Constraint Satisfaction Problems (I)

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University

Outline
- Constraint Satisfaction Problems (CSPs)
- Backtracking Search
- Local Search
- The Structure of Problems
- Summary
Constraint Satisfaction Problems

A state of a CSP is defined by an assignment of values to some or all of the variables, \( \{ X_i = v_i, X_j = v_j, \ldots \} \).
- consistent (legal): violating no constraint.
- complete: all variable assigned

A solution to a CSP is a complete and consistent assignment.
- Some CSPs also require a solution that maximizes an objective function.
Constraint Satisfaction Problems

- Example: Coloring Australia with three colors
  - variables?
  - domains?
  - constraints?

Constraint graph

- Nodes = variables
- Arcs = constraints
Constraint Satisfaction Problems

- Treating a problem as a CSP confers several important benefits:
  - standard pattern
    - The successor function and goal test can be written in a generic way that applies to all CSPs.
  - generic heuristics
  - simplification of the solution process

Constraint Satisfaction Problems

- A CSP can be given an incremental formulation as a standard search problem.
  - Initial state: the empty assignment
  - Successor function: a legal value can be assigned to any unassigned variable.
  - Goal test: the assignment is complete.
  - Path cost: a constant cost for every step.
Constraint Satisfaction Problems

- Every solution must be a complete assignment and therefore appears at depth $n$ if there are $n$ variables.
  $\Rightarrow$ DFS is a popular algorithm for CSPs.

- The path by which a solution is reached is irrelevant.
  $\Rightarrow$ Local search works well with the complete-state formulation. (To be discussed later)

Types of domains
- discrete and finite
  - e.g. 3-coloring, 8-queens, boolean CSPs (3SAT)
- discrete but infinite
  - e.g. job scheduling
  - A constraint language is required.
  - Special solution algorithms exist for linear constraints on integer variables, but none exists for non-linear constraints.
- continuous
  - e.g. linear programming
Applications of CSP

- Real-world problems
  - **Location**: building facilities and supplying every customer with the minimal total cost
  - **Job shop scheduling**: processing all jobs following the capacity and route constraints with the minimal makespan
  - **Car sequencing**: sequencing the cars on the assembly line so that no workstation capacity is exceeded
  - **Cutting stock**: using different cutting patterns to meet the demand of different-shaped pieces with the minimal cost


Applications of CSP

- Real-world problems
  - **Vehicle routing**: finding tours for the finite-capacity vehicles to visit each customer once with the minimal cost
  - **Time tabling**: generating a schedule to satisfy the constraints about a set of students, examinations, rooms, and available time periods

Applications of CSP

- Location problem

we obtain the formulation

\[
\text{minimize } \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_{i}y_{i} \]

subject to

\[
\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, \ldots, n, \\
y_{i} - x_{ij} \geq 0, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \\
x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \\
y_{i} \in \{0, 1\}, \quad i = 1, \ldots, m.
\]

where:
- \( m \): number of potential locations
- \( n \): number of customers

Applications of CSP

- **Cutting stock**

  \( m \): different shaped pieces
\( d_i \): demand of each shape
\( n \): different cutting patterns
\( q_{ij} \): each pattern yields \( q_{ij} \) units of shape \( i \)
\( x_j \): the number of times pattern \( j \) is used
\( p \): totally \( p \) cutting actions
\( c_j \): the cost of pattern \( j \)

minimize \( \sum_{j=1}^{n} c_j x_j \)
subject to
\[ \sum_{j=1}^{n} q_{ij} x_j \geq d_i, \quad i = 1, \ldots, m, \]
\[ \sum_{j=1}^{n} x_j = p, \]
\[ x_j \in \{0, 1, \ldots, p\}, \quad j = 1, \ldots, n. \]

- **Job shop scheduling**

Objective functions:
make span, total flow time, etc.

Constraints:
(1) capacity
(2) precedence

Decision variables:
start time of operations.


“Constraint Satisfaction Problems,” Artificial Intelligence, Spring, 2010
Applications of CSP

- Vehicle routing

Objective functions:
- # routes, total distance, etc.

Constraints:
- \( \sum_{j=0}^{s} x_{ij} = 1, \quad i = 1, \ldots, n, \)
- \( \sum_{i=0}^{s} x_{ij} = 1, \quad j = 1, \ldots, n, \)
- \( \sum_{j=1}^{s} x_{ij} - \sum_{i=1}^{s} x_{ji} = 0, \)

subtour elimination constraints,
vehicle capacity constraints.

