International Journal of Information Technology and Knowledge Management
July-December 2008, Volume 1, No. 2, pp. 537-547

CHOOSING THE BEST HEURISTIC FOR A NP-PROBLEM

Narendhar Maaroju & Deepak Garg

Abstract

Nowadays computers are used to solve incredibly complex problems. But in order to manage a problem we should develop an algorithm. Sometimes the human brain is not able to accomplish this task. Moreover, exact algorithms might need centuries to solve a formidable problem. In such cases heuristic algorithms that find approximate solutions but have acceptable time and space complexity play indispensable role. In present, all known algorithms for NP-complete problems are requiring time that is exponential in the problem size. Heuristics are a way to improve time for determining an exact or approximate solution for NP-problems. In our paper we want to analyze what are the possible heuristics available for NP-problems and we explain the characteristics and performance of each heuristic. Finally we analyze efficient heuristic out of all available heuristics for NP-problem.

Keywords: Heuristic, NP-problems, Hill climbing, Simulated Annealing, Evolutionary Algorithms, Support Vector Machines, premature convergence.

1. Introduction

The most important among a variety of topics that relate to computation are algorithm validation, complexity estimation and optimization. Wide part of theoretical computer science deals with these tasks. Complexity of tasks in general is examined by studying the most relevant computational resources like execution time and space. The classification of problems that are solvable with a given limited amount of time and space into well-defined classes is a very intricate task, but it can help incredibly to save time and money spent on the algorithms design.

Modern problems tend to be very intricate and relate to analysis of large data sets. Even if an exact algorithm can be developed its time or space complexity may turn out to be unacceptable. But in reality it is often sufficient to find an approximate or partial solution. Such admission extends the set of techniques to cope with the problem. We discuss heuristic algorithms which suggest some approximations to the solution of optimization problems. In such problems the objective is to find the optimal of all possible solutions that is one that minimizes or maximizes an objective function. The objective function is a function used to evaluate a quality of the generated solution. Many real-world issues are easily stated as optimization problems.

The collection of all possible solutions for a given problem can be regarded as a search space, and optimization algorithms, in their turn, are often referred to as search algorithms. Approximate algorithms entail the interesting issue of quality estimation of the solutions they find. Taking into account that normally the optimal solution is unknown, this problem can be a real challenge involving strong mathematical analysis. In connection with the quality issue the goal of the heuristic algorithm is to find as good solution as possible for all instances of the problem. There are general heuristic strategies that are successfully applied to manifold problems.

The term *heuristic* is used for algorithms, which find solutions among all possible ones, but they do not guarantee that the best will be found; therefore they may be considered as approximate and not accurate algorithms. These algorithms, usually find a solution close to the best one and they find it fast and easily. Sometimes these algorithms can be accurate, that is they actually find the best solution, but the algorithm is still called heuristic until this is proven for all problem instances. The method used from a heuristic algorithm is one of the known methods, such as greediness, but in order to be easy and fast the algorithm ignores or even suppresses some of the problem's demands.

2. Heuristic Algorithms

A heuristic algorithm [5] is an algorithm that using a strategy that does not examine all possible solutions to a problem. Heuristic algorithms make no attempt to find the perfect solution to the problem. Instead, heuristic algorithms look for a "good enough" solution in an acceptable amount of time. A heuristic algorithm is one that will provide a solution close to the optimal, but may or may not be optimal. The concept of heuristic solutions to problems normally solved via non-polynomial time algorithms has changed the way programmers regard NP and NP-Complete problems.

Two fundamental goals in computer science are finding algorithms with provably good run times and with provably good or optimal solution quality. A *Heuristic* is an algorithm that abandons one or both of these goals; for example, it usually finds pretty good solutions, but there is no proof the solutions could not get arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case.

The principal advantages of heuristic algorithms are that such algorithms are (often) *conceptually simpler* and (almost always) much *cheaper computationally* than optimal algorithms.

3. Heuristic Techniques

3.1. Hill-Climbing

Hill climbing [4] is a variant of generate-and-test in which feedback from the test procedure is used to help the generator decide which direction to move in the search space. In a pure generate-and-test procedure, the test function responds with only a yes or no. But if the test function is augmented with a heuristic function that provides an estimate of how close a given state is to a goal state.

