Chapter 5
Escaping local optima

Companion slides of
How to Solve It: Modern Heuristics
by Zbigniew Michalewicz and David B. Fogel
(Springer-Verlag, 2nd edition, 2004)
prepared by
José Fernando Oliveira
Maria Antónia Carravilla

Traditional problem-solving strategies

• Either they guarantee discovering the global solution, but are too expensive (i.e., too time consuming) for solving typical real-world problems,
• Or they have a tendency of "getting stuck" in local optima.
• Since there is almost no chance to speed up algorithms that guarantee finding the global solution

design algorithms that are capable of escaping local optima
Modified local search procedures

Basic local search procedure
(one starting point → one run)

procedure local search
begin
    $x$ = some initial starting point in $S$
    while improve($x$) ≠ 'no' do
        $x$ = improve($x$)
        return($x$)
    end
end

The subprocedure improve($x$) returns a new point $y$ from the neighborhood of $x$, i.e., $y \in N(x)$, if $y$ is better than $x$, otherwise it returns a string "no," and in that case, $x$ is a local optimum in $S$.

Simulated annealing procedure

procedure simulated annealing
begin
    $x$ = some initial starting point in $S$
    while not termination-condition do
        $x$ = improve?($x$, $T$)
        update($T$)
        return($x$)
    end
end

Executed until some external "termination condition" is satisfied as opposed to the requirement of local search to find an improvement.

The function "improve? ($x$, $T$)" doesn't have to return a better point from the neighborhood of $x$. It just returns an accepted solution $y$ from the neighborhood of $x$, where the acceptance is based on the current temperature $T$.

The parameter $T$ is updated periodically, and the value of this parameter influences the outcome of the procedure "improve?"

Tabu search procedure

procedure tabu search
begin
    $x$ = some initial starting point in $S$
    while not termination-condition do
        $x$ = improve?($x$, $H$)
        update($H$)
        return($x$)
    end
end

As in simulated annealing, the function "improve? ($x$, $H$)" returns an accepted solution $y$ from the neighborhood of $x$, which need not be better than $x$, but the acceptance is based on the history of the search $H$. 
SIMULATED ANNEALING

Procedure iterated hill-climber

begin
  $t \leftarrow 0$
  initialize $best$
  repeat
    $local \leftarrow FALSE$
    select a current point $v_c$ at random
    evaluate $v_c$
    repeat
      select all new points in the neighborhood of $v_c$
      select the point $v_n$ from the set of new points
      with the best value of evaluation function $eval$
      if $eval(v_n)$ is better than $eval(v_c)$ then
        $v_c \leftarrow v_n$
      else $local \leftarrow TRUE$
    until $local$
    $t \leftarrow t + 1$
    if $v_c$ is better than $best$ then
      $best \leftarrow v_c$
  until $t = MAX$
end

Maximization

Restarting is the only way to escape from a local optimum

Always returns a local optimum
Stochastic hill-climber

- Instead of checking all of the solutions in the neighborhood of a current point \( v_c \) and selecting the best one, select only one point, \( v_n \), from this neighborhood.
- Accept this new point, i.e., \( v_c \leftarrow v_c \), with some probability that depends on the relative merit of these two points, i.e., the difference between the values returned by the evaluation function for these two points.

Stochastic hill-climber

\[
\text{procedure } \text{stochastic hill-climber} \\
\begin{align*}
  t &\leftarrow 0 \\
  \text{select a current solution } v_c \text{ at random} \\
  \text{evaluate } v_c \\
  \text{repeat} \\
  &\quad \text{select the solution } v_n \text{ from the neighborhood of } v_c \\
  &\quad \text{accept } v_n \text{ with probability } \\
  &\quad \quad \frac{e^{\text{eval}(v_c) - \text{eval}(v_n)}}{1 + e^{\text{eval}(v_c) - \text{eval}(v_n)}} \\
  &\quad t \leftarrow t+1 \\
  \text{until } t = \text{MAX} \\
\end{align*}
\]

- The stochastic hill-climber has only one loop.
  - We don't repeat its iterations starting from different random points.
- The newly selected point is "accepted" with some probability \( p \).
  - This means that the rule of moving from the current point \( v_c \) to the new neighbor, \( v_n \), is probabilistic.
- It is possible for the new accepted point to be worse than the current point.
- The probability of acceptance depends on the difference in merit between these two competitors, i.e., \( \text{eval}(v_c) - \text{eval}(v_n) \), and on the value of an additional parameter \( T \).
- \( T \) remains constant during the execution of the algorithm.
How much better/worse is the new neighbor?

