Metaheuristic Optimization: Local Search (Hill Climbing)

Adaptive and Cooperative Algorithms (ECE 457A)

ECE, MME, and MSCI Departments,
University of Waterloo, ON, Canada

Course Instructor: Benyamin Ghojogh
Fall 2023

Local Search (Hill Climbing)

Local Search (Hill Climbing)

- In local search, also called hill climbing algorithm, we get some neighborhood and we search for the best fitness value among different points in the neighborhood.
- We can consider a neighborhood around the current solution and either evaluate all points in the neighborhood or sample from the neighborhood.

Algorithm Local search

Initialize the solution $\boldsymbol{x}$
while *not converged* **do**
    Generate neighborhood $\mathcal{N}(\boldsymbol{x})$
    for *points in* $\mathcal{N}(\boldsymbol{x})$ **do**
        Evaluate fitness function
        if *better fitness* **then**
            Update the solution
            Optional: Break the inner loop
Return the solution $\boldsymbol{x}$

Improving Local Search

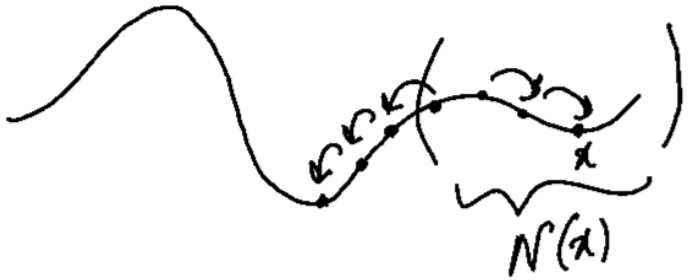
Local search is prone to get stuck in local best.
We can improve local search in various ways:

- Iterative with different initializations
  - Multi-start local search
  - Iterative local search (ILS)
- Change in objective function (landscape) for simplifying the problem
  - Guided local search (GLS)
  - Smoothing method
  - Noisy method (jitter)
- Using different neighborhoods
  - Variable Neighborhood Search (VNS)
  - Generalized Neighborhood Search (GNS)
- Accepting non-improving neighbors: such as simulated annealing (it will be explained later)

Iterative with different initializations

Multi-start local search

- In multi-start local search, after finding the local best, we sample a point in some neighborhood of the solution and restart the local search algorithm from that point.
- We can do this for several times and select the best solutions among them.

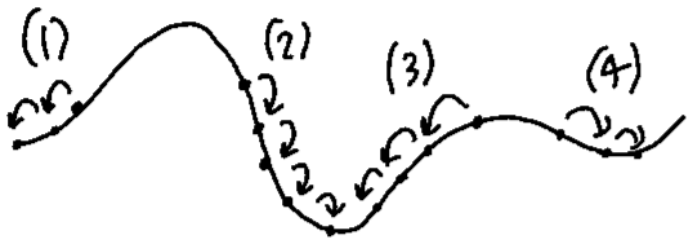


Algorithm Multi-start local search

solutions = []
Initialize the solution $\boldsymbol{x}$
for *multiple times* **do**
    $\boldsymbol{x} \leftarrow$ Local-search($\boldsymbol{x}$)
    solutions.append($\boldsymbol{x}$)
    $\boldsymbol{x} \leftarrow$ sample from $\mathcal{N}(\boldsymbol{x})$
Return best among solutions

Iterative local search (ILS)

- The multi-start local search may still get stuck in the same local best every time.
- In Iterative local search (ILS) [1, 2], after finding the local best, we sample a point, not in some neighborhood of the solution, but randomly from somewhere in the landscape. Then, we restart the local search algorithm from that point.
- This lets local search start from various points in the landscape.
- We can do this for several times and select the best solutions among them.



