DM63
HEURISTICS FOR
COMBINATORIAL OPTIMIZATION

Lecture 2

Heuristic Methods and Metaheuristics for Construction Search

Marco Chiarandini

Outline

1. Example Problems
   - The Constraint Satisfaction Problem
   - Problem Solving
   - The Single Machine Total Tardiness Problem

2. Heuristic Methods

3. Metaheuristics for Construction Heuristics
   - Complete Tree Search
   - Rollout/Pilot Method
   - Limited Discrepancy Search
   - Beam Search

Past Lecture

▶ Course Introduction
▶ Combinatorial Problems
   ▶ Graph Coloring
   ▶ Propositional Satisfiability
▶ Computational Complexity
▶ Solution Methods
▶ Construction Heuristics for the Traveling Salesman Problem
▶ Software Development

Systematic search is often better suited when ...
▶ proofs of insolubility or optimality are required;
▶ time constraints are not critical;
▶ problem-specific knowledge can be exploited.

Local search is often better suited when ...
▶ non linear constraints and non linear objective function;
▶ reasonably good solutions are required within a short time;
▶ problem-specific knowledge is rather limited.

Complementarity:
Local and systematic search can be fruitfully combined, e.g., by using local search for finding solutions whose optimality is proved using systematic search.
1. Example Problems
   The Constraint Satisfaction Problem
   Problem Solving
   The Single Machine Total Tardiness Problem

2. Heuristic Methods

3. Metaheuristics for Construction Heuristics
   Complete Tree Search
   Rollout/Pilot Method
   Limited Discrepancy Search
   Beam Search

Example Problems

They are chosen because conceptually concise, intended to illustrate the development, analysis and presentation of algorithms

Although real-world problems tend to have much more complex formulations these problems capture their essence

Preprocessing Rules

Polynomial simplifications

SAT (CNF formulae)
   ▶ elimination of duplicates
   ▶ elimination tautological clauses
   ▶ elimination subsumed clauses
   ▶ elimination clauses with pure literals
   ▶ elimination unit clauses and unit propagation

k-coloring
   ▶ Remove under-constrained nodes
   ▶ Remove subsumed nodes
   ▶ Merge nodes that must have the same color

Constraint Satisfaction Problem

▶ Given: a triple \((X, D, C)\), where \(X\) is a set of variables, \(D\) is a domain of values, and \(C\) is a set of constraints. Every constraint is a pair \((t, R)\), where \(t\) is a tuple of variables and \(R\) is a relation (or a set of tuples of values; all these tuples having the same number of elements)

   An evaluation of the variables is a function from variables to \(v : X \rightarrow D\). Such an evaluation satisfies a constraint \(((x_1, \ldots, x_n), R)\) if \((v(x_1), \ldots, v(x_n)) \in R\).

▶ Task: A solution is an evaluation that satisfies all constraints.
Problem Solving as by George Pólya

George Pólya’s 1945 book *How to Solve It*:

- First, you have to understand the problem.
- After understanding, then make a plan.
- Carry out the plan.
- Look back on your work. How could it be better?

Pólya’s First Principle: Understand the Problem

- Do you understand all the words used in stating the problem?
- What are you asked to find or show?
- Can you restate the problem in your own words?
- Can you think of a picture or a diagram that might help you understand the problem?
- Is there enough information to enable you to find a solution?

Pólya’s Second Principle: Devise a plan

There are many reasonable ways to solve problems. The skill at choosing an appropriate strategy is best learned by solving many problems. You will find choosing a strategy increasingly easy. A partial list of strategies is included:

- Guess and check
- Make an orderly list
- Eliminate possibilities
- Use symmetry
- Consider special cases
- Use direct reasoning

Also suggested:

- Look for a pattern
- Draw a picture
- Solve a simpler problem
- Use a model
- Work backward
- Be ingenious

Pólya’s third Principle: Carry out the plan

Need is care and patience, given that you have the necessary skills. Persist with the plan that you have chosen. If it continues not to work discard it and choose another. Don’t be misled, this is how mathematics is done, even by professionals.

