Very Large Graph: Report

Thibaud Chominot

thibaud.chominot@epita.fr

Abstract

In this report, I propose 3 strategies to compute an approximation of the diameter and radius of a very large, sparse graph. An approximation of some center and diametral vertices is also computed in some capacity. The implementation is built on top of the igraph C library). I provide outputs of running the different strategies on graphs available at http://data.complexnetworks.fr/Diameter/ and then compare and interpret these results.

1 Introduction

The goal is to find approximations for the diameter and radius as well as some center vertices for graph with a very large number of nodes V and a number of edges E very small relative to V^2 (i.e a large, sparse graph). The diameter and radius are defined as the maximum and minimum eccentricities in the graph (respectively), therefore such approximations techniques are based on fast ways to find the eccentricity of a node, namely the BFS traversal.

1.1 Basics

Let's clarify some terminology for the rest of the report first:

- A path between two vertices u and v is a sequence of edges, with the first edge starting from u and the last edge leading to v.
- The distance $dist(u, v)$ is the length of the shortest path from u to v.

- The eccentricity $ecc(u)$ of a vertex u is $dist(u, v)$ where v is the furthest vertex reachable from u in the graph. In other words, the eccentricity of a vertex is the maximum distance to that vertex. We note $Ecc(u)$ the set of vertices which are the furthest away from u (i.e the set of vertices which could be v in $ecc(u) = dist(u, v)$).

- The diameter of a graph is the maximum eccentricity which can be found in the graph. Also, the set of diametral vertices is the set of vertices whose eccentricity is the diameter of the graph.

- The radius of a graph is the minimum eccentricity which can be found in the graph. Also, the center of a graph is the set of vertices whose eccentricity is the radius of the graph.

In this project specifically:

- The identifier vid refers to a vertex id.

- The identifiers ecc and max_dist refer to the eccentricity of a node.

- The identifier ecc_vids refers to the set of nodes on the eccentricity of a vertex ($Ecc(u)$).

With these terms defined, we quickly recognize that computing the eccentricty of every single node in the graph and keeping track the maximum and minimum as well as their associated vertices would give us with certainty the diameter, radius, center and full set of diametral vertices. Howver, computing the eccentricity of a point is costly on such large graphs, making this solution unrealistic. That being said, computing the eccentricity of vertices is the basic underlying mechanism for finding good approximations.

The idea is that we don't actually need to compute every vertex's eccentricity but only the eccentricity of 2 vertices : a diametral one and a central one. Therefore, if we can find a strategy which maximizes our chances of having two such vertices, we can find both the radius and diameter relatively fast.

In order to actually compute the eccentricity of a vertex, one has to compute the distance of u to every other vertex in the graph and retain the maximum. The most straightforward way of doing so is via BFS, an algorithm which traverses every node of the graph by increasing order of distance from the root vertex (u in this case). In doing so, it also provides a

spanning tree which is a tree representing the shortest paths to every other nodes of the graph. Thus, the set $Ecc(u)$ is merely the set of vertices on the deepest level of this tree. This kind of traversal runs in $O(V + E)$ time complexity.

1.2 About approximations and their accuracy

Given that, for any vertex u in the graph: $R \leq ecc(u) \leq D \leq 2.R \leq 2.ecc(u)$, where R is the radius of the graph ($R = \min_u ecc(u)$) and D is the diameter of the graph ($D = \max_u ecc(u)$). The approximations we are looking to provide are actually what are called min_ecc and max_ecc in the implementation. Thus: $R \leq min\_ecc$ and $max\_ecc \leq D \leq 2.min\_ecc$

We are therefore trying to provide an interval for the diameter and an upper bound for the radius. The only way to get a guaranteed value for the diameter is if $max\_ecc = 2.min\_ecc$. In such a case, since $2.min\_ecc = max\_ecc \leq D \leq 2.R$, we find that $2.min\_ecc \leq 2.R$ and since $R \leq min\_ecc$, the radius value would also be guaranteed to be exact (and not merely an upper bound). Therefore, under these conditions we would have $R = min\_ecc$ and $D = max\_ecc$. However such a case is only achievable when $D = 2.R$ which is not true of every graph. Even then, we also need to find the best values for min_ecc and max_ecc in an acceptable time.

