Computing the diameter

The diameter of a network is a relevant measure (whose meaning depends on the semantics of the network itself), which has been almost always considered while analyzing real-world networks such as biological, collaboration, communication, road, social, and web networks. Informally, the diameter is the maximum distance between two vertices (in this chapter we will always refer to connected, unweighted, and undirected graphs). Many algorithmic results have been presented in the last few decades concerning the computation of the diameter. In the general case, the best known solution is still, more or less, based on the computation of all-pairs shortest paths. The time complexity of this solution is $O(n^{\omega})$, where $\omega < 2.38$, in the case of dense graphs (by using efficient algorithms for matrix multiplication), and $O(mn)$ in the case of sparse graphs (by simply performing a breadth-first search from each node of the graph). This complexity is not feasible whenever we deal with real-world networks, since these networks may contain several millions of nodes and several billions of edges.

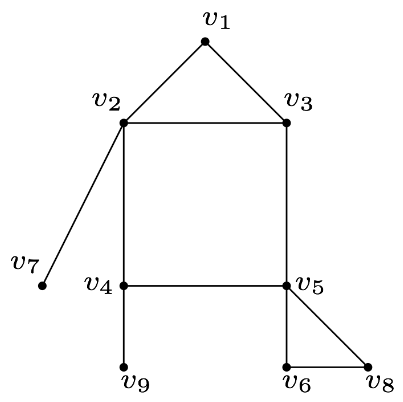

Figure 33: The diameter of this graph is 4. Indeed, the maximum distance between two nodes of the graph is the distance between v_7 and v_6 (which is equal to the distance from v_7 to v_8).

Lower bounds on diameter computation time

Since the very beginning of theoretical computer science and until recent years, the duality between NP-hard problems and polynomial-time solvable problems has been considered the threshold distinguishing "easy" from "hard" problems. However, polynomial-time algorithms might not be as efficient as one expects: for instance, in real-world networks with millions or billions of nodes, also quadratic-time algorithms might turn out to be too slow in practice, and a *truly sub-quadratic* algorithm would be a significant improvement, where an algorithm is said to be truly sub-quadratic if its time-complexity is $O(n^{2-\epsilon})$ for some $\epsilon > 0$, where n is the input size. Following the main ideas behind the theory of NP-completeness, and not being able to show that a specific polynomial-time solvable problem might or might not admit a faster algorithm, researchers have recently started to prove that the existence of such an algorithm would imply faster solutions for other well-known and widely studied problems. As an example, a great amount of work started from the analysis of the Strong Exponential Time Hypothesis (in short, *SETH*),[4] which has been used as a tool to prove the hardness of polynomial-time solvable problems. This hypothesis says that there is no algorithm for solving the k-Sat problem in time $O((2-\epsilon)^n)$, where $\epsilon > 0$ does not depend on k. SETH has become a starting point for proving the "hardness" of many other problems, especially in the field of graph analysis. Note that all these results do not deal with the notion of completeness, but they simply prove that "a problem is harder than another", using a kind of reduction between problems, relying on the fact that the easiest problem has been studied for years and no efficient algorithm has been found.

[4] R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001

In general, a (combinatorial) problem A is a function which associates to any string $x \in I_A$, where I_A is the the set of *instances* of the problem, a set of strings $A(x)$ (each string in $A(x)$ is called a *solution* of x). If, for any $x \in I_A$, $A(x) \in \{\{0\},\{1\}\}$, then the problem A is said to be a decision problem. Given two problems A and B, we say that A is reducible to B is there exist two functions f and g such that, for any $x \in I_A$, $x' = f(x) \in I_B$, and, for any $y' \in B(x')$, $y = g(x, y') \in A(x)$. In the theory of NP-completeness, the two functions f and g are usually required to be computable in polynomial time. However, since our goal is to analyze the complexity of problems which are solvable in polynomial time, in the following we will require that f and g are computable in *linear* time. Indeed, if this is the case and if A is reducible to B, then a sub-quadratic algorithm for B would imply a sub-quadratic algorithm for A.

