Algorithms and Complexity

Does $P = NP$?

Peaceful Coexistence Algorithm® – Is a polynomial time algorithm

1998

Yerevan, Armenia

1. Introduction

In January, 1989 I got an order to formulate a Timetable for a concrete school. After getting aquatinted closer with the problem, it turned out that the problem is not a simple one and for its solution a principal and systematized approach is needed. Moreover, after collecting some information on this theme it turned out that the so called TimeTable problem is NP_complete and it is necessary to solve very often.

It is known that this problem is the subject of integer Programming (IP). As it is well known, the feasible region of an IP problem is a subset of the corresponding Linear Programming (LP) relaxation of that problem (the same problem, only without those constraints which state that certain variables in the problem should be integers). There are many universal algorithms for solution of LP problem, for example the well known Simplex Algorithm, or Karmarker's Algorithm, etc. However, it turns out that there is no a universal algorithm for the solution of IP problem. Moreover, very often it is practically even impossible to solve them using techniques, described in the available for us literature[1]. In every concrete case an individual approach is needed, and many such algorithms for various special cases are developed.

After investigating the corresponding literature, and making sure that a general solution for the General TimeTable Problem(GTP) simply does not exist, I set a goal to develop an universal polynomialal time agorithm for the solution of that problem. After a four years of researches, in June 1993 I finally got the first results: I formulated its theoretical foundation and developed an polynomialal time algorithm for solving GTP. Besides, I developed a computer program on the C++ language. Definite results have already existed: a timetable for a concrete school is constructed.

I find, that I obtained an universal algorithm for solving any multidimensional and complex Timetable Problem. This means, that it is possible to solve any other problems in the NPC class for any instances.

Later in the report the general formulation of GTP is brought, and the results of the solution for one particular school are introduced. I suggest, that it is rather possible to judge about the problem and give its scientific evaluation having this general formulation. In connection with this, I would be glad to know the opinions of specialists of Operations Research, especially Integer Programming and TimeTable about this work, and it is of high importance and necessity for me.

Based on this, and several other factors, my strong feeling is that the algorithm developed by me is a very strong and powerful technique, and so the formulation brought is worthy for a careful attention.

2. Optimization problems

Optimization problems seem to divide naturally into two categories: those with continues variables, those with discrete variables, which we call combinatorial. In the continues problems, we are generally looking for a set of real numbers or even a function; in the combinatorial problems, we are generally looking for an object from a finite, or possibly countable infinite, set-typically an integer, set, permutation, or graph. These two kinds of

[1] Winston, Wayne L, Operation Research. Applications and Algorithms. Duxbury Press, Belmont, California, 1991, p. 448.

problems generally have quite different flavors, and the methods for solving them have become quite divergent.

3. Linear mixed-integer programming problem(MIP)

We write the MIP as

Max (cx+hy : Ax+Gy≤b, $x\in Z^n_+$, $y\in R^p_+$), where Z^n_+ is the set of nonnegative integer n-dimensional vectors, R^p_+ is the set of nonnegative real p-dimensional vectors, and $x=(x_1,\dots,x_n)$ and $y=(y_1,\dots,y_n)$ are the variables or unknowns. An instance of the problem is specified by the data (c,h,A,G,b) , with c an n-vector, h a p-vector, A an (m·n) matrix, G an (m·p) matrix and b an m-vector. This problem is called mixed because of presence of both integer and continuous(real) variables.

The set S=($x\in Z^n_+$, $y\in R^p_+$, Ax+Gy≤b), is called the feasible region, and an $(x,y)\in S$ is called a feasible solution. An instance is said to be feasible if $S\neq\emptyset$. The function z=cx+hy is called the objective function. A feasible point (x^0,y^0) for which the objective function is as large as possible, that is, $cx^0+hy^0\geq cx+hy$ for all $(x,y)\in S$, is called an optimal solution. If (x^0,y^0) is an optimal solution, cx^0+hy^0 is called the optimal value or weight of the solution.

A feasible instance may not have an optimal solution. We say that an instance is unbounded if for any $\omega\in R^1$ there is an $(x,y)\in S$ such that $cx+hy>\omega$. We use the notation $z=\infty$ for an unbounded instance.

Thus to solve an instance of MIP means to procedure an optimal solution or to show that it is either unbounded or infeasible.

The linear programming problem(LP) max (hy : Gy≤b, $y\in R^p_+$) is the special case of MIP in which there are no integer variables.

The linear (pure) integer programming problem(IP) max (cx : Ax≤b, $x\in Z^n_+$) is the special case of MIP in which there are no continuous variables.

