Review on greedy algorithm

Yizhun Wang

School of Chemistry and Chemical Engineering, Southeast University, Nanjing, JiangSu Province, China, 210000

2528073342@qq.com

Abstract: The greedy algorithm is a commonly used algorithm design idea that can provide efficient solutions to many practical problems. This paper aims to review and summarize the basic ideas, characteristics and application fields of greedy algorithms, and discuss their advantages and limitations. Firstly, the basic concepts of greedy algorithms are introduced, including the greedy selection properties and optimal substructures. Then, some classic greedy algorithms such as the backpack problem, the activity selection problem, and the minimum spanning tree problem are introduced, and the concept of time complexity is introduced. Next, the application of greedy algorithms in practical problems, such as scheduling problems, network routing, and graph generation, will be discussed. Finally, the advantages of the greedy algorithm and the limitation of the inability to obtain the global optimal solution will be evaluated, and the improvement direction combined with other algorithms will be proposed.

Keywords: Greedy Algorithm, Optimal Substructure, Backpack Problem, Minimum Spanning Tree, Advantages and Limitations.

1. Introduction

The greedy algorithm can be traced back to the early Dijkstra algorithm [1]. The greedy algorithm was proposed in 1959 by Dutch computer scientist Edsger Wyb Dijstr to solve the unit shortest path problem. It has the property of greed, that is, at each step, it takes the local optimal solution in the current state according to the characteristics of the problem. The core of this strategy is to derive the global optimal solution from the local optimal solution. The main advantages of greedy algorithms are simplicity, efficiency, and ease of implementation. It typically does not require sorting input data or calculating complex data structures, so it has a low time complexity. This review will introduce in detail the basic concepts and principles of greedy algorithms and analyze several classic greedy algorithms. Then, the application of greedy algorithms to practical problems will be explored, and its advantages and limitations will be discussed. Finally, some possible improvements and expansion directions will be proposed to further improve the application effect and performance of the greedy algorithm. The greedy algorithm has strong practicality and efficiency, but there are also some limitations and flaws. This review paper will reveal its theoretical and practical challenges, and put forward unsolved problems and research directions, which will help promote the in-depth research of greedy algorithms, find new application scenarios and improve methods. In addition, this review will summarize the practical experience of the application of greedy algorithms, and provide guidance and suggestions for solving problems using greedy algorithms in various fields. This makes sense for engineers and practitioners to better apply greedy algorithms to solve real-world problems.

2. The principle of the greedy algorithm

2.1. Greedy choice nature

The nature of greedy selection means that when faced with a problem and needs to make a decision, the greedy algorithm will give preference to the solution that currently seems optimal, regardless of the global optimal solution. The key to greedy choice is that the choice at each step must be locally optimal, even if this choice may result in the final result not necessarily being globally optimal.

Greedy selection is generally suitable for satisfying problems with specific properties that have optimal substructure properties and that produce optimal solutions. The optimal substructure property means that the overall optimal solution to the problem can be achieved by a series of local optimal choices.

The selection process in a greedy algorithm typically consists of the following steps:

(1). Start with the initial solution to the problem (empty solution).

(2). The current solution is gradually extended through a series of local optimal choices.

(3). For each step, the choice must satisfy the greedy selection nature, that is, the current choice must be the solution that currently seems optimal.

(4). Repeat steps (2) and (3) until the termination condition of the problem is reached.

An important feature of greedy selection is that it does not backtrack, i.e. once a choice is made, it does not change. This is because greedy algorithms are usually based on local optimal selection, focusing only on the optimal solution of the current step, and do not consider future choices and influences [2].

2.2. Optimal substructure

An optimal substructure is a problem-solving idea that is commonly used in dynamic programming algorithms and greedy algorithms. An optimal substructure is one in which the optimal solution of a problem contains the optimal solution of its subproblems.

Specifically, the property that a problem has an optimal substructure means that the optimal solution of the problem can be constructed from the optimal solution of the subproblem. That is, the optimal solution of the problem can be obtained by selecting the optimal solution to a subproblem and then doing further operations according to some rules or algorithms.

The nature of optimal substructure enables users to solve problems more efficiently. When a user faces a complex problem, it can be divided into multiple interrelated sub-problems. By solving the subproblem and taking advantage of the properties of the optimal substructure, users can solve it step by step from simple to complex, and finally obtain the optimal solution of the entire problem.

