All}(N, \_) = N$$

$$\text{Improving}(N, a) = \{n \in N \mid \textsc{Cost}(n) \prec \textsc{Cost}(a)\}$$

$$\text{NonWorsening}(N, a) = \{n \in N \mid \textsc{Cost}(n) \preceq \textsc{Cost}(a)\}$$

$$\text{Feasible}(N, \_) = \{n \in N \mid \textsc{Violation}_S(n) = 0\}$$

... and many more, including compositions of some of the above ...

Examples (SELECT)

$$\text{First}(N, \_) = \text{the first element in the admissible neighbour set } N$$

$$\text{Random}(N, \_) = \text{random}(N)$$

$$\text{Best}(N, \_) = \text{random}(\arg\min_{n \in N} \textsc{Cost}(n))$$

$$\text{IfImproving}(\{n\}, a) = \textbf{if } \textsc{Cost}(n) \prec \textsc{Cost}(a) \textbf{ then } n \textbf{ else } a$$

... and many more ...

Examples (Heuristics as SELECT ∘ ADMISSIBLE ∘ NEIGHBOURS)

Systematic or partial exploration of the neighbourhood:

- First improving neighbour:

  First(Improving(NEIGHBOURS(a), a), _)

- Steepest descent (aka gradient descent):

  Best(Improving(NEIGHBOURS(a), a), _)

- Min-conflict:

  Best(All(AssignViolatingVar(a), a), _)

- . . . and many more . . .

Random walk (pick *one* random neighbour and decide whether to select it):

- Random improvement:

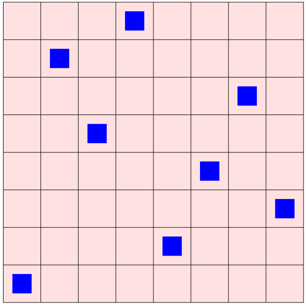
  IfImproving(All(RandomAssign(a), _), a)

- Metropolis (1953): see Simulated Annealing (at slide 61)
- . . . and many more . . .

UPPSALA UNIVERSITET

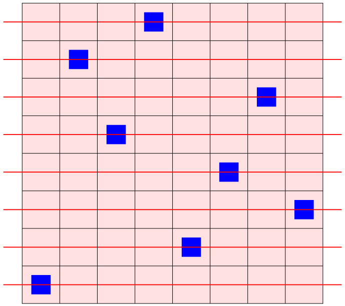
Outline

n Queens



Place n queens on an $n \times n$ board such that no two queens attack each other:

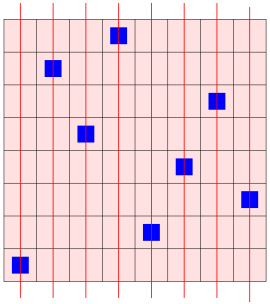
n Queens



Place n queens on an $n \times n$ board such that no two queens attack each other:

1. No two queens are on the same row.

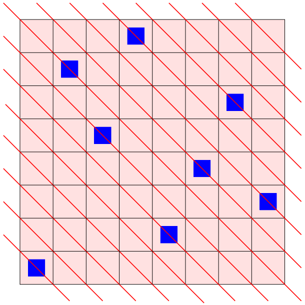
n Queens



Place n queens on an $n \times n$ board such that no two queens attack each other:

1. No two queens are on the same row.
2. No two queens are on the same column.

n Queens

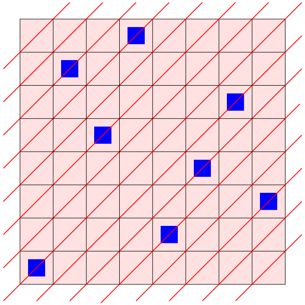


Place n queens on an $n \times n$ board such that no two queens attack each other:

1. No two queens are on the same row.
2. No two queens are on the same column.
3. No two queens are on the same down-diagonal.

UPPSALA UNIVERSITET

n Queens



Place n queens on an $n \times n$ board such that no two queens attack each other:

1. No two queens are on the same row.
2. No two queens are on the same column.
3. No two queens are on the same down-diagonal.
4. No two queens are on the same up-diagonal.

Naïve Formulation

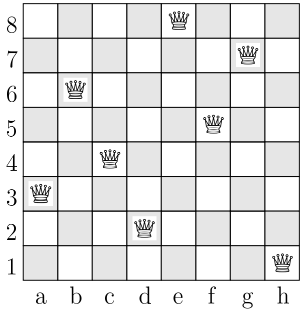
Let the decision variable $Q[r, c]$, over domain $\{0, 1\}$, denote the number of queens in column c of row r, with $r \in 1 \mathinner{\ldotp\ldotp} n$ and $c \in 1 \mathinner{\ldotp\ldotp} n$:

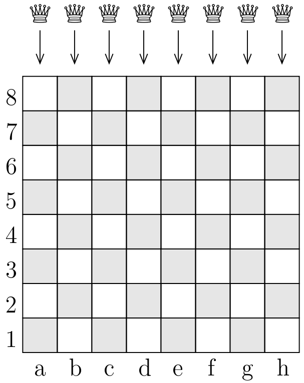
1. No two queens are on the same row: soft: $\forall r \in 1 \mathinner{\ldotp\ldotp} n : \sum_{c=1}^{n} Q[r, c] = 1$
2. No two queens are on the same column: soft: $\forall c \in 1 \mathinner{\ldotp\ldotp} n : \sum_{r=1}^{n} Q[r, c] = 1$
3. No two queens are on the same down-diagonal:
   soft: … left as an exercise …
4. No two queens are on the same up-diagonal:
   soft: … left as an exercise …

Number of assignments in the search space: 2^{n^2}, or $\binom{n^2}{n}$ if we are careful.

