ON THE CLASSIFICATION OF NP COMPLETE PROBLEMS AND THEIR DUALITY FEATURE

WenhongTian[1,2], WenxiaGuo[1] and Majun He[1]

[1]University of Electronic Science and Technology of China, Chengdu, China
[2]Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Science

ABSTRACT

NP Complete (abbreviated as NPC) problems, standing at the crux of deciding whether P=NP, are among hardest problems in computer science and other related areas. Through decades, NPC problems are treated as one class. Observing that NPC problems have different natures, it is unlikely that they will have the same complexity. Our intensive study shows that NPC problems are not all equivalent in computational complexity, and they can be further classified. We then show that the classification of NPC problems may depend on their natures, reduction methods, exact algorithms, and the boundary between P and NP. And a new perspective is provided: both P problems and NPC problems have the duality feature in terms of computational complexity of asymptotic efficiency of algorithms. We also discuss about the NPC problems in real-life and shine some lights on finding better solutions to NPC problems.

KEYWORDS

P Problems, NP problems, NP Complete Problems,Reduction, The Duality Feature

1. INTRODUCTION

In 1971, Cook [1] firstly established a theorem that a class of problems can be P-reducible (polynomial time reducible) to each other and each of them can be P-reducible to Boolean Satisfiability (SAT) problem, this class of problems is called NP (nondeterministic polynomial time) problems. Karp [2] applied Cook's 1971 theorem that the SAT problem is NP Complete (also called the Cook-Levin theorem) to show that there is a polynomial time reduction from the SAT problem to each of 21 combinatorial problems, thereby showing that they are all NP complete (NPC). This was one of the first demonstrations that many natural computational problems occurring throughout computer science are computationally intractable, and it drove interest in the study of NP-completeness and the P versus NP problem [3]. The P versus NP problem, determining whether or not it is possible to solve NP problems quickly, is one of the principal unsolved problems in computer science today and listed as one of seven millennium problems [4], challenging tens of thousands of researchers.

Simply speaking, P problems mean that the class of problems can be solved exactly in polynomial time while NPC problems stands for a class of problems which can be solved in nondeterministic polynomial time by Turing machine. NPC problems has far-reaching consequences to other problems in mathematics, biology, philosophy and cryptography. More specifically, in Big O-notation of computational complexity for asymptotic efficiency of algorithms, P problems can be solved in polynomial time of $O(n^k)$ for some constant k where n is the size of input to the problem, while NPC problems may have computational complexity of $O(2^{cn})$ including both exponential time and sub-exponential time, where c is constant larger than zero.

Karp [2] ever claimed that if any of NPC problems have efficient polynomial time algorithms, then they all do. It is for this reason that research into the P versus NP problem centers on NPC problems, i.e., looking for efficient polynomial time algorithms for NPC problems. Through decades' efforts by many researchers, it is still an open question. There are quite many results but none of them is commonly accepted yet by the research community.

Intuitively, NP problems is the set of all decision problems for which the instances where the answer is "yes" have efficiently verifiable proofs of the fact that the answer is indeed "yes". More precisely, these proofs have to be verifiable in polynomial time by a deterministic Turing machine. In an equivalent formal definition, NP problems is the set of decision problems where the "yes"-instances can be accepted in polynomial time by a non-deterministic Turing machine [4]. Simply speaking, P-class problems mean that the class of problems can be solved exactly in polynomial time while NP (non-deterministic polynomial) problem stands for a class of problems for which each can be solved in polynomial time by a nondeterministic Turing machine. NP problems has far-reaching consequences to other problems in mathematics, biology, philosophy and cryptography.

The complexity class P is contained in NP, and NP contains many important problems. The hardest of which are NP Complete (NPC) problems. A decision problem *d* is NPC if : *d* is in NP, and every NP problem is reducible to *d* in polynomial time [2]. The most important open question in complexity theory, is the P versus NP problem which asks whether polynomial time algorithms actually exist for NPC problems and all NP problems.

