The Applications of NP-hardness optimizations problem

Ahmed ALRIDHA1*, Abbas Musleh Salman2, Dr Ahmed Sabah Al-Jilawi 3

1,2 Ministry of Education, General Directorate of Education in Babylon.
3Department of Mathematics -University of Babylon.

Emails: amqa92@yahoo.com, abbas.mmm2019@gmail.com, aljelawy2000@yahoo.com

Abstract. In this paper we present a study on the problem of NP hardness and their applications in computer science. In addition, our study sheds light on the most important applications present in our daily life and how the problem of NP stiffness is an important primary focus in it. Moreover, the aim was to search for further modifications in order to obtain optimal methods for their adoption. Finally, the aim of this study is to seek to solve the decision problem of NP hardness to achieve the desired goals with the optimal result.

Keywords: Combinatorial Optimization, K-cluster problem, Optimization algorithm, Graph theory, Semidefinite programming

1. Introduction

In our daily life, we face many situations and problems for which we may be able to provide effective solutions, and sometimes the optimal solution cannot be provided [1]. An example of such situations is the Rubik’s Cube or Sudoku puzzle, in which the solution can be found in the shortest time period with a stroke of luck. Here we can say that we were able to find a solution to the non-deterministic polynomial [2]. Hence the problems we face are classified according to the ability to find optimal solutions, where (P) represents polynomial problems that are easy to find solution, and NP-hard which is a non-deterministic polynomial. Also there are NP- hard and NP-complete set. Polynomials are functions of n, such as n2 or n3, elevated to various powers. And polynomial time means that every polynomial function of n will be approximately the time a computer takes to find the optimal solution to the problem if a problem contains n components. Decision problems can be considered in the P-class if at least one polynomial-time algorithm is available to solve the problem. In this case, the solution time of the algorithm is bounded by a polynomial in n, where n represents the length of the input [4]. A decision problem is NP-complete if any other NP problem can be reduced in polynomial time by a deterministic Turing machine to the NP-complete problem [9]. Also, a problem, not necessarily a decision problem, is NP-hard if any other NP problem can be reduced in polynomial time by a deterministic Turing machine to the NP-hard problem. The NP-complete problems are the hardest problems in NP in that a polynomial-time algorithm to solve this kind of problem is unlikely to exist, unless P=NP . However, asymptotically faster algorithms for NP problems can exist if P=NP . NP problem have a short proof that the answer is ‘yes’”. However, the coNP complexity class consists of decision problems for which there is a short proof that the answer is ‘no’” (see Piotr Wojciechowski the com- plexity class coNP) [11]. Finally, the ultimate goal of optimization is to find the best solution to the problem in all fields such as engineering, physics, medicine ... etc.
2. The concept of NP-hardness

One of the key principles underlying the theory of computational complexity is that an algorithm can be considered efficient if and only if it runs in polynomial time. There are several books and articles with comprehensive coverage of NP-hardness, see e.g.,[13],[14]. The amount of steps it takes on inputs of size N is not greater than \( c_1 \cdot N^{c_2} \) where adequate constants are \( c_1 \) and \( c_2 \). Dijkstra's well-known algorithm for determining the shortest paths in a graph, for instance, makes \( c \cdot n^2 \) steps at most, where \( n \) is the graph's number of vertices; it is therefore an efficient algorithm. Besides, P denotes the set of decision problems (i.e., yes/no output problems) for which a polynomial time algorithm exists. For instance, the problem of whether a number (the input) is divisible by 3 is in P. A more advanced definition of the NP problem class. Basically, If this positive response can be effectively verified (i.e. in polynomial time) for all inputs for which the answer is 'yes', the decision problems are in NP.

