Topic 17: Constraint-Based Local Search[1]
(Version of 18th January 2019)

Pierre Flener

Optimisation Group
Department of Information Technology
Uppsala University
Sweden

Course 1DL441:
Combinatorial Optimisation and Constraint Programming,
whose part 1 is Course 1DL451:
Modelling for Combinatorial Optimisation

[1] Based on an early version by Magnus Ågren (2008)

(Meta-) Heuristics for Local Search
Local Search
Heuristics
Example 1: Graph Partitioning
Example 2: Travelling Salesperson
Meta-Heuristics
Constraint-Based Local Search
Modelling
Violation Functions
Probing Functions
Comparison with CP
Example: The COMET System
Hybrid Methods
Bibliography

Outline

1. **(Meta-) Heuristics for Local Search**
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. **Constraint-Based Local Search**
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. **Example: The COMET System**
4. **Hybrid Methods**
5. **Bibliography**

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. Constraint-Based Local Search
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

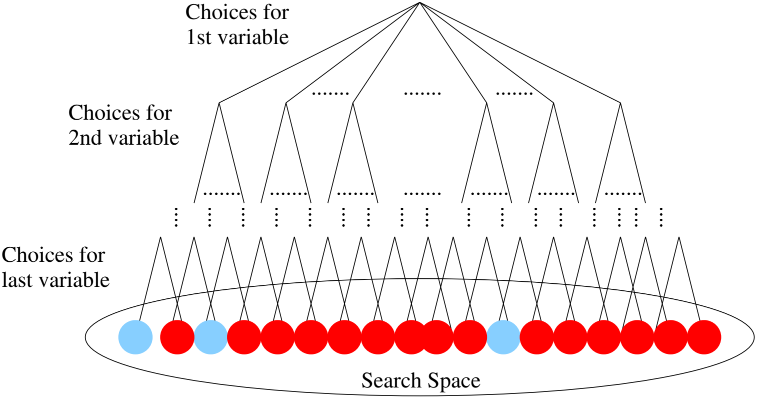
(Meta-) Heuristics for Local Search
Local Search
Heuristics
Example 1: Graph Partitioning
Example 2: Travelling Salesperson
Meta-Heuristics
Constraint-Based Local Search
Modelling
Violation Functions
Probing Functions
Comparison with CP
Example: The COMET System
Hybrid Methods
Bibliography

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. Constraint-Based Local Search
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

So Far: Inference + Systematic Search

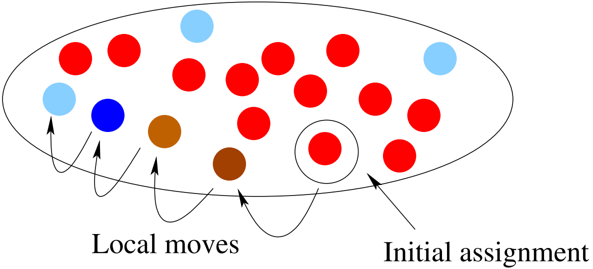
- The variables become fixed 1-by-1.
- Stop when solution or unsatisfiability proof is obtained.
- Search space from a systematic-search viewpoint:



(Meta-) Heuristics for Local Search
Local Search
Heuristics
Example 1: Graph Partitioning
Example 2: Travelling Salesperson
Meta-Heuristics
Constraint-Based Local Search
Modelling
Violation Functions
Probing Functions
Comparison with CP
Example: The COMET System
Hybrid Methods
Bibliography

Now: Inference + Local Search

- Each variable is fixed all the time.
- Search proceeds by moves: each move modifies the values of a few variables in the current assignment, and is selected upon probing the cost impacts of several candidate moves, called the neighbourhood.
- Stop when a good enough assignment has been found, or when an allocated resource has been exhausted, such as time spent or iterations made.



Example (BIBD: AED assignment after *i* moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	✓	–	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal growth load: Every plot grows 3 grains.
   Currently satisfied: zero violation.
2. Equal sample size: Every grain is grown in 3 plots.
   Satisfied by initial assignment and each move: implicit.
3. Balance: Every grain pair is grown in 1 common plot.
   But, e.g., oats & rye are grown in $2 > 1$ common plots.

Example (BIBD: AED assignment after *i* moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	✓	–	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal growth load: Every plot grows 3 grains.
   Currently satisfied: zero violation.
2. Equal sample size: Every grain is grown in 3 plots.
   Satisfied by initial assignment and each move: implicit.
3. Balance: Every grain pair is grown in 1 common plot.
   But, e.g., oats & rye are grown in $2 > 1$ common plots.

Selected move: let plot6 instead of plot5 grow oats.

Example (BIBD: AED assignment after *i* moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal growth load: Every plot grows 3 grains.
   Currently satisfied: zero violation.
2. Equal sample size: Every grain is grown in 3 plots.
   Satisfied by initial assignment and each move: implicit.
3. Balance: Every grain pair is grown in 1 common plot.
   But, e.g., oats & rye are grown in $2 > 1$ common plots.

Selected move: let plot6 instead of plot5 grow oats.

Example (BIBD: AED assignment after $i+1$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal growth load: Every plot grows 3 grains.
   But plot5 grows $2 < 3$ grains; plot6 grows $4 > 3$ grains.
2. Equal sample size: Every grain is grown in 3 plots.
   Satisfied by initial assignment and each move: implicit.
3. Balance: Every grain pair is grown in 1 common plot.
   But, e.g., corn & oats are grown in $2 > 1$ common plots.

Example (BIBD: AED assignment after $i + 1$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal growth load: Every plot grows 3 grains.
   But plot5 grows $2 < 3$ grains; plot6 grows $4 > 3$ grains.
2. Equal sample size: Every grain is grown in 3 plots.
   Satisfied by initial assignment and each move: implicit.
3. Balance: Every grain pair is grown in 1 common plot.
   But, e.g., corn & oats are grown in $2 > 1$ common plots.

Selected move: let plot5 instead of plot6 grow corn.

Example (BIBD: AED assignment after $i + 1$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal growth load: Every plot grows 3 grains.
   But plot5 grows $2 < 3$ grains; plot6 grows $4 > 3$ grains.
2. Equal sample size: Every grain is grown in 3 plots.
   Satisfied by initial assignment and each move: implicit.
3. Balance: Every grain pair is grown in 1 common plot.
   But, e.g., corn & oats are grown in $2 > 1$ common plots.

Selected move: let plot5 instead of plot6 grow corn.

Example (BIBD: AED assignment after $i + 2$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal growth load: Every plot grows 3 grains.
   Currently satisfied: zero violation.
2. Equal sample size: Every grain is grown in 3 plots.
   Satisfied by initial assignment and each move: implicit.
3. Balance: Every grain pair is grown in 1 common plot.
   Currently satisfied: zero violation.

Stop search: All constraints are satisfied.

Terminology and Choices

Consider a constraint problem with constraints $\{c_1, \ldots, c_n\}$ and optionally an objective function f, which is here to be minimised, without loss of generality:

Definition

A satisfying (or feasible) assignment maps all decision variables to domain values that satisfy all the constraints c_i.

Property: A satisfying assignment actually is a solution to a constraint satisfaction problem (CSP), but it may be sub-optimal for a constrained optimisation problem (COP).

Assume function COST gives the cost of an assignment s:

- CSP: $\textsc{Cost}(s) = \sum_{i=1}^{n} \textsc{Violation}(c_i, s)$
- COP: $\textsc{Cost}(s) = \alpha \cdot \sum_{i=1}^{n} \textsc{Violation}(c_i, s) + \beta \cdot f(s)$

for problem-specific VIOLATION and parameters α and β.

Definition

A soft constraint c has a function $\textsc{Violation}(c, s)$ that returns zero if c is satisfied under the assignment s, else a positive value depending on the level of violation.

Example: $\textsc{Violation}(x \leq y, s) =$ **if** $s(x) \leq s(y)$ **then** 0 **else** $s(x) - s(y)$

Definition

A one-way constraint is kept satisfied during search, as one of its variables is defined by a total function on the others.

