L9 – Stochastic Local Search

Outline

• A New Search Paradigm
• SLS Definitions
• Basic Hill Climbing (example on M/G-NR-TSP)
• Various SLS Ideas (all on TSP)
• Small Experiments throughout the Lecture
Recall: Lecture 1

- COP = Combinatorial Optimization Problem
- Many of them are NP-hard
- Still remember the $3C_2$ reality?
- This time, we will also sacrifice optimality
  - But unlike Approximation Algorithms, this time we will **NOT** have any guarantee of the solution quality...
  - Theoretical Computer Scientists won’t like this...
Solving NP-hard Combinatorial Optimization Problems (COPs) through **Complete Search** that **sacrifices speed** is usually by iteratively (or recursively) generate and evaluate (all) candidate solutions

- e.g. Try all $(N-1)!$ possible TSP tours one by one, evaluate them, and report the best (minimal one)

- Note: Evaluating one candidate solution (e.g. compute the cost of a given TSP tour) is typically computationally much cheaper than finding one (out of possibly many) optimal solutions (e.g. find the optimal TSP tour)
A New Search Paradigm

What you already know: **Systematic Search**:

- Traverse search space for given problem instance in a *systematic manner*
- **Complete**: Guaranteed to eventually find (optimal) solution, or to determine that no solution exists

A New Paradigm: **Local Search**:

- Start at a (random) position in search space
- Iteratively move from a position to its neighbouring position, usually (but not always) perturbative (next slide)
- **Typically incomplete**: Not guaranteed to find (optimal) solutions, cannot determine insolubility with certainty...
A New Search Paradigm, Continued

• Perturbative Search
  – search space = complete candidate solutions
  – search step = modification of one/more sol. components
  – e.g. swap two edges (2-exchange) in a TSP tour

• Constructive Search (aka construction heuristics)
  – search space = partial candidate solutions
  – search step = extension with one/more sol. components
  – e.g. from one vertex, go to nearest neighbor vertex, the Greedy Nearest Neighbor heuristic
Systematic versus Local Search

• **Completeness**: Advantage of systematic search, but not always relevant, e.g., when existence of solutions is guaranteed by construction or in real-time situations (e.g. TSP when input is a complete graph).

• **Any-time property**: Positive correlation between run-time and solution quality or probability; typically more readily achieved by Local Search.

• **Complementarity**: Local and Systematic Search can be fruitfully combined, e.g., by using Local Search for finding solutions whose optimality is proven using Systematic Search.
When to use?

• **Systematic search** is often better suited when ...
  – proofs of insolubility or optimality are required;
  – time constraints are not critical;

• **Local search** is often better suited when ...
  – reasonably good solutions are required within a short time;
  – parallel processing is used;
The term Stochastic in SLS

• Many prominent local search algorithms use randomised (stochastic) choices in generating and modifying candidate solutions.

• These Stochastic Local Search (SLS) algorithms are one of the most successful and widely used approaches for solving hard combinatorial problems.

• Some well-known SLS methods and algorithms:
  – Evolutionary (Genetic) Algorithms
  – Simulated Annealing
  – Tabu Search (Steven’s old favourite due to his PhD)
SLS — global versus local view

- S = solution, C = current search position

Diagram: 
- Improving local move(s)
- Plateau move(s)
- Non-improving local move(s)
For a given problem instance $\pi$ of a COP:

- **search space** $S(\pi)$
  - e.g., for TSP: set of all possible TSP tours

- **solution set** $S'(\pi) \subseteq S(\pi)$
  - e.g., for TSP: TSP tours of minimum length

- **neighbourhood relation** $N(\pi) \subseteq S(\pi) \times S(\pi)$
  - e.g., for TSP: 2-exchange neighbourhood

- **set of memory states** $M(\pi)$
  - May be not used in some memoryless SLS algorithms
  - e.g., tabu list in Tabu Search algorithm (next lecture)
Continued:

- **(init)ialization function**: \( \emptyset \rightarrow D(S(\pi) \times M(\pi)) \)
  - Specifies probability distribution over initial search positions and memory states

- **step function**: \( S(\pi) \times M(\pi) \rightarrow D(S(\pi) \times M(\pi)) \)
  - Maps each search position and memory state onto probability distribution over subsequent, neighbouring search positions and memory states

