An introduction to heuristic algorithms

Natallia Kokash

Department of Informatics and Telecommunications
University of Trento, Italy
email: kokash@dit.unitn.it

Abstract. Nowadays computers are used to solve incredibly complex problems. But in order to manage with 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 manage with formidable challenges. In such cases heuristic algorithms that find approximate solutions but have acceptable time and space complexity play indispensable role. In this paper heuristics, their areas of application and the basic underlying ideas are surveyed. We also describe in more detail some modern heuristic techniques, namely Evolutionary Algorithms and Support Vector Machines.

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 studying the most relevant computational resources like execution time and space. The ranging 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. A vast collection of papers were dedicated to algorithm development. A short historical overview of the fundamental issues in theory of computation can be found in [1]. We do not discuss precise definition of algorithm and complexity. The interested reader can apply for the information to one of the fundamental books on theory of algorithms, e.g. [2], [3].

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 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 paper is organized as follows. Section 2 presents some essential information about algorithms and computational complexity. Section 3 describes prevalent heuristic techniques, Support Vector Machines and Evolutionary Algorithms are presented. Some intractable problems that could help to understand deeper importance of heuristics are also mentioned. Finally, the last section is devoted to the conclusion.

2 Algorithms and complexity

It is difficult to imagine the variety of existing computational tasks and the number of algorithms developed to solve them. Algorithms that either give nearly the right answer or provide a solution not for all instances of the problem are called *heuristic algorithms*. This group includes a plentiful spectrum of methods based on traditional techniques as well as specific ones. For the beginning we sum up the main principles of traditional search algorithms.

The simplest of search algorithms is **exhaustive search** that tries all possible solutions from a predetermined set and subsequently picks the best one.

Local search is a version of exhaustive search that only focuses on a limited area of the search space. Local search can be organized in different ways. Popular *hill-climbing* techniques belong to this class. Such algorithms consistently replace the current solution with the best of its neighbors if it is better than the current. For example, heuristics for the problem of intragroup replication for multimedia distribution service based on Peer-to-Peer network is based on hill-climbing strategy [4].

Divide and conquer algorithms try to split a problem into smaller problems that are easier to solve. Solutions of the small problems must be combinable to a solution for the original one. This technique is effective but its use is limited because there is no a great number of problems that can be easily partitioned and combined in a such way.

Branch-and-bound technique is a critical enumeration of the search space. It enumerates, but constantly tries to rule out parts of the search space that cannot contain the best solution.

Dynamic programming is an exhaustive search that avoids re-computation by storing the solutions of subproblems. The key point for using this technique is formulating the solution process as a recursion.

A popular method to construct successively space of solutions is **greedy** technique, that is based on the evident principle of taking the (local) best choice at each stage of the algorithm in order to find the global optimum of some objective function.

Usually heuristic algorithms are used for problems that cannot be easily solved. Classes of time complexity are defined to distinguish problems according to their "hardness". Class **P** consists of all those problems that can be solved on a deterministic Turing machine in polynomial time from the size of the input. Turing machines are an abstraction that is used to formalize the notion of algorithm and computational complexity. A comprehensive description of them can be found in [3]. Class **NP** consists of all those problems whose solution can be found in polynomial time on a non-deterministic Turing machine. Since such a machine does not exist, practically it means that an exponential algorithm can be written for an NP-problem, nothing is asserted whether a polynomial algorithm exists or not. A subclass of NP, class **NP-complete** includes problems such that a polynomial algorithm for one of them could be transformed to polynomial algorithms for solving all other NP problems. Finally, the class **NP-hard** can be understood as the class of problems that are NP-complete or harder. NP-hard problems have the same trait as NP-complete problems but they do not necessary belong to class NP, that is class NP-hard includes also problems for which no algorithms at all can be provided.

In order to justify application of some heuristic algorithm we prove that the problem belongs to the classes NP-complete or NP-hard. Most likely there are no polynomial algorithms to solve such problems, therefore, for sufficiently great inputs heuristics are developed.

