Chapter 1
Introduction to Multi-objective Optimization

1.1 Introduction

In the field of Artificial Intelligence (AI), search algorithms have been popular since their invention. A search algorithm is typically designed to search and find a desired solution from a given set of all possible solutions to maximize/minimize one or multiple objectives. Depending on the mechanism of a search method, this set of solution can be searched entirely or partially. A search algorithm starts with an initial state (solution), and the ultimate goal is to find a target state (solution). Note that there might be multiple targets in case of multi-objective search that will be discussed in a later section. One of the main challenges in the field of AI is that the set that should be searched by a search algorithm exponentially grows proportional to the size of the problem and the number of objectives. This was not an issue in the past when the problems were less complex and challenging. These days, however, this issue should be addressed when solving a wide range of problems.

1.2 Uninformed and Heuristic AI Search Methods

One of the most well-regarded classifications in the area of AI search algorithms divides such methods into two classes: uninformed and informed. In the former class, an algorithm does not have any additional information about its distance to the goal state. This means that an uninformed search is only aware of the problem definition and should make decisions without knowing the quality of each solution during the search. Each action in such methods is equally good, and a lot of people refer to them as blind search methods. This results in being computationally more expensive and slow when solving large-scale problems. Some of the most popular blind search methods are Breadth-First Search (BFS), Depth-First Search (DFS), brute-force search, and Iterative DFS.

S. Mirjalili and J. S. Dong, *Multi-Objective Optimization using Artificial Intelligence Techniques*, SpringerBriefs in Computational Intelligence,
https://doi.org/10.1007/978-3-030-24835-2_1

In the latter class, informed search, there is additional information that shows an estimate of the distance between the current state and the target state. Such methods are often called heuristics search algorithms [1] as well. A heuristic algorithm leverages a heuristic function to make educated decisions when taking actions, therefore, each action is not equally good as opposed to uninformed search methods. The heuristic function allows evaluating each action and chooses the most promising ones. As the result, informed search methods skip a large portion of the search set (space) and are less computationally expensive than uninformed search algorithms. They can be applied to large-scale problems as well, which is one of the main reason why they are of the most popular search methods lately. Some of the most well-regarded heuristic algorithms are greedy search [2], hill climbing [3], and A* [4].

An example of an uniformed and an informed search can be seen in Fig. 1.1. The problem is to find the highest peak on a terrain using a one-wheeler robot. The terrain is divided into 144 nodes as it can be seen as a grid in Fig. 1.1. The uninfomed method is an uninformed search and goes through all the states. The path highlighted in this figure shows that this search goes through each node row by row. Although the path stopped on the highest peak in this figure, brute force algorithm keeps searching until the last node and compared them all to find the highest one. This guarantees finding the best solution for any terrain, but it is computationally expensive.

Figure 1.1b shows an informed (heuristic) search method, in which all the possible neighboutin points from a given node is evaluated first and the best one (with the maximum elevation) is chosen. This algorithm is called hill climbing, which is one of the most conventional heuristic methods. If we follow the best solution each time, Fig. 1.1 shows that th robot will reach the highest hill with only nine steps. This algorithm is fast similarly to other heuristics. However, it is not complete and if we stat from a wrong initial solution, the navigator will lead the robot to hill that are not the highest hills on the terrain. This is shown in Fig. 1.2. This figures shows that changing the initial position of robot highly change its performance despite the higher speed of its navigation algorithm compared to the brute-force search algorithm.

1.3 Popularity of AI Heuristics and Metaheuristics

As the above example showed, a heurisitc algorithm finds “good” solutions in a reasonable time. This originated from the educated decisions that such methods make to chose more promising states from a given set of possible states. In the example, choosing one neighbor out of four each time using their altitudes cuts down the size of search space significantly. The key point here is that they are unreliable in finding the best possible state (solution) for a given problem. They are normally used when we cannot afford having informed search methods. A heuristic algorithm is problem specific. For instance, A* uses a heuristic function that calculates the spatial distance to the goal state so it can be used to traverse a tree. This mean that a problem should be represented using a state space tree [5] to be solved by the A* algorithms.