- **termination function**: \( S(\pi) \times M(\pi) \rightarrow D(\{T, F\}) \)
  - Determines the termination probability for each search position and memory state
Generic SLS Algorithm

```
procedure SLS-Minimisation(π')
    input: problem instance π' ∈ Π'
    output: solution s ∈ S'(π') or ∅
    (s, m) := init(π');
    ̂s := s;
    while not terminate(π', s, m) do
        (s, m) := step(π', s, m);
        if f(π', s) < f(π', ̂s) then
            ̂s := s;
        end
    end
    if ̂s ∈ S'(π') then
        return ̂s
    else
        return ∅
    end
end SLS-Minimisation
```
Continued:

- **neighborhood (set)** of candidate solution $s$:
  \[ N(s) := \{ s' \in S \mid N(s, s') \} \]

- **neighborhood graph** of problem instance $\pi$:
  \[ G_N(\pi) := (S(\pi), N(\pi)) \]
  - We will discuss more of “Fitness Landscape” in next two lectures

- **k-exchange neighbourhood**: candidate solutions $s$ and $s'$ are called neighbours iff $s$ differs from $s'$ in at most $k$ solution components
  - 2-exchange neighbourhood for TSP
    (solution components = edges in given graph)
Search steps in the 2-exchange neighbourhood for the TSP
Continued:

- **search step** (or move): Pair of search positions $s, s'$ for which $s'$ can be reached from $s$ in one step, i.e., $N(s, s')$ and $\text{step}(s, m)(s', m') > 0$ for some memory states $m, m' \in M$.

- **search trajectory**: Finite sequence of search positions $(s_0, s_1, ..., s_k)$ such that $(s_{i-1}, s_i)$ is a search step for any $i \in \{1, ..., k\}$.
  - We will see more about animation of search trajectory that I did during my PhD days in the next two lectures.

- **search strategy**: Specified by init and step function; to some extent independent of problem instance and other components of SLS algorithm.
Continued:

– **Evaluation function** $g(\pi) : S(\pi) \rightarrow \mathbb{R}$ that maps candidate solutions of a given problem instance $\pi$ onto real numbers, such that global optima correspond to solutions of $\pi$;
  - used for ranking or assessing neighbors of current search position to provide guidance to search process.

– **Evaluation versus objective functions**:
  - Evaluation function: Part of SLS algorithm.
  - Objective function: Integral part of optimization problem.
Hill-Climbing for (M/G-NR-)TSP

Also known as **Iterative Improvement/Descent**

- **search space S**: set of all possible TSP tours
- **solution set S'**: set of TSP tours of minimum length
- **neighbourhood relation N**: 2-exchange neighbourhood
- **set of memory states M**: \{0\}, not used
- **init**: classic greedy nearest neighbour heuristic
- **step**: uniform random choice from improving neighbors, i.e., \( \text{step}(s)(s') := \frac{1}{|I(s)|} \) if \( s' \in I(s) \), and 0 otherwise, where \( I(s) := \{s' \in S \mid N(s, s') \text{ and } g(s') < g(s)\} \)
- **terminates** when no improving neighbor available
Intermezzo: Experiments (1/2)
Incremental updates (aka delta evaluations)

- **Key idea**: Calculate effects of differences between the current search position $s$ and its neighbours $s'$ on evaluation function value.

- Evaluation function values often consist of *independent contributions of solution components*; hence, $g(s)$ can be efficiently calculated from $g(s')$ by differences between $s$ and $s'$ in terms of solution components.
  - That is, we do not re-compute everything from scratch

- Typically crucial for the efficient implementation of various SLS algorithms.
Example: Incremental updates for TSP

- solution components = edges of a given graph $G$

- standard 2-exchange neighbourhood, i.e., neighbouring round trips $p$ and $p'$ differ only in two edges

- $w(p') = w(p)$
  - 2 edges in $p$ but not in $p$
  + 2 edges in $p'$ but not in $p$

- This can be done in Constant time (i.e. 4 arithmetic operations), compared to Linear time (i.e. $n$ arithmetic operations for graph with $n$ vertices) for computing $w(p')$ from scratch.
Continued:

- **Local minimum**: Search position without improving neighbours w.r.t. given evaluation function $g$ and neighbourhood $N$, i.e., position $s \in S$ such that $g(s) \leq g(s')$ for all $s' \in N(s)$.

- **Strict local minimum**: Search position $s \in S$ such that $g(s) < g(s')$ for all $s' \in N(s)$.

- **Local maximum** and **strict local maximum** are defined analogously.

- **Local minimum/maximum** is also called as local optima.

- What we want: **Global optima**
SLS Ideas: Escaping Local Optima

Main Problem of simple Hill-Climbing:

- (Quick) stagnation in local optima of evaluation function $g$.

