Parameterized Complexity

Rebecca Robinson

September 28, 2006

1 Parameterized complexity

In classical complexity, a decision problem is specified by:

- The input to the problem.
- The question to be answered.

Example:

VERTEX COVER

Instance: A graph $G = (V, E)$, and a positive integer k.

Question: Does G have a vertex cover of size $\leq k$? (that is, a collection of vertices V' of G such that for all edges $v_1 v_2$ of G either $v_1 \in V'$ or $v_2 \in V'$.)

A parameterized problem is one where the input to the problem is considered as consisting of two parts, i.e., a pair of strings $(x, y) \in \Sigma^* \times \Sigma^*$. The string y is defined as the *parameter*.

In parameterized complexity, the problem is specified by:

- The input to the problem.
- The aspects of the input that constitute the parameter.
- The question to be answered.

Example:

VERTEX COVER

Instance: A graph $G = (V, E)$

Parameter: A positive integer k

Question: Does G have a vertex cover of size $\leq k$?

Helpful for problems that are NP-hard when the complexity is analysed in terms of the input size only, but which can be solved in a time that is polynomial in the input size and exponential in some parameter k.

By fixing k at a small value, these problems become tractable despite their traditional classification.

2 Fixed-parameter tractability

A parameterized problem $L \subseteq \Sigma^* \times \Sigma^*$ is *fixed-parameter tractable* (FPT) if there is an algorithm that correctly decides, for input $(x, y) \in \Sigma^* \times \Sigma^*$, whether $(x, y) \in L$ in time $f(k)n^\alpha$, where $|x| = n$, $|y| = k$, α is a constant (independent of k), and f is an arbitrary function [Downey and Fellows, *Parameterized Complexity*].

3 Methods for finding FPT algorithms

3.1 Bounded Search Tree Method

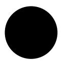
Theorem (Downey and Fellows, 1992): VERTEX COVER is solvable in time $O(2^k|V(G)|)$.

Proof:

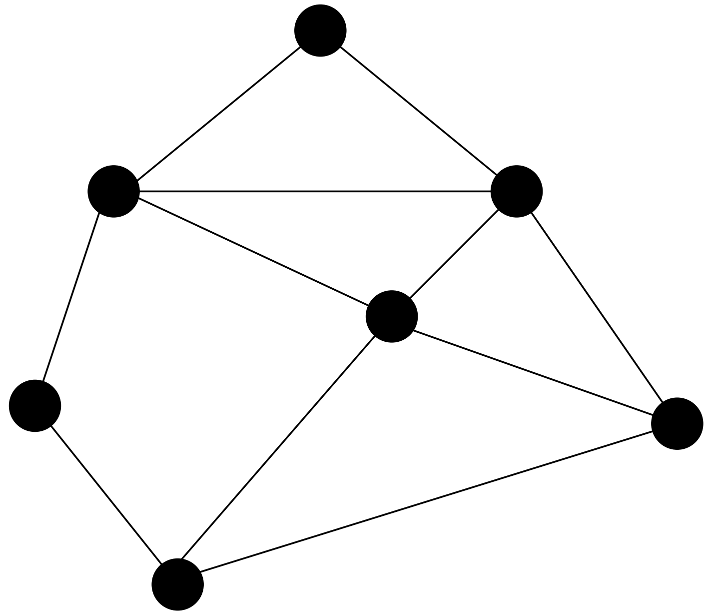
Construct a binary tree of height k. Label each node of the tree with (a) the vertices included so far in a possible vertex cover, and (b) those parts of graph G that are yet to be covered.

Label the root of the tree with the empty set and graph G:

Search tree

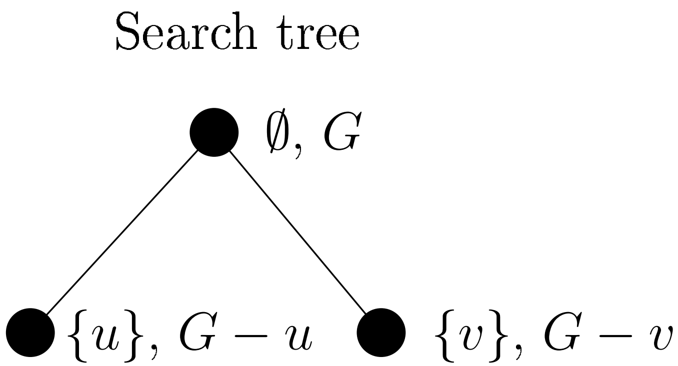


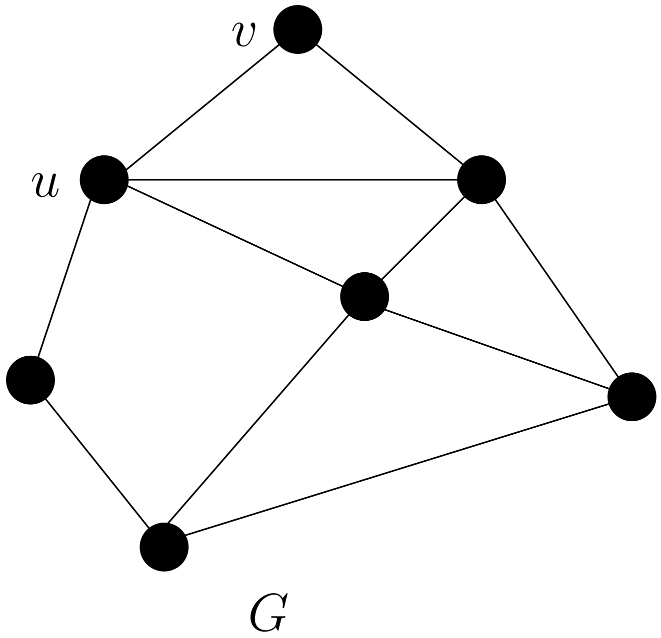
$\emptyset$, G



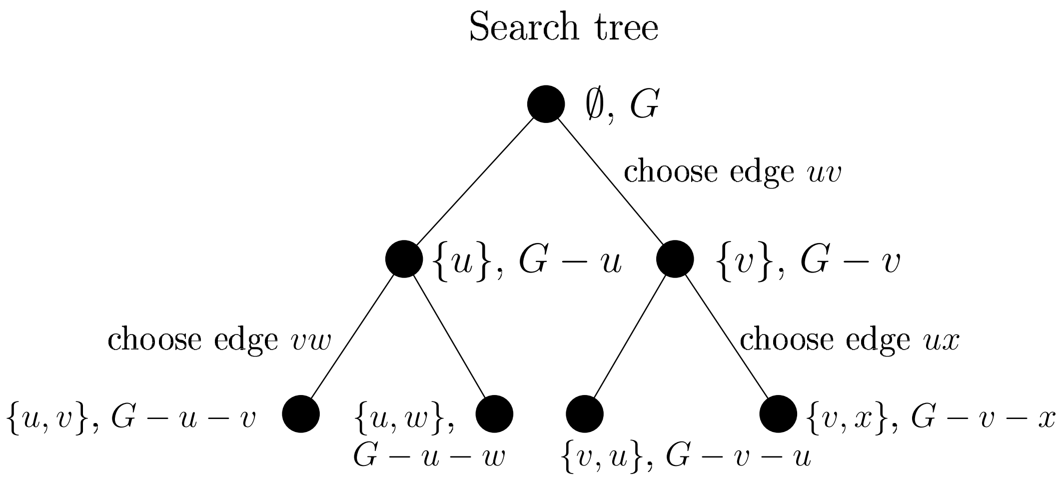
G

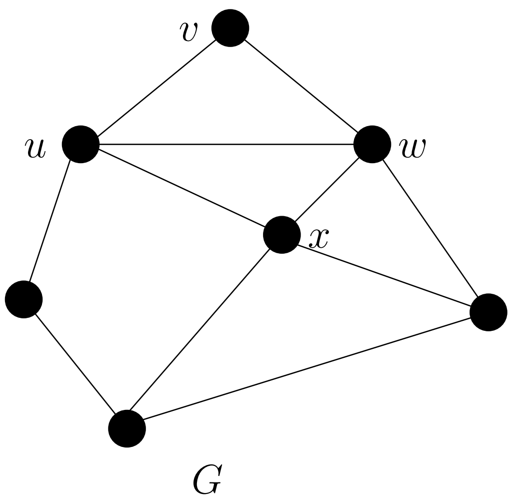
Choose an edge $uv \in E$, then create the two children of the root node corresponding to the two possibilities for the vertex cover.





Continue the process...





If a node is created in the tree that is labelled with a graph having no edges, then a vertex cover has been found. If no such node exists at tree height $\leq k$, then there is no vertex cover of size $\leq k$.

Since the height of the search tree is at most k, the total number of nodes in the tree is less than 2^k. For each node, the task of choosing an edge from G in order to create the two child nodes must be executed. Thus the complexity of the algorithm is $O(2^k|V(G)|)$.