Algorithm Iterative local search (ILS)

```
solutions = []
Initialize the solution x
for multiple times do
    x ← Local-search(x)
    solutions.append(x)
    x ← sample from the optimization landscape
Return best among solutions
```

Change in objective function

Guided local search (GLS)

- Guided local search (GLS) (1997) [3] modifies the landscape gradually to simplify the landscape so local search does not get stuck in the local bests.
- GLS does not work on all problems. For using it, the features should be represented as binary (existing or not existing).
- If it is not binary, we may be able to convert it to a binary string either by thresholding each feature or converting the float values to binary representation.
- Assume in some iteration of the local search, the solution is $\boldsymbol{x} = [x_1, \ldots, x_d]^\top \in [0,1]^d$. We define:

$$I_j(\boldsymbol{x}) := \begin{cases} 1 & \text{if } x_j = 1 \text{ (if } x_i \text{ contributes to the solution)} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

- In GLS, every feature x_j should also have some cost c_j. For example, the overall cost can be the summation of c_j's.
- We also define the utility function for a solution $\boldsymbol{x}$:

$$U_j(\boldsymbol{x}) = I_j(\boldsymbol{x}) \times \frac{c_j}{1 + p_j}, \tag{2}$$

where $p_j, \forall j \in \{1, \ldots, d\}$ are the penalties initialized to zero. Every time, we see which of $\{U_j(\boldsymbol{x})\}_{j=1}^d$ is the largest. The penalty p_j is incremented by one if $U_j(\boldsymbol{x})$ is the largest utility.

Guided local search (GLS)

- GLS penalizes the features which exist in the minimum solution of optimization but have highest value of cost!
- However, it does not desire to penalize the same feature ever and ever again. So, every time it penalizes a feature, it reduces its change to be penalized again by putting p_j in the denominator of its utility.
- Then, after incrementing the penalty, GLS modifies the cost function by regularizing it:

$$c'(\boldsymbol{x}) = c(\boldsymbol{x}) + \lambda \sum_{j=1}^{d} p_j I_j(\boldsymbol{x}), \tag{3}$$

  where $\lambda > 0$ is the regularization parameter.
- This regularized cost function finds local minimums and penalize them to become peaks rather than valleys. Then, the cost function becomes smoother and easier for the global minimum to be found.

Guided local search (GLS)

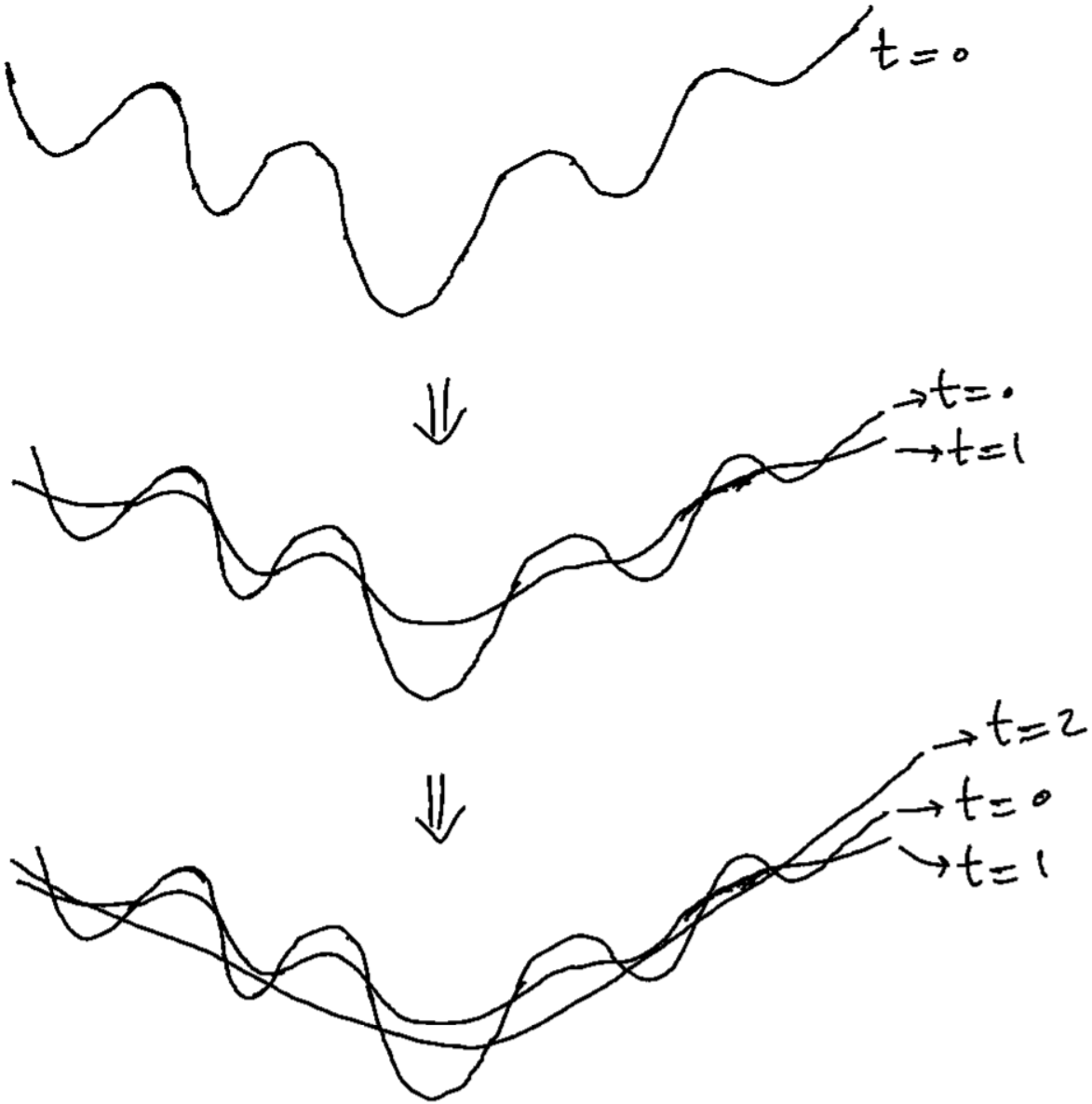
Algorithm Guided local search (GLS)