1.3 Sweeping

The classic technique to approximate the diameter is called a double sweep. The idea is to perform 2 successive BFS where the second one starts from the last vertex traversed by the first one. This double sweep technique allows us to find the eccentricity of the furthest node from the starting one in only 2 BFS. The idea of sweeping can be further expanded upon and declined in many forms (some of which are explored later in this report) as they are the basic blocks of any strategy.

When performing a sweep, the only parameter is the starting point: choosing the best starting point is the key to improve both the accuracy of the approximations and the speed at which we find them. In this regard, the purpose of the first sweep in the double sweep technique is to provide a good starting point for the second one. That being said, the starting point of the double sweep itself is still an important parameter. One might consider a

triple sweep in which the first sweep's purpose is to find a good starting point for the double sweep that follows. This pattern can be expanded indefinitely into n sweeps when the first $n-1$ sweeps are to find a good starting point for the last.

Another way to improve the starting node of the second sweep is to not merely use the last vertex of the first sweep but other nodes in the spanning tree (such as a random one among the last level of the tree for instance).

The important point to note about sweeping is that performing one can only ever improve our approximation and understanding of the graph since we are looking for maximmum and minimum eccentricities. At worst the sweep does not improve our approximations, but it cannot degrade them in any way. This is also true of the set of diametral vertices and center vertices which I will describe in more details later.

1.4 About igraph

This project is implemented in C using the igraph library. The latter is a general purpose graph processing library which provides lots of useful features for such a project, in particular a BFS implementation and useful data structures. The graphs are stored as edgelists which is particularly suited to sparse graphs. However, the library was not designed to process very large graphs and therefore does not provide any approximation feature per se.

2 Implementation and design

The usage is described in the accompanied README file. The program does the following:

1. Load the graph in memory
2. Start the chosen strategy from a randomly selected vertex
3. Print the resulting approximations

In this section, I describe mostly the second step: the strategies implemented, the underlying sweeping mechanism, the problems I encountered ...

There are 4 levels of abstraction, only the top one is exposed to the main function of the program. Each layer is designed to only be used from the one directly on top of it. The goal of this design is to simplify as much as possible the interface for a user wanting to merely use an already existing strategy or implement a new one. The layers of abstraction are:

	Layer	Description
3	strategy	self contained approximation method
2	sweeps	typical sweeps implementation
1	sweep-core	single sweep mechanism, updates approximations
0	igraph_bfs	bfs implemented by the igraph library

Figure 1: Program design and layers of abstraction

The sweep function provided by sweep-core is a wrapper around the BFS offered by igraph which takes care of updating approximations so that higher layers need not to worry about it.

The sweeps layer defines some typical sweeps and sequences of sweeps such as the classic double sweep.

The strategy layer defines some strategies. A strategy is a single function which calls a number of typical sweeps. An example of a (poor) strategy would be double sweep then double sweep again from a random vertex. At this level, there's no need to worry about keeping track of eccentricities and approximations as the sweep-core layer ensures the best approximation is kept for the performed sweeps. This layer needs only to worry about the

sweeps to perform: how many, which kinds, in what order and from which starting points.

If someone were to expand on this project to try different strategies, adding to the upper layers is easier and modifying the lower layers is typically not needed. For instance, implementing the previously described (poor) strategy of double sweeping twice can be done in a single function of 3 lines in the strategy source file.

2.1 Loading and preprocessing the graph

The first step is to load the graph in memory. This is pretty straightforward as igraph provides a function to read an edgelist. We only need to make sure we skip the degrees lines as igraph makes no use of those, and then we leave the rest of the file for igraph to read and build the graph. However, loading the graph in igraph this way takes an especially long time as can be seen in the profiler data given in the annexes (fig 4). I have not investigated this issue further nor tried to load the graph in other ways.

The next step is to preprocess the graph. The only preprocessing performed is the selection of the main cluster. Igraph provides a useful function to identify clusters, their size and the cluster of each vertex which run in $O(V + E)$ time (and probably makes use of the BFS). Using this function, I originally filled a vector with all vertices to remove and made use of an igraph function to remove all vertices of a vector from a graph but profiling data showed that the preprocessing took a very long time and slowed the whole program considerably. I then realized that the removal of vertices from the graph is performed in $O(V + E)$ time complexity. There is no other way to remove vertices from the graph offered by the library. Therefore, instead of removing the unwanted vertices I decided to build a vector of vertices in the main cluster and pick the starting point from this vector. Since the rest of the program uses only vertices found in the spanning tree of BFS, only the main cluster would be considered. If a strategy requires more than just a starting point, it can easily be passed this vector for extra nodes to choose from (though none of the strategies proposed here needed that).