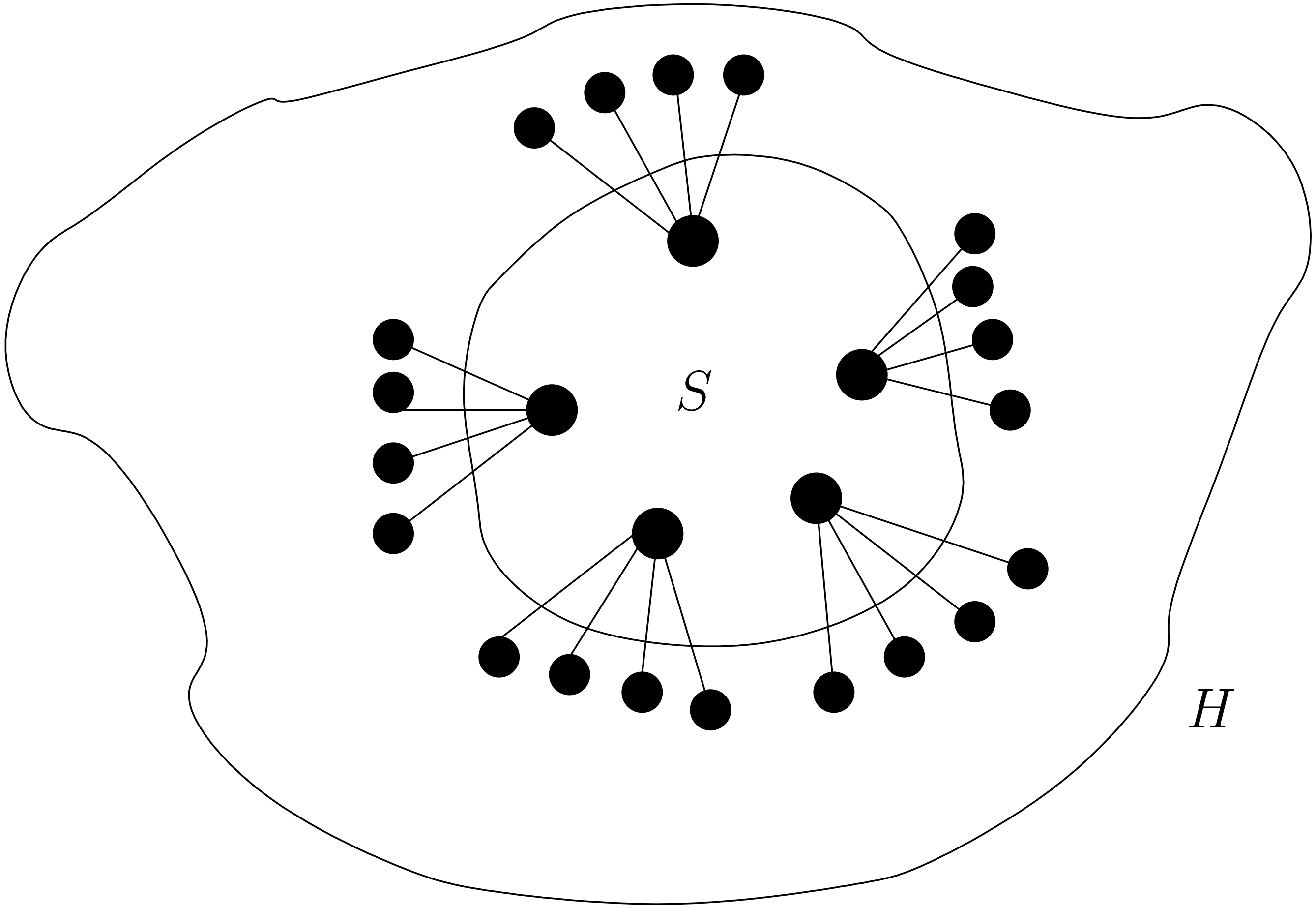
3.2 Reduction to a Problem Kernel

Theorem (Buss, 1989): VERTEX COVER is solvable in time $O(n + k^k)$.

Proof:

For a simple graph H, any vertex of degree $> k$ must belong to every k-element vertex cover of H.

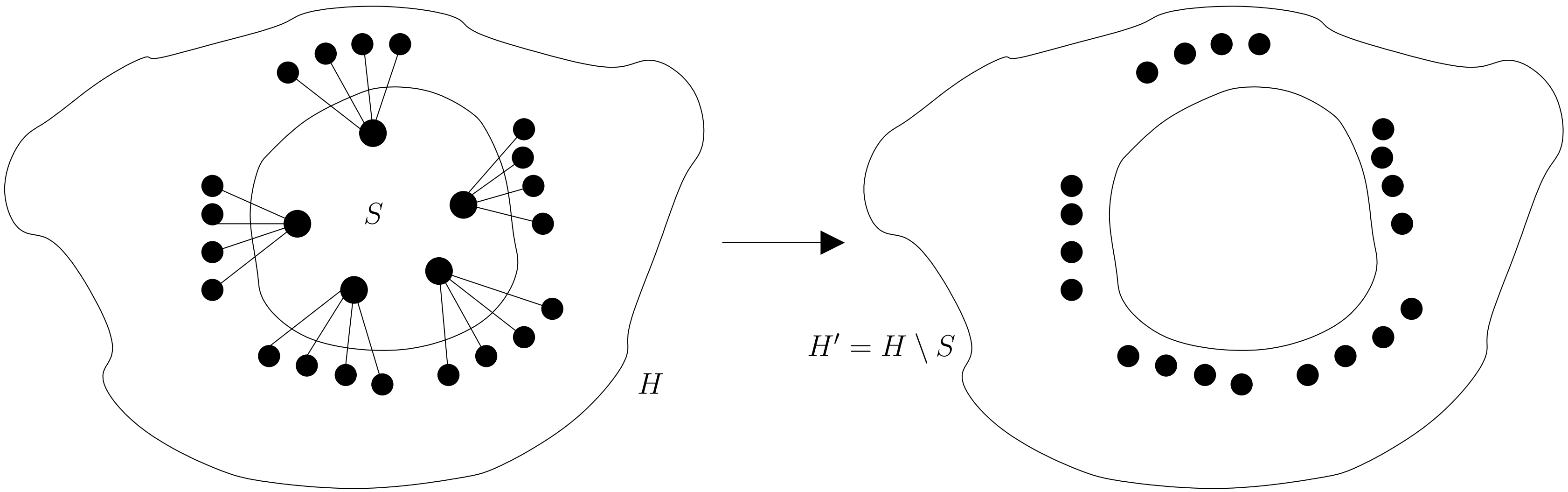
Let S be the set of all vertices in H of degree $> k$. Let $p = |S|$.



If $p > k$, there is no k-vertex cover.

Let $k' = k - p$.