Decision variables:
- \( x_{ij} = \begin{cases} 
1 & \text{if a vehicle travels directly from customer } i \text{ to customer } j, \\
0 & \text{otherwise.} \end{cases} \)

Applications of CSP

- Games
  - Battleship
  - Mastermind
  - Minisame
  - Word puzzle
  - Minesweeper
  - Sudoku
  - 8-queens

http://4c.ucc.ie/web/outreach/index.html
Job Opportunity

--------------------------------------------------------------------

The Centre for Industrial Management, Traffic and Infrastructure of the Katholieke Universiteit Leuven has a vacancy for a doctoral student.

--------------------------------------------------------------------

Robust railway timetable for the Belgian Railways.

For 2012, the Belgian Railway Company (Infrabel) will design a new railway timetable. Infrabel wants the Centre for Industrial Management (CIB) to help them design a new and more robust timetable, using operations research (linear programming, mixed integer programming, etc.)

Description

A few years ago, a CIB doctoral student developed a few timetabling principles and applied these to the whole Inter City (IC) network.

Simulations have showed that the new IC timetable could reduce the passenger waiting time with 40% and the number of missed connections from 9 to 3%. At this moment, Infrabel is trying to introduce these principles in the way they design the timetable.

This doctoral research should assist Infrabel in designing the new timetable for 2012 and making it more robust. During peak hours, many trains face small delays. However small, these delays could cause many passengers to miss their connection. Infrabel wants to know how to deal with these small delays, when designing the timetable.
Job Opportunity

Questions

- How to deal with the limited capacity of the Noord-Zuid-As in Brussels? The Noord-Zuid-As consists of only 6 tracks between the North and the South of Brussels with one intermediate station (Brussels-Central) and almost all trains in Belgium have to run through this Noord-Zuid-As.
- How to deal with limited capacity in different stations?
- How to take into account crew schedules and the availability of train material?

Functions

- A four year doctoral research position at K.U. Leuven;
- (80%) scientific research in the field of operations research applied to real-life railway scheduling problems, in close cooperation with Infrabel;
- (20%) educational tasks: seminars, workshops, thesis coaching, etc.

(*Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010)
Constraints

- Types
  - unary
    - e.g. SA ≠ green
  - binary
    - e.g. SA ≠ NSW
  - higher-order

How to represent the constraints between the variables?

Constraints

\[ \text{Alldiff } (F,T,U,W,R,O) \]

- \( O + O = R + 10 \cdot X_1 \)
- \( X_1 + W + W = U + 10 \cdot X_2 \)
- \( X_2 + T + T = O + 10 \cdot X_3 \)
- \( X_3 = F \)

auxiliary variables
**Constraints**

- Absolute vs. preference constraints
  - Many real-world CSPs include preference constraints indicating some solutions are preferred.
    - e.g. timetabling problem
  - CSPs with preferences can be solved using optimization search methods, either path-based or local.
    - Here we do not discuss this kind of problem further.

**Constraint Satisfaction Problems (II)**

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University
Backtracking Search for CSPs

- We have formulated the CSPs as search problems.
- Let’s apply the search algorithms that we learned to solve them.
  - e.g. BFS
    - Given \( n \) variables, each with \( x \) possible values.
    - The branching factor at level \( k \) is \((n-k+1)x\).
    - We generate a tree with \( n! \cdot x^n \) leaves.

  But, there are only \( x^n \) possible complete assignments!

Backtracking Search for CSPs

- Commutativity
  - A problem is commutative if the order of application of any given set of actions has no effect on the outcome.
  - CSPs are commutative.
  - Thus, all CSP search algorithms generate successors by considering possible assignments for only a single variable at each level.
    - With this restriction, the number of leaves becomes \( x^n \).
    - The ordering of variables could be important.
Backtracking Search for CSPs

- **Backtracking search**
  - It is a DFS, which
  - chooses values for one variable at a time, and
  - backtracks when a variable has no legal values.

- Because the representation of CSPs is standardized, there's no need of a domain-specific initial state, successor function, or goal test.
Backtracking Search for CSPs

Exercise

- 孫一美、許毅源、汪茜茜和陶復邦要拍一張合照，請為他們安排左右次序：
  - 汪茜茜想要站在陶復邦的左側
  - 孫一美不想要站在右二
  - 許毅源不想要站在最左側
  - 陶復邦不想要站在最右側
  - 孫一美不是跟許毅源站在一起，就是分站兩側
Improving the Backtracking Search for CSPs

- Plain backtracking is an uninformed search algorithm.
- In our previous classes, we have seen how to improve the performance of uninformed search by domain-specific heuristics.
- Here, we will show how to improve the backtracking search by general-purpose heuristics.