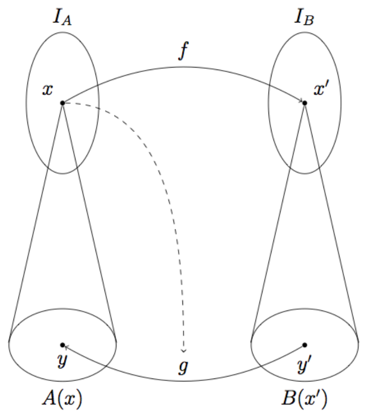

Figure 34: A problem A is reducible to a problem B if we can map any instance of A to an instance of B, and if we can recover a solution for A starting from a solution from B.

From SETH to quadratic SAT

Let us consider an "artificial" variation k-Sat* of k-Sat which is quadratic-time solvable, but not solvable in time $(n^{2-\epsilon})$ unless SETH is false. In this variation, an instance is formed by two sets of variables $X = \{x_i\}$ and $Y = \{y_j\}$ of the same size, a set C of clauses over these variables, such that each clause has at most size k, and the two power sets $\mathcal{P}(X)$ and $\mathcal{P}(Y)$ (which are used to change the input size). The solution to an instance of k-Sat* is 1 if there is an evaluation of all variables that satisfies all clauses, 0 otherwise. This problem differs from the classic one only by the input size. In this way, a quadratic-time algorithm exists for k-Sat*, which simply tries all possible assignments of the variables. However, if m is the number of variables and $n = 2^{\frac{m}{2}}$ is the input size, since both X and Y have size $\frac{m}{2}$, an algorithm running in time $(n^{2-\epsilon})$ with ϵ not depending on k would imply an algorithm solving k-Sat in time

$(2^{\frac{m}{2}(2-\epsilon)}) = \left(\left(2^{\frac{2-\epsilon}{2}}\right)^m\right)$, where m is the number of variables. This latter result contradicts SETH.

From quadratic SAT to disjoint sets

The Two Disjoint Set (in short, *TDS*) problem is defined as follows. Given a set X and a collection $\mathcal{C}$ of subsets of X, the solution is 1 if there are two disjoint sets $C, C' \in \mathcal{C}$, 0 otherwise. Clearly, this problem can be solved in quadratic time. Let us now show that k-SAT* is reducible (in linear time) to TDS, thus implying that this latter problem is not solvable in sub-quadratic time, unless SETH is false.

Let X_1 be the set of the possible evaluations of $\{x_i\}$, X_2 the set of possible evaluations of $\{y_j\}$, C the set of clauses of an instance I of k-SAT*. We define $f(I) = (X, \mathcal{C})$, where $X = C \cup \{t_1, t_2\}$, and $\mathcal{C}$ is the collection made by sets of clauses not satisfied by an assignment in X_1 or X_2 (and t_1, t_2 are used to distinguish between assignments in X_1 and X_2). More formally, $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$, where

$$\mathcal{C}_1 := \{\{t_1\} \cup \{c \in C : x \text{ does not satisfy } c\} : x \in X_1\},$$

and

$$\mathcal{C}_2 := \{\{t_2\} \cup \{c \in C : x \text{ does not satisfy } c\} : x \in X_2\}.$$

This way, the size of $f(I)$ is linear in the size of I. Moreover, it is possible to compute f by analyzing all sets of evaluation of variables one by one, and for each of them check which clauses are verified. For each evaluation in X_1 or X_2, the checking time is proportional to the number of clauses, which is at most $(\log^k(n))$, where n is the input size of k-SAT*. It remains to prove that the output of the problem is preserved: we will prove that there is a bijection g between the pairs of disjoint sets and the satisfying assignments of the formula of k-SAT*. In particular, two sets that are both in $\mathcal{C}_1$ or both in $\mathcal{C}_2$ cannot be disjoint because of t_1 and t_2. As a consequence, two disjoint sets correspond to an evaluation of all variables: the evaluation satisfies ϕ if and only if for each clause there is a variable contained in the evaluation if and only if there is no clause contained in both sets.