Towards a Better Formulation

Can we define fewer decision variables by exploiting a property of solutions?





Yes, there must be exactly one queen per column! (Prove it by contradiction.)

Better Formulation

Let the decision variable $R[c]$, over domain $1 \mathinner{..} n$, denote the row of the queen in column c, with $c \in 1 \mathinner{..} n$:

1. No two queens are on the same row:
   soft: $\text{AllDifferent}([R[1], \dots, R[n]])$
2. No two queens are on the same column:
   hard: This is now guaranteed by the choice of the decision variables!
3. No two queens are on the same down-diagonal:
   soft: $\text{AllDifferent}([R[1] - 1, \ \dots, \ R[n] - n])$
4. No two queens are on the same up-diagonal:
   soft: $\text{AllDifferent}([R[1] + 1, \ \dots, \ R[n] + n])$

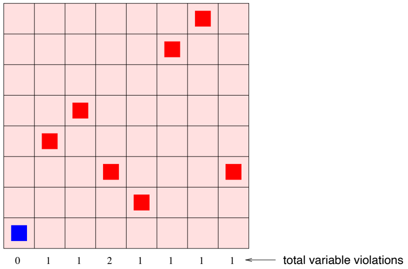
Number of assignments in the search space: now n^n “only”.
Number of assign moves: $n \cdot (n - 1)$, each probed differentially in $\mathcal{O}(1)$ time.
Number of swap moves: $\frac{n \cdot (n-1)}{2}$, but bad connectivity: empty rows remain so.

Even Better Formulation

Let the decision variable $R[c]$, over domain $1\,..\,n$, denote the row of the queen in column c, with $c \in 1\,..\,n$:

1. No two queens are on the same row: We make this constraint implicit and thus hard by $\textsc{InitialAssignment}(R, n)$ returning a random permutation of $1\,..\,n$, in $\mathcal{O}(n)$ time, and $\textsc{Neighbours}(a)$ trying only swap moves on a.
2. No two queens are on the same column: hard: ... as before ...
3. No two queens are on the same down-diagonal: soft: ... as before ...
4. No two queens are on the same up-diagonal: soft: ... as before ...

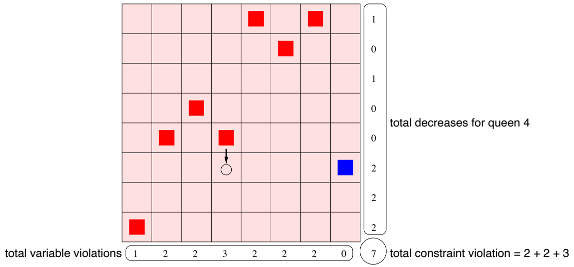
Number of assignments in the search space: now $n!$ "only".
Number of constraints to probe: now only 2 (instead of 3).
Number of assign moves: still $n \cdot (n-1)$, but they all violate hard constraint $\#1$.
Number of swap moves: still $\frac{n \cdot (n-1)}{2}$, good connectivity: reachable solutions.



Under the better formulation; let the upper-left corner have coordinates $(1,1)$:

1. AllDifferent($[R[1], \ldots, R[8]]$)
   The violation of AllDifferent($[8,5,4,6,7,2,1,6]$) is 1.
3. AllDifferent($[R[1]-1, \ \ldots, \ R[8]-8]$)
   The violation of AllDifferent($[7,3,1,2,2,-4,-6,-2]$) is 1.
4. AllDifferent($[R[1]+1, \ \ldots, \ R[8]+8]$)
   The violation of AllDifferent($[9,7,7,10,12,8,8,14]$) is 2.

Total violation: $1+1+2=4$.


1
0
1
0
0
2
2
2
total decreases for queen 4
total variable violations
1 2 2 3 2 2 2 0
7
total constraint violation = 2 + 2 + 3

Probing an assign move in $\mathcal{O}(1)$ time, under the better formulation:

1. AllDifferent($[R[1], \ldots, R[4], \ldots, R[8]]$)
   Decrease for $R[4] := 6$ on AllDifferent($[8, 5, 4, 5, 1, 2, 1, 6]$) is ± 0.
3. AllDifferent($[R[1] - 1, \ \ldots, \ R[4] - 4, \ \ldots, \ R[8] - 8]$)
   Decrease for $R[4] := 6$ on AllDifferent($[7, 3, 1, 1, -4, -4, -6, -2]$) is 1.
4. AllDifferent($[R[1] + 1, \ \ldots, \ R[4] + 4, \ \ldots, \ R[8] + 8]$)
   Decrease for $R[4] := 6$ on AllDifferent($[9, 7, 7, 9, 6, 8, 8, 14]$) is 1.