Questions

- Which variable should be assigned?
- Which value should be tried?
- What are the implications of the current assignment for other unassigned variables?
- How can the search avoid repeating the failure?
Variable & Value Ordering

function BACKTRACKING-Search(csp) returns a solution, or failure
   return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(Variables[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment according to Constraints[csp] then
         add { var = value } to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failure then return result
         remove { var = value } from assignment
      return failure

Variable & Value Ordering

- Which variable will you assign next?
Variable & Value Ordering

- The minimum remaining values (MRV) heuristic -
  
  "choosing the variable with the fewest legal values."

It also has been called "the most constrained variable" or "fail-first" heuristic.

---

Variable & Value Ordering

- Stone and Stone (1986) showed that with the MRV heuristic, they were able to solve 96-queens problem using only a personal computer.
- The standard backtracking method could not solve the 30-queens problem in a reasonable amount of time.
Variable & Value Ordering

- The MRV heuristic has no help in choosing the first region to color.
- The degree heuristic
  - It selects the variable that is involved in the largest number of constraints on other unassigned variables.
  - It is a good tie-breaker for the MRV heuristic.

Example: The MRV + The degree heuristic
Variable & Value Ordering

- Once a variable is selected, the algorithm must decide the order in which to examine its values.

- The least-constraining-value heuristic
  - It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.
  - Note that if all solutions are to be found, the ordering doesn’t matter. The same holds if there are no solutions.

Example: The least-constraining-value heuristic

[Diagram of constraint graph with variables and constraints, indicating a choice point for value selection.]
Variable & Value Ordering

- Kale (1990) developed a backtracking-based algorithm that can be used to solve 1000-queens problems by incorporating the least-constraining-value heuristic into Stone and Stone’s (1986) algorithm.


Variable & Value Ordering

- The MRV heuristic chooses the most constrained variable, whereas the least-constraining-value heuristic chooses the least constrained value.

Are they conflicting with each other?

- Every variable needs to be assigned, but not every value should be tried.
- Assigning the most constrained variable first cuts off unnecessary branches, but trying the most constrained value just wastes your time.
Constraint Satisfaction Problems (III)

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University

Constraint Propagation

- By looking at some of the constraints earlier in the search, we can drastically reduce the search space.
- In the following we will describe:
  - forward checking
  - arc consistency (2-consistency)
  - k-consistency
  - handling of special constraints
Constraint Propagation

- **Forward checking**
  - Whenever a variable $X$ is assigned, it checks each unassigned variable $Y$ and deletes all inconsistent values.
  - It is a good partner for the MRV heuristic.

"Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010
Constraint Propagation

- Forward checking: example 2


"Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010
Constraint Propagation

- **Forward checking**
  - Whenever a new variable is considered, checking the consistency is no longer necessary.
  - Forward checking does more work than the simple backtracking when each assignment is added.
  - It reduces the size of the search tree and reduces the overall amount of work.

Try: http://www-users.cs.york.ac.uk/~frisch/Demos/An/Queens.html

Constraint Propagation

- **Arc consistency**
  - Although forward checking detects many inconsistencies, it does not detect all of them.
**Constraint Propagation**

- **Arc consistency**
  - Given two variables $X$ and $Y$, the arc $X \rightarrow Y$ is consistent if,

  for every value $x$ of $X$, there is some value $y$ of $Y$ that is consistent with $x$.

  - We can make an arc consistent by deleting the values that causes it to be inconsistent.
    - This is helpful for early detection of an inconsistency that is not detected by pure forward checking.

---

**Constraint Propagation**

- **Arc consistency: example 1**

  Original

  $x$ \[ \{1, \ldots, 5\} \] $x < y - 2$ $y$ \[ \{1, \ldots, 5\} \]

  $(x, y)$ consistent

  $x$ \[ \{1, 2\} \] $x < y - 2$ $y$ \[ \{1, \ldots, 5\} \]

  $(x, y)$ & $(y, x)$ consistent

  $x$ \[ \{1, 2\} \] $x < y - 2$ $y$ \[ \{4, 5\} \]
Constraint Propagation

Arc consistency: example 2

WA  NT  Q  NSW  V  SA  T

1: Q₆ → Q₄
2: Q₅ → Q₆
3: Q₄ → Q₅
4: Q₅ → Q₃


Constraint Propagation

Arc consistency: example 3

WA  NT  Q  NSW  V  SA  T

1: Q₆ → Q₄
2: Q₅ → Q₆
3: Q₄ → Q₅
4: Q₅ → Q₃


"Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010
Constraint Propagation

- Arc consistency checking can be applied
  - either as a preprocessing step,
  - or as a propagation step.
- In either case, the process must be applied repeatedly until no more inconsistencies remain.
  - This is because whenever a value is deleted to remove an arc inconsistency, a new arc inconsistency could arise.