\[ p = \frac{1}{1 + e^{\frac{eval(v_c) - eval(v_n)}{T}}} \]

- \( eval(v_c) - eval(v_n) \to -\infty \)
  - \( v_n \) is infinitely better than \( v_c \) \( \to \) acceptance of \( v_n \) almost sure \((p \to 1)\)

- \( eval(v_c) - eval(v_n) \to +\infty \)
  - \( v_n \) is infinitely worse than \( v_c \) \( \to \) acceptance of \( v_n \) almost impossible \((p \to 0)\)

Maximization

What is the role of the parameter \( T \)?

\[ p = \frac{1}{1 + e^{\frac{eval(v_c) - eval(v_n)}{T}}} \]

- \( T \) big \( \to \) exponent of \( e \) small \( \to p \) close to \( 1 \)
  independently of \( eval(v_c) - eval(v_n) \)

- \( T \) small \( \to \) exponent of \( e \) big \( \to p \) close to \( 0 \)
  independently of \( eval(v_c) - eval(v_n) \)
Simulated annealing

• The parameter $T$ changes during the run.
  – It starts with high values of $T$
    • procedure similar to a purely random search
  – gradually decreases the value of $T$
  – towards the end of the run, the values of $T$ are quite small
    • the final stages of simulated annealing merely resemble an ordinary hill-climber.

• Always accepts new points if they are better than the current point.

Procedure simulated annealing

begin
  $t \leftarrow 0$
  initialize $T$
  select a current point $v_c$ at random
  evaluate $v_c$
  repeat
    repeat
      select a new point $v_n$ in the neighborhood of $v_c$
      if $\text{eval}(v_c) < \text{eval}(v_n)$ then
        $v_c \leftarrow v_n$
      else if random$[0,1) < e^{-\frac{\text{eval}(v_c) - \text{eval}(v_n)}{T}}$ then
        $v_c \leftarrow v_n$
    until (termination-condition)
  $T \leftarrow g(T, t)$
  $t \leftarrow t+1$
  until (halting-criterion)
end
An analogy taken from thermodynamics/metallurgy

- To grow a crystal, you start by heating a row of materials to a molten state.
- You then reduce the temperature of this *crystal melt* until the crystal structure is *frozen in*.
- Bad things happen if the cooling is done too quickly.
  - Some irregularities are locked into the crystal structure and the trapped energy level is much higher than in a perfectly structured crystal.
- A similar problem occurs in metallurgy when heating and cooling metals.

Analogies between a physical system and an optimization problem

<table>
<thead>
<tr>
<th>Physical System</th>
<th>Optimization Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>feasible solution</td>
</tr>
<tr>
<td>energy</td>
<td>evaluation function</td>
</tr>
<tr>
<td>ground state</td>
<td>optimal solution</td>
</tr>
<tr>
<td>temperature</td>
<td>control parameter $T$</td>
</tr>
</tbody>
</table>
Problem specific and generic decisions

- Problem specific decisions
  - What is a solution?
  - What are the neighbors of a solution?
  - What is the cost of a solution?
  - How do we determine the initial solution?
- Generic (simulated annealing) decisions
  - How do we determine the initial "temperature" $T$?
    - Probability of 50% of a worst case for $eval(v_s) - eval(v_n)$ being accepted.
  - How do we determine the cooling ratio $g(T, t)$?
    - Geometric cooling schedule: $T_{t+1} = \alpha \times T_t$ ($0 < \alpha < 1$)
  - How do we determine the termination condition?
    - Fixed ("long") number of iterations
  - How do we determine the halting criterion?
    - Number of iterations without change in $v_c$

 Typical simulated annealing behavior
Simulated annealing behavior depending on the cooling ratio $\alpha$

![Graph showing simulated annealing behavior with cooling ratios $\alpha = 0.95$ and $\alpha = 0.99$.]
Tabu search

• A "memory" forces the search to explore new areas of the search space.
• We can memorize some solutions that have been examined recently and these become tabu (forbidden) points to be avoided in making decisions about selecting the next solution.
• Note that tabu search is basically deterministic (as opposed to simulated annealing), but it's possible to add some probabilistic elements to it.