Total decrease for $R[4] := 6$: $0 + 1 + 1 = 2$.

Heuristic

Let S be the set of the 3 soft AllDifferent constraints of the better formulation, and compare with the generic heuristic on slide 25:

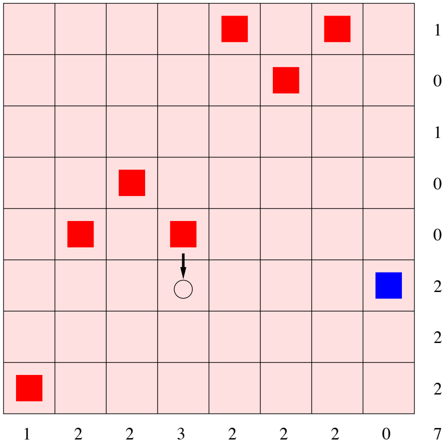
```
$a := \{R[c] \mapsto \text{random}(1\,..\,n) \mid c \in 1\,..\,n\}$          // $a$ is the current assignment
$a^* := a$                                                     // $a^*$ is the so far best assignment
$k := 0$                                                       // $k$ is the move counter
**while** $\text{VIOLATION}_S(a^*) > 0$ **and** $k < 50 \cdot n$ **do**
    $a := \text{Best}(\text{All}(\text{AssignMostViolatingVar}(a), a), a)$      // min-conflict
    **if** $\text{VIOLATION}_S(a) < \text{VIOLATION}_S(a^*)$ **then** $a^* := a$
    $k := k + 1$
**return** $a^*$
```

In English: Move a random most violating queen to a random row where the total number of attacks decreases most.

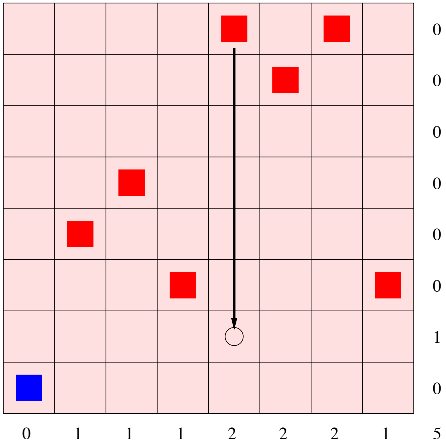
In SLS parlance: min-conflict: Reassign a random most violating decision variable a random value for which the total violation decreases most.

UPPSALA UNIVERSITET

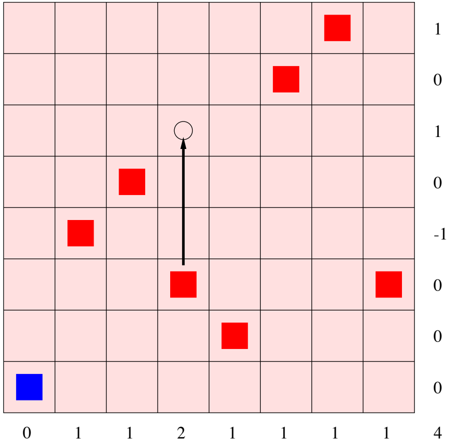
8 Queens: Sample Run, under the Better Formulation



8 Queens: Sample Run, under the Better Formulation



8 Queens: Sample Run, under the Better Formulation

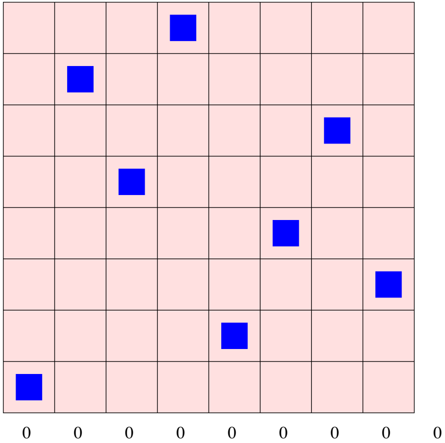


8 Queens: Sample Run, under the Better Formulation

. . . and so on, hopefully until a solution . . .

UPPSALA UNIVERSITET

8 Queens: Sample Run, under the Better Formulation



UPPSALA UNIVERSITET

Outline

Sudoku (reminder)

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

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

A Sudoku is a 9-by-9 array of integers in the range $1\,..\,9$.
Some of the elements are provided as hints.
The remaining elements have to satisfy the following constraints:

1. The elements in each row are distinct.
2. The elements in each column are distinct.
3. The elements in each boldfaced 3-by-3 block are distinct.