Whenever a value is deleted to remove an arc inconsistency, a new arc inconsistency could arise.
Constraint Propagation

function AC-3(csp) returns the CSP, possibly with reduced domains
inputs: csp, a binary CSP with variables \{X_1, X_2, \ldots, X_n\}
local variables: queue, a queue of arcs, initially all the arcs in csp
while queue is not empty do
(X_i, X_j) ← REMOVE-FIRST(queue)
if RM-INCONSISTENT-VALUES(X_i, X_j) then
for each X_k in NEIGHBORS(X_i) do
add (X_k, X_i) to queue

function RM-INCONSISTENT-VALUES(X_i, X_j) returns true if remove a value
removed ← false
for each x in DOMAIN[X_i] do
if no value y in DOMAIN[X_j] allows (x, y) to satisfy constraint(X_i, X_j)
then delete x from DOMAIN[X_i]; removed ← true
return removed

"Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010

Constraint Propagation

- Time complexity of AC-3
  - arcs (assuming binary CSP)
  - Each arc can be inserted \(O(d^2)\) times, where \(d\) is the maximal number of possible values.
  - Checking consistency can be done in \(O(n^2d^3)\).

⇒ Worst-case complexity

- Do we have to add all \(X_k\) with arc pointing to \(X_i\) into the queue whenever any value of \(X_i\) is deleted? ⇒ AC-4

"Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010
Constraint Propagation

- **AC-4** (Mohr and Henderson, 1986)

**Notations**

\[ N = \{i, j, \ldots \} \] is the set of nodes, with \(|N| = n, \)
\[ A = \{b, c, \ldots \} \] is the set of labels, with \(|A| = a, \)
\[ E = \{(i, j) \} |(i, j) \text{ is an edge in } N \times N, \text{ with } |E| = e, \]
\[ A_j = \{b \mid b \in A \text{ and } (i, b) \text{ is admissible} \}, \]
\[ R_1 \text{ is a unary relation, and } (i, b) \text{ is admissible if } R_1(i, b), \]
\[ R_2 \text{ is a binary relation, and } (i, b) - (j, c) \text{ is admissible if } R_2(i, b, j, c). \]
\[ S_{nc} = \{(i, b) \mid c \text{ at node } j \text{ supports } b \text{ at node } i \}; \]

---

**Constraint Propagation**

- **AC-4**

**Step 1. Construction of the data structures** \[ O(ea^2) \]

```
1 M := 0; S_{cn} := Empty_set;
2 for (i, j) \in E do // O(e)
3 for b \in A do // O(a)
4 begin
5 Total := 0;
6 for c \in A do // O(a)
7 if R(i, b, j, c) then
8 begin
9 Total := Total + 1;
10 Append(S_{nc}, (i, b));
11 end;
12 if Total = 0 then M[i, b] := 1; A_i := A_i \{b\};
13 else Counter[(i, j), b] := Total;
14 end;
15 initialize List with \{(i, b) \mid M[i, b] = 1\}; // O(na)
```

---

Constraint Propagation

To verify the arc consistency, each arc must be inspected at least once, which takes $O(d^2)$.

Thus, AC4 is optimal in terms of worst-case complexity.
Constraint Propagation

- **Forward checking** can only be applied to the constraints in which exactly one variable is unassigned.
- **Arc consistency** algorithms can only be applied to constraints in which exactly two variables are unassigned.

⇒ High arity constraints are undesirable and should be avoided.

---

Because CSPs include 3SAT as a special case, we do not expect a polynomial-time algorithm that can decide whether a CSP is consistent.

⇒ Arc consistency does not reveal every possible inconsistency.
Constraint Propagation

- A CSP is \( k \)-consistent if,

for any \( k-1 \) variables and for any consistent assignment to those variables, a consistent value exists for any \( k \)th variable.

- \( 1 \)-consistency (node consistency)
- \( 2 \)-consistency (arc consistency)
- \( 3 \)-consistency (path consistency)

Constraint Propagation

- A CSP is strongly \( k \)-consistent if it is \( j \)-consistent for all \( j \leq k \).

- If a constraint graph containing \( n \) nodes is strongly \( n \)-consistent, then a solution can be found without any search.

- However, the worst-case complexity of the algorithm for \( n \)-consistency in an \( n \)-node graph is exponential.
Constraint Propagation

- An ordered constraint graph is a constraint graph whose vertices have been ordered linearly.


Constraint Propagation

- The width at a vertex in an ordered constraint graph is the number of arcs from previous vertices.
- The width of an ordered constraint graph is the maximum of the width of its vertices.
- The width of a constraint graph is the minimum width of all the derived ordered constraint graphs.