It is widely believed that NP!=P in 2002 [5]. In 2012, 10 years later, the same poll was repeated [6]; the number of researchers who answered was 126 (83%) believed the answer to be no, 12 (9%) believed the answer is yes, 5 (3%) believed the question may be independent of the currently accepted axioms and therefore is impossible to prove or disprove, 8 (5%) said either don't know or don't care or don't want the answer to be yes nor the problem to be resolved. On the other hand, researchers in the world never stop to tackle this problem. Fortnow [7] reviewed the status of the P versus NP problem on its importance, attempts to prove P≠NP and the approaches dealing with NPC problems. At the Web site [8] ,Woeginger provides an unofficial archivist of about 116 claims for the P versus NP problem from 1986 to April 2016, among them, 49 (42%) believed the answer to be no, 62 (53%) believed the answer is yes, the other 5 (5%) think Undecidable, or Unprovable or Unknown, though none of them is commonly accepted yet by the research community.

Through decades, NPC problems are treated as one class. Observing that most of NPC problems have different natures, in this study, we show that NPC problems are actually not the same in computational complexity, and they can be further classified. We propose classifications in terms of their natures, reduction methods, exact algorithms, and the boundary between the P versus NP problem, and a new perspective: both P problems and NPC problems have the duality feature in terms of computational complexity of asymptotic efficiency of algorithms. We also discuss NPC problems in the real-life and shine light on finding better algorithms for them.

2. THE NPC PROBLEMS CAN BE CLASSIFIED BASED ON THEIR NATURES

NPC problems have different natures; they can be classified into six basic genres [2, 9], i.e., Satisfaction, Packing, Covering, Partitioning, Sequencing, Numerical computing.

Satisfaction problems are the first kind of NPC problems. Examples include Circuit Satisfiability, SAT, 3SAT. SAT problem is the first and fundamental combinatorial problem identified as NPC

[1]. The problem can be stated as: given a set of clauses c_1, c_2,…,c_k over a set of variables $X=\{x_1, x_2, \ldots, x_n\}$, does there exist a satisfying truth assignments? A special case of SAT is that all clauses contain exactly three terms (corresponding to distinct variables). This problem is called 3SAT. Intuitively, the computational complexity of SAT is O(2^k); as k close to n, it becomes to O(2^n); 3SAT has similar computational complexity.

Packing problems have the following structure: a collection of objects is given, and one wants to choose at least k of them; a set of conflicts exist among the objects, which preventing from choosing certain groups simultaneously. Examples include maximum Independent Set (MIS) problem (with computational complexity of O($k^2 C(n, k)$) by brute-force method where $C(n, k)$ is the combination function of choosing k elements from n elements), Set packing problem (with the same computational complexity as MIS) etc..

Covering problems have a nature contrast to packing problems: a collection of objects is given, and one wants to choose a subset that collectively achieves a certain goal (the goal may not clear known before computing), with only k of the objects can be chose. Examples include Vertex cover problem (VCP) with computational complexity of O($k^2 C(n, k)$), Set cover problem etc.. Covering problems regularly complementary to Packing problems, therefore they have the same or similar complexities.

Partitioning problems involve a search over all ways to divide up a collection of objects into subsets so that each object appears in exactly one of the subsets. Examples include 3-Dimensional matching, Graph coloring etc., both with computational complexity of $O(2^n)$ by brute-force method.

Sequencing problems involve search over the set of all permutations of a collection of objects in some order. Examples include Traveling Salesman problems (TSP) with computational complexity of O($n^2 2^n$) by dynamic programming (9),Hamiltonian Cycle problem (HCP).

Numerical computing problems involve choosing a subset from a collection, collectively achieves a given goal (the goal (sum) is clearly stated). Examples include Subset Sum problem with computational complexity of O($\sum_{k=1}^{n}(C(n,k))$)$=O(2^n)$ by brute-force method while O(nT) can be achieved by dynamic programming where T is the target value. In this complexity of O(nT) may not be considered as NPC problem. We will discuss this further later.

We can observe that the six genres have different natures and computational complexities. If brute-force (trivial exhaustive) method is applied, Numerical Computing ≤ Packing ≤ Covering ≤ SAT ≤ Partitioning ≤ Sequencing problems in complexity. However, Numerical computing problems can have much less computational complexity by dynamic programming, very different nature than others; Sequencing problems are harder than others. We will further discuss this in the following.

3. NPC Problems Can be Classified by Reduction Methods

NPC problems require that every problem in NP is reducible to a NPC (or SAT) problem in polynomial time [2]. We observe that reduction methods of NP Complete problems are different for different problems. They can be classified as:

Special formulation of NPC problems.This kind of NPC problems are easy to reduction. For examples, SAT problem and 3SAT. Since 3SAT is just a special case of SAT, so it is very easy to reduction between them. Similarly, 3-coloring problem and k-coloring problem.The reduction between them takes negligible time. Because of its expressive flexibility, 3SAT is often a useful starting point for reduction. Similarly graph coloring problem and 3-coloring problem are in this line.

