Metaheuristic

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.[1][2] 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.[3]

Compared to optimization algorithms and iterative methods, metaheuristics do not guarantee that a globally optimal solution can be found on some class of problems.[3] Many metaheuristics implement some form of stochastic optimization, so that the solution found is dependent on the set of random variables generated.[2] In combinatorial optimization, by searching over a large set of feasible solutions, metaheuristics can often find good solutions with less computational effort than optimization algorithms, iterative methods, or simple heuristics.[3] As such, they are useful approaches for optimization problems.[2] Several books and survey papers have been published on the subject.[2][3][4][5][6]

Most literature on metaheuristics is experimental in nature, describing empirical results based on computer experiments with the algorithms. But some formal theoretical results are also available, often on convergence and the possibility of finding the global optimum.[3] Many metaheuristic methods have been published with claims of novelty and practical efficacy. While the field also features high-quality research, many of the publications have been of poor quality; flaws include vagueness, lack of conceptual elaboration, poor experiments, and ignorance of previous literature.[7]

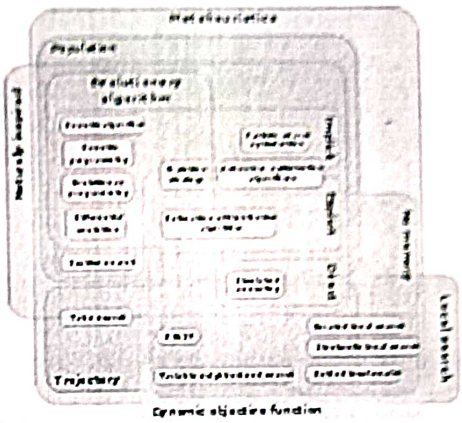
Properties

These are properties that characterize most metaheuristics:[3]

- Metaheuristics are strategies that guide the search process.
- The goal is to efficiently explore the search space in order to find near–optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- Metaheuristics are not problem-specific.

Classification

There are a wide variety of metaheuristics[2] and a number of properties with respect to which to classify them.[3]



Euler diagram of the different classifications of metaheuristics.[8]

Local search vs. global search

One approach is to characterize the type of search strategy.[3] One type of search strategy is an improvement on simple local search algorithms. A well known local search algorithm is the hill climbing method which is used to find local optimums. However, hill climbing does not guarantee finding global optimum solutions.

Many metaheuristic ideas were proposed to improve local search heuristic in order to find better solutions. Such metaheuristics include simulated annealing, tabu search, iterated local search, variable neighborhood search, and GRASP.[3] These metaheuristics can both be classified as local search-based or global search metaheuristics.

Other global search metaheuristic that are not local search-based are usually population-based metaheuristics. Such metaheuristics include ant colony optimization, evolutionary computation, particle swarm optimization, genetic algorithm, and rider optimization algorithm[9]

Single-solution vs. population-based

Another classification dimension is single solution vs population-based searches.[3][6] Single solution approaches focus on modifying and improving a single candidate solution; single solution metaheuristics include simulated annealing, iterated local search, variable neighborhood search, and guided local search.[6] Population-based approaches maintain and improve multiple candidate solutions, often using population characteristics to guide the search; population based metaheuristics include evolutionary computation, genetic algorithms, and particle swarm optimization.[6] Another category of metaheuristics is Swarm intelligence which is a collective behavior of decentralized, self-organized agents in a population or swarm. Ant colony optimization,[10] particle swarm optimization,[6] social cognitive optimization are examples of this category.

Hybridization and memetic algorithms

A hybrid metaheuristic is one that combines a metaheuristic with other optimization approaches, such as algorithms from mathematical programming, constraint programming, and machine learning. Both components of a hybrid metaheuristic may run concurrently and exchange information to guide the search.

On the other hand, Memetic algorithms[11] represent the synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. An example of memetic algorithm is the use of a local search algorithm instead of a basic mutation operator in evolutionary algorithms.

Parallel metaheuristics

A parallel metaheuristic is one that uses the techniques of parallel programming to run multiple metaheuristic searches in parallel; these may range from simple distributed schemes to concurrent search runs that interact to improve the overall solution.

Nature-inspired and metaphor-based metaheuristics

A very active area of research is the design of nature-inspired metaheuristics. Many recent metaheuristics, especially evolutionary computation-based algorithms, are inspired by natural systems. Nature acts as a source of concepts, mechanisms and principles for designing of artificial computing systems to deal with complex computational problems. Such metaheuristics include simulated annealing, evolutionary algorithms, ant colony optimization and particle swarm optimization. A large number of more recent metaphor-inspired metaheuristics have started to attract criticism in the research community for hiding their lack of novelty behind an elaborate metaphor.[7]

Applications

Metaheuristics are used for combinatorial optimization in which an optimal solution is sought over a discrete search-space. An example problem is the travelling salesman problem where the search-space of candidate solutions grows faster than exponentially as the size of the problem increases, which makes an exhaustive search for the optimal solution infeasible. Additionally, multidimensional combinatorial problems, including most design problems in engineering[12][13][14] such as form-finding and behavior-finding, suffer from the curse of dimensionality, which also makes them infeasible for exhaustive search or analytical methods. Metaheuristics are also widely used for jobshop scheduling and job selection problems. Popular metaheuristics for combinatorial problems include simulated annealing by Kirkpatrick et al.,[15] genetic algorithms by Holland et al.,[16] scatter search[17] and tabu search[18] by Glover. Literature review on metaheuristic optimization,[19] suggested that it was Fred Glover who coined the word metaheuristics.[20]

