**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

$$L \geq \min_{x \in X, x^* \in X^*} \max \text{dist}(x, x^*),$$

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

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

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

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

---

**3.5 The A* Algorithm**

*Note:* $A^*$ is actually a complete algorithm, so should have been presented earlier.

$A^*$ 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 $x \in X, x' \in N(x)$

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

---
**A*: Path Length Estimation**

An important characteristic 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 $g(x) = \text{dist}(x_0, x)$ is the shortest presently known distance from the start node.

A heuristic $h(x)$ is **admissible**, if it underestimates the true remaining minimal 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.

**Note 1:** This result 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.

3.6 Tabu Search (Glover 1986)

Note: Now we return to local search algorithms.

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.

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.

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 $c(x') < c(x)$). 

function TABU($c$, $tt$):

- $x$ ← 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;
  - $x ← x'$
- end while;
- return best $x$ seen so far.
3.7 Record-to-Record Travel (Dueck 1993)

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

function RRT($c$, $\delta$):

\[
\begin{align*}
  x & \leftarrow \text{initial feasible solution;} \\
  x^* & \leftarrow x; c^* \leftarrow c(x); \\
  \text{while moves} & < \text{max\_moves do} \\
  & \text{choose some } x' \in N(x); \\
  & \quad \text{if } c(x') \leq c^* + \delta \text{ then } x \leftarrow x'; \\
  & \quad \text{if } c(x') < c^* \text{ then} \\
  & \quad \quad x^* \leftarrow x'; c^* \leftarrow c(x') \\
  & \text{end while;} \\
  \text{return } x^*.
\end{align*}
\]
RRT in Action ($\delta = 2$)
**3.8 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 ← 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 ← (t with variable \( x \) flipped)
        end while;
    return t.
```

**3.9 The WalkSAT Algorithm (Selman et al. 1996)**

*Idea:* NoisyGSAT focused on the unsatisfied clauses.

```
function NoisyGSAT(F,p):
    t ← 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 ← (t with variable \( x \) flipped)
        end while;
    return t.
```
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 $\alpha$ near the “satisfiability threshold” $\alpha = 4.267$, the optimal value of $p$ seems to be about $p = 0.57$. 

I.N. & P.O. Spring 2006