So, some simple mechanisms to improve it:

- **Restart**: Re-initialize search whenever a local optima is encountered.
  - Often rather ineffective due to cost of initialization.

- **Non-improving steps**: In local optima, allow selection of candidate solutions with *equal* or *worse* evaluation function value, e.g., using minimally worsening steps.
  - Can lead to long walks in plateaus, i.e., regions of search positions with identical evaluation function.

- Neither of these mechanisms is guaranteed to always escape effectively from local optima.
SLS Ideas: Search Strategy

Diversification vs Intensification

- Goal-directed and randomized components of SLS strategy need to be balanced carefully.
- **Intensification**: Aims to greedily increase solution quality or probability, e.g., by exploiting the evaluation function.
- **Diversification**: Aims to prevent search stagnation by preventing search from getting trapped in confined regions.
- **Examples**:
  - Iterative Improvement (II): intensification strategy.
  - Uninformed Random Walk (URW): diversification strategy.
- Balanced combination of intensification and diversification mechanisms forms the basis for advanced SLS methods.
Note about Local Optima

Note:

- Local minima depend on $g$ and neighborhood relation $N$.
- Larger neighborhoods $N(s)$ induce:
  - Neighborhood graphs with smaller diameter,
  - Fewer local minima.
- Ideal case is the exact neighborhood, i.e., neighborhood relation for which any local optimum is also guaranteed to be a global optimum.
  - Typically, exact neighborhoods are too large to be searched effectively (exponential in size of problem instance).
We face a trade-off situation here:

- Using larger neighborhoods can improve performance of Hill-Climbing (and other SLS methods).
  - Example: 2-exchange neighborhood to 3-exchange neighborhood :O
- But the time required for determining improving search steps increases (sometimes significantly) with neighborhood size.
- So we have to decide if the effectiveness of larger neighborhoods worth the additional time complexity of search steps.
Neighborhood Pruning:

- **Idea**: Reduce size of neighborhoods by excluding neighbors that are likely/guaranteed not to yield improvements in \( g \).
- **Note**: Crucial for large neighborhoods, but can be also very useful for small neighborhoods.
- **Example**: *Candidate lists* for the TSP
  - Problem intuition: High-quality solutions likely include short edges.
  - Candidate list of vertex \( v \): list of \( v \)'s nearest neighbours (limited number), sorted according to increasing edge weights.
  - Search steps (e.g., 2-exchange moves) always involve edges to elements of candidate lists.
  - Significant impact on performance of SLS algorithms for the TSP.
SLS Ideas: Pivoting Rules

How to choose improving neighbor in each step?

- **Best Improvement** (a.k.a. gradient descent, greedy Hill-Climbing): Choose maximally improving neighbor, i.e., randomly select from \( I^*(s) := \{ s' \in N(s) \mid g(s') = g^* \} \), where \( g^* := \min\{ g(s') \mid s' \in N(s) \} \).
  - Notice that this requires evaluation of all neighbors in each step.

- **Alternative:** **First Improvement**: Evaluate neighbors in fixed order, choose the first improving step encountered.
  - Note: Can be much faster than Best Improvement,
  - Overall quality may be weaker overall (but can also be better due to faster evaluation time per iteration on fixed time limit),
  - Order of evaluation can have significant impact on performance.
Recall: Local minima are relative to neighborhood.

- Key idea: To escape from local minima of a given neighborhood relation, we can switch to a different neighborhood relation.
- Use $k$ neighborhood relations $N_1, N_2, \ldots, N_k$, (typically) ordered according to increasing neighborhood size.
- Always use smallest neighborhood that facilitates improving steps.
- Upon termination, candidate solution is locally optimal w.r.t. all neighborhoods
SLS Time Complexity

• (Very) hard to analyze
• Usually $O(\#\text{iterations} \times \text{polynomial\_cost\_per\_iteration})$
  – But if we use techniques like variable neighborhood, the cost per iteration can be different :S...
• Others just set execution time limit and just run the SLS until the execution time limit has elapsed
  – Like in our experiment so far...
Some More Experiments (2/2)
Summary

- Introducing a new search paradigm: Stochastic Local Search (SLS)
- SLS Definitions
- Hill-Climbing SLS on an example NP-hard COP: The M/G-NR-TSP
- Various SLS Ideas
  - No proof, all “heuristics” :O...
- (Most) ideas are experimented directly on a certain M/G-NR-TSP problem