Complementary NPC problems. For instance, Maximum Independent Set (MIS) which forms the largest independent set of nodes without connection by edgesamong them and Minimum Vertex Cover (MVC) which forms the smallest vertex set of nodes covering all the edges in a graph. Let a graph $G=(V, E)$ where V is its node set and E is its edge set. Then$I$ is an independent set if and only if its complement ($V-I$) is a vertex cover, this is proved in [9].Similarly CLIQUE problem and Vertex cover problem are complementary. The reduction between complementary NPC problems takes negligible time.

Hard to reduction NPC problems. For instance, HCP (Hamiltonian Cycle problem) and 3SAT. The connection between these two NPC problems is not obvious. Actually it is quite complicated to do reduction between them. It may take quite some time to design and implement the reduction in this case, as shown in [8].

Also in some other cases, the reduction can change the size of the original NPC problem.For instance, 3SAT≤p CLIQUE (≤p stands for P-reducible) is easy as done in [10];However, the reduction doubles the number of nodes for CLIQUE since the negation of each variable is included as a node. Therefore, the computational complexity is increased from 2^n to 2^{2n} if implemented by trivial solution. Similarly 3SAT≤p SSP [8], increasing the number of variables from n to $2(n+k)$ where k is the number of clauses in 3SAT. We cannot safely say they are still at the same level of computational complexity in this case.

Summarizing these scenarios, what we can say is that NPC problems are not equivalent in term of reduction among them. The reduction methods may take different time, **and the reduction may change the size of the original NPC problem, which making their computational complexityindeed different**.

4. THE CLASSIFICATION CAN BE BASED ON EXACT SOLUTIONS TO SOME NPC PROBLEMS

Though it is very hard, researchers never stop to find more efficient exact solutions than trivial (brute-force) solutions to NPC problems. In **[11]**, Woeginger surveyed that some fast, super-polynomial time algorithms which solve NPC problems to optimum, and found that some NPC problems have better and faster exact algorithms than others; there is a wide variation in the worst case complexities of known exact algorithms, as shown in Table 1. Notice that these problems are representatives of the six basic genres in NPC problems. It can be seen that SSP≤MIS=SCP<3CP<3SAT<TSP in term of complexity, the list can be added. These evidences indicate that the computational complexity of exact solutions to some NPC problems can be further classified; we may classify NPC problems in more detailed categories such as in term of c^n for each c increasing by 0.1. There may exist a sequence of NPC problems X_1, X_2, X_3, … , in NP, each strictly harder than the previous one if we sort them, as also suggested in **[9]**.

Tarjan and Trojanowski**[15]** also found that even for NPC problems it is sometimes possible to develop algorithms which are substantially better in the worst case than the obvious enumeration algorithms (trivial brute-force solutions).

For MIS alone, trivial solution has complexity of O(2^n), and exact solutions with complexities of O(1.4422^n) in 1965 [16], 1.2599^n in 1977 [15], 1.2346^n in 1986 [17], 1.2227^n in 1999 [11], 1.2125^n in 2012 [18], 1.1996^n in 2013 [12] are respectively found, researchers are still finding better exact solutions to it. Although no polynomial time algorithm has yet been discovered for NPC problems, one can see that the computational complexities of some exact solutions become close to polynomial or called super-polynomial [11] or pseudo-polynomial [9].

Table 1. The complexities of exact solutions to some NPC problems from [11] and some recent papers *

Problem	Complexity	Paper	Genre
MIS (maximum independent set)	1.1996^n	[12]	Packing
Set covering problem (SCP) *	1.1996^n		Covering
3CP (3-coloring of planar graph)	1.3446^n	[13]	Partitioning
SSP (subset sum problem)*	nT	[9]	Numerical
3SAT	1.4802^n	[14]	Satisfaction
TSP	$n^2 2^n$	[9,11]	Sequencing

**: In Table 1, cited paper and their genres are also given. Notice that T is the target value for SSP and dynamic programming is applied for SSP; and SCP has similar complexity as MIS [9].*

5. NPC PROBLEMS AND THE P VERSUS NP PROBLEMS ARE CLOSELY RELATED TO THE SIZE OF THE PROBLEM

One reason that researchers still cannot find efficient solutions to NPC problems may be that the size (scale) is always increasing. For instances, the node number of TSP problem varies from a few tens to tens of thousands or more during recent 60 years of research. This is the curse of the dimension.