Since deleting vertices from a graph is so costly, I also gave up on another preprocessing idea where nodes with exactly one neighbour would be deleted if there existed at least one other node with exactly one neighbour at exactly distance 2 of the first one. The idea was to keep only at most 1 leaf attached to any node in the graph and thus speed up later sweeps by removing useless

nodes.

2.2 The graph_data structure

```
struct graph_data {
  igraph_integer_t min_ecc;
  igraph_integer_t max_ecc;
  igraph_vector_t max_ecc_vids;
  igraph_vector_t diametral_candidates;
  igraph_vector_t center_vertices;
};
```

This structure is meant to hold the persisting data between sweeps. In other words, this is where the results are stored and updated regularly as the program runs.

- **min_ecc:** holds the lowest eccentricity found across all performed sweeps
- **max_ecc:** holds the highest eccentricity found across all performed sweeps
- **max_ecc_vids**: holds the set of vertices which have been sweeped from and found to be of the same eccentricity as max_ecc
- **diametral_candidates:** holds a vector of vertices which are good candidates for being diametral (more details below)
- **center_vertices:** holds the set of vertices which have been sweeped from and found to be of the same eccentricity as min_ecc

The first 2 fields store approximations for the radius and diameter.

The max_ecc_vids vector contains all known vertices of eccentricity max_ecc. Its main purpose is to avoid adding them to the diametral candidates vector (which would be a waste of time since we have already performed a sweep from each of the vertices in max_ecc_vids).

The diametral_candidates vertices contains an unordered list of vertices which are found to be of eccentricity greater or equal to max_ecc. We know this because these have been found to be on the last level of the spanning tree for which the root is one of the nodes in max_ecc_vids. In other words, this vector contains vertices which are known to be at distance max_ecc from one of the vertices in max_ecc_vids, therefore their eccentricity is at least max_ecc

and they are good candidates for being diametral nodes. For a performance reason explained in the following subsection, the vector is not sorted and a given vertex may appear multiple times. However, a function to sort and remove multiples from a vector is provided with this project and called on this before printing the results.

The center_vertices vector contains all known central vertices assuming that min_ecc is the radius. It is not meant to be exhaustive but if min_ecc is equal to the radius, the exhaustive set of central vertices can be easily found with a custom BFS which only enqueues neughbours if the eccentricity of the current node is the found radius. This was not implemented in this project as it is not the main purpose of it and it would have slowed down considerably the program. It should be noted that some strategies naturally fill this vector more than others. Also, one could easily write a strategy which builds up a list of center candidates without taking the time to check their eccentricity using the center sweep described later.

2.3 The sweep-core layer

The sweep function is a wrapper around the igraph_bfs function which takes care of updating the contents of the graph_data structure. It returns the eccentricity of the provided "start" vertex. Optionnal pointer arguments of the sweep function allows the user to get more informations about the sweep if they need it. Those optionnal informations are:

- **out_end:** the last vertex traversed by the bfs
- **ecc_vids:** a vector which is filled with the last level of the spanning tree (i.e $Ecc(start)$)
- **parent:** a vector which represents the spanning tree of the bfs in the form of an array of parent. For any node u that was traversed by the bfs, parent[u] contains the vid of the parent of u in the spanning tree.

The igraph bfs allows us to provide vectors to fill with the bfs data such as a distance vector in which the distance for each node is written. However, using this feature would cost a lot of memory ($O(V)$) when we only need to know the maximum distance. The igraph bfs also allows us to pass a pointer to a function which is called on every node as they are traversed so we use that instead to better control the memory usage. The callback function has

access to all available data on the specific node it is called for. This includes the distance of the node as well as the id of the next node to be traversed. The latter is set to -1 for the last node of the bfs, therefore we are able to store the id of the last node in constant memory space using the handler.

We also keep track of the nodes which are at the maximum distance (if the user asked for them by providing a vector for ecc_vids). To do so, we simply add every vertex to the vector and clear it each time the distance increases. Finally we do a similar processing for diametral_candidates but only if the distance is at least max_ecc. Also, we avoid adding vertices which are already in max_ecc_vids as diametral candidates since those are vertices we already have sweeped from.

Once the bfs is finished, we know the eccentricity so we can update min_ecc, max_ecc, max_ecc_vids and center_vertices. We also remove the start node from the diametral candidates since it has now been sweeped from.