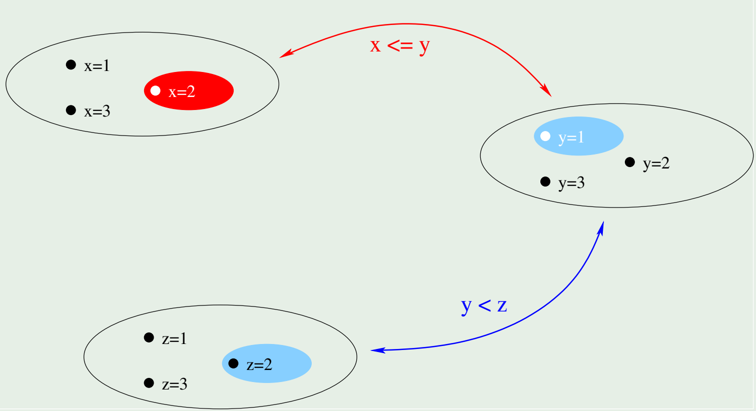
Example: For $p = x \cdot y$: if x or y is reassigned by a move to assignment s, then $s(p)$ is to be set to $s(x) \cdot s(y)$.

Definition

A violating variable in a constraint c unsatisfied, or violated, under assignment s can be reassigned, not necessarily within its domain, so that $\textsc{Violation}(c, s)$ decreases.

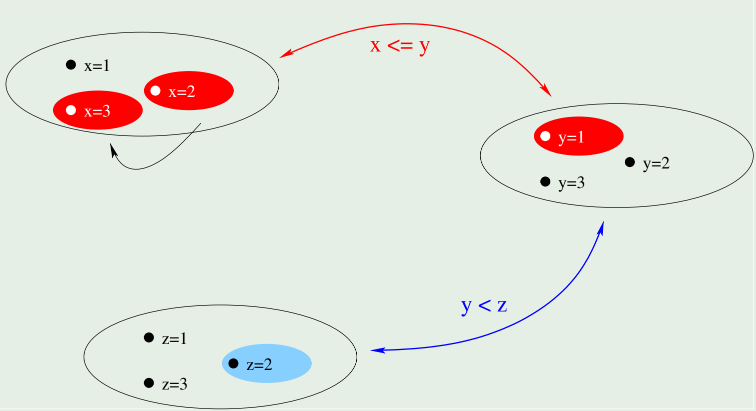
Example ($x, y, z \in \{1, 2, 3\} \wedge x \leq y \wedge y < z$)

Unsatisfying assignment (the constraint $x \leq y$ is violated; the decision variables x and y are violating wrt $x \leq y$):



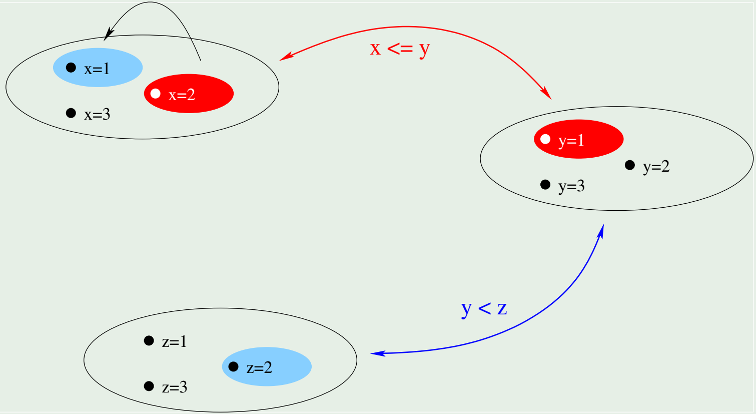
Example ($x, y, z \in \{1, 2, 3\} \wedge x \leq y \wedge y < z$)

Candidate move $x := 3$, reaching another unsatisfying assignment (the constraint $x \leq y$ is still violated; the decision variables x and y are still violating wrt $x \leq y$):



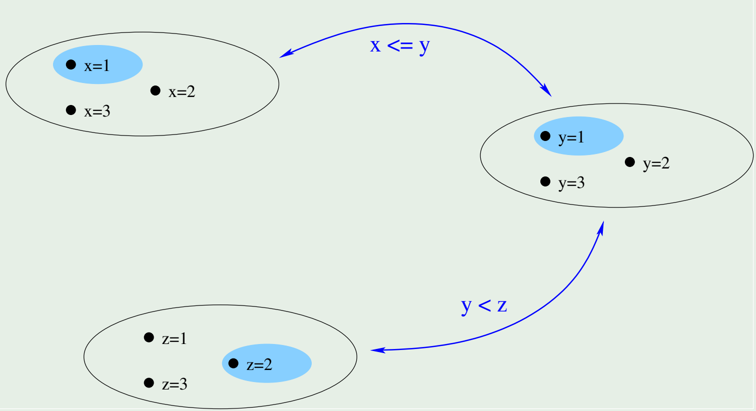
Example ($x, y, z \in \{1, 2, 3\} \wedge x \leq y \wedge y < z$)

Another candidate move $x := 1$, reaching a satisfying assignment (there are no more violated constraints or violating variables):



Example ($x, y, z \in \{1, 2, 3\} \wedge x \leq y \wedge y < z$)

Another candidate move $x := 1$, reaching a satisfying assignment (there are no more violated constraints or violating variables):



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

+ Will find an (optimal) solution, if one exists.
+ Will give a proof of unsatisfiability, otherwise.
– May take a long time to complete.
– Sometimes does not scale well to large instances.
– May need a lot of tweaking: search strategies, ...

Local Search: (Hoos and Stützle, 2004)

+ May find an (optimal) solution, if one exists.
– Can rarely give a proof of unsatisfiability, otherwise.
– Can rarely guarantee that a found solution is optimal.
+ Often scales much better to large instances.
– May need a lot of tweaking: heuristics, parameters, ...

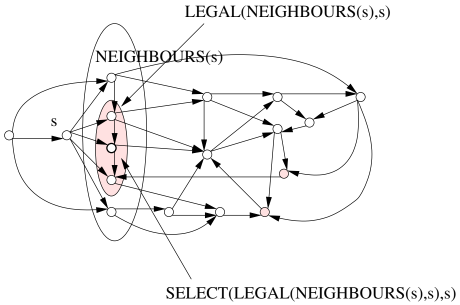
Local search trades completeness and quality for speed!

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. Constraint-Based Local Search
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

Local-Search Heuristics: Outline

- Start from an initial assignment.
- Iteratively move to a neighbour assignment.
- Aim for a satisfying assignment minimising COST.
- Main operation: Move from the current assignment to a selected assignment among its legal neighbours:

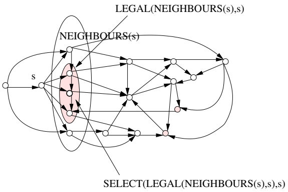


Local-Search Heuristics: Generic Algorithm

$s := \text{INITIALASSIGNMENT}()$
$k := 0; s^* := s$ // s^* is the so far best assignment
while $\sum_{i=1}^{n} \text{VIOLATION}(c_i, s) > 0$ **and** $k < \mu$ **do**
  $k := k+1; s := \text{SELECT}(\text{LEGAL}(\text{NEIGHBOURS}(s), s), s)$
  if $\text{COST}(s) < \text{COST}(s^*)$ **then** $s^* := s$
return s^*

where (may need a meta-heuristic to escape local optima):

- NEIGHBOURS(s) returns the neighbours of s.
- LEGAL(N, s) returns the legal neighbours in N w.r.t. s.
- SELECT(M, s) returns a selected element of M w.r.t. s.



Examples (LEGAL)

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

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

$$\text{ViolatingVar}(N, s) = \{n \in N \mid n(x) \neq s(x) \text{ for a violating variable } x\}$$

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

Examples (SELECT)

$$\text{First}(M, s) = \text{the first element in } M$$

$$\text{Best}(M, s) = \text{random}\left(\left\{n \in M \mid \textsc{Cost}(n) = \min_{t \in M} \textsc{Cost}(t)\right\}\right)$$

$$\text{RandomImproving}(M, s) = \textbf{let } n = \text{random}(M) \textbf{ in if } \textsc{Cost}(n) < \textsc{Cost}(s) \textbf{ then } n \textbf{ else } s$$

Local Search: Sample Heuristics

Examples (Heuristics for SELECT ∘ LEGAL)

Systematic (partial) exploration of the neighbourhood:

- First improving neighbour: First(Improving(N, s), s)
- Steepest / Gradient descent: Best(Improving(N, s), s)
- Min-conflict: Best(ViolatingVar(N, s), s)
- . . .

