A Polynomial Time Algorithm for Hamilton Cycle (Path)

Lizhi Du

Abstract: **This research develops a polynomial time algorithm for Hamilton Cycle(Path) and proves its correctness. A program is developed according to this algorithm and it works very well. This paper declares the research process, algorithm as well as its proof, and the experiment data. Even only the experiment data is a breakthrough.**

Keywords: **Computer Algorithm, Hamilton Path, Hamilton Cycle, Polynomial Time, Computational Complexity**

Ⅰ INTRODUCTION

Finding Hamilton cycles(paths) in simple graphs is a classical NP Complete problem, known to be difficult both theoretically and computationally (see [5],[6],[7]). In spite of recent advances, the problem presents an imminent scientific challenge.

Over the past decades, Hamilton cycles and paths have been widely studied. One direction is for their existence conditions. Most existence conditions for general graphs depend on degree sums. Using these techniques, a graph is usually provably Hamiltonian only there are sufficiently many edges in the graph. Yet such results are often possible make sense that counterexamples exist when the conditions are weakened. Another direction is to design a random algorithm which usually succeeds in finding Hamilton cycles or paths with high probability, or works well for some classes of graphs.

Yet no general polynomial time algorithms have been developed for Hamilton cycles or paths. People even dare not to try to do so, only because the problem is NP Complete and its polynomial time algorithm means NP=P.

So, the challenging job still is: to develop a polynomial time algorithm for all general graphs, i.e., no part of graphs have specialties on this algorithm so as to make the algorithm cannot work on them, and to prove the algorithm is correct theoretically.

The main problem is: why many random algorithms work well on most graphs, only cannot work on a little part of graphs. Why we cannot overcome the little part? My point of view is: the little part does not have any strong logical reason to be special.

I develop a polynomial time algorithm for finding Hamilton cycle(path) in undirected graphs. This algorithm works very smooth for all kinds of undirected graphs, i.e., for graphs with the same vertex size, except a little cases which very quick, their run time are very close to one another. So using my algorithm, there is no graph to cost much more time than others, i.e., there is no “dead angle” for my algorithm. My algorithm looks like a random algorithm. Yes, it is a random algorithm. But if I can exhaustively compute all undirected graphs using my algorithm in all possible steps, of course, the random algorithm becomes a general algorithm. I find a way to compute “all” graphs by only computing a limited number of graphs. This is my big breakthrough. I cost many years time to find it. A program on this algorithm has been tested over a hundred million times for graphs whose vertex number is between 100 to 10000, no fails.

Manuscript received January 12, 2010.
Author Lizhi Du is with the College of Computer Science and Technology, Wuhan University of Science and Technology P.R. of China
Tel: 86 13554171855 Email: dlz95@sohu.com

Ⅱ STUDY PROCESS AND METHODS

A. Why the NP problem is so difficult?

Since the NP problem comes to the world, numerous scholars have been studying it. However, up to now, people can't still confirm: is the NP equal to the P or not? Where is the reason?

In computer science, divide and conquer is an important algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. After divided a big part into small parts and studied each small part, we should combine all the small parts together to study, because these small parts have connections to each other, in order to get a final result, all these connections must be considered thoroughly. The NP-complete’s difficulty lies that: its parts’ connections are very intensive and complicated. If an algorithm(polynomial time) fully considers these connections, still can not get the end result, it of course means that NPC can not be solved in polynomial time. The question is: by now, no any algorithm(polynomial time) can fully handles the intensive and complicated connections among all parts of a NPC. For example: using a tree to stand up an algorithm for a NPC. In

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

layer 0, it resolves to N parts, in layer 1, it resolves to N-1 parts, and so on. Thus, its time complicacy is N!. However, because all nodes in this tree have many relations to each other, a node can be got from its father, it may also be got from its brothers or grandfather. So, many nodes repeat the others' information. If we can do our best to delete the redundant nodes, the time complexity would be much less than N!.