Since the handler is called for every node, an important goal of the design was to make sure that this function runs in constant time complexity. The vector calls to push an element at the back is amortized constant, typically reallocation is done only when spanning the first few levels of the tree. There is a binary search but it is performed on the max_ecc_vids vector which is bounded by the number of calls to sweep since we only ever add anything to this vector in the sweep function, at most once per call. There is a constant number of sweeps in every strategy (10 at most), therefore this binary search is negligible compared to V. However, the diametral candidates vector can grow much more depending on the graph: keeping it sorted and unique (meaning that no value appears more than once in the vector) in the handler would require a binary search and an insertion at a specific inside position each time, which is $O(S \log(S))$ where S is the size of the vector (which is not necessarily negligible compared to V). Keep in mind that everything inside the handler is called V times for a single sweep. Therefore, keeping the diametral candidates sorted and unique is not viable to do in the callback. I provide a function to sort and unique a vector which runs in $O(S \log(S))$. It would be a better idea to call this function on the diameter candidates in the sweep function (after the bfs), however I chose to not do that either since I don't think it matters much that this vector contains multiples, and it would risk making the sweep $O((V + E) \log(V))$ when S is comparable to V. A strategy or sweep sequence which requires candidates to be sorted and unique could easily call the provided function itself in between sweeps and

reach the same result.

In conclusion, this layer of abstraction takes care of updating the approximation data so that the more interesting parts (namely the strategies) can just sweep without keeping track of these values themselves.

2.4 Sweeps

There are a lot of recurring sweep patterns between different strategies. In order to avoid code replication, but also name, identify and analyze the efficency of these patterns, they are implemented in this layer. Strategies should not call the core sweep function directly, they should use these sweeps instead.

- **single sweep:** a mere wrapper around sweep which simplifies the interface
- **double sweep (simple):** the classic double sweep as described in the course. The second sweep starts from the first sweep's last visited vertex.
- **double sweep (random):** a double sweep where the second sweep starts from a randomly selected vertex from the ecc_vid vector (i.e from the set of all vertices which are the furthest from the start)
- **center sweep:** a single sweep which finds and returns the vid of the center of the spanning tree (by going "up" the branch of the last visited vertex half of its distance)
- **radius-center sweep:** a double sweep where the second sweep is a center sweep. The first one is called "radius" as it is meant to be started from a central node and thus improve the radius (min_ecc) approximation. After the second sweep, the newly found center node is returned. The idea is to start from the center and find another central node while performing a double sweep.
- **center-radius sweep:** a double sweep meant to improve the min_ecc (radius) approximation. It should be started from a diametral candidate. The second sweep starts from the center found by the first one (hopefully a central vertex). This is a rather classic pattern used in the most straightforward strategy to approximate the radius.

- **multiple sweeps:** a natural extension of the simple double sweep where an arbitrary number of sweeps is performed instead of just 2 (each sweep starts from the previous' last visited vertex).

2.5 Strategies

I propose 3 strategies in this report, all of which are implemented and tested on the inet, ip and p2p graphs (I don't have enough ram to test the web graph).

- **Classic:** It performs 7 sweeps: the first 3 are meant to find good diametral candidates and maximum eccentricity (i.e the lower bound for the diameter). They are performed as a multiple sweep (with $n = 3$). The last 4 sweeps are 2 consecutives center-radius sweeps meant to find a good approximation for the minimum eccentricity (i.e radius and upper bound for the diameter) and 1 or 2 center nodes. The intended behaviour of this strategy is to find the diameter, then use the results to find the radius while improving the diameter (mostly its upper bound). The layout is:

  1. **1 x triple sweep:** to approximate the maximum eccentricity and find diametral candidates
  2. **2 x center-radius sweeps:** from a diametral candidate, to approximate the radius

- **Center Chain:** It performs 8 sweeps. The idea behind it is that double sweeping from the center leads to better approximations and may avoid the shuriken problem described in the course. The core part of it therefore consists of chaining a radius-center sweep into another. This "chain" could theoretically be expanded as much as needed, but for the graphs I tested, 2 radius-center sweeps were enough. The layout is as follows:

  1. **1 x double sweep:** to get some good diametral candidates
  2. **1 x center sweep:** from a diametral candidate, to find a first center to start with
  3. **2 x radius-center sweeps:** the core of this strategy, each of them starts from the last vertex returned by their preceding sweep (the

first's return value is "chained" as the start vertex of the second one)

4. **1 x single sweep:** to check the last center's eccentricity (essentially a "radius" sweep)

As you can see, the first 3 sweeps are only for preparing the chain. The point of the last sweep is to check the last central candidate returned. One could argue that it is not needed but I believe that this can save the inaccuracies brought by a poor start node for the strategy (basically, I conjecture that the diameter settles faster than the radius because we use double sweeps and therefore we should finish with a radius sweep to ensure that the radius approximation is "as good as" the diameter one).

