Local search and random search

Juan Antonio Pérez Juárez
University of Guadalajara
Guadalajara, Jalisco
juan.perez0996@alumnos.udg.mx

I. INTRODUCCIÓN

Optimization is a fundamental discipline in computational sciences and applied mathematics, playing a key role in the development of efficient algorithms for solving complex problems. Among the most studied techniques are local search and random search, methods that effectively explore the solution space through different strategies. This article analyzes and compares the performance of both approaches using the Sphere function, defined as:

$$f(x, y) = x^2 + y^2$$

Which is widely employed as a benchmark in optimization due to its simplicity and well-known properties. The main objective is to illustrate the characteristics, advantages, and limitations of local search and random search, providing a basis for understanding their applicability to real-world problems.

II. OPTIMIZATION

In computer science and mathematical optimization, a metaheuristic is a higher-level procedure or heuristic designed to find, generate, tune, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem or a machine learning problem, especially with incomplete or imperfect information or limited computation capacity.
Metaheuristics sample a subset of solutions which is otherwise too large to be completely enumerated or otherwise explored. Metaheuristics may make relatively few assumptions about the optimization problem being solved and so may be usable for a variety of problems.

III. LOCAL SEARCH

In computer science, local search is a heuristic method for solving computationally hard optimization problems. Local search can be used on problems that can be formulated by finding a solution that maximizes a criterion among several candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

Local search algorithms are widely applied to numerous hard computational problems, including problems from computer science (particularly artificial intelligence), mathematics, operations research, engineering, and bioinformatics. Examples of local search algorithms are WalkSAT, the 2-opt algorithm for the Traveling Salesman Problem and the Metropolis–Hastings algorithm.
Most problems can be formulated in terms of search space and target in several different ways. For example, for the traveling salesman problem a solution can be a route visiting all cities and the goal is to find the shortest route. But a solution can also be a path and being a cycle is part of the target.

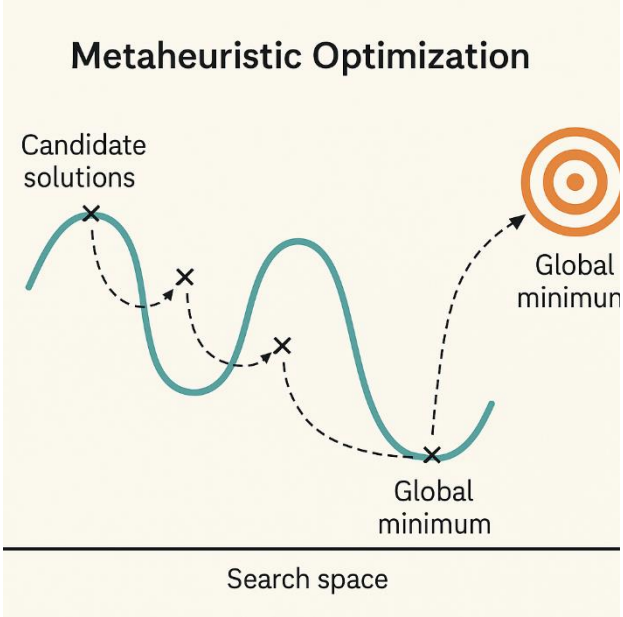

Fig. 1 Illustration of metaheuristic optimization in a search space. Candidate solutions explore the landscape to identify the global minimum, overcoming local minimum through global search strategies.

A local search algorithm starts from a candidate solution and then iteratively moves to a neighboring solution; a neighborhood being the set of all potential solutions that differ from the current solution by the minimal possible extent. This requires a neighborhood relation to be defined on the search space. As an example, the neighborhood of vertex cover is another vertex cover only differing by one node. For Boolean satisfiability, the neighbors of a Boolean assignment are those that have a single variable in an opposite state. The same problem may have multiple distinct neighborhoods defined on it; local optimization with neighborhoods that involve changing up to k components of the solution is often referred to as k-opt.