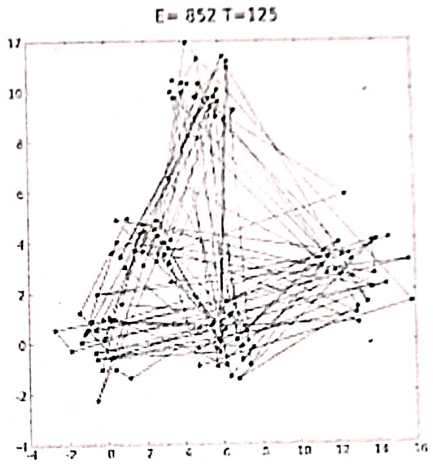
Metaheuristic Optimization Frameworks

A MOF can be defined as "a set of software tools that provide a correct and reusable implementation of a set of metaheuristics, and the basic mechanisms to accelerate the implementation of its partner subordinate heuristics (possibly including solution encodings and technique-specific operators), which are necessary to solve a particular problem instance using techniques provided".[21]

There are many candidate optimization tools which can be considered as a MOF of varying feature: Comet, EvA2, evolvica, Evolutionary::Algorithm, GAPlayground, jaga, JCLEC, JGAP, jMetal, n-genes, Open Beagle, Opt4j, ParadisEO/EO, Pisa, Watchmaker, FOM, Hypercube, HotFrame, Templar,

Simulated annealing

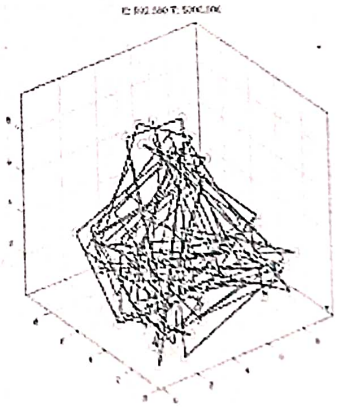
Simulated annealing (**SA**) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem. It is often used when the search space is discrete (for example the traveling salesman problem, the boolean satisfiability problem, protein structure prediction, and job-shop scheduling). For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to exact algorithms such as gradient descent or branch and bound.



Simulated annealing can be used to solve combinatorial problems. Here it is applied to the travelling salesman problem to minimize the length of a route that connects all 125 points.

The name of the algorithm comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material to alter its physical properties. Both are attributes of the material that depend on their thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy or Gibbs energy. Simulated annealing can be used for very hard computational optimization problems where exact algorithms fail; even though it usually achieves an approximate solution to the global minimum, it could be enough for many practical problems.

The problems solved by SA are currently formulated by an objective function of many variables, subject to several constraints. In practice, the constraint can be penalized as part of the objective function.



Travelling salesman problem in 3D for 120 points solved with simulated annealing.

Similar techniques have been independently introduced on several occasions, including Pincus (1970),[1] Khachaturyan et al (1979,[2] 1981[3]), Kirkpatrick, Gelatt and Vecchi (1983), and Cerny (1985).[4] In 1983, this approach was used by Kirkpatrick, Gelatt Jr., Vecchi,[5] for a solution of the traveling salesman problem. They also proposed its current name, simulated annealing.

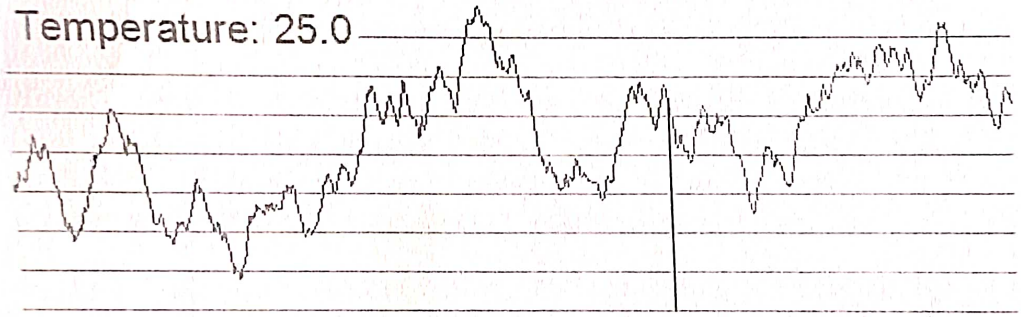
This notion of slow cooling implemented in the simulated annealing algorithm is interpreted as a slow decrease in the probability of accepting worse solutions as the solution space is explored. Accepting worse solutions allows for a more extensive search for the global optimal solution. In general, simulated annealing algorithms work as follows. The temperature progressively decreases from an initial positive value to zero. At each time step, the algorithm randomly selects a solution close to the current one, measures

its quality, and moves to it according to the temperature-dependent probabilities of selecting better or worse solutions, which during the search respectively remain at 1 (or positive) and decrease toward zero.