There are basically two types of hill climbing techniques are available. The first one is *simple hill climbing*, and the second one is *steepest-ascent hill climbing*. A useful variation on *simple hill climbing* considers all the moves from the current state and selects the best one as the next state, where as in the *steepest-ascent hill climbing* (or *gradient search*) the first state that is better than the current state is selected.

Hill climbing is not always very effective. It is particularly un suited to problems where the value of the heuristic function drops off suddenly as we move away from a solution. Hill climbing is a local method, by which we mean that it decides what to do next by looking only at the "immediate" consequences of its choice rather than by exhaustively exploring all the consequences. Hill-climbing algorithm is effective, but it has a significant drawback called *premature convergence*. Since it is "greedy", it always finds the nearest local optima of low quality. The goal of modern heuristics is to overcome this disadvantage.

Premature Convergence

When a genetic algorithms population converges to something, which is not the solution, we wanted.

3.2. Simulated Annealing

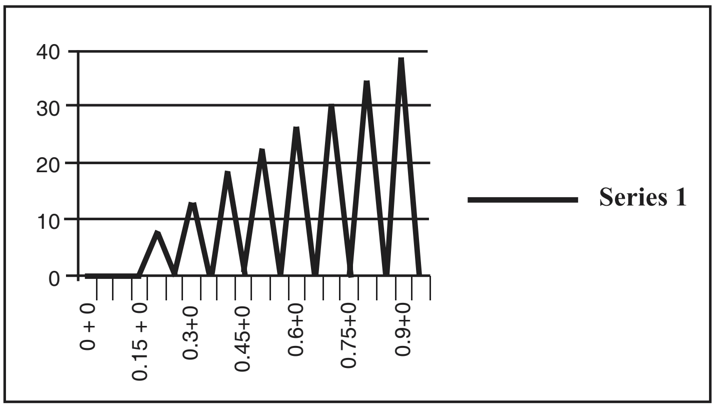
Simulated annealing[4] can be defined as a technique to find a good solution to an optimization problem by trying random variations of the current solution. A simulated annealing is a variation of hill climbing in which, at the begging of the process, some downhill moves may be made. In order to be compatible with standard usage in discussions of simulated annealing, in which we use the term *objective function* instead of *heuristic function*. Similar to the hill climbing method, if the performance of the best performing neighbour is better than the performance of the base sequence, then that neighbour is selected as the base sequence for the next iteration. However, if the current iteration is not able to find a neighbour performing better than the base sequence, the algorithm can still migrate to the best neighbour based on its current *temperature*. Simulated annealing [Kirkpatrick *et al*., 1983] as a computational process is patterned after the physical process of annealing, in which physical substances such as metals are melted(i.e., raised to high energy levels) and then gradually cooled until some solid state is reached. The goal of this process is to produce a minimal energy final state. The worse solution is generally accepted with a probability based on the Boltzmann probability distribution:

$$P = \mathrm{Exp}\,(-\Delta E/kT)$$

Where ΔE = (value of current)–(value of new state) is the positive change in the energy level, T is the temperature, and k is the Boltzmann's constant.

An important component of the simulated annealing algorithm is the *annealing schedule*, which determines the initial temperature and how it is lowered from high to low values. Basically an annealing schedule has three components. The first is the initial value to be used for temperature. The second is the criteria that will be used to decide when the temperature of the system should be reduced. The third is the amount by which the temperature will be reduced each time it is changed.

However, as expected, the number of steps to a local minimum during each iteration for each runs increases with increase in the initial temperature and the annealing schedule step. As the starting temperature and annealing schedule step are increased, the algorithm accepts more poorly performing solutions before halting. However, this increase in the number of steps to local optimal does not translate into any significant performance improvement for our experiments.



Temp + step

Figure 1: Increase in the Number of Steps to Local Minimum with Increases in Initial Temperature and Annealing Schedule Step

3.3. Tabu Search

Tabu search [5] extends the idea to avoid local optima by using memory structures. The problem of simulated annealing is that after "jump" the algorithm can simply repeat its own track. Tabu search prohibits the repetition of moves that have been made recently.

