4 Local Search

For realistic problems, complete search trees can be extremely large and difficult to prune effectively. It may often be more useful to get a reasonably good solution fast, rather than the globally optimal one after a long wait. In such cases, *local search* methods provide an interesting alternative.

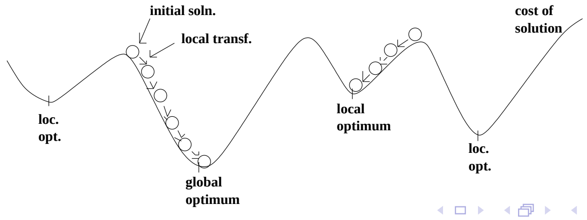
Assume that the search space X has some *neighbourhood structure* N, whereby for each solution $x \in X$, a set of "structurally close" solutions $N(x) \subseteq X$ can be easily generated from x by local transformations.

For instance, in the case of OPT-SAT one could have:
$N(t) =$
$\{$truth assignments t' that differ from t at exactly one variable$\}$,
and in the case of OPT-SG:
$N(\sigma) =$
$\{$spin configurations σ' that differ from σ at exactly one spin$\}$.

4.1 Deterministic Local Search

The simple *deterministic local search* method works by iteratively improving a given solution by neighbourhood transformations, as long as possible:

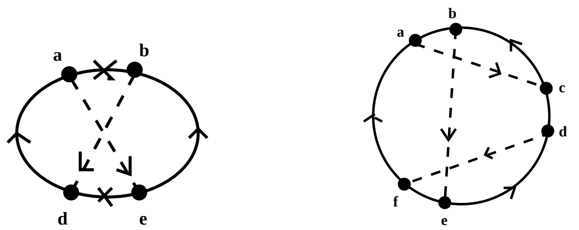
function det_LS (X, N, c):
  choose arbitrary initial solution $x \in X$;
  repeat
    find some $x' \in N(x)$ such that $c(x') < c(x)$;
    $x \leftarrow x'$
  until no such x' can be found;
  return x.



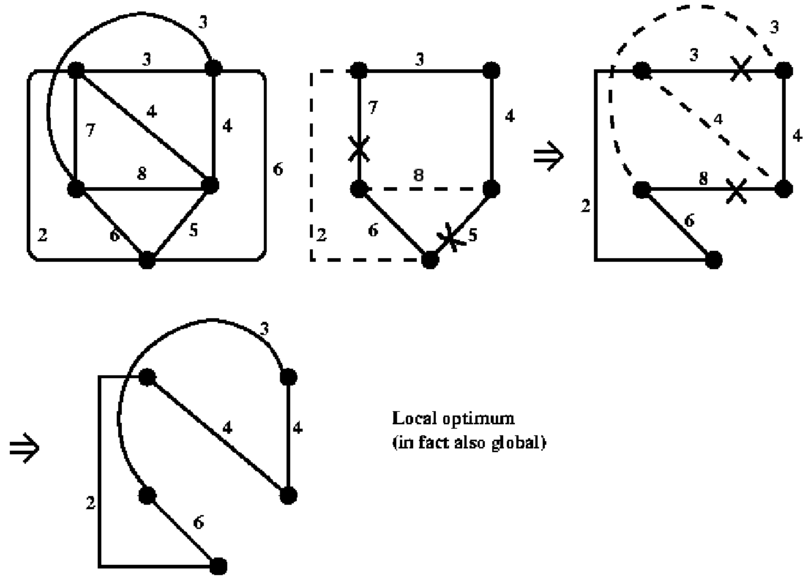
Local Search for TSP

Local search based on *Lin-Kernighan neighbourhoods* (figure below) has been experimentally shown to produce quite good results for the TSP. E.g. search based on the 3-opt neighbourhoods consistently produces tours only a few % longer than optimum. An even more powerful idea is to compose 2-opt steps into larger ones as long as tour improves.

