Artificial Intelligence

Unit-3 (Brute Force Search)

Artificial Intelligence 01CE0702

Marwadi University

Department of Computer Engineering

Shilpa Singhal

Brute Force Search

- In AI, brute-force search is a method of problem solving in which all possible solutions are systematically checked for correctness. It is also known as exhaustive search or complete search.
- It is used to find a solution to a problem by repeatedly trying each possible way until one works.
- The main disadvantage of brute-force search is that it can be very time-consuming. For example, if there are a million possible solutions, it would take a long time to check each one.
- Despite its disadvantages, brute-force search is a powerful tool that can be used to solve many difficult problems.

Such an algorithm can be of two types:

- **Optimizing:** In this case, the best solution is found. To find the best solution, it may either find all the possible solutions to find the best solution or if the value of the best solution is known, it stops finding when the best solution is found. For example: Finding the best path for the travelling salesman problem. Here best path means that travelling all the cities and the cost of travelling should be minimum.
- **Satisficing:** It stops finding the solution as soon as the satisfactory solution is found. Or example, finding the travelling salesman path which is within 10% of optimal.

Advantages

- There are many benefits to using brute-force search in AI. First, it is a very simple and straightforward algorithm that can be easily implemented.
- Second, it is guaranteed to find a solution if one exists.
- Third, it can be easily parallelized, meaning that it can take advantage of modern computer architectures to speed up the search.
- Finally, it is often used as a baseline algorithm against which more sophisticated algorithms can be compared.

Disadvantages

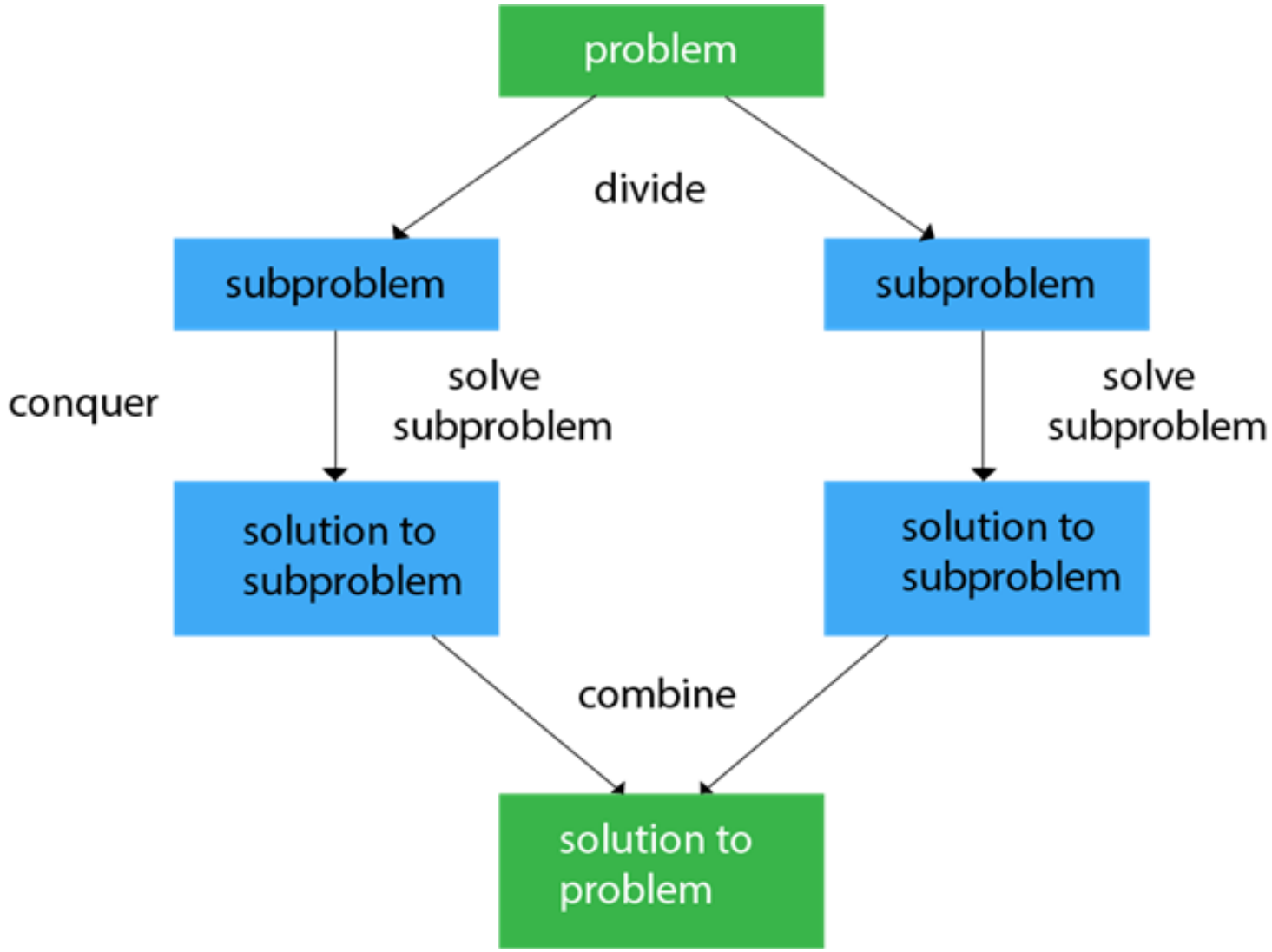
- There are a few drawbacks to using brute-force search in AI. First, it can be very time consuming.
- Second, it can be very resource intensive, especially if the search space is large.
- Third, it can sometimes find sub-optimal solutions, since it does not use any heuristics or domain knowledge.
- Finally, it can be vulnerable to local minima, meaning that it can get stuck in a sub-optimal solution.