Naïve Formulation

Let the decision variable $S[r, c]$, over domain $1\,..\,9$, denote the number in column c of row r, with $r \in 1\,..\,9$ and $c \in 1\,..\,9$:

0 The instance-specific hints are respected:

$$\text{soft: for example, } S[1,1] = 8 \;\wedge\; S[2,3] = 3 \;\wedge\; \cdots \;\wedge\; S[9,7] = 4$$

1 The elements in each row are distinct:

$$\text{soft: } \forall r \in 1\,..\,9 : \text{AllDifferent}([S[r,1], \ldots, S[r,9]])$$

$$\text{or, in array slicing notation: } \forall r \in 1\,..\,9 : \text{AllDifferent}(S[r,..])$$

2 The elements in each column are distinct:

$$\text{soft: } \forall c \in 1\,..\,9 : \text{AllDifferent}(S[..,c])$$

3 The elements in each boldfaced 3-by-3 block are distinct:

$$\text{soft: } \forall i, j \in \{0,3,6\} : \text{AllDifferent}(S[i+1\,..\,i+3,\; j+1\,..\,j+3])$$

Number of assignments in the search space: $9^{9 \cdot 9}$.

Better Formulation

Let the decision variable $S[r, c]$, over domain $1 \mathinner{..} 9$, denote the number in column c of row r, with $r \in 1 \mathinner{..} 9$ and $c \in 1 \mathinner{..} 9$:

1. The instance-specific hints are respected *and* the elements in each row are distinct: We make *all* these constraints implicit and thus hard
by INITIALASSIGNMENT(S) returning for each row a random permutation of $1 \mathinner{..} 9$ that respects at least its hints, in $\mathcal{O}(1)$ time, and
by NEIGHBOURS(a) considering only swap moves on a within a row.
2. The elements in each column are distinct: soft: . . . as before . . .
3. The elements in each boldfaced 3-by-3 block are distinct: . . . as before . . .

Number of assignments in search space: $\sum_{i=1}^{9}(9-h_i)^{9-h_i}$, for h_i hints in row i.
Number of constraints to probe: now only $2 \cdot 9$ (instead of $\sum_{i=1}^{9} h_i + 3 \cdot 9$).
Number of assign moves: $\sum_{i=1}^{9}(9-h_i)(8-h_i)$, but all violate hard constraint 1.
Number of swap moves: $\sum_{i=1}^{9}\binom{9-h_i}{2}$, good connectivity: reachable solutions.

Digression

- Consider a constraint AllDifferent(Y) whose decision variables are over a domain D with $|D| \geq |Y|$ (else the constraint is trivially unsatisfiable).
- For both Sudoku (where $|Y| \leq 9$ for each row) and the even better formulation of n Queens, we have $|D| = |Y|$: we made this constraint implicit, and thus hard, by INITIALASSIGNMENT returning for Y a random permutation of D and NEIGHBOURS considering only swap moves on Y (as assign moves would violate hard constraint $\#1$), which achieves good neighbourhood connectivity: from any assignment, an [optimal] solution is reachable.
- But what if $|D| > |Y|$? In order to make the constraint implicit, and thus hard, and yet achieve good neighbourhood connectivity, NEIGHBOURS must consider swap moves $Y[i] :=: Y[j]$ and assign moves $Y[i] := d$ where value d is currently not used: $a(Y[j]) \neq d$ for all indices j within Y.

Initialisation and Probing

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

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

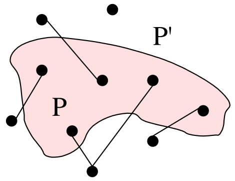
8	6	9	2	7	3	5	4	1
4	1	3	6	8	2	9	7	5
5	7	3	4	9	1	2	8	6
6	5	2	8	3	7	1	9	4
1	8	6	9	4	5	7	2	3
7	4	9	1	6	8	5	3	2
2	3	1	7	9	4	5	6	8
4	2	8	5	9	6	3	1	7
6	9	5	3	2	7	4	8	1

Any swap move on non-hints within a row, say $S[8,5] :=: S[8,7]$, preserves the row being a hint-respecting permutation of $1\,..\,9$ and corresponds to 2 assign moves, $S[8,5] := a(S[8,7])$ and $S[8,7] := a(S[8,5])$. Recall slide 13: we can differentially probe the AllDifferent violation decreases on blocks 8 & 9 to be both 0, and those on columns 5 & 7 to be 0 and 1 respectively, all in $\mathcal{O}(1)$ time, so the probed swap has a total decrease of $0+0+0+1=1$ and is improving.