B. Study process

To try to overcome the NP problem, one has two ways: or thinks that the NP is unequal to the P, then give the united proof theoretically or prove some NP problem time complexity's low bounds is higher than polynomial time; or thinks the NP is equal to the P, then proves any NPC has a polynomial time algorithm. The current academic circles incline to think that the NP is unequal to the P(see[2]). One of their reasons is: the computation process should be different from the test process in time cost. However, the polynomial includes very wide scope, one power of N is a polynomial, 10,000 powers of N is also a polynomial. Therefore there is reason to think such possibility: low polynomial time for "test", high polynomial time for "compute". I strongly belive that NP is equal to P. Why? What is my logic? First, by now, a lot of NPC have been discovered, and the number may increase continuely. If any one of them has polynomial time algorithm, it means all the NP problems have polynomial time algorithm. This fact itself strongly implies that: the complexity of NP problems is not very uneven, but is uniform. They may have some uniform rules(of course each one has its specialty). These uniform rules are meaningful only in polynomial, especially when considering that many NP problems in most cases can be solved quickly. Secondly, let's take an example: Hamilton path. If a undirected graph has only N nodes and N-1 lines, sure, we can quickly get the path. Along with the increment of its lines, the complicacy increases. But the line number is limited, less than N times N in total. And the line number increases, the possibility to get more paths also increases. If I can discover the intricate relations between these two factors, I may get a path within polynomial time.

After determining the research direction, next step is to choose an available NPC as the research object. A good choice can make my job much easier than other choices. By a lot of comparing, contrasting and analysing, I choose the Hamilton path(cycle) as my study object. Reasons:(1) Hamilton path is a strong NPC(see [3]). Usually a strong NPC is harder than un-strong one, but as it is strong, I can try to get a polynomial time algorithm for it in any way,the result is always valid;(2) I use "divide" and "conquer" method to do it. After dividing,the mutual relative imformations among all parts is easier to discover for Hamilton path than for other NPC, because Hamilton path can "naturally" show its relative information by showing that whether two vertices are connected. Thus we donot need to cost much time to get this information. Especially, by comparing different combinations and their different results, a lot of relative information can be got easily.

C. Principle and Method

This algorithm uses the principle of "divide" and "conquer", does its best to utterly make use of the relative informations among all parts. First, using the idea from the Greedy Algorithm, for each step, the algorithm try to get the final result as quickly as possible. In order to limit its time to polynomial, the algorithm draws references from the State-Space Method. The state-space contains at least one final result(if exist). For each element in the state-space, the calculation time is polynomial and all elements number is polynomial, so the algorithm is polynomial. In order to limit the number of elements in the state-space to polynomial, the algorithm draws references from the Genetic Algorithm. In each step, optimize the elements in the state-space, only keeps and calculates the optimized elements. The biggest speciality of this algorithm is: dynamically to combine some parts, dynamically to transform them, by comparing different combinations' results, to deduce the relative informations in as less as possible time, so as to make all the algorithm polynomial.

My algorithm's key technologies are: 1)It is based on a lot of generalizations, but its correctness and polynomial complexity have been proved.2)In its calculating process, it dynamically decides its calculating direction according to new produced messages. Dynamic adjustment is my algorithm's big specialty. 3)My algorithm has found a method to transform "infinite" to "finite". This is a very new method which does not appear before. I deduce this method may be a common way to solve all NPC.

D. Algorithm

(1) Assume the graph has at least one Hamilton cycle. I first get a cycle which includes all n vertices in any way. In this cycle, some two neighbor vertices may not be adjacent, I call this point "break point". For each break point, I add an edge between the two neighbor vertices except for one, i.e. only one break point to be left(remember the added edges, later, each time delete one, then do the algorithm from a NEW start). I call the one break point being left "main break point". Now my algorithm only needs to handle this one break point. Each time handles one break point, at most n times(because the number of the added edges at most n). So it does not affect the polynomial.

(2) At each step, cut a segment from the main break point, insert the segment in some place of the cycle. How to cut and insert? The rule is: make the number of new break points the least, and one new break point must be different from all former main break points(this new one as the new main break point ,I use a symmetric matrix to record all main break points, this guarantees my algorithm's polynomial). Notes: when calculating the number of new break points for getting the least, if more than one case have the same least number, compare their next step(only one time "next step", do not need to continue to do so);also, avoid inserting the same segment in different places consecutively. Then with the

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

new main break point, do the same job until getting a Hamilton Cycle. If one step does not get a new break point and there are some break points at other place, get one of them as the new main break point and continue.

Example 1:

dhijklabcmpefqr*u vertex r and u are not adjacent, as the main break point.

Other edges: d-q, h-c, i-l, j-u, a-p,m-f,e-r.