Take an example to illustrate the concept of optimal substructure. Suppose there is a problem: in an array of length n, find the length of the longest increasing subsequence. The optimal substructure here can be understood as the length of the longest ascending subsequence of the current array, which can be obtained by multiplying the length of the longest ascending subsequence of the subarray. We can compare the size of the current element with the previous element to determine whether the current element can be added to the subsequence, so as to obtain the length of the longest increasing subsequence.

The optimal substructure is an important problem-solving idea that provides us with a way to disassemble and solve complex problems. By rationally designing subproblem solving strategies and taking advantage of the properties of optimal substructures, we can efficiently solve many practical problems.

3. Classic greedy algorithm

3.1. Backpack problems

The backpack problem is a classic combinatorial optimization problem, and the greedy algorithm is a common method for solving the backpack problem.

The 01 backpack problem is the most basic backpack problem, which requires that in the given weight and value of the item, select some items to put in the backpack, so that the total weight of the items in the backpack does not exceed the capacity of the backpack, and at the same time make the total value of the items in the backpack the maximum. The application of the greedy algorithm in the 01 backpack problem is to select the lightest weight item to put in the backpack at a time until the backpack's capacity reaches the upper limit.

In the backpack problem, we have a backpack and a set of items, each with its own weight and value. The backpack can only carry a certain amount of weight, and the goal is to select some items to fit into the backpack so that the total value of the items in the backpack is maximized. The idea of the greedy algorithm is to select the current optimal item to put in the backpack each time until it can no longer be put in.

For example, suppose there are 4 items with a weight of [1, 2, 2, 3], corresponding values of [10, 12, 15, 20], and a backpack’s capacity is 5.

Table 1. The processing of greedy algorithm.

Item	Weight	Value
Item1	2	12
Item2	1	10
Item3	3	20
Item4	2	15

The greedy algorithm will first select items weighing 1 to put in the backpack, and the remaining backpack capacity is 4. Then select the item with a weight of 2 to put in the backpack, and the remaining backpack capacity is 2. Finally choose another item weighing 2 to put in the backpack, which has a backpack capacity of 0. In this way, the total value of the items in the backpack is 10+12+15=37, which reaches the maximum value.

Specifically, the greedy algorithm can solve the backpack problem by following the following steps [3]:

(1). Calculate the unit value (i.e. the ratio of value to weight) of each item.

(2). Sort items by unit value from largest to smallest.

(3). Select the items with the highest unit value to put in your backpack until the backpack can't hold the next item or the items have all been placed in the backpack.

3.2. Activity Selection Issues

The activity selection problem in the greedy algorithm is an optimization problem based on the greedy strategy. In this problem, given the start and end times for a set of activities, the goal of the strategy is to find the maximum number of non-conflicting activity collections.

The basic idea of the greedy algorithm to solve the activity selection problem is to decompose the problem into a series of subproblems, select the optimal subproblem solution each time, and add it to the final solution set.

The specific resolution steps are as follows:

(1). Sort all activities by their end time.

(2). Select the first activity and place it in the solution set.

(3). For the remaining activities, check in turn whether the current activity does not conflict with the selected activity, and if not, select the activity and add it to the decompilation.

(4) Repeat step (3) until all activities have been completed.

The correctness of the greedy algorithm is based on the nature of greedy selection, that is, the activity with the earliest end time of each selection can leave more time for subsequent activity selection.

The greedy algorithm for the activity selection problem can be solved within the time complexity of O(nlogn), where n is the number of activities. This is because the sort operation takes O(nlogn) time.

In conclusion, the activity selection problem in the greedy algorithm is an optimization problem based on the greedy strategy, and the optimal solution set is constructed by selecting the activity with the earliest end time each time [4].

3.3. Minimal spanning tree issues

The minimum spanning tree (MST) problem is an important problem in graph theory, where the goal is to find a spanning tree in a weighted undirected connected graph such that the sum of the weights on the edges of the tree is minimal.

The minimum spanning tree problem has the following characteristics:

(1). The graph must be connected; that is, there must be a path between any two nodes in the graph.

(2). The edges in the graph must have weight, which can be real, integer, or other measurable values.

(3). The minimum spanning tree is a tree diagram, that is, a non-cyclic connection graph.

There are many classical solutions to the minimum spanning tree problem, among which the Prim algorithm and the Kruskal algorithm are commonly used.

The Prim algorithm is a greedy algorithm. Starting with one node, select the edge with the smallest weight connected to the tree to join the tree each time until all nodes are connected to the tree, resulting in the smallest spanning tree [5].