Hillier and Lieberman outlined the tabu search stopping criterion by, for example, using a fixed number of iterations, a fixed amount of CPU time, or a fixed number of consecutive iterations without an improvement in the best objective function value.

Also stop at any iteration where there are no feasible moves into the local neighborhood of the current trial solution[21,22]. Tabu Search includes

(i) Dealing with an objective function that isdifficult to evaluate.

(ii) Dealing with constraints

(iii) Probabilistic selection of candidate solutions.

(iv) Variations of the tabu mechanism/list.

(v) More sophisticated versions of aspiration criteria.

(vi) Frequency – based memory

(vii) Dealing with continuous variables

3.4. Swarm Intelligence

Swarm Intelligence was introduced in 1989. It is an artificial intelligence technique, based on the study of collective behaviour in decentralized, self-organized, systems. Two of the most successful types of this approach are Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). The main advantage of swarm intelligence techniques[6] is that they are impressively resistant to the local optima problem.

The typical swarm intelligence system has the properties that It is composed of many individuals. The individuals are relatively homogeneous. The interactions among the individuals are based on simple behavioural rules that exploit only local information that the individuals exchange directly or via the environment.

Ant colony optimization or ACO is a heuristic optimization algorithm that can be used to find approximate solutions to difficult combinatorial optimization problems. In ACO artificial ants build solutions by moving on the problem graph and they, mimicking real ants, deposit artificial pheromone on the graph in such a way that future artificial ants can build better solutions. ACO has been successfully applied to an impressive number of optimization problems. Particle swarm optimization or PSO is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an *n*-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after each time step. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large number of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

3.5. Evolutionary Algorithms

Evolutionary algorithms, as the name implies, are a class of heuristics that emulate natural evolutionary processes. Sometimes the adjective "genetic" is used in lieu of "evolutionary". Evolutionary Algorithms succeed in tackling *premature convergence* by considering a number of solutions simultaneously. In artificial intelligence, an Evolutionary Algorithm (EA)[7, 8] is a subset of evolutionary computation, a generic population-based heuristic optimization algorithm. An EA uses some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Evolution of the population then takes place after the repeated application of the above operators. It is easy to see that this scheme falls in the category of generate-and-test algorithms. the evolution (fitness) function represents a heuristic estimation of solution quality and the search process is driven by the variation and the selection operators. Evolutionary Algorithms (EA) posses a number of features that can help to position them within in the family of generate-and-test methods:

- EAs are population based, i.e., they processed a whole collection of candidate solutions simultaneously,
- EAs are mostly use recombination to mix information of more candidate solutions into a new one,
- EAs are stochastic,

3.6. Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well[9].

The following table lists the neural network types supported by the *Neural Networks* package along with their typical usage.

Other advantages include:

(1) Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

Network type	*Typical use(s) of the network*
Radial basis function	function approximation, classification, dynamic systems modelling
Feedforward	function approximation, classification, dynamic systems modelling
Dynamic	dynamic systems modelling, time series
Hopfield	classification, auto-associative memory
Perceptron	classification
Vector quantization	classification
Unsupervised	clustering, self-organizing maps, Kohonen networks

(2) Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.

(3) Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

(4) Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

In examining the data for a classification problem, some reasonable questions to ask may include the following:

- Are all classes equally represented by the data?
- Are there any outliers, that is, data samples dissimilar from the rest?

Answers to these questions might reveal potential difficulties in using the given data for training. If so, new data may be needed.

4. Analysis and Results

This paper on "**Choosing the best Heuristic for a NP-Problem**" describes various concepts, arguments and applications related to the algorithms, complexity classes and heuristics. In this paper we analyzes the various classes of problems based on the complexity. And in section 4, we explained the basic concepts of Heuristics, different Heuristics strategies for NP-problems.

The following tables illustrate the performance of different heuristics for different kind of NP-problems.