Constraint Propagation

Theorem
- If a constraint graph is strongly $k$-consistent, and $k > w$, where $w$ is the width, then a search order exists that is backtrack free.

E. Freuder, "Backtrack-free and back-bounded search," In Search in Artificial Intelligence, eds. L. Kanal and V. Kumar, 343-369, 1988.

Constraint Propagation
- Running stronger consistency checks will
  - take more time, but
  - will have a greater effect in
    - reducing the branching factor and
    - detecting inconsistent partial assignments.

- In practice, determining the appropriate level of consistency checking is mostly an empirical science.
Constraint Propagation

- **Experimental results:** *Less is More*
  - Nadel (1988) empirically compared five algorithms that primarily differ in the degrees of arc consistency.
    - The algorithm applied forward checking only outperformed the algorithm applied arc consistency.
  - “Experiments by other researchers with a variety of problems also indicate that it is better to apply constraint propagation only in a limited form.” – V. Kumar (1992)


Constraint Propagation

- **Experimental results:** *Two rights may make a wrong*

- **Benchmark instances:**
  - Binary CSP represented by connected graphs
  - 50 variables, domain of 8 values
  - Density: the fraction of the possible constraints beyond the minimum (49) constraints
  - Tightness: the fraction of all possible pairs of values from the domains of two variables that are not allowed.

Experimental results: *Two rights may make a wrong*

**AC+FC:** arc consistency pre-processing followed by forward checking

**FC:** forward checking

Constraint Propagation

- Experimental results: *Two rights may make a wrong*
  - Sometimes the preprocessing step already takes more effort than just doing forward checking.

<table>
<thead>
<tr>
<th></th>
<th>AC</th>
<th>FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2</td>
<td>13.96</td>
<td>14.84</td>
</tr>
<tr>
<td></td>
<td>847</td>
<td>119,886</td>
</tr>
<tr>
<td></td>
<td>14.84</td>
<td>13.726</td>
</tr>
<tr>
<td></td>
<td>1.083</td>
<td>282.143</td>
</tr>
</tbody>
</table>

*Constraint Satisfaction Problems*, Artificial Intelligence, Spring, 2010


---

Constraint Propagation

- Experimental results: *Two rights may make a wrong*
  - They took some easy problems where AC-FC was inferior to FC and ran AC-FC and FC on them without MRV heuristic (lexical ordering).
  - The phenomenon of preprocessing making matters worse did indeed disappear.
    - Without the ordering, however, performance was much worse than either FC or AC-FC with the ordering.

"Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010

Constraint Propagation

Experimental results: *More is more*

- MAC: based on AC-4
- FC: forward checking


Adding some additional consistency in the form of AC preprocessing alone can decrease performance.

However, adding even more consistency processing in the form of AC preprocessing plus full AC maintenance, can help.

MAC outperformed FC throughout the density/tightness space, except on some very easy problems.
Tea Time

- 4-Queen search tree drawer
  - [http://www-users.cs.york.ac.uk/~frisch/Demos/Yang/4QueensDemo.html](http://www-users.cs.york.ac.uk/~frisch/Demos/Yang/4QueensDemo.html)
- Minesweeper
  - [http://consystlab.unl.edu/our_work/minesweeper.html](http://consystlab.unl.edu/our_work/minesweeper.html)
- Sudoku solver
  - [http://www.youtube.com/watch?v=JtTTHe93WNI](http://www.youtube.com/watch?v=JtTTHe93WNI)

Constraint Satisfaction Problems (IV)

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University
Constraint Formulation

Every higher-order, finite-domain constraint can be reduced to a set of binary constraints if enough auxiliary variables are introduced.

Example 1 Propositional satisfiability (SAT) problems can be formulated as CSPs. Consider a SAT problem with 6 propositions, $x_1, \ldots, x_6$, and 4 clauses, (1) $x_1 \lor x_3 \lor x_6$, (2) $\neg x_1 \lor \neg x_3 \lor x_4$, (3) $x_3 \lor \neg x_5 \lor x_6$ and (4) $x_2 \lor x_4 \lor \neg x_5$. In one CSP representation of this SAT problem, there is a variable for each proposition, $x_1, \ldots, x_6$, each variable has the domain of values \{0, 1\}, and there is a constraint for each clause, $C_1(x_1, x_3, x_6)$, $C_2(x_1, x_3, x_4)$, $C_3(x_3, x_5, x_6)$ and $C_4(x_2, x_4, x_5)$. Each constraint specifies the value combinations that will make its corresponding clause true. For example, $C_4(x_2, x_4, x_5)$, the constraint associated with the clause $x_2 \lor x_4 \lor \neg x_5$, allows all tuples over the variables $x_2$, $x_4$, and $x_5$ except the falsifying assignment $(0, 0, 1)$.

