Lecture 10
Stochastic Local Search
(4.8)
Lecture Overview

Recap
• Domain Splitting for Arc Consistency
• Local Search
• Stochastic Local Search (SLS)
• Comparing SLS
Arc Consistency Algorithm

• Go through all the arcs in the network
• Make each arc consistent by pruning the appropriate domain, when needed
• Reconsider arcs that could be turned back to being inconsistent by this pruning
• Eventually reach a ‘fixed point’: all arcs consistent
Which arcs need to be reconsidered?

- When AC reduces the domain of a variable $X$ to make an arc $\langle X, c \rangle$ arc consistent, which arcs does it need to reconsider?

AC does not need to reconsider other arcs

- If arc $\langle Y, c \rangle$ was arc consistent before, it will still be arc consistent.
  “Consistent before” means each element $y_i$ in $Y$ must have an element $x_i$ in $X$ that satisfies the constraint. Those $x_i$ would not be pruned from $\text{Dom}(X)$, so arc $\langle Y, c \rangle$ stays consistent

- If an arc $\langle X, c_i \rangle$ was arc consistent before, it will still be arc consistent
  The domains of $Z_i$ have not been touched

- Nothing changes for arcs of constraints not involving $X
Arc Consistency Algorithm: Complexity

- Let’s determine Worst-case complexity of this procedure (compare with DFS $O(d^n)$)
  - let the max size of a variable domain be $d$
  - let the number of variables be $n$
  - The max number of binary constraints is $\frac{n \cdot (n-1)}{2}$
  - How many times, at worst, the same arc can be inserted in the ToDoArc list? $O(d)$
  - How many steps are involved in checking the consistency of an arc? $O(d^2)$

- Overall complexity: $O(n^2d^3)$

- Compare to $O(d^N)$ of DFS. Arc consistency is MUCH faster

So did we find a polynomial algorithm to solve CPSs?
No, AC does not always solve the CPS. It is a way to possibly simplify the original CSP and make it easier to solve
Arc Consistency Algorithm: Interpreting Outcomes

• Three possible outcomes (when all arcs are arc consistent):
  • Each domain has a single value,
    ✓ e.g. built-in AlSpace example “Scheduling problem 1”
    ✓ We have: a (unique) solution.
  • At least one domain is empty,
    ✓ We have: No solution! All values are ruled out for this variable.
    ✓ e.g. try this graph (can easily generate it by modifying Simple Problem 2)
  • Some domains have more than one value,
    ✓ There may be: one solution, multiple ones, or none
    ✓ Need to solve this new CSP (usually simpler) problem:
      - same constraints, domains have been reduced
Lecture Overview

- Recap
- Domain Splitting for Arc Consistency
  - Local Search
  - Stochastic Local Search (SLS)
  - Comparing SLS
Search vs. Domain Splitting

- Arc consistency ends: Some domains have more than one value → may or may not have a solution
  A. Apply Depth-First Search with Pruning or
  
  B. **Split the problem** in a number of disjoint cases CSP\(_i\): for instance

  CSP with \(\text{dom}(X) = \{x_1, x_2, x_3, x_4\}\) becomes

  CSP\(_1\) with \(\text{dom}(X) = \{x_1, x_2\}\) and
  CSP\(_2\) with \(\text{dom}(X) = \{x_3, x_4\}\)

- Solution to CSP is the **union** of solutions to CSP\(_i\)
Example

Run “Scheduling Problem 2” in Alspace
- Try spitting on E (select 2 first, then 3 then 4)
Another Example

“Crossword 1” in Aispace,

- try splitting on D3 and then A3 (always “select half”)
Domain splitting

For each subCSP, which arcs have to be on the ToDoArcs list when we get the subCSP by splitting the domain of X?