3 Heuristic techniques

Branch-and-bound technique and dynamic programming are quite effective but their time-complexity often is too high and unacceptable for NP-complete tasks. 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.

Simulated annealing algorithm [5], invented in 1983, uses an approach similar to hill-climbing, but occasionally accepts solutions that are worse than the current. The probability of such acceptance is decreasing with time.

Tabu search [6] 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.

Swarm intelligence [7] was introduced in 1989. It is an artificial intelligence technique, based on the study of collective behavior in decentralized, self-organized, systems. Two of the most successful types of this approach are *Ant Colony Optimization* (ACO) and *Particle Swarm Optimization* (PSO). In ACO artificial ants build solutions by moving on the problem graph and changing it in such a way that future ants can build better solutions. PSO deals with problems in which a best solution can be represented as a point or surface in an n-dimensional space. The main advantage of swarm intelligence techniques is that they are impressively resistant to the local optima problem.

Evolutionary Algorithms succeed in tackling premature convergence by considering a number of solutions simultaneously. Later we discuss this group of algorithms more elaborately.

Neural Networks are inspired by biological neuron systems. They consist of units, called neurons, and interconnections between them. After special training on some given data set Neural Networks can make predictions for cases that are not in the training set. In practice Neural Networks do not always work well because they suffer greatly from problems of *underfitting* and *overfitting* [8]. These problems correlate with the accuracy of prediction. If a network is not complex enough it may simplify the laws which the data obey. From the other point of view, if a network is too complex it can take into account the noise that usually assists at the training data set while inferring the laws. The quality of prediction after training is deteriorated in both cases. The problem of premature convergence is also critical for Neural Networks.

Support Vector Machines (SVMs) extend the ideas of Neural Networks. They successfully overcome premature convergence since convex objective function is used, therefore, only one optimum exists. Classical divide and conquer technique gives elegant solution for separable problems. In connection with SVMs, that provide effective classification, it becomes an extremely powerful instrument. Later we discuss SVM classification trees, which applications currently present promising object for research.

Description and comparative analysis of simulated annealing, tabu search, neural networks and evolutionary algorithms can be found in [9].

3.1 Evolutionary algorithms

Evolutionary algorithms are methods that exploit ideas of biological evolution, such as *reproduction*, *mutation* and *recombination*, for searching the solution of an optimization problem. They apply the principle of survival on a set of potential solutions to produce gradual approximations to the optimum. A new set of approximations is created by the process of selecting individuals according to their objective function, which is called fitness for evolutionary algorithms, and breeding them together using operators inspired from genetic processes. This process leads to the evolution of populations of individuals that are better suited to their environment than their ancestors.

The main loop of evolutionary algorithms includes the following steps:

1. Initialize and evaluate the initial population.
2. Perform competitive selection.
3. Apply genetic operators to generate new solutions.
4. Evaluate solutions in the population.
5. Start again from point 2 and repeat until some convergence criteria is satisfied.

Sharing the common idea, evolutionary techniques can differ in the details of implementation and the problems to which they are applied. *Genetic programming* searches for solutions in the form of computer programs. Their fitness

is determined by the ability to solve a computational problem. The only difference from *evolutionary programming* is that the latter fixes the structure of the program and allows their numerical parameters to evolve. *Evolution strategy* works with vectors of real numbers as representations of solutions, and uses self-adaptive mutation rates.

The most successful among evolutionary algorithms are *Genetic Algorithms* (GAs). They have been investigated by John Holland in 1975 and demonstrate essential effectiveness. GAs are based on the fact that the role of mutation improves the individual quite seldom and, therefore, they rely mostly on applying recombination operators. They seek solutions of the problems in the form of strings of numbers, usually binary.