Random walk (pick a neighbour and decide on selecting it):

- Random improvement: RandomImproving(All(N, s), s)
- . . .

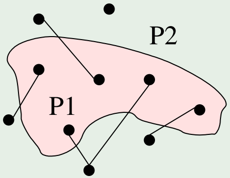
(Meta-) Heuristics for Local Search
Local Search
Heuristics
Example 1: Graph Partitioning
Example 2: Travelling Salesperson
Meta-Heuristics
Constraint-Based Local Search
Modelling
Violation Functions
Probing Functions
Comparison with CP
Example: The COMET System
Hybrid Methods
Bibliography

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. Constraint-Based Local Search
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

Example (Graph Partitioning)

- **Problem:** Given a graph $G = (V, E)$, find a balanced partition $\langle P_1, P_2 \rangle$ of V that minimises the number of edges with end-points in both P_1 and P_2.
- **Definition:** A balanced partition $\langle P_1, P_2 \rangle$ of V satisfies $P_1 \cup P_2 = V$, $P_1 \cap P_2 = \varnothing$, and $-1 \leq |P_1| - |P_2| \leq 1$.
- **Example:**



We will now come up with a greedy local-search algorithm for this problem.

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
2. The **cost** of an assignment (COST).
3. The **neighbourhood function** (NEIGHBOURS).
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ A balanced partition $\langle P_1, P_2 \rangle$ of $G = (V, E)$.
2. The **cost** of an assignment (COST).
3. The **neighbourhood function** (NEIGHBOURS).
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ A balanced partition $\langle P_1, P_2 \rangle$ of $G = (V, E)$.
2. The **cost** of an assignment (COST).
   ☞ The number of edges with one end-point in each set:
3. The **neighbourhood function** (NEIGHBOURS).
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** (InitialAssignment).
   ☞ A balanced partition $\langle P_1, P_2 \rangle$ of $G = (V, E)$.
2. The **cost** of an assignment (Cost).
   ☞ The number of edges with one end-point in each set:
   $\textsc{Cost}(\langle P_1, P_2 \rangle) = f(\langle P_1, P_2 \rangle) = |\{(a, b) \in E \mid a \in P_1 \wedge b \in P_2\}|$
3. The **neighbourhood function** (Neighbours).
4. The **legal-neighbour selection function** (Legal).
5. The **neighbour selection function** (Select).

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** (InitialAssignment).
   ☞ A balanced partition $\langle P_1, P_2 \rangle$ of $G = (V, E)$.
2. The **cost** of an assignment (Cost).
   ☞ The number of edges with one end-point in each set:
   $\textsc{Cost}(\langle P_1, P_2 \rangle) = f(\langle P_1, P_2 \rangle) = |\{(a, b) \in E \mid a \in P_1 \wedge b \in P_2\}|$
3. The **neighbourhood function** (Neighbours).
   ☞ Swapping two vertices:
4. The **legal-neighbour selection function** (Legal).
5. The **neighbour selection function** (Select).

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ A balanced partition $\langle P_1, P_2 \rangle$ of $G = (V, E)$.
2. The **cost** of an assignment (COST).
   ☞ The number of edges with one end-point in each set:
   $\text{COST}(\langle P_1, P_2 \rangle) = f(\langle P_1, P_2 \rangle) = |\{(a, b) \in E \mid a \in P_1 \land b \in P_2\}|$
3. The **neighbourhood function** (NEIGHBOURS).
   ☞ Swapping two vertices: $\text{NEIGHBOURS}(\langle P_1, P_2 \rangle) = \{\langle P_1 \setminus \{a\} \cup \{b\}, P_2 \setminus \{b\} \cup \{a\} \rangle \mid a \in P_1 \land b \in P_2\}$
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** (InitialAssignment).
   ☞ A balanced partition $\langle P_1, P_2 \rangle$ of $G = (V, E)$.
2. The **cost** of an assignment (Cost).
   ☞ The number of edges with one end-point in each set:
   $\text{Cost}(\langle P_1, P_2 \rangle) = f(\langle P_1, P_2 \rangle) = |\{(a, b) \in E \mid a \in P_1 \land b \in P_2\}|$
3. The **neighbourhood function** (Neighbours).
   ☞ Swapping two vertices: $\text{Neighbours}(\langle P_1, P_2 \rangle) = \{\langle P_1 \setminus \{a\} \cup \{b\}, P_2 \setminus \{b\} \cup \{a\} \rangle \mid a \in P_1 \land b \in P_2\}$
4. The **legal-neighbour selection function** (Legal).
   ☞ The improving neighbours:
5. The **neighbour selection function** (Select).

Example (Graph Partitioning: Choices)

We must define:

1. The **initial assignment** ($\textsc{InitialAssignment}$).
   ☞ A balanced partition $\langle P_1, P_2\rangle$ of $G = (V, E)$.
2. The **cost** of an assignment ($\textsc{Cost}$).
   ☞ The number of edges with one end-point in each set:
   $\textsc{Cost}(\langle P_1, P_2\rangle) = f(\langle P_1, P_2\rangle) = |\{(a, b) \in E \mid a \in P_1 \land b \in P_2\}|$
3. The **neighbourhood function** ($\textsc{Neighbours}$).
   ☞ Swapping two vertices: $\textsc{Neighbours}(\langle P_1, P_2\rangle) =$
   $\{\langle P_1 \setminus \{a\} \cup \{b\}, P_2 \setminus \{b\} \cup \{a\}\rangle \mid a \in P_1 \land b \in P_2\}$
4. The **legal-neighbour selection function** ($\textsc{Legal}$).
   ☞ The improving neighbours:
   $\textsc{Legal}(N, s) = \text{Improving}(N, s)$
5. The **neighbour selection function** ($\textsc{Select}$).

Example (Graph Partitioning: Choices)

We must define:

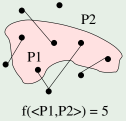
1. The **initial assignment** (InitialAssignment).
   ☞ A balanced partition $\langle P_1, P_2 \rangle$ of $G = (V, E)$.
2. The **cost** of an assignment (Cost).
   ☞ The number of edges with one end-point in each set:
   $\textsc{Cost}(\langle P_1, P_2 \rangle) = f(\langle P_1, P_2 \rangle) = |\{(a, b) \in E \mid a \in P_1 \wedge b \in P_2\}|$
3. The **neighbourhood function** (Neighbours).
   ☞ Swapping two vertices: $\textsc{Neighbours}(\langle P_1, P_2 \rangle) =$
   $\{\langle P_1 \setminus \{a\} \cup \{b\}, P_2 \setminus \{b\} \cup \{a\} \rangle \mid a \in P_1 \wedge b \in P_2\}$
4. The **legal-neighbour selection function** (Legal).
   ☞ The improving neighbours:
   $\textsc{Legal}(N, s) = \text{Improving}(N, s)$
5. The **neighbour selection function** (Select).
   ☞ A random best legal neighbour:

Example (Graph Partitioning: Choices)

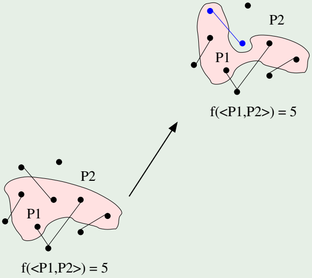
We must define:

1. The **initial assignment** ($\textsc{InitialAssignment}$).
   ☞ A balanced partition $\langle P_1, P_2\rangle$ of $G = (V, E)$.
2. The **cost** of an assignment ($\textsc{Cost}$).
   ☞ The number of edges with one end-point in each set:
   $\textsc{Cost}(\langle P_1, P_2\rangle) = f(\langle P_1, P_2\rangle) = |\{(a, b) \in E \mid a \in P_1 \wedge b \in P_2\}|$
3. The **neighbourhood function** ($\textsc{Neighbours}$).
   ☞ Swapping two vertices: $\textsc{Neighbours}(\langle P_1, P_2\rangle) =$
   $\{\langle P_1 \setminus \{a\} \cup \{b\}, P_2 \setminus \{b\} \cup \{a\}\rangle \mid a \in P_1 \wedge b \in P_2\}$
4. The **legal-neighbour selection function** ($\textsc{Legal}$).
   ☞ The improving neighbours:
   $\textsc{Legal}(N, s) = \text{Improving}(N, s)$
5. The **neighbour selection function** ($\textsc{Select}$).
   ☞ A random best legal neighbour:
   $\textsc{Select}(M, s) = \text{Best}(M, s)$

