Efficient Optimization of Diameter and Average Shortest Path Length of a Graph using Path Count Index

Hiroshi Inoue
inouehrs@gmail.com

IBM Research – Tokyo

Abstract. In this talk, we present an efficient method to find an undirected and unweighted graph having a smaller diameter and a shorter average shortest path length (ASPL) with given order and degree. For this problem, local search is one of the most common techniques to find a better solution. However, due to the high computation cost of the objective functions (diameter and ASPL of a graph), the naive local search is not practical for large graphs. Our proposed technique reduces the computation cost drastically by using alternative objective functions that estimate the real objective functions. The key for the higher performance is that what we actually need for the local search is not the diameter and ASPL of the graph, but the changes of them due to a small modification applied for the graph; the changes can be calculated using only the local information around the modified edges without involving the whole graph. To calculate the changes without fully recomputing the node-to-node distances, a new data structure called *Path Count Index* is introduced. Our technique accelerated the local search significantly (by order of 10^5 in maximum). Although the Path Count Index consumed additional memory space, it was possible to execute the local search for graphs with up to 10,000 nodes on a commodity PC with 4 GB of system memory.

1 Introduction

Local search is one of the most common heuristic optimization methods that can be applicable for wide range of combinational optimization problems including the order/degree problem of *GraphGolf*. The basic approach of the local search is quite simple: 1) make a small modification to the current solution, e.g. with 2-opt method in graph problems, 2) compute the objective function for the modified solution, 3) take the modified solution as the current best solution if the objective function is improved. If the objective function is not improved revert the modification. In the order/degree problem, however, the naive local search is not practical for large graphs due to the high computation cost of the objective functions (diameter and ASPL of a graph). Calculating these objective functions require all pairs shortest path information whose computation cost is $O(n^3)$ by Floyd–Warshall algorithm. Here n shows the number of nodes in the graph. For example, with the largest graph size of *GraphGolf* this year, n = 100,000, our implementation takes about two hours to compute the all pairs shortest path of a graph using eight threads and is too slow to use in the local search. Hence, it is critical to reduce the computation cost of the objective functions to efficiently find a graph having smaller diameter and ASPL. The proposed technique reduces this computation cost drastically by using lighter-weight alternative functions that give mostly accurate estimates for the real objective functions.

2 Path Count Index

The proposed optimization method speeds up the local search especially for large graphs by reducing the computation cost of the objective function. The key for the higher performance is that what we actually need for the local search is not the current value of the objective functions (i.e. the diameter and ASPL) for the graph, but the changes in the objective functions due to a small modification made in the graph. Since the changes can be calculated from the local information around the modified edges without involving the entire graph, it is quite efficient for large graphs compared to recomputing the objective functions. To compute the changes without fully recomputing the node-to-node distances, a new data structure called *Path Count Index* is introduced.

The Path Count Index is a table that holds the number of paths for all combinations of {source node, destination node, path length}. We limit the path length in the table up to the predefined threshold L_{max} and we don't count paths that include a cycle. Obviously, the shortest path length between two nodes can be easily obtained from the Path Count Index by looking up the non-zero entry having the shortest path length. If there is no non-zero entry for the given node pair, we assume that the shortest path length is $L_{max}+1$. This simple table is used to compute the changes in the diameter and the ASPL when an edge is added to or removed from the graph as follow. When adding an edge to the graph, we enumerate all paths newly created and update the Path Count Index. While updating the Path Count Index for each new path, we check whether this new path is the shortest path between the two nodes. If the new path is the shortest path, the difference between the previous and the new shortest paths is the reduction in the total shortest path length. We execute similar steps while removing an edge from the graph; we can easily calculate the increases in the total shortest path length. This information is enough to compute the changes by 2-opt; one step of 2-opt just removes two edges and adds two edges. This computation is an estimation of the change because we limit the maximum path length stored in the Path Count Index to limit the overhead in both memory and CPU time for bookkeeping. But we empirically found that this estimation works well in random graphs with L_{max} equal to or slightly less than the diameter of the graph. If we set L_{max} equal to (or larger than) the diameter of the current graph, we can detect the changes in the diameter while computing the changes in the total path length at the same time.

3 Implementation and Performance

We implemented this approach in C++. Each entry of the Path Count Index is limited to up to 64-bit and we pack entries for all path lengths by using bit operations. The current implementation is not parallelized for all part except for the rarely executed exact computation of the node-to-node distances. The talk will cover more detail of the implementation including additional optimization techniques and meta heuristics. Though the Path Count Index consumed additional memory space, it was possible to execute the local search with the Path Count Index for graphs with up to 10,000 nodes on a commodity PC with 4-GB memory and up to 100,000 nodes on a machine with 64-GB memory. The improvements in performance were impressive; for largest graph size of this year n = 100,000 nodes and degree d = 20, for example, the execution time of one 2-opt operation reduced from more than 10 hours to about 0.1 sec (speed up in order of 10^5) using one thread.

Although we introduce the Path Count Index for the order/degree problem, we expect that our base idea is potentially useful for wider range of problems on dynamic graphs.