Tour transformations defining the Lin-Kernighan 2-opt and 3-opt neighbourhoods:



A 2-Opt Descent to Local Optimum for TSP



4.2 Simulated Annealing

Local (nonglobal) minima are obviously a problem for deterministic local search, and many heuristics have been developed for escaping from them.

One of the most widely used is *simulated annealing* (Kirkpatrick, Gelatt & Vecchi 1983, Černy 1985), which introduces a mechanism for allowing also cost-increasing moves in a controlled stochastic way.

The amount of stochasticity is regulated by a *computational temperature* parameter T, whose value is during the search decreased from some large initial value $T_{init} \gg 0$ to some final value $T_{final} \approx 0$. A proposed move from a solution x to a worse solution x' is accepted with probability $e^{-\Delta c/T}$, where $\Delta c > 0$ is the cost difference of the solutions.

function SA(X, N, c):
    $T \leftarrow T_{init}$;
    $x \leftarrow x_{init}$;
    while $T > T_{final}$ **do**
        $L \leftarrow$ sweep(T);
        for L times **do**
            choose $x' \in N(x)$ uniformly at random;
            $\Delta c \leftarrow c(x') - c(x)$;
            if $\Delta c \leq 0$ **then** $x \leftarrow x'$ **else**
                choose $r \in [0,1)$ uniformly at random;
                if $r \leq \exp(-\Delta c/T)$ **then** $x \leftarrow x'$;
        end for;
        $T \leftarrow$ lower(T)
    end while;
    return x.

Cooling Schedules

An important question in applying simulated annealing is how to choose appropriate functions lower(T) and sweep(T), i.e. what is a good "cooling schedule" $\langle T_0, L_0\rangle, \langle T_1, L_1\rangle, \ldots$

There are theoretical results guaranteeing that if the cooling is "sufficiently slow", then the algorithm almost surely converges to globally optimal solutions. Unfortunately these theoretical cooling schedules are astronomically slow.

In practice, it is customary to just start from some "high" temperature T_0, and after each "sufficiently long" sweep L decrease the temperature by some "cooling factor" $\alpha \approx 0.8 \ldots 0.99$, i.e. to set $T_{k+1} = \alpha T_k$.

Theoretically this is much too fast, but often seems to work well enough. No one really understands why.

Convergence of Simulated Annealing

View the search space X with neighbourhood structure N as a *graph* (X,N). Assume that this graph is undirected, connected, and of degree r. (Each node=solution has exactly r neighbours.)

Denote by $X^* \subseteq X$ the set of globally optimal solutions. The following result was proved by Geman & Geman (1984) and Mitra, Romeo & Sangiovanni-Vincentelli (1986):

Theorem. Consider a simulated annealing computation on structure (X, N, c). Assume the neighbourhood graph (X, N) is connected and regular of degree r. Denote:

$$\Delta = \max\{c(x') - c(x) \mid x \in X, x' \in N(x)\}.$$

Choose

$$L \geq \max_{x \in X \setminus X^*} \min_{x^* \in X^*} \operatorname{dist}(x, x^*),$$

where $\operatorname{dist}(x, x^*)$ is the shortest-path distance in graph (X, N) from node x to node x^*. Suppose the cooling schedule used is of the form $\langle T_0, L\rangle, \langle T_1, L\rangle, \langle T_2, L\rangle, \ldots$, where for each cooling stage $\ell \geq 2$:

$$T_\ell \geq \frac{L\Delta}{\ln \ell} \quad (\text{but } T_\ell \xrightarrow[\ell \to \infty]{} 0).$$

Then the distribution of states visited by the computation converges in the limit to π^*, where

$$\pi^*_x = \begin{cases} 0, & \text{if } x \in X \setminus X^*, \\ 1/|X^*|, & \text{if } x \in X^*. \end{cases}$$

4.3 Tabu Search (Glover 1986)

Idea: Prevent a local search algorithm from getting stuck at a local minimum, or cycling at a set of solutions with the same objective function value, by maintaining a limited history of recent solutions (*tabu list*) and excluding those solutions from the move selection process.

function TABU(c, tt):
    $x \leftarrow$ initial feasible solution;
    initialise TL to $\{x\}$;
    while moves $<$ max_moves **do**
        remove from TL solutions entered there
            more than tt moves ago;
        choose an $x' \in N(x) \setminus TL$ of minimum cost;
        add x to TL;
        $x \leftarrow x'$
    end while;
    return best x seen so far.