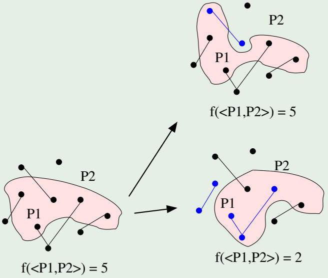
Example (Graph Partitioning: Sample Run)



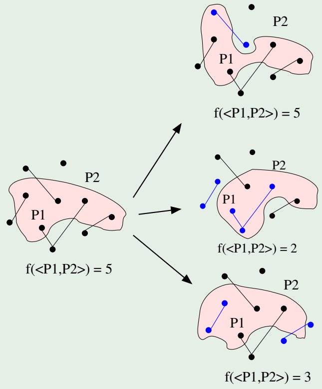
Example (Graph Partitioning: Sample Run)



Example (Graph Partitioning: Sample Run)

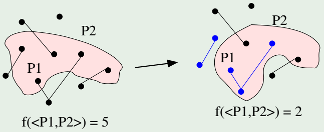


Example (Graph Partitioning: Sample Run)



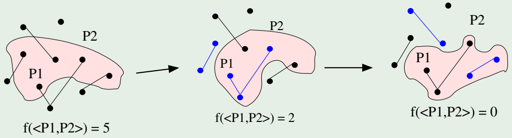
and 22 other probed neighbours $\langle P_1, P_2 \rangle$,
but none of which with $f(\langle P_1, P_2 \rangle) < 2$

Example (Graph Partitioning: Sample Run)



(Meta-) Heuristics for Local Search
Local Search
Heuristics
Example 1: Graph Partitioning
Example 2: Travelling Salesperson
Meta-Heuristics
Constraint-Based Local Search
Modelling
Violation Functions
Probing Functions
Comparison with CP
Example: The COMET System
Hybrid Methods
Bibliography

Example (Graph Partitioning: Sample Run)



and 24 other probed neighbours $\langle P_1, P_2 \rangle$,
obviously none of which with $f(\langle P_1, P_2 \rangle) < 0$:
the trivial lower bound was reached, so search can stop,
with proven optimality (this is rare)!

Example (Graph Partitioning)

Fundamental property of the chosen neighbourhood:
If an assignment s is a balanced partition,
then each partition in NEIGHBOURS(s) is also balanced.

- Only satisfying assignments are considered, including the generated initial assignment.
- The balance constraints are not modelled explicitly.
- This is a common and often crucial technique: some constraints are explicit (either soft or one-way), while other constraints are implicit, in the sense that they are satisfied by the generated initial assignment and kept satisfied during search by the neighbourhood. Constraints are hard (either implicit or one-way) or soft.
- The size of the neighbourhood is $\binom{|V|}{2}^2$.
- The search space is connected: any optimal solution can be reached from any assignment.

Outline

1. (Meta-) Heuristics for Local Search
- Local Search
- Heuristics
  - Example 1: Graph Partitioning
  - Example 2: Travelling Salesperson
- Meta-Heuristics

2. Constraint-Based Local Search
- Modelling
- Violation Functions
- Probing Functions
- Comparison with CP

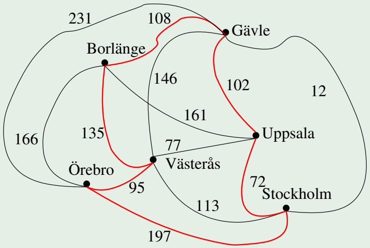
3. Example: The COMET System

4. Hybrid Methods

5. Bibliography

Example (Travelling Salesperson)

- **Problem:** Given a set of cities with connecting roads, find a tour (a Hamiltonian circuit) that visits each city exactly once, with the minimum travel distance.
- **Representation:** We see the set of cities as vertices V and the set of roads as edges E in a (not necessarily complete) undirected graph $G = (V, E)$.



- **Example:** s:

We now design a local-search heuristic for this problem.

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
2. The **cost** of an assignment (COST).
3. The **neighbourhood function** (NEIGHBOURS).
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that TOUR(s): NP-hard!
2. The **cost** of an assignment (COST).
3. The **neighbourhood function** (NEIGHBOURS).
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that TOUR(s): NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
3. The **neighbourhood function** (NEIGHBOURS).
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that $\text{Tour}(s)$: NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
   $$\text{Cost}(s) = f(s) = \sum_{(a,b) \in s} \text{Distance}(a, b)$$
3. The **neighbourhood function** (NEIGHBOURS).
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that $\text{TOUR}(s)$: NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
   $$\text{COST}(s) = f(s) = \sum_{(a,b)\in s} \text{Distance}(a,b)$$
3. The **neighbourhood function** (NEIGHBOURS).
   ☞ Replace two edges on the tour by two other edges:
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that $\text{TOUR}(s)$: NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
   $$\text{COST}(s) = f(s) = \sum_{(a,b) \in s} \text{Distance}(a, b)$$
3. The **neighbourhood function** (NEIGHBOURS).
   ☞ Replace two edges on the tour by two other edges:
   $$\text{NEIGHBOURS}(s) = \{s \setminus \{g, h\} \cup \{i, j\} \mid g, h \in s \land i, j \in E \setminus s\}$$
4. The **legal-neighbour selection function** (LEGAL).
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that TOUR(s): NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
   $$\text{COST}(s) = f(s) = \sum_{(a,b)\in s} \text{Distance}(a, b)$$
3. The **neighbourhood function** (NEIGHBOURS).
   ☞ Replace two edges on the tour by two other edges:
   $$\text{NEIGHBOURS}(s) = \{s \setminus \{g, h\} \cup \{i, j\} \mid g, h \in s \wedge i, j \in E \setminus s\}$$
4. The **legal-neighbour selection function** (LEGAL).
   ☞ The improving neighbours that define a tour:
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that TOUR(s): NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
   $\text{COST}(s) = f(s) = \sum_{(a,b)\in s} \text{Distance}(a, b)$
3. The **neighbourhood function** (NEIGHBOURS).
   ☞ Replace two edges on the tour by two other edges:
   $\text{NEIGHBOURS}(s) = \{s \setminus \{g, h\} \cup \{i, j\} \mid g, h \in s \land i, j \in E \setminus s\}$
4. The **legal-neighbour selection function** (LEGAL).
   ☞ The improving neighbours that define a tour:
   $\text{LEGAL}(N, s) = \{n \in N \mid \text{COST}(n) < \text{COST}(s) \land \text{TOUR}(n)\}$
5. The **neighbour selection function** (SELECT).

Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that $\textsc{Tour}(s)$: NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
   $\textsc{Cost}(s) = f(s) = \sum_{(a,b) \in s} \text{Distance}(a, b)$
3. The **neighbourhood function** (NEIGHBOURS).
   ☞ Replace two edges on the tour by two other edges:
   $\textsc{Neighbours}(s) = \{s \setminus \{g, h\} \cup \{i, j\} \mid g, h \in s \wedge i, j \in E \setminus s\}$
4. The **legal-neighbour selection function** (LEGAL).
   ☞ The improving neighbours that define a tour:
   $\textsc{Legal}(N, s) = \{n \in N \mid \textsc{Cost}(n) < \textsc{Cost}(s) \wedge \textsc{Tour}(n)\}$
5. The **neighbour selection function** (SELECT).
   ☞ A random best legal neighbour:

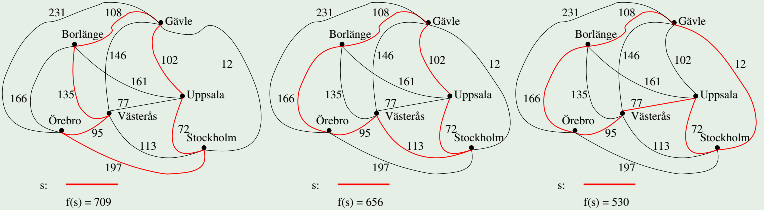
Example (Travelling Salesperson: Choices)