From disjoint sets to diameter computation

Finally, we show that TDS is reducible (in linear time) to the problem of computing the diameter of a graph. Hence, this latter problem is not solvable in sub-quadratic time unless SETH is false.

Given an input $I = (X, \mathcal{C})$ of TDS, construct a graph $G = (X \cup \mathcal{C}, E)$, where each pair in X is connected, and for each set $C \in \mathcal{C}$ we add an edge from C to its elements. Since each vertex is at distance

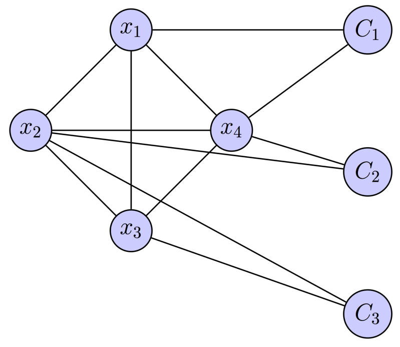

Figure 35: The reduction from disjoint sets to diameter computation. In this case, $C_1 = \{x_1, x_2\}$, $C_2 = \{x_2, x_4\}$, and $C_3 = \{x_3, x_4\}$. The diameter is 3 since C_1 and C_3 are disjoint.

1 from X, the diameter is 2 or 3: it is 3 if and only if there exist two different vertices $C, C' \in \mathcal{C}$ with no common neighbor. It is clear that this happens if and only if C and C' are disjoint.

Problem of the chapter. The reduction from TDS to diameter computation is quadratic in the number of elements in X. How can we make it linear?

Heuristics for computing the diameter

In this section (and in the next one) we will explore the power of breadth-first search in order to exactly compute the diameter of a graph. Clearly, the height of any BFS tree provides us with both a lower bound on the diameter. Thus, a simple heuristic to estimate these values consists of executing a fixed number of random BFSs, and reporting the best bound found for each of them: unfortunately, no useful bound on the performed error can be provided and even experimentally this heuristic turns out to be not always precise. For this reason, we have to deal with the problem of appropriately choosing the vertices from which the BFSs have to be performed. For example, the so-called 2-sweep heuristic picks one of the farthest vertices x from a random vertex r and returns the distance of the farthest vertex from x, while the 4-sweep heuristic picks the vertex in the middle of the longest path computed by a 2-sweep execution and performs another 2-sweep from that vertex. Both methods work quite well and very often provide tight bounds. Indeed, in the case of special classes of graphs, they can even be (almost) exact: for example, the 2-sweep method gives the exact value of the diameter for trees, yields an approximation with additive error 1 for chordal graphs and interval graphs, and within 2 for AT-free graphs and hole-free graphs. Adaptations of these methods to directed graphs have also been proposed, and, even in this case, these techniques are very efficient and provide very good bounds on real-world networks.

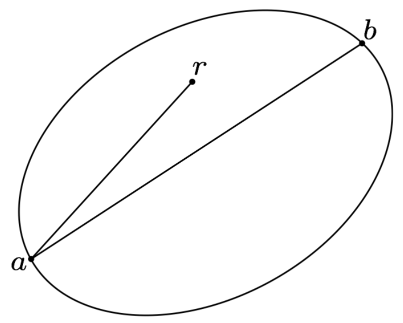

Figure 36: The 2-sweep heuristics picks one of the farthest vertices a from a random vertex r and returns the distance of the farthest vertex b from a.

Just to give an example of the efficacy of the 2-sweep heuristics, by choosing r as the highest degree node, the following table shows some experimental results obtained in the case of 80 graphs arising in different application areas.

