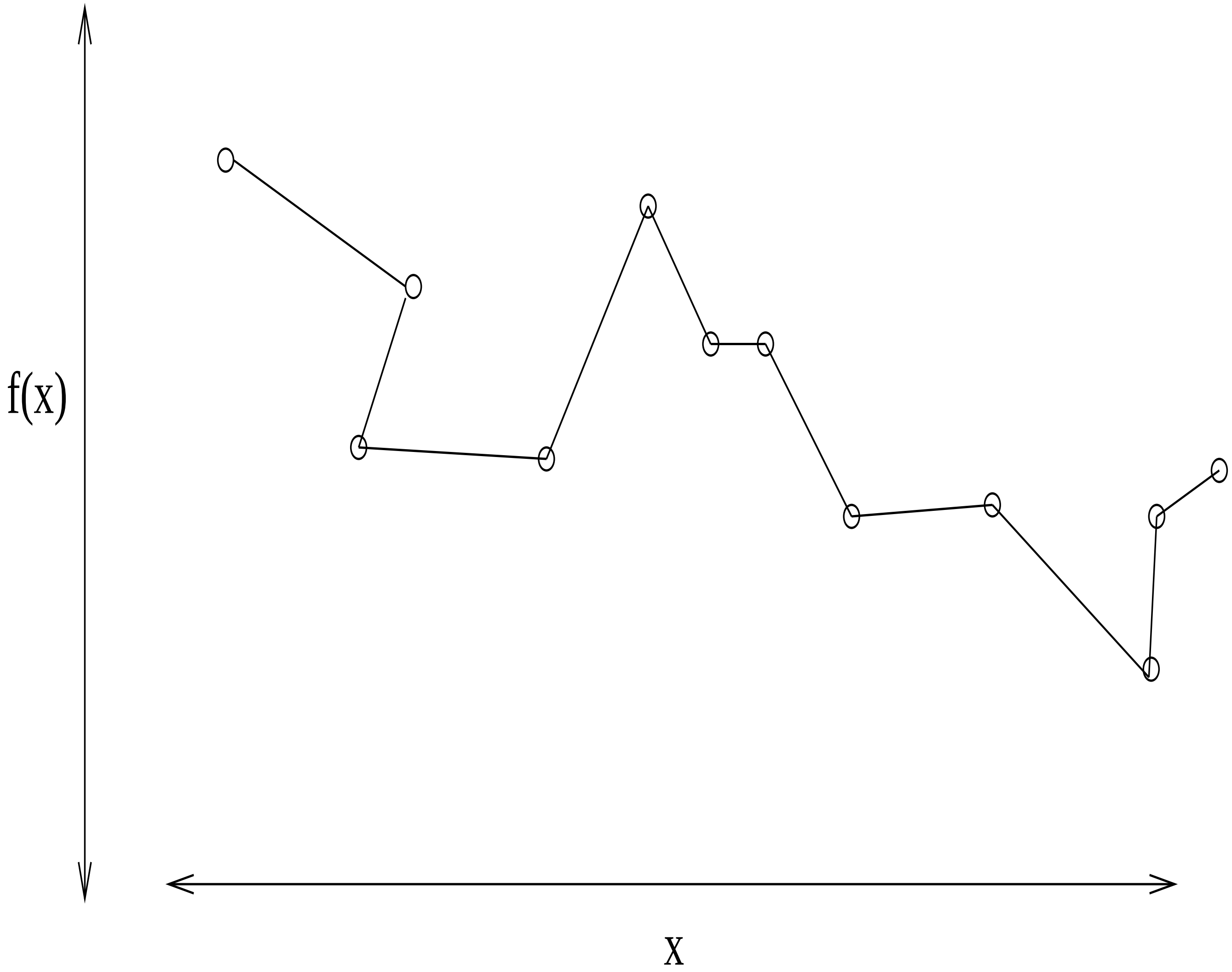
ALGORITHMS AND HEURISTICS FOR COMBINATORIAL OPTIMIZATION PROBLEMS: CASE STUDY OF ACADEMIC COURSE SCHEDULING

Saleh Elmohamed

Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY 13244

January 22, 2000

LOCAL SEARCH


f(x)
x

Main idea of Local Search

- Never actually try to write the whole graph,
- Just move from one possible solution to a "nearby" possible solution,
- Stop when "optimal" solution is found or time runs out.

Local Search Algorithm

To set up an algorithm we need to have:

1. Set of possible solutions $\Longrightarrow$ vertices in the graph
2. **neighbors** of each vertex:
   - For x, $N(x)$ is set of adjacent vertices.
   - $N(x)$ is easy to compute from x.
   - if $y \in N(x)$ then $x \in N(y) \Longrightarrow$ represent neighbors as undirected edges.
   - $N(x)$ need to be polynomially bounded in the input size so neighbors of a node are easy to search through.
3. Cost function: $f\colon \mathcal{S} \longrightarrow \mathcal{R}$, where $\mathcal{S}$ is the set of feasible solutions.