We must define:

1. The **initial assignment** (INITIALASSIGNMENT).
   ☞ An edge set $s \subseteq E$ so that $\text{TOUR}(s)$: NP-hard!
2. The **cost** of an assignment (COST).
   ☞ The sum of all distances on the tour:
   $$\text{COST}(s) = f(s) = \sum_{(a,b)\in s} \text{Distance}(a, b)$$
3. The **neighbourhood function** (NEIGHBOURS).
   ☞ Replace two edges on the tour by two other edges:
   $$\text{NEIGHBOURS}(s) = \{s \setminus \{g, h\} \cup \{i, j\} \mid g, h \in s \wedge i, j \in E \setminus s\}$$
4. The **legal-neighbour selection function** (LEGAL).
   ☞ The improving neighbours that define a tour:
   $$\text{LEGAL}(N, s) = \{n \in N \mid \text{COST}(n) < \text{COST}(s) \wedge \text{TOUR}(n)\}$$
5. The **neighbour selection function** (SELECT).
   ☞ A random best legal neighbour:
   $$\text{SELECT}(M, s) = \text{Best}(M, s)$$

Example (Travelling Salesperson: Sample Run)

Three consecutive improving satisfying assignments:



Example (Travelling Salesperson)

Fundamental property of the chosen neighbourhood:
Not all neighbours are satisfying assignments.

- The TOUR constraint must be modelled explicitly, for example in the LEGAL function (as above), or by allowing moves to unsatisfying assignments (as discussed in the next section).
- This neighbourhood is called 2-swap, since we swap two edges on the tour.
- It generalises to k-swap, for $k \geq 2$.
- The size of the neighbourhood is $\binom{|s|}{k} \cdot \binom{|E \setminus s|}{k}$:
  - 210 neighbours for our instance and $k = 2$.
  - 350 neighbours for our instance and $k = 3$.

Outline

1. (Meta-) Heuristics for Local Search

- Local Search
- Heuristics
  - Example 1: Graph Partitioning
  - Example 2: Travelling Salesperson
- Meta-Heuristics

2. Constraint-Based Local Search

- Modelling
- Violation Functions
- Probing Functions
- Comparison with CP

3. Example: The COMET System

4. Hybrid Methods

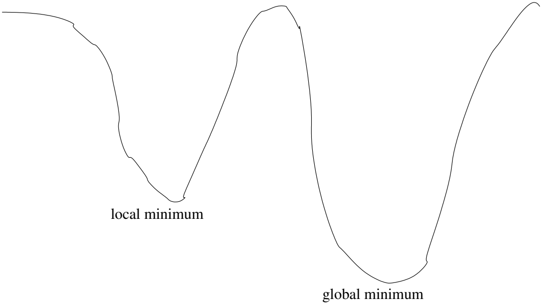
5. Bibliography

Heuristics drive the search to (good enough) solutions:

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

Metaheuristics drive the search to global optima of COST:

- Avoid cycles of moves & escape local optima of COST.
- Explore many parts of the search space.
- Focus on promising parts of the search space.



Examples (Metaheuristics)

- Tabu search (1986):
  forbid recent moves from being done again.
- Simulated annealing (1983):
  perform random moves and accept degrading ones with a probability that decreases over time.
- Genetic algorithms (1975):
  use a pool of candidate solutions and cross them.

Tabu Search (Glover and Laguna, 1997)

- In order to escape local optima, we must be able to accept worse assignments, that is assignments that increase the value of COST.
- To avoid ending up in cycles, tabu search remembers the last λ assignments in a tabu list and makes them tabu (or taboo): moves in this list cannot be chosen, even if this implies increasing the value of COST.

Tabu Search

$s := \textsc{InitialAssignment}()$
$k := 0; s^* := s$ // s^* is the so far best assignment
$\tau := [s]$ // initialise the tabu list
while $\sum_{i=1}^{n} \textsc{Violation}(c_i, s) > 0 \land k < \mu$ **do**
  $k := k + 1; s := \text{Best}(\text{NonTabu}(\textsc{Neighbours}(s), \tau), \tau)$
  $\tau := \tau :: s$ // but keep only the last λ assignments
  if $\textsc{Cost}(s) < \textsc{Cost}(s^*)$ **then**
    $s^* := s$
return s^*

function $\text{NonTabu}(N, \tau)$
return $\{n \in N \mid n \notin \tau\}$

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. **Constraint-Based Local Search**
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

Evaluation of Local Search

We have seen local-search algorithms for two problems:

- It is hard to reuse (parts of) a local-search algorithm of one problem for other problems.
- We want reusable software components!

In constraint-based local search (CBLS) (Van Hentenryck and Michel, 2005):

- A problem is modelled as a conjunction of constraints, whose predicates declaratively encapsulate inference algorithms specific to common combinatorial substructures and are thus reusable.
- A master search algorithm operates on the model, guided by user-indicated/designed (meta-)heuristics.

CBLS by itself makes no contributions to the design of local-search (meta-)heuristics, but it eases their formulation and improves their reusability.

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. **Constraint-Based Local Search**
   - **Modelling**
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

Definition

Each constraint predicate has a violation function: the violation of a constraint is zero if it is satisfied, else a positive value proportional to its dissatisfaction.

Example

For `a <= b`, let α and β be the current values of `a` and `b`: define the violation to be $\alpha - \beta$ if $\alpha \not\leq \beta$, and 0 otherwise.

Definition

A constraint with violation is explicit in a CBLS model and soft: it can be violated during search but ought to be satisfied in a solution.

Definition

A one-way constraint is explicit in a CBLS model and hard: it is kept satisfied during search.

Example

For `p = a * b`, whenever the value α of `a` or the value β of `b` is modified by a move, the value of `p` is automatically modified by the solver so as to remain equal to $\alpha \cdot \beta$.

CBLS solvers offer a syntax for one-way constraints, such as `p <== a * b` in OscaR.cbls, but Gecode and MiniZinc do not make such a distinction.

Definition

An implicit constraint is not in a CBLS model but hard: it is kept satisfied during search by choosing a satisfying initial candidate solution and only making satisfaction-preserving moves, by the use of a constraint-specific neighbourhood.

Example

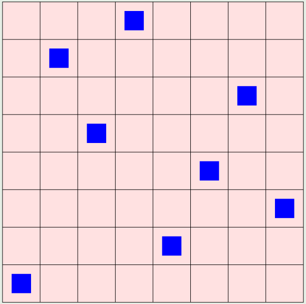
For `all_different(...)`, the initial candidate solution has distinct values for all variables, and the neighbourhood only has moves that swap the values of two variables, assuming the number of variables is equal to the number of values.

When building a CBLS model, a MiniZinc backend must:

- Aptly assort the otherwise all explicit & soft constraints.
- Add a suitable heuristic and meta-heuristic.

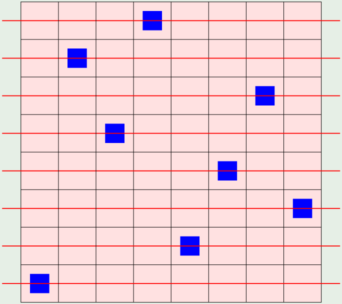
This is much more involved than just flattening and solving.

Example (8 Queens)



Place 8 queens on a chess board such that no two queens attack each other:

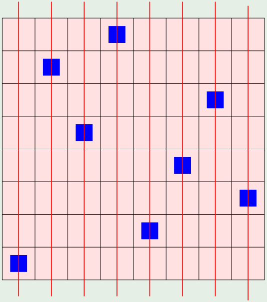
Example (8 Queens)



Place 8 queens on a chess board such that no two queens attack each other:

1. No two queens are on the same row.

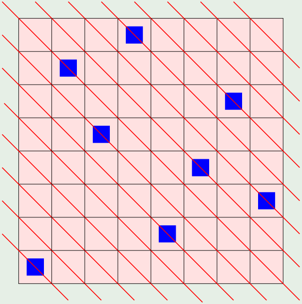
Example (8 Queens)



Place 8 queens on a chess 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.

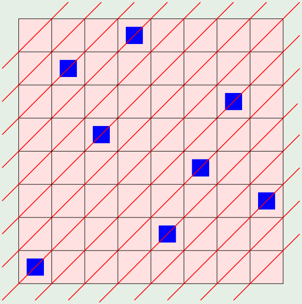
Example (8 Queens)



Place 8 queens on a chess 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.