Category	# of Networks	2-sweep	
		# of Networks in which 2-sweep is tight	Maximum error
Protein-Protein Interaction	14	11	1
Collaboration	14	12	1
Undirected Social	4	4	0
Undirected Communication	36	34	2
Autonomous System	2	1	1
Road	3	1	14
Word Adjacency	7	4	1

Exactly computing the diameter

In general, heuristics cannot guarantee the correctness of the results obtained. For this reason, a major further step in the diameter computation is the design of bound-refinement algorithms. These methods apply a heuristic and try to validate the result found or improve it until they successfully validate it. Even if in the worst case their time complexity is $O(mn)$, they turn out to be linear in practice.

The general schema of these methods is the following. They perform the breadth-first searches one after the other according to a specific order of the nodes. While doing this, they

- refine a diameter lower bound (that is the maximum eccentricity found until that moment);
- they upper bound the eccentricities of the remaining nodes to be analyzed;
- they stop whenever the remaining nodes cannot have eccentricity higher than the lower bound.

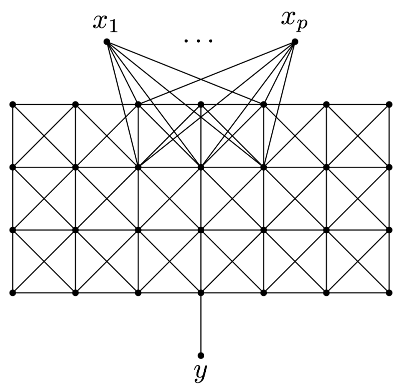

Figure 37: With input this grid with k rows and $1 + 3k/2$ columns, the 2-sweep heuristics can return k, while the diameter of the graph is instead $3k/2$.

Clearly, the order of the nodes which is used by these methods is a very important factor. A good order can be obtained by analyzing how nodes are placed in the breadth-first search tree of a "starting" node u. In order to define this order, let us introduce some notations. Let u be any node in V and let us denote the set $\{v \mid d(u,v) = \mathrm{ecc}(u)\}$ of nodes at maximum distance $\mathrm{ecc}(u)$ from u as $F(u)$. Let $F_i(u)$ be the *fringe* set of nodes at distance i from u (note that $F(u) = F_{\mathrm{ecc}(u)}(u)$). Let $B_i(u) = \max_{z \in F_i(u)} \mathrm{ecc}(z)$ be the maximum eccentricity among these nodes.

We can now prove the following statement: *for any* $1 \leq i < \mathrm{ecc}(u)$ *and* $1 \leq k < i$, *and for any* $x \in F_{i-k}(u)$ *such that* $\mathrm{ecc}(x) > 2(i-1)$, *there exists* $y \in F_j(u)$ *such that* $d(x,y) = \mathrm{ecc}(x)$ *with* $j \geq i$. To this aim, let us first observe that, for any x and y in V such that $x \in F_i(u)$ or

$y \in F_i(u)$, we have that $d(x,y) \leq B_i(u)$: this is due to the fact that $d(x,y) \leq \min\{\mathrm{ecc}(x), \mathrm{ecc}(y)\} \leq B_i(u)$. Let us also observe that, for any $1 \leq i,j \leq \mathrm{ecc}(u)$ and for any $x \in F_i(u)$ and $y \in F_j(u)$, we have $d(x,y) \leq i+j \leq 2\max\{i,j\}$. We are now ready to prove the statement. Since $\mathrm{ecc}(x) > 2(i-1)$, then there exists y_x whose distance from x is equal to $\mathrm{ecc}(x)$ and, hence, greater than $2(i-1)$. If y_x was in $F_j(u)$ with $j < i$, then from the second observation it would follow that

$$d(x,y_x) \leq 2\max\{i-k,j\} \leq 2\max\{i-k,i-1\} = 2(i-1),$$

which is a contradiction. Hence, y_x must be in $F_j(u)$ with $j \geq i$.