solutions $=$ []
Initialize the solution $\boldsymbol{x}$ as a binary vector
while *not converged* **do**
    $\boldsymbol{x} \leftarrow$ Local-search$(c(\boldsymbol{x}), \boldsymbol{x})$
    solutions.append$(\boldsymbol{x})$
    for $j \in \{1, \ldots, d\}$ **do**
        $I_j(\boldsymbol{x}) := \begin{cases} 1 & \text{if } x_j = 1 \text{ (if } x_i \text{ contributes to the solution)} \\ 0 & \text{otherwise.} \end{cases}$
        $U_j(\boldsymbol{x}) = I_j(\boldsymbol{x}) \times \frac{c_j}{1+p_j}$
        $k = \arg\max\{U_1(\boldsymbol{x}), \ldots, U_d(\boldsymbol{x})\}$
        $p_k = p_k + 1$
    $c(\boldsymbol{x}) \leftarrow c(\boldsymbol{x}) + \lambda \sum_{j=1}^{d} p_j I_j(\boldsymbol{x})$
Return best among solutions

Smoothing method

- We can iteratively and gradually smooth the objective function.
- In this way, the local bests of the object function gradually disappear so that we can find the global bests more easily. In initial iterations, if we get stuck in a local best, we will gradually get out of it to be able to find the global best.
- We can use any method for smoothing the function. For example, we can use a window averaging. In this approach, we make a grid with fine steps in the domain of function and for every point in the grid, the cost/fitness at the point is replaced by the cost/fitness of its neighboring points in some window in the grid.
- The size of the window is a hyperparameter determined at every iteration. The larger the window size is, the smoother function becomes.

Smoothing method



Smoothing method

Algorithm Smooth-the-objective

Input: landscape $f(.)$, window size s
make a grid with some fine step in the domain of landscape
Initialize the smoothed landscape $g(.)$ with $f(.)$
for *each point* $\boldsymbol{x}$ *in the grid* **do**
    sum $\leftarrow$ 0, neighbors_count $\leftarrow$ 0
    for *each neighbor* $\boldsymbol{y}$ *in grid in window of size* s *around* $\boldsymbol{x}$ **do**
        sum $\leftarrow$ sum $+ f(\boldsymbol{y})$
    $g(\boldsymbol{x}) \leftarrow$ sum / *neighbors_count*
Return $g(\boldsymbol{x})$