The prevalent area for applying genetic algorithms are optimization problems requiring large scale high performance computing resources. For instance, the problem of effective resource allocation in conjunction with Plane Cover Multiple-Access (PCMA) scheme has been examined in [10]. Its objective is to maximize the attainable capacity of packet-switching wireless cellular networks. The main issue to consider for resource allocation is to minimize the number of Unit of Bandwidth (UB) that should be allocated. The problem has been proven to be in the class NP-hard. Authors applied genetic algorithm instead of greedy search used before. Computer simulation has been performed for the idealized cellular system with one base station that can support m connections required one UB per second, constructed in a cluster of B cells. As result, it is stated that the genetic algorithm can improve the system capacity utilization.

Substantial increase of the consumer demand on computers and mobile phones made network optimization problems extremely relevant. Being general technique that can be easily modified under the various conditions, evolutionary algorithms are widely used in this area.

To give an example let us mention Adaptive Mesh Problem (AMP), that has a goal to minimize the required number of base stations of cellular network to cover a region. Like the previous discussed problem, AMP is NP-hard. One of the evolutionary techniques, called *Hybrid Island Evolutionary Strategy* (HIES) has been applied to tackle this problem [11]. It represents evolutionary algorithm borrowing features from two types of genetic algorithms, namely *island model GA* and *fine-grained or cellular GA*. The original problem has been transformed into a geometric meshing generation problem and specific genetic operators, *micromutation*, *macromutation* and *crossover*, have been altering an array of hexagonal cells. Initially regular honeycomb was transformed to irregular mesh that fits better to the real-life conditions. The desired result, the reduction of the number of base stations, has been achieved.

Some other examples of evolutionary algorithms can be found in [12], which is devoted to machine learning theory problems.

3.2 Support Vector Machines

Statistical learning theory concerns the problem of choosing desired functions on the basis of empirical data. Its fundamental problem is generalization, which

consists in inferring laws for future observations, given only a finite amount of data. Support Vector Machines is the most prominent approach among modern results in this field.

The basic principles of SVMs have been developed by Vapnik in 1995 [13]. Owing to their attractive qualities they immediately gained a wide range of applications. SVMs diverged from *Empirical Risk Minimization* (ERM) principle embodied by conventional neural networks, which minimizes the error on the training data, and use *Structural Risk Minimization* (SRM) principle, which minimizes an upper bound on the expected risk [14]. SVMs support classification and regression tasks based on the concept of optimal separator.

The *classification problem* can be stated as a problem of data set separation into classes by the functions which are induced from available instances. We will refer to such functions as *classifiers*.

In a *regression task* we have to estimate the functional dependence of the dependent variable y on a set of independent variables x. It is assumed that the relationship between the independent and dependent variables is given by a deterministic function f with some additive noise.

Consider the problem of separating the set of vectors belonging to two classes,

$$\{(x^1, y^1), ..., (x^l, y^l)\},\ x \in R^n, y \in \{-1, 1\},$$

with a hyperplane,

$$\langle w, x \rangle + b = 0,$$

where w, b are parameters, $\langle w, x \rangle$ denotes inner product (fig. 1).

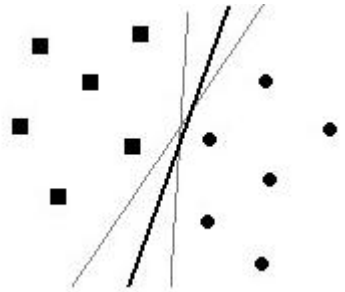

Fig. 1. Classification problem

The objective is to separate classes by the hyperplane without errors and maximize the distance between the closest vector to the hyperplane. Such a hyperplane is called *optimal separating hyperplane*. According to the results [14], the optimal separating hyperplane minimizes

$$\Phi(w) = \frac{1}{2}\|w\|^2, \tag{1}$$

under the constraints

$$y^i[\langle w, x\rangle + b] \geq 1. \tag{2}$$

The solution of optimization problem (1), (2) is given by the saddle point of the Lagrangian functional. Points that have non-zero Lagrange multipliers are called *Support Vectors* and they are used to describe the generated classifier. Usually the support vectors is a small subset of the training data set. This fact provides the main attractive quality of the SVMs, namely low computational complexity.

