Heuristic Algorithms for $\mathcal{NP}$-Hard Problems

Radu Ştefan Mincu

-summary-

The challenges of solving $\mathcal{NP}$-hard problems are everywhere. From thinking about acquiring groceries under a limited capacity (a classic knapsack problem), to planning for a multi-country holiday tour (featuring the famous travelling salesman problem), solving such problems is and always has been ubiquitous.

Why would these issues need more attention now than in the past? In the last century, due to the advances of computational power, a good amount of research has dealt with finding out exactly how much can be done with computers. Needless to say, many tasks have become or are close to becoming fully automated by computer algorithms (e.g. automatic translation). However, some problems remain stubborn and cannot be easily handled by our current computational models (even with the advent of quantum computing), the knapsack and traveling salesman problem being well-known examples.

In this thesis we concern ourselves with approaching difficult computational problems by several methods (such as heuristic algorithms) and subsequently providing evidence that our methods are sound. However, first of all, what is considered a difficult computational problem? And why would we go out of our way trying to solve it?

We find ourselves in a time and age where we are faced with handling computationally difficult problems in practically all areas having to do with scheduling, planning, cost analysis, transportation, many of which are a part of everyday life. For example, the problem of finding the cheapest airliner ticket between airport A and airport B in today's airliner systems is computationally difficult (even in very restricted forms, see [4, 8]), yet it is a problem that must be answered millions of times each and every day.

The previous example emphasizes the importance of solving difficult problems: surely we need to be able to find the correct flight such that we may arrive at our destination by boarding the aircraft of companies worth billions of U.S. dollars. To illustrate the prevalence of difficult problems, look no further than being tasked with grocery shopping under the constraint of physical space limitation (such as having to pack your car with

the most important groceries in your shopping list - a simple example of a knapsack problem). We can perceive these problems as not simple, though we have yet to define the concept of problem difficulty.

How do we determine whether a problem is difficult or not? If one is unable produce a reasonable algorithm for solving the problem, does it mean that a problem is difficult? Are there any problems for which we are simply unable to provide such a reasonable algorithm? These are all questions that need to be answered in order to grasp the concept of difficult computational problem. Furthermore, what do we mean by *reasonable* algorithm?

Fortunately, there exists well established groundwork that gives a suitable answer to these questions in the fields of the theories of computation, computational complexity and algorithm analysis. These long studied fields have developed instruments to characterize the efficiency of algorithms that we can apply to a given problem. More precisely, we may be concerned with the usage of two main resources, namely time (i.e. duration of the process of computation required to provide a solution) and space (i.e. memory capacity needed to keep record of all items required for the process of computation to successfully complete). The consumption of these two resources of time and memory leads to the definition of two abstractions regarding algorithms: the *time complexity* and *space complexity* of the various algorithms employed to solve different problems. These two notions are defined in terms of asymptotic necessities of computation time and machine memory required to compute the solution to a given problem as the size of the input increases.

It turns out that many common problems can be solved efficiently in terms of the required resources. If the resources of space and time can be bounded by some *polynomial* functions, then they are easy to solve by computers i.e. they are *tractable*. Examples include:

- sorting a collection of items (e.g. lexicographic sorting of book titles in a library)
- finding an item efficiently (e.g. searching for a name in the telephone book by locating the section with the first letter, then narrowing down using the second letter and so on - is related to both hashing and binary search algorithms)
- locating the shortest combination of roads between two points on a map (maybe less straightforward for humans, but computers can handle this task easily)

Other problems are not easy to solve, but are easy to verify. Consider the subset sum problem, where, given a set of positive and negative integers, one must determine a subset of integers which sums to zero. Given a solution, we can quickly tell if it is correct by adding everything up. However, finding such a subset for an arbitrarily large problem instance is much more resource demanding than the previously mentioned problems.

In fact, we do not know *if algorithms exist which easily solve the problems*

that can be verified quickly. Moreover, researchers have been unable to prove neither existence, nor non-existence of such an algorithm. This question known as *the $\mathcal{P}$ vs. $\mathcal{NP}$ problem* has left scientists baffled for a long time, so much that $\mathcal{P}$ *vs.* $\mathcal{NP}$ is part of the famous *Millenium Prize Problems* [3].

