Optimization and local search

Prof Dr Marko Robnik-Šikonja

Analysis of Algorithms and Heuristic Problem Solving
November 2019
Search

* search is a basic problem solving mechanism
* many algorithms can be viewed as search algorithms
* problem states
* state space (reachable states - a graph, a tree)

$$S = \{S; S_Z \xrightarrow{*} S\}$$

* connections between states, a neighborhood generator N(S)
State space representation

*State space: \( S = \{ S; S_Z \xrightarrow{*} S \} \)

*quality of a state: \( q(S) \)

*optimal solution: \( S_0 = \arg \min_{S \in S} q(S) \)

*local optimum

\[ S_l = \{ S; \forall S \rightarrow S', q(S) \leq q(S') \} \]
Properties of local search

* local search, LS; local optimization, LO
* LS starts in a randomly generated state (solution) and tries to optimize it using local transformations
* the set of transformations determines the complexity of the algorithm
* algorithm reruns return different solutions
* ergo: repeat LS and return the overall best solution
LS basic scheme

LS($S_0$) { // $S_0$ is a starting state
    $S_m$ = $S_0$
    do {
        $N(S_0) = \{S; S_0 \rightarrow S\}$
        $S_0 = \arg\min_{S' \in N(S_0)} q(S')$
        if ( $q(S_0) < q(S_m) $ )
            $S_m = S_0$
        else
            break;
    } while (true) ;
    return( $S_m$ ) ;
}
LS problems

* local and global extremes
* plato,
* ridge
Local extremes
Plato
Ridge
Metropolis algorithm and simulated annealing

* generalization of greedy LS
* if better neighbor exists, move to it
* otherwise choose a random neighbor, but accept better neighbors with larger probability
* in time, stochastic search turns into deterministic LS
Physical background

* idea from thermodynamics
* Boltzmann distribution law says that the probability of a system being in a state with energy $E_i$ is proportional to:

\[ P(E_i) = e^{-\frac{E_i}{kT}} \]

where $T$ is temperature and $k$ a positive constant.
* therefore the probability of low energy state is larger if the temperature is lower
* to reach low energy state i.e., a nice crystal (optimal state), the molten matter has to be cooled slowly
* cooling too fast gives a suboptimal state (imperfect crystal). The slower we cool the matter, the more probable we get a nice crystal (but the algorithm will be slower).
Simulated annealing - the idea

* use the idea to introduce stochastic element to LS
* next state is selected stochastically
* better neighbors are selected with higher probability
* use temperature as a knob for stochastic behavior
* larger temperature implies larger probability for acceptance of worse neighbor and vice versa
* with $T = 0$ the algorithm is deterministic
Search

* start with a random state $S$
* select random neighbor $S'$
* if $q(S') < q(S)$ move to $S'$ with probability 1
* otherwise with probability

$$P(S \rightarrow S') = e^{-\frac{(q(S')-q(S))}{T}}$$
Metropolis algorithm

Metropolis($S_0$, $T$) { // $S_0$ is a starting state, $T$ is a temperature
    $S = S_m = S_0$ ;
    do {
        select $S'$ randomly from neighborhood $N(S) = \{S'; S \rightarrow S'\}$
        if ( $q(S') < q(S_m)$ )
            $S_m = S'$ ;
        if ( $q(S') < q(S)$ )
            $S = S'$ ; // move
        else
            with probability $e^{\frac{-(q(S')-q(S))}{T}}$ make a move $S = S'$
    } while (! stopping condition) ;
    return( $S_m$ ) ;
}
Annealing

* decrease temperature while it is not close to zero
* slower decreasing will search larger portion of the space and increases probability for optimal state
* usually a geometrical rule is used

\[ T' = \lambda T, \quad 0 < \lambda < 1 \]

* typically: \( \lambda = 0.95 \)
* end with deterministic LS
Algorithm SA

\[ SA(S_0, \lambda, T) \{ // S_0 \text{ is a starting state,} \]
\[ // \lambda \text{ is annealing schedule, } T \text{ is starting temperature} \]
\[ S = S_m = S_0 ; \]
\[ \text{do } \{ \]
\[ \text{randomly select } S' \text{ from } N(S) = \{S'; S \rightarrow S'\} \]
\[ \text{if ( } q(S') < q(S_m) \text{ )} \]
\[ S_m = S' ; \]
\[ \text{if ( } q(S') < q(S) \text{ )} \]
\[ S = S' ; // \text{ move} \]
\[ \text{else } \{ \]
\[ \text{with probability } e^{-\frac{(q(S')-q(S))}{T}} \text{ make a move } S = S' \]
\[ T = \lambda T ; \]
\[ \} \]
\[ \text{while ( ! stopping criterion ) } ; \]
\[ LS(S_m) ; // \text{ end with pure LS} \]
\[ \text{return} ( S_m ) ; \]
\[ \} \]
Max-cut and LS

* state space representation
* define neighborhood
* proof of LS being a 2-approximation algorithm
Algorithm with LS

Max-Cut-Local (G, w) {
    Pick a random node partition (A, B)

    while (∃ improving node v) {
        if (v is in A) move v to B
        else move v to A
    }

    return (A, B)
}
Neighborhood selection

* large enough not to stop too fast in local extreme
* small enough not to be too computationally expensive
* an example: K-L heuristics for max-cut
Best response dynamics

* multicast routing problem
* each agent searches the best solution for itself
* stability of solutions and Nash equilibrium
* relation to local search
* social choice
* price of stability
Guiding several agents

Best-Response-Dynamics(G, c) {
    Pick a path for each agent

    while (not a Nash equilibrium) {
        Pick an agent i who can improve by switching paths
        Switch path of agent i
    }
}

• the algorithm provably stops in Nash equilibrium
• define a potential function whose value strictly decreases in each step
Literature