3. Search Spaces and Objective Functions. Complete vs. Local Search

3.1. Search Spaces and Objective Functions

An instance I of a combinatorial search or optimisation problem Π determines a *search space* X of candidate solutions.

The computational difficulty in such problems arises from the fact that X is typically exponential in the size of I (= HUGE).

E.g. SAT:
Instance F = propositional formula on n variables $\{x_1, \ldots, x_n\}$.
Search space X = all truth assignments $t : \{x_1, \ldots, x_n\} \to \{0,1\}$.
Goal: find $t \in X$ that makes F true.

Size of $X = 2^n$ points ($0/1$-vectors).

Search Spaces and Objective Functions (II)

Note that if SAT formulas are required to be in conjunctive normal form (as in e.g. 3-SAT), then it can also be viewed as an optimisation problem:

OPT-3-SAT:
Instance F = family of m 3-clauses on n variables $\{x_1, \ldots, x_n\}$.
Search space X = all truth assignments $t : \{x_1, \ldots, x_n\} \to \{0,1\}$.
Objective (cost) function: $c(t)$ = number of clauses not satisfied by t.
Goal: minimise $c(t)$.

Search Spaces and Objective Functions (III)

OPT-TSP:
Instance: An $n \times n$ matrix D of distances d_{ij} between n "cities".
Search space: X = all permutations ("tours") π of $\{1, \ldots, n\}$.
Cost function: $d(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$.
Goal: minimise $d(\pi)$.

Note: Here $|X| = n!$. (More precisely: $|X| = (n-1)!/2$, if the starting points and orientations of tours are ignored.)

Search Spaces and Objective Functions (IV)

OPT-SG (or SPIN GLASS GROUND STATE):
Instance: An $n \times n$ matrix C of "coupling constants" c_{ij} between n "spins" and an n-vector h ("external field").
Search space: X = all "spin configurations" $\sigma \in \{-1,1\}^n$.
Cost function ("Hamiltonian"):

$$H(\sigma) = -\sum_{\langle i,j \rangle} c_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i.$$

Goal: minimise $H(\sigma)$.

Here again $|X| = 2^n$.

3.2 Complete Search Methods

Backtrack Search

Bactrack search is a systematic method to search for a satisfying, or an optimal solution x in a search space X.

function backtrack(I:instance; x:partialsol):
  if x is a complete solution **then**
    return x
  else
    for all extensions $e_1, \dots e_k$ to x **do**
      $x' \leftarrow$ backtrack($I, x \oplus e_i$);
      if x' is a complete solution **then return** x'
    end for;
    return fail
  end if.

Backtrack Search (II)

For instance, in the case of SAT, each partial truth assignment $t : \{x_1, \dots, x_i\} \to \{0, 1\}$ has two possible extension e_0 and e_1: one assigns value 0 to variable x_{i+1} and the other assigns value 1.

In the case of TSP, the partial solutions could be nonrepeating sequences of cities (initial segments of tours), and the extensions could be choices of next city. (Also other arrangements are possible).

Backtrack Search in Games

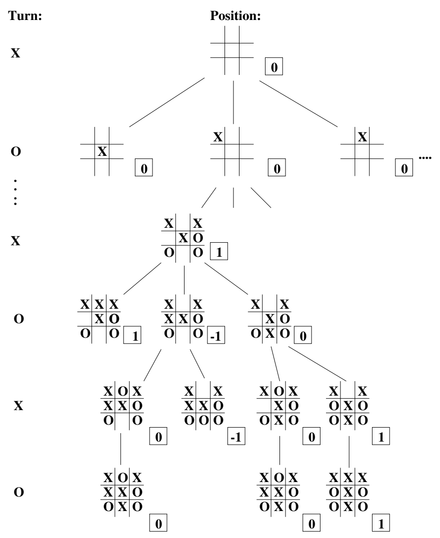
A backtrack search generates a *search tree* of increasingly complete partial solutions.

Let us illustrate this in the case of evaluating position values in a game of 3×3 noughts-and-crosses.

Associate to each incomplete position t in the game its *payoff value* for player X:

$$\text{payoff}(t) = \begin{cases} 1 & \text{if X has a winning strategy from } t, \\ -1 & \text{if O has a winning strategy from } t, \\ 0 & \text{if neither has a winning strategy from } t. \end{cases}$$

The complete annotated search, or *game tree* of this game is illustrated on the next slide.