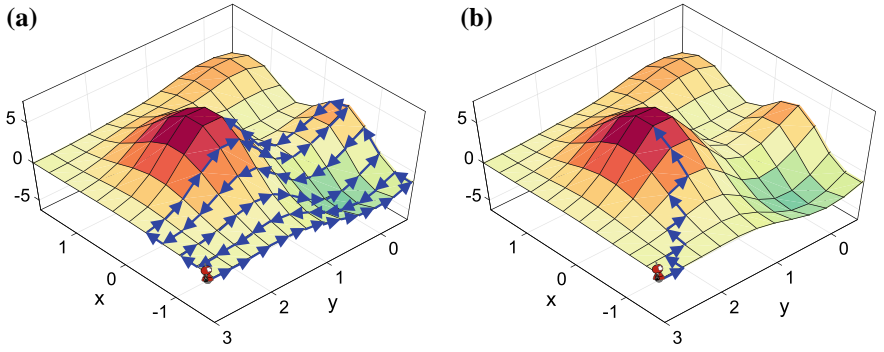

Fig. 1.1 **a** Uninformed search: a brute force search require to check all the possible states (locations) on the terrain, which is equal to 144 (14*14) states. The path that leads to the highest hill is highlighted. **b** Following the location with the highest altitude in the neighbored of each solution is a heurisitc algorithm and requires nine steps only to find the highest peak on the terrain

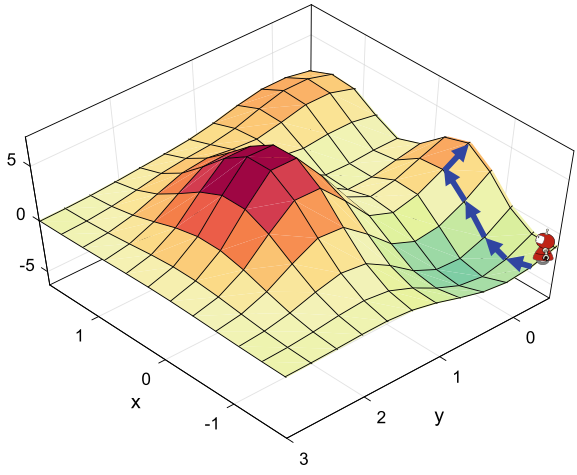

Fig. 1.2 The heuristic algorithm in Fig. 1.1 will lead the robot to a hill that is not the highest hill on the terrain

Problem dependency of heuristic algorithms motivated researchers to design metaheuristics [6], which make few assumption about the problem. This makes them more applicable than heuristics algorithms. Metaheuristics mostly employ stochastic operators to be able to efficiently explore the search space in order to find near-optimal solutions. As opposed to deterministic algorithms, such stochastic techniques find different solutions in each run. Another advantage of metaheuristics is the gradient-free mechanism. They do not need to calculate the derivative of the problem to find its best solution. In other words, metaheurstics consider a problem as a black box.

The above-mentioned advantages have led to an increasable popularity of metaheurstics in a wide rage of fields [7]. The majority of metaherustics follow the same model of search. They start with a set of initial solutions (states) for a given problem. This set may include one or multiple solutions. In the former class, the algorithm

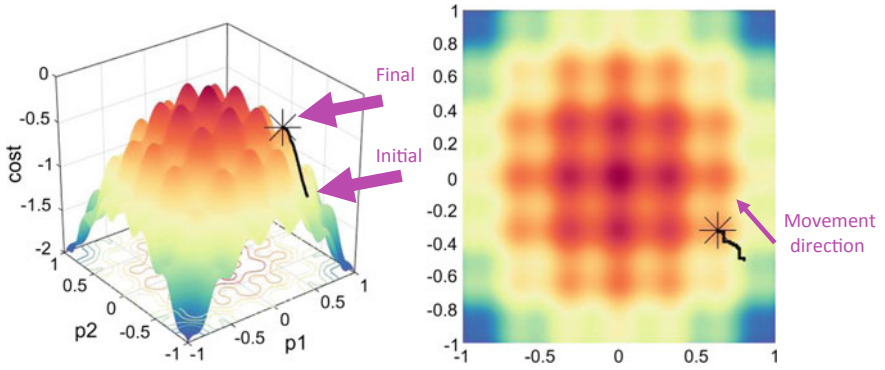

Fig. 1.3 A simple hill climbing algorithm, in which the algorithm starts from a initial point and chooses the best neighbouring solutions each time. This figure shows that such a behaviour, which is called exploitation, may result in finding a sub-optimal (often called locally optimal) solution. This originates from the fact that the algorithm always selects the best solution, whereas sometimes it needs to use a "bad" solution to avoid trapping in locally optimal solutions

is called a single-solution metaheuristic. In the latter case, the algorithm is called population-based. Regardless of the number of solutions in the set, it is constantly changed depending on the algorithm's structure. It means that algorithm iteratively changes the set and evaluates the solutions. This process is stopped when a certain accuracy is achieved or after a given maximum number of iterations.