4. Algorithms and Complexity

Computers can only carryout algorithms: that is, precise and universally understood sequence of instructions that solve any instances of rigorously defined computational problems. It should not be a surprise that this intuitive concept of an algorithm can be defined rigorously. The corresponding mathematical object is called a Turing machine, after the British mathematical Alan M. Turing, who invented it in 1936.

If mathematical formalisms like Turing machines led the mathematicians of the 1930s to the study of undecidable problems, the digital computers of today present us with decidable problems. That is, in principle there is an algorithm that would correctly solve any instance of the problem. This is not always considered satisfactory, however, because excessive time requirements may render an algorithm completely useless.

5. Reasonable vs. Unreasonable Time

When should we consider a computational problem satisfactorily solved? The answer obviously lies in the performance of the known algorithms for this problem. If there is an algorithm for this problem that is not too time consuming-the prime criterion we use here- then the problem may be considered solved, and not otherwise. In fact, as we have pointed out, it is the rate of growth of the best known time bound that is going to determine the practical utility of the algorithm. What rates of growth, then, should we consider as acceptable solutions to computational problems?

Today there is general agreement among computer scientists that an algorithm is a practically useful solution to a computational problem only if its complexity grows polynomialy with respect to the size of the input. For example, algorithms of complexity O(n) or $O(n^3)$ are acceptable in this school of thought. Naturally, algorithms for which the asymptotic complexity is not a polynomial itself but is bounded by a polynomial, also qualify. Examples are n^{13} and n·lg(n).

To understand the significance of polynomially bounded algorithms as a class, let us consider the remaining algorithms, those that violate all polynomial bounds-for large enough instances, that is we usually refer to these as exponential algorithms, because 2^n is the paradigm of nonpolynomial rates of growth. Other examples of exponential rates of growth are k^n (any fixed k>1), n!, , n^n and n·lg(n) It is obvious that, when the size of the input grows, any polynomial algorithm will eventually become more efficient than any exponential one (see Table)

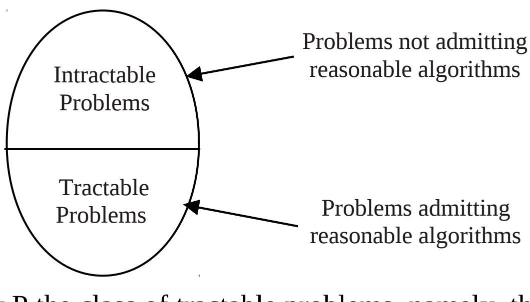
The growth of polynomial and exponential functions.

N	10	100	1000
nlog(n)	33	664	9966
n^3	1000	1,000,000	10^9
10^6n^8	10^{12}	10^{22}	10^{30}
2^n	1024	$1.27x10^{30}$	1.05x301
$n^{\log(n)}$	2099	$1.93x10^{13}$	$7.89x10^{29}$
n!	3,628,800	10^{158}	$4x10^{2567}$

6. The Classes P and NP

An algorithm whose order-of-magnitude time performance is bounded from above by a polynomial function of N, where N is the size of its inputs, is called a polynomial time algorithm, and will be referred to here as a reasonable algorithm. Similarly, an algorithm that, in the worst case, requires super-polynomial, or exponential-time, will be called unreasonable.

As far as the algorithmic problem is concerned, whereas a problem that admits only unreasonable or exponential-time solution termed intractable.The sphere of all algorithm problems can therefore be divided into two major classes as illustrated below.



We will denote by P the class of tractable problems, namely, those that admit polynomial-time algorithms. On the other hand, we will denote by NP the class of problems those that are apparently intractable, but become “tractable” by using magical no determinism. The N and P stand for Nondeterministic Polynomial time, so that problem is said to be in NP if it admits a short certificate.

We now address the question of the existence of hardest problems in NP. $X \in NP$ is said to be NP-complete if all problems in NP can be polynomial reduced to X. The set of NP-complete problems is denoted by NPC. Intuitively, an NP-complete problem is a computational problem that is as hard as any reasonable problem. The traveling salesman problem (TSP), which has hunted two generation of mathematicians with its stubborn resistance to all kinds of attacks, is an NP-complete problem. So will such hard nuts as integer linear programming, the satisfiability problem, timetable problem and a host of other well-known, difficult computational problems. This class NP-complete problems has the following very interesting properties:

a. no NP-complete problem can be solved by any known polynomial algorithm.
b. if there is a polynomial algorithm for any NP-complete problem, then there are polynomial algorithms for all NP-complete problems.

