3 Search Spaces and Objective Functions. Complete Search Methods

3.1 Search spaces and objective functions (1/4)

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(D):
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 (2/4)

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:

3-SAT(O):
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: $c(t)$ = # clauses not satisfied by t.
Goal: minimise $c(t)$.

Search spaces and objective functions (3/4)

TSP(O):
Instance: An $n \times n$ matrix D of distances d_{ij} between n "cities".
Search space: X = all permutations ("tours") π of $\{1, \ldots, n\}$.
Objective 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 (4/4)

SPINGLASS (SPIN GLASS GROUND STATE)
Instance: An $n \times n$ matrix J of "coupling constants" J_{ij} between n "spins" and an n-vector h ("external field"). Together these define an objective function ("Hamiltonian") that for any spin state or configuration $s \in \{-1, 1\}^n$ has value

$$H(s) = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j - \sum_i h_i s_i .$$

Goal:
(O) Find a spin configuration $s \in \{-1, 1\}^n$ that minimises $H(s)$ ("ground state" of the system).

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

3.2 Complete Search Methods

- Backtrack search
- The DPLL procedure
- Branch-and-bound search
- The A* algorithm

3.2.1 Backtrack search (1/2)

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 $e_1, \ldots e_k$ 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 (2/2)

For instance, in the case of SAT, each partial truth assignment $t : \{x_1, \ldots, x_i\} \rightarrow \{0,1\}$ 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).

3.2.2 The DPLL (Davis-Putnam-Logemann-Loveland) procedure

A backtrack search method for testing satisfiability of a set of clauses Σ on variable set V. Basic outline:

- If Σ is empty, return “satisfiable”.
- If Σ contains an empty clause, return “unsatisfiable”.
- If Σ contains a unit clause $c = x^{\pm}$, assign to x a value which satisfies c, simplify the remaining clauses correspondingly, and call DPLL recursively.
- Otherwise select an unassigned $x \in V$, assign $x \leftarrow 1$, simplify Σ, and call DPLL recursively. If this call returns “satisfiable”, then return “satisfiable”; else assign $x \leftarrow 0$, simplify Σ, and call DPLL recursively again.

T–79.4201 Search Problems and Algorithms

3.2.3 Branch-and-bound search (1/2)

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:

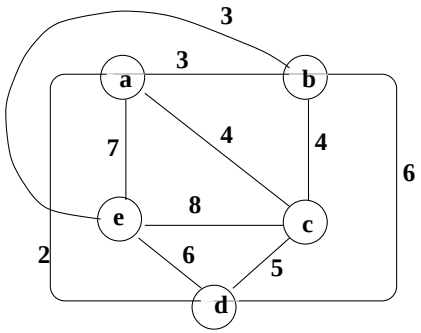
$$\begin{aligned} d(\pi) &= \frac{1}{2}\sum_i \{(d_{ij} + d_{jk}) \mid \text{at city } j \text{ tour } \pi \text{ uses links } ij \text{ and } jk\} \\ &\geq \frac{1}{2}\sum_j \min_{i,k}(d_{ij} + d_{jk}). \end{aligned}$$

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

T–79.4201 Search Problems and Algorithms

Branch-and-bound search (2/2)

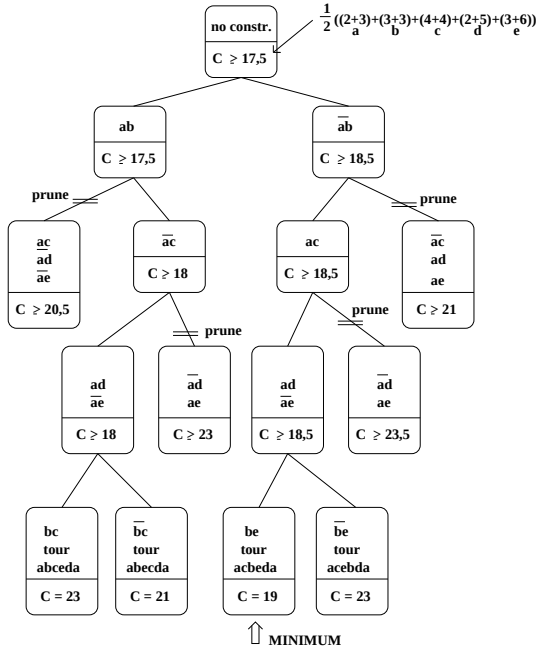
Consider the following small TSP instance:



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.

I.N. & P.O. Autumn 2007

T–79.4201 Search Problems and Algorithms



3.2.4 The A* algorithm

A* is basically a reformulation of the branch-and-bound search technique in terms of path search in graphs.