A. arcs \(<Z_i, r(Z_i, X)>\)

B. arcs \(<Z_i, r(Z_i, X)>\) and \(<X, r(Z_i, X)>\)

C. arcs \(<Z_i, r(Z, X)>\) and \(<Y, r(X, Y)>\)

D. All arcs in the figure

E. All 8 arcs related to constraints involving X: \((c_1, c_2, c_3, c)\)
Domain splitting

- For each subCSP, which arcs have to be on the ToDoArcs list when we get the subCSP by splitting the domain of \( X \)?

\[
\begin{align*}
\text{Z}_1 & \quad \text{THESE} \quad \text{c}_1 \\
\text{Z}_2 & \quad \text{THESE} \quad \text{c}_2 \\
\text{Z}_3 & \quad \text{THESE} \quad \text{c}_3 \\
\text{X} & \quad \text{c} \\
\text{Y} & \quad \text{c}_4 \\
\text{A} &
\end{align*}
\]

C. arcs \( < Z_i, r(Z,X) > \) and \( < Y, r(X,Y) > \)
Searching by domain splitting

CSP, apply AC

If domains with multiple values

Split on one

CSP₁, apply AC

If domains with multiple values

Split on one

CSP₂, apply AC

If domains with multiple values

Split on one
Another formulation of CSP as search

Arc consistency with domain splitting

- States: vector \((D(V_1), ..., D(V_n))\) of remaining domains, with \(D(V_i) \subseteq \text{dom}(V_i)\) for each \(V_i\)
- Start state: vector of original domains \((\text{dom}(V_1), ..., \text{dom}(V_n))\)
- Successor function:
  - reduce one of the domains + run arc consistency
- Goal state: vector of unary domains that satisfies all constraints
  - That is, only one value left for each variable
  - The assignment of each variable to its single value is a model
- Solution: that assignment
Arc consistency + domain splitting: example

3 variables: A, B, C
Domains: all \{1,2,3,4\}
A=B, B=C, A\neq C

\[
\begin{align*}
A &\in \{1,3\} & A &\in \{2,4\} \\
\{1,3\}, \{1,2,3,4\}, \{1,2,3,4\} &\rightarrow AC (arc consistency) \rightarrow \{1,2,3,4\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{1,3\}, \{1,3\}, \{1,3\} &\rightarrow AC \rightarrow \{1,3\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{1,3\}, \{1\}, \{1,3\} &\rightarrow AC \rightarrow \{1,3\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{2,4\}, \{1,2,3,4\}, \{1,2,3,4\} &\rightarrow AC \rightarrow \{2,4\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{2,4\}, \{2,4\}, \{2,4\} &\rightarrow AC \rightarrow \{2,4\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{2,4\}, \{2\}, \{2,4\} &\rightarrow AC \rightarrow \{2,4\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{2,4\}, \{4\}, \{2,4\} &\rightarrow AC \rightarrow \{2,4\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{2,4\}, \{4\}, \{2,4\} &\rightarrow AC \rightarrow \{2,4\}, \{1,2,3,4\}, \{1,2,3,4\} \\
\{\}, \{\}, \{\} &\rightarrow AC \rightarrow \{\}, \{\}, \{\} \\
\{\}, \{\}, \{\} &\rightarrow AC \rightarrow \{\}, \{\}, \{\} \\
\{\}, \{\}, \{\} &\rightarrow AC \rightarrow \{\}, \{\}, \{\} \\
\{\}, \{\}, \{\} &\rightarrow AC \rightarrow \{\}, \{\}, \{\}
\end{align*}
\]

No solution

No solution

No solution
Arc consistency + domain splitting: another example

3 variables: A, B, C
Domains: all \{1,2,3,4\}
A=B, B=C, A=C

Solution

\(\{1,3\} \quad \text{and} \quad \{2,4\}\)
Searching by domain splitting

If domains with multiple values

CSP, apply AC

Split on one

CSP₁, apply AC

If domains with multiple values

Split on one

CSP₂, apply AC

If domains with multiple values

Split on one

How many CSPs do we need to keep around at a time?
Assume solution at depth \( m \) and \( b \) children at each split