- **Balanced:** It performs 10 sweeps. It is based on the classic strategy. I wanted to see how adding more sweeps to the original strategy would improve the results. I called it "balanced" since this strategy stemmed from the observation that out of the 7 sweeps in the classic strategy, only 2 are "radius" sweeps. The goal was to increase the overall number of sweeps and adjust the ratio of "radius" sweeps to see if we could maintain the same max_ecc accuracy while improving the min_ecc accuracy. The structure of this strategy is identical to the first one: the first part has been reduced from a triple sweep to a double sweep while the second part has been doubled from 2 center-radius sweeps to 4. In total, 4 out of the 10 sweeps are "radius" sweeps (closer to a half than the classic strategy was). The first part is reduced to a double sweep since for every "radius" sweep in the second part, there is a center sweep starting from a diametral candidates (and thus improving the max_ecc).

  1. **1 x double sweep:** initial double sweep, finds good initial diametral candidates
  2. **4 x center-radius sweeps**: each should improve both the diameter and radius

3 Results

In this section we compare the results and times of running the different strategies on 3 of the graphs available at http://data.complexnetworks.fr/Diameter/. Only the relevant parts of the logs are displayed here, you may finf the full output logs at the end of this report (in the annexes). Note that all the tests in this report were run sequentially on the same machine under the same conditions.

	Classic	**Center Chain**	**Balanced**
inet	Diameter: 31 ~ 34 Radius : <= 17 Center : 208406	Diameter: 31 ~ 32 Radius : <= 16 Center : 165547	Diameter: 31 ~ 34 Radius : <= 17 Center : 208406
ip	Diameter: 9 ~ 10 Radius : <= 5 Center : 8	Diameter: 9 ~ 10 Radius : <= 5 Center : 8	Diameter: 9 ~ 10 Radius : <= 5 Center : 8, 102
p2p	Diameter: 9 ~ 12 Radius : <= 6 Center : 1432588	Diameter: 9 ~ 12 Radius : <= 6 Center : 2306314, 2345071, 4803962	Diameter: 9 ~ 12 Radius : <= 6 Center : 18632, 346974, 1432588

Figure 2: Strategies comparison table - Results

We notice from figure 2 that all 3 stategies give similar approximations most of the time (at least on those 3 graphs). On the inet graph, both the classic and balanced strategies are beaten by the center chain which gives a much better approximation for min_ecc. The classic and balanced strategies actually give the exact same results on this instance, therefore the extra sweeps in balanced don't seem to matter here, while the double sweeps started from the center in the second strategy seem to work especially well on this graph.

Other than that, there is barely any noticeable difference between the 3 strategies' results. The classic strategy tends to give less central vertices than the other other 2 strategies which makes perfect sense since only 2 of its sweeps could theoretically add a node to this set. Actually the number of central vertices seems to be somewhat proportionnal the number of "radius" sweeps in the strategy (4 in balanced, 3 in center chain, 2 in classic). While I stated earlier that finding the exhaustive set of central vertices is not among the main goals of this project, finding more is better and is also an indicator of the confidence for the radius approximation. The more central vertices are given, the more radius sweeps have confirmed the radius approximation. A similar case could be made for diametral vertices and candidates (shown in the full output logs only), however (in both cases) this idea needs further investigation as a low number of candidates or a high number of "confirmed" min/max eccentricity vertices could also mean that the strategy is fundamentally broken and didn't even get close to the vertices it needed in order to find the actual diameter and radius. On diametral vertices again, the results are pretty similar between strategies. The only noticeable exception is that center chain gives a very low number of diametral candidates on p2p compared to the other strategies. I believe this is because a higher maximum eccentricity is found after the third sweep: no sweep starts from a diametral candidate after the third one (under the center chain strategy). Does this mean that the few candidates are more likely to actually be diametral than the many found by other strategies ? I guess yes only if the approximation is accurate. This implies one of two things: either this strategy has a significant risk of completely missing the best approximation for the maximum eccentricity or it has a tendency to find better diametral candidates than the others.