Constraint Formulation

## Dual transformation

**Example 2** In the dual transformation of the CSP given in Example 1, there are 4 dual variables, $c_1, \ldots, c_4$, one for each constraint in the original formulation as shown in Figure 1. For example, the dual variable $c_1$ corresponds to the non-binary constraint $C_1(x_1, x_3, x_6)$ and the domain of $c_1$ contains all possible tuples except $(0, 0, 0)$. As an example of a dual constraint, the constraint between $c_1$ ($C_1(x_1, x_3, x_6)$) and $c_2$ ($C_2(x_1, x_3, x_4)$) requires that the first and second arguments of the tuples assigned to $c_1$ and $c_2$ agree. Hence, $\{c_1 \leftarrow (0, 0, 1)\}$ is compatible with $\{c_2 \leftarrow (0, 0, 0)\}$, but $\{c_1 \leftarrow (0, 0, 1)\}$ is incompatible with $\{c_2 \leftarrow (1, 0, 0)\}$.

Example 3 In the hidden transformation of the CSP given in Example 1, there are 10 variables (6 ordinary variables and 4 dual variables), as shown in Figure 2. As an example of a hidden constraint, the constraint between \( C_1(x_1, x_3, x_6) \) and \( x_1 \) requires that the first argument of the tuple assigned to \( C_1 \) agrees with the value assigned to \( x_1 \). Hence, \( \{ x_1 \leftarrow (0, 0, 1) \} \) is compatible with \( \{ x_1 \leftarrow 0 \} \), and \( \{ x_1 \leftarrow (0, 0, 1) \} \) is incompatible with \( \{ x_1 \leftarrow 1 \} \).
Constraint Formulation

- The dual transformation can in fact be built from the hidden transformation
  - by composing the hidden constraints between the dual variables and the ordinary variables to obtain dual constraints, and then
  - discarding the hidden constraints and ordinary variables.

- Combinations of different algorithms and formulations have different performance.


Constraint Formulation

- Elimination of high-arity constraints
  - adding extra variables
  - incorporating redundant constraints
  - preferring a few variables with large domains to many variables with small domains
  - e.g. If we want to pick two distinct values from 1...100, we may have one formulation with $2^{100}$ possible solutions and another formulation with $100^2$ possible solutions.
**Constraint Formulation**

- In many problems, if solutions exist, there are classes of equivalent solutions.
  - Such symmetries may cause difficulties for a search algorithm.
  - If possible, include additional constraints which allow only one solution from each class.

**Handling of Special Constraints**

- **Alldiff constraint**
  - It requires that all the variables involved must have distinct values.
  - One simple form of consistency detection is:

    If there are $m$ variables involved in the constraint, and they have $n < m$ possible distinct values altogether, the constraint cannot be satisfied.
Handling of Special Constraints

A simple algorithm for checking **Alldiff**

1. **Remove** any variable that has a singleton domain, and delete that variable’s value from the domains of the remaining variables.
2. **Repeat** as long as there are singleton-domain variables.
3. If at any point any empty domain is produced or there are more variables that domain values left, an inconsistency has been detected.

**Atmost** constraint (resource constraint)

- Example: four tasks, 10 personnel
  \[ \text{atmost}(10, T_1, T_2, T_3, T_4) \]

- A simple detection is to check the sum of the minimum values of the current domains.
  - e.g. The constraint cannot be satisfied if the domain of each variable is \( \{3, 4, 5, 6\} \).
Handling of Special Constraints

- Atmost constraint (resource constraint)
  - We can also enforce consistency by deleting the maximum value of any domain if it is not consistent with the minimum values of the other domains.

  e.g. Given atmost(10, T_1, T_2, T_3, T_4), the values 5 and 6 can be deleted if the domain is {2, 3, 4, 5, 6}.

- Bound constraint
  - Example:
    
    (1) Flight271 ∈ [0, 165]
    (2) Flight272 ∈ [0, 385]
    (3) Flight271 + Flight272 = 420


We say a CSP is bounds-consistent if for every variable X, and for both the lower and upper bound values of X, there exists some value of Y that satisfied the constraint between X and Y, for every variable Y.
Intelligent Backtracking

**Chronological backtracking**

```python
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(Variables/csp), assignment, csp
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp]
            then
                add { var = value } to assignment
                result ← RECURSIVE-BACKTRACKING(assignment, csp)
                if result ≠ failure then return result
                remove { var = value } from assignment
        return failure
```

"Constraint Satisfaction Problems," Artificial Intelligence, Spring, 2010

Intelligent Backtracking

Conflicts set for SA is (Q, NSW, V).