Real World Applications

- **Constraint Satisfaction:** One example is in solving constraint satisfaction problems. Constraint satisfaction problems are problems where a set of constraints must be satisfied in order for a solution to be considered valid. These types of problems are often found in real-world scenarios such as planning, scheduling, and resource allocation.
- **Pathfinding** is the process of finding a path from one point to another. This can be done with a graph data structure, where each node represents a location and each edge represents a possible path between two nodes. By using brute-force search, all possible paths can be considered and the shortest path can be found.

Divide and Conquer

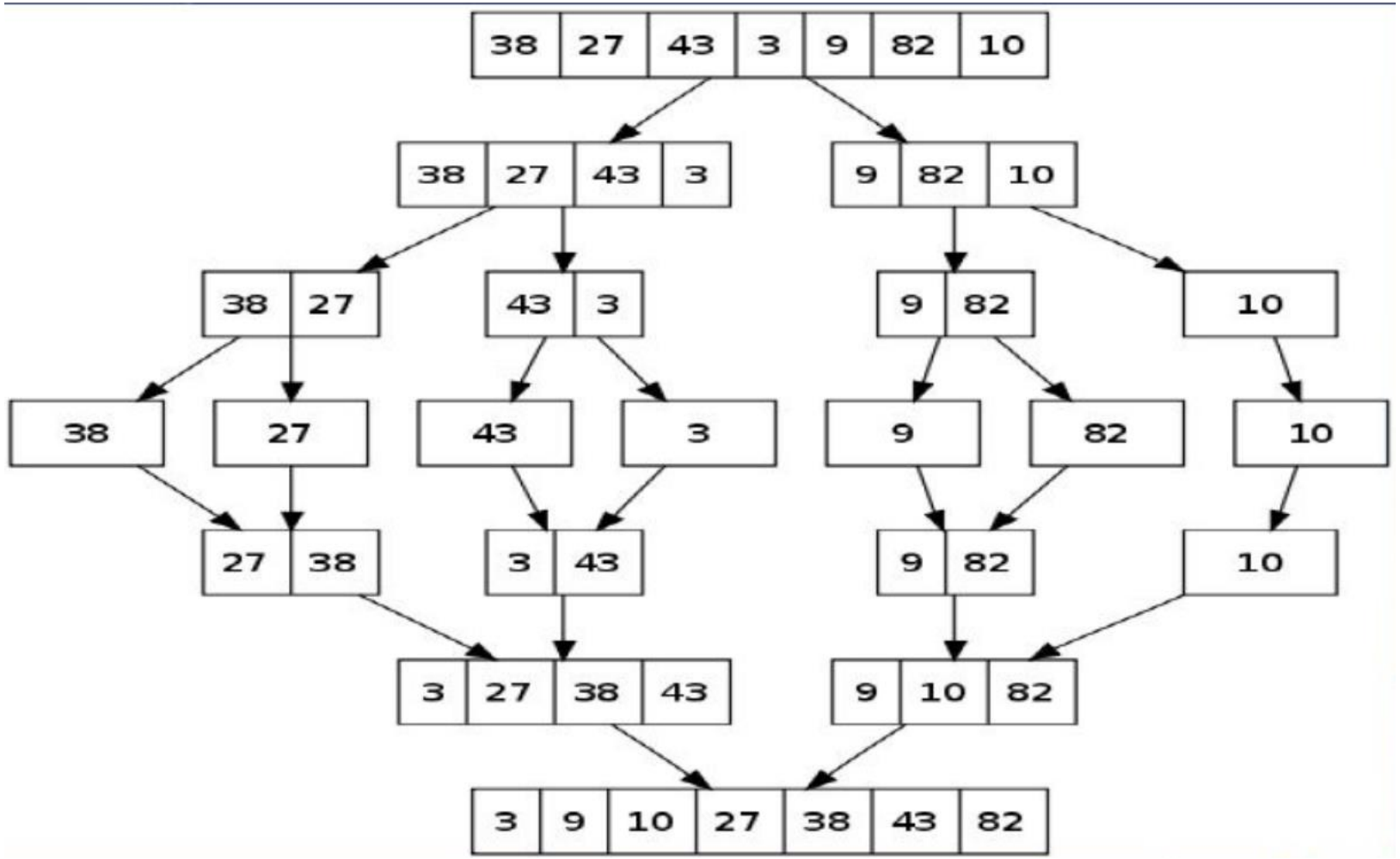
- In algorithmic methods, the design is to take a dispute on a huge input, break the input into minor pieces, decide the problem on each of the small pieces, and then merge the piecewise solutions into a global solution. This mechanism of solving the problem is called the Divide & Conquer Strategy.
- Divide and Conquer algorithm consists of a dispute using the following three steps.
- **Divide** the original problem into a set of subproblems.
- **Conquer:** Solve every subproblem individually, recursively.
- **Combine:** Put together the solutions of the subproblems to get the solution to the whole problem.


problem
divide
subproblem
subproblem
conquer
solve
subproblem
solve
subproblem
solution to
subproblem
solution to
subproblem
combine
solution to
problem

Algorithms using Divide and Conquer Approach:

- **Binary Search:** The binary search algorithm is a searching algorithm, which is also called a half-interval search or logarithmic search. It works by comparing the target value with the middle element existing in a sorted array. After making the comparison, if the value differs, then the half that cannot contain the target will eventually eliminate, followed by continuing the search on the other half. We will again consider the middle element and compare it with the target value. The process keeps on repeating until the target value is met. If we found the other half to be empty after ending the search, then it can be concluded that the target is not present in the array.
- **Quicksort:** It is the most efficient sorting algorithm, which is also known as partition-exchange sort. It starts by selecting a pivot value from an array followed by dividing the rest of the array elements into two sub-arrays. The partition is made by comparing each of the elements with the pivot value. It compares whether the element holds a greater value or lesser value than the pivot and then sort the arrays recursively.
- **Merge Sort:** It is a sorting algorithm that sorts an array by making comparisons. It starts by dividing an array into sub-array and then recursively sorts each of them. After the sorting is done, it merges them back.
- **Closest Pair of Points:** It is a problem of computational geometry. This algorithm emphasizes finding out the closest pair of points in a metric space, given n points, such that the distance between the pair of points should be minimal.

Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82