Typically, every candidate solution has more than one neighbor solution; the choice of which one to select is taken using only information about the solutions in the neighborhood of the current assignment, hence the name local search. When the choice of the neighbor solution is done by taking the one locally maximizing the criterion, a

greedy search, the metaheuristic takes the name hill climbing. When no improving neighbors are present, local search is stuck at a locally optimal point. This local-optima problem can be cured by using restarts (repeated local search with different initial conditions), randomization, or more complex schemes based on iterations, like iterated local search, on memory, like reactive search optimization, on memory-less stochastic modifications, like simulated annealing.

Local search does not provide a guarantee that any given solution is optimal. The search can terminate after a given time bound or when the best solution found thus far has not improved in each number of steps. Local search is an anytime algorithm; it can return a valid solution even if it's interrupted at any time after finding the first valid solution. Local search is typically an approximation or incomplete algorithm because the search may stop even if the current best solution found is not optimal. This can happen even if termination happens because the current best solution could not be improved, as the optimal solution can lie far from the neighborhood of the solutions crossed by the algorithm.

A. Advantages

1) Simplicity: Local Search algorithms are straightforward to implement and require minimal parameter tuning.

2) Efficiency: These methods can quickly converge to a solution, especially in problems where the global optimum is near the initial candidate.

3) Low Memory Usage: Local Search typically requires only a small amount of memory, as it focuses on the current solution and its immediate neighbors.

4) Effective for Smooth Landscapes: Local Search performs well in search spaces with a single optimum or few local minima.

B. Disadvantages

1) Susceptibility to Local Minima: Local Search can easily become trapped in local optima, failing to find the global optimum in complex landscapes.

2) Limited Exploration: The algorithm explores only the neighborhood of the current solution, which may result in poor coverage of the overall search space.

3) Dependence on Initial Solution: The quality of the final solution can be highly dependent on the starting point.

4) Not Suitable for Highly Multimodal Functions: In problems with many local minima, Local Search may perform poorly compared to global search methods.

IV. RANDOM SEARCH

The random search strategy consists of sampling solutions over the entire search space using a uniform probability distribution. Each future sample is independent of the samples that precede it.

Pseudocode for Random Search.

```
Input: NumIterations, ProblemSize, SearchSpace
Output: Best
Best ← ∅;
foreach iter_i ∈ NumIterations do
    candidate_i ← RandomSolution(ProblemSize, SearchSpace);
    if Cost(candidate_i) < Cost(Best) then
        Best ← candidate_i;
    end
end
return Best;
```

Fig. 2 Pseudocode for the Random Search algorithm. The procedure iteratively generates candidate solutions at random within the defined search space. For each iteration, the cost of the newly generated candidate is evaluated and compared with the current best solution. If the candidate exhibits a lower cost, it replaces the current best solution. After all iterations, the algorithm returns the best solution found.

The strategy has a complex time and minimal memory, as it requires only a candidate solution construction routine and a candidate solution evaluation routine, both of which can be calibrated using the approach.

The worst performance for locating optima is worse than a search domain enumeration, since the random search has no memory and can perform blind resampling.

Random search can return a reasonable approximation of the optimal solution within a reasonable time frame with low problem dimensionality, although the approach does not scale well to the size of the problem (such as the number of dimensions).

The results can be used as the basis for another research technique, such as a search technique. local search (such as the Hill Climbing algorithm), which can be used to locate the best solution in the vicinity of the good candidate solution.

```python
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
# define hyperparameter space
param_dist = {
    'n_estimators': [10, 100, 1000],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 10],
    'min_samples_split': [2, 10, 100],
    'min_samples_leaf': [1, 10, 100]
}
# define model
model = RandomForestClassifier()
# define search algorithm
search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=100)
# fit search algorithm
search.fit(X_train, y_train)
```

Fig. 3 Implementing random search in python.

A. Advantages

1) Efficiency: Random search is often more efficient than grid search because it explores the hyperparameter space more efficiently. This is because it does not waste time exploring areas of the space that are unlikely to be optimal.

2) Better Performance: Random search is also less likely to get stuck in local minima than grid search. This

is because it explores a wider range of hyperparameters, making it more likely to find the global minimum.

