From NP to P
Musings on a Programming Contest Problem

Edward Corwin
Antonette Logar
Mathematics and CS
SDSM&T
Rapid City, SD 57701
edward.corwin@sdsmt.edu

ABSTRACT

A classic analysis of algorithms problem is to find the longest path through a graph that visits every node at most once. Without restrictions, this version of the problem is known to be NP-complete. However, variations on the longest path problem are known to be computable in polynomial time. For example, only allowing graphs without cycles yields a solution in time $O(n^2)$. An ACM regional programming contest problem presented a variation on the longest path problem in which the nodes in the graph were numbered such that nodes were connected by an edge if and only if the numbers assigned to the nodes differed by one digit. This work examines some of the variations on the longest path problem which would take the problem from NP to P and examines the conjecture that the restriction placed on the graph as given in the contest is not one of those variations.

1. INTRODUCTION

In the areas of Analysis of Algorithms and Theory of Computation, seeming small changes to a problem can change the complexity from P to NP or vice versa. As a quick review, recall that a problem is said to be in the class P if it can be solved in polynomial time by a deterministic Turing machine. These are problems that can likely be solved in a reasonable time for large data sets. Problems are said to be in NP if they can be solved in polynomial time by a nondeterministic Turing machine. Note that this means that P is a subset of NP, but it is not known if P is a proper subset of NP. That is, it is not known if P and NP are equal. However, it is widely believed that NP does indeed include problems that are not in P, in particular problems that are NP-complete. These problems have the property that every problem in NP can be reduced to them in polynomial time implying that a polynomial time solution to any NP-complete problem would give polynomial time solutions to all problems in NP and show that P = NP. This is a very brief overview of the terms, much more can be found in any text on the Theory of Computation and many texts on Algorithms, for example [1-3]. Perhaps the most widely discussed NP-complete problem is the Traveling Salesman Problem (TSP) (technically the decision version of TSP, but that is not important for this discussion). Many other problems are known to be NP-complete, essentially equal in difficulty to TSP, including the Longest Path Problem discussed below.

2. THE LONGEST PATH PROBLEM

The Longest Path Problem (LPP) is the problem of finding the longest simple path in a graph. That is, find the longest sequence of adjacent vertices so that no vertex appears more than once in the sequence. This problem is known to be NP-complete [3]. As mentioned above, this means that it is very unlikely that a polynomial time solution exists making it infeasible to solve this problem for large graphs. For small graphs, a simple recursive algorithm that exhaustively searches all possibilities will give the correct answer, but this algorithm will take far too much time to be reasonable for graphs with many vertices.

A quick approximation shows how this is not feasible for even modestly large graphs. A brute force solution to LPP can be on the order of $n!$ since it could try every permutation of nodes to see which gives the longest path. As an example of how slow this can be, consider the fact that 10! = 3,628,800. Assume that such a problem can be solved in 1 millisecond. Next, look at 20! which is about 2.4×10^{18} or about 6.7×10^{11} times as big as 10!. This would take more than 6×10^{8} seconds to solve which is 10^{7} minutes, 1.7×10^{5} hours, 694 days, or 2 years. Moving to size 30 would multiply this by well over 10^{10}, meaning billions of years to solve. Making computers 1,000 times faster or networking 1,000 computers would only allow the problem size solvable in a given time to increase by 3 or so for each improvement. Thus, this problem is not feasible to solve for large values of n using this brute-force approach.

3. VARIATIONS ON NP-COMPLETE PROBLEMS

Sometimes, a seemingly minor change to an NP-complete problem can move the problem into P. For example, the Bitonic TSP changes TSP to require the graph to be on the Euclidean plane and that the path be strictly left to right followed by strictly right to left. This problem can be solved in polynomial time by a dynamic programming algorithm [2]. Similarly, the longest path problem can be modified to restrict the graph to being directed and acyclic. For a directed acyclic graph (dag), a dynamic programming algorithm can solve LPP in polynomial time. This works since the algorithm can keep track of the longest path from any given vertex *v* by visiting all vertices adjacent to *v* and not have to worry that the longest path from an adjacent vertex goes through *v*.

```
best_length(start node)
  {
  if (done previously)
    return previous value;

  for all nodes adjacent from start node
    {
    length = best_length(adjacent node) + 1;
    if (length > best so far)
      {
      best so far = length;
      }
    }  // end for

  return best so far and mark as previously done;
  }
```

Figure 1: Pseudocode for dynamic programming solution

The best_length function will be called starting with every node. This loop is O(N) where N is the number of nodes in the graph. The loop and recursive call in the function will be executed E times where E is the number of edges in the graph (assuming adjacency lists are used). Thus, this algorithm is O(N + E) or $O(N^2)$ since there are at most N^2 edges in any digraph, fewer for dags.