The simulation can be performed either by a solution of kinetic equations for density functions[6][7] or by using the stochastic sampling method.[5][8] The method is an adaptation of the Metropolis–Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, published by N. Metropolis et al. in 1953.[9]

Overview

The state of some physical systems, and the function $E(s)$ to be minimized, is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary *initial state*, to a state with the minimum possible energy.



Simulated annealing searching for a maximum. The objective here is to get to the highest point. In this example, it is not enough to use a simple hill climb algorithm, as there are many local maxima. By cooling the temperature slowly the global maximum is found.

The basic iteration

At each step, the simulated annealing heuristic considers some neighboring state s^* of the current state s, and probabilistically decides between moving the system to state s^* or staying in-state s. These probabilities ultimately lead the system to move to states of lower energy. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.

The neighbours of a state

Optimization of a solution involves evaluating the neighbours of a state of the problem, which are new states produced through conservatively altering a given state. For example, in the travelling salesman problem each state is typically defined as a permutation of the cities to be visited, and the neighbors of any state are the set of permutations produced by swapping any two of these cities. The well-defined way in which the states are altered to produce neighboring states is called a "move", and different moves give different sets of neighboring states. These moves usually result in minimal alterations of the last state, in an attempt to progressively improve the solution through iteratively improving its parts (such as the city connections in the traveling salesman problem).

Simple heuristics like hill climbing, which move by finding better neighbour after better neighbour and stop when they have reached a solution which has no neighbours that are better solutions, cannot guarantee to lead to any of the existing better solutions – their outcome may easily be just a local optimum, while the actual best solution would be a global optimum that could be different. Metaheuristics use the neighbours of a solution as a way to explore the solution space, and although they prefer better neighbours, they also accept worse neighbours in order to avoid getting stuck in local optima; they can find the global optimum if run for a long enough amount of time.

Acceptance probabilities

The probability of making the transition from the current state s to a candidate new state s_{new} is specified by an *acceptance probability function* $P(e, e_{new}, T)$, that depends on the energies $e = E(s)$ and $e_{new} = E(s_{new})$ of the two states, and on a global time-varying parameter T called the *temperature*. States with a smaller energy are better than those with a greater energy. The probability function P must be positive even when e_{new} is greater than e. This feature prevents the method from becoming stuck at a local minimum that is worse than the global one.

When T tends to zero, the probability $P(e, e_{new}, T)$ must tend to zero if $e_{new} > e$ and to a positive value otherwise. For sufficiently small values of T, the system will then increasingly favor moves that go "downhill" (i.e., to lower energy values), and avoid those that go "uphill." With $T = 0$ the procedure reduces to the greedy algorithm, which makes only the downhill transitions.

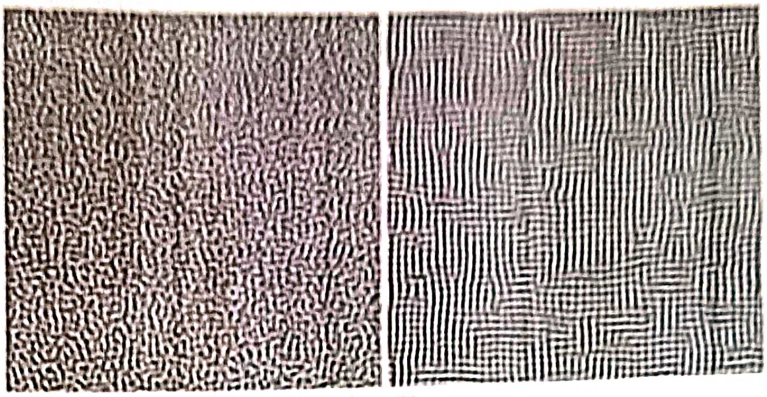
In the original description of simulated annealing, the probability $P(e, e_{new}, T)$ was equal to 1 when $e_{new} < e$—i.e., the procedure always moved downhill when it found a way to do so, irrespective of the temperature. Many descriptions and implementations of simulated annealing still take this condition as part of the method's definition. However, this condition is not essential for the method to work.

The P function is usually chosen so that the probability of accepting a move decreases when the difference $e_{new} - e$ increases—that is, small uphill moves are more likely than large ones. However, this requirement is not strictly necessary, provided that the above requirements are met.

Given these properties, the temperature T plays a crucial role in controlling the evolution of the state s of the system with regard to its sensitivity to the variations of system energies. To be precise, for a large T, the evolution of s is sensitive to coarser energy variations, while it is sensitive to finer energy variations when T is small.

The annealing schedule

The name and inspiration of the algorithm demand an interesting feature related to the temperature variation to be embedded in the operational characteristics of the algorithm. This necessitates a gradual reduction of the temperature as the simulation proceeds. The algorithm starts initially with T set to a high value (or infinity), and then it is decreased at each step following some *annealing schedule*—which may be specified by the user, but must end with $T = 0$ towards the end of the allotted time budget. In this way, the system is expected to wander initially towards a broad region of the search space containing good solutions, ignoring small features of the energy function; then drift towards low-energy regions that become narrower and narrower, and finally move downhill according to the steepest descent heuristic.