The P=NP? problem, as it is called, has been open since it was posed in 1971, and is one of the most difficult unresolved problems in computer science. It is definitely the most intriguing. either all of these interesting and important problems can be solved reasonably by computer, or none of them can. Many of the most talented theoretical computer scientists have worked on the problem, but to no avail. Most of them believe that P≠NP, meaning that the NP-complete problems are inherently intractable, but no one knows for sure. In any case, showing that an algorithmic problem is NP-complete is regarded as weighty evidence of its probably intractability.

7. Research on Complexity Classes and Intractability

In the mid-1960s people began to realize the importance of obtaining polynomial-time algorithms for algorithms problems, and significance of the dividing line of above mentioned figure became apparent. Ever since, the issues and concepts discussed in this chapter have been the subject matter of intense and widespread research by many theoretical computer scientists.

Even now and then, a polynomial-time algorithm or an exponential-time lower bound is found for a problem whose tractable\intractable status was unknown. A rather striking example is linear planning, better known as linear programming. Linear planning is a general framework within which we can phrase many kinds of planning problems arising in organizations where time, resources and personnel constraints have to be met in a cost-efficient way. The linear planning problem, it must be emphasized , is not NP-complete, but the best algorithm that anyone was able to find for it was an exponential time procedure known as the Simplex method. Despite the fact that certain inputs forced the simplex method to run for an exponential amount of time, they were rather contrived, and tended not to arise in practice; when the method was used for real problems, even of nontrivial size, it usually performed very well. Nevertheless, the problem was not known officially to be in P, nor was there a lower bound to show that it wasn't.

In the spring of 1979 the Armenian mathematician Leonid G.Khachian published an ingenious proof that a certain algorithm for LP is polynomial, thus resolving a longstanding open question. Khachian's result is based on work of other soviet mathematicians on nonlinear programming and is drastically different from most previous approaches to LP in that it almost completely disregards the combinatorial nature of the problem. The exponential-time simplex method outperformed it in many of the cases arising in practice. Nevertheless, it is show that linear programming is in P. Moreover, recent work based on this algorithm has produced more efficient versions, and people currently believe that before long there will be a fast polynomial-time algorithm for linear planning, that will be useful in practice for all inputs of reasonable size.

8. Conclusion

The final word on the intractability of NP-complete problems cannot be written until the P=NP? conjecture is resolved. Despite the great theoretical importance of this problem and the declared interest of many computer scientists, no proof of this conjecture appears to be in sight. In fact, it now seems very likely that the answer to this question will not come about without the development of entirely new mathematical methodology.

The class NPC contains around 1000 different algorithmic problems(e.g., Traveling Salesman Problem, Scheduling and Matching problems, Coloring Maps and Graphs, etc.) arising in such areas as Combinatorics, Operation Researches, Economics, Graph Theory, Game Theory, Logic, Statistical Applications. Recent scientific applications involve problems in molecular biology, high-energy physics and x-ray crystallography. A political applications concerns the division of a region into election districts.

The fact that these problems are NP-complete together with the widespread and well-founded confidence that this implies intractability-has led many researches to reassess their strategy for attacking these problems.

After about ten years of researches I finally got a fundamental polynomial time solution for the P=NP? Problem.

References

Christos H. Papadimitriou and Kenneth Steiglitz(1982) Combinatorial Optomization: Algorithms and Complexity, Prentice-Hall.

George L.Nemhauser and Laurence A. Wolsey(1999) Integer and Combinatorial Optimization, John Wiley & Sons, Inc.

David Harel(1992)Algorithmics: The Spirit of Computing, Addisson-Wesley Publishing, Inc.

Formulation and theoretical foundation

of

The General Timetable Problem

By Karlen Gharibyan

MAY 5, 2005

The purpose of this book is to provide the solution of P=NP? problem.
The P=NP? problem was emerged in 1971. This is one of the longstanding unresolved problems in computer science. Its solution is directly related to the determination of time complexity of NP-complete problems. Karlen Gharibyan presents a final and complete fundamental polynomial time solution for the P=NP? problem.

Printed in the Republic of Armenia **ISBN 0000000000**

Om tat sat

Does P=NP?

Karlen G. Gharibyan[2,3]

SPIRIT OF SOFT LLC, 48, Mamikonyanc Street, 0051, Yerevan, Armenia

May 05, 2006

SUMMARY

The P=NP? problem was emerged in 1971. This is one of the longstanding unresolved problems in computer science. Its solution is directly related to the determination of time complexity of NP-complete problems.