Most basic LS algorithm

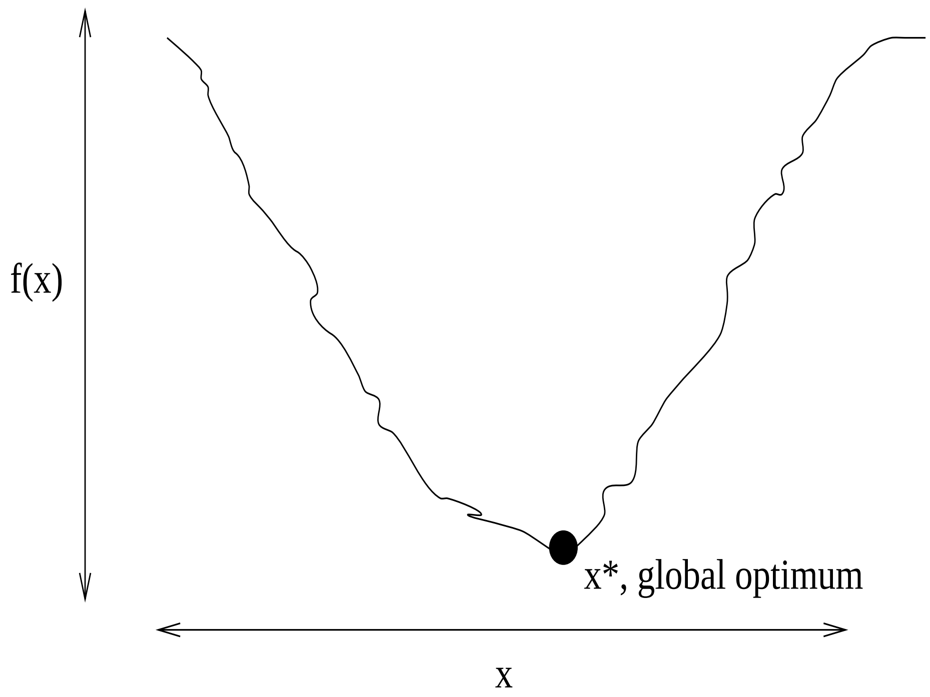
Say, to minize a cost function:

- Pick a starting point x.
- While there is a neighbor y of x with $f(y) < f(x)$, move to it; that is, set x to y and continue.
- Return the final solution.

Reasoning Behind Local Search

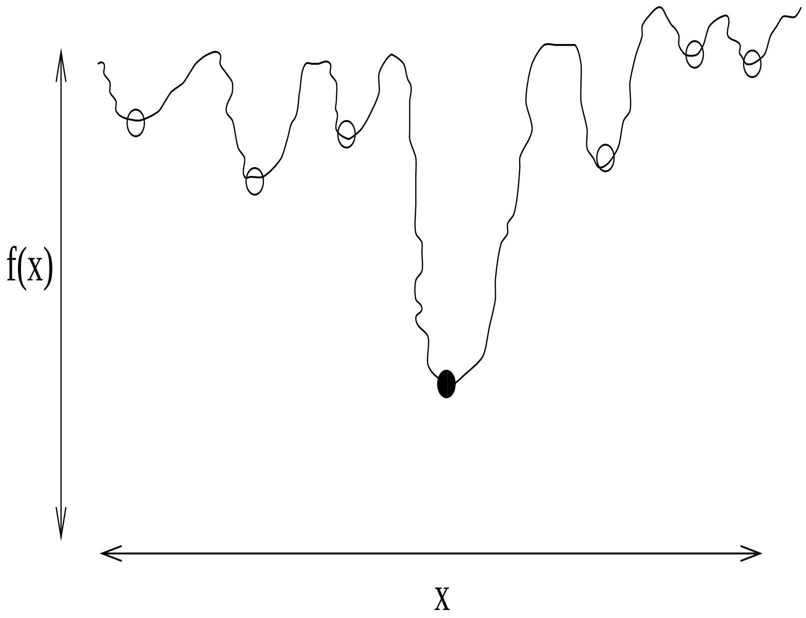
Getting better and better solutions, we should end up with a good one.

Pictorially, a nice state space that we would like to have:



Problems with the rationale ...

- Solution space might not looked as the above above ...
- Example, cost function might not change smoothly when moving from a state to its neighbor ...
- There might be several **local optima**, in which case the algorithm hone in one and get stuck there ... As shown below:



The Key Points ...

- The way the problem is set up is quite important...
- When choosing possible moves between solutions – that is, when constructing the mapping that gives us the neighborhood of each node — we are setting up:
  1. how local search behave,
  2. how cost function will change between neighbors,
  3. how many local optima there are.
- Setting up the right neighborhood, so that solution space looks the way we like ... is essential.