A. \( O(bm) \)
B. \( O(b^m) \)
B. \( O(m^b) \)
B. \( O(b^2) \)
Searching by domain splitting

CSP, apply AC

If domains with multiple values

Split on one

CSP₁, apply AC

If domains with multiple values

Split on one

CSP₂, apply AC

If domains with multiple values

Split on one

How many CSPs do we need to keep around at a time?
Assume solution at depth \( m \) and \( b \) children at each split

\( O(bm) \): It is DFS
Systematically solving CSPs: Summary

• Build Constraint Network
• Apply Arc Consistency
  • One domain is empty →
  • Each domain has a single value →
  • Some domains have more than one value →

• Apply Depth-First Search with Pruning OR
• Split the problem in a number of disjoint cases
  • Apply Arc Consistency to each case, and repeat
## Limitation of Systematic Approaches

- Many CSPs (scheduling, DNA computing, etc.) are simply too big for systematic approaches.
- If you have $10^5$ vars with $\text{dom}(\text{var}_i) = 10^4$

<table>
<thead>
<tr>
<th>Systematic Search</th>
<th>Constraint Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching factor $b = $</td>
<td>Size =</td>
</tr>
<tr>
<td>Solution depth $d = $</td>
<td>Complexity of AC =</td>
</tr>
<tr>
<td>Complexity =</td>
<td></td>
</tr>
</tbody>
</table>
Limitation of Systematic Approaches

- Many CSPs are simply too big for systematic approaches
- If you have $10^5$ vars with dom(var$_i$) = $10^4$

<table>
<thead>
<tr>
<th>Systematic Search</th>
<th>Constraint Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching factor $b = 10^4$</td>
<td>Size = $O(10^5 + 10^5 \times 10^5)$</td>
</tr>
<tr>
<td>Solution depth $d = 10^5$</td>
<td>Time Complexity of AC = $O((10^5)^2 \times 10^4^3)$</td>
</tr>
<tr>
<td>Time Complexity = $O((10^4)^{10^5})$</td>
<td></td>
</tr>
</tbody>
</table>

21
Learning Goals for CSP

• Define possible worlds in term of variables and their domains
• Compute number of possible worlds on real examples
• Specify constraints to represent real world problems differentiating between:
  • Unary and k-ary constraints
  • List vs. function format
• Verify whether a possible world satisfies a set of constraints (i.e., whether it is a model, a solution)
• Implement the Generate-and-Test Algorithm. Explain its disadvantages.
• Solve a CSP by search (specify neighbors, states, start state, goal state). Compare strategies for CSP search. Implement pruning for DFS search in a CSP.
• Define/read/write/trace/debug the arc consistency algorithm. Compute its complexity and assess its possible outcomes
• Define/read/write/trace/debug domain splitting and its integration with arc consistency
Lecture Overview

• Recap
• Domain Splitting for Arc Consistency
Local Search
• Stochastic Local Search (SLS)
• Comparing SLS
Local Search: Motivation

- Solving CSPs is NP-hard
  - Search space for many CSPs is huge
  - Exponential in the number of variables
  - Even arc consistency with domain splitting is often not enough

- Alternative: local search
  - use algorithms that search the space locally, rather than systematically
  - Often finds a solution quickly, but are not guaranteed to find a solution if one exists (thus, cannot prove that there is no solution)
Local Search

• Idea:
  • Consider the space of complete assignments of values to variables (all possible worlds)
  • Neighbours of a current node are similar variable assignments
  • Move from one node to another according to a function that scores how good each assignment is

• Useful method in practice
  - Best available method for many constraint satisfaction and constraint optimization problems
Definition: A local search problem consists of a:

**CSP**: a set of variables, domains for these variables, and constraints on their joint values.

A **node** in the search space will be a complete assignment to all of the variables.