Fast Slow

Example illustrating the effect of cooling schedule on the performance of simulated annealing. The problem is to rearrange the pixels of an image so as to minimize a certain potential energy function, which causes similar colors to attract at short range and repel at a slightly larger distance. The elementary moves swap two adjacent pixels. These images were obtained with a fast cooling schedule (left) and a slow cooling schedule (right), producing results similar to amorphous and crystalline solids, respectively.

For any given finite problem, the probability that the simulated annealing algorithm terminates with a global optimal solution approaches 1 as the annealing schedule is extended.[10] This theoretical result, however, is not particularly helpful, since the time required to ensure a significant probability of success will usually exceed the time required for a complete search of the solution space.[11]

Pseudocode

The following pseudocode presents the simulated annealing heuristic as described above. It starts from a state s_0 and continues until a maximum of k_{max} steps have been taken. In the process, the call neighbour(s) should generate a randomly chosen neighbour of a given state s; the call random(0, 1) should pick and return a value in the range [0, 1], uniformly at random. The annealing schedule is defined by the call temperature(r), which should yield the temperature to use, given the fraction r of the time budget that has been expended so far.

- Let $s = s_0$
- For $k = 0$ through k_{max} (exclusive):
  - $T \leftarrow$ temperature(1 - $(k+1)/k_{max}$)
  - Pick a random neighbour, $s_{new} \leftarrow$ neighbour(s)
  - If $P(E(s), E(s_{new}), T) \geq$ random(0, 1):
    - $s \leftarrow s_{new}$
- Output: the final state s

Selecting the parameters

WIKIPEDIA
The Free Encyclopedia

Tabu search

Tabu search (TS) is a metaheuristic search method employing local search methods used for mathematical optimization. It was created by Fred W. Glover in 1986[1] and formalized in 1989.[2][3]

Local (neighborhood) searches take a potential solution to a problem and check its immediate neighbors (that is, solutions that are similar except for very few minor details) in the hope of finding an improved solution. Local search methods have a tendency to become stuck in suboptimal regions or on plateaus where many solutions are equally fit.

Tabu search enhances the performance of local search by relaxing its basic rule. First, at each step *worsening* moves can be accepted if no improving move is available (like when the search is stuck at a strict local minimum). In addition, *prohibitions* (henceforth the term *tabu*) are introduced to discourage the search from coming back to previously-visited solutions.

The implementation of tabu search uses memory structures that describe the visited solutions or user-provided sets of rules.[2] If a potential solution has been previously visited within a certain short-term period or if it has violated a rule, it is marked as "tabu" (forbidden) so that the algorithm does not consider that possibility repeatedly.

Background

The word *tabu* comes from the Tongan word to indicate things that cannot be touched because they are sacred.[4]

Tabu search is a metaheuristic algorithm that can be used for solving combinatorial optimization problems (problems where an optimal ordering and selection of options is desired).

Current applications of TS span the areas of resource planning, telecommunications, VLSI design, financial analysis, scheduling, space planning, energy distribution, molecular engineering, logistics, pattern classification, flexible manufacturing, waste management, mineral exploration, biomedical analysis, environmental conservation and scores of others. In recent years, journals in a wide variety of fields have published tutorial articles and computational studies documenting successes by tabu search in extending the frontier of problems that can be handled effectively – yielding solutions whose quality often significantly surpasses that obtained by methods previously applied. A comprehensive list of applications, including summary descriptions of gains achieved from practical implementations, can be found in [5]

Basic description

Tabu search uses a local or neighborhood search procedure to iteratively move from one potential solution x to an improved solution x' in the neighborhood of x, until some stopping criterion has been satisfied (generally, an attempt limit or a score threshold). Local search procedures often become stuck in poor-scoring areas or areas where scores plateau. In order to avoid these pitfalls and

explore regions of the search space that would be left unexplored by other local search procedures, tabu search carefully explores the neighborhood of each solution as the search progresses. The solutions admitted to the new neighborhood, $N^*(x)$, are determined through the use of memory structures. Using these memory structures, the search progresses by iteratively moving from the current solution x to an improved solution x' in $N^*(x)$.

Tabu search has several similarities with simulated annealing, as both involve possible downhill moves. In fact, simulated annealing could be viewed as a special form of TS, whereby we use "graduated tenure", that is, a move becomes tabu with a specified probability.

These memory structures form what is known as the tabu list, a set of rules and banned solutions used to filter which solutions will be admitted to the neighborhood $N^*(x)$ to be explored by the search. In its simplest form, a tabu list is a short-term set of the solutions that have been visited in the recent past (less than n iterations ago, where n is the number of previous solutions to be stored — is also called the tabu tenure). More commonly, a tabu list consists of solutions that have changed by the process of moving from one solution to another. It is convenient, for ease of description, to understand a "solution" to be coded and represented by such attributes.

Types of memory

The memory structures used in tabu search can roughly be divided into three categories:[6]

- Short-term: The list of solutions recently considered. If a potential solution appears on the tabu list, it cannot be revisited until it reaches an expiration point.
- Intermediate-term: Intensification rules intended to bias the search towards promising areas of the search space.
- Long-term: Diversification rules that drive the search into new regions (i.e., regarding resets when the search becomes stuck in a plateau or a suboptimal dead-end).

