Chapter 1

Fixed Parameter Tractable Algorithms

By Sariel Har-Peled, September 20, 2023[1] Version: 0.4

> "Napoleon has not been conquered by man. He was greater than all of us. But god punished him because he relied on his own intelligence alone, until that prodigious instrument was strained to breaking point. Everything breaks in the end."
>
> – – Carl XIV Johan, King of Sweden.

1.1. Introduction

Consider an optimization problem that has some parameter associated with it. In many cases, the parameter might just be the size of the solution, as in the following problem.

Problem 1.1.1 (***Steiner's tree problem***). Given an undirected graph $G = (\mathsf{V}, \mathsf{E})$, with positive weights on the edges, and a set of *terminals* $X \subseteq \mathsf{V}$ of size k. Compute the cheapest subgraph that contains X.

This problem is NP-Hard, and it thus unlikely (unless P = NPC) that it can be solved in polynomial time. Consider however the problem when $k = |X|$ is small (i.e., the parameter!). It is not hard [2] to verify that it can be solved in $n^{O(k)}$ time, which is of limited interest.

Definition 1.1.2. Consider an algorithm $A(I)$ for a problem, where I is an instance of a problem of size n, and the associated parameter of the problem is k. Then algorithm A is ***fixed parameter tractable*** (***FPT***) if the running time of A is at most $f(k)n^{O(1)}$, where $f(k)$ is some function that depends only on k.

Here, if the problem is NP-Hard, and it has an FPT algorithm, then $f(k)$ would probably to be exponential (or worse) function of k. Nevertheless, the "separation" between n and k in an FPT algorithm is quite a surprising and desirable property.

For the Steiner's tree problem, for example, we present below an FPT algorithm with running time $O(4^k n^2 + n^3)$.

1.2. Steiner tree

Consider an instance of the Steiner's tree problem. An undirected graph $G = (\mathsf{V}, \mathsf{E})$, with $n = |\mathsf{V}|$, and $X \subseteq \mathsf{V}$, with $k = |X|$.

Observe that the optimal Steiner tree for a set of terminals X is not the minimum spanning tree of X (even that requires a careful definition). Specifically, consider the instance shown in Figure 1.1.

[2] But tedious, it is always tedious.

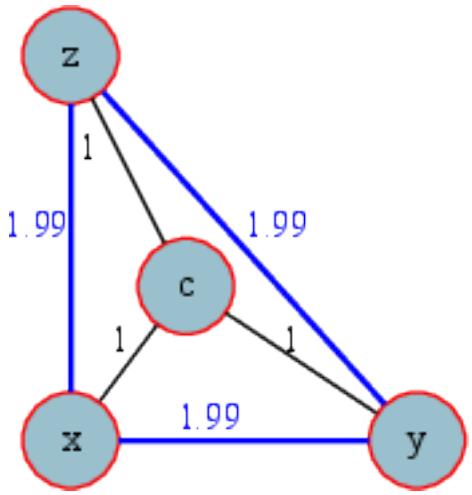

Figure 1.1: The optimal Steiner tree for the terminals $\{x, y, z\}$, is formed by their "MST", which is a path formed by their two blue edges. Rather, it is the three middle edges together with the middle vertex c.

The idea is to solve the problem recursively by specifying the subset of terminals that needs to be connected. We also want to specify how to connect it to the rest of the graph. To this end, we will specify an additional anchor vertex u that also has to be included in the Steiner tree. Specifically, for $S \subseteq X$, and a vertex $u \in \mathsf{V}$, let $f(u, S)$ be the cost of the cheapest Steiner tree connecting all the vertices of S that also includes the vertex u.

Consider the optimal solution T for $f(u, S)$. There are several possibilities:

- If $u \in S$, then we can remove it from S, since $f(u, S) = f(u, S - u)$ in this case.
- If removing u from T breaks T into two connected components, then we can recursively solve the problem on the two subtrees.
- If $|S| = 1$, then the shortest path between u and the element inside S realizes the desired solution.
- The remaining case is that u is a leaf. Let v be the internal node of T, of degree at least two where u connects to the tree. We can then use v as the breaking point, and add the shortest path from u to v (in particular, the case above that u is a breakpoint can be handled by setting $v = u$).
- There is one remaining possibility that the path from u to v includes a vertex of S. We can then set v to be this vertex on the path.

It is easy to verify that all these cases are covered by the following recurrence:

$$f(u, S) = \begin{cases} 0 & S = \emptyset \\ d_G(u, s) & S = \{s\} \\ f(u, S - u) & u \in S \\ \min_{v \in \mathsf{V}} \min_{T : T \subset S, T \neq \emptyset} \Big(d_G(v, u) + f(u, S \setminus T) + f(v, T) \Big) & \text{otherwise.} \end{cases}$$

The task at hand is to compute $\min_{v \in \mathsf{V}} f(v, X)$. This function has $2^k n$ different recursive calls. Each recursive call takes $n2^k$ time to handle.Thus, with memoization, we get an algorithm to solve the original problem in $O(4^k n^2)$ time.