S

tep 1: dhijklabcmperqf*u the pair f*u is different from r*u,cut the segment

rq, insert it between e f. Case 1—one new break point.

Step 2:dhijklabcmfqrep*u

Step 3:dhijklaperqfmcb*u

Step 4:dhi*bcmfqrepalkju

Step 5:dhilaperqfmcb*kju

Step 6:dhilabcmfqrep*kju

Step 7:dhi*perqfmcbalkju

Step 8:dhi*fqrepmcbalkju

Step 9:dhi*rqfepmcbalkju

Step 10:dhilabcmpefqr*kju

Step 11:dhilabcmperqf*kju

Step 12:dhilabc*fmperq*kju Case 2—two new break points, you cannot get less. Choice q*k as the new main break point, another break point can be the same as formers.

Step 13:dhilabc*fqrepm*kju m*k as the new main break point, cannot choice c*f at current step

Step 14:dhi*mperqf*cbalkju

Step 15:dhi*qrepmf*cbalkju

Step 16:dqrepmf*cba*hilkju the main cycle has 3 parts

Step 17:dqrepmf*abchilkju the main cycle has 2 parts

Step 18:dqrefmpabchilkju Case 3—no new break point, only one part

E. Proof Sketch:

My main proof idea is: using limited number of graphs, each graph only has 12 vertices(also 11,10), to constitute unlimited number of all kinds of graphs. Then I only need to prove all cases of graphs with 12 vertices(also 11,10) to fit my algorithm(need to combine and to separate) by calculating all.

Ⅲ EXPERIMENT DATA

Though I have theoretically proved this algorithm. Here I give the experiment data.

A program on this algorithm has been designed in VC++. Not losing generality, for a undirected graph with N nodes, node number is 0,1,2…N-1, the algorithm calculates Hamilton path from node 0 to node N-1. The input data is randomly produced un-directed graphs. In order to test the program, each graph includes a randomly produced Hamilton path which the program does not know. I have tested the program over one hundred million inputs, no one fail. The data is as Table 1 (computer:HP PC, CPU:Intel 1G, Memery:1G):

Table 1 experiment data

number of Nodes	calculation times on different inputs	success times	fail times	average run time
100	100000000	100000000	0	0.0014 second
1000	10000000	10000000	0	0.07 second
10000	10000	10000	0	48 seconds

Many NP-complete problems turn out to be easy for random inputs (see[9],[10]).Hamilton path problem is solvable in linear average time (see[8]).But the algorithm in [8] is a random algorithm. My algorithm is a fixed algorithm that fits all graphs, and is proved theoretically.

When randomly producing the un-directed graphs, I try to make the graphs as hard as possible to calculate. A lot of tests show that when its average vertex degree is about 3 or 4 or 5, the graph is hardest to calculate(Even its biggest vertex degree is 3, this problem still is NP-Complete [4]). With the vertex number much greater, the hardest average vertex degree may increase very slowly. Also I try to produce each edge with different probability in a graph independently.

REFERENCES

[1] Alfred V.Aho.etc. The Design and Analysis of Computer Algorithms[M]. New York: Addison Wesley Publishing Company，1974.

[2] Chen-zhiping, Xu-zongben. Computing Mathmatics[M]. Peking: Science Express,2001.

[3] Christos H. Papadimitriou. Computational Complexity[M]. New York: Addison Wesley Publishing Company，1994.

[4] Sara Baase etc. Computer Algorithms: Introduction to Design and Analysis[M]. New York: Addison Wesley Publishing Company，2000.

[5] R.Diestel,Graph Theory,Springer, New York 2000

[6] M.R.Garey,D.S.Johnson,Computers and Intractability:A Guid to the Theory of NP-Completeness,Freeman,San Francisco,1979

[7] L.Lovasz,Combinatorial problems and exercises, Noth-Holland, Amsterdam(1979)

[8] Yuri Gurevich and Saharon Shelah Expected computation time for Hamiltonian Path Problem SIAM J. on Computing 16:3 (1987) 486—502

[9] Yuri Gurevich,Complete and Incomplete Randomized NP Problems 28th Annual Symposium on Foundations of Computer Science(1987), 111-117.

[10] D.Johnson,The NP-completeness column-an ongoing guid,Journal of Algorithms 5(1984), 284-299

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)