Short-term, intermediate-term and long-term memories can overlap in practice. Within these categories, memory can further be differentiated by measures such as frequency and impact of changes made. One example of an intermediate-term memory structure is one that prohibits or encourages solutions that contain certain attributes (e.g., solutions that include undesirable or desirable values for certain variables) or a memory structure that prevents or induces certain moves (e.g. based on frequency memory applied to solutions sharing features in common with unattractive or attractive solutions found in the past). In short-term memory, selected attributes in solutions recently visited are labelled "tabu-active." Solutions that contain tabu-active elements are banned. Aspiration criteria are employed to override a solution's tabu state, thereby including the otherwise-excluded solution in the allowed set (provided the solution is "good enough" according to a measure of quality or diversity). A simple and commonly used aspiration criterion is to allow solutions which are better than the currently-known best solution.

Short-term memory alone may be enough to achieve solutions superior to those found by conventional local search methods, but intermediate and long-term structures are often necessary for solving harder problems.[7] Tabu search is often benchmarked against other metaheuristic methods — such as simulated annealing, genetic algorithms, ant colony optimization algorithms, reactive search optimization, guided local search, or greedy randomized adaptive search. In addition, tabu search is sometimes combined with other metaheuristics to create hybrid methods. The most common tabu

search hybrid arises by joining TS with scatter search,[8][9] a class of population-based procedures which has roots in common with tabu search, and is often employed in solving large non-linear optimization problems.

Pseudocode

The following pseudocode presents a simplified version of the tabu search algorithm as described above. This implementation has a rudimentary short-term memory, but contains no intermediate or long-term memory structures. The term "fitness" refers to an evaluation of the candidate solution, as embodied in an objective function for mathematical optimization.

```
1  sBest ← s0
2  bestCandidate ← s0
3  tabuList ← []
4  tabuList.push(s0)
5  while (not stoppingCondition())
6      sNeighborhood ← getNeighbors(bestCandidate)
7      bestCandidate ← sNeighborhood[0]
8      for (sCandidate in sNeighborhood)
9          if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
10             bestCandidate ← sCandidate
11         end
12     end
13     if (fitness(bestCandidate) > fitness(sBest))
14         sBest ← bestCandidate
15     end
16     tabuList.push(bestCandidate)
17     if (tabuList.size > maxTabuSize)
18         tabuList.removeFirst()
19     end
20 end
21 return sBest
```

Lines 1-4 represent some initial setup, respectively creating an initial solution (possibly chosen at random), setting that initial solution as the best seen to date, and initializing a tabu list with this initial solution. In this example, the tabu list is simply a short term memory structure that will contain a record of the elements of the states visited.

The core algorithmic loop starts in line 5. This loop will continue searching for an optimal solution until a user-specified stopping condition is met (two examples of such conditions are a simple time limit or a threshold on the fitness score). The neighboring solutions are checked for tabu elements in line 9. Additionally, the algorithm keeps track of the best solution in the neighbourhood, that is not tabu.

The fitness function is generally a mathematical function, which returns a score or the aspiration criteria are satisfied — for example, an aspiration criterion could be considered as a new search space is found[4]). If the best local candidate has a higher fitness value than the current best (line 13), it is set as the new best (line 14). The local best candidate is always added to the tabu list (line 16) and if the tabu list is full (line 17), some elements will be allowed to expire (line 18). Generally, elements expire from the list in the same order they are added. The procedure will select the best local candidate (although it has worse fitness than the sBest) in order to escape the local optimal.

This process continues until the user specified stopping criterion is met, at which point, the best solution seen during the search process is returned (line 21).

Example: the traveling salesman problem

The traveling salesman problem (TSP) is sometimes used to show the functionality of tabu search.[7] This problem poses a straightforward question: given a list of cities, what is the shortest route that visits every city? For example, if city A and city B are next to each other, while city C is farther away, the total distance traveled will be shorter if cities A and B are visited one after the other before visiting city C. Since finding an optimal solution is NP-hard, heuristic-based approximation methods (such as local searches) are useful for devising close-to-optimal solutions. To obtain good TSP solutions, it is essential to exploit the graph structure. The value of exploiting problem structure is a recurring theme in metaheuristic methods, and tabu search is well-suited to this. A class of strategies associated with tabu search called ejection chain methods has made it possible to obtain high-quality TSP solutions efficiently [10]

On the other hand, a simple tabu search can be used to find a satisficing solution for the traveling salesman problem (that is, a solution that satisfies an adequacy criterion, although not with the high quality obtained by exploiting the graph structure). The search starts with an initial solution, which can be generated randomly or according to some sort of nearest neighbor algorithm. To create new solutions, the order that two cities are visited in a potential solution is swapped. The total traveling distance between all the cities is used to judge how ideal one solution is compared to another. To prevent cycles – i.e., repeatedly visiting a particular set of solutions – and to avoid becoming stuck in local optima, a solution is added to the tabu list if it is accepted into the solution neighborhood, $N^*(x)$.

New solutions are created until some stopping criterion, such as an arbitrary number of iterations, is met. Once the simple tabu search stops, it returns the best solution found during its execution.