Theorem 1.2.1. *The Steiner tree problem can be solved, for a set of k terminals in a graph with n nodes, in $O(4^k n^2 + n^3)$ time.*

Theorem 1.2.2. *The $O(n^3)$ term is for pre-computing all the pairwise distances in the graph. The rest is described above.*

1.3. Vertex cover

1.3.1. Greedy maximal matching

Definition 1.3.1. A set of edges $M \subseteq \mathsf{E}(\mathsf{G})$ in a graph G is a ***matching***, if no two edges of M share an endpoint. A set M is ***maximal matching*** if one can *not* add any other edge of G to M, and still get a valid matching.

Observation 1.3.2. *If M is a maximal matching, then all the other edges in the graph (that are not in M) share at least one endpoint with one of the edges of M. Indeed, if not, this offending edge can be added to the matching, contradicting its maximality.*

Given a graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, computing a maximal matching is straightforward. Mark all the vertices of G as unused, and start with an empty set M. Scan the edges of E. If currently scanning an edge e, and it has both endpoints marked as unused, then add e to M, and mark both endpoints as used, and continue the scanning process. This algorithm clearly runs in $O(|\mathsf{V}| + |\mathsf{E}|)$ time. We thus get the following.

Lemma 1.3.3. *Given an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, with n vertices and m edges, the above algorithm computes a maximal matching in G in $O(n + m)$ time.*

1.3.2. An FPT algorithm for vertex cover

Definition 1.3.4. Given a graph G, a set $X \subseteq \mathsf{V}(\mathsf{G})$ is a ***vertex cover*** if for all edges $uv \in \mathsf{E}(\mathsf{G})$, we have that at least one of the endpoints of X is in G (i.e., $|\{u, v\} \cap X| > 0$).

Problem 1.3.5. Given a graph G and a parameter k, the ***vertex cover problem*** as whether or not G has a vertex cover of size $\leq k$.

There are many different FPT algorithms for this problem. We present the arguably the simplest one. We begin with an observation.

Observation 1.3.6. *If a graph G has a matching M of size t, then any vertex cover for G must include one vertex in each edge of M. Indeed, the vertex-cover needs to cover each edge of M, and the edges of M are disjoint.*

Observation 1.3.7. *If M is a maximal matching in a graph G, and let X be the set of vertices that are not used by the edges of M. Then X is an independent set – that is, for no two vertices $x, y \in X$, the edge $xy \in \mathsf{E}(\mathsf{G})$. Indeed, if it was, then xy can be added to M to get a bigger matching, but this is a contradiction to the maximality of M.*

1.3.2.1. The algorithm

The input is the graph G, and the parameter k. The algorithm first compute a maximal matching M of G. If $|M| > k$, then by the above observation the given instance does not have a vertex cover of size $\leq k$.

Otherwise, let $U = \mathsf{V}(M)$ be the set of vertices of the matching M (i.e., $|U| = 2|M| \leq 2k$). The algorithm would guess the set $S \subseteq$ (which is supposed equal to $\mathrm{opt} \cap U$, where opt is the optimal solution). The algorithm would scan the edges of G, and enlarge it into a vertex cover of all the edges of G. To this end, let initially $T = S$, and the algorithm scan the edges of the graph one by one. When scanning the edge $e = uv$:

- If $|\{u, v\} \cap S| > 0$ then continue.
- If $\{u, v\} \subseteq U = \mathsf{V}(M)$, then S is not the correct guess of opt $\cap\, U$. Reject this guess, and continue to the next one.
- If $\{u, v\} \subseteq U = \mathsf{V}(M)$, then S is not the correct guess of opt $\cap\, U$. Reject this guess, and continue to the next one.
- Otherwise, by Observation 1.3.2, either u or v must be in M. Assume it is u. Since u is not in S, then the algorithm must add v to the cover. Namely, the algorithm set $T \leftarrow T \cup v$.

If the algorithm scans all the edges, then T is a valid candidate to be a vertex cover. If $|T| \leq k$, then the algorithm returns YES, otherwise, it continues to the next guess for S.

Correctness. Observe that if S is guessed correctly, that is $S = \text{opt} \cap U$, then $T = \text{opt}$, as a vertex is added to T only if failing to adding to T would prevent it from being a cover.

Running time. Since $|U| \leq 2k$, there are at most 2^{2k} different guesses for subsets of vertices of S. Completing such a guess S into the cover T, takes $O(n + m)$ time. We conclude that the running time of this algorithm is $O(4^k(n + m))$.

Theorem 1.3.8. *Given a graph* G *with n vertices and m edges, and a parameter k, one can decide in* $O(4^k(n + m))$ *time if* G *has a vertex cover of size at most k.*

1.4. Bibliographical notes

The topic of FPT algorithms is quite interesting. A very nice book on the topic is by Cygan *et al.* [CFK+15] (which was available for free online).

Bibliography

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.