Pólya’s fourth Principle: Review/Extend

Much can be gained by taking the time to reflect and look back at what you have done, what worked and what didn’t. Doing this will enable you to predict what strategy to use to solve future problems.
"If you can’t solve a problem, then there is an easier problem you can solve: find it."

### The Single Machine Total Tardiness Problem

**Given:** a set of \( n \) jobs \( \{ J_1, \ldots, J_n \} \) to be processed on a single machine and for each job \( J_i \), a processing time \( p_i \), a weight \( w_i \) and a due date \( d_i \).

**Task:** Find a schedule that minimizes the total weighted tardiness \( \sum_{i=1}^{n} w_i \cdot T_i \)

where \( T_i = (C_i - d_i, 0) \) \( (C_i \) completion time of job \( J_i)\)

**Example:**

<table>
<thead>
<tr>
<th>Job</th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
<th>( J_4 )</th>
<th>( J_5 )</th>
<th>( J_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Due date</td>
<td>6</td>
<td>13</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>Weight</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Sequence \( \phi = J_3, J_1, J_5, J_4, J_1, J_6 \)

<table>
<thead>
<tr>
<th>Job</th>
<th>( J_3 )</th>
<th>( J_1 )</th>
<th>( J_5 )</th>
<th>( J_4 )</th>
<th>( J_1 )</th>
<th>( J_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion time</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>12</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>( T_i )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( w_i \cdot T_i )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>15</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

### Outline

1. Example Problems
   - The Constraint Satisfaction Problem
   - Problem Solving
   - The Single Machine Total Tardiness Problem

2. Heuristic Methods

3. Metaheuristics for Construction Heuristics
   - Complete Tree Search
   - Rollout/Pilot Method
   - Limited Discrepancy Search
   - Beam Search

### Heuristic Paradigms

#### Construction Heuristics

They are closely related to search tree techniques but correspond to a single path from root to leaf:

- search space = partial candidate solutions
- search step = extension with one or more solution components

**Construction Heuristic (CH):**

\( s := \emptyset \)

While \( s \) is not a complete solution:

- choose a solution component \( c \)
- add the solution component to \( s \)
Construction Heuristics
An important class of Construction Heuristics are greedy algorithms.
▶ Strategy: always make the choice which is the best at the moment.
▶ They are not generally guaranteed to find globally optimal solutions (but sometimes they do: Minimum Spanning Tree, Single Source Shortest Path, etc.)

Heuristic Paradigms
Perturbative search
▶ search space = complete candidate solutions
▶ search step = modification of one or more solution components
▶ iteratively generate and evaluate candidate solutions
   ▶ decision problems: evaluation = test if solution
   ▶ optimization problems: evaluation = check objective function value
   ▶ evaluating candidate solutions is typically computationally much cheaper than finding (optimal) solutions

Iterative Improvement (II):
While \( s \) has better neighbors:
   choose a neighbor \( s' \) of \( s \)
   such that \( f(s') < f(s) \)
   \( s := s' \)

Example: Uninformed random walk for SAT

```plaintext
procedure URW-for-SAT(F, maxSteps)
input: propositional formula \( F \), integer maxSteps
output: model of \( F \) or \( \emptyset \)
choose assignment \( \varphi \) of truth values to all variables in \( F \)
   uniformly at random;
steps := 0;
while not((\( \varphi \) satisfies \( F \)) and (steps < maxSteps)) do
   select \( x \) uniformly at random from \{ \( x' \) | \( x' \) is a variable in \( F \) and
   changing value of \( x' \) in \( \varphi \) decreases the number of unsatisfied clauses\};
   steps := steps + 1;
end
if \( \varphi \) satisfies \( F \) then
   return \( \varphi \)
else
   return \( \emptyset \)
end
end URW-for-SAT
```

Heuristic Methods
Meta-heuristics are general purpose guidance criteria to enhance these two heuristic paradigms.

Meta-heuristics
General guidance criteria over lower level heuristics.