The Kruskal algorithm is also a greedy algorithm, which starts with the edge with the least weight, selects the edge with increasing weight in turn, and guarantees that the selected edge does not form a ring until all nodes are connected to the tree, resulting in the smallest spanning tree. Joseph Kruskal first presented this approach in a paper that was published in 1956 [6].

The minimum spanning tree problem has many applications, such as network design, power delivery, communication networks, and other fields [7].

4. Application of greedy algorithms

4.1. Scheduling issues

The greedy algorithm has a wide range of applications in scheduling problems. A scheduling problem is a problem in which a set of tasks or jobs are assigned to a set of resources or processors to complete all tasks or jobs optimally.

In scheduling problems, greedy algorithms usually use local optimal solutions to construct global optimal solutions. Specifically, the greedy algorithm selects the currently optimal task or job, then allocates it to available resources or processors, and repeats the process until all tasks or jobs are assigned [8].

The application of the greedy algorithm to scheduling problems includes the following aspects:

(1). Skip idle time: The greedy algorithm can assign tasks or jobs as soon as resources are available. This minimizes idle time for resources.

(2). Shortest job priority: The greedy algorithm can select the task or job with the shortest execution time, which can ensure the shortest waiting time for the task or job.

(3). Earliest deadline first: The greedy algorithm can select tasks or jobs with the earliest deadlines, which ensures that tasks or jobs can be completed before the deadline.

(4) Least time-consuming priority: The greedy algorithm can choose the task or job with the shortest execution time, which can minimize the total execution time.

4.2. Traveling Salesman Issues

Traveling salesman problem: The greedy algorithm can also be applied to the traveling salesman problem (TSP) [9]. TSP is a classic combinatorial optimization problem that requires finding the shortest path between a given series of cities, such that each city passes once and finally returns to the starting city. The greedy algorithm can employ two strategies in TSP: the nearest neighbor strategy and the least spanning tree strategy. The nearest neighbor strategy is to start from one starting city and select the closest and least visited city each time as the next city to visit until all cities have been visited. This method is simple and fast, but it does not guarantee an optimal solution. The minimum spanning tree strategy is to build a minimum spanning tree and then traverse the nodes on the tree through DFS (depth-first search) to get a path. This approach also does not guarantee an optimal solution, but is generally better than the nearest neighbor strategy.

5. Advantages and limitations of greedy algorithms

5.1. Advantages

A significant advantage of greedy algorithms is their low complexity.

Since the greedy algorithm only considers the immediate optimal solution and does not consider the global optimal solution, it does not require additional space to store intermediate states or alternative solutions. At runtime, greedy algorithms usually only require a small amount of fixed extra space, such as for storing some local variables or data needed during calculations, so their spatial complexity is low [10].

In addition, in general, the time complexity of the greedy algorithm is low, mainly for the following two reasons:

(1). Based on local optimum: The choice of each step of the greedy algorithm is based on the local optimal solution in the current state, and does not consider global optimality. Therefore, only the local optimal case needs to be considered at each step, and there is no need to traverse all possible solutions. This local optimal selection strategy reduces the amount of computation and makes the greedy algorithm more efficient.

(2). No backtracking: Unlike other search algorithms, greedy algorithms do not backtrack after each step of selection. Once a decision is made, it doesn't change. This saves computation time and avoids double counting on invalid paths.

In addition, the greedy algorithm has the following four advantages:

(1). Simple and easy to understand: The greedy algorithm builds the optimal solution to the whole based on the optimal selection of each step, and the logic is relatively simple, easy to understand and implement.

(2). High efficiency: Since the greedy algorithm only focuses on the optimal solution of the current step and does not consider the global calculation, it can quickly find a good approximate solution to some problems.

(3). Can solve some optimization problems: For some specific types of problems, greedy algorithms can find global optimal solutions, such as some backpack problems, graph theory problems, etc.

(4). Can be used as an optimization strategy for other algorithms: The greedy algorithm can be used as an optimization strategy or auxiliary algorithm for other algorithms to improve the efficiency of other algorithms through the selection of greedy strategies.

5.2. limitations

The main limitation of the greedy algorithm is that the local optimal solution does not necessarily give the global optimal solution.

Although the greedy algorithm selects the current optimal solution at each step, this local optimal strategy does not necessarily lead to the global optimal solution. In some cases, greedy algorithms may fall into a local optimal solution and fail to reach a global optimal solution.

Greedy algorithms are generally applied when the problem has no after-effect (i.e., state decisions at each stage are only relevant to the current state) and an optimal substructure (i.e., the optimal solution of the problem contains the optimal solution of the subproblem). If the problem does not meet these two conditions, the greedy algorithm may not get the correct solution [2].