3) Less Sensitive to Noise: Random search is less

B. Disadvantages

1) More Computationally Intensive: Random search can be more computationally intensive than other optimization algorithms because it requires more random samples to achieve the same level of accuracy.

2) No Guarantees: Random search does not provide any guarantees about the quality of the solution. It only finds the best solution that it discovers through random sampling.

3) Requires Tuning: Random search requires tuning of the hyperparameters of the search algorithm itself, such as the number of random samples and the distribution from which the hyperparameters are sampled.

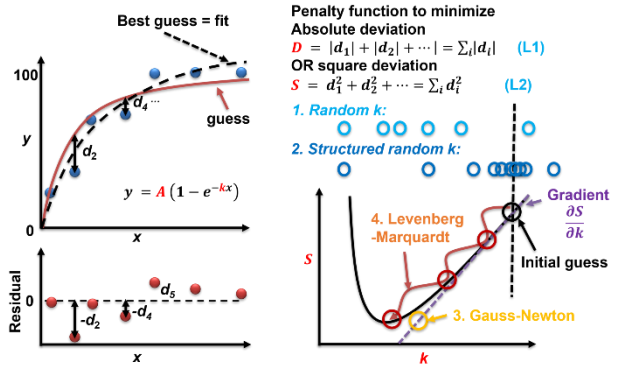

Fig. 4 Scheme of random search using a non-linear regression problem as an example. The goal is to minimize the value of the penalty function. The right bottom shows a few example methods: 1. Non-structured random search, 2. structured random search, 3. Gauss-Newton algorithm, and 4. Levenberg-Marquardt algorithm. 1,2 do not need to know the gradient and 3,4 have to calculate the gradient and usually minimize on both A and k parameters at the same time.

V. DIFFERENCES

A. Exploration Strategy

1) In Local Search

Starts from an initial solution and explores its local neighborhood by making small, incremental changes.

2) In Random Search

Samples solutions randomly from the entire search space without regard to previous solutions or their neighborhoods.

B. Exploitation vs. Exploration

1) Local Search

Focuses on exploiting local information to iteratively improve the solution.

2) In Random Search

sensitive to noisy evaluations of the objective function than other optimization algorithms because it averages over many random samples.

Emphasizes global exploration, with no mechanism for exploiting previously found good solutions.

C. Risk of Local Optima

1) In Local Search

Prone to getting trapped in local optima due to its neighborhood-based search.

2) In Random Search

Less likely to get stuck in local optima because it samples the whole search space.

D. Dependence on Initial Solution.

1) In local search: The quality of the final solution is often influenced by the starting point.

2) In Random search: Independent of any initial solution, as each candidate is generated randomly.

VI. CONCLUSIONS

In computer science it's important to have in mind all the possible solutions and always look for other kinds of solutions.

That's why metaheuristics algorithms look for the most efficiency and accurate answer.

The comprehension of this topic is fundamental for the undertaking of Artificial Intelligence.

This could be breathtaking and hard to swallow. But that is the importance of this class.

We, as humanity, are closer to the next singularity, and I want to stay as close as it possible.

REFERENCIAS

[1] Robert Sedgewick, Algorithms, 1984, p. 84.

[2] Antoniou, Andreas; Lu, Wu-Sheng (2021). Practical Optimization (PDF). Texts in Computer Science (2nd ed.). Springer. p. 1. doi:10.1007/978-1-0716-0843-2. ISBN 978-1-0716-0841-8.

[3] S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, "A novel

ultrathin elevated channel low-temperature poly-Si TFT," IEEE

Electron Device

Lett., vol. 20, pp. 569–571, Nov. 1999.

[4] Glover, F.; Kochenberger, G.A. (2003). Handbook of metaheuristics. Vol. 57. Springer, International Series in Operations Research & Management Science. ISBN 978-1-4020-7263-5.

[5] "What is Random search | Ai Basics | Ai Online Course," www.aionlinecourse.com. https://www.aionlinecourse.com/ai-basics/random-search

[6} Russell, S. J., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd ed.).