With rapid increased speed of modern computers, large instances of NPC problems can be solved efficiently. Table2 shows records of optimum solutions to some TSP problems.

One can see that the size of the problem is increasing as year goes. Notice that the instance with 1904711 nodes is still not yet solved exactly but just has a good lower bound **[19]**. Similar results are also observed for HCP and other NPC problems. For real-life problems with small or moderate number of variables, they may be solved to exactly very easily. **Since NPC problems can be P-reducible to each other, once we find a good efficient algorithm for one NPC, it should also work for other NPC problems**. However, if the problem size increases to very large, finding efficient solutions to NPC problems become intractable.

Table 2.Records of optimum solutions to TSP problems [19] where$n$ is the number of nodes in TSP. All TSP problems in the table are solved to optimum except for the last one

n	Year (solved)	Node type
48	1954	USA cities
64	1971	random nodes
80	1975	random nodes
120	1977	Germany cities
318	1987	cities
532	1987	USA cities
666	1987	World cities
1002	1987	cities
2392	1987	cities
3038	1992	cities
13509	1998	USA cities
15112	2001	cities
24978	2004	Sweden cities
85900	2006	cities
100000	2009	Japan
1904711	2010*	World TSP Challenge

6. THE DUALITY FEATURES OF P AND NP PROBLEMS

The following definitions are based on the computational complexity of different problems in the worst case.

Definition 1: The asymptotic efficiency of algorithms [9,10]: concerns with how the running time of an algorithm increases with the size of the input *in the limit*, as the size of the input increases without bound.

Definition 2: The O-notation of computational complexity of an algorithm: asymptotically bounds a function from above within a constant factor [9,10]. For a given function $g(n)$, we denote by O($g(n)$) :

O($g(n)$)={$f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for $n \geq n_0$}.

Fig.1 shows the intuition behind O-notation. For all values n to the right of n_0, the value of function $f(n)$ is on or below $g(n)$.

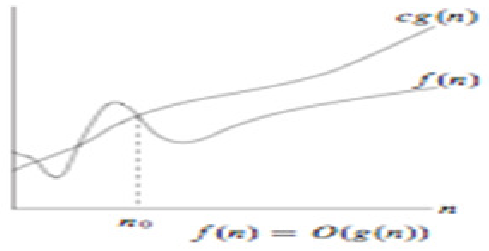

Fig.1 f(n)=O(g(n)) [10]

Since O-notation describes an upper bound, when we use it to bound the worst-case running time of an algorithm, we have a bound on the running time of the algorithm on every input. For example, the doubly nested loop structure of the insertion sort algorithm has an O(n^2) upper bound; equivalently, we mean that the worst-case running time is O(n^2).

Definition 3: P-class problems in term of O-notation of computational complexity: they have computational complexity of O(n^k) for some constant k where n is the size of input to the problem.

Definition 4: NPC problems: their exact solutions may have computational complexity of O(2^{cn}) **in O-notation**, where n is the size of input to the problem and c is a positive constant.

TABLE I shows computational complexities of exact solutions to some NPC problems (abstracted from [11]).We also consider that the representation of the input to the problem by m bits (in binary) and the space complexity of a problem by s bits (in binary in memory).

Fact 1: A regular system (the computer hardware, called the system in the following) has capability of handling B_0 numbers in term of computational and space complexity, i.e., handling the input efficiently within reasonable time (may be within a few hours or minutes depending on the applications) without overflowing the system, where B_0 may be related to the memory size or whichever (CPU, memory, disk et.) is the bottleneck of the system.

Based on the definitions above, the capabilityB_0can be represented by

$$B_0=\boldsymbol{min}(2^{cn}, n^k, 2^m, 2^s) \qquad (1)$$

Table 3. The summary of variables

Variables	Meaning
k, c, n_0	apositive constant
n,e	The size of inputs to a problem
s, m	The number of bits (binary)
B_0	**min**(2^{cn}, n^k, 2^m, 2^s)
W, C, b	apositive number

For example, Subset Sub problem (SSP) is pseudo-polynomial time solvable with computational complexity of O(nW) [9, 10], it is polynomial time only when both n and W are less than B_0 (or bounded by a polynomial function) while it is NPC problem when both n and W are very large (beyond B_0).