The Minimax Rule

The payoff values for all positions can be computed by augmenting a backtrack search of the game tree with computations according to the following *minimax rules:*

Position type	*Payoff value*
final (complete)	can be determined directly
X moves	payoff = max {payoff values of immed. extensions}
O moves	payoff = min {payoff values of immed. extensions}

Bounded Depth Search

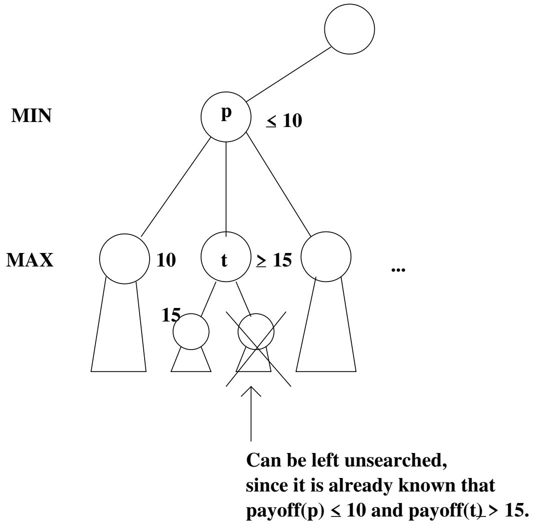
Game trees in realistic games are usually evaluated only to some predetermined *lookup depth* k, at which some heuristic *evaluation function* eval(t) is applied to estimate the payoff values of the incomplete positions t.


```
function payoff (t:position; k:depth; m:{MIN, MAX}):
    if k = 0 or t is a final position then
        return eval(t)
    else
        if m = MAX then v ← −∞ else v ← ∞;
        for all t's extensions s do
            if m = MAX then
                v ← max(v, payoff(s, k − 1, MIN))
            else
                v ← min(v, payoff(s, k − 1, MAX))
        end if;
        return v
    end if.
```

Alpha-Beta Pruning

The size of a search tree can often be considerably reduced by eliminating branches that cannot improve an already known solution. In the case of game trees this process is known as *alpha-beta pruning*.

During the backtrack search of the game tree, maintain at each node t an *intermediate payoff value*, which for MIN nodes is an upper bound on the eventual true payoff value, and for MAX nodes a lower bound. Then when it is clear that no further search below a given node t can improve the payoff value of its father, the remaining subtrees of t cab be pruned. (See next slide.)



function payoff$\alpha\beta$ (t, k, m, pv); $\quad$ pv = father's intermediate payoff
 if $k = 0$ or t is a final position **then return** eval(t)
 else
  if m = MAX **then** $v \leftarrow -\infty$ **else** $v \leftarrow \infty$;
  for all t's extensions s **do**
   if m = MAX **then**
    $v \leftarrow$ max(v, payoff$\alpha\beta$(s, $k-1$, MIN, v));
    if $v \geq pv$ **then return** v
   else
    $v \leftarrow$ min(v, payoff$\alpha\beta$ (s, $k-1$, MAX, v));
    if $v \leq pv$ **then return** v
   end if;
  return v
 end if.

Branch-and-Bound Search

Similar pruning techniques can greatly improve the efficiency of backtrack search in optimisation problems.

Consider e.g. the TSP problem and choose:

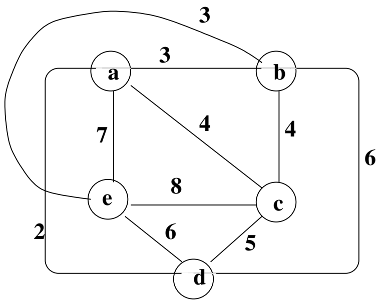
Partial solution: A set of edges (links) that have been decided to either include or exclude from the complete solution tour.
Bounding heuristic: Let the TSP instance under consideration be given by distance matrix $D = d_{ij}$. Then the following inequality holds for any complete tour π:

$$\begin{aligned} d(\pi) &= \frac{1}{2}\sum_j \{(d_{ij} + d_{jk}) \mid \text{at city } j \text{ tour } \pi \text{ uses links } ij \text{ and } jk\} \\ &\geq \frac{1}{2}\sum_j \min_{i,k}(d_{ij} + d_{jk}). \end{aligned}$$