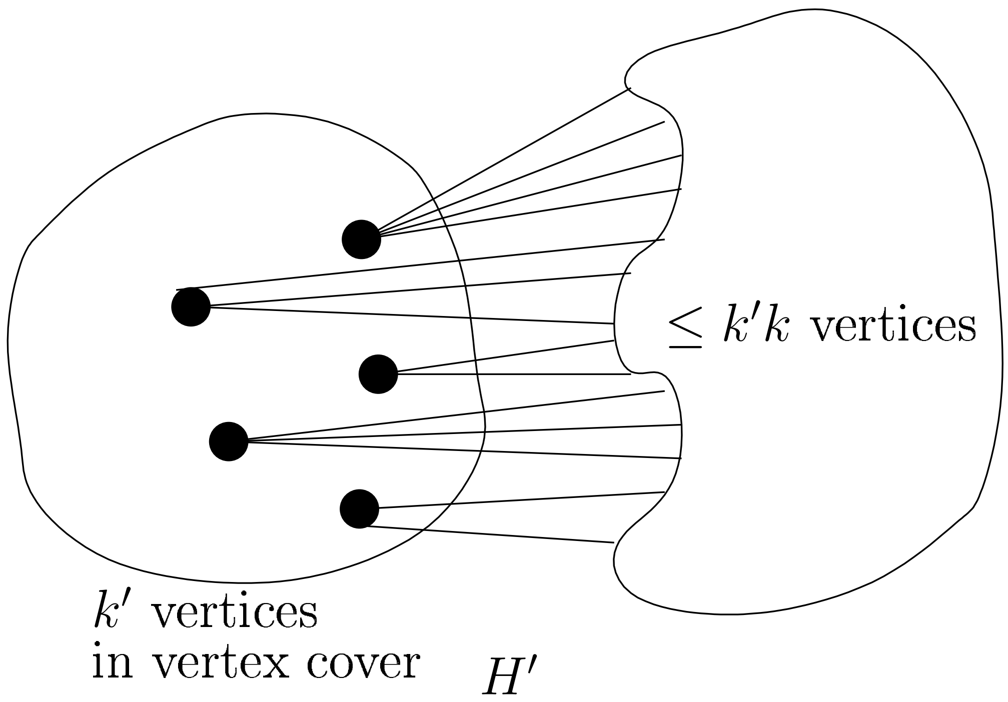
Discard all p vertices of degree $> k$ and the edges incident to them. Call the resulting graph H'.



For H to have a k-vertex cover, H' must have a k'-vertex cover.

The vertices in H' have degree bounded by k.

For H' to have a k'-vertex cover, H' must have no more than $k'(k+1)$ vertices - the k' vertices of the vertex cover plus a maximum of k vertices adjacent to each of those vertices: $k' + (k' \times k) = k'(k+1)$.



Thus, if H' has more than $k'(k+1)$ vertices, reject.

Since we have now limited the size of H' to a function of the parameter k, we can establish in constant time if H' has a k'-vertex cover.

If H' has no k'-vertex cover, reject. Otherwise, any k'-vertex cover of H' plus the p vertices from the initial step gives a k-vertex cover of H.

Finding all p vertices of degree $> k$ in H takes linear time.

Since H' has $\leq k'(k+1)$ vertices, there are at most $\binom{O(k)}{k'}$ possibilities to check in finding a k'-vertex cover of H', giving this step a complexity of $O(k^k)$.

Thus the complexity of the whole procedure is $O(n + k^k)$.

3.3 Reduction to a Problem Kernel: Example 2

k-LEAF SPANNING TREE

Instance: A graph $G = (V, E)$

Parameter: A positive integer k

Question: Is there a spanning tree of G (a tree that contains all the vertices in $V(G)$) with at least k leaves?

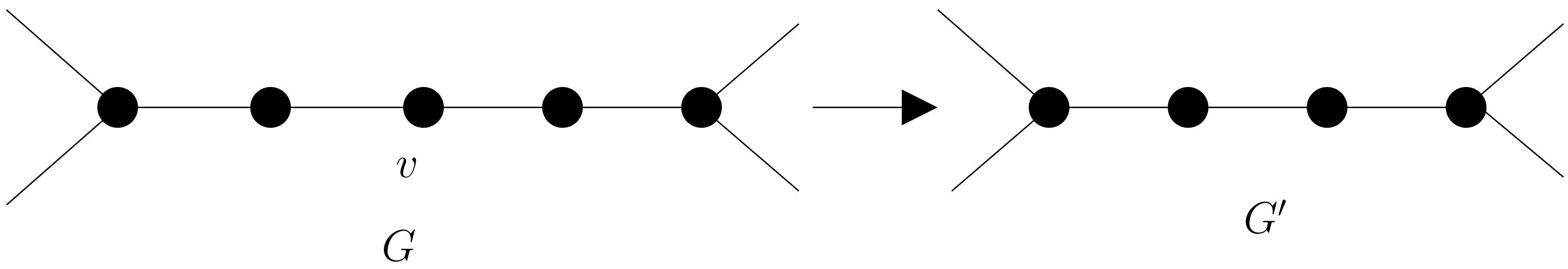
Theorem (Downey, Doyle and Fellows, 1995):

k-LEAF SPANNING TREE is solvable in time $O(n + (2k)^{4k})$.

Suppose G has a k-leaf spanning tree. Then G must be connected.

