2nd. Part

• Modeling
  – Primality/Duality
  – Global Constraints
• Constraint programming
  – examples in CHOCO
• Soft Constraints
  – Models
  – Algorithms

Modeling

• Any CSP can be formulated in different (equivalent) ways
• The efficiency of the solving algorithms can vary dramatically
• No strong results are known
• Active line of research
• Alternative formulations:
  – Primal/Dual
  – Primitive/Global constraints
Primal/Dual

Primal CSP: \((X, D, C)\)
- \(X = \{x_1, x_2, \ldots, x_n\}\), \(D = \{d_1, d_2, \ldots, d_k\}\), \(C = \{c_1, c_2, \ldots, c_r\}\)

\(c \in C\), \(\text{var}(c) = \{x_i, x_j, \ldots, x_k\}\) scope
\(\text{rel}(c) \subseteq d_i \times d_j \times \ldots \times d_k\) permitted tuples

Dual CSP: \((X', D', C')\)
- \(X' = \{x'_1, x'_2, \ldots, x'_r\}\), \(D' = \{d'_1, d'_2, \ldots, d'_r\}\), where \(d'_i = \text{rel}(c_i)\)
- \(C' = \{c'_{ij}\}\), binary constraints
\(\text{var}(c'_{ij}) = \{x_i, x_j\}\)
\(\exists c'_{ij} \in C' \iff \text{rel}(c_i) \cap \text{rel}(c_j) \neq \emptyset\)
\(\text{rel}(c'_{ij}) = \) consistent pairs of tuples
Example: Crossword puzzles

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>16</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

- **a**
- **aardvark**
- **aback**
- **abacus**
- **abaft**
- **abalone**
- **abandon**
- **Mona Lisa**
- **monarch**
- **monarchy**
- **monarda**
- **zymurgy**
- **zyrian**
- **zythum**

Primal model (Non-binary)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>16</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

- **variables:**
  - one for each unknown letter (cell)
- **domains:**
  - ‘a’, ‘z’
- **constraints:**
  - contiguous letters must form words in dictionary
### Dual model (binary)

- **Variables**: one for each unknown word across and down
- **Domains**: words from dictionary
- **Constraints**: intersecting words must agree on common letter

![Binary Model](image)

### Global Constraints

- $c$ is global iff:
  - $\text{arity}(c) = r > 2$
  - $c$ is logically equivalent to $(c_1, c_2, \ldots, c_k)$ binary
  - $\text{AC}(c)$ prunes more than $\text{AC}(c_1, c_2, \ldots, c_k)$

**Propagation**:
- There is a specialized efficient algorithm (exploits the semantics)

**Catalog**:
- set of global constraints
- best known algorithms for propagation
Example: all-different

\[ x \{1, 2\} \neq \neq \neq \neq y \{1, 2\} \]

3 binary constraints, they are AC, no pruning

Example: all-different

\[ x \{1, 2\} \neq \neq \neq y \{1, 2\} \]

3 binary constraints, they are AC, no pruning

\[ z \{1, 2\} \neq \neq \neq \{1, 2\} \]

1 ternary constraint, it is not AC, AC pruning \(\rightarrow\) empty domain no solution!!
Example: all-different

• Enforcing arc-consistency:
  – \( n \) variables, \( d \) values
  – \( n(n-1)/2 \) binary constraints: \( O(n^2 d^2) \)
  – 1 \( n \)-ary constraint:
    • general purpose algorithm \( O(d^n) \)
    • specialized algorithm \( O(n^2 d^2) \)

Constraint Programming

Declarative Programming: you declare

• Variables
• Domains
• Constraints
  and ask the SOLVER to find a solution!!

SOLVER offers:

• Implementation for variables / domains / constraints
• Hybrid algorithm: backtracking + incomplete inference
• Global constraints + optimized AC propagation
• Empty domain detection
• Embedded heuristics


## Constraint Logic Programming

- **Logic Programming:**
  - implements chronological backtracking
- **Constraint logic programming:**
  - extension including constraint satisfaction facilities
- **Existing solvers:**
  - Chip (www.cosytec.com)
  - Eclipse (www-icparc.doc.ic.ac.uk/eclipse)
  - Sicstus Prolog (www.sics.se/sicstus)
  - ...

## Imperative Constraint Programming

Library to be included in your (procedural) program

Provides:
- Special objects:
  - Variables / Domains / Constraints (global)
- Special functions to find:
  - One solution / the next solution

- **Existing Solvers:**
  - Ilog Solver (www.ilog.com)
  - Choco (www.choco-constraints.net)
**CHOCO**

- Library for modeling and solving combinatorial problems
- Intended for academic purposes
- Plus:
  - Free software (GPL from FSF)
  - Simple
  - Efficient
  - Generic
- Minus:
  - Implemented in Claire (which is implemented in C++)
  - Not (completely) stable