3. Establishing NP-hardness

Proving that a problem T is NP-hard requires demonstrating a reduction to L from an established NP-hard problem [12]. In addition, showing that L is NP-complete requires demonstrating that it is in NP. Also, the latter step is relatively simple in many practical cases. (However it requires L to be a decision problem, while optimization problem can also be NP-hard. However most NP-hard optimization problem can easily be transformed into an NP-complete decision problem. The problem of finding the longest path in a graph, for example, can be turned into the following decision problem: the input is a graph and a number k and the question is if in the graph there is at least a path of length k.). Finally, many topics are proclaimed to be NP-hard or NP-complete in computer science papers, but few of these statements are actually proved. This is poor practice because at first glance, not anything that appears to be a difficult problem is really NP-hard; by proving it the only way to guarantee that a problem is NP-hard.
4. Misconceptions

It is possible to group most of the concepts I found into two categories: those relating to deciding that a problem is NP-hard and those relating to the effects of NP-hardness. And the first one deals with the word NP itself [15]:

- NP means non-polynomial.
- If the search space is exponential, then the problem is NP-hard.
- Adding more constraints to a problem makes it harder to compute.
- In particular, applying more limitations to an NP-hard problem results in an issue that is also NP-hard.
- Problems that are difficult to solve by an engineer in practice are NP-hard.
- If similar problems are NP-hard, then the problem at hand is also NP-hard.
- It is not possible to solve NP-hard problems optimally.

5. Classification

Now, in theoretical computer science, the classification and complexity of definitions of common problems have two major groups and are classified as follows [1, 2, 4]:

Easy → P
Medium → NP
Hard → NP Complete
Hardest → NP-Hard

In the diagram below, the relationship between them is explained:

![Diagram showing the relationship between classification of NP hardness](image)

**Figure 2.** The relationship between Classification of NP-Hardness.

6. Examples of NP-hardness problem

A common example of a problem can be represented as follows [5, 7]:

1. Max-Cut problem which has the form

\[
\text{(Max-Cut) maximize } \sum_{ij} w_{ij} \left(1 - g_i \right) \\
\text{subject to } g \in \{0,1\}^n
\]  

.................. (1)
2. Max-k-Cluster problem which has the form

\[
\text{(Max- k-Cluster)} \quad \text{maximize} \quad \frac{1}{2} \sum_{i,j} w_{ij} g_i g_j \\
\text{subject to} \quad \sum_{i=1}^{n} g_i = k \\
g \in \{0,1\}^n
\] ……………… (2)

3. Max-Independent-Set(MIS) problem which has the form

\[
\text{(Max- k-Cluster)} \quad \text{maximize} \quad \sum_{i} w_{ij} g_i \\
\text{subject to} \quad g_i g_j = 0 \\
g \in \{0,1\}^n
\]

For classifying the water spring data, cluster analysis can be used.

7. Hard Problems

Some problems are hard to solve because no polynomial time algorithm is known. Also, most combinatorial optimization problems are hard Popular NP-hard problems. Now we introduce Popular NP-hard problems [6,8]:

- Traveling Salesman
- N-Queens
- Bin packing
- 0/1 knapsack
- Graph partitioning
- Hard problems in computer science
- And others.

We will choose NP-hardness problems in computer science as an example and discuss how to optimize it.

7.1 Hard problems in computer science

Many problems in computer science are “polynomial time” problems. We know of an algorithm that solves the problem exactly in \(O(n^k)\) time, for some constant \(k\). Many other problems have no known polynomial time algorithm. For example, problems whose fastest known algorithm takes \(O(2^n)\) time [exponential time]. Other problems cannot be solved at all in general such as given a program written in Matlab, does that program ever terminate. Also, A version of the halting problem [10].

8. Some Applications of NP-hardness problem

8.1 Exponential time algorithms

Many other problems have no known polynomial time algorithm. In addition, algorithms whose best solution takes \(O(2^n)\) time. If it takes 1 second to solve such a problem with \(n = 100\). Then it takes 2 seconds to solve for \(n = 101\). And it takes 250 seconds to solve for \(n = 150\). About 3.6 million years [5,6]
8.2 Travelling Salesman Problem (TSP)

We can describe a classic hard problem in CS as follows:
Problem statement: firstly, given a weighted, complete graph with n nodes. secondly, compute a tour that starts and ends at the same nodes, and visits all other nodes. Moreover, Find such a tour that has the lowest total cost. Finally, a tour is also just a permutation of the nodes.