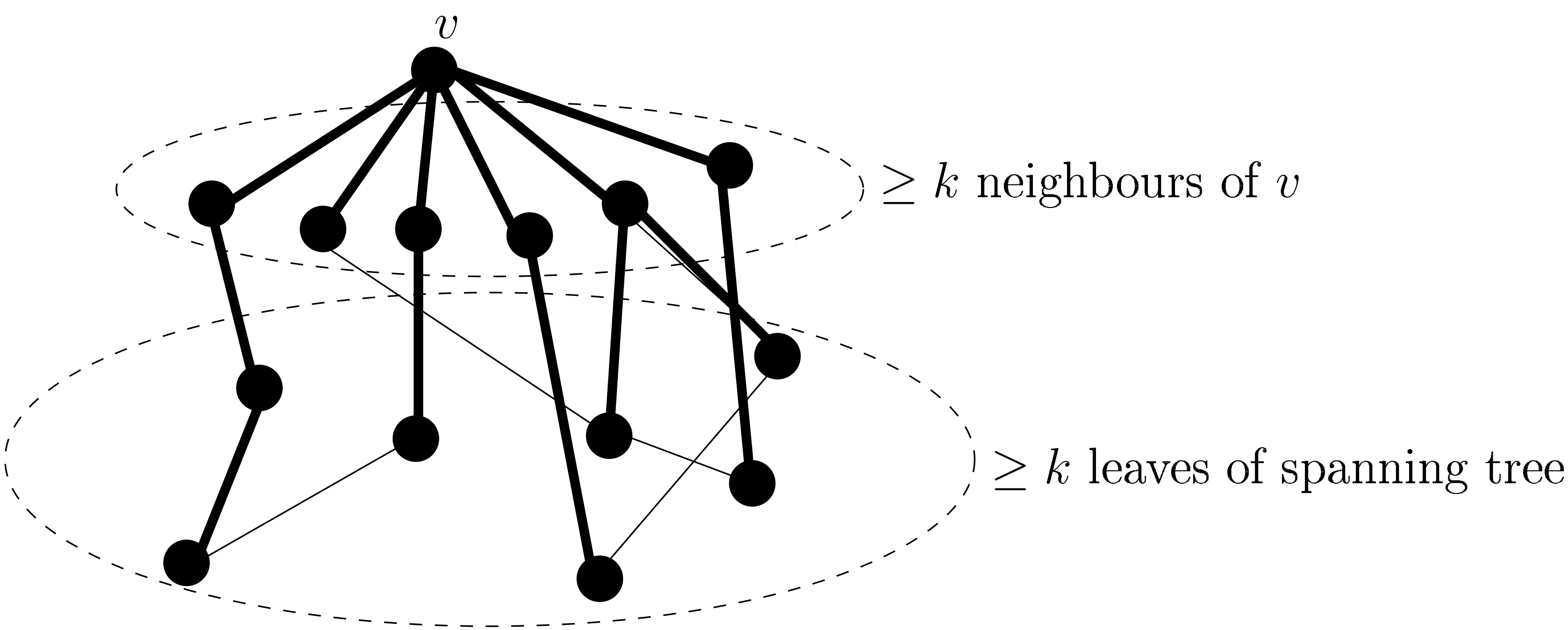
A vertex v is called *useless* if it has degree exactly 2, and both of its neighbours have degree exactly 2.

A useless vertex v is *resolved* by deleting v from G and adding an edge between its two neighbours. Call G' the graph obtained from G by resolving all useless vertices. This process can be completed in linear time.



Algorithm for k-LEAF SPANNING TREE

Step 1. Check whether G is connected and whether there is a vertex of degree $\geq k$. If G has such a vertex, the answer is *yes*.



Step 2. Compute G'. If G' has at least $4(k+2)(k+1)$ vertices, the answer is *yes*. (See proof later on.)

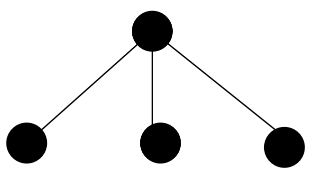
Step 3. If G' has fewer than $4(k+2)(k+1)$ vertices, exhaustively analyse G'. G has a k-leaf spanning tree if and only if G' does.

Claim:

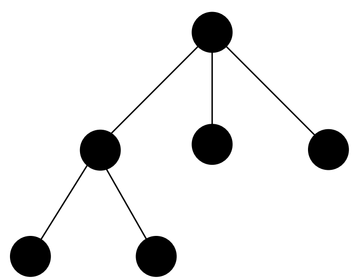
If H is a connected simple resolved graph with at least $4(k+2)(k+1)$ vertices, H has a spanning tree with at least k leaves.

Proof:

(*) If a tree T has i internal (nonleaf) vertices of degree ≥ 3, then T has at least $i+2$ leaves.



Simplest tree with 1 internal vertex of degree ≥ 3 has 3 leaves



Need to add at least 2 leaves to increase number of vertices of degree ≥ 3

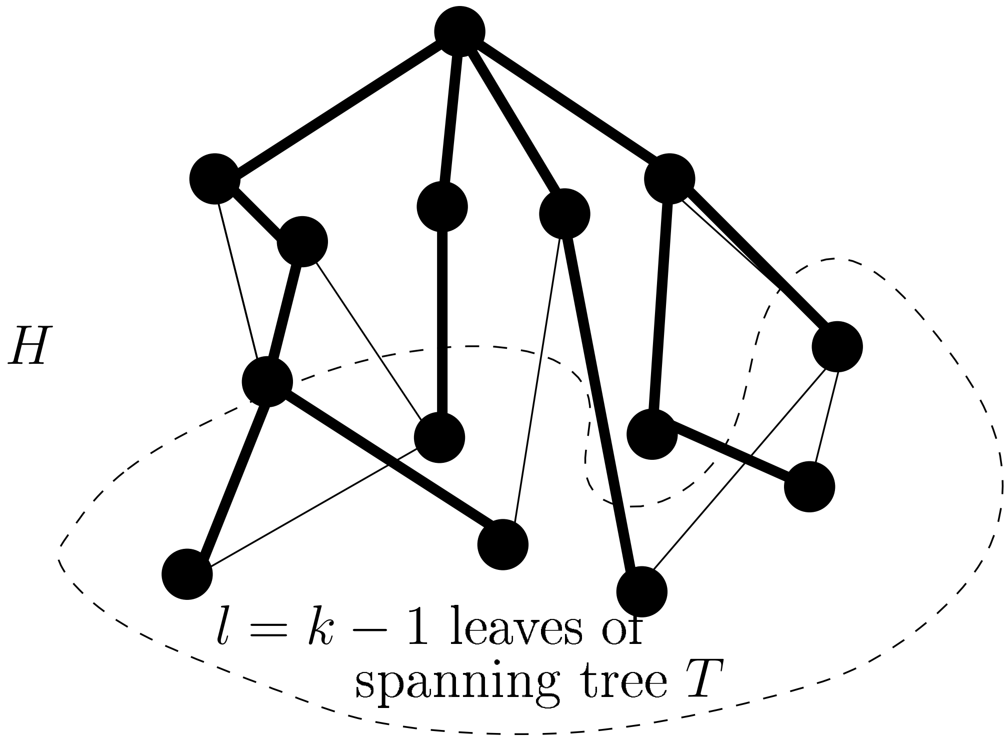
Suppose H has at least $4(k+2)(k+1)$ vertices.