Example (8 Queens)



Place 8 queens on a chess 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.

Example (8 Queens: CBLS Models)

Let variable $R[i]$ represent the row of the queen in col. i:

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:

Example (8 Queens: CBLS Models)

Let variable $R[i]$ represent the row of the queen in col. i:

1. No two queens are on the same row:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] \neq R[j]$,
   that is `distinct`$([R[1], \ldots, R[8]])$
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:

Example (8 Queens: CBLS Models)

Let variable $R[i]$ represent the row of the queen in col. i:

1. No two queens are on the same row:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] \neq R[j]$,
   that is `distinct`$([R[1], \ldots, R[8]])$
2. No two queens are on the same column:
   Guaranteed by the choice of the decision variables.
3. No two queens are on the same down-diagonal:
4. No two queens are on the same up-diagonal:

Example (8 Queens: CBLS Models)

Let variable $R[i]$ represent the row of the queen in col. i:

1. No two queens are on the same row:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] \neq R[j]$,
   that is $\texttt{distinct}([R[1], \ldots, R[8]])$
2. No two queens are on the same column:
   Guaranteed by the choice of the decision variables.
3. No two queens are on the same down-diagonal:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] - i \neq R[j] - j$,
   that is $\texttt{distinct}([R[1] - 1, \ldots, R[8] - 8])$
4. No two queens are on the same up-diagonal:

Example (8 Queens: CBLS Models)

Let variable $R[i]$ represent the row of the queen in col. i:

1. No two queens are on the same row:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] \neq R[j]$,
   that is `distinct`$([R[1], \ldots, R[8]])$
2. No two queens are on the same column:
   Guaranteed by the choice of the decision variables.
3. No two queens are on the same down-diagonal:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] - i \neq R[j] - j$,
   that is `distinct`$([R[1] - 1, \ldots, R[8] - 8])$
4. No two queens are on the same up-diagonal:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] + i \neq R[j] + j$,
   that is `distinct`$([R[1] + 1, \ldots, R[8] + 8])$

Example (8 Queens: CBLS Models)

Let variable $R[i]$ represent the row of the queen in col. i:

1. No two queens are on the same row:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] \neq R[j]$,
   that is $\texttt{distinct}([R[1], \ldots, R[8]])$
2. No two queens are on the same column:
   Guaranteed by the choice of the decision variables.
3. No two queens are on the same down-diagonal:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] - i \neq R[j] - j$,
   that is $\texttt{distinct}([R[1] - 1, \ldots, R[8] - 8])$
4. No two queens are on the same up-diagonal:
   $\forall i, j \in 1..8$ **where** $i < j : R[i] + i \neq R[j] + j$,
   that is $\texttt{distinct}([R[1] + 1, \ldots, R[8] + 8])$

Better model: Make the row constraint implicit, by using a random permutation of 1..8 as initial assignment and using a neighbourhood that keeps the row constraint satisfied.

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. **Constraint-Based Local Search**
   - Modelling
   - **Violation Functions**
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

Constraint Predicates in Local Search

Every predicate of a soft constraint c is equipped with:

- A constraint violation function $\textsc{Violation}(c, s)$, which estimates how much c is violated under the current assignment s: $\textsc{Violation}(c, s) = 0$ if and only if c is satisfied, and $\textsc{Violation}(c, s) > 0$ otherwise.
- A variable violation function $\textsc{Violation}(c, s, x)$, which estimates how much a suitable change of the value of the decision variable x can decrease $\textsc{Violation}(c, s)$.
- ... (to be continued)

At the constraint-system level:

- The system constraint violation under s of a constraint system $\{c_1, \dots, c_n\}$ is $\sum_{i=1}^{n} \textsc{Violation}(c_i, s)$.
- The system variable violation under s of a variable x in a system $\{c_1, \dots, c_n\}$ is $\sum_{i=1}^{n} \textsc{Violation}(c_i, s, x)$.

Violations

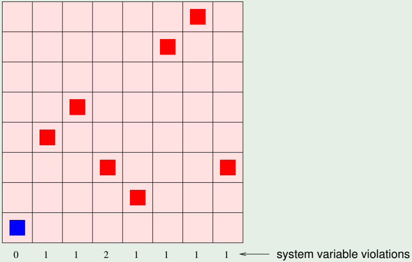
Example ($x \neq y$)

- When $x = 4$ and $y = 4$:
  - The constraint violation is 1: the constraint is violated.
  - The variable violations of x and y are both 1.
- When $x = 4$ and $y = 5$:
  - The constraint violation is 0: the constraint is satisfied.
  - The variable violations of x and y are both 0.

Example (`distinct`$([x_1, x_2, x_3, x_4])$)

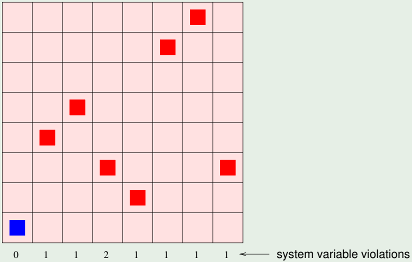
- When $x_1 = 5$, $x_2 = 5$, $x_3 = 5$, $x_4 = 6$, with domain D:
  - The constraint violation is 2, since at least two variables must be changed to reach a satisfying assignment: $\textsc{Violation} = \sum_{v \in D} \max(\text{occ}[v] - 1,\ 0)$, where $\text{occ}[v]$ stores the current number of occurrences of value v.
  - The variable violations of x_1, x_2, x_3 are 1, and 0 for x_4.

Example (8 Queens: Violations)



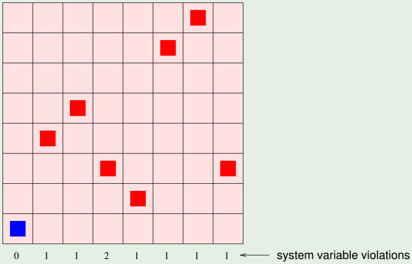
- $\texttt{distinct}([R[1], \ldots, R[8]])$
- $\texttt{distinct}([R[1]-1, \ldots, R[8]-8])$
- $\texttt{distinct}([R[1]+1, \ldots, R[8]+8])$

Example (8 Queens: Violations)



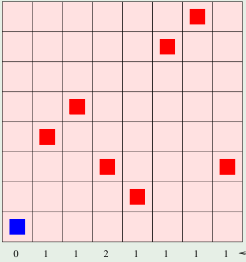
- $\texttt{distinct}([R[1], \ldots, R[8]])$
  The violation of $\texttt{distinct}([8, 5, 4, 6, 7, 2, 1, 6])$ is 1.
- $\texttt{distinct}([R[1] - 1, \ldots, R[8] - 8])$
- $\texttt{distinct}([R[1] + 1, \ldots, R[8] + 8])$

Example (8 Queens: Violations)



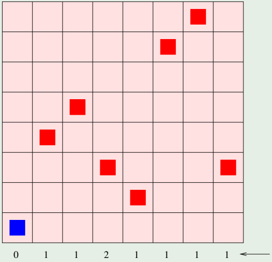
- distinct$([R[1], \ldots, R[8]])$
  The violation of distinct$([8, 5, 4, 6, 7, 2, 1, 6])$ is 1.
- distinct$([R[1] - 1, \ldots, R[8] - 8])$
  The violation of distinct$([7, 3, 1, 2, 2, -4, -6, -2])$ is 1.
- distinct$([R[1] + 1, \ldots, R[8] + 8])$

Example (8 Queens: Violations)



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

Example (8 Queens: Violations)



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

The system constraint violation is $1+1+2=4$.

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics

2. **Constraint-Based Local Search**
   - Modelling
   - Violation Functions
   - **Probing Functions**
   - Comparison with CP

3. Example: The COMET System

4. Hybrid Methods

5. Bibliography

Constr. Predicates in Local Search (cont'd)

Every predicate of a soft constraint c is also equipped with:

- An assignment delta function $\text{DELTA}(c, s, x := v)$, which estimates the increase of $\text{VIOLATION}(c, s)$ upon a probed $x := v$ assignment move for variable x and its domain value v.
- A swap delta function $\text{DELTA}(c, s, x :=: y)$, which estimates the increase of $\text{VIOLATION}(c, s)$ upon a probed $x :=: y$ swap move for two variables x and y.