Re-coloring T cannot resolve the problem with SA!
Intelligent Backtracking

- In general, the conflict set for variable $X$ $\text{conf}(X)$ is the set of assigned variables that are connected to $X$ by constraints.

- Conflict-directed backtracking
  - If every possible value for $X_i$ fails, backjump to the most recent $X_i$ in $\text{conf}(X_j)$, and set $\text{conf}(X_i) \leftarrow \text{conf}(X_i) \cup \text{conf}(X_j) \setminus \{X_i\}$.

Local Search for CSPs

- Local search is very effective in solving many CSPs.
  - They use a complete-state formulation.
  - The initial state assigns a value to every variable.
  - The successor function usually works by changing the value of one variable at a time.
Local Search for CSPs

- **Example: 8-queens**

  Neighborhood function 1
  ![Image 1]

  Neighborhood function 2
  ![Image 2]

**Min-conflicts heuristic**
- select the value that results in the minimum number of conflicts with other variables

  ```python
  function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
  max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
      if current is a solution for csp then return current
      var ← a randomly chosen, conflicted variable from VARIABLES[csp]
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
  return failure
  ```

*Artificial Intelligence: A Modern Approach, 2nd ed., Figure 5.8*
Local Search for CSPs

- **Min-conflicts heuristic**

Another advantage of local search is that it can be used in an online setting when the problem changes.

- e.g. The bad weather at an airport renders the predetermined schedule infeasible. We would like to repair the schedule with a minimum number of changes.

A backtracking with the new set of constraints usually requires much more time than local search and might find a solution with many changes.
Local Search for CSPs

- **A case study on solving SAT problem** (Wu and Wah 2000)

Original formulation:

\[
\min_{x \in \{0, 1\}^n} N(x) = \sum_{i=1}^{n} U_i(x)
\]

\(U_i(x) = 0\) if the \(i\)th clause is satisfied; otherwise, \(U_i(x) = 1\).

\(\lambda\) is a weight vector.

- **Idea:** Different clauses may have different degrees of difficulty to be satisfied. A constrained formulation of SAT with dynamically changing penalties on each clause is intuitively preferable.


---

Local Search for CSPs

- **A case study on solving SAT problem** (Wu and Wah 2000)

New formulation (Discrete Lagrangian formulation):

\[
\min_{x \in \{0, 1\}^n} L_d(x, \lambda) = N(x) + \sum_{i=1}^{n} \lambda_i U_i(x)
\]

\(U_i(x) = 0\) if the \(i\)th clause is satisfied; otherwise, \(U_i(x) = 1\).

\(\lambda\) is a weight vector.

Local Search for CSPs

DLM-2000-SAT (Wu and Wah 2000)

1. Reduce the original SAT instance;
2. Generate a random starting point using a fixed seed;
3. Initialize $L_2 = 0$;
4. while solution not found and time not used up do
5. Pick $x_j \notin \text{TabuList}$ that reduces $L_d$ the most;
6. Flip $x_j$;
7. if $\# \text{Flips} \geq w_s = 0$ then
8. Update the queue on historical points end if
9. Maintain TabuList;
10. if $\# \text{Flips} \geq \theta_1$ then
11. $\lambda_i \leftarrow \lambda_i + \delta_i$;
12. if $\# \text{Adj} \neq \emptyset \# \theta_2 = 0$ then
13. $\lambda_i \leftarrow \lambda_i - \delta_i$; end if;
14. end if
15. end while

Clause reduction:
- Assign values to variables in single-variable clauses.

TabuList:
- It records the variables that are flipped recently.
- They are prohibited from being flipped for a period.
- The purpose of TabuList and the record of historical points is to avoid search cycles and traps.

Local Search for CSPs

- **DLM-2000-SAT** (Wu and Wah 2000)
  - **Historical points:**
    - It records the points that have been visited recently.
    - A distance penalty is calculated from a candidate point to the historical points $x_i^*$.

$$\text{distance\_penalty} = \sum \min(\theta_1, |x - x_i^*|),$$

$$L_d(x, \lambda) = N(x) + \sum_{i=1}^{\delta_1} \lambda U_i(x) - \text{distance\_penalty}.$$  

- A parameter $\theta_1$ is used to make the penalty not be a dominant factor.