1.4 Exploration Versus Exploitation in Heuristics and Metaheuristics

It is worth mentioning here that stochastic operators in metaheursitcs increase their exploration, which refers to the discovery of different regions in a search space. Exploitation is opposed to exploration where an algorithm tries to search locally instead of globally. To better understand these two conflicting behaviours in meta-heuritics and other search methods, Figs. 1.3 and 1.5 are given. Figure 1.3 shows a simple hill climbing algorithm, in which the algorithm starts from an initial point and chooses the best neighbouring solutions each time. This figure shows that such a behaviour, which is called exploitation, may result in finding a sub-optimal (often called locally optimal) solution. This originates from the fact that the algorithm always "goes up" in Fig. 1.3. To get to the highest peak, however, the algorithm might also need to step down sometimes to pass the valleys and discover new and possibly better hills. Therefore, pure exploitation is good for linear problems (see Fig. 1.4).

As opposed to exploitation, an algorithm takes 'bad' actions occasionally to find more promising regions and avoid getting trapped in sub-optimal solutions of a

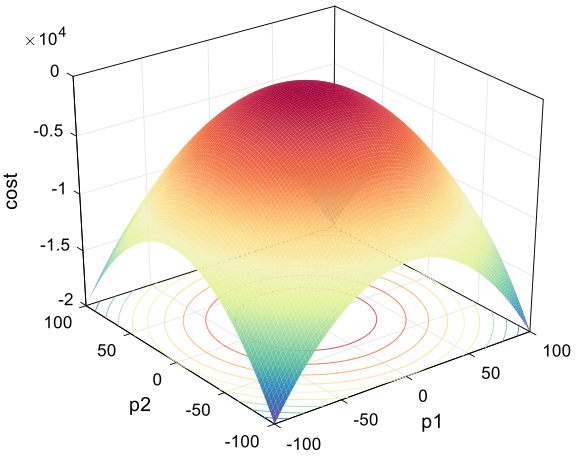

Fig. 1.4 An example of a problem with linear search space. To solve such problems, there is no need to perform exploration since exploitation leads us to the best solution. This is because there is no locally optimal solutions

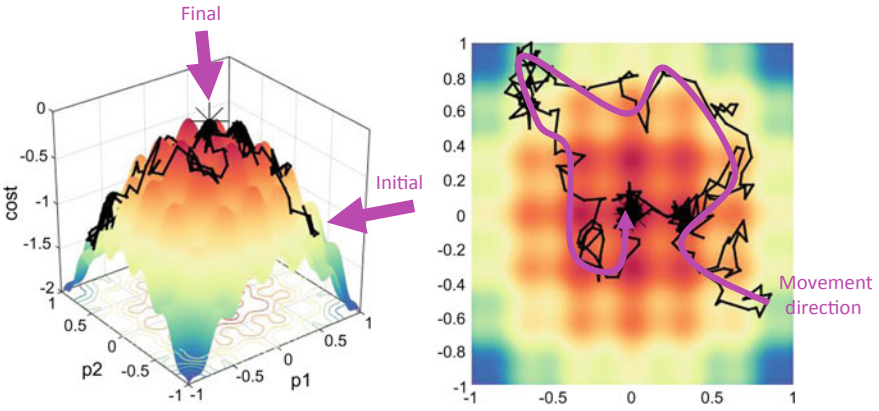

Fig. 1.5 Simulated Annealing (SA) is used which gives a certain probably to choosing downhill steps. This causes exploration of the search space as the arrow shows. Such a behavior allows the algorithm to avoid a large number of sub-optimal solutions

search space. This is shown in Fig. 1.5. In this experiment, Simulated Annealing (SA) [8] is used which gives a certain probability to choose downhill steps. This causes exploration of the search space as the arrow shows in Fig. 1.5. Such a behavior allows the algorithm to avoid a large number of sub-optimal solutions. To find an accurate solution, however, the algorithm needs to exploit the search space at some point. In SA, this is done by decreasing the probability of choosing downhill decrease proportional to the iteration. Overall, exploration is required when solving non-linear problems and should be accompanied with exploitation.