As expected, the time it takes to run the program is proportionnal to the number of sweeps performed and to the size of the graph (shown by figure 3). However, the impact of the extra sweeps in center chain and balanced is only really significant in bigger graphs such as p2p.

	Classic	**Center Chain**	**Balanced**
inet	real 0m28.291s user 0m27.314s sys 0m0.901s	real 0m30.656s user 0m29.594s sys 0m0.985s	real 0m35.516s user 0m34.297s sys 0m1.121s
ip	real 0m31.302s user 0m28.819s sys 0m2.418s	real 0m33.457s user 0m30.878s sys 0m2.508s	real 0m37.650s user 0m34.663s sys 0m2.900s
p2p	real 5m54.186s user 5m34.044s sys 0m19.129s	real 6m17.783s user 5m55.815s sys 0m20.875s	real 7m0.827s user 6m34.282s sys 0m25.309s

Figure 3: Strategies comparison table - Times

Therefore, we have found that:

- The classic strategy is fast and gives decent results in most cases
- The center chain strategy gives noticeably better results with only 1 extra sweep (so still rather fast)
- The balanced strategy shows that adding or balancing sweeps to the classic strategy is inefficient: the results are the same with maybe a bit more confidence yet the time it takes is considerably higher

It seems that finding good starting points for sweeping leads to better results than merely sweeping more. Therefore a good strategy focuses on setting up a few good sweeps rather than repeating a sweeping pattern randomly. Finally, it seems that finding the minimum eccentricity is harder that finding the maximum eccentricity.

4 Improvements and ideas

In this section, I make a list of ideas to explore and improvements which could be made to this project.

1. **Different start nodes** Currently, each of the strategies are started from a vertex randomly selected in the main cluster. Investigating the option to start from a specific node in the main cluster such as the node with the highest or lowest degree (number of neighbours) could lead to promising results (since it seems that starting points matter a lot). It was originally a planned feature of this project (to be implemented in pick-point.c) but I was late on my schedule so I dropped it.

2. **Convergent strategies** The strategies proposed in this project have a fixed number of sweeps. An idea to investigate is that of strategies which keep sweeping until the approximations converge (or a maximum number of iterations is reached). In light of the results of this report, this could prove inefficient, at the very least hard to implement (since there are often a lot of candidates and only a few make changes to the approximations). However, it could also prove to be more reliable than the strategies proposed here.

3. **Interactive mode** Rather than trying to find the best strategy, another approach we could take is human supervision. An interactive mode could ask a user which kind of sweep to perform from which starting point at every step. The user would stop once satisfied with the approximations. The idea of a strategy would be completely replaced with the user's decisions. While this would take more time overall, it could be a good way to ensure the provided result complies with the user's requirements. Also, since a sweep can only every improve the approximations, the user cannot make decisions with a strictly negative impact on the results (which I find to be an intersting property).

4. **Turning towards traditionnal AI** One could think of training a model such as a neural network to use the aforementionned interactive mode to reach an ideal results/time ratio. This is a more far-fetched idea, but replacing a strategy made of a static set of instruction with a supervised learning AI model could potentially lead to excellent results. Although it would probably not improve our understanding of very large graphs.

5. **Parallel processing** Performing sweeps in parallel would be easy: simply make a copy of the graph_data structure per thread, start a sweep on each thread, then merge back each of the graph_data structure to the main thread (keep fields depending on the best minimum/maximum eccentricities found among all threads). However, performing sweeps in parallel might not prove very useful in practice: it would allow us to do more sweeps in the same amount of time, but the results seem to indicate that setting up a few good sweeps leads to better results than just repeating the same sweeps more times. Setting up good sweeps is kind of an inherently sequential process since it requires sweeping to find good diametral/central candidates to sweep from later on. That being said, even if the improvement in results is small, this is still an improvement to be considered since it would essentially have no cost in time or memory (the graph is stored once, sweeps are fully parallel; the only cost is setting up threads and copying/merging the approximations data).