As a consequence of the above statement, we have that if lb is the maximum eccentricity among all the eccentricities of the nodes in or below the level i, then the eccentricities of all the nodes above the level i is bounded by $\max\{lb, 2(i-1)\}$. Indeed, for any node x above the level i, there are two cases:

- $ecc(x) \leq 2(i-1)$. Hence, $ecc(x) \leq \max\{lb, 2(i-1)\}$.
- $ecc(x) > 2(i-1)$. In this case, there is a node y below the level i whose eccentricity $ecc(y)$ is greater than or equal to $ecc(x)$. Hence, $lb \geq ecc(y) \geq ecc(x)$ and $ecc(x) \leq \max\{lb, 2(i-1)\}$.

This result suggests to perform the breadth-first searches one after the others following the order induced by the breadth-first search tree of a node u, starting from the nodes in $F(u)$, and continuing in a bottom-up fashion. In particular, at each level i, we compute the eccentricities of all its nodes: if the maximum eccentricity found is greater than $2(i-1)$ then we can avoid traversing the remaining levels, since the eccentricities of all their nodes cannot be greater than lb (since the eccentricity of the remaining nodes is bounded by $\max\{lb, 2(i-1)\}$). In other words, the algorithm for computing the diameter is the following. Given a node u:

- Set $i = \mathrm{ecc}(u)$ and $M = B_i(u)$.
- If $M > 2(i-1)$, then return M; else, set $i = i-1$ and $M = \max\{M, B_i(u)\}$, and repeat this step.

Observe that in the worst case, this algorithm has the same time-complexity of the "textbook" algorithm (that is, the one performing a breadth-first search starting from each node of the graph). A "bad" case is when all nodes have very similar eccentricities or their breadth-first search trees are very similar, like in the case of a cycle.

Observe also that the choice of the starting node heavily affects the performance of this method (that is, the number of executed breadth-first searches). Usually, two reasonable choices are a high degree

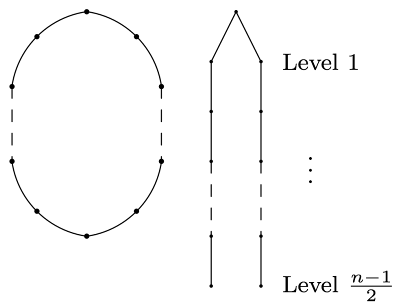

Figure 38: In the case of a cycle with n nodes (n odd), the diameter is $\frac{n-1}{2}$, and each node has the same breadth-first search tree. The loop is repeated until $2(i-1) \geq \frac{n-1}{2}$, that is $i \geq \frac{n+3}{4}$, and stops the first time that $2(i-1) < \frac{n-1}{2}$. The total number of iterations is equal to $\frac{n-1}{2} - \frac{n+3}{4} + 2 = \frac{n+3}{4}$. The total number of breadt-first searches is $\frac{n+3}{2}$.

node or the node in the middle of the path computed by the 2-sweep heuristics. In particular, choosing a node with highest degree seems to be very often a better choice. It is worth emphasizing that, by using this method, the diameter of the Facebook graph, containing more than 700 millions of nodes and more than 68 billions of edges) has been computed by performing only 17 breadth-first searches! The value of the diameter was 41...

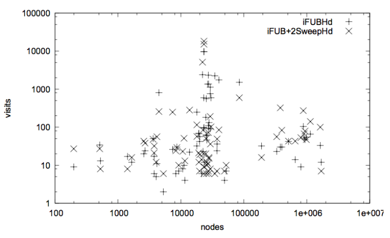

Figure 39: The number of the breadth-first searches as a function of the number of nodes.

Problem of the chapter. Choose a (undirected unweighted connected) social network with a significant number of nodes and edges. Apply the diameter algorithm described in this section. Compare the execution times with the textbook approach and when selecting different starting nodes.