In the case when the training data is not linearly separable there are two approaches. The first is to introduce an additional function associated with misclassification. Another one is to use a more complex function to describe the boundary. The optimization problem in general is posed to minimize the classification error as well as the bound on the VC dimension of the classifier. The *VC dimension* of a set of functions is p if exists a set of p points, such that these points can be separated in all 2^p possible configurations using this functions, and there is no set of q points, $q > p$, satisfying this property.

Whole complex and vast theory of SVMs cannot be covered in this article. A lot of methods based on the ideas of SVMs have been developed in the last years. Among them is SVM classification tree algorithm, successfully applied in text and image classification.

A classification tree consists of internal and external nodes connected by branches. Each internal node performs a split function that partitions a training data into two disjoint subsets and each external node contains a label indicating the predicted class of a given feature vector. Recently this method has been applied to deal with the classification complexity of membership authentication problem [15], a typical issue in digital security schemes. Its objective is to distinguish membership class M from the nonmembership class $G - M$ in the human group G. An SVM classifier is trained using two partitioned subgroups, and, finally, the trained SVM tree is used for identifying the membership of an unknown person. Experimental results have shown that the proposed method has a better performance and robustness than previous approaches.

4 Conclusion

This paper has presented an overview of heuristics, that are approximate techniques to solve optimization problems. Usually heuristic algorithms are developed to have low time complexity and applied to the complex problems. We briefly defined basic traditional and modern heuristic strategies. Evolutionary algorithms and Support Vector Machines were considered more comprehensively. Due to their eminent characteristics they gained a great popularity. Recently appeared research results confirm the fact that their applications can be significantly enlarged in the future.

The current paper does not pretend to be complete. It would be interesting to carry out a more profound survey of heuristics, compare implementation complexity and accuracy of the different approximate algorithms. But this task

cannot be easily accomplished because of the enormous bulk of information. We even did not touch upon such a prominent area for heuristic algorithms as planning and scheduling theory. But we hope that our work makes clear the extreme importance of heuristics in modern computer science.

References

1. S. A. Cook. "An overview of computational complexity", in *Communication of the ACM*, vol. 26, no. 6, June 1983, pp. 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. Z. Xiang, Q. Zhang, W. Zhu, Z. Zhang, Y. Q. Zhang. "Peer-to-Peer Based Multimedia Distribution Service", in *IEEE Transactions on Multimedia*, vol. 6, no. 2, Apr. 2004, pp. 343–355.
5. M. E. Aydin, T. C. Fogarty. "A Distributed Evolutionary Simulated Annealing Algorithm for Combinatorial Optimization Problems", in *Journal of Heuristics*, vol. 24, no. 10, Mar. 2004, pp. 269–292.
6. R. Battiti. "Reactive search: towards self-tuning heuristics", in *Modern heuristic search methods*. Wiley&Sons, 1996, pp. 61-83.
7. R. Eberhart, Y. Shi, and J. Kennedy. Swarm intelligence. Morgan Kaufmann, 2001.
8. B. Kröse, P. Smagt. An introduction to Neural Networks. University of Amsterdam, Nov. 1996.
9. D. Karaboga, D. Pham. Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks. Springer Verlag, 2000.
10. 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*, vol. 9, no. 1, Feb. 2005, pp.74–80.
11. J.C. Crput, A. Koukam, T. Lissajoux, A. Caminada. "Automatic Mesh Generation for Mobile Network Dimensioning Using Evolutionary Approach", in *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 1, Feb. 2005, pp. 18–30.
12. F. Divina, E. Marchiori. "Handling Continuous Attributes in an Evolutionary Inductive Learner", in *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 1, Feb. 2005, pp. 31–43.
13. V. N. Vapnik. The Nature of Statistical Learning Theory. Springer Verlag, 1995.
14. S. Gunn. "Support Vector Machines for Classification and Regression", Technical Report, May 1998, http://www.ecs.soton.ac.uk/∼srg/publications/pdf/SVM.pdf.
15. 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*, vol. 16, no 2, Mar. 2005, pp. 436–446.