6.854 Advanced Algorithms	
Lecture 14: 10/27/2004	Lecturer: David Karger
Scribes: Pramook Khungurn	

Fixed Parameter Tractability and Treewidth

14.1 Fixed Parameter Tractability

One of the ways to cope with hard problems in practice is to realized that problem instances may be associated with some "parameters" related to their complexity. If the parameter of an instance is small, we might have some hope in finding a polynomial time solution to that instance. This idea has been investigated in the 1990's by Downian and Fellowes in their work on fixed parameter tractability.

We begin by giving formal definitions of parameter and fixed parameter tractability.

Definition 1 *A* parameter *is a function from problem instances to the set of natural numbers.*

In this way, a parameter is an integer associated with an instance of a problem. Clearly, the input size n is a parameter (although we might not be able to achieve anything new by seeing n in this light). Other examples include the number of processors in a multi-processor scheduling problem, and the number of distinct item sizes in the bin packing problems. The value of the optimal solution can also be regarded as a parameter.

Definition 2 *We say that a problem is* fixed parameter tractable (FPT) *with respect to parameter k if there exists a solution running in $f(k) \times n^{O(1)}$ time, where f is a function of k which is independent of n.*

We investigate two techniques used to construct fixed paramemter algorithms: bounded search tree method, and kernelization.

14.1.1 Bounded Search Tree Method

The bounded search tree method attempts to do an exhaustive search on the problem space. However, it uses the parameter to cut of branches of the search tree so that the running time is still polynomial in n. We illustrate the method by showing that the vertex cover problem is fixed parameter tractable with respect to the size of the optimal cover.

Recall that in the vertex cover problem, we are given a graph $G = (V, E)$. We would like to find the subset W with the least number of elements such that, for all $(v_1, v_2) \in E$, either $v_1 \in W$ or $v_2 \in W$.

Let parameter k denote the size of W. If k is small, we can naively check every subsets of V with k elements whether it covers every edge. However, this does not show that the vertex cover problem is fixed parameter tractable with respect to k. The reason is that the time complexity of this algorithm is at least the number of k-subsets of V which is $\binom{n}{k}$ which is of order n^k.

A better solution uses the observation that for every edge, one end point is in the cover. Hence, we can perform an exhaustive search by picking an edge, choosing one end point to include in the cover, update the graph, and recurse. Note that since the size of the smallest vertex cover is k, we do not need to search more than k depths. The psuedocode of the algorithm is as follows.

VERTEX-COVER(G, C)
1 If $|C| = k$, and G still contain an enge, **return** $\emptyset$.
2 If G contains no edges, then **return** C.
3 Pick an edge (v, w).
4 $G' \leftarrow G$ with vertex v and its incident edges removed.
5 $C' \leftarrow C \cup \{v\}$
6 If VERTEX-COVER$(G', C') \neq \emptyset$, then **return** C.
7 $G'' \leftarrow G$ with vertex w and its incident edges removed.
8 $C'' \leftarrow C \cup \{w\}$
9 If VERTEX-COVER$(G'', C'') \neq \emptyset$, then **return** C.

At each depth, we branch into at most two subtrees, and we remove at most n edges from the graph. Thus, the time complexity of the algorithm is $O(2^k n)$, which is in the form that we want. Hence, the vertex cover problem is fixed parameter tractable.

14.1.2 Kernelization

Roughly speaking, a *kernel* is a problem instance whose solution can be easily (i.e. in polynomial time) extended to a solution to the original problem. In other words, the kernel of a problem instance is its "hard" part. By identifying the kernel of a problem instance, we get a smaller problem, and if the size of the kernel is small, we can hope to solve the problem much faster.

As an example, consider again the vertex cover problem with parameter k, the size of the smallest vertex cover. Observe that every vertex with degree more than k has to be in the cover. The reason is that if it is not, then all of its neighbors must be in the cover to cover its incident edges, and this makes the size of the cover greater than k. So, we can add such vertices to the cover, remove their incident edges, and try to find an optimal vertex cover in the remaining graph. This remaining graph is the kernel of the problem because its smallest vertex cover, unioned with the set of vertices with degree more than k in the original graph, gives an optimal solution to the original problem.

If the size of the kernel is bounded by $f(k)$, for some function f independent of n, the time complexity of solving the kernel is $O(g(k))$, for some function g independent of n. Hence, once we find the kernel of the problem, there exists an algorithm whose running time is $O(g(k)+n^{O(1)})$. Note that the $n^{O(1)}$ term comes from identifying the kernel and extending the solution of the kernel to that of the original problem.

The vertex cover problem is a nice example. Observe that the remaining graph has maximum degree

k. Since the original graph can be covered by k vertices, so can the remaining graph. This implies that there are at most k^2 edges, and, hence, at most $2k^2$ vertices. Using the algorithm from the last section, we can solve the kernel in $O(2^k k^2)$ time. As we can count the degree of each vertex in $O(m+n)$ time, we achieve an $O(2^k k^2 + m + n)$ algorithm.

The following theorem relates fixed parameter tractability to kernelization in a very surprising way.