Local Search variations

Assuming the goal is to minimize the cost function ...

- **Hill-climbing**: one moves to a vertex of lower (or possibly equal) cost.
- **Metropolis rule**: pick a random neighbor...
  - if cost is lower, move there.
  - if cost is higher, move there with some probability (depending on the cost differential).

  The idea is that possibly moving to a worse state helps avoid getting trapped at local minima.

Local Search variations – continue

- **Simulated annealing**: Similar to Metropolis rule, except the probability of going to a higher cost neighbor varies with time. Same as a physical system being cooled down over time.
- **Parallel search**: ("go with the winners") do multiple searches in parallel, occasionally killing off searches that appear less successful and replacing them with copies of searches that appear to be doing better.
- **Genetic algorithms**: Quite related to local search. An important difference is that instead of keeping one solution at a time, a group of them (called a population) is kept, and the population changes at each step.

SIMULATED ANNEALING – ANNEALING SCHEDULES

geometric cooling

- New temperature $(T^{'})$ is computed using: $T^{'} = \alpha T$ where α $(0 < \alpha < 1)$ denotes the cooling factor.
- Disadvantages:
  - the scheme does not take account of the state of the system in the process of searching the solution space...
  - difficult to adaptively modify the intensity of the search depending on the difficulty of the problem...
  - at high T almost all proposed interchanges are (mostly) accepted, even though many of them could be non-productive.

reheating as a function of cost

Key points about the scheme ...

- Compute **specific heat** (C_H): a measure of the variance of the cost values of states at a given temperature... This is needed to compute T occurring at a phase transition.
- A phase transition occurs at a temperature $T(C_H^{max})$ when the specific heat is maximal (C_H^{max}) $\Longrightarrow$ change in the state ordering.
- If current solution has a high cost $\Longrightarrow$ the super-structure may require re-arrangement. How?
  - done by raising the temperature to a level which is higher than the phase transition temperature $T(C_H^{max})$.
- In general: the higher the current best cost, the higher the temperature which is required to escape the local minimum.

reheating as a function of cost – continue

At each T, the algorithm generates a set of configs $\mathcal{C}(T)$. Let C_i denotes cost of config i, $C(T)$ is avg cost, and $\sigma(T)$ is STD of cost at T.

- At T, the prob. dist. for configurations is: $P_i(T) = \frac{e^{\frac{-C_i}{kT}}}{\sum_j e^{\frac{-C_j}{kT}}}$
- The average cost is computed as: $< C(T) >= \Sigma_{i \in \mathcal{C}} C_i P_i(T)$
- The average square cost is: $< C^2(T) >= \Sigma_{i \in \mathcal{C}} C_i^2 P_i(T)$
- Variance of the cost is: $\sigma^2(T) =< C^2(T) > - < C(T) >^2$
- Now, the specific heat is defined as: $C_H(T) = \frac{\sigma^2(T)}{T^2}$

$T(C_H^{max})$ at which the maximum specific heat occurs, or at which the sys undergoes a phase transition, can thus be found.

Reheating sets the new T to be: $T = K \cdot C_b + T(C_H^{max})$

where K is a parameter and C_b is the current best cost.

Adaptive Cooling

A new T is computed based on the specific heat, i.e. the STD of all costs obtained at the current T.

main idea: to keep the system close to equilibrium, by cooling slower close to the phase transition, where the specific heat is large.

Let T_j denote the current temp, at step j of the schedule. Compute $\sigma(T_j)$ from equation then new T_{j+1} is computed as follows:

$$T_{j+1} = T_j \cdot e^{-\frac{aT_j}{\bar{\sigma}(T_j)}}$$

where a is a tunable parameter.

$\sigma(T_j)$ is smoothed out in order to avoid any dependencies of the temperature decrement on large changes in the STD. The smoothed STD $\bar{\sigma}$:

$$\bar{\sigma}(T_{j+1}) = (1-\omega)\sigma(T_{j+1}) + \omega\sigma(T_j)\frac{T_{j+1}}{T_j}$$

and set ω to 0.95

Academic Time Period	Algorithm	Scheduled (average) %	Highest Scheduled %	Lowest Scheduled %
First Semester	SA (geometric)	93.90	95.12	85.20
	SA (adaptive)	98.80	99.20	95.00
	SA (cost-based)	100.0	100.0	100.0
Second Semester	SA (geometric)	95.00	98.95	89.40
	SA (adaptive)	99.00	99.50	98.50
	SA (cost-based)	100.0	100.0	100.0
Third Semester	SA (geometric)	97.60	98.88	90.90
	SA (adaptive)	100.0	100.0	100.0
	SA (cost-based)	100.0	100.0	100.0