The purpose of this work is to provide the solution of P=NP? problem.

KEY WORDS: algorithms and complexity, timetable scheduling, computer science.

1. INTRODUCTION

In this work the **Peaceful Coexistence Algorithm (PCA)** is introduced. It is a polynomial time algorithm that the **General Timetable Problem** (GTP**)** solves for any instances. The **GTP** is NP-complete. This means, that each problem in NP may be solved in polynomial time, including 3SAT problem.

2. FORMULATION OF GTP

As the matter of fact, the problem itself is very versatile and complex, that is why, for the sake of simplification a certain secondary school is observed as an object of implementation.

In the reality the generality is kept, because all ideas, denotations, considerations, which are done for the school, in essence are the same for the general problem and may be generalized without any difficulty, if it is necessary.

[2] Correspondence to: Karlen G.Gharibyan, apt. #45, Home 5, 4-th lane, Vratsakan street, 0051,Yerevan, Armenia
[3] E-mail: karlen.gharibyan@spiritofsoft.com

2.1. Assumptions

In school, there are N_c classes, N_r classrooms, N_t teachers and N_s subjects are taught.

In school, there are ten different level classes {1,2,3,4,5,6,7,8,9,10} and a number of classes of the same level are possible. The numbers from 1 to 10 and English letters are used in order to construct additional codes(AC) for identification of all classes (e.g. if there are 3 of the first level classes and 4 of the second level classes and etc., then their addidional codes are 1A, 1B, 1C; 2A, 2B, 2C, 2D; … etc.).

Each subject of N_s subjects in school is to be taught in a separate classroom.

Each teacher in school teaches only one subject.

Each subject is considered to be either Sharing or Unsharing.

Any Sharing subject is to be taught by two teachers in two subgroups of the same class in two separate classrooms.

Sharing teachers is a group of exactly two teachers in a class, which teach the same Sharing subject in the class.

Any Unsharing subject in a class is to be taught by one teacher only.

Any Sharing subject in a class is to be taught by a group of Sharing teachers only.

Every day the beginning of all first lessons of a class is the same.

The duration of any lesson is considered to be one hour.

2.2. Input Data Variables

The following variables are used for the Input Data of GTP:

c,r,t,s,d,l - lower case indexes of entries of arrays, which are usually used as codes of a class, classroom, teacher, subject, day and lesson respectively.

N_c - the number of classes in school.

N_s - the number of subjects in school.

N_g=2- the number of subgroups into which a class is to be split if a Sharing subject is taught.

N_t - the number of teachers in school.

$S=\{S_s/s=1,\ldots,N_s\}$ - the set of subjects which are taught in school, where S_s is the name of subject s and index s is the code of the subject.

$C=\{C_c\ /\ c=1,\ldots,N_c\}$ - the set of classes in school, where C_c is AC of a class and index c is the code of the class.

$Q^+=\{q^+_c\ /\ c=1,\ldots,N_c\}$ - q^+_c is the total quantity of lessons of class c to be timetabled for a week.

$D^o=\{d_c\ /\ c=1,\ldots,N_c\}$ - d_c is the number of days, that can be timetabled for the lessons of class c for a week.

$B=\{b_c\ /\ c=1,\ldots,N_c\}$ - b_c is the first hour, that must be timetabled for the lessons of class c every day for a week.

$E=\{e_c=b_c+ceil(q^+_c/d_c)-1\ /\ c=1,\ldots,N_c\}$ - e_c is the last possible hour, that can be timetabled for the lessons of class c every day for a week (see Appendix 1).

$F=\{f_c=q^+_c$ - floor$(q^+_c/d_c)*d_c$ / $c=1,\ldots,N_c\}$ - f_c is the number of last hours that can be timetabled for the lessons of class c for a week (see Appendix 1).
$N_d=\max\{d_1,d_2,\ldots,d_{Nc}\}$ - the maximum number of days, that can be timetabled for the lessons of school for a week.
$N_l=\max\{e_1,e_2,\ldots,e_{Nc}\}$ - the last hour, that can generally be timetabled for the lessons of school every day for a week

$T=\{T_t$ / $t=1,\ldots,N_t\}$ - the set of teachers in school, where T_t is the name of teacher t and index t is the code of the teacher.