Theorem 1 *The following three statements are equivalent.*

1. *The problem is fixed parameter tractable, i.e, there exists an $O(f(k) \cdot n^{O(1)})$ algorithm to solve it.*
2. *There exists $O(f(k) + n^{O(1)})$ algorithm for the problem.*
3. *The problem can be reduced to kernel of size at most $f(k)$.*

The proof of the theorem, however, is not constructive. (OMG!) Hence, although we know that a problem is fixed parameter tractable, we may not be able to find its kernelization and its $O(f(k) + n^{O(1)})$ algorithm easily.

One might guess that every problem in NP is fixed parameter tractable. However, people believe otherwise. In complexity theory, there is a notion of complexity class $W[1]$. A problem is said to be $W[1]$-complete if a fixed-parameter polyonomial-time algorithm for that problem yields a fixed-parameter polynomial time algorithm for every other problem in $W[1]$. It has been proven that $CLIQUE$ and $INDEPENDENT - SET$ are in $W[1]$. As no one has been able to prove that any of the two problems is fixed parameter tractable, this provides a strong evidence against every NP problem's being fixed parameter tractable.

14.2 Treewidth

14.2.1 Motivation

Many graph problems can be formulated as assigning to each vertex a value from a set of choices, such that the assignment satisfies some constraints. Hence, we can employ the following canonical search algorithm to solve them.

```
FIND-OPT(G, s)
1  Pick a vertex v such that s[v] has not been assigned.
2  m ← worst possible value
3  for each x such that s[v] can take value x
4        do s[v] ← x
5           r ← FIND-OPT(G, s)
6           if r is better than m
7              then m ← r
8  return m
```

The problem with this search algorithm is that its running time is exponential. We might hope to improve its performance by memoization/dynamic programming, i.e., memorizing solutions to subproblems that has been solved. However, in order to do this, the total number of subproblems must be small. The following example illustrates an example when such improvement can be made.

Consider the problem of finding a maximum mathcing on a tree: we are given a tree $T = (V, E)$, and we would like to find the largest subset F of E such that no two edges in F shares a vertex. Since a tree is a bipartite graph, we can use max-flow to solve this problem, but it turns out to be an overkill.

Suppose we pick a vertex r to be the tree's root. For each vertex v, let

- $f^+(v) =$ the size of the maximum matching in the subtree rooted at v in which v is matched;
- $f^-(v) =$ the size of the maximum matching in the subtree rooted at v in which v is not matched.

Clearly, the size of the optimal solution is $\max\{f^+(r), f^-(r)\}$. Let $x \prec y$ denote the sentence "x is the parent of y." We have that, if v is a leaf, $f^+(v) = f^-(v) = 0$. Otherwise,

$$\begin{aligned} f^+(v) &= 1 + \sum_{v \prec w} \left(f^-(w) + \sum_{\substack{v \prec u \\ u \neq w}} \max\{f^+(u), f^-(u)\} \right), \\ f^-(v) &= \sum_{v \prec w} \max\{f^+(w), f^-(w)\}. \end{aligned}$$

With this dynamic programming approach, we can solve the problem in $O(n)$.

One can hope to solve the maximum matching problem in general graphs by designating a vertex as a root, and compute $f^+(v)$ and $f^-(v)$ for each vertex v according to some spanning tree. However, this does not lead to a correct solution because subproblems in general graphs tend to interact. For example, if vertex w is a child of vertex v, it might be the case that when we attempt to calculate the optimal solution in which v is matched to w, the subproblem we solved for w requires that v is matched to some other vertex.

Nevertheless, the good news is that in many graph problems, feasibility is determined locally through interaction with neighbors. In other words, the values that a vertex can take is dependent on only the values that its neighbors take. For example, in the maximum matching above, a vertex can only be paired with one of its neighbors that has not been paired with any other vertex. This is where the concept of treewidth kicks in.

14.2.2 Definition

Definition 3 *An* elimination ordering *is a permutation on the set of vertices V of graph G. We write the ordering by n-tuple $(v_1, v_2, \ldots, v_n)$, where $v_i \in V$, and $v_i \neq v_j$ for all i, j.*

Consider the following process. We are given a graph G and an elimination ordering $(v_1, v_2, \ldots, v_n)$.

1. $i \leftarrow 1$
2. For all pairs of neighbors u and w of v_i, add edge (u, w) to G.
3. Remove v_i and its incident edges from G.
4. $i \leftarrow i + 1$
5. If $i > n$, then terminate. If not, go to step 2.

Definition 4 *The* treewidth induced by elemination ordering $(v_1, v_2, \ldots, v_n)$ of G *is the maximum of all neighborhood size of vertices just before they are removed.*

Definition 5 *The* treewidth *of a graph G is the minimum treewidth induced by some elimination ordering.*

For example, a tree has treewidth 1 because we can keep eliminating leaves. In fact, it is possible, but not obvious to show every graph with treewidth 1 is a tree. Graphs with treewidths 2 are series-parallel graphs which are analogues of networks of resistors (without the resistors) that can be decomposed into nested series and parallel connections. Graphs with treewidth more than 3 are hard to classify.

In the next lecture, we will show that SAT is fixed parameter tractable with respect to treewidth.