A fun little variation on the LPP problem for dags is given in [4]. The problem is to find the longest alphabetically-ordered edit step ladder between two given words. An edit step ladder is a sequence of words such that any two consecutive words differ by changing, adding, or deleting one character. That is, COAT, BOAT, BRAT, RAT, RAY, BRAY, FRAY is an edit step ladder from COAT to FRAY. However, notice that the proposed problem requires the words to be in alphabetical order, so this is not allowed as a solution to this version of the problem. The requirement that words be in alphabetical order makes the underlying graph directed and acyclic and allows the problem to be

solved in polynomial time by a dynamic programming algorithm similar to the algorithm given above.

There are other types of graphs for which polynomial-time solutions to LPP are known, such as a cactus, a graph in which any edge is in at most one cycle. For a discussion of such graphs, see [5].

4. THE CONTEST PROBLEM

In the 2006 North Central Regional ACM Programming Contest, problem 9 required contestants to find the length of the longest sequence that could be formed from a given set of five-digit integers that changes one digit at each step in the sequence and never repeats a value. This is equivalent to finding the longest path in a graph where nodes are identified by five-digit integers and adjacent if the two integers differ in one digit position. Thus, for an input case consisting of the six integers 59304 58304 8300 48304 19304 58303, the correct answer would be four since 19304 59304 58304 48304 has length four and no sequence has length five. Note that there are other sequences of length four, but that does not change the fact that the longest sequence is length four. Since the maximum data set size for this problem was given as 10000, an efficient solution is required.

The programming contest problem does not require the values in the sequence to be increasing. This means that there are cycles in the graph corresponding to most input data sets. For example, the data set 12345, 12346, and 12347 is fully connected and contains cycles of length two and length three. (Note that path length is characterized by the number of vertices visited, not the number of edges, as required by the statement of the contest problem.) This means that the solution to the very similar sounding edit step ladder problem from [4] does not work here since that problem did require increasing sequences, thus removing all cycles from the underlying graph. If the contest problem statement had required the sequences to be increasing, then the problem would have been solvable in polynomial time using the dynamic programming algorithm given above.

While cycles are not allowed in the longest sequence, the presence of cycles in the graph means that the dynamic programming algorithm given above fails. The longest sequence from a given node *v* is one step longer than the longest from any of its neighbors only if the longest path from its neighbors does not go through *v*, thus, we cannot mark nodes as visited and not visit them again. For example, given the data set 12345 12346 22346, the longest path from 12346 contains two vertices (either 12346 12345 or 12346 22346), but the longest path in the graph contains three vertices. Notice that to find the longest path starting at 12346, we cannot find the longest path from each of its neighbors and simply add one. The longest path starting at 12345 is three which would imply that the longest path from 12346 is four – an obviously incorrect solution. The problem stems from the fact that the longest path from 12346 includes the path from 12345 includes 12346 already. The dynamic programming approach only works if cycles are not possible.

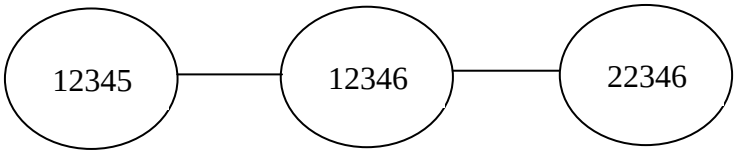

Figure 2: A graph for which the dynamic programming solution fails

This means that the contest problem is not obviously equivalent to the dag version of LPP, leaving the question as to whether it is equivalent to the full LPP and, hence, NP-complete. Unfortunately, there is no easy answer to this question since not all graphs can be represented as the type of graph in the contest problem. For example, consider a graph that is a cycle of length four with no additional connections. To represent this as an instance of the contest problem, start with 12345 and change one digit at a time for four steps and return to 12345. Since it doesn't matter which digit is changed first and what it is changed to, start with 12345 62345. Next, change a different digit or there is a triangle in the graph which violate the assumption that the resulting graph would a cycle with four nodes. This gives 12345 62345 67345. Next, return to 12345 in two steps. The only way to accomplish this is to make the next step 17345. This gives the cycle 12345 62345 67345 17345 12345.

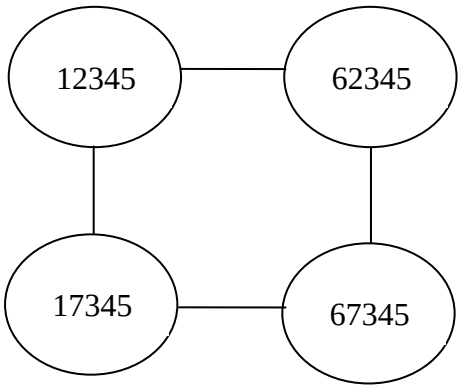

Figure 3: A four cycle is possible

A complete graph on four vertices can be represented by four values that differ in the first digit such as 12345 62345 72345 82345.