$Q=\{q_{c,t,s}$ / $c=1,\ldots,N_c$, $t=1,\ldots,N_t$, $s=1,\ldots,N_s\}$ - $q_{c,t,s}$ is the quantity of lessons of subject s to be taught by teacher t in class c, such that the following equality holds (see Appendix 1):

$$q_c^+ = \sum_{t=1}^{N_t}\sum_{s=1}^{N_s} q_{c,t,s} - \left(\sum_{t=1}^{N_t}\sum_{s=1}^{N_s}(q_{c,t,s} * \mathrm{IsSh}(c,s))\right)/N_g \,.$$

N_q - the total quantity of lessons of school:

$$N_q = \sum_{c=1}^{N_c} q_c^+ \,.$$

2.3. Output Data Variables

$Z=\{z_{c,l,d}$ / $c=1,\ldots,N_c$, $l=1,\ldots,N_l$, $d=1,\ldots,N_d\}$ - the timetable of the school for a week where $z_{c,l,d}$ is the entry of Z. At the beginning all entries of Z are 0. After constructing Z, if $z_{c,l,d}$ is not zero, then it is the code of a subject, that is to be taught by corresponding teachers in class c at hour l on day d.

$Z_c=\{z_{c,l,d}$ / $l=1,\ldots,N_l$, $d=1,\ldots,N_d\}$ - the timetable of class c for a week.

2.4. Decision problem of GTP

Form the timetable of school, by timetabling the entries of Z for the lessons of all subjects so that no overlapping exists:

subject to:

$$\sum_{c=1}^{N_c}\sum_{d=1}^{d_c}\sum_{l=b_c}^{e_c}\{z_{c,l,d} \neq 0\,?\,1:0\} = N_q; \qquad \text{Lessons constraint;}$$

$$\sum_{c=1}^{N_c}\{t \in T(c,l,d)\ ?\ 1\ :\ 0\} \leq 1; \qquad \text{No overlapping constraint for any fixed values of } t,l,d \text{ (see Appendix 1).}$$

All the variables are integers.

Appendix 2.

			1	2	3	4	5	6	$d_1=5$			
$C_1=1A$ Level 1(c=1)	$b_1=1$	1										
		2										
		3										
		4										
	$e_1=5$	5							$f_1=3$			
		6										
		7										
		8										
		9										
		10										
	$N_1=11$	11										

			1	2	3	4	5	6	$d_5=5$			
$C_1=2A$ Level 2(c=5)		1										
		2										
		3										
		4										
		5										
	$b_5=6$	6										
		7										
		8										
		9										
	$e_5=5$	10							$f_5=5$			
	$N_1=11$	11										

			1	2	3	4	5	6	$d_9=5$			
$C_1=3A$ Level 3(c=9)		1										
		2										
		3										
		4										
		5										
	$b_9=6$	6										
		7										
		8										
		9										
	$e_9=6$	10										
	$N_1=11$	11							$f_9=1$			

			1	2	3	4	5	6	$d_{13}=6$			

			1	2	3	4	5	6	d_{17}=6			
C_{17}=5A Level 5(c=17)		1										
		2										
		3										
		4										
		5										
	b_{17}=1	6										
		7										
		8										
		9										
	e_{17}=6	10										
	N_1=11	11							f_{17}=4			

			1	2	3	4	5	6	d_{21}=6			
C_{21}=6A Level 6(c=21)		1										
		2										
		3										
		4										
		5										
	b_{21}=1	6										
		7										
		8										
		9										
	e_{21}=6	10										
	N_1=11	11							f_{21}=4			

			1	2	3	4	5	6	d_{25}=6			
C_{25}=7A Level 5(c=25)	b_{25}=1	1										
		2										
		3										
		4										
		5										
	e_{25}=6	6							f_{25}=6			
		7										
		8										
		9										
		10										
	N_1=11	11										

			1	2	3	4	5	6	d_{29}=6			
C_{25}=8A Level 8(c=29)	b_{29}=1	1										
		2										
		3										
		4										
		5										
		6										
	e_{29}=7	7							f_{29}=1			
		8										
		9										
		10										
	N_1=11	11										

			1	2	3	4	5	6	d_{33}=6			
C_{33}=7A Level 9(c=33)	b_{33}=1	1										
		2										
		3										
		4										
		5										
	e_{33}=6	6							f_{33}=2			
		7										
		8										
		9										
		10										
	N_1=11	11										

			1	2	3	4	5	6	d_{37}=6			
C_{37}=10A Level 5(c=37)	b_{37}=1	1										
		2										
		3										
		4										
		5										
	e_{37}=6	6							f_{37}=2			
		7										
		8										
		9										
		10										
	N_1=11	11										