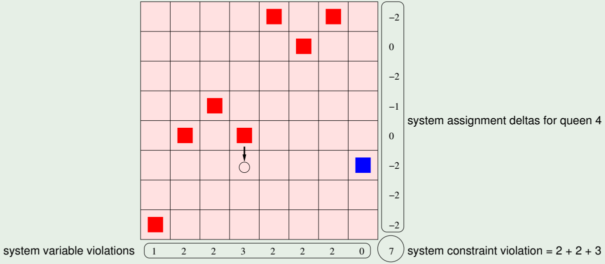
The more negative a delta the better!

At the constraint-system level:

- The system assignment delta under s of $x := v$ in a system $\{c_1, \dots, c_n\}$ is $\sum_{i=1}^{n} \text{DELTA}(c_i, s, x := v)$.
- The system swap delta under s of $x :=: y$ in a system $\{c_1, \dots, c_n\}$ is $\sum_{i=1}^{n} \text{DELTA}(c_i, s, x :=: y)$.

Other kinds of moves can be added.

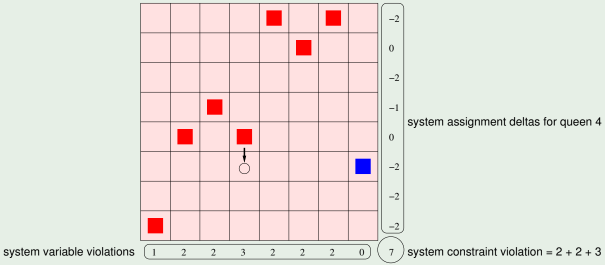
Example (8 Queens: Computing Deltas in $\mathcal{O}(1)$ Time)



- $\texttt{distinct}([R[1], \ldots, R[4], \ldots, R[8]])$
- $\texttt{distinct}([R[1] - 1, \ldots, R[4] - 4, \ldots, R[8] - 8])$
- $\texttt{distinct}([R[1] + 1, \ldots, R[4] + 4, \ldots, R[8] + 8])$

The violation increases by $[\text{occ}[v] \geq 1] - [\text{occ}[s(x)] \geq 2]$ upon $x := v$.

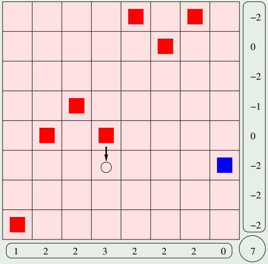
Example (8 Queens: Computing Deltas in $\mathcal{O}(1)$ Time)



- $\texttt{distinct}([R[1], \ldots, {\color{red}R[4]}, \ldots, R[8]])$
  Delta of $R[4] := 6$ in $\texttt{distinct}([8, 5, 4, {\color{red}5}, 1, 2, 1, 6])$ is ± 0.
- $\texttt{distinct}([R[1] - 1, \ldots, {\color{red}R[4] - 4}, \ldots, R[8] - 8])$
- $\texttt{distinct}([R[1] + 1, \ldots, {\color{red}R[4] + 4}, \ldots, R[8] + 8])$

The violation increases by $[\mathrm{occ}[v] \geq 1] - [\mathrm{occ}[s(x)] \geq 2]$ upon $x := v$.

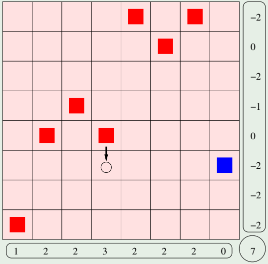
Example (8 Queens: Computing Deltas in $\mathcal{O}(1)$ Time)



- $\texttt{distinct}([R[1], \ldots, R[4], \ldots, R[8]])$
  Delta of $R[4] := 6$ in $\texttt{distinct}([8, 5, 4, 5, 1, 2, 1, 6])$ is ± 0.
- $\texttt{distinct}([R[1] - 1, \ldots, R[4] - 4, \ldots, R[8] - 8])$
  Delta of $R[4] := 6$ in $\texttt{distinct}([7, 3, 1, 1, -4, -4, -6, -2])$ is -1.
- $\texttt{distinct}([R[1] + 1, \ldots, R[4] + 4, \ldots, R[8] + 8])$

The violation increases by $[\text{occ}[v] \geq 1] - [\text{occ}[s(x)] \geq 2]$ upon $x := v$.

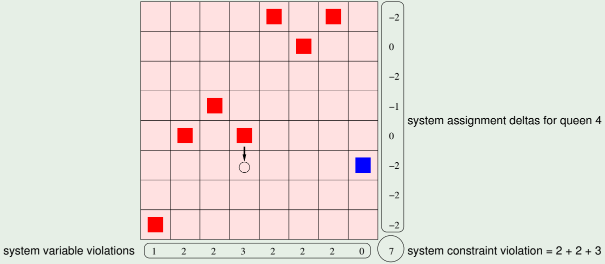
Example (8 Queens: Computing Deltas in $\mathcal{O}(1)$ Time)



- $\texttt{distinct}([R[1],\ldots,R[4],\ldots,R[8]])$
  Delta of $R[4] := 6$ in $\texttt{distinct}([8,5,4,5,1,2,1,6])$ is ± 0.
- $\texttt{distinct}([R[1]-1,\ldots,R[4]-4,\ldots,R[8]-8])$
  Delta of $R[4] := 6$ in $\texttt{distinct}([7,3,1,1,-4,-4,-6,-2])$ is -1.
- $\texttt{distinct}([R[1]+1,\ldots,R[4]+4,\ldots,R[8]+8])$
  Delta of $R[4] := 6$ in $\texttt{distinct}([9,7,7,9,6,8,8,14])$ is -1.

The violation increases by $[\mathrm{occ}[v] \geq 1] - [\mathrm{occ}[s(x)] \geq 2]$ upon $x := v$.

Example (8 Queens: Computing Deltas in $\mathcal{O}(1)$ Time)



- $\mathtt{distinct}([R[1], \ldots, \color{red}{R[4]}, \ldots, R[8]])$
  Delta of $R[4] := 6$ in $\mathtt{distinct}([8, 5, 4, \color{red}{5}, 1, 2, 1, 6])$ is ± 0.
- $\mathtt{distinct}([R[1] - 1, \ldots, \color{red}{R[4] - 4}, \ldots, R[8] - 8])$
  Delta of $R[4] := 6$ in $\mathtt{distinct}([7, 3, 1, \color{red}{1}, -4, -4, -6, -2])$ is -1.
- $\mathtt{distinct}([R[1] + 1, \ldots, \color{red}{R[4] + 4}, \ldots, R[8] + 8])$
  Delta of $R[4] := 6$ in $\mathtt{distinct}([9, 7, 7, \color{red}{9}, 6, 8, 8, 14])$ is -1.

The system assignment delta of $R[4] := 6$ is $0 + (-1) + (-1) = -2$.

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. **Constraint-Based Local Search**
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. Bibliography

Constraint Predicates in Local Search (end)

- The functions equipping a constraint predicate can be used to guide the local search:
  - The constraint violation function helps to select promising constraint(s) in order to select promising decision variable(s) to reassign in a move.
  - The variable violation function helps to select promising decision variable(s) to reassign in a move.
  - The delta functions help to select a move in a good direction for a variable, constraint, or constraint system.
- The violation functions are the counterpart of the subsumption checking of systematic CP-style solving.
- The probing functions are the counterpart of the propagators of systematic CP-style solving.
- These functions must be implemented for highest time and space efficiency, as they may be queried in the probing of the neighbourhood at each search iteration.

Modelling for Local Search

When solving combinatorial problems by local search, the idea is often to exploit the presence of symmetries by doing nothing, rather than by making the search space smaller as with CP / MIP / SAT / SMT-style systematic search.

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics

2. Constraint-Based Local Search
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP

3. **Example: The COMET System**

4. Hybrid Methods

5. Bibliography

The COMET System

COMET was a language and a tool for the modelling and solving of constraint problems.