The previously mentioned subset sum problem is part of the $\mathcal{NP}$-hard category of problems. These problems are not known to admit algorithms that use polynomial amounts of space and time. Researchers believe that these problems need at least an *exponential* amount of time to solve and this requirement makes the problems *intractable*, i.e. not easily handled by computers.

At first glance, some problems can only be solved by trying to enumerate all possible solutions and see if they are correct. For the subset sum problem, these possible solutions number 2^n, where n is the number of items in the original set. Since 2^n is an *exponential* function, we need at most an *exponential* amount of time to check every possible combination and solve the problem. This method is known commonly as *brute-force search* or *exhaustive search* in the *solution space* (of all possible solutions). To get better results what is needed is some insight into the structure of the problem, allowing the use of other techniques that exploit said structure.

Three common approaches to $\mathcal{NP}$-hard problems are briefly introduced below:

1. Exact methods. What if we need an optimum solution for an $\mathcal{NP}$-hard problem? We know that we can solve the problem by iterating through its exponentially-sized solution space. We can solve small instances to optimality using this method, but how far can we push the size of the input for the problem to be solvable in reasonable time? Some ideas can be found in the paradigm of Integer Linear Programming.
2. Approximation algorithms. What if we don't require the optimum solution, but instead a solution reasonably close to the optimum? If we give up on the need for optimality, can we find polynomial time algorithms that produce a good solution? This is the field of study of approximation algorithms, the purpose of which is to obtain solutions proven to be within a factor of the optimum *on all instances*, yet only using polynomial time.
3. Heuristic algorithms. Sometimes we need to solve larger instances that exact methods cannot cope with. In some of these cases, approximation algorithms are either unknown, have poor approximation factors or simply do not work well in practice. Heuristic algorithms come in to fill the gap and provide practical solutions. Since they are often not approximation algorithms in the sense that we have no proof of their approximation factor, these algorithms are usually backed up by experimental evidence that they are suitable for the problem at hand.

In this thesis we focus on the heuristic methods for $\mathcal{NP}$-hard problems. Despite this statement, we are often faced with having to show that our

methods work well in practice. Consequently, some attention is needed towards finding optimum solutions on smaller or otherwise restricted instances in order to provide an idea about the performance of our heuristics. In the case of approximation algorithms, the quality of the algorithm is given by the approximation ratio guarantee. However, in the case of heuristic approaches, there is no such proof. As such, we resort to experimental evidence by providing some relevant instances for which the optimum is known (found by using exact methods) and showing how close we can get to the optimum by using the designed heuristics.

Structure of the Thesis

The present thesis is structured as follows:

Chapter 2) introduces some **preliminaries about local search and metaheuristic frameworks**, as well as setting these topics in the context of combinatorial optimization. We also describe the connection to approximation algorithms and exact methods. We present three well-known metaheuristic frameworks, namely simulated annealing, tabu search and genetic algorithms, that have seen usage in our approaches presented in the subsequent chapters.

Chapter 3) presents **the min-max 2-coloring problem**, an edge coloring problem on graphs motivated by applications in wireless mesh network optimization. The metaheuristics used here are based on simulated annealing and tabu search and are thoroughly described and tested. The work in this chapter has been published in the proceedings of the 24th International Computing and Combinatorics Conference (COCOON2018) [6].

Chapter 4) describes **the repetition-free longest common subsequence problem** and showcases a new heuristic based on dynamic programming that can stay competitive with the state-of-the-art metaheuristic approaches based on ant colony optimization, while being much easier to implement and maintaining low running time. The work in this chapter has been published in the proceedings of the 25th International Symposium on String Processing and Information Retrieval (SPIRE2018) [5].

Chapter 5) reveals **the string factorization problem**, where the goal is to split a string into as many distinct substring as possible. Despite the simple definition, the problem is difficult to handle. We describe a greedy-based heuristic that adds the immediately following distinct substring at each step (starting "at the left"). While attempting to prove that this algorithm is in fact a $\frac{1}{2}$-approximation, we draw a parallel to a less restricted problem.