Table 1
Travelling Salesman Problem[14, 15]

Heuristic strategy	
Hill climbing	Not effective,
	Because no agenda is maintained.
Simulated annealing	1. Current solution wandering from neighbour to neighbour as the computation proceeds.
	2. Examines neighbours in random order.
	3. Schema leaves several operations and definitions unspecified.
	4. As the temperature goes down, the probability of accepting bad moves decreases.
Swarm intelligence	1. Not tested
Tabu search	1. Implementation of tabu search degrades substantially as N increases.
	2. Only makes uphill moves when it is stuck in local optima
Evolutionary algorithms	1. without the local optimization
Neural networks	1. Multiple random starts were allowed.
	2. Best solution they ever found on such an instance was still more than17% above optimal.
	3. very sensitive
	4. N^2 neurons are required

Table 2
Time Table Problem [21, 22, 23]

Hill climbing	1. Not effective,
	Because no agenda is maintained.
	2. It can easily be verified the search space for this kind of problem is very large
	3. The best solution within a reasonable small amount of time depends on neighbourhoods.
Simulatedannealing	1. In our experiments we decided to use a reduction factor of 0.9 and an initial acceptance probability of 0.8 to cool down quite slow
	2. Current solution wandering from neighbour to neighbour as the computation proceeds.
	3. Examines neighbours in random order.
	4. Schema leaves several operations and definitions unspecified.
	5. The cooling factor decrease Factor is set to 0.9.
Swarm intelligence	1. Not tested
Tabu search	1. TS is able to find better solution until the end of the computation
	2. Implementation of tabu searchdegrades substantially as N increases.
	3. Only makes uphill moves when it is stuck in local optima
	4. The best regular tabu list length seems to be approx. 40 elements

Evolutionary algorithms	1. without the local optimization 2. EAs give lower total penalties compared with man-made schedules. 3. The best individual of a generation will survive and 5% of the individuals. 4. Every resource list of the individual is subject to mutation with a probability of 0.5%.
Neural networks	1. Multiple random starts were allowed. 2. Best solution they ever found on such an instance was still more than17% above optimal. 3. very sensitive 4. N^2 neurons are required

Table 3
Generation Expansion Problem [16,17]

Heuristic strategy	
Hill climbing	1. Not tested
Simulatedannealing	1. SA iteratively searches the neighbour by adding some random number with the current solution. The best solution is readily accepted. The worst solution is also accepted by comparing with a random number (0, 1), which avoids trapping in local minima. 2. In each step, the algorithm picks a *random* move. If it improves the objective function ($\Delta E > 0$), it is accepted. Otherwise, the *bad* move is only accepted with a probability $e^{\Delta E/T}$
Swarm intelligence(ACO)	1. Then ants are placed randomly in the first stage and allowed to move based on the probability. After the ants completed the tour, the objective function and fitness function values for the individuals are calculated.
Tabu search	1. This reduces the size of neighbourhood. Then the combinations between the capacities should be taken as the neighbours keeping other two stages unaltered. Similarly for the other two stages, the neighbours are determined. The candidate list is formed with the combination of these neighbors. The best neighbour among the candidate list is moved to the "*Tabu list*" for a pre-specified number of generations.
Evolutionary algorithms	1. The uniform binary window and head-to-head crossover have SR of 60%, whereas the arithmetic crossover has 30%. The stochastic crossover randomly selects one of the three crossover strategiesmentioned above and has an SR of 72%. 2. without the local optimization.
Neural networks	1. Multiple random starts were allowed. 2. Best solution they ever found on such an instance was still more than17% above optimal. 3. very sensitive.

Table 4
Vertex Cover Problem

Hill climbing	Not effective,
	Because no agenda is maintained.
Simulatedannealing	1. Current solution wandering from neighbour to neighbour as the computation proceeds.
	2. Examines neighbours in random order.
	3. schema leaves several operations and definitions unspecified
Swarm intelligence	1. Not tested
Tabu search	1. Implementation of tabu searchdegrades substantially as *N* increases.
	2. Only makes uphill moves when it is stuck in local optima
Evolutionary algorithms	1. without the local optimization.
	2. The algorithm stops when the population reaches a stable state.
Neural networks	1. Multiple random starts were allowed.
	2. Best solution they ever found on such an instance was still more than17% above optimal.
	3. very sensitive
	4. N^2 neurons are required

6. CONCLUSION

After analysing the different heuristics for some well-known problems, we conclude that, based on the problem characteristics different heuristics are efficient for different problems. We can't say particular heuristic is efficient for all NP-problems. Based on problem criteria and characteristics one of the heuristic is efficient.