One of a handful of fundamental problems that theoretical computer scientists turn to again and again to test the limits of successful computation is the traveling salesperson dilemma. Williamson said that the new finding "is the first step towards demonstrating that the frontiers of efficient computation are actually better than what we thought." Although there is possibly no efficient method that always seeks the shortest trip, something just as good can be found: the shortest tree connecting all the towns, implying a network of connections without closed loops (or "edges"). This tree is used by Christofides' algorithm as the backbone for a round-trip tour, adding additional edges to turn it into a round trip.

Because every arrival is followed by a departure, every round-trip route must have an even number of edges into each city. If any city in a network has an even number of ties, then the edges of the network must trace a round trip. It turns out that the opposite is also true.
This standardized property lacks the shortest tree connecting all the cities, as every city at the end of a branch has only one connection to another city. So Christofides found the best way to connect pairs of cities with odd numbers of edges in order to turn the shortest tree into a round trip. He then proved that the resulting round trip would never be more than 50% longer than the best round trip possible.
He invented perhaps the most popular approximation algorithm in theoretical computer science in doing so, one that typically forms the first example in textbooks and courses.

"The simple algorithm is known to everyone," said Alantha Newman of the University of the Grenoble Alpes and the National Center for Scientific Research in France.

Computer scientists have long speculated that an approximation algorithm that outperforms the algorithm of Christofides could exist. After all, his quick and intuitive algorithm is not always such an efficient way to plan a route for a traveling salesperson, because you might not be able to select the shortest tree connecting the cities. For eg, if this tree has several branches, each city would need to be matched with another city at the end of a branch, possibly causing lots of costs.

So Oveis Gharan, Saberi and Singh defined a random process to build their algorithm that selects a tree connecting all the cities, so that the likelihood that a given edge is in the tree is equal to the fraction of that edge in the best fractional path. There are many such random processes, so the scientists selected one that tends to produce trees with many cities that are evenly connected. After this random process spits out a particular tree, its algorithm plugs it into the scheme of Christofides to fit cities [8,11].
9. Optimization problems
An optimization problem includes optimizing or minimizing some function relative to some set, reflecting in a certain situation a number of options available. The feature makes it possible to compare the various options to decide which one could be better.
More formally, we describe the problem of optimization as follow:

\[
\begin{align*}
\text{minimize } f(t) & \quad \text{objective function} \\
\text{subject to } g(t) = 0 & \quad \text{Equality Constraints} \\
\text{h}(t) \geq 0 & \quad \text{Inequality Constraints}
\end{align*}
\]

10. Basic Concept
In this section, we will briefly discuss some basic concepts and definitions [1, 5]

Convex Sets: Let \( M \subseteq \mathbb{R}^n \). If the line segment between any two points in \( M \) lies in \( M \), i.e. , then \( M \) is said to be convex.

Convex Functions: Let \( M \subseteq \mathbb{R}^n \) be a nonempty convex set. If \( f: M \rightarrow \mathbb{R} \) satisfies

Then \( f \) is said to be a convex function on \( M \).

Conic Optimization: A set \( V \) subset of \( \mathbb{R}^n \) is a cone when with every \( m \) belong \( V \); the whole ray \( \{\alpha m|\alpha > 0\} \) is also belongs to the set \( V \). In other form \( \alpha m \in V \) for all \( m \in C, \alpha > 0 \).

Inner product: A function \((\cdot,\cdot): \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}\) is an Inner product if following is hold

1. \( (t,t) > 0, (t, t) = 0 \iff t = 0 \) (Positivity).
2. \( (t,r) = (r, t) \) (symmetric)
3. \( (t+r,z)=(t,z)+(r,z) \) (additivity).
4. \( (\alpha t, r)=\alpha(t,r) \) (homogeneity).
11. Optimization algorithm

Even with lots of computing resources, the standard optimization algorithm takes a few hours to a few days to complete. And if it takes a day or more to find optimal solutions, it is simply not feasible for a business to run multiple scenarios in order to find the scenario that works best, especially in the case of changing circumstances (e.g. road closures).

In optimization, we solve complex problems with optimization in minutes or even seconds. Our algorithm splits the problem into several sub-issues with more individualized algorithms that can be solved. For example, by quickly removing possibilities that are not possible, we have a sub-algorithm that greatly reduces calculation time.