If H has a vertex of degree k, then H has a k-leaf spanning tree.

Suppose then that H has no vertex of degree k.

Let T be a spanning tree of H with maximum number of leaves l.

Suppose $l \leq k - 1$.



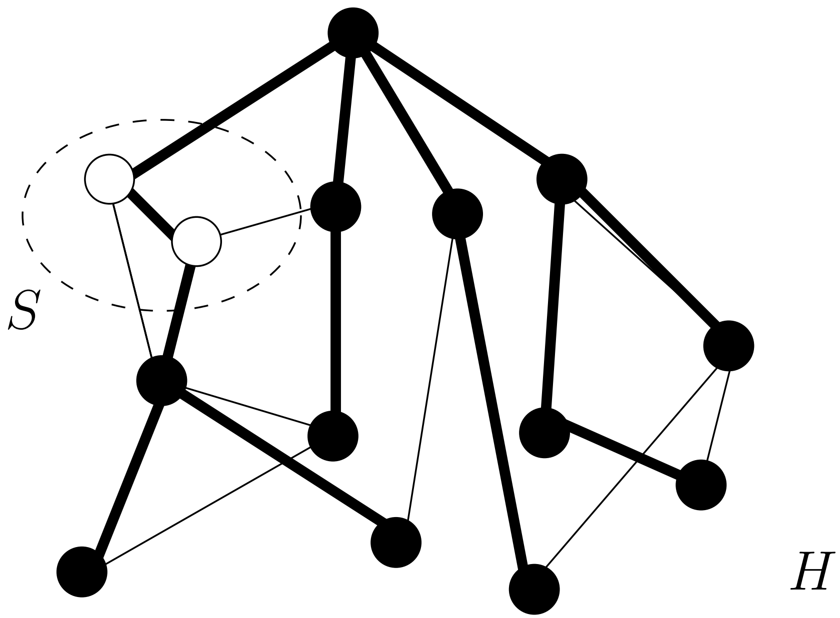
The number of internal vertices in T with degree ≥ 3 is $\leq k - 3$ (from (*))

So T has:

- $4(k+2)(k+1)$ vertices
- at most $k-1$ leaves
- at most $k-3$ vertices with degree ≥ 3

Thus T must have at least $4(k+2)(k+1)-(k-3)-(k-1)$ vertices of degree 2.

Let S be the set of all vertices of degree 2 in T that are not adjacent in H to any leaf of T.



Since there are at most $k - 1$ leaves in T, and the maximum degree of any vertex in H is $k - 1$, the maximum possible number of vertices in H that are adjacent to some leaf in T is $(k-1)(k-1)$.

Since T has at least $4(k+2)(k+1) - (k-3) - (k-1)$ vertices of degree 2, the size of S is at least:

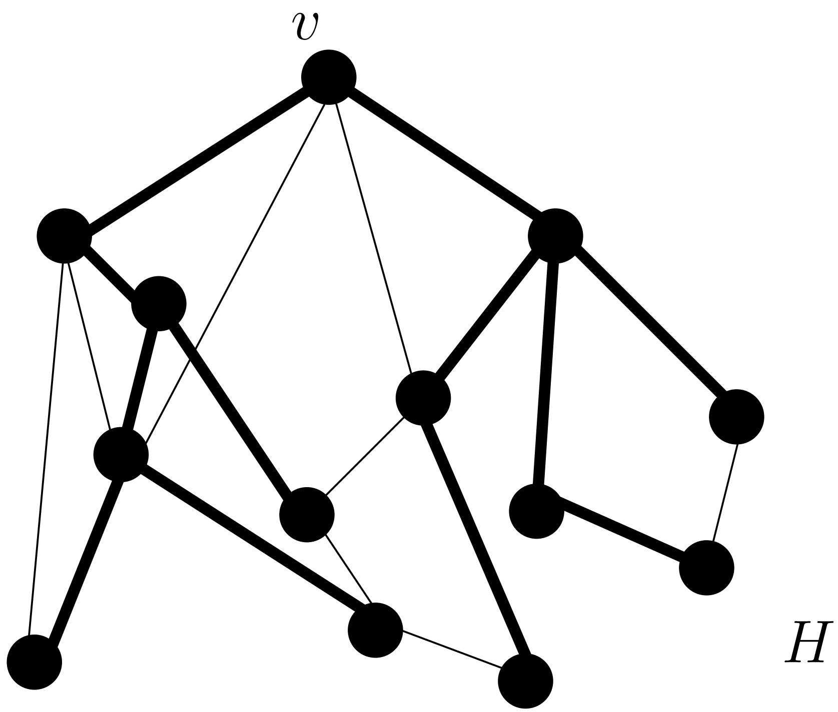
$$4(k+2)(k+1) - (k-3) - (k-1) - (k-1)(k-1) =$$
$$3k^2 + 12k + 11 = 3(k+1)(k+3) + 2$$

There are two cases.