Outline

Graph Partitioning

- **Problem:** Given a graph $G = (V, E)$, find a balanced partition $\langle P, P' \rangle$ of V that minimises the number of edges with vertices in both P and P'.
- **Definition:** A balanced partition $\langle P, P' \rangle$ of V satisfies three constraints: $P \cup P' = V$, and $P \cap P' = \varnothing$, and $-1 \leq |P| - |P'| \leq 1$.
- **Example:** A graph and a balanced partition of objective value 5:

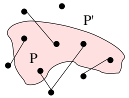


Algorithmic Choices

- **Initial assignment:** InitialAssignment returns a random balanced partition $\langle P, P' \rangle$ of V in $\mathcal{O}(n)$ time, with $n = |V|$, by putting a random $\lfloor \frac{n}{2} \rfloor$ vertices of V into P and the remaining $\lceil \frac{n}{2} \rceil$ vertices of V into P'.
- **Neighbourhood:** Exchange two vertices:
  $\textsc{Neighbours}(P, P') = \{\langle P \setminus \{i\} \cup \{j\}, P' \setminus \{j\} \cup \{i\} \rangle \mid i \in P \wedge j \in P'\}$
  Number of moves: $\lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil$, with good connectivity: optima are reachable.
- **Cost:** The objective value (the number of edges with vertices in both P and P'), because the three balance constraints are implicit and thus hard:
  $\textsc{Cost}(P, P') = f(P, P') = |\{(i, j) \in E \mid i \in P \wedge j \in P'\}|$
  Each exchange move can be probed differentially in $\mathcal{O}(1)$ time.
- **Admissible neighbours:**
  $\textsc{Admissible}(N, \langle P, P' \rangle) = \text{Improving}(N, \langle P, P' \rangle)$
- **Neighbour selection:** $\textsc{Select}(N, \langle P, P' \rangle) = \text{Best}(N, \langle P, P' \rangle)$

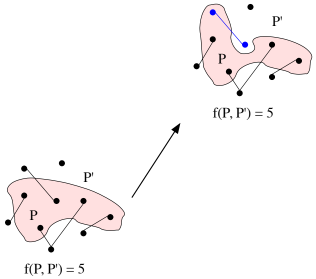
UPPSALA UNIVERSITET

Sample Run

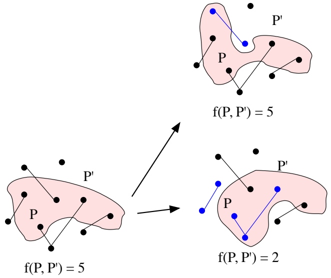


f(P, P') = 5

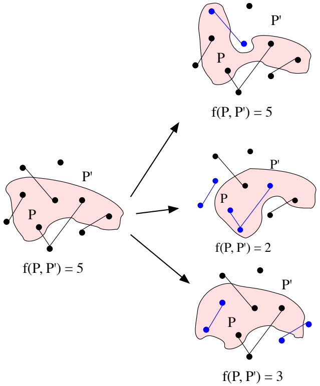
Sample Run



Sample Run

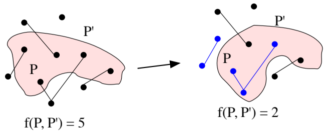


Sample Run

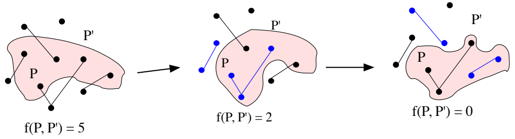


and 22 other probed neighbours $\langle P, P' \rangle$,
but none with $f(P, P') < 2$

Sample Run



Sample Run

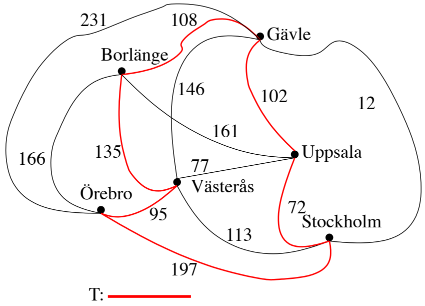


and 24 other probed neighbours $\langle P, P' \rangle$,
obviously none of which with $f(P, P') < 0$:
the trivial lower bound is reached, so search can stop, with proven optimality!

Outline

Travelling Salesperson Problem (TSP)

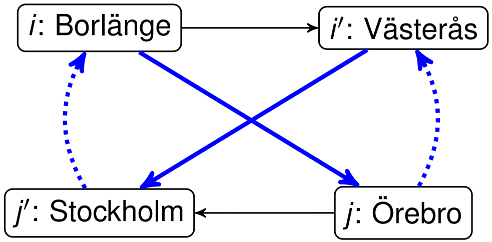
- **Problem:** Given a set of cities with connecting roads, find a Hamiltonian circuit (tour) visiting each city exactly once, with minimal travel distance.
- **Formulation:** We see the cities as vertices V and the roads as edges E of a (not necessarily complete) directed graph $G = (V, E)$.
- **Example:**