One of the fascinating developments arising from our efficient use of cloud computing for our algorithms is called distributed optimization. We can solve several problems in parallel (i.e. simultaneously) using on-demand cloud resources (rather than on-premise installations) and using carefully built algorithms that allow parallelization, so that the overall optimization problem can be completed. To sum up, users will easily see how much the strategy would cost them with each optimization, and they can change their test scenarios accordingly [5, 11].

12. Algorithm

A new algorithm has broken a record of 44 years to find the best approximate solutions to the problem of the traveling salesperson who seeks the shortest round for any collection of citations [13, 15].

![Figure 4. The older algorithm of Travelling Salesman Problem](image)

![Figure 5. The old methods](image)
13. Applications Of TSP [4,5,7]

There are some applications of TSP we will introduce in more details as below:

1. Practicing tennis. Start with a basket of 200 or so tennis balls. When there are depleted balls, we have 200 balls lying on and around the court. The balls are to be picked up by a robot (more realistically, the tennis player). The robot starts from its station visits each ball exactly once (i.e., picks up each ball) and returns to its station.

2. Manufacturing. A robot arm is used to drill holes in a sheet of metal. Vertex TSP of N+1.

3. 8x8 Chessboard N-Queens Problem Any piece placed in the same column, row, or diagonal will attack a queen placed on a n x n chessboard
4. Subway challenge

This challenge has three key variations:
A ride that needs a passenger, but not necessarily the entire line, to cross any line. (A-Class)
Full-system ride that needs every station to stop for a passenger. (B Class)
Skip-stop ride, which allows only one rider to travel through each station. (From Class C).
The Moscow subway, for example, optimal complete rout [12]
14. Conclusions
NP-hardness classifications were reviewed, the most important types present were addressed, and the general form of each type. The applications of the NP-hardness in our daily life were studied and the new algorithm was introduced to achieve the goal with the best results. Finally, even with lots of computing technique, the standard optimization algorithm takes a few hours to a few days to complete and goal is to get the best solution in faster time.

References:

[1] Al-Jilawi, A. (2019). Solving the Semidefinite Programming Relaxation of Max-cut Using an Augmented Lagrangian Method. Northern Illinois University.

[2] Hochba, D. S. (Ed.). (1997). Approximation algorithms for NP-hard problems. ACM Sigact News, 28(2), 40-52.

[3] Gould, N. (2006). An introduction to algorithms for continuous optimization.

[4] Du, Y., & Ruszczyński, A. (2017). Rate of convergence of the bundle method. Journal of Optimization Theory and Applications, 173(3), 908-922.

[5] Mäkelä, M. (2002). Survey of bundle methods for nonsmooth optimization. Optimization methods and software, 17(1), 1-29.

[6] Hestenes, M. R. (1969). Multiplier and gradient methods. Journal of optimization theory and applications, 4(5), 303-320.

[7] Aloise, D., Deshpande, A., Hansen, P., & Popat, P. (2009). NP-hardness of Euclidean sum-of-squares clustering. Machine learning, 75(2), 245-248.

[8] Chen, J. E. (2005). Parameterized computation and complexity: a new approach dealing with NP-hardness. Journal of Computer Science and Technology, 20(1), 18-37.

[9] Du, Y., & Ruszczyński, A. (2017). Rate of convergence of the bundle method. Journal of Optimization Theory and Applications, 173(3), 908-922.

[10] Garey, M. R., & Johnson, D. S. (2002). Computers and Intractability, vol. 29.

[11] Vassilevska Williams, V. (2015). Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In 10th International Symposium on Parameterized and Exact Computation (IPEC 2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[12] Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1980). Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. SIAM Journal on Computing, 9(3), 558-565.

[13] Garey, M. R., & Johnson, D. S. (1990). A Guide to the Theory of NP-Completeness. Computers and Intractability, 37-79.

[14] de Oliveira, W., & Tcheou, M. P. (2019). An inertial algorithm for DC programming. Set-Valued and Variational Analysis, 27(4), 895-919.

[15] Mann, Z. Á. (2017). The top eight misconceptions about NP-hardness. Computer, 50(5), 72-79.