Tabu Search: Practical Considerations

To save tabu list memory and access time, it may be worthwhile not to store complete solutions in the list, but just the recent *moves* (local transformations). This, however, introduces the problem that a move may be superfluously tabu at time t from the context of some earlier solution $x_{t'}$, $t' < t$, whereas it would lead to an interesting new solution in the context of solution x_t.

To resolve this issue, heuristics for overriding the tabu rule have been introduced, such as "always accept objective-improving moves" (i.e. such that $c(x') < c(x)$).

4.4 Record-to-Record Travel (Dueck 1993)

Idea: Candidate solution can move freely within a tolerance δ of the best ("record") solution value found so far. When a new record solution is found, the tolerance level falls correspondingly.

function RRT(c, δ):
  $x \leftarrow$ initial feasible solution;
  $x^* \leftarrow x$; $c^* \leftarrow c(x)$;
  while moves $<$ max_moves **do**
    choose some $x' \in N(x)$;
    if $c(x') \leq c^* + \delta$ **then** $x \leftarrow x'$;
    if $c(x') < c^*$ **then**
      $x^* \leftarrow x'$; $c^* \leftarrow c(x')$
  end while;
  return x^*.

4.5 Local Search for Satisfiability: GSAT (Gu, Selman et al. 1992)

Idea: View propositional satisfiability as an optimisation problem, where $c = c_F(t)$ is the number of unsatisfied clauses in formula F under truth assignment t. Apply a greedy (deterministic) local search strategy to minimise $c(t)$.

function GSAT(F):
  $t \leftarrow$ initial truth assignment;
  while flips $<$ max_flips **do**
    if t satisfies F **then return** t
    else
      find a variable x whose flipping in t causes
        largest decrease in $c(t)$ (if no decrease is
        possible, then smallest increase);
      $t \leftarrow$ (t with variable x flipped)
  end while;
  return t.

NoisyGSAT (Selman et al. ∼ 1996)

Idea: Augment GSAT by a fraction p of random walk moves.

function NoisyGSAT(F,p):
  $t \leftarrow$ initial truth assignment;
  while flips $<$ max_flips **do**
    if t satisfies F **then return** t
    else
      with probability p, pick a variable x
        uniformly at random;
      with probability $(1-p)$, do basic GSAT move:
        find a variable x whose flipping causes
        largest decrease in $c(t)$ (if no decrease is
        possible, then smallest increase);
      $t \leftarrow$ (t with variable x flipped)
  end while;
  return t.

4.6 The WalkSAT Algorithm (Selman et al. 1996)

Idea: NoisyGSAT *focused* on the unsatisfied clauses.

function WalkSAT(F,p):
  $t \leftarrow$ initial truth assignment;
  while flips $<$ max_flips **do**
    if t satisfies F **then return** t **else**
      choose a random unsatisfied clause C in F;
      if some variables in C can be flipped without
        breaking any presently satisfied clauses,
        then pick one such variable x at random; else:
      with probability p, pick a variable x in C unif. at random;
      with probability $(1-p)$, do basic GSAT move:
        find a variable x in C whose flipping causes
        largest decrease in $c(t)$;
      $t \leftarrow$ (t with variable x flipped)
  end while;
  return t.

WalkSAT vs. NoisyGSAT

The focusing seems to be important: in the (unsystematic) experiments in Selman et al. (1996), WalkSAT outperforms NoisyGSAT by several orders of magnitude. Later experimental evidence by other authors corroborates this.

Good values for the "noise" parameter p seem to be about $p \approx 0.5$. For instance, for large randomly generated 3-SAT formulas with clauses-to-variables ratio α near the "satisfiability threshold" $\alpha = 4.267$, the optimal value of p seems to be about $p = 0.57$.