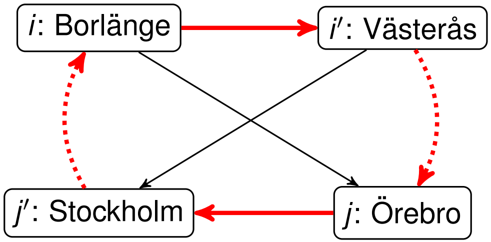


Algorithmic Choices

- **Initial assignment:** InitialAssignment returns a random edge set $T \subseteq E$ that forms a tour: but this is NP-hard! If need be, we can make G a complete graph by adding infinite-distance edges: now *any* permutation of V yields a tour and takes $\mathcal{O}(n)$ time, with $n = |V|$, to select randomly or construct via a greedy nearest-neighbour algorithm.
- **Neighbourhood:** For example (see next slide), replace two edges on the current tour by two edges outside the current tour so that it still is a tour. Number of moves: $\binom{n}{2} = \mathcal{O}(n^2)$, with good connectivity: reachable optima.
- **Cost:** The objective value (the sum of the distances on the tour), because the tour-ness constraint (Hamiltonicity) is implicit and thus hard: $\textsc{Cost}(T) = f(T) = \sum_{(a,b)\in T} D(a,b)$
- **Admissible neighbours**: $\textsc{Admissible}(N,T) = \text{Improving}(N,T)$
- **Neighbour selection:** $\textsc{Select}(N,T) = \text{Best}(N,T)$

Example: Two-Exchange Move



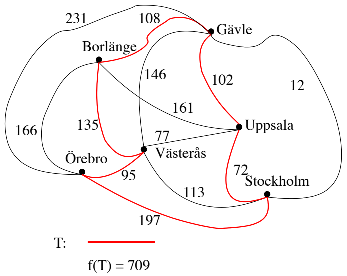


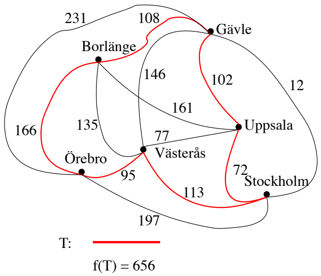
On the blue tour: replace 2 edges $i \to j$ & $i' \to j'$ by the 2 edges $i \to i'$ & $j \to j'$.
The resulting red tour replaces the sub-path $j \rightsquigarrow i'$ by the corresponding $i' \rightsquigarrow j$.
The decrease in cost is $D(i,j) + D(i',j') - (D(i,i') + D(j,j'))$
if the distance matrix D is symmetric: differential probing in $\mathcal{O}(1)$ time.
There are many other kinds of move for vehicle routing problems, some with more than 2 replaced edges, or without the symmetry assumption, or both.

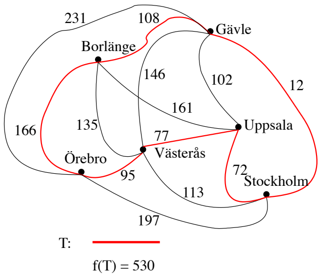
UPPSALA UNIVERSITET

Sample Run

Two consecutive improving two-exchange moves:







Outline

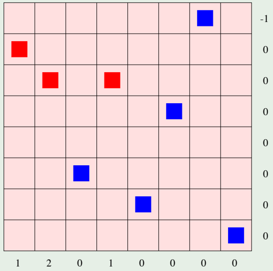
Recap

A heuristic (recall slide 29) drives the search to (good enough) solutions:

- Which decision variables are modified in a move?
- Which new values do they get in the move?

But heuristics tend to drive the search to *local* minima of Cost!

Example (8 Queens: Local Minimum)



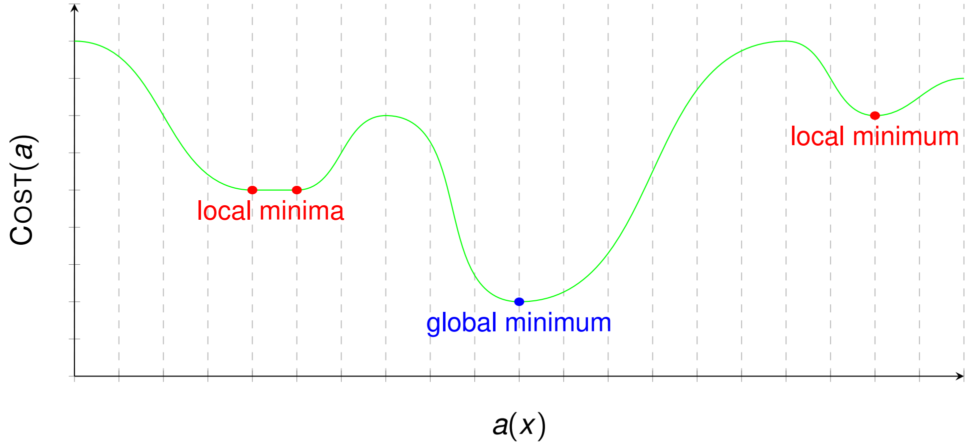
Under the better formulation, and assuming the heuristic of slide 38:

- Queen 2 must be selected, as Queen 2 is the only most violating queen.
- Queen 2 is moved to any of the rows 2 to 8, as the total violation would decrease by -1 (that is increase by 1) if Queen 2 were moved to row 1.
- Queen 2 remains the only most violating queen!
- Queen 2 is selected over and over again.

A metaheuristic can escape this local minimum of $\text{COST} = \text{VIOLATION}_S = 2$.