Algorithm Local search with the smoothing method

solutions = []
Initialize the solution $\boldsymbol{x}$ and the window size s
for *multiple times* **do**
    $\boldsymbol{x} \leftarrow$ Local-search($\boldsymbol{x}$) on landscape
    solutions.append($\boldsymbol{x}$)
    landscape $\leftarrow$ Smooth-the-objective(landscape, s)
Return best among solutions

Noisy method (jitter)

- In every iteration of the local search, we can add some noise (also called jitter) for the initialization of the next local search.
- Therefore, if we have got stuck in a local best, we can get out of it.
- The more the jitter is, the more exploration we have compared to exploitation.
- Optionally, we can start with large jitter initially and gradually we decrease the amount of jitter because initially we should explore the landscape but at the end, we are supposed to be closer the actual solution.

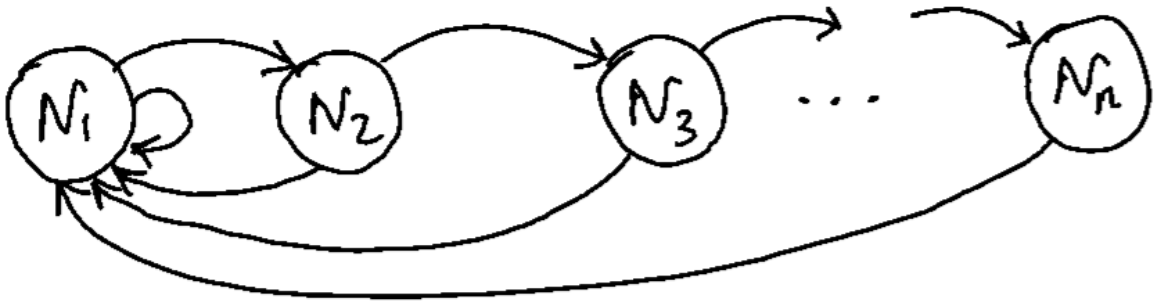
Algorithm Local search with jitter

solutions $=$ []
Initialize the solution $\boldsymbol{x}$ and the jitter's standard deviation σ
for *multiple times* **do**
    $\boldsymbol{x} \leftarrow$ Local-search($\boldsymbol{x}$)
    solutions.append($\boldsymbol{x}$)
    $\sigma \leftarrow$ decrement(σ)
    $\epsilon \leftarrow$ Gaussian$(0, \sigma^2 \boldsymbol{I})$
    $\boldsymbol{x} \leftarrow$ sample from $\mathcal{N}(\boldsymbol{x} + \epsilon)$
Return best among solutions