**Neighbour relation**: an edge in the search space will exist when the neighbour relation holds between a pair of nodes.

**Scoring function**: $h(n)$, judges cost of a node (want to minimize)
- E.g. the number of constraints violated in node $n$.
- E.g. the cost of a state in an optimization context.
• Given the set of variables \( \{ V_1, \ldots, V_n \} \), each with domain \( \text{Dom}(V_i) \)
• The start node is any assignment \( \{ V_1 / \nu_1, \ldots, V_n / \nu_n \} \).
• The **neighbors** of node with assignment
  \[ A = \{ V_1 / \nu_1, \ldots, V_n / \nu_n \} \]
  are nodes with assignments that **differ from A for one value only**
Search Space

- Only the current node is kept in memory at each step.
- Very different from the systematic tree search approaches we have seen so far!
- Local search does NOT backtrack!
Iterative Best Improvement

• How to determine the neighbor node to be selected?
• Iterative Best Improvement:
  • select the neighbor that optimizes some evaluation function
• Which strategy would make sense? Select neighbor with ...

A. Maximal number of constraint violations
B. Similar number of constraint violations as current state
C. No constraint violations
D. Minimal number of constraint violations
Iterative Best Improvement

• How to determine the neighbor node to be selected?

• Iterative Best Improvement:
  • select the neighbor that optimizes some evaluation function

• Which strategy would make sense? Select neighbour with …

  Minimal number of constraint violations

• Evaluation function:
  \( h(n) \): number of constraint violations in state \( n \)

• Greedy descent: evaluate \( h(n) \) for each neighbour, pick the neighbour \( n \) with minimal \( h(n) \)

• Hill climbing: equivalent algorithm for maximization problems
  • Here: Maximize number of satisfied constraints
Hill Climbing

NOTE: Everything that will be said for Hill Climbing is also true for Greedy Descent.

evaluation/scoring function

Current state/possible world

(assume both vars have integer domain)
Example: N-Queens

• Put n queens on an n × n board with no two queens on the same row, column, or diagonal (i.e. attacking each other)

• Positions a queen can attack
Example: N-queen as a local search problem

**CSP**: N-queen CSP
- One variable per column; domains \{1,\ldots,N\} \to row where the queen in the \(i\)th column seats;
- Constraints: no two queens in the same row, column or diagonal

**Neighbour relation**: value of a single column differs

**Scoring function**: number of constraint violations (i.e., number of attacks)
Example: Greedy descent for N-Queen

For each column, assign randomly each queen to a row (a number between 1 and N)

Repeat
  • For each column & each number: Evaluate how many constraint violations changing the assignment to that number would yield
  • Choose the column and number that leads to the fewest violated constraints; change it

Until solved

Each cell lists h (i.e. #constraints unsatisfied) if you move the queen in that column into the cell
h = 5

h = ?

h = ?
General Local Search Algorithm

1: Procedure Local-Search(V,dom,C)

2: Inputs

3: V: a set of variables

4: dom: a function such that dom(X) is the domain of variable X

5: C: set of constraints to be satisfied

6: Output complete assignment that satisfies the constraints

7: Local

8: A[V] an array of values indexed by V

9: repeat

10: for each variable X do

11: A[X] ←a random value in dom(X);

12: while (stopping criterion not met & A is not a satisfying assignment)

13: Select a variable Y and a value V ∈dom(Y)

14: Set A[Y] ←V

15: if (A is a satisfying assignment) then

16: return A

17: until termination
General Local Search Algorithm

1: **Procedure** Local-Search(V, dom, C)
2:    **Inputs**
3:       V: a set of variables
4:       dom: a function such that dom(X) is the domain of variable X
5:       C: set of constraints to be satisfied
6:    **Output** complete assignment that satisfies the constraints
7:    **Local**
8:       A[V] an array of values indexed by V
9:    **repeat**
10:       **for each** variable X **do**
11:         A[X] ← a random value in dom(X);
12:    **until** termination
13:       **while** (stopping criterion not met & A is not a satisfying assignment)
14:         Select a variable Y and a value V ∈ dom(Y)
15:         Set A[Y] ← V
16:    **if** (A is a satisfying assignment) **then**
17:       **return** A
18:    **until** termination

Random initialization

Local Search Step
Based on local information.
E.g., for each neighbour evaluate how many constraints are unsatisfied.

