Tabu search

From Wikipedia, the free encyclopedia

Tabu search, created by Fred W. Glover in 1986[1] and formalized in 1989,[2][3] is a local search method used for mathematical optimization.

Local searches take a potential solution to a problem and check its immediate neighbors (that is, solutions that are similar except for one or two 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 these techniques by using 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 "taboo" ("tabu" being a different spelling of the same word) so that the algorithm does not consider that possibility repeatedly.

Contents

Basic Description

Tabu search is a metaheuristic local search algorithm that can be used for solving combinatorial optimization problems (problems where an optimal ordering and selection of options is desired - an example, the traveling salesman problem, is detailed below).

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 $N^*(x)$.

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 - n is also called the tabu tenure).

The memory structures used in tabu search can be divided into three categories[4]:

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

One example of an intermediate-term memory structure is one that prohibits solutions that contain certain attributes (e.g., solutions to the traveling salesman problem which include undesirable arcs) or a memory structure that prevents certain moves (e.g. an arc that was added to a TSP tour cannot be removed in the next n moves). Selected attributes in solutions recently visited are labeled "tabu-active." Solutions that contain tabu-active elements are banned.

Short-term memory alone may be enough to achieve solution superior to those found by conventional local search methods, but intermediate and long-term structures are often necessary for solving harder problems.[5] Intermediate and long-term structures primarily serve to intensify and diversify the search (the short-term memory also intensifies the search by temporarily locking in certain locally attractive attributes, i.e., those belonging to moves recently evaluated to be "good"[4]).

One major issue with Tabu Search is that it is only effective in discrete spaces.[6] It is rare that a search would visit the same real-value point in space multiple times, making a tabu list worthless. One possible solution is to apply a similarity measure and reject solutions that violate this similarity threshold.

Another problem with Tabu Search is that if the search space is very large or of high dimensionality, it is easy to remain within a small area of the search space. To work around this pitfall, some implementations of tabu search create a tabu list consisting of the attributes of a solution, rather than entire candidate solutions.[6] Tabu lists containing attributes (rather than whole solutions) can be more effective for some domains, although they raise a new problem. When a single attribute is marked as tabu, this typically results in more than one solution being tabu. Some of these solutions that must now be avoided could be of excellent quality and might not have been visited. To mitigate this problem, "aspiration criteria" are introduced: these override a solution's tabu state, thereby including the otherwise-excluded solution in the allowed set. A commonly used aspiration criterion is to allow solutions which are better than the currently-known best solution.

Tabu search is often benchmarked against other local search methods - such as Reactive search optimization, Hill climbing, Guided Local Search, or greedy randomized adaptive search procedure - or other metaheuristic methods - such as Simulated annealing, genetic algorithms, or Ant colony optimization algorithms.

Pseudo-code

The following pseudocode, adapted from,[7] presents the tabu search algorithm as described above. This implementation has the required short-term memory, but contains no intermediate or long-term memory structures.

```
1:   s ← s0
2:   sBest ← s
3:   tabuList ← null
4:   while (not stoppingCondition())
5:      candidateList ← null
```

```
6:      for(sCandidate in sNeighborhood)
7:           if(not containsTabuElements(sCandidate, tabuList))
8:                candidateList ← candidateList + sCandidate
9:           end
10:     end
11:     sCandidate ← LocateBestCandidate(candidateList)
12:     if(fitness(sCandidate) > fitness(sBest))
13:          sBest ← sCandidate
14:          tabuList ← featureDifferences(sCandidate, sBest)
15:          while(size(tabuList) > maxTabuListSize)
16:               ExpireFeatures(tabuList)
17:          end
18:     end
19:  end
20:  return(sBest)
```

Lines 1-3 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 an empty tabu list. 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 proper algorithm starts in line 4. 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. In line 5, an empty candidate list is initialized. The neighboring solutions are checked for tabu elements in line 7. If the solution does not contain elements on the tabu list, it is added to the candidate list (line 8).

The best candidate on the candidate list is chosen in line 11 (generally, solutions are evaluated according to a provided mathematical function, which returns a fitness score). If that candidate has a higher fitness value than the current best (line 12), it is set as the new best (line 13) and its features are added to the tabu list (line 14). At this point, if the tabu list is full (line 15), some elements will be allowed to expire (line 16). Generally, elements expire from the list in the same order they are added.

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 20).

Example: Traveling salesman problem

The traveling salesman problem (TSP) is often used to show the functionality of tabu search.[5] This problem poses a straightforward question - given a list of cities, is there a way to order that list to minimize the distance travelled while still visiting 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 to the TSP is an NP-hard task, heuristic-based approximation methods (such as local searches) are useful for devising close-to-optimal solutions.

Tabu search can be used to find a satisficing solution for the traveling salesman problem (that is, a solution that satisfies an adequacy criterion, rather than the absolutely optimal solution). First, the tabu 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 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 continue to be created until some stopping criteria, such as an arbitrary number of iterations, is met. Once the tabu search stops, the best solution returned is the solution with the shortest distance traveled while visiting all cities.

TS-PSO Algorithm

Particle swarm optimization (PSO) is a global search algorithm while TS is a local search algorithm. Zhang et al.[8] combined PSO and TS to create a novel hybrid TS-PSO algorithm which conducts both global search and local search in each iteration so that the probability of finding the optimal minima significantly increase.

References

1. ^ F. Glover and C. McMillan (1986). "The general employee scheduling problem: an integration of MS and AI". *Computers and Operations Research*.
2. ^ ***a b*** Fred Glover (1989). "Tabu Search - Part 1". *ORSA Journal on Computing* **1** (2): 190–206.
3. ^ Fred Glover (1990). "Tabu Search - Part 2". *ORSA Journal on Computing* **2** (1): 4–32.
4. ^ ***a b*** Fred Glover (1990). "Tabu Search: A Tutorial". *Interfaces*.
5. ^ ***a b*** M. Malek, M. Huruswamy, H. Owens, M. Pandya (1989). *Serial and parallel search techniques for the traveling salesman problem*.
6. ^ ***a b*** Sean Luke (2009). *Essentials of Metaheuristics* (http://cs.gmu.edu/~sean/book/metaheuristics/) . http://cs.gmu.edu/~sean/book/metaheuristics/.
7. ^ Clever Algorithms - Tabu Search (http://www.cleveralgorithms.com/nature-inspired/stochastic/tabu_search.html)
8. ^ Zhang, Yudong; Lenan Wu (2011). "A Hybrid TS-PSO Optimization Algorithm" (http://www.aicit.org/JCIT/ppl/%20JCIT_MAY_18.pdf) . *Journal of Convergence Information Technology* **6** (5). http://www.aicit.org/JCIT/ppl/%20JCIT_MAY_18.pdf.

External links

- Visualization of the Tabu search algorithm (Applet) (http://siebn.de/other/tabusearch/)
- Metaheuristic International Conference (MIC 2011) - Udine (http://mic2011.diegm.uniud.it/)
- The Reactive Search Community (http://www.reactive-search.org/)
- LION Conference on Learning and Intelligent Optimization techniques (http://www.intelligent-optimization.org/LION5/)

Categories: Optimization algorithms and methods