Definition 6: the transformability of P-class problems to NPC problems: in term of computational complexity when the size of the representation of the inputs become very large, P-class problems with computational complexity of $O(W)$ can become NPC problems with computational complexity of O(2^{cn}).

Lemma 1. Dynamic programming algorithm for Subset Sum problem (SSP) is pseudo-polynomial time.

Proof. The proof is provided in [9]. For completeness, the proof is restated here. The dynamic programming algorithm (DPA) for SSP has running time of O(nW), which is reasonable and looks like polynomial when W is small, but becomes hopelessly impractical as W (and the numbers of elements in the set) grow large. Consider, for example, an instance with 100 numbers, each of which is 100 bits long. Then the input is only 100×100=10,000 digits, but W is now roughly 2^{100} (2^n). Since integers will typically be given in bit representation, or base-*b* representation where *b* can be any integer. The quantity *W* is really exponential in the size of the input (2^n); the DPA for SSP was not a polynomial-time algorithm, but referred to it as pseudo-polynomial, to indicate that it ran in polynomial-time in the magnitude of the input numbers, but not polynomial in the size of their representation. ■

Based on **Lemma** 1 and above conditions, our main result is the following theorem.

Theorem 1:The instances of SSP can have computational complexity of NPC problems and P-class problems, this is called the duality feature in this paper.

Proof: The proof is straightforward based on above definitions and **Fact** 1. Consider that P problems can be solved in polynomial time of $O(n^k)$ for some constant *k* where *n* is the size of input to the problem (**Definition** 3) while NPC problems have computational complexity of $O(2^{cn})$ (**Definition** 4).

We consider the following two cases from **computational complexity point of view,** especially taking the representation of the input into account:

1) If $((n^k \le B_0)$ & $(2^{cn} > B_0))$, then P <NPC (or called P≠NPC). This is because that the system can handle efficiently the P problem (when $n^k \le B_0$) but cannot handle efficiently the NPC problem.

2) If $((n^k > B_0)$ & $(2^{cn} > B_0))$, then P->NPC. This is because that the system cannot handle either one efficiently, the P problem becomes a NPC problem or the P problem is pseudo-polynomial time by **Definition** 5 in this case.

This completes the proof. ■

One of the obvious evidences forP problems and NPC problems are mutual transformable is the Subset Sum problem (SSP): it is one of 21 NPC problems in original list of Karp's paper [2], named as Knapsack problem; however it is pseudo-polynomial time of $O(nW)$ [9], it will be polynomial time solvable if both *n* and *W* are less than B_0 (or bounded by a polynomial function) while it is NPC problem when both *n* and *W* are very large (beyond B_0 in representation).

Observation 1: Even one can find efficient solution (polynomial time algorithm) to one or more NPC problems, some pseudo-P problems may become NPC problems in computation-al complexity when the representation of the input to them become very large.

Observation 1 means that there still exist other problems, especially pseudo-polynomial time problems to become NPC problems in computational complexity even if one finds efficient polynomial time for one or more NPC problems. Then why do we still work on the P versus NP problem? Maybe just for finding better efficient solutions.

7. DISCUSSION AND CONCLUSION

As P-class problems have different computational complexities, we find NPC problems also have different computational complexities. Since NPC problems are P-reducible to each other, currently very good algorithms for SSP and MIS are already found, we may find better efficient algorithms for NPC problems in the near future.

NPC problems in many cases are man-made (artificial), they are much easier to solve in real world. For instance, SAT problem is from practical Circuit Satisfiability problem (CSP). Its computational complexity actually depends on the number of inputs since it involves n variables with k clauses. However, in practice, it is unlikely to have too large exponential number of combination inputs for any circuit but reasonable number of inputs, which should be polynomial time solvable. Thinking about any real circuit, if we have to check the truth assignment by exponential-time, the circuits may not be useful.

Cloudert [20] also found that exact coloring of real-life graphs is easy since real-life graphs always have small or moderate number of nodes and edges, which can be solved to optimum very easily using today's desktop computer. Similarly exact solution to 3-coloring of real-life planar graphs is very easy, since the world map only have about 200 countries there.