In order to obtain very high quality solutions it is often successful to combine these heuristic paradigms into an hybrid algorithm.
A note on terminology

Heuristic Methods ≡ Metaheuristics ≡ Local Search Methods ≡ Stochastic Local Search Methods ≡ Hybrid Metaheuristics

Method ≠ Algorithm

Stochastic Local Search (SLS) algorithms allude to:
- **Local Search**: informed search based on local or incomplete knowledge as opposed to systematic search
- **Stochastic**: use randomized choices in generating and modifying candidate solutions. They are introduced whenever it is unknown which deterministic rules are profitable for all the instances of interest.

Outline

1. Example Problems
   - The Constraint Satisfaction Problem
   - Problem Solving
   - The Single Machine Total Tardiness Problem

2. Heuristic Methods

3. Metaheuristics for Construction Heuristics
   - Complete Tree Search
   - Rollout/Pilot Method
   - Limited Discrepancy Search
   - Beam Search

Uninformed Complete Tree Search

- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening depth-first search
- Bidirectional Search

Informed Complete Tree Search

- Informed search algorithm: exploit problem-specific knowledge
- Best-first search: node that "appears" to be the best selected for expansion based on an evaluation function $f(x)$
- Implemented through a priority queue of nodes in ascending order of $f$
**Informed Complete Tree Search**

**Greedy best-first search**
- Evaluates nodes on the basis of an heuristic function $f(n) = h(n)$
- Incomplete version: perform only one descent
- Complete version: need to avoid repeated states

**A* best-first search**
- The priority assigned to a node $x$ is determined by the function
  \[
  f(x) = g(x) + h(x)
  \]
  - $g(x)$: cost of the path so far
  - $h(x)$: heuristic estimate of the minimal cost to reach the goal from $x$
- It is optimal if $h(x)$ is an
  - admissible heuristic: never overestimates the cost to reach the goal
  - consistent: $h(n) \leq c(n, a, n') + h(n')$
A* best-first search

Possible choices for admissible heuristic functions

▶ optimal solution to an easily solvable relaxed problem
▶ optimal solution to an easily solvable subproblem
▶ learning from experience by gathering statistics on state features
▶ preferred heuristics functions with higher values (provided they do not overestimate)
▶ if several heuristics available $h_1, h_2, \ldots, h_m$ and not clear which is the best then:

$$h(x) = \max\{h_1(x), \ldots, h_m(x)\}$$

---

Rollout/Pilot Method

Derived from A*

▶ Each candidate solution is a collection of $m$ components $s = (s_1, s_2, \ldots, s_m)$.
▶ Master process add components sequentially to a partial solution $S_k = (s_1, s_2, \ldots, s_k)$
▶ At the $k$-th iteration the master process evaluates seemingly feasible components to add based on a look-ahead strategy based on heuristic algorithms.
▶ The evaluation function $H(S_{k+1})$ is determined by sub-heuristics that complete the solution starting from $S_k$
▶ Sub-heuristics are combined in $H(S_{k+1})$ by
  ▶ weighted sum
  ▶ minimal value

---

Speed-ups:

▶ halt whenever cost of current partial solution exceeds current upper bound
▶ evaluate only a fraction of possible components

---

A* best-first search

Drawbacks

▶ Time complexity: In the worst case, the number of nodes expanded is exponential, but it is polynomial when the heuristic function $h$ meets the following condition:

$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

$h^*$ is the optimal heuristic, i.e. the exact cost to get from $x$ to the goal.

▶ Memory usage: In the worst case, it must remember an exponential number of nodes.
Several variants: including iterative deepening A* (IDA*), memory-bounded A* (MA*) and simplified memory bounded A* (SMA*) and recursive best-first search (RBFS)
**Limited Discrepancy Search**

- A discrepancy is a branch against the value of an heuristic
- Ex: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second
  best is chosen
- Explore the tree in order of an increasing number of discrepancies

**Beam Search**

Possible extension of tree based construction heuristics:
- maintains a set $B$ of $bw$ (beam width) partial candidate solutions
- at each iteration extend each solution from $B$ in $fw$ (filter width)
  possible ways
- rank each $bw \times fw$ candidate solutions and take the best $bw$
  partial solutions
- complete candidate solutions obtained by $B$ are maintained in $B_f$
- Stop when no partial solution in $B$ is to be extend