Given:

- search graph [neighbourhood structure] (X, N)
- start node $x_0 \in X$
- set of goal nodes $X^* \subseteq X$
- edge costs $c(x, x') \geq 0$ for $x \in X$, $x' \in N(x)$

Goal: find a (minimum-cost) path from x_0 to some $x \in X^*$.

function A*(X, N, x_0, c, h):
    place x_0 in OPEN; set $g(x_0) = 0$, $f(x_0) = h(x_0)$;
    while OPEN $\neq \emptyset$ **do**
        choose some $x \in$ OPEN for which $f(x)$ is minimum;
        if $x \in X^*$ **then return** {found path to x};
        move x from OPEN to CLOSED;
        for all $x' \in N(x)$ **do**
            if x' is not yet in OPEN or CLOSED **then**
                estimate $h(x')$;
                compute $f(x') = g(x') + h(x')$,
                    where $g(x') = g(x) + c(x, x')$;
                place x' in OPEN
            else {x' is already in OPEN or CLOSED}
                recompute $f(x') = g(x') + h(x')$;
                if x' was in CLOSED and its f-value decreased **then**
                    move x' from CLOSED to OPEN
    end while;
    return fail {no path to goal found}.

A*: Path length estimation

An important feature of A* is that the remaining distance from a node x to a goal node is estimated by some heuristic $h(x) \geq 0$.

As the algorithm visits a new node, it is placed in a set OPEN. Nodes in OPEN are selected for further exploration in increasing order of the evaluation function

$$f(x) = g(x) + h(x),$$

where $g(x) \geq \text{dist}(x_0, x)$ is the shortest presently known distance from the start node. (Here $\text{dist}(x_0, x)$ denotes the true minimum-cost distance from x_0 to x.)

A heuristic $h(x)$ is admissible, if it underestimates the true remaining minimum-cost distance $h^*(x)$, i.e. if for all $x \in X$:

$$h(x) \leq h^*(x) := \min_{x^* \in X^*} \text{dist}(x, x^*).$$

A*: Convergence

A basic property of the A* algorithm is the following:

Theorem. Assume that the heuristic h is admissible. If the graph (X, N) is finite, and some path from x_0 to X^* exists, then A* returns one with a minimum cost.

Proof idea. By induction on the number of search steps, using the fact that h is admissible ($h(x) \leq h^*(x)$), establish the following:

- When a node x is moved from OPEN to CLOSED, all the preceding nodes x' on some min-cost path from x_0 to x are already in CLOSED and satisfy $g(x') = \text{dist}(x_0, x')$. After the move the same holds for x. (Thus with an admissible h, nodes are never re-OPENed.)
- Until a goal node x^* is encountered, there is always some node x in OPEN that lies on some min-cost start-to-goal path (and thus will eventually be expanded).

A*: Notes

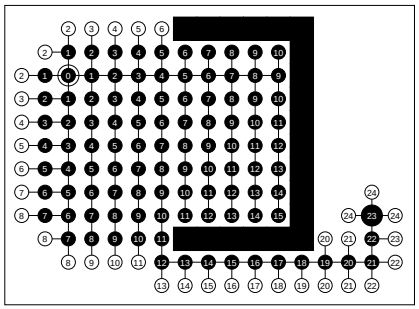
Note 1: The A* convergence theorem holds even for infinite search graphs satisfying some structural conditions. (Every node has only finitely many neighbours and all infinite paths have infinite cost.)

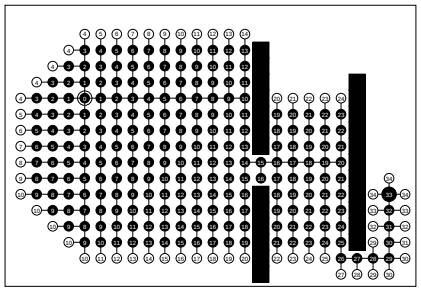
Note 2: Convergence of the algorithm can be guaranteed also for nonadmissible heuristics, but very little can be said about the cost of the paths returned in that case.

Note 3: The special case $h(x) \equiv 0$ reduces to the well-known Dijkstra's algorithm for shortest paths in graphs.

A*: Examples

In these two examples of A* search in graphs with obstacles, the heuristic $h(x)$ is taken to be the Manhattan (square-block) distance from a node x to the goal node x^* when the obstacles are ignored. The white nodes are in OPEN and the black nodes in CLOSED when the algorithm terminates.





A* applied to SAT

Given propositional formula F on n variables $\{x_1, \ldots, x_n\}$ in conjunctive normal form, choose:

- search graph (T, N), where
  - T = all partial truth assignments $t : \{x_1, \ldots, x_n\} \to \{0, 1, \bot\}$.
  - $(t, t') \in N$ if there is a unique variable x_i for which $t(x_i) = \bot$, $t'(x_i) = 0/1$.
- start node $t_0 = \bot^n$.
- goal nodes T^* = truth assignments satisfying F.
- edge costs $c(t, t') = 1$ for $t \in T$, $t' \in N(t)$.

Actually, a more relevant edge cost function for a formula F with m clauses would be:

$c(t, t') = m + (\#\text{clauses unsatisfied by } t') - (\#\text{clauses unsatisfied by } t)$.

What would be good heuristics in this case?