6. **Meta-strategies** These would take a considerably longer time to get a more reliable result. They would start by running a first strategy on the graph, and depending on the results would run another strategy to finally merge the approximations. This is highly hypothetical but the idea would be that combining strategies is probably more reliable against any graph, and investigating relations between a strategy's results and another's effectiveness could maybe lead to relatively long but consistent solutions. For example, if 3 strategies give completely different central vertices, how safe is it to assume that the radius approximation is poor and run a more aggressive, radius oriented strategy? (probably not very safe, it's a poor example)

5 Annexes

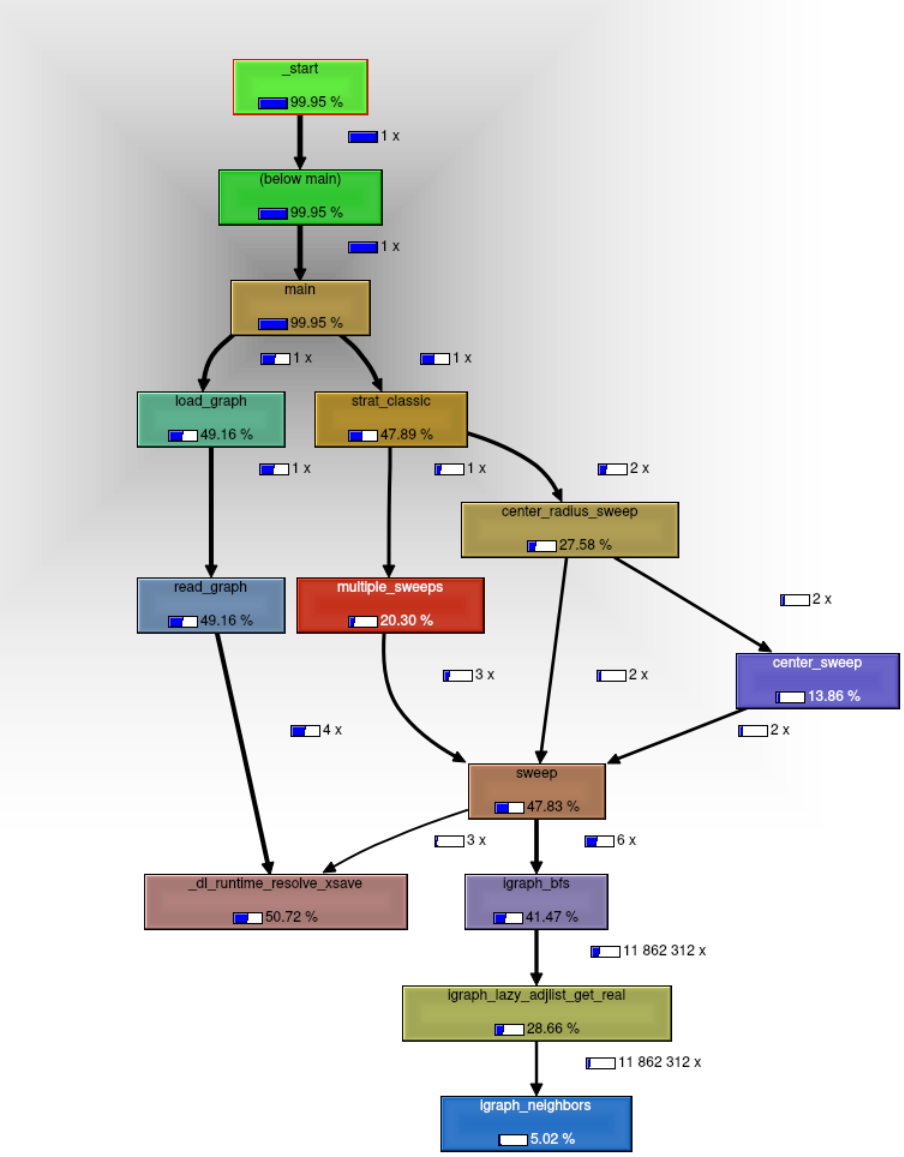

Figure 4: Profile for the classic strategy on the inet graph

5.1 Outputs

```
sh$ time ./vlg classic data/inet

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  multiple sweeps (3) from 1222967: OK
  center_radius sweep from 1511903: OK
  center_radius sweep from 1717718: OK
OK

Diameter: 31 ~ 34
Radius  : <= 17
Center  : 208406

Diametral vertices:
  Known maximum eccentricity vertices    : 1511903, 1717718
  Number of diametral vertices candidates: 0

real  0m28.291s
user  0m27.314s
sys   0m0.901s
```

```
sh$ time ./vlg center_chain data/inet

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  initial double sweep (random) from 1222967: OK
  center sweep from 1511903: OK
  radius_center (double) sweep from 208406: OK
  radius_center (double) sweep from 165547: OK
  last (single) sweep for the radius from 208405: OK
OK

Diameter: 31 ~ 32
Radius  : <= 16
Center  : 165547

Diametral vertices:
  Known maximum eccentricity vertices    : 1511903
  Number of diametral vertices candidates: 1

real  0m30.656s
```

```
user   0m29.594s
sys    0m0.985s
```

```
sh$ time ./vlg balanced data/inet

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  double sweep (simple) from 1222967: OK
  center_radius sweep from 1511903: OK
  center_radius sweep from 1717718: OK
  center_radius sweep from 1717718: OK
  center_radius sweep from 1717718: OK
OK

Diameter: 31 ~ 34
Radius  : <= 17
Center  : 208406

Diametral vertices:
  Known maximum eccentricity vertices    : 1511903, 1717718
  Number of diametral vertices candidates: 0

real   0m35.516s
user   0m34.297s
sys    0m1.121s
```

```
sh$ time ./vlg classic data/ip

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  multiple sweeps (3) from 2002926: OK
  center_radius sweep from 175534: OK
  center_radius sweep from 21673: OK
OK

Diameter: 9 ~ 10
Radius  : <= 5
Center  : 8

Diametral vertices:
  Known maximum eccentricity vertices    : 21673, 175534,
      772689, 2093252
  Number of diametral vertices candidates: 256