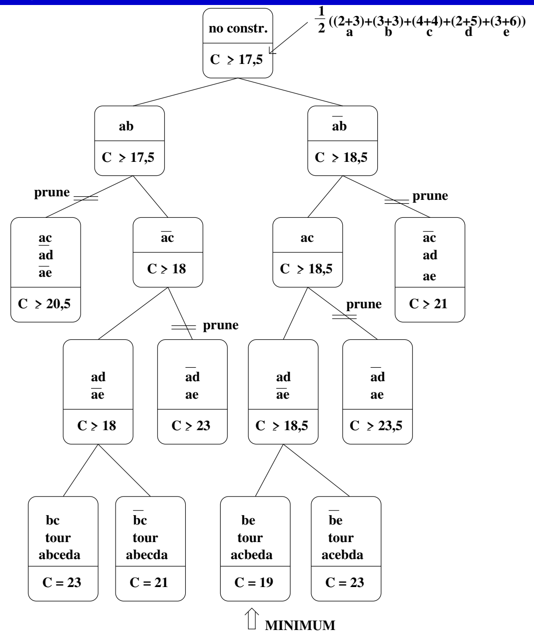
This estimate can be used to lower bound the length of tours achievable from any given partial solution, and prune the search tree correspondingly.

Branch-and-Bound Search (II)

Consider the following small TSP instance:



Using the above lower-bounding heuristic, the search tree for the minimum tour on this instance can be pruned as presented on the following slide.



3.3 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.

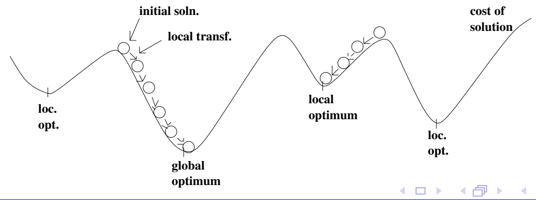
Assume that the search space X has some *neighbourhood structure* N, whereby for each solution $x \in X$, a set of "structurally close" solutions $N(x) \subseteq X$ can be easily generated from x by local transformations.

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$\}$.

Deterministic Local Search

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

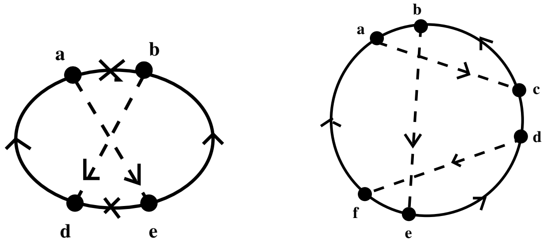
function det_LS (X, N, c):
  choose arbitrary initial solution $x \in X$;
  repeat
    find some $x' \in N(x)$ such that $c(x') < c(x)$;
    $x \leftarrow x'$
  until no such x' can be found;
  return x.



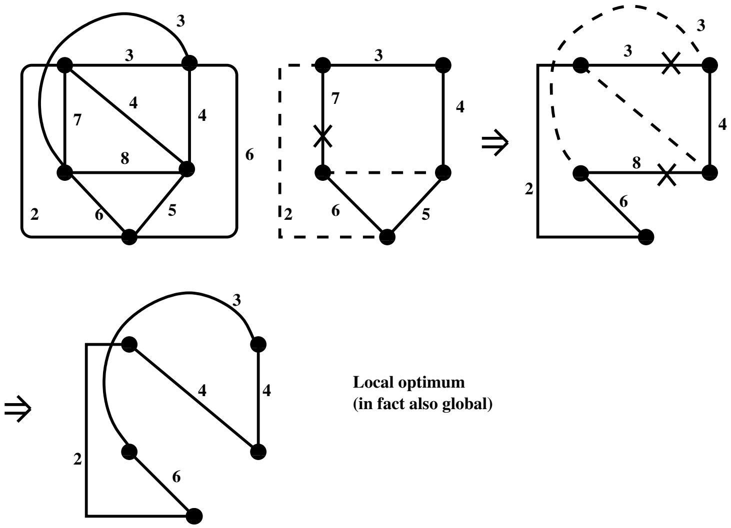
Local Search for TSP

Local search based on *Lin-Kernighan neighbourhoods* (figure below) has been experimentally shown to produce quite good results for the TSP. In particular, search based on the 3-opt neighbourhoods consistently produces tours only a few % longer than optimum.

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



A 2-Opt Descent to Local Optimum for TSP



3.4 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.