We are baffled by the discovery that the less restricted problem seems to have the same optimum as the original in all of our experiments, although there is no easy way to prove this conjecture. The work in this chapter is submitted for publication in conference proceedings and is currently under review.

Chapter 6) showcases **the longest filled common subsequence problem**, another string problem where the goal is to find the longest common subsequence between a first unmodified string and a second string in which we are allowed to insert symbols from a given finite set. We show that we can drastically reduce the search space by shifting our point of view from inserting into the second input string towards deleting from the first input string. We present a simple local search based heuristic algorithm taking advantage from this approach. The work in this chapter has been published in the proceedings of the 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC2018) [7]. The work was also featured in a PhD session (and displayed in poster format as well) at the same conference, where it won the *Best PhD Paper* award.

Chapter 7) brings up **generalized function matching**, a combinatorial pattern matching problem variant where pattern symbols may match entire text substrings. We suggest a heuristic approach based on suffix trees that yields good results in practice. The work in this chapter has been published in the proceedings of the 23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES2019).

Chapter 8) has to deal with a very important problem, that of **kidney exchange programmes (KEPs) in an international setting**. Patients may exchange their willing but incompatible donors among themselves using the KEP framework. This has become common practice across Europe and in the United States, where KEPs function on a per-country or per-transplant-centre basis, respectively. We explore the benefits of KEP collaboration using computational simulations. In this work, we use only exact methods for tackling the problem because of ethical reasons: almost all stakeholders (transplant centres, clinicians, surgeons and most importantly, the patients and the donors) would almost certainly reject non-optimal solutions. Moreover, although the problem is intractable, it is still possible to solve optimally in a matter of seconds for up to 300 patients using the state of the art solvers and known ILP models. The work in this chapter has been published as a short paper in the proceedings of the 8th VOCAL Optimization Conference: Advanced Algorithms (VOCAL2018) [1]. An extended journal version is currently under

review at the Central European Journal of Operations Research (CJOR) and is available on arXiv [2].

Chapter 9) is the concluding chapter where we summarize our findings and present future work directions.

The author of this thesis, Radu Mincu, mentions here that the majority of the work is published rightfully featuring his PhD supervisor, Dr. Alexandru Popa as a co-author. The exception is Chapter 8, which is the result of a more extended collaboration with colleagues from Hungary (Dr. Péter Biró and Márton Gyetvai), as well as India (Utkarsh Verma). The work in Chapter 7 presents a publication authored only by Radu Mincu, to satisfy a recently introduced PhD programme criterion.

References

[1] Péter Biró, Márton Gyetvai, Radu Stefan Mincu, Alexandru Popa, and Utkarsh Verma. Ip solutions for international kidney exchange programmes. In *VOCAL 2018. 8th VOCAL Optimization Conference: Advanced Algorithms*, pages 17–22, 2018.

[2] Péter Biró, Márton Gyetvai, Radu Stefan Mincu, Alexandru Popa, and Utkarsh Verma. Ip solutions for international kidney exchange programmes. *arXiv preprint arXiv:1904.07448*, 2019.

[3] Arthur M Jaffe. The millennium grand challenge in mathematics. *Notices of the AMS*, 53(6), 2006.

[4] Carl Marcken and Edwin Karat. Generating flight schedules using fare routings and rules, May 19 2005. US Patent App. 10/714,525.

[5] Radu Mincu and Alexandru Popa. *Better Heuristic Algorithms for the Repetition Free LCS and Other Variants: 25th International Symposium, SPIRE 2018, Lima, Peru, October 9-11, 2018, Proceedings*, pages 297–310. 01 2018.

[6] Radu Mincu and Alexandru Popa. *Heuristic Algorithms for the Min-Max Edge 2-Coloring Problem*, pages 662–674. 06 2018.

[7] Radu Stefan Mincu and Alexandru Popa. Heuristic algorithms for the longest filled common subsequence problem. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 449–453. IEEE, 2018.

[8] Sara Robinson. Computer scientists find unexpected depths in airfare search problem. *SIAM News*, 35(6), 2002.