Using different neighborhoods

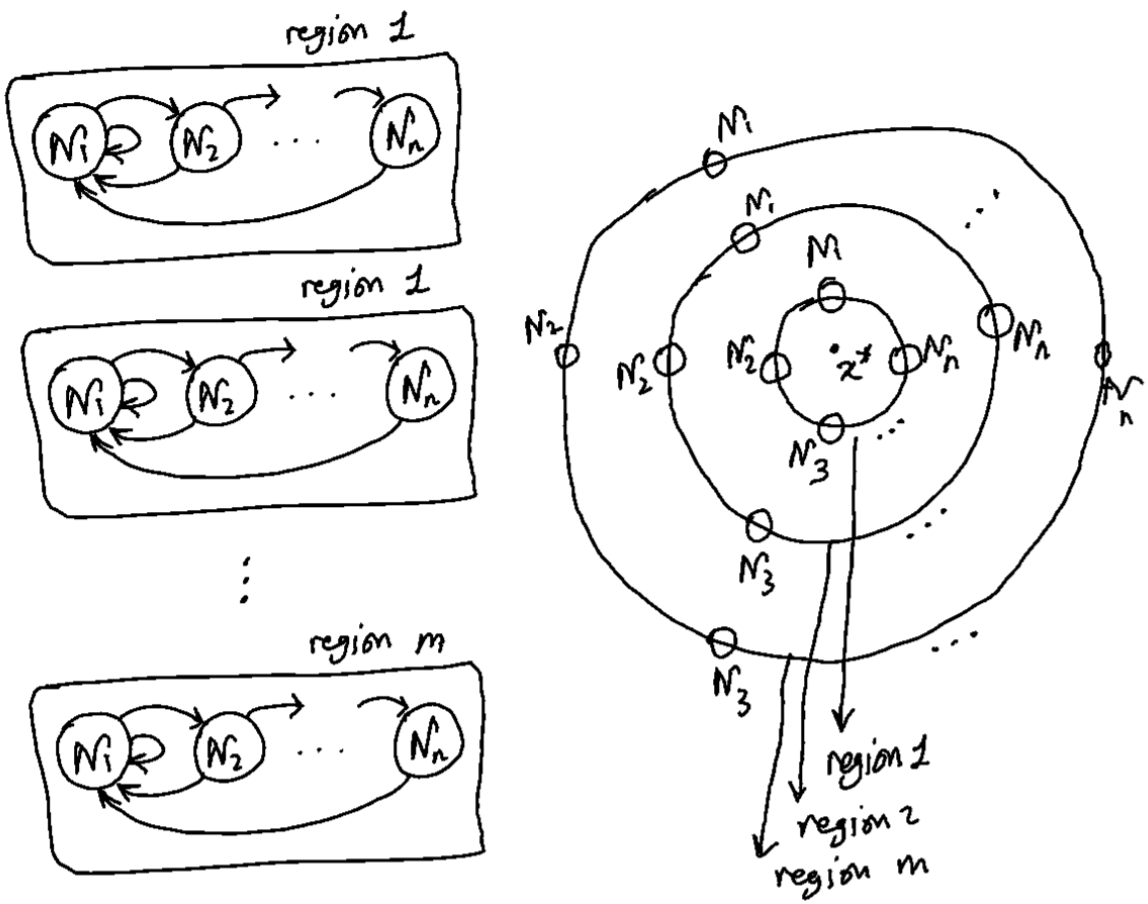
Variable Neighborhood Search (VNS)

- Variable Neighborhood Search (VNS) (2010) [4] performs local search as a finite state machine [5].
- In VSN, we select several neighborhoods $\{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$.
- First, we perform local search in the optimization landscape.
- We do local search in the area $\mathcal{N}_1$. We check if a better solution is found in this area. If yes, we update the best solution, then we go back to this area $\mathcal{N}_1$ and search it more. If not, we go to the next neighborhood $\mathcal{N}_2$ and search it.
- We check if a better solution is found in the area $\mathcal{N}_2$. If yes, we update the best solution, then we go back to this area $\mathcal{N}_1$ and start searching from there again. If not, we go to the next neighborhood $\mathcal{N}_3$ and search it.



Generalized Neighborhood Search (GNS)

- Generalized Neighborhood Search (GNS) (2015) [6] performs VNS in parallel in several regions of the landscape and then returns the best among solutions of the VNS regions as the solution.
- Usually, the regions of neighborhoods are chosen layer-wise as shown in this figure.



Acknowledgment

- Some slides of this slide deck are inspired by teachings of Prof. Saeed Sharifian at the Amirkabir University of Technology, Department of Electrical Engineering.

References

[1] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of metaheuristics*, pp. 320–353, Springer, 2003.

[2] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," *Handbook of metaheuristics*, pp. 129–168, 2019.

[3] C. Voudouris, *Guided local search for combinatorial optimisation problems*. PhD thesis, University of Essex, 1997.

[4] P. Hansen, N. Mladenović, and J. A. Moreno Perez, "Variable neighbourhood search: methods and applications," *Annals of Operations Research*, vol. 175, pp. 367–407, 2010.

[5] J. Wang, *Handbook of finite state based models and applications*. CRC press, 2012.

[6] N. Bouhmala, K. Hjelmervik, and K. I. Øvergaard, "A generalized variable neighborhood search for combinatorial optimization problems," *Electronic Notes in Discrete Mathematics*, vol. 47, pp. 45–52, 2015.