In addition, the greedy algorithm has the following limitations:

(1). No regret mechanism: Greedy algorithms usually have no mechanism for fallback, and once a choice is made, it cannot be undone. This means that once a wrong choice is made, it may not be corrected, resulting in a distortion of the final choice result.

(2). Need to satisfy the greedy selection nature: The greedy algorithm requires the problem to have the nature of greedy selection, that is, the global optimal solution can be obtained through local greedy selection. However, not all problems have this nature, and greedy algorithms cannot get the right results for problems that cannot meet this nature.

(3). Need to be able to decompose subproblems: Greedy algorithms usually need to break down the original problem into several subproblems to solve. This means that the problem must have a nature that can be broken down into subproblems; otherwise, greedy algorithms cannot be used.

In summary, the greedy algorithm can get better results on some problems, but for complex problems with many constraints, the limitations of the greedy algorithm will lead to its inability to get the correct results. In practical applications, it is necessary to carefully choose whether to use greedy algorithms based on the characteristics and requirements of the problem.

5.3. Optimal solution

The greedy algorithm can be optimized in combination with the following algorithms:

(1). Divide and conquer algorithm: The greedy algorithm can be optimized on the basis of the divide and conquer algorithm, and the appropriate greedy strategy can be selected to make the sub-problem of divide and conquer easier to solve [11].

(2). Dynamic programming: The greedy algorithm can find the candidate solution of the optimal solution in the process of dynamic programming, thereby reducing the state space and computation amount of dynamic programming [12].

(3). Backtracking algorithm: The greedy algorithm can be used as a heuristic search strategy for the backtracking algorithm, and the most likely next step is selected according to the greedy criterion, thereby improving the efficiency of the backtracking algorithm [13].

6. Conclusion

In this review, we introduce the basic concepts of greedy algorithms, their properties, as well as their examples and applications, discuss their limitations, and give other possible strategies to solve problems. Here you need to specify the basic concepts, their nature, examples and applications, limitations, and other possible strategies to solve the problem. However, this article has certain shortcomings. First, there is a lack of comprehensive information, in which the techniques discussed may not cover new algorithms, and the application examples discussed involve only a representative part of greedy algorithms, not comprehensive ones. The second is the limitations of the academic field, this article may be more suitable for beginners or non-professional readers, and for researchers engaged in related professional theories, this article may not be in-depth.

In the future, researchers can further explore the improvement and optimization of greedy algorithms, and have solved more complex problems: clarify when the use of greedy algorithms cannot get the global optimal solution, and avoid the loopholes caused by the use of greedy algorithms.

References

[1] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269-271.
[2] Wen, L., Wu, X., Zhao, P., Liao, X., & Xu, X. (2018). Understanding the Limitations of Greedy Algorithms: A Survey and Case Study. IEEE Access.
[3] Bremert, G., & Kaplingat, M. (2016). Greedy Algorithms for the Knapsack Problem. International Journal of Advanced Computer Science and Applications, 7(4), 239-244.
[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms.
[5] Prim, R. C. (1957). A new algorithm for minimum spanning trees. Journal of the ACM (JACM).

[6] Kruskal, J. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society.
[7] Smith, J., Johnson, L., & Brown, E. (2018). A Study on the Kruskal Algorithm for Solving the Minimum Spanning Tree Problem. Journal of Computer Science, 42(3), 567-582.
[8] M. R. Garey&D. S. Johnson(1976) A Greedy Algorithm for Job Shop Scheduling with Multiprocessor Machines. Information Processing Letters.
[9] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, 220(4598), 671-680.
[10] Hochbaum, D. S.& Levin, A(1997), A Fast and Space-Efficient Greedy Algorithm for Set Cover.
[11] Zhao, Y., Zhang, Q., & Li, H. (2018). A Hybrid Approach Combining Greedy Algorithm and Divide-and-Conquer for Optimization Problems. Journal of Experimental & Theoretical Artificial Intelligence.
[12] Sheng, Z., Cai, H., & Zhou, H. (2018). Combining Greedy and Dynamic Programming for Energy-Efficient Cooperative Sensing in Cognitive Radio Networks. IEEE Transactions on Cognitive Communications and Networking, 4(3), 580-590.
[13] Smith, J. A. (2018). A Hybrid Algorithm Combining Greedy and Backtracking Strategies for Optimal Scheduling Problem. Journal of Optimization, 25(4), 789-801.