COMET had a CBLS back-end (Van Hentenryck and Michel, 2005), as well as CP (systematic search with propagation) and MIP (mixed integer linear programming) back-ends:

- High-level software components (constraint predicates) for formulating constraint models of problems.
- High-level constructs for specifying search algorithms.
- An open architecture allowing user-defined extensions.

COMET was free of charge for academic purposes.
It inspired, among others, the CBLS back-end of OSCAR, available for free at http://oscarlib.org.

Example (8 Queens: COMET CBLS Model)

```
import cotls;
Solver<LS> m();
int n = 8;
range Size = 1..n;
UniformDistribution distr(Size);
var{int} R[Size](m,Size) := distr.get();
ConstraintSystem<LS> S(m);
S.post(alldifferent(R));
S.post(alldifferent(all(i in Size) R[i]-i));
S.post(alldifferent(all(i in Size) R[i]+i));
m.close();
```

Define an array `R` of 8 variables and initialise each variable with a random (possibly repeated) value in the domain 1..8.

Better: Make the row constraint implicit, by using a random permutation of 1..8 as initial assignment.

Example (8 Queens: COMET CBLS Search)

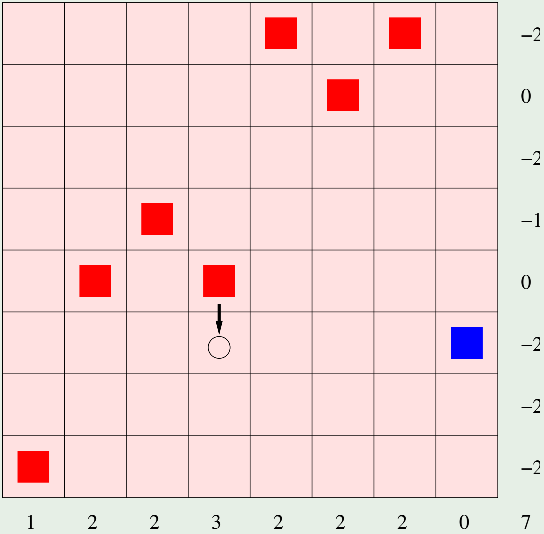
```
int iter = 0;
while (S.violations() > 0 && iter < 50 * n) {
  selectMax(i in Size)(S.violations(R[i]))
    selectMin(r in Size)(S.getAssignDelta(R[i],
      R[i] := r;
  iter++;
}
```

In words:

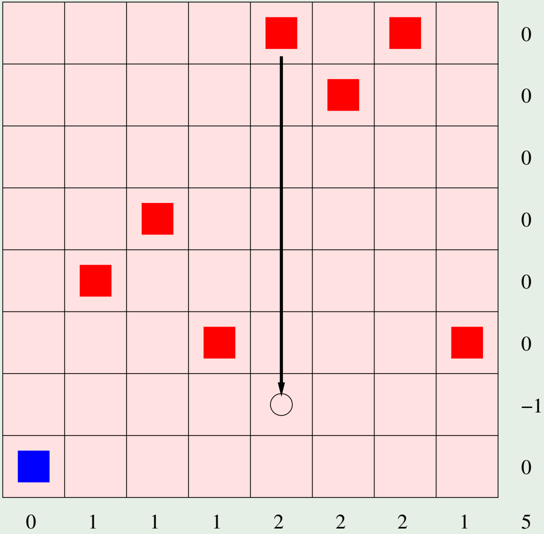
while there are a violated constraint in system `S` and iterations left **do**
  select a variable `R[i]` with the maximum violation in system `S`
    select a value `r` with the minimum assignment delta for `R[i]` in `S`
      assign value `r` to decision variable `R[i]`
  increment the iteration counter

Better: Keep the row constraint satisfied by a neighbourhood of swap moves `R[i] :=: R[j]`.

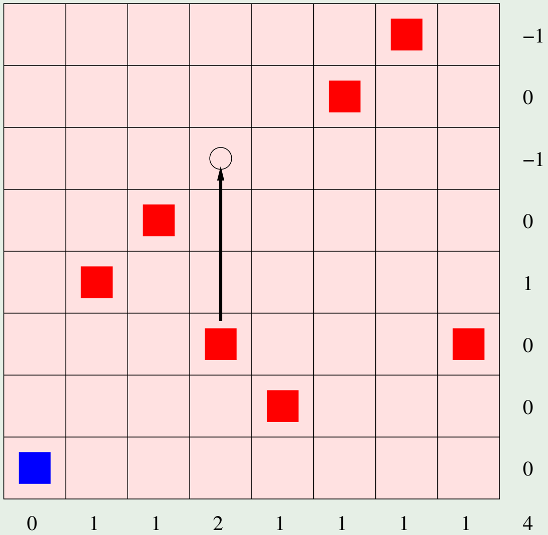
Example (8 Queens: Sample Run)



Example (8 Queens: Sample Run)



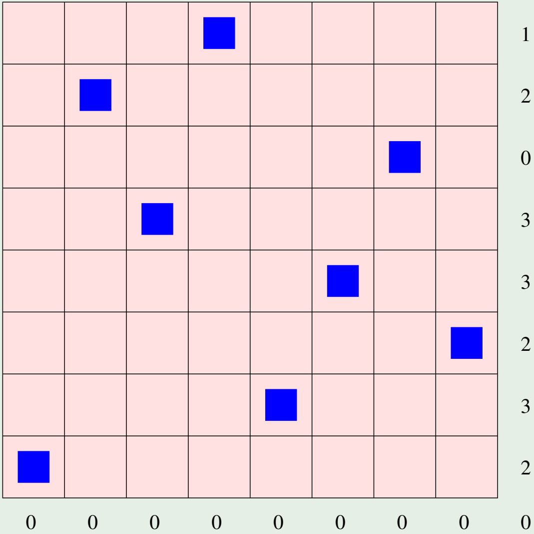
Example (8 Queens: Sample Run)



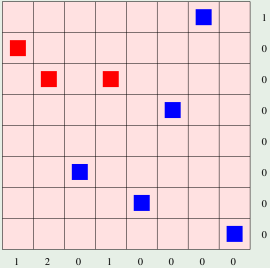
Example (8 Queens: Sample Run)

. . . and so on, until . . .

Example (8 Queens: Sample Run)



Example (8 Queens: Local Minimum)



- Queen 2 is selected, as the only most violating queen.
- Queen 2 is placed on one of rows 2 to 8, as the system violation will increase by 1 if she is placed on row 1.
- Queen 2 remains the only most violating queen!
- Queen 2 is selected over and over again.

A meta-heuristic is needed to escape this local minimum.

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. Constraint-Based Local Search
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. **Hybrid Methods**
5. Bibliography

Hybridising Systematic and Local Search

Compare with the generic algorithm of slide 16:

Example (Large Neighbourhood Search (Shaw, 1998))

$p :=$ the CSP where all variables have their full domains
$s :=$ First(Solutions(p)) // systematic search
$k := 0$; $s^* := s$ // s^* is the so far best assignment
while $k < \mu$ **do**
  $k := k + 1$
  $p :=$ the COP where some variables are frozen (e.g., fixed to their values in s^*), the other variables are thawed (e.g., have their full domains), and the objective function is strictly bounded by $f(s^*)$
  $s :=$ SELECT(Solutions(p), _) // limited syst. search
  if s exists **then** $s^* := s$
return s^*

Outline

1. (Meta-) Heuristics for Local Search
   - Local Search
   - Heuristics
     - Example 1: Graph Partitioning
     - Example 2: Travelling Salesperson
   - Meta-Heuristics
2. Constraint-Based Local Search
   - Modelling
   - Violation Functions
   - Probing Functions
   - Comparison with CP
3. Example: The COMET System
4. Hybrid Methods
5. **Bibliography**

Reference

- Hoos, Holger H. and Stützle, Thomas.
  Stochastic Local Search: Foundations & Applications.
  Elsevier / Morgan Kaufmann, 2004.
- Glover, Fred W. and Laguna, Manuel.
  Tabu Search.
  Kluwer Academic Publishers, 1997.
- Van Hentenryck, Pascal and Michel, Laurent.
  Constraint-Based Local Search.
  The MIT Press, 2005.
- Shaw, Paul.
  Using constraint programming and local search methods to solve vehicle routing problems.
  Proceedings of CP 1998, Lecture Notes in Computer Science, volume 1520, pages 417–431, Springer, 1998.