Local Minima

What happens when $X = \{x\}$ and the only moves are $x := x \pm 1$:



A metaheuristic drives the search to *global* optima of Cost:

- How to avoid cycles of moves? How to escape local optima of Cost?
- Diversification: How to explore many parts of the search space?
- Intensification: How to focus on promising parts of the search space?

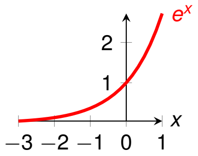
Examples (Metaheuristics)

- Randomised iterative improvement: with some small probability, move to a random neighbour, else use a systematic or partial heuristic (of slide 29).
- Simulated annealing (1983): pick a random move and make it even if it is non-improving, with a probability that exponentially decreases over time.
- Tabu search (Glover, 1986): forbid recent moves from being made again.
- Genetic algorithms: use a pool of current assignments and cross them.
- ...

Each metaheuristic could be the topic of at least one full lecture.

Simulated Annealing (Kirkpatrick, Gelatt, & Vecchi, 1983)

- In metallurgy, annealing is a hardening process for a metal, by cooling it down from a suitable temperature $t > 0$ that is neither too high nor too low.
- Pick a random neighbour n of the current assignment a.
- If $\text{Cost}(n) < \text{Cost}(a)$, then the move from a to n is improving & is made.
- Else the move is non-improving and is made with probability $e^{\frac{\text{Cost}(a)-\text{Cost}(n)}{t}}$ where $e \approx 2.71828$ is the base of the natural logarithm.



We have $0 < e^x \leq 1$ when $x \leq 0$: so e^x can serve as a probability and it decreases when x decreases.

Note that $\Delta = \text{Cost}(a) - \text{Cost}(n) \leq 0$, so that the probability decreases exponentially when t decreases (due to the division by t) or Δ decreases.

- Let Cooling be the cooling schedule, such that $\text{Cooling}(t) \leq t$.

Simulated Annealing

Compare with the generic algorithm of slide 25, assuming $\text{Cost}(a) \in \mathbb{R}$:

```
$a := \text{InitialAssignment}(X, D)$                // $a$ is the current assignment
$a^* := a$                                           // $a^*$ is the so far best assignment
$t := \text{InitialTemperature}$                     // initialise the temperature
while $\text{Cost}(a^*)$ is not trivially minimal and $\text{Budget}(X, D)$ is not spent do
    $a := \text{Metropolis}(\text{All}(\text{RandomAssign}(a), \_), a, t)$
    if $\text{Cost}(a) < \text{Cost}(a^*)$ then $a^* := a$
    $t := \text{Cooling}(t)$    // Ex: either $t$ or geometric cooling to $\alpha \cdot t$ for $0 < \alpha < 1$
return $a^*$
```

where:

$$\text{Metropolis}(\{n\}, a, t) =$$
$$\textbf{if } \text{Cost}(n) < \text{Cost}(a) \textbf{ then } n \textbf{ else } n \text{ with probability } e^{\frac{\text{Cost}(a) - \text{Cost}(n)}{t}} \text{ otherwise } a$$

High t: like random moves, but this might spot where a global minimum lies.
Low t: like random improvement (slide 29), hopefully to a good local minimum.

Tabu Search (1986; see book by Glover & Laguna, 1997)

- In order to escape local optima, we must allow worsening moves.
- To avoid ending up in cycles, we remember the last λ assignments in a short-term memory called the tabu list and make them tabu (aka taboo, that is forbidden), where λ is a (function on the problem parameters that returns a) hyperparameter, called the tabu tenure: those assignments cannot be moved to, even if this means having to make a worsening move.
- An element in the tabu list usually actually consists of features of a forbidden assignment (such as the move to it) or a forbidden move (such as the involved decision variables).
- An aspiration criterion is an exception mechanism.
  In tabu search, one can allow a move on tabu features if it improves on a^*.

Tabu Search

Compare with the generic algorithm of slide 25 and add aspiration:

```
$a := \textsc{InitialAssignment}(X, D)$    // $a$ is the current assignment
$a^* := a$    // $a^*$ is the so far best assignment
$T := [\text{Features}(a), \ldots, \text{Features}(a)]$    // initialise the tabu list to length $\lambda$
while $\textsc{Cost}(a^*)$ is not trivially minimal and $\textsc{Budget}(X, D)$ is not spent do
    $a := \text{Best}(\text{NonTabu}(\textsc{Neighbours}(a), T), T)$
    if $\textsc{Cost}(a) \prec \textsc{Cost}(a^*)$ then $a^* := a$
    $T := \text{tail}(T) +\!\!+ [\text{Features}(a)]$    // keep only the last $\lambda$ elements
return $a^*$
```

where $\text{NonTabu}(N, T) = \{n \in N \mid \text{Features}(n) \notin T\}$.

In practice, list T is often implemented as an array, indexed by attribute tuples: $T[f]$ is the iteration until which a move with features f is tabu; initialised to -1, it is updated at iteration i to $i + \lambda$ if a move with features f is made.