```

```
real  0m31.302s
user  0m28.819s
sys   0m2.418s
```

```
sh$ time ./vlg center_chain data/ip

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  initial double sweep (random) from 2002926: OK
  center sweep from 65818: OK
  radius_center (double) sweep from 8: OK
  radius_center (double) sweep from 8: OK
  last (single) sweep for the radius from 770398: OK
OK

Diameter: 9 ~ 10
Radius  : <= 5
Center  : 8

Diametral vertices:
  Known maximum eccentricity vertices    : 65818, 154340, 772044
  Number of diametral vertices candidates: 257

real  0m33.457s
user  0m30.878s
sys   0m2.508s
```

```
sh$ time ./vlg balanced data/ip

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  double sweep (simple) from 2002926: OK
  center_radius sweep from 175950: OK
  center_radius sweep from 26148: OK
  center_radius sweep from 771060: OK
  center_radius sweep from 83299: OK
OK

Diameter: 9 ~ 10
Radius  : <= 5
Center  : 8, 102
```

```
Diametral vertices:
  Known maximum eccentricity vertices    : 26148, 83299, 175950,
      771060, 772689
  Number of diametral vertices candidates: 255

real  0m37.650s
user  0m34.663s
sys   0m2.900s
```

```
sh$ time ./vlg classic data/p2p

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  multiple sweeps (3) from 1888114: OK
  center_radius sweep from 2641627: OK
  center_radius sweep from 4006715: OK
OK

Diameter: 9 ~ 12
Radius  : <= 6
Center  : 1432588

Diametral vertices:
  Known maximum eccentricity vertices    : 2034294, 2641627,
      4006715, 5664968
  Number of diametral vertices candidates: 22

real  5m54.186s
user  5m34.044s
sys   0m19.129s
```

```
sh$ time ./vlg center_chain data/p2p

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  initial double sweep (random) from 1888114: OK
  center sweep from 2060154: OK
  radius_center (double) sweep from 4803962: OK
  radius_center (double) sweep from 2345071: OK
  last (single) sweep for the radius from 2306314: OK
OK

Diameter: 9 ~ 12
```

```
Radius   : <= 6
Center   : 2306314, 2345071, 4803962

Diametral vertices:
  Known maximum eccentricity vertices     : 2034294, 2060154
  Number of diametral vertices candidates: 2

real   6m17.783s
user   5m55.815s
sys    0m20.875s
```

```
sh$ time ./vlg balanced data/p2p

Loading graph: OK
Finding the main cluster: OK
Sweeps:
  double sweep (simple) from 1888114: OK
  center_radius sweep from 398586: OK
  center_radius sweep from 135496: OK
  center_radius sweep from 4006715: OK
  center_radius sweep from 5505958: OK
OK

Diameter: 9 ~ 12
Radius   : <= 6
Center   : 18632, 346974, 1432588

Diametral vertices:
  Known maximum eccentricity vertices     : 135496, 398586,
      4006715, 5505958, 5664968
  Number of diametral vertices candidates: 21

real   7m0.827s
user   6m34.282s
sys    0m25.309s
```

List of Figures

Contents