Tabu search applied to SAT

• SAT problem with $n = 8$ variables.
• Initial assignment: $x = (0,1,1,1,0,0,0,1)$.
• Evaluation function: weighted sum of a number of satisfied clauses, where the weights depend on the number of variables in the clause.
  – In this case, the evaluation function should be maximized (i.e., we're trying to satisfy all the clauses).
  – Let us assume that the initial assignment has the value 27.
Tabu search applied to SAT

• Neighborhood of \( x \): consists of eight other solutions, each of which can be obtained by flipping a single bit in the vector \( x \).

• We evaluate them and select the best one.
  – Suppose that flipping the third variable generates the best evaluation (say, 31), so this new vector yields the current best solution.

Keep a record of our moves: memory structures for bookkeeping

• We will remember the index of a variable that was flipped, as well as the “time” when this flip was made, so that we can differentiate between older and more recent flips.

• A vector \( M \) will serve as our memory:
  – Initialized to 0.
  – At any stage of the search, the entry \( M(i) = j \) (when \( j \neq 0 \)) might be interpreted as “\( j \) is the most recent iteration when the \( i \)-th bit was flipped”.
  – \( j = 0 \) implies that the \( i \)-th bit has never been flipped.
Memory structures for bookkeeping

- In iteration $t$ we check if the chosen bit $i$ was or not flipped in the last $k$ (let us say, 5) iterations:
  \[ M(i) \geq t - 5 \Rightarrow \text{tabu status} \]

Alternative memory structure

- After a number of iterations the information stored in memory is erased.
- Assuming that any piece of information can stay in a memory for at most, say, five iterations, an alternative interpretation of an entry $M(i) = j$ (when $j \neq 0$) could be: “the $i$-th bit was flipped $5-j$ iterations ago.”
- Contents of memory after iteration 1:
  
  \[
  \begin{array}{cccccccc}
  0 & 0 & 5 & 0 & 0 & 0 & 0 & 0
  \end{array}
  \]
  
  For the next five iterations, the third bit position is not available (i.e., is tabu).
Tabu list

• After four additional iterations selecting the best neighbor and making an appropriate flip:

```
 3 0 1 5 0 4 2 0
```

– Bits 2, 5, and 8 may be flipped any time,
– bit 1 is not available for the next three iterations,
– bit 3 isn’t available but only for the next iteration,
– bit 4 (which was just flipped) is not available for the next five iterations...

Tabu list

• Tabu search utilizes the memory to force the search to explore new areas of the search space.
• The memorized flips that have been made recently are tabu (forbidden) for selecting the next solution.
• Thus at the next iteration (iteration 6), it is impossible to flip bits 1, 3, 4, 6, and 7, since these bits were tried “recently”.
• These forbidden (tabu) solutions (i.e., the solutions that are obtained from the forbidden flips) are not considered, so the next solution is selected from a set of $X_2$, $X_5$, and $X_8$. 
Aspiration criterion

• At any stage, there is a current solution being processed which implies a neighborhood, and from this neighborhood, tabu solutions are eliminated from possible exploration.
  – Too restrictive policy.
  – It might happen that one of the tabu neighbors provides an excellent evaluation score, eventually much better than the score of any solution considered previously.

Aspiration criterion

• In order to make the search more flexible, tabu search:
  – considers solutions from the whole neighborhood,
  – evaluates them all, and selects a non-tabu solution as the next current solution, whether or not this non-tabu solution has a better evaluation score than the current solution.
  – however, if an outstanding tabu solution is found in the neighborhood, such a superior solution is taken as the next point.
  – This override of the tabu classification occurs when a so-called aspiration criterion is met.
Other memory structures

• The memory structure discussed so far can be labeled as *recency-based* or *recent-term* memory, as it only records some actions of the last few iterations.

• This structure might be extended by a so-called *frequency-based* or *long-term* memory, which operates over a much longer horizon.

Frequency-based memory applied to SAT

• Vector $H \rightarrow$ long-term memory.
  – Initialized to 0
  – At any stage of the search the entry $H(i) = j$ is interpreted as “during the last $h$ iterations of the algorithm, the i-th bit was flipped $j$ times”.
  – Usually, the value of the horizon $h$ is quite large, at least in comparison with the horizon of recent-term memory.