UPPSALA UNIVERSITET

Large Neighbourhood Search (survey: Pisinger & Røpke)

Take a COP $\langle X, D, C, f \rangle$ and compare with the generic algorithm of slide 25:

```
$P := \langle X, D, C \rangle$                                   // the corresponding CSP
$a :=$ First(Solutions($P$), _) under systematic search
$a^* := a$                                   // $a^*$ is a so far best assignment
while Cost($a^*$) is not trivially minimal and Budget($X, D$) is not spent do
    $P := \langle X,\ D,\ C \cup \{f(X) \leq f(a^*(X))\},\ f \rangle$, but where some decision
    variables are frozen (e.g., are fixed to their values in $a^*$) and the other
    decision variables are thawed (aka relaxed) (e.g., have $D$ as domain)
    $a :=$ Best(Solutions($P$), _) under budget-bounded systematic search
    if some $a$ was found then $a^* := a$                  // it is surely no worse
return $a^*$
```

where Solutions(P) denotes the stream of solutions to the CSP or COP P.

Examples (Diversification: explore many parts of search space)

- Generic heuristic, upon adding restarts: raise their probability.
- Simulated annealing: raise the temperature t.
- Tabu search: raise the tabu tenure λ.
- Large neighbourhood search: lower the percentage of frozen variables.

Examples (Intensification: focus on promising parts of search space)

- Generic heuristic, upon adding restarts: lower their probability.
- Simulated annealing: lower the temperature t.
- Tabu search: lower the tabu tenure λ.
- Large neighbourhood search: raise the percentage of frozen variables.

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

Collected Insights

1. Use high-level decision variables whenever possible.

   Examples:

   - Use an integer (or enumerated) decision variable with a domain of k values instead of an array of k Boolean (or $0/1$) decision variables. [Ex]
   - Use a set decision variable of cardinality k instead of an array of k integer (or enumerated) decision variables. [Ex]

2. Formulate some problem constraints as hard constraints, meaning that they cannot be violated during search, so that they need not be probed:

   - either by the choice of the decision variables and their domains; [Ex]
   - or as implicit constraints, under INITIALASSIGNMENT and NEIGHBOURS; [Ex]
   - or as one-way constraints, which functionally define decision variables that are not candidates for moves, but are to be updated upon each move. [Ex]

Collected Insights (continued)

3. Soften each remaining constraint c into a soft constraint, meaning that it may be violated during search. Design the functions $\textsc{Violation}_c^{(x)}$ and $\textsc{Decrease}_c^{(x)}$ that you need. [Ex]

4. Try to bundle multiple soft constraints into an equivalent single soft constraint, if the latter can be probed asymptotically faster. [Ex]

5. Make sure the neighbourhood has good connectivity:
from any assignment, an [optimal] solution should be reachable,
else diversification is crucial. [Ex1, Ex2, Ex3, Ex4, Ex5]

6. Make probing as efficient as possible, for example by maintaining auxiliary data structures for some constraints. [Ex]

Collected Insights (end)

7. Try to derive a bound on the objective function that either is a constant or can be computed in polynomial time from the problem parameters, so that search can be stopped with proven optimality if the objective value reaches that bound and the total violation of the soft constraints is 0. [Ex]

8. Prefer randomised choices over deterministic choices.

9. Try to exploit the presence of symmetries by doing nothing (rather than by making the explored search space smaller, as with systematic search by CP, MIP, SAT, or SMT). One reason might be that symmetry-breaking constraints forbid some solutions, but do not change the search space and hence do not prevent the search from moving in their direction.

Systematic Search (as in CP, MIP, SAT, and SMT):

$+$ Will ultimately find an (optimal) solution, if a solution exists.

$+$ Will ultimately give a proof of the optimality of a found solution.

$+$ Will ultimately give a proof of unsatisfiability, if no solution exists.

$-$ Often does not scale well to large instances, and is hard to parallelise.

$-$ May need a lot of tweaking: search strategies, ...

Stochastic Local Search:

$\pm$ Might find an (optimal) solution, if a solution exists.

$-$ Can rarely give a proof of the optimality of a found solution.

$-$ Can rarely give a proof of unsatisfiability, if no solution exists.

$+$ Often scales well to large instances, and is easy to parallelise.

$-$ May need a lot of tweaking: (meta)heuristics, hyperparameters, ...

Stochastic local search trades guaranteed solution quality for solving speed!

Outline

Bibliography

- Hoos, Holger H. and Stützle, Thomas. Stochastic Local Search: Foundations & Applications. Elsevier / Morgan Kaufmann, 2004.
- Van Hentenryck, Pascal and Michel, Laurent. Constraint-Based Local Search. The MIT Press, 2005.
- Glover, Fred W. and Laguna, Manuel. Tabu Search. Kluwer Academic Publishers, 1997.
- Kirkpatrick, S. and Gelatt, Jr., C. D. and Vecchi, M. P. Optimization by simulated annealing. *Science* 220(4598):671–680, May 1983.
- Pisinger, David and Røpke, Stefan. Large neighborhood search. *Handbook of Metaheuristics* (3rd edition), pages 99–127, Springer, 2019.