REFERENCES

[1] S. A. Cook. "An Overview of Computational Complexity", in *Communication of the ACM*, **26**, (6), (June 1983), 401–408.

[2] T. Cormen, Ch. Leiserson, R. Rivest. *Introduction to Algorithms*. MIT Press, (1989).

[3] M. R. Garey, D. S. Johnson. *Computers and Intractability*. Freeman & Co, (1979).

[4] Prasad A. Kulkarni, David B. Whalley, Gary S. Tyson, Jack W. Davidson" Evaluating Heuristic Optimization Phase Order Search Algorithms". *International Symposium on Code Generation and Optimization*, (2007), 157–169.

[5] M. E. Aydin, T. C. Fogarty. "A Distributed Evolutionary Simulated Annealing Algorithm for Combinatorial Optimization Problems", in *Journal of Heuristics*, **24**, (10), (Mar. 2004), 269–292.

[6] R. Battiti. "Reactive Search: Towards Self-tuning Heuristics", in *Modern Heuristic Search Methods*. Wiley & Sons, (1996), 61–83.

[7] E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, ser. Santa Fe Institute Studies in the Sciences of Complexity. New York, N.Y.: Oxford University Press, (1999).

[8] X. Wu, B. S. Sharif, O. R. Hinton. "An Improved Resource Allocation Scheme for Plane Cover Multiple Access Using Genetic Algorithm", in *IEEE Transactions on Evolutionary Computation*, **9**, (1), (Feb. 2005), 74–80.

[9] F. Divina, E. Marchiori. "Handling Continuous Attributes in an Evolutionary Inductive Learner", in *IEEE Transactions on Evolutionary Computation*, **9**, (1), Feb. 2005, 31–43.

[10] Lawrence, S., Giles, C. L, Tsoi, A. C., Back, A. D. Face Recognition: A Convolutional Neural Network Approach, *IEEE Transactions on Neural Networks, Special Issue on Neural Networks and Pattern Recognition*, **8**(1), (1997), 98–113.

[11] S. S. Keerthi. Efficient Turing of SVM Hyperparameters using Radius/Margin Bound and Iterative Algorithms. *IEEE Transactions on Neural Networks*, **13**(5), (2002), 1225–1229.

[12] S. Gunn. "Support Vector Machines for Classification and Regression", *Technical Report*, (May 1998).

[13] S. Pang, D. Kim, S. Y. Bang. "Face Membership Authentication Using SVM Classification Tree Generated by Membership-Based LLE Data Partition", in *IEEE Transactions on Neural Networks*, **16**, (2), (Mar. 2005), 436–446.

[14] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem*. Wiley, Chichester, (1985).

[15] B. L. Golden and W. R. Stewart, ''Empirical Analysis of Heuristics,'' in *The Traveling Salesman Problem* (eds.), John Wiley & Sons, Chichester, (1985), 207–249.

[16] Kirkpatrick, S., Jr., C. G., & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science,* **220.**

[17] Y.-M. Park, J.-R. Won, J.-B. Park, and D.-G. Kim, "Generation Expansion Planning based on an Advanced Evolutionary Programming," *IEEE Trans. Power Syst.*, **14**, (1), (Feb. 1999), 299–305.

[18] J. Zhu and M.-Y. Chow, "A Review of Emerging Techniques on Generation Expansion Planning," *IEEE Trans. Power Syst.*, **12**, (4), (Nov. 1997), 1722–1728.

[19] G. M. White and B. S. Xie, "Examination Timetables and Tabu Search with Longer Term Memory," *Proc. of the International Conference on the Practice and Theory of Timetabling*, (2000), 184–201.

[20] M. L. Ng, *et al*, "Solving Constrained Staff Workload Scheduling Problems using Simulated Annealing Technique," *Proc. of the International Conference on the Practice and Theory of Timetabling*, (2000), 355–371.

[21] F. Glover, ''Tabu Search – Part I, II'' *ORSA J. Comput.* **1** (1989), (190–206).

Narendhar Maaroju & Deepak Garg
Computer Science and Engineering Department
Thapar University, Patiala