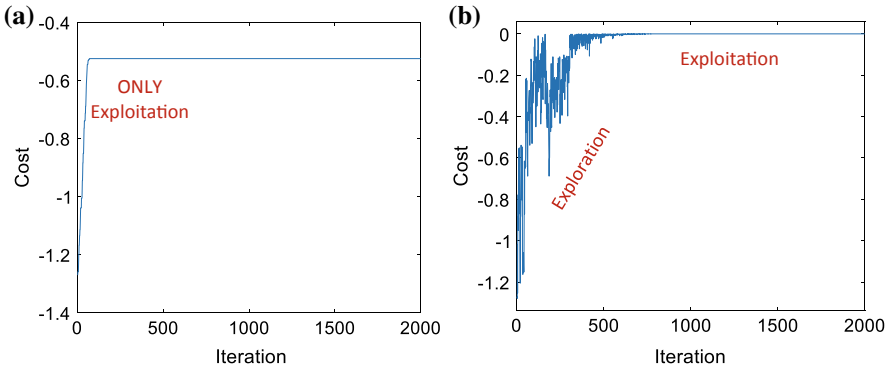

Fig. 1.6 **a** This is the convergence curve of the Hill Climbing algorithm, in which there is no fluctuations in the curve, and the algorithm quickly converges towards a non-improving points in the initial stages of the search process. **b** The Simulated Annealing algorithms shows a different convergence patterns. In the first 500 iterations, the solution in each iteration faces abrupt changes in the objective value which is due to taking downhill steps that leads to worse cost. However, this is changed after near 600 iterations, in which the algorithm starts the exploitation phase and finds the tallest peak around the most promising solution found in the last iteration

Exploration and exploitation can be observed by looking at the fluctuation in the altitude of the solution (which is often referred as cost or fitness) in the landscape represented in Figs. 1.3 and 1.5. To further investigate such behavious, Hill Climbing and Simulated Annealing are both required to search the global optimum in 2000 iterations and the fluctuation of the solution in each run is shown in Fig. 1.6.

Figure 1.6a shows that convergence curve of the Hill Climbing algorithm. In can be observed that there is no fluctuations in the curve, and the algorithm quickly converges towards a non-improving points in the initial stages of the search process. As discussed above, this is desired for a linear problem due to the lack of sub-optimal solutions. As illustrated in Fig. 1.6b, however, the Simulated Annealing algorithms shows a different convergence patterns. In the first 500 iterations, the solution in each iteration faces abrupt changes in the objective value which is due to taking downhill steps that leads worse cost. However, this is changed after nearly iteration 600, in which the algorithm start exploiting the exploitation phase and finds the tallest peak around the most promising solution found in the last iteration.

The above discussion and figures showed that exploration and exploitation are both required and should be balanced when searching for the global solution of problems with non-linear search spaces. Several figure showed how an algorithm might be trapped in a sub-optimal solution. To see the impact of performing only exploration, Fig. 1.7 is given. This figure shows that the coverage of the search space is very high when the exploration is at its highest level. Figure 1.7a and b show the algorithm searches more than 2/3 of the search space. Figure 1.7c illustrates the changes in the cost values, in which the fluctuation can be seen from the first to the last iterations. The issue here is that there is no systematic convergence and the final

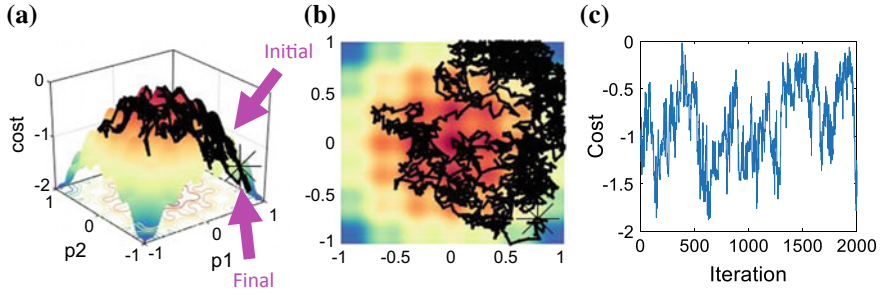

Fig. 1.7 **a** and **b** show the algorithm searches more than 2/3 of the search space. **c** illustrates the changes in the cost values, in which the fluctuation can be seen from the first to the last iterations. The issue here is that there is no systematic convergence and the final solution obtained (see the arrow in Fig. 1.7a) is even worse than the case where the algorithm only performs exploitation (see Fig. 1.3)

solution obtained (see the arrow in Fig. 1.7a) is even worse than the case where the algorithm only performs exploitation (see Fig. 1.3).