References

1. Fred Glover (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". *Computers and Operations Research*. **13** (5): 533–549. doi:10.1016/0305-0548(86)90048-1 (https://doi.org/10.1016%2F0305-0548%2886%2990048-1).
2. Fred Glover (1989). "Tabu Search – Part 1". *ORSA Journal on Computing*. **1** (2): 190–206. doi:10.1287/ijoc.1.3.190 (https://doi.org/10.1287%2Fijoc.1.3.190).
3. Fred Glover (1990). "Tabu Search – Part 2". *ORSA Journal on Computing*. **2** (1): 4–32. doi:10.1287/ijoc.2.1.4 (https://doi.org/10.1287%2Fijoc.2.1.4).
4. "Courses" (http://www.ise.ncsu.edu/fangroup/ie789.dir/IE789F_tabu.pdf) (PDF).
5. F. Glover; M. Laguna (1997). *Tabu Search* (https://archive.org/details/tabusearch00glov_0). Kluwer Academic Publishers. ISBN 978-1-4613-7987-4.
6. Fred Glover (1990). "Tabu Search: A Tutorial". *Interfaces*.
7. M. Malek; M. Huruswamy; H. Owens; M. Pandya (1989). "Serial and parallel search techniques for the traveling salesman problem". *Annals of OR: Linkages with Artificial Intelligence*.
8. F. Glover, M. Laguna & R. Marti (2000). "Fundamentals of Scatter Search and Path Relinking". *Control and Cybernetics*. **29** (3): 653–684.
9. M. Laguna & R. Marti (2003). *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers. ISBN 9781402073762.

WIKIPEDIA
The Free Encyclopedia

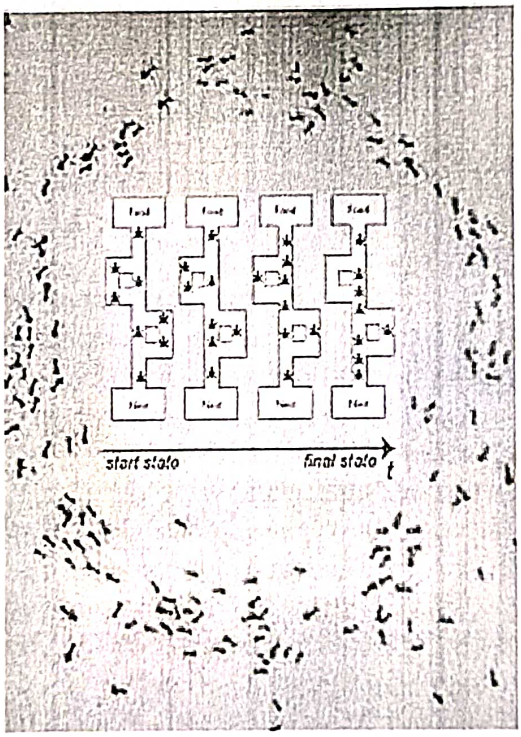
Ant colony optimization algorithms

(Redirected from Ant colony optimization)

In computer science and operations research, the **ant colony optimization** algorithm (ACO) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. Artificial ants stand for multi-agent methods inspired by the behavior of real ants. The pheromone-based communication of biological ants is often the predominant paradigm used.[2] Combinations of artificial ants and local search algorithms have become a method of choice for numerous optimization tasks involving some sort of graph, e.g., vehicle routing and internet routing.

Ant behavior was the inspiration for the metaheuristic optimization technique

As an example, ant colony optimization[3] is a class of optimization algorithms modeled on the actions of an ant colony.[4] Artificial 'ants' (e.g. simulation agents) locate optimal solutions by moving through a parameter space representing all possible solutions. Real ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.[5] One variation on this approach is the bees algorithm, which is more analogous to the foraging patterns of the honey bee, another social insect.



The main quality of the colonies of insects, ants or bees lies in the fact that they are part of a self-organized group in which the keyword is simplicity.

Every day, ants solve complex problems due to a sum of simple interactions, which are carried out by individuals.

The ant is, for example, able to use the quickest way from the anthill to its food simply by following the way marked with pheromones.

When a colony of ants is confronted with the choice of reaching their food via two different routes of which one is much shorter than the other, their choice is entirely random. However, those who use the shorter route reach the food faster and therefore go back and forth more often between the anthill and the food.[1]

This algorithm is a member of the ant colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. Initially proposed by Marco Dorigo in 1992 in his PhD thesis,[6][7] the first algorithm was aiming to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several

problems have emerged, drawing on various aspects of the behavior of ants. From a broader perspective, ACO performs a model-based search[8] and shares some similarities with estimation of distribution algorithms.

Overview

In the natural world, ants of some species (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but instead to follow the trail, returning and reinforcing it if they eventually find food (see Ant communication).[9]

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. The influence of pheromone evaporation in real ant systems is unclear, but it is very important in artificial systems.[10]

The overall result is that when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to many ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve.