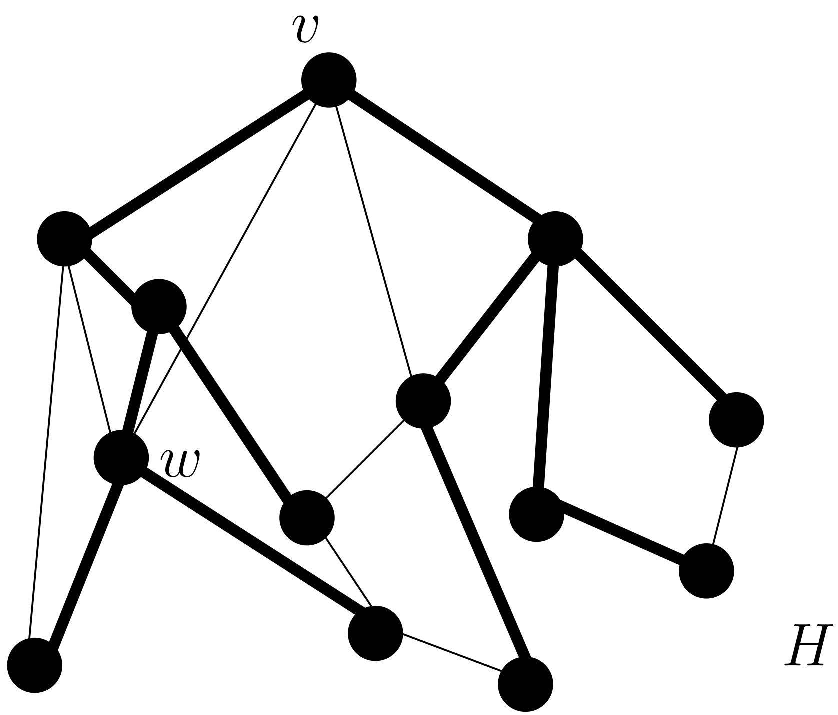
Case 1

Suppose at least $2k - 4$ *of the* $3(k+1)(k+3)+2$ *vertices in* S *have degree* ≥ 3 *in* H*.*

Let v be some such vertex not adjacent in H to any leaf of T. Regard T as rooted with root v.

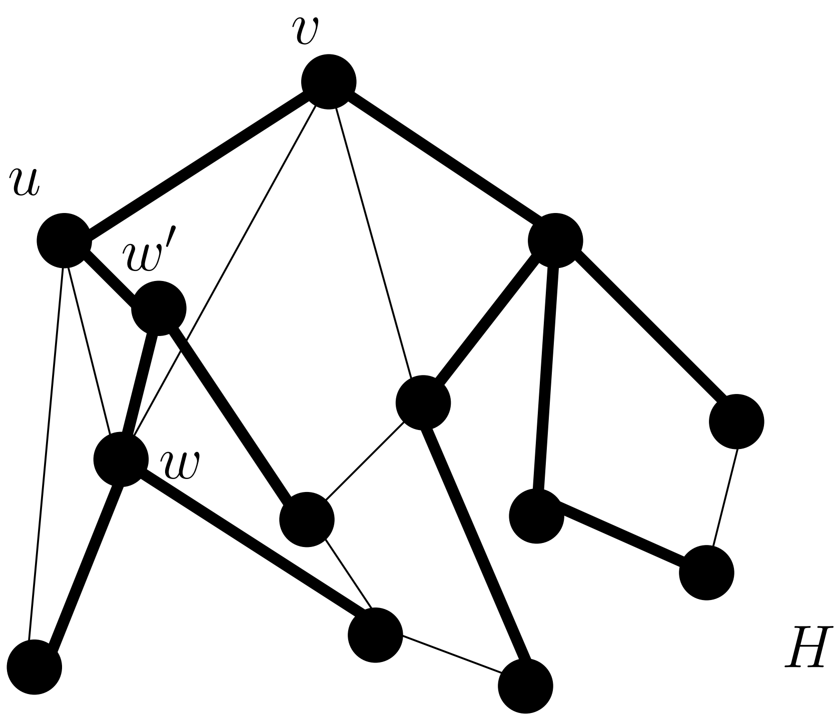


Since v has degree 2 in T, but degree ≥ 3 in H, v must be adjacent to some vertex w in H which is not a child of v in T, such that w is not a leaf of T.



Suppose firstly that there is some choice of v such that one internal vertex of the path from v to w has degree 2 in T.

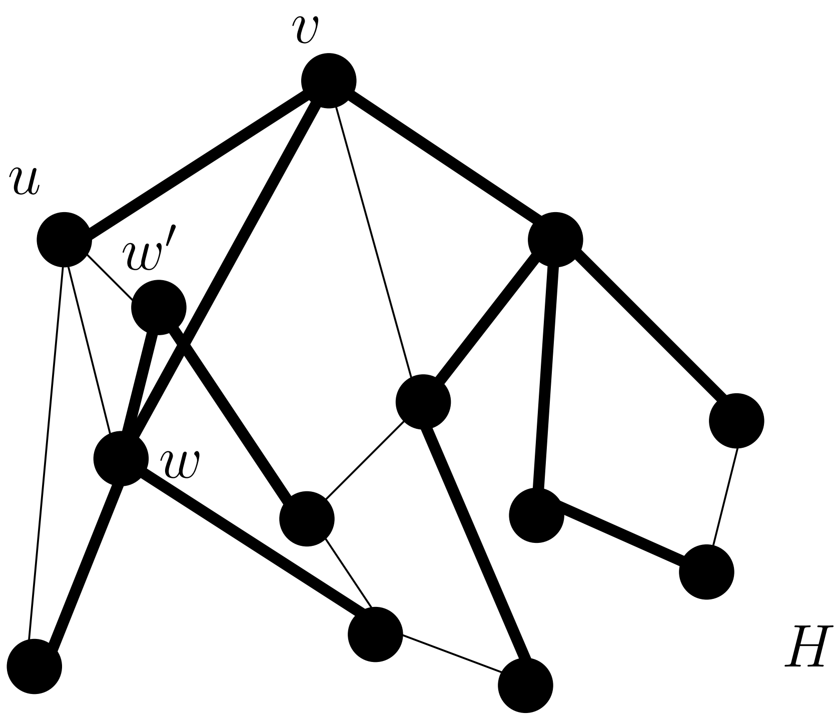
Let u be some node of degree 2 in T on the path from v to w. Let w' be the child of u.