Another popular classification of metaheuristic is evolutionary, swarm-based, and physics-based algorithms [9]. In the first class, an algorithm mimics evolutionary phenomena in nature. For instance, Genetic Algorithms (GA) [10] mimics the way that organism's best genes propagate through generations to increase the chance of survival. In the second class, the problem solving of a swarm that originates from the social intelligence of individuals. For instance, Ant Colony Optimization (ACO) [11] mimics the process of finding the shortest path from a nest to a food source in an ant colony. In fact, ACO simulates the local interactions between ants using pheromone that allows solving such a complex problem in a very dynamic environment. In the last class, physics-based algorithm, physical phenomena are the main source of inspiration. A popular algorithm in this class is Simulated Annealing, which simulates the annealing process (heating and controlled cooling) of a material to increase its crystal's size.

1.5 Different Methods of Multi-objective Search (Optimization)

Regardless of the class metaheuristics, they are mostly considered as optimization algorithms. They have been largely employed in the literature to find the optimal values for the parameters of a problem to maximize or minimize an objective (often called as cost or fitness) function. If the problem has one objective, the algorithm is called a single-objective optimization. In a single-objective problem, there is one global solution to find. In a multi-objective problem, however, there are more than one objective, which might be in-conflict. This makes the process of optimization a lot

more difficult than single-objective problems due to addressing multiple objectives. There are different methods in the literature of multi-objective meta-herustics to handle multiple objectives as follows:

- A priori methods
- A posteriori methods
- Interactive methods

In a priori method, multiple objectives are aggregated into a single objective. For instance, this process can be done using a weight for each objective, which allows us to define the importance of objectives. After the weighted aggregation, a single-objective optimization algorithm can be employed to find the global optimum. There are several drawbacks with this method that will be discussed in a later chapter.

In a posteriori method, the multi-objective formulation of the problem is first maintained. An algorithm (often multi-objective) is then employed to find the best trade-offs between the objectives. This leads to finding a set of solutions called Pareto optimal solution set. This method has its own pros and cons as well that will be discussed later on. However, it is worth mentioning here that a decision making process is required after the optimization process to choose one of the solutions.

In the interactive methods, decision makers are involved during the optimization process. This method of multi-objective optimization is often called human-in-the-loop optimization due to the direct impact of a human on the space searched by an algorithm.

1.6 Scope and Structure of the Book

In this book, the preliminaries, essential definitions, and state-of-the-art optimization algorithms to solve multi-objective optimization problems will be presented. It is tried to present several algorithms to show how multi-objective optimization is done using the three above-mentioned paradigms. The rest of the book is organized as follows:

Chapter 2 provides preliminaries and essential definitions of multi-objective optimization specially. A priori, a posteriori and interactive multi-objective optimization are also discussed in details. Two conventional metaheuristics including Multi-objective PSO (MOPSO) and multi-objective GA (NSGA-II) are presented and analyzed in Chaps. 3 and 4. Chapter 5 covers the recently-proposed multi-objective Grey Wolf Optimizer (MOGWO).

References

1. Kanal L, Kumar V (eds) (2012) Search in artificial intelligence. Springer Science & Business Media, New York
2. Resende MG, Ribeiro CC (2003) Greedy randomized adaptive search procedures. In: Handbook of metaheuristics. Springer, Boston, (pp 219–249)

3. Yuret D, De La Maza M (1993) Dynamic hill climbing: overcoming the limitations of optimization techniques. In: The second Turkish symposium on artificial intelligence and neural networks. Citeseer, (pp 208–212)
4. Liu X, Gong D (2011) A comparative study of A-star algorithms for search and rescue in perfect maze. In: 2011 International conference on electric information and control engineering. IEEE, (pp 24–27)
5. Esposito F, Malerba D, Semeraro G (1993) Decision tree pruning as a search in the state space. In: European conference on machine learning. Springer, Berlin (pp 165–184)
6. BoussaïD I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci 237:82–117
7. Osman IH, Kelly JP (1997) Meta-heuristics theory and applications. J Oper Res Soc 48(6):657–657
8. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
9. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67
10. Goldberg DE, Holland JH (1988) Genetic algorithms and machine learning. Mach Learn 3(2):95–99
11. Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344(2–3):243–278