Ambient networks of intelligent objects

New concepts are required since “intelligence” is no longer centralized but can be found throughout all minuscule objects. Anthropocentric concepts have been known to lead to the production of IT systems in which data processing, control units and calculating forces are centralized. These centralized units have continually increased their performance and can be compared to the human brain. The model of the brain has become the ultimate vision of computers. Ambient networks of intelligent objects and, sooner or later, a new generation of information systems that are even more diffused and based on nanotechnology, will profoundly change this concept. Small devices that can be compared to insects do not dispose of a high intelligence on their own. Indeed, their intelligence can be classed as fairly limited. It is, for example, impossible to integrate a high performance calculator with the power to solve any kind of mathematical problem into a biochip that is implanted into the human body or integrated in an intelligent tag which is designed to trace commercial articles. However, once those objects are interconnected they dispose of a form of intelligence that can be compared to a colony of ants or bees. In the case of certain problems, this type of intelligence can be superior to the reasoning of a centralized system similar to the brain.[11]

Nature offers several examples of how minuscule organisms, if they all follow the same basic rule, can create a form of collective intelligence on the macroscopic level. Colonies of social insects perfectly illustrate this model which greatly differs from human societies. This model is based on the co-operation of independent units with simple and unpredictable behavior.[12] They move through their surrounding area to carry out certain tasks and only possess a very limited amount of information to do so. A colony of ants, for example, represents numerous qualities that can also be applied to a

network of ambient objects. Colonies of ants have a very high capacity to adapt themselves to changes in the environment as well as an enormous strength in dealing with situations where one individual fails to carry out a given task. This kind of flexibility would also be very useful for mobile networks of objects which are perpetually developing. Parcels of information that move from a computer to a digital object behave in the same way as ants would do. They move through the network and pass from one knot to the next with the objective of arriving at their final destination as quickly as possible.[13]

Artificial pheromone system

Pheromone-based communication is one of the most effective ways of communication which is widely observed in nature. Pheromone is used by social insects such as bees, ants and termites; both for inter-agent and agent-swarm communications. Due to its feasibility, artificial pheromones have been adopted in multi-robot and swarm robotic systems. Pheromone-based communication was implemented by different means such as chemical [14][15][16] or physical (RFID tags,[17] light,[18][19][20][21] sound[22]) ways. However, those implementations were not able to replicate all the aspects of pheromones as seen in nature.

Using projected light was presented in an 2007 IEEE paper by Garnier, Simon, et al. as an experimental setup to study pheromone-based communication with micro autonomous robots.[23] Another study presented a system in which pheromones were implemented via a horizontal LCD screen on which the robots moved, with the robots having downward facing light sensors to register the patterns beneath them.[24][25]

Algorithm and formula

In the ant colony optimization algorithms, an artificial ant is a simple computational agent that searches for good solutions to a given optimization problem. To apply an ant colony algorithm, the optimization problem needs to be converted into the problem of finding the shortest path on a weighted graph. In the first step of each iteration, each ant stochastically constructs a solution, i.e. the order in which the edges in the graph should be followed. In the second step, the paths found by the different ants are compared. The last step consists of updating the pheromone levels on each edge.

```
procedure ACO_MetaHeuristic is
    while not terminated do
        generateSolutions()
        daemonActions()
        pheromoneUpdate()
    repeat
end procedure
```

Edge selection

Each ant needs to construct a solution to move through the graph. To select the next edge in its tour, an ant will consider the length of each edge available from its current position, as well as the corresponding pheromone level. At each step of the algorithm, each ant moves from a state x to state y, corresponding to a more complete intermediate solution. Thus, each ant k computes a set $A_k(x)$ of feasible expansions to its current state in each iteration, and moves to one of these in probability. For ant k, the probability p_{xy}^k of moving from state x to state y depends on the combination of two values,

the *attractiveness* η_{xy} of the move, as computed by some heuristic indicating the *a priori* desirability of that move and the *trail level* τ_{xy} of the move, indicating how proficient it has been in the past to make that particular move. The *trail level* represents a posteriori indication of the desirability of that move.

In general, the kth ant moves from state x to state y with probability

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{z \in \text{allowed}_x} (\tau_{xz}^\alpha)(\eta_{xz}^\beta)}$$

where τ_{xy} is the amount of pheromone deposited for transition from state x to y, $0 \leq \alpha$ is a parameter to control the influence of τ_{xy}, η_{xy} is the desirability of state transition xy (*a priori* knowledge, typically $1/d_{xy}$, where d is the distance) and $\beta \geq 1$ is a parameter to control the influence of η_{xy}. τ_{xz} and η_{xz} represent the trail level and attractiveness for the other possible state transitions.

Pheromone update

Trails are usually updated when all ants have completed their solution, increasing or decreasing the level of trails corresponding to moves that were part of "good" or "bad" solutions, respectively. An example of a global pheromone updating rule is

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_k^m \Delta\tau_{xy}^k$$

where τ_{xy} is the amount of pheromone deposited for a state transition xy, ρ is the *pheromone evaporation coefficient*, m is the number of ants and $\Delta\tau_{xy}^k$ is the amount of pheromone deposited by kth ant, typically given for a TSP problem (with moves corresponding to arcs of the graph) by

$$\Delta\tau_{xy}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

where L_k is the cost of the kth ant's tour (typically length) and Q is a constant.

Common extensions

Here are some of the most popular variations of ACO algorithms.