Greedy descent: select Y and V to minimize #unsatisfied constraints at each step
Example: N-Queens

Each cell lists $h$ (i.e. #constraints unsatisfied) if you move the queen from that column into the cell.
Example: N-Queens

- Which move should we pick in this situation?
The problem of local minima

- Which move should we pick in this situation?
  - Current cost: $h=1$
  - No single move can improve on this
  - In fact, every single move only makes things worse ($h \geq 2$)

- Locally optimal solution
  - Since we are minimizing: local minimum
Problems with Iterative Best Improvement

It gets misled by locally optimal points
• (Local Maxima/ Minima)

Most research in local search is about finding effective mechanisms for escaping from local minima/maxima
Lecture Overview

• Recap
• Domain Splitting for Arc Consistency
• Local Search

Stochastic Local Search (SLS)
• Comparing SLS
Stochastic Local Search

• We will use greedy steps to find local minima
  • Move to neighbour with best evaluation function value

• We will use randomness to escape local minima
Stochastic Local Search (SLS) for CSPs

- **Start node**: random assignment
- **Goal**: assignment with zero unsatisfied constraints
- **Heuristic function $h$**: number of unsatisfied constraints
  - Lower values of the function are better

- Stochastic local search is a mix of:
  - **Greedy descent**: move to neighbor with lowest $h$
  - **Random walk**: take some random steps
    - i.e., move to a neighbor with some randomness
  - **Random restart**: reassigning values to all variables
General SLS Algorithm

1: Procedure Local-Search(V,dom,C)
2: Inputs
3: V: a set of variables
4: dom: a function such that dom(X) is the domain of variable X
5: C: set of constraints to be satisfied
6: Output complete assignment that satisfies the constraints
7: Local
8: A[V] an array of values indexed by V
9: repeat
10: for each variable X do
11: A[X] ← a random value in dom(X);
12: Random restart
13: while (stopping criterion not met & A is not a satisfying assignment)
14: Select a variable Y and a value V ∈ dom(Y)
15: Set A[Y] ← V
16: if (A is a satisfying assignment) then
17: return A
18: until termination

Extreme case 1: random sampling.
Restart at every step:
Stopping criterion is “true”
General SLS Algorithm

1: **Procedure** Local-Search(V, dom, C)

2: **Inputs**

3: V: a set of variables

4: dom: a function such that dom(X) is the domain of variable X

5: C: set of constraints to be satisfied

6: **Output** complete assignment that satisfies the constraints

7: **Local**

8: A[V] an array of values indexed by V

9: repeat

10: for each variable X do

11: A[X] ← a random value in dom(X);

12: while (stopping criterion not met & A is not a satisfying assignment)

13: Select a variable Y and a value V ∈ dom(Y)

14: Set A[Y] ← V

15: if (A is a satisfying assignment) then

16: return A

17: until termination

Extreme case 2: greedy descent
Select the neighbor with best h value (select at random among neighbors with same h)

Random restart
Tracing SLS algorithms in Alspace

• Let’s look at these algorithms in Alspace:
  - Greedy Descent
  - Random Sampling

• Simple scheduling problem 2 in Alspace:

![Alspace](image)

Stochastic Local Search Based CSP Solver

version 4.6.0
Initial assignment: 6 unsatisfied constraints

New assignment for D generates 4 unsatisfied constraints
Greedy descent vs. Random sampling

• **Greedy descent** is
  • good for finding local minima
  • bad for exploring new parts of the search space

• **Random sampling** is
  • good for exploring new parts of the search space
  • bad for finding local minima

• A mix of the two, plus some additional random choices, can work very well
Which randomized method would work best in each of the these two search spaces?