**Choco: 1st example**

```c
[sillyCSP() : void
-> let pb := choco/makeProblem("Silly CSP",3),
    x := choco/makeIntVar(pb, "x", 1, 3),
    y := choco/makeIntVar(pb, "y", 1, 3),
    z := choco/makeIntVar(pb, "z", 1, 3) in
  (choco/post(pb, x + y == z),
   choco/post(pb, x > y),
   choco/solve(pb,false),
   printf("~S~S~S\n",x,y,z))]
```
Choco: 2nd example

\[
\text{queens}(\text{n:integer, all:boolean})
\rightarrow \text{let pb := choco/makeProblem(" n queens", n),}
\quad \text{queens := list(choco/makeIntVar(pb, "Q" /+ string(i), 1, n) | i in (1 .. n) ) in}
\quad (\text{for i in (1 .. n)}
\text{ \quad for j in (i + 1 .. n)}
\text{ \quad let k := j - i in}
\text{ \quad \quad ( choco/post(pb, queens[i] !== queens[j]),}
\text{ \quad \quad choco/post(pb, queens[i] !== queens[j] + k),}
\text{ \quad \quad choco/post(pb, queens[i] !== queens[i] + k) ),}
\text{choco/solve(pb, all)) ]}
\]

Soft Constraints

- Motivation
- Models:
  - Fuzzy CSP
  - Weighted CSP
  - Valued CSP
- Algorithms:
  - Search
  - Dynamic programming
  - Approximate algorithms
Motivation

• Using the classical CSP framework:
  – Many problems have many solutions
    • Algorithms either give the first one they find or all of them
    • Typically, the user likes some solutions more than others
  – Many problems do not have any solution
    • Algorithms just report failure
    • Typically, the user can identify some non critical constraint

Soft CSP

• Problems:
  – Variables and domains as in classical CSP
  – Mandatory constraints (hard)
  – Preference constraints (soft)
• Feasible solution:
  – Complete assignment which satisfies every hard constraint
• Optimal solution:
  – Preferred feasible solution, according to soft constraints
• Complexity:
  – Np-hard
  – Much harder than classical CSP
Soft Constraints Models

- Max-csp [Freuder and Wallace 92]
- Fuzzy CSP [Dubois et al 93]
- Lexicographic CSP [Fargier et al 93]
- Weighted CSP
- Probabilistic CSP [Fargier and Lang 93]
- Valued CSP [Schiex et al 95]
- Semiring-based CSP [Bistarelli et al 95]

Notación

- Variables: $i, j, k, ...$
- Domains: $D_i, D_j, ...$
- Values: $a, b, ...$
- (Binary) constraints: $c_{ij}$
- Tuples: $\tau$
- Projection: $\tau_{[i,j]}$
Classical CSP

- Expressable as classical logic
- **Constraints**: boolean functions
  - \( C_{ij}(a,b) = \text{true}/\text{false} \)

- **Task of interest**:

\[
\exists \tau \ \forall c_{ij} \ c_{ij}(\tau_{[i,j]})
\]

Fuzzy CSP

- Extension of classical CSP to **fuzzy logic**
  - **Conjunction**: t-norm (minimum)
  - **Disjunction**: t-conorm (maximum)
  - \( c_{ij}(a,b) \in [0,1] \)

- **Task**:

\[
\max_{\tau} \{ \min_{c_{ij}} \{ c_{ij}(\tau_{[i,j]}) \} \} \]
Weighted CSP

- Preferences are expressed as costs
  - Constraints: cost functions
    \[ c_{ij}(a, b) \in \{0, 1, \ldots, \infty\} \]
  - Task:
    \[ \min\left\{ \sum_{ij} c_{ij}(\tau_{i,j}) \right\} \]

Example

- Airlines flight scheduling:
  - Input:
    - Aircrafts, airports
    - Flights: (origin, destination, frequency)
    - Requirements:
      - No more than four legs per flight
      - 1 hour < transfer time < 5 hours
      - ...
  - Output:
    - Schedule: each flight is a sequence of scheduled legs
Example

- **Classical CSP:**
  - Consistent schedules

- **Fuzzy CSP:**
  - Schedules where every flight is reasonably good
    - Maximizes the quality of the worst flight

- **Weighted CSP:**
  - Schedules where, globally, flights are good
    - Maximizes the sum of qualities over flights
    - Some flights can be very inconvenient

Valued CSP (VCSP)

[Schiex et al 95]

- Axiomatic model aiming at maximal generality
- It includes all previous models
- **Valuation structure** \((E,\ast,> )\):
  - \( E \) is the set of valuations
    - Totally ordered by “>”, the maximum element is “\( \top \)”, the minimum element is “\( \bot \)”. 
  - \( \ast \) is the aggregation of valuations
    - Binary operation on \( E \), commutative and associative. 
    - \( \bot \) is the identity
    - \( \top \) is absorbing
    - \( \ast \) grows monotonically
Valued CSP

• (Soft) constraints:
  \[ c_{ij}(a, b) \in E \]

• Task:
  \[ \min_{\tau} \left\{ \sum_{i,j} c_{ij}(\tau_{[i,j]}) \right\} \]

Soft CSP Solving

• Preliminaries

• Exact algorithms:
  – Branch and Bound
  – Partial Forward Checking
  – Reversible Dacs
  – Russian Doll Search
  – Bucket Elimination

• Approximate algorithms:
  – Local search approaches
  – Interval approximation
### Solving Valued CSP

- **Classical CSP:**
  - First part of this tutorial
  - Forward checking, $k$-consistency, MAC, ...
- **Idempotent CSP:**
  - Algorithms and properties from classical CSP extend easily
- **Non idempotent CSP:**
  - Algorithms and properties from classical CSP do not extend so easily
  - Last part of this tutorial
  - For simplicity, we will consider Max-csp