---

Local Search for CSPs

- **DLM-2000-SAT** (Wu and Wah 2000)
  - **Increment of $\lambda$**
    - When the number of flat moves exceeds a threshold ($\delta_1$), weights of all unmatched clauses increase.
  - **Decrement of $\lambda$**
    - When the number of increments of $\lambda$ exceeds a threshold ($\delta_2$), weights of all clauses decrease.
    - It keeps the weighted penalty be within a reasonable range so that the effect of distance penalty is maintained.
Local Search for CSPs

- **DLM-2000-SAT** (Wu and Wah 2000)
  - This algorithm is able to solve many hard SAT problems with very high success rate.
  - See

Local Search for CSPs

- **Give a try**
  - The satisfiability library
    - [http://www.satlib.org/](http://www.satlib.org/)
  - DLM2000
  - International SAT competition
    - [http://www.satcompetition.org/](http://www.satcompetition.org/)
The Structure of Problems

- An important observation from the constraint graph is that coloring of T is independent of coloring of other states.

⇒ independent subproblems

Independence can be found by looking for connected components of the constraint graph.

- Suppose a problem with n variables can be divided into n/c independent subproblems with c variables.

⇒ Time complexity is reduced from $O(d^n)$ to $O(d^c \cdot n/c)$. 
The Structure of Problems

- Completely independent subproblems are delicious but rare.
- The simplest case is when the constraint graph forms a tree: any two variables are connected by at most one path.
- Any tree-structured CSP can be solved in time linear to the number of variables.

![Diagram of a tree-structured CSP](image)

Algorithm:

1. Choose any variable as the root.
2. Order the variables so that every node's parent in the tree precedes it in the ordering.
3. Label the variables $X_1, \ldots, X_n$ in order. (Now every variable except the root has exactly one parent.)
4. For $j$ from $n$ down to 2, apply arc consistency to the arc $(\text{parent}(X_j), X_j)$.
5. For $j$ from 1 to $n$, assign any value of $X_j$ consistent with the value assigned for $\text{parent}(X_j)$.
The Structure of Problems

- Analysis of the algorithm:
  - In step (4), by applying the arc-consistency checking in the reverse order ensures that any deleted values cannot endanger the consistency of arcs that have been processed.
  - After step (4), the CSP is directionally arc-consistent, so the assignment in step (5) requires no backtracking.
  - Time complexity is $O(n^2)$.

We have an efficient algorithm for trees. How can we deal with more general constraint graphs?

1. Assign values to some variables so that the remaining variables form a tree.
2. Construct a tree decomposition.
**The Structure of Problems**

- **Method 1:** cutset conditioning

(1) Choose a subset $S$ from $\text{VARIABLES}[csp]$ such that the constraint graph becomes a tree after removing $S$. (S is called a cycle cutset.)

(2) For each possible assignment to the variables in $S$ that satisfies all constraints in $S$,
   - (2.1) Remove from the domains of the remaining variables any values that are inconsistent.
   - (2.2) If the remaining CSP has a solution, return it.
The Structure of Problems

- **Cutset conditioning**: the analysis

  - If the cycle cutset has size $c$, the total runtime is $O(d^c \cdot (n-c)d^2)$.
  - If the graph is nearly a tree, the savings over straight backtracking will be huge.
  - Finding the smallest cycle cutset is NP-hard, but several efficient approximation algorithms are known.

The Structure of Problems

- **Method 2**: tree decomposition

  ![Diagram of tree decomposition](image-url)
The Structure of Problems

- Tree decomposition: requirements
  - Every variable appears in at least one subproblem.
  - If two variables are connected by a constraint in the original problem, they must appear together along with the constraint in at least one of the subproblems.
  - If a variable appears in two subproblems, it must appear in every subproblem along the path connecting those two subproblems.

- Tree decomposition: the algorithm
  1. Solve each subproblem independently.
  2. If any subproblem has no solution, return failure.
  3. View each subproblem as a “mega-variable” whose domain is its solutions.
  4. Solve the constraints connecting the subproblems using the efficient algorithm for trees.
The Structure of Problems

- CSPs with constraint graphs of bounded tree size are solvable in polynomial time.
- Unfortunately, finding the decomposition with minimal tree size is NP-hard.
  - There are heuristic methods that work well in practice.

Summary

- Constraint satisfaction problems
  - Variables + constraints
  - Terms: domain, values, assignment, consistency
- Backtracking search
  - minimum remaining values heuristic
  - degree heuristic
  - least-constraining-value heuristic
  - forward chaining
  - arc consistency
  - conflict-directed backjumping
Summary

- Local search with the min-conflicts heuristic
- Structure of the CSP
  - Tree-structured problem can be solved in linear time.
  - Converting a graph to a tree
    - cutset conditioning
    - tree decomposition

References