A. Greedy descent with random steps best on 1
   Greedy descent with random restart best on 2

B. Greedy descent with random steps best on 2
   Greedy descent with random restart best on 1

C. equivalent
Which randomized method would work best in each of the these two search spaces?

- Greedy descent with random steps best on 2
- Greedy descent with random restart best on 1

- But these examples are simplified extreme cases for illustration
  - in practice, you don’t know what your search space looks like

- Usually integrating both kinds of randomization works best
Greedy descent vs. Random sampling

- **Greedy descent** is
  - good for finding local minima
  - bad for exploring new parts of the search space

- **Random sampling** is
  - good for exploring new parts of the search space
  - bad for finding local minima

A mix of the two, plus some additional random choices, can work very well
General Local Search Algorithm

**Procedure** Local-Search\( (V, \text{dom}, C) \)

**Inputs**
- \( V \): a set of variables
- \( \text{dom} \): a function such that \( \text{dom}(X) \) is the domain of variable \( X \)
- \( C \): set of constraints to be satisfied

**Output**
- Complete assignment that satisfies the constraints

**Local**
- \( A[V] \): an array of values indexed by \( V \)

repeat

for each variable \( X \) do
- \( A[X] \leftarrow \) a random value in \( \text{dom}(X) \)

while (stopping criterion not met & \( A \) is not a satisfying assignment)
- Select a variable \( Y \) and a value \( V \in \text{dom}(Y) \)
- Set \( A[Y] \leftarrow V \)

if (\( A \) is a satisfying assignment) then
- return \( A \)

until termination

Sometime select the best neighbor during the local search

Sometime select the neighbor at random (random step)
General Local Search Algorithm

1: Procedure Local-Search(V, dom, C)
2: Inputs
3: V: a set of variables
4: dom: a function such that dom(X) is the domain of variable X
5: C: set of constraints to be satisfied
6: Output
7: complete assignment that satisfies the constraints
8: Local
9: A[V] an array of values indexed by V
10: repeat
11: for each variable X do
12: A[X] ← a random value in dom(X);
13: while (stopping criterion not met & A is not a satisfying assignment)
14: Select a variable Y and a value V ∈ dom(Y)
15: Set A[Y] ← V
16: if (A is a satisfying assignment) then
17: return A
18: until termination

Sometime do a random restart
**Stochastic Local Search for CSPs: details**

Examples of ways to add randomness to local search for a CSP

**One stage selection** of variable and value:
- Sometime chose the **best neighbour**
- Sometime choose a **random variable-value pair**

**Two stage selection** (first select variable V, then new value for V):
- Selecting **variables**:
  - Sometimes choose the variable which participates in the **largest number of conflicts**
  - Sometimes choose a **random variable** that participates in **some conflict**
  - Sometimes choose a **random variable**
- Selecting **values**
  - Sometimes choose the **best value** for the chosen variable
  - Sometimes choose a **random value** for the chosen variable
Different ways of selecting neighbors

• **One stage selection**: consider all assignments that differ in exactly one variable.

How many of those are there for N variables and domain size d?

A. $O(Nd)$
B. $O(d^N)$
C. $O(N^d)$
D. $O(N+d)$
Different ways of selecting neighbors

- **One stage selection**: consider all assignments that differ in exactly one variable. How many of those are there for $N$ variables and domain size $d$?

  A. $O(Nd)$

- **Two stage selection**: first choose a variable (e.g. the one involved in the most conflicts), then best value

  ✓ Lower computational complexity: $N+d$ checks. But less progress per step
Learning Goals for Local Search So Far

- Implement local search for a CSP.
  - Implement different ways to generate neighbors
  - Implement scoring functions to solve a CSP by local search through either greedy descent or hill-climbing.