Assignment 2
COMP6741: Parameterized and Exact Computation

2016, Semester 2

Assignment 2 is based on group work. Each group consists of 4–6 students. Post on the Forum if you are looking for a group to join, or if your group is short of members.

For the solutions to this assignment, you may rely on all theorems, lemmas, and results from the lecture notes. If any other works (articles, Wikipedia entries, lecture notes from other courses, etc.) inspired your solutions, please cite them and give a list of references at the end.

If you have questions about this assignment, please post them to the Forum.

Due date. This assignment is due on Friday, 7 October 2016, at 23.59 AEST. Submitting x days after the deadline, with x > 0, reduces the grade by 20 · x per cent.

Submission. Submit a TAR(.GZ) or ZIP archive with the following files

- a report in PDF format (the first page should contain the names and Student IDs of each group member), and
- all source files of the implementations, and
- two shell scripts mycompile.sh and myrun.sh such that mycompile.sh compiles the code (in case you use an interpreted language, this shell script might simply do nothing), and myrun.sh executes the second algorithm implementation on a randomly generated instance.

Submit this archive using the command

give cs6741 a2 <myarchive>

from the CSE network, or use the WebCMS3 frontend for give.

Assignment

This assignment centers around discovering small diameter parts of a graph.

The distance between two vertices u and v in a graph G is the length of the shortest path between u and v, i.e., the smallest number of edges one needs to traverse to reach v from u. The diameter of a graph G is the largest distance between any two vertices in G. We will consider the following problem.

<table>
<thead>
<tr>
<th>Small Diameter Components (SDC)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A graph $G = (V, E)$ and non-negative integers $k$, $\ell$, and $d$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is there a subset $S \subseteq V$ of at most $k$ vertices such that $G - S$ has at least $\ell$ connected components and each connected component of $G - S$ has diameter at most $d$?</td>
</tr>
</tbody>
</table>

Let $G = (V, E)$ be a graph and let $d$ be a non-negative integer. A $d$-diameter set of $G$ is a subset of vertices $S \subseteq V$ such that $G[S]$ has diameter at most $d$. A vertex subset $S \subseteq V$ is a maximal $d$-diameter set if it is not a subset of another $d$-diameter set. Let $a \in V$ be a vertex in the graph $G$. An anchored $(a, d)$-diameter set of $G$ is a diameter-$d$ set that contains the vertex $a$. We also consider the following problem.
Anchored Small Diameter Component (ASDC)

Input: A graph \( G = (V, E) \), a vertex \( a \in V \), and non-negative integers \( s \) and \( d \).

Question: Does \( G \) have an anchored \((a, d)\)-diameter set of size at least \( s \)?

1. For any \( d, n \geq 0 \), denote by \( \kappa_d(n) \) the largest number of maximal \( d \)-diameter sets a graph on \( n \) vertices can have. In other words, there is a graph on \( n \) vertices that has \( \kappa_d(n) \) \( d \)-diameter sets, but no graph on \( n \) vertices has more than \( \kappa_d(n) \) \( d \)-diameter sets. Let \( \kappa(n) := \max_{0 \leq d \leq n} \{ \kappa_d(n) \} \).

   (a) Upper bound \( \kappa(n) \) as a function of \( n \). Aim to show that \( \kappa(n) = O(1.4656^n) \). [10 points]

   (b) Lower bound \( \kappa(n) \) as a function of \( n \). Aim to show that \( \kappa(n) = \Omega(1.1892^n) \). [5 points]

2. Design and analyze an FPT branching algorithm for SDC with parameter \( k + d \) for the case where \( \ell = 1 \). [10 points]

3. Implement this algorithm in C, C++, Java, or Python. (If you failed to solve the previous question, implement some other branching algorithm for SDC.)

   (a) Write high-quality, re-usable code. [5 points]

   (b) A first implementation should implement the algorithm as is, without trying to optimize the running time.

   (c) In a second implementation, try to optimize the running time in practice. In your report, answer the following questions. What data structures do you use for graphs? How do you ensure that small modifications to the graph do not take too much time? How do you pass the instance in the recursive calls of your branching algorithm? Is it necessary to create a copy of the graph for each recursive call? Do you use any heuristics for branching? Do you perform simplification rules (if any) at every recursive call or only periodically? Can you re-use some information from polynomial-time computations in other branches? [20 points]

   (d) Implement a method to generate instances for SDC where \( d \) is an even number based on the following.

      • Generate \( x = k + 1 \) random trees \( T_1, \ldots, T_x \) with depth \( d/2 \) such that the tree \( T_i \) has two leaves \( u_i \) and \( v_i \) at distance \( d \) from each other.

      • Add a set \( S \) of \( k \) vertices (for Yes-instances) or \( k + 1 \) vertices (for No-instances).

      • For each \( i \in \{1, \ldots, x\} \), add an edge between \( u_i \) and a random vertex from \( S \).

      • Randomly add edges to the resulting graph while making sure that the distance between every vertex \( u_i, v_i \in \{1, \ldots, x\} \), and every vertex in \( S \) remains at least \( d + 1 \), and making sure that no added edge has an endpoint in both \( T_i \) and \( T_j \) for any distinct \( i, j \in \{1, \ldots, x\} \).

   Refine and discuss this implementation in your report. How do you generate (pseudo-)random objects? How do you ensure that \( v_i \) is at distance \( d \) from \( u_i \)? How can you extend this method to handle odd values for \( d \)? How do you quickly decide whether the addition of an edge would decrease the distance between a vertex \( v_i \) and \( S \)? How dense can you make the resulting graph? [10 points]

   (e) Run both implementations of the SDC algorithm on such generated instances with a time-out of 10 minutes. Start with instances where \( d = 4, k = 15 \), and \( |V| = 500 \) and vary the density of the graphs (number of edges divided by number of vertices), and increase or decrease the values of \( d, k, \) and \( |V| \) based on how fast the previous instances could be solved. Describe your environment (processor, programming language, how do you measure execution time, etc.) and save the instances in files so that your results will be reproducible. How many instances do you generate? Graph the results (number of solved instances, average time, etc.) according to the value of the parameter and/or the number of vertices. Compare and interpret the results. Do they match your expectations? Do Yes- or No-instances seem harder to solve? Which instances seem particularly hard to solve? [15 points]
4. Show that ASDC has a polynomial kernel for parameter $s + d$ when $d > 1$. [15 points]

5. Show that SDC is W[1]-hard for parameter $\ell$. [5 points]

6. Show that SDC is para-NP-hard for parameter $d$.
   A parameterized problem is para-NP-hard if there exists a constant value $c \in \mathbb{R}$ such that the problem is NP-hard when the parameter is equal to $c$. [5 points]

**Bonus Question**

- What is the parameterized complexity (FPT or W[1]-hard) of SDC for parameter $k + \ell$? [5 points]