Also TSP problems in real world may only have a few tens nodes (there is hardly a case that a salesman has to travel more than 100 sites at one time), which can be solve exactly and easily by heuristic algorithm such as LKH [21]. Notice that we show the records of TSP benchmarks in Table 2. Just for challenging, the current largest TSP problem has 1904711 nodes, other benchmark problems with nodes less than 1904711 are reported to be solved exactly to optimum [19].

We also observe that approximation algorithms regularly perform much better than exact algorithms for many NPC problems. For TSP problem, LKH algorithm is heuristic (benchmark), known to solve many TSP problems to optimum within reasonable time (estimated computational complexity of $O(n^{2.2})$). Comparing against the exact solution of TSP with computational complexity of $O(n^2 2^n)$ by Dynamic Programming, LKH performs much better and faster; the exact solution can only solve TSP with a few tens (less than 30) of nodes using today's fast desktop computer while LKH can solve TSP with a tens of thousands of nodes using the same desktop computer. Similarly for Hamiltonian Cycle problem (HCP) and other problems, heuristic algorithm may perform much better than exact algorithm for NPC problems. The current author also proves that a near optimal result can be assured for TSP problems using LKH algorithm [22]. In this case, we may consider using heuristic algorithms instead of exact solutions.

As another perspective, it is recently proven mathematically that memcomputing machines (a novel non-Turing paradigm) have the same computational power of nondeterministic Turing machines [26]. Therefore, they can solve NPC problems in polynomial time with resources that only grow polynomially with the input size.

Donald Knuth, who ever proposed the name of NP Completeness by a poll, said in 2014, "I've come to believe that P = NP, namely that there does exist an integer M and an algorithm that will solve every n-bit problem belonging to the class NP in n^M elementary steps" [26]. However, we still not yet find the M.

NPC problems and P versus NP problem challenge many researchers to tackle them. In this paper, we show that NPC problems are actually not equivalent in computational complexity except for trivial or brute-force solutions. We show that the classification of NPC problems may depend on their natures, reduction methods, exact algorithms, and the boundary between P and NP. These may shine light on rethinking NPC problems and P versus NP problem, hopefully help to find better solutions for them.

NPC problems have different natures, they can be classified into six basic genres [2, 7], i.e., Satisfaction, Packing, Covering, Partitioning, Sequencing, Numerical computing.

Originally, PRIMES and Graph Isomorphism were hard to determine to be in NPC or not in Cook's paper [1]. In 2004, PRIMES is found in P-class and accepted by the research community [10]. And in 2015, Graph isomorphism is reported to have Quasipolynomial time solution [11, 12], though the results are still under verification. These show some new perspectives and trends on NPC problems.

As another perspective, it is recently proved mathematically that memcomputing machines (a novel non-Turing paradigm) have the same computational power of nondeterministic Turing machines [13]. Therefore, they can solve NPC problems in polynomial time with resources that only grow polynomially with the input size.

NPC problems and the P versus NP problem challenge many researchers to tackle them through decades of efforts. In this paper, a new perspective is also provided: P class problems and NPC problems can be transformable in terms of computational complexity. Hopefully, this may shine light on solving the P versus NP problem.

ACKNOWLEDGMENTS

This research is supported by National Science Foundation of China (NSFC) with grant ID (61672136, 61650110513)); Xi Bu ZhiGuang Plan of CAS (R51A150Z10); Science and Technology Plan of Sichuan Province (2016GZ0322).

REFERENCES

[1] S.A. Cook (1971). The Complexity of Theorem Proving Procedures, Proceedings of the third annual ACM symposium on Theory of computing. pp. 151158, March of 1971.

[2] R. M. Karp (1972), Reducibility Among Combinatorial Problems, In R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum. pp. 85-103.

[3] Clay Mathematics Institute, http://www.claymath.org/millennium-problems/millennium-prize-problems.

[4] Wikipedia, http://en.wikipedia.org/wiki/NP-complete.

[5] W. I. Gasarch, (June 2002). The P=?NP poll , SIGACT News 33 (2): 34-47. doi:10.1145/1052796.1052804. Retrieved 2008-12-29.

[6] W. I. Gasarch, The Second P=?NP poll, SIGACT News 74, 2012.

[7] Lance Fortnow, The Status of the P versus NP problem, Communications of the ACM. 52 (9): 78–86, 2009.