• Thus after, say 100 iterations with $h = 50$, the long-term memory $H$ might have the following values:

```
5 7 11 3 9 8 1 6
```

• These frequency counts show the distribution of moves throughout the last 50 iterations.
Search diversification

- The principles of tabu search indicate that this type of memory might be useful to **diversify** the search.
  - For example, the frequency-based memory provides information concerning which flips have been under-represented (i.e., less frequent) or not represented at all, and we can diversify the search by exploring these possibilities.
  - Usually the value of the evaluation score is decreased by some penalty measure that depends on the frequency, and the final score implies the winner.  
    \[ \text{eval}(x') - \text{penalty}(x') \]

A general structure for a short-term memory tabu search algorithm

```plaintext
begin
  tries ← 0
  repeat
    generate an initial solution \( v_c \)
    count ← 0
    repeat
      generate a (sub)neighborhood of \( v_c \)
      for all solution \( v_n \) in the (sub)neighborhood of \( v_c \)
        if \( v_n \) is the local best-so-far (for a given 'tries') then
          if modification\((v_c,v_n)\) is not tabu or
            (modification\((v_c,v_n)\) is tabu and \( v_n \) satisfies aspiration criterion) then
              update local best solution information
              \( v_c \) ← 'best' local solution
              update tabu list and other variables
            if \( v_n \) is the global best-so-far (for all 'tries') then
              update global best solution information

        count ← count + 1
    until count = ITER
  tries ← tries + 1
until tries = MAX-TRIES
end
```
Problem specific and generic decisions

• Problem specific decisions
  – What is a solution?
  – What are the neighbors of a solution?
  – What is the cost of a solution?
  – How do we determine the initial solution?

• Generic (tabu search) decisions
  – Sub-neighborhood size?
  – How to sample the neighborhood?
  – Tabu status to solutions or modifications?
    • Which modification attributes record in the tabu list?
      – Memory space needed to save the tabu attributes.
      – Tabu list insertion and removal operations efficiency.
      – Fast verification if an attribute is in the tabu list.
  – Tabu list size
    • Too small: frequent cycles and eventually premature convergence of the algorithm.
    • Too large: worst final solution because too many solutions may not be reachable.
    • Dynamic (variable size) tabu lists are an option.
  – Long-term memory list size?
  – When to diversify the search?
  – How do we stop the algorithm?

MORE FORMAL
ALGORITHM DESCRIPTIONS
**Local Search**

\[ x_n = \text{generate\_initial\_solution}(); \]
\[ x^* = x_n; \]
\[ F^* = F(x_n); \]

repeat
   \[ V(x_n) = \text{neighbourhood}(x_n); \]
   \[ x_{n+1} = \text{best\_solution}(V(x_n)); \]
   if \( F(x_{n+1}) \leq F(x_n) \) then
      \[ x_n = x_{n+1}; \]
      \[ x^* = x_{n+1}; \]
      \[ F^* = F(x_{n+1}); \]

while \( F(x_{n+1}) \leq F(x_n); \)

**Simulated Annealing**

\[ x_n = \text{generate\_initial\_solution}(); \]
\[ x^* = x_n; \quad F^* = F(x_n); \]
\[ T_n = \text{initial\_temperature}(); \]

repeat
   for \( k = 1 \) to \( L_n \)
      \[ x = \text{random\_neighbour}(x_n); \]
      if \( F(x) \leq F(x_n) \) then
         \[ x_n = x; \]
      if \( F(x) < F^* \) then
         \[ x^* = x; \quad F^* = F(x); \]
      else
         if \( p \in [0, 1[ \leq p(n) \) then
            \[ x_n = x; \]
      actualize\!(L_n, T_n); \]
   while stop\_criterion == false;

\[ T_{t+1} = \alpha \times T_t \quad (0 < \alpha < 1) \]
Tabu Search

\[ x_n = \text{generate\_initial\_solution}(); \]
\[ x^* = x_n; \quad F^* = F(x_n); \]
\textbf{repeat}
\[ V'(x_n) = \text{sub-neighbourhood}(x_n); \]
\[ \overline{F} = \infty; \]
\textbf{for all} \( x' \in V'(x_n) \)
\[ \textbf{if} \ F(x') < \overline{F} \textbf{ then} \]
\[ m' = \text{modification}(x_n, x'); \]
\[ \textbf{if} \ m' \notin \text{tabu\_list} \textbf{ or } (m' \in \text{tabu\_list} \textbf{ and } F(x') < F^*) \textbf{ then} \]
\[ \overline{x} = x'; \quad \overline{m} = m'; \quad \overline{F} = F(x'); \]
\[ x_n = \overline{x}; \]
\[ \text{insert\_tabu\_list}(\overline{m}); \]
\[ \textbf{if} \ \overline{F} < F^* \textbf{ then} \]
\[ x^* = \overline{x}; \quad F^* = \overline{F}; \]
\textbf{while} \ \text{stop\_criterion} == \text{false};