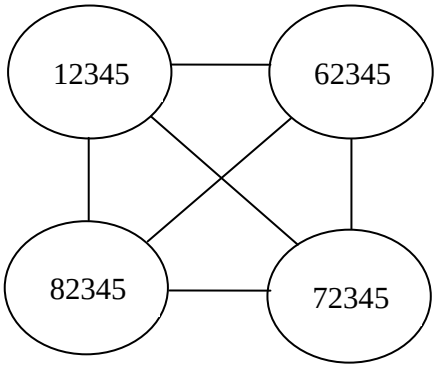

Figure 4: The complete graph on four vertices is possible

But, now consider the four cycle with one diagonal. Without loss of generality, start at one of the vertices of degree two and number that 12345 and run through the cycle argument given above. That is, start 12345 62345 and then 67345 since the first and third vertices are not adjacent. Now, the next vertex needs to be adjacent to the other three which is impossible. Two be adjacent to both 12345 and 62345, it must end with 2345, and to be adjacent to 62345 and 67345, it must be of the form 6*d*345 for some digit *d*. The only number that satisfies both requirements is 62345, which is not allowed to be reused.

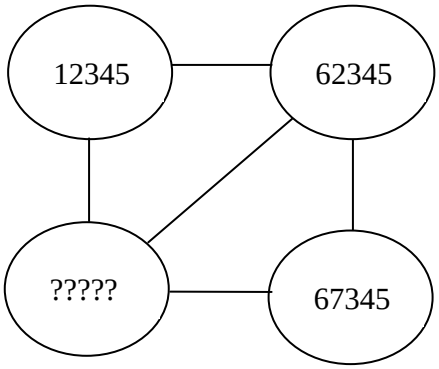

Figure 5: The complete graph on four vertices with one edge removed is not possible

The above discussion implies that we cannot directly infer that the contest problem is NP-complete from the fact that LPP is NP-complete. The contest problem is a proper subset of the full LPP, and, conceivably might, like the dag version of LPP, not be NP-complete. [Again, we assume here that P is a proper subset of NP else all problems in P are NP complete.] In order to prove that the contest problem is NP-complete, it would need to be shown that a problem already proven to be NP-complete can be reduced in polynomial time to the contest problem. This has not yet be accomplished.

Many possible solutions to the contest problem have been attempted, but, so far, it has resisted solution in reasonable time for large data sets. Programs that run fast do not always find the longest path, and problems that find the longest path do not run in a reasonable amount of time. For example, a program that assumes the values must be increasing runs fairly quickly but only finds a path of length 14 in a particular data set with 10000 nodes, while a heuristic using a depth first search that is not guaranteed to find the longest path finds a path of length 6310 for the same data set. The fact that complex cycle structures can exist in contest problem instances while graph classes for which LPP can be solved in polynomial time have severe restrictions on cycle structures adds another piece of evidence suggesting that no polynomial-time solution exists.

5. CONCLUSION

The Longest Path Problem for arbitrary graphs is known to be NP-complete. However, variations can make the problem solvable in polynomial time. The variation considered here requires the nodes in a graph to be numbered with five-digit integers and requires that nodes are adjacent precisely when their integers differ in one digit position. It has been shown that this does define a subset of graphs, an arbitrary graph cannot be represented with these restrictions, and therefore potentially reduces the complexity of the LPP for this class of problem. However, a survey of variations that are solvable in polynomial time has failed to uncovered one which allows for complex cycle structures, where cycles can intersect at many edges and with many other cycles. It has also been shown that the variation discussed here does indeed allow for complex cycles. If the problem had restricted valid paths to those visiting nodes in increasing numerical order, the cycles would have been removed and the problem would clearly be in P. As stated, evidence, including timings of a large number of cases, indicates this variation is NP-complete but that has not yet been conclusively proven. It is also not known if the degree of complexity of the cycle structure in a graph can be quantified nor if there is a point, or complexity value, beyond which a variation switches from P to NP. More work needs to be done to test these hypotheses.

References

[1] Brookshear, J. G., Theory of Computation, Benjamin/Cummings, 1989.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., Introduction to Algorithms, 2nd edition, MIT Press, 2001.

[3] Sipser, M., Introduction to the Theory of Computation, PWS, 1997.

[4] Skiena, S. S., and Revilla, M. A., Programming Challenges: The Programming Contest Training Manual, Springer 2003.

[5] Uehara, R and Uno, Y., On Computing Longest Paths in Small Graph Classes, 2005, www.jaist.ac.jp/~uehara/pdf/longest.pdf