Ant system (AS)

The ant system is the first ACO algorithm. This algorithm corresponds to the one presented above. It was developed by Dorigo.[26]

Ant colony system (ACS)

In the ant colony system algorithm, the original ant system was modified in three aspects:

1. The edge selection is biased towards exploitation (i.e. favoring the probability of selecting the shortest edges with a large amount of pheromone);
2. While building a solution, ants change the pheromone level of the edges they are selecting by applying a local pheromone updating rule;
3. At the end of each iteration, only the best ant is allowed to update the trails by applying a modified global pheromone updating rule.[27]

Elitist ant system

In this algorithm, the global best solution deposits pheromone on its trail after every iteration (even if this trail has not been revisited), along with all the other ants. The elitist strategy has as its objective directing the search of all ants to construct a solution to contain links of the current best route.

Max-min ant system (MMAS)

This algorithm controls the maximum and minimum pheromone amounts on each trail. Only the global best tour or the iteration best tour are allowed to add pheromone to its trail. To avoid stagnation of the search algorithm, the range of possible pheromone amounts on each trail is limited to an interval $[\tau_{max},\tau_{min}]$. All edges are initialized to τ_{max} to force a higher exploration of solutions. The trails are reinitialized to τ_{max} when nearing stagnation.[28]

Rank-based ant system (ASrank)

All solutions are ranked according to their length. Only a fixed number of the best ants in this iteration are allowed to update their trials. The amount of pheromone deposited is weighted for each solution, such that solutions with shorter paths deposit more pheromone than the solutions with longer paths.

Parallel ant colony optimization (PACO)

An ant colony system (ACS) with communication strategies is developed. The artificial ants are partitioned into several groups. Seven communication methods for updating the pheromone level between groups in ACS are proposed and work on the traveling salesman problem. [29]

Continuous orthogonal ant colony (COAC)

The pheromone deposit mechanism of COAC is to enable ants to search for solutions collaboratively and effectively. By using an orthogonal design method, ants in the feasible domain can explore their chosen regions rapidly and efficiently, with enhanced global search capability and accuracy. The orthogonal design method and the adaptive radius adjustment method can also be extended to other optimization algorithms for delivering wider advantages in solving practical problems.[30]

Recursive ant colony optimization

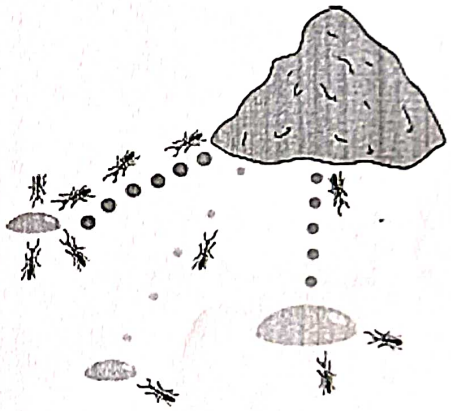
It is a recursive form of ant system which divides the whole search domain into several sub-domains and solves the objective on these subdomains.[31] The results from all the subdomains are compared and the best few of them are promoted for the next level. The subdomains corresponding to the selected results are further subdivided and the process is repeated until an output of desired precision is obtained. This method has been tested on ill-posed geophysical inversion problems and works well.[32]

Convergence

For some versions of the algorithm, it is possible to prove that it is convergent (i.e., it is able to find the global optimum in finite time). The first evidence of convergence for an ant colony algorithm was made in 2000, the graph-based ant system algorithm, and later on for the ACS and MMAS algorithms. Like most metaheuristics, it is very difficult to estimate the theoretical speed of convergence. A performance analysis of a continuous ant colony algorithm with respect to its various parameters (edge selection strategy, distance measure metric, and pheromone evaporation rate) showed that its performance and rate of convergence are sensitive to the chosen parameter values, and especially to the value of the pheromone evaporation rate.[33] In 2004, Zlochin and his colleagues[34] showed that COAC-type algorithms could be assimilated methods of stochastic gradient descent, on the cross-entropy and estimation of distribution algorithm. They proposed these metaheuristics as a "research-based model".

Applications

Ant colony optimization algorithms have been applied to many combinatorial optimization problems, ranging from quadratic assignment to protein folding or routing vehicles and a lot of derived methods have been adapted to dynamic problems in real variables, stochastic problems, multi-targets and parallel implementations. It has also been used to produce near-optimal solutions to the travelling salesman problem. They have an advantage over simulated annealing and genetic algorithm approaches of similar problems when the graph may change dynamically; the ant colony algorithm can be run continuously and adapt to changes in real time. This is of interest in network routing and urban transportation systems.



Knapsack problem: The ants prefer the smaller drop of honey over the more abundant, but less nutritious, sugar

The first ACO algorithm was called the ant system[26] and it was aimed to solve the travelling salesman problem, in which the goal is to find the shortest round-trip to link a series of cities. The general algorithm is relatively simple and based on a set of ants, each making one of the possible round-trips along the cities. At each stage, the ant chooses to move from one city to another according to some rules:

1. It must visit each city exactly once;
2. A distant city has less chance of being chosen (the visibility);
3. The more intense the pheromone trail laid out on an edge between two cities, the greater the probability that that edge will be chosen;
4. Having completed its journey, the ant deposits more pheromones on all edges it traversed, if the journey is short;