[8] P vs NP website, http://www.win.tue.nl/gwoegi/P-versus-NP.htm

[9] J. Kleinberg, E. Tardos, Algorithm Design, 2006, Pearson Education Asia Limited, pp. 475-479.

[10] T. Cormen,C. Leiserson, R.L. Rivest,C. Stein, Introduction to Algorithm, 2nd edition, MIT Press, 2004.

[11] G. J. Woeginger, Exact algorithms for NP-hard Problems: A Survey, Combinatorial optimization - Eureka, you shrink! Pages 185 - 207 , Springer-Verlag New York, Inc. New York, NY, USA ©2003

[12] M. Xiao, H. Nagamochi, Exact Algorithms for Maximum Independent Set, Algorithms and Computation, Volume 8283 of the series Lecture Notes in Computer Science pp 328-338.

[13] R. Beigel and D.Epspstein, 3-coloring in time O(1.3446^n): A no MIS algorithm, Proceedings of the 36th Annual Symposium Foundations of Computer Science (FOCS'1995),444-453.

[14] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C.H. Papadimitriou, P. Raghavan, and U. Scho¨ning, A deterministic (2 – 2/(k + 1))^n algorithm for k-SAT based on local search, Theoretical Computer Science 289 (2002) 69–83.

[15] RE.Tarjan and AE. Trojanowski, Finding a maximum independent set, SIAM J. Comput.,vol.6, No.3, Sept. 1977.

[16] J.W. Moon and L. Moser, On cliques in graphs. Israel Journal of Mathematics 3, 23–28,1965.

[17] T. Jian, An O(2^(0.304n)) algorithm for solving maximum independent set problem. IEEE Transactions on Computers 35, 847–851,1986.

[18] N. Bourgeois, B. Escoffier, V. T. Paschos, J. M. M. van Rooij, Fast algorithms for max independent set, Algorithmica 62(1-2), 2012.

[19] W. Cook, In Pursuit of the Traveling Salesman, Princeton University Press, 2012.

[20] O. Coudert, Exact Coloring of Real-Life Graphs Is Easy, in Proceedings of DAC '97 Proceedings of the 34th annual Design Automation Conference, Pages 121-126, Anaheim, California, USA — June 09 - 13, 1997

[21] K. Helsgaun, General k-opt submoves for the Lin-Kernighan TSP heuristic. Mathematical Programming Computation, 2009,doi: 10.1007/s12532-009-0004-6.

[22] W. Tian, C. Huang, X. Wang, A Near Optimal Approach for Symmetric Traveling Salesman Problem in Eclidean Space, To appear in Proceedings of ICORES 2017, Portugal, also available at arxiv https://arxiv.org/pdf/1502.00447.pdf

[23] L. Babai, Graph Isomorphism in Quasipolynomial Time, https://arxiv.org/abs/1512.03547, Dec 11, 2015.

[24] A. Cho , Science news, mathematician-claims-breakthrough-complexity-theory, http://www.sciencemag.org/news/2015/11/mathematician-claims-breakthrough-complexity-theory, Nov. 11, 2015.

[25] M. Agrawal, N. Kayal, N. Saxena, PRIMES is in P, The Annals of Mathematics, Pages 781-793 from Volume 160, Issue 2, 2004.

[26] F. L. Traversa, C. Ramella, F. Bonani and M.D. Ventra, memcomputing NP-complete problems in polynomial time using polynomial resources and collective states, Science, Vol. 1, no. 6, e1500031, Nov. 2015.

AUTHORS

Prof. WenhongTian has a PhD from Computer Science Department of North Carolina State University. He is a professor at University of Electronic Science and Technology of China. His research interests include dynamic resource scheduling algorithms and management in Cloud Data Centers, dynamic modeling and performance analysis of communication networks. He published about 40 journal and conference papers, and 3 English books in related areas. He is a member of ACM, IEEE and CCF.

Ms. WenxiaGuo is a PhD candidate at University of Electronic Science and Technology of China. Her research interests include approximation algorithm for NP-hard problems, and scheduling algorithms for resource allocation in Cloud Computing and BigData processing.

Mr. Majun He is a master student at University of Electronic Science and Technology of China. Her research interests include approximation algorithm for NP-hard problems, and scheduling algorithms for resource allocation in Cloud Computing and BigData processing.