Change T into a new tree T' as follows:

Make u a leaf by deleting edge uw'.

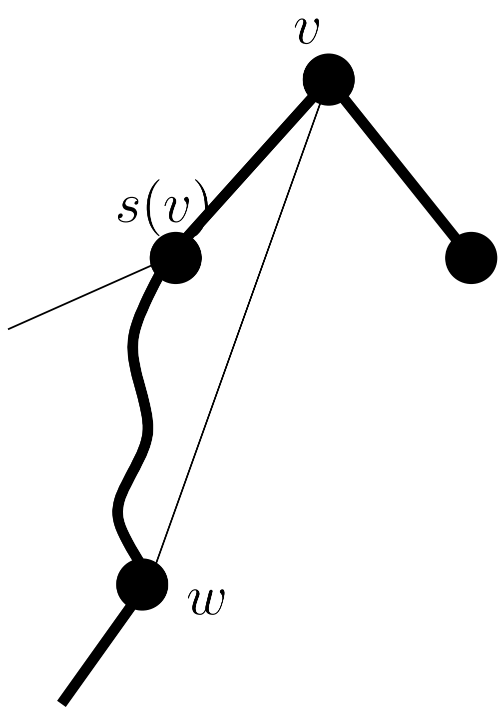
Add a new edge vw to the new tree.



Since neither v nor w were leaves in T, this must increase the net number of leaves.

Suppose now that for any choice of v and w, every internal vertex on the path in T from v to w is of degree ≥ 3 in T.

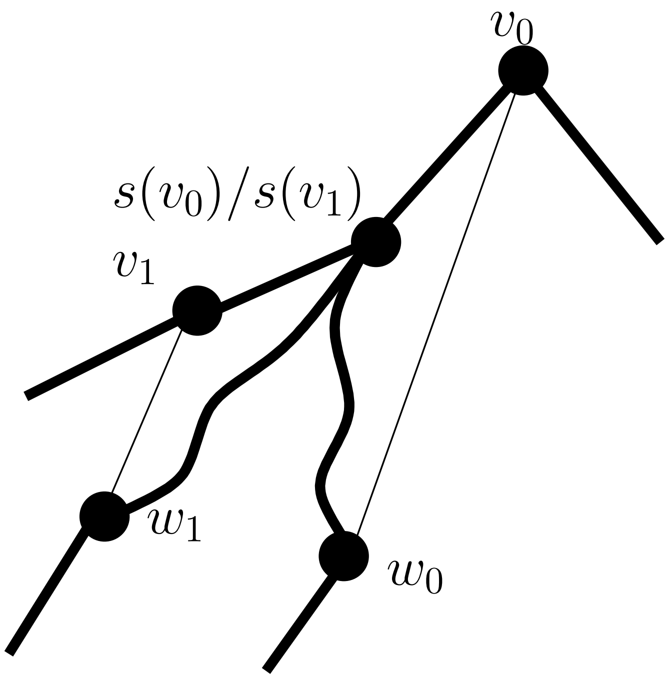
For each possible choice of v, call $s(v)$ the child of v that lies on the path in T from v to w.



There are $2k-4$ possible choices of v, so there are also $2k-4$ possible values for $s(v)$. However, some values of $s(v)$ may be the same for different choices of v — they may not all be distinct vertices.

Let v_0 be some choice of v. Let w_0 be the vertex adjacent to v_0 in H but not in T.

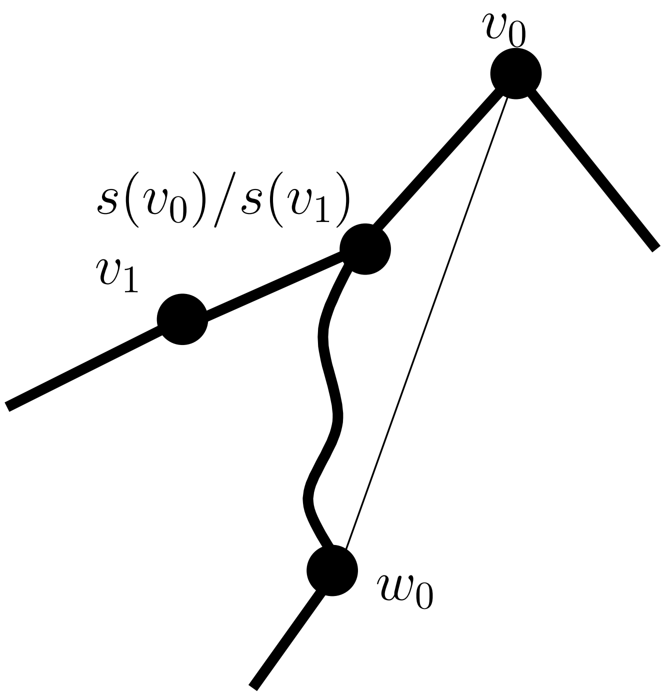
Suppose that the vertex $s(v_0)$ is also an s-value for some other choice of v, say v_1. Let w_1 be the vertex adjacent to v_1 in H but not in T.



The vertex v_1 cannot lie internally on the path from $s(v_0)$ to w_0, since v_1 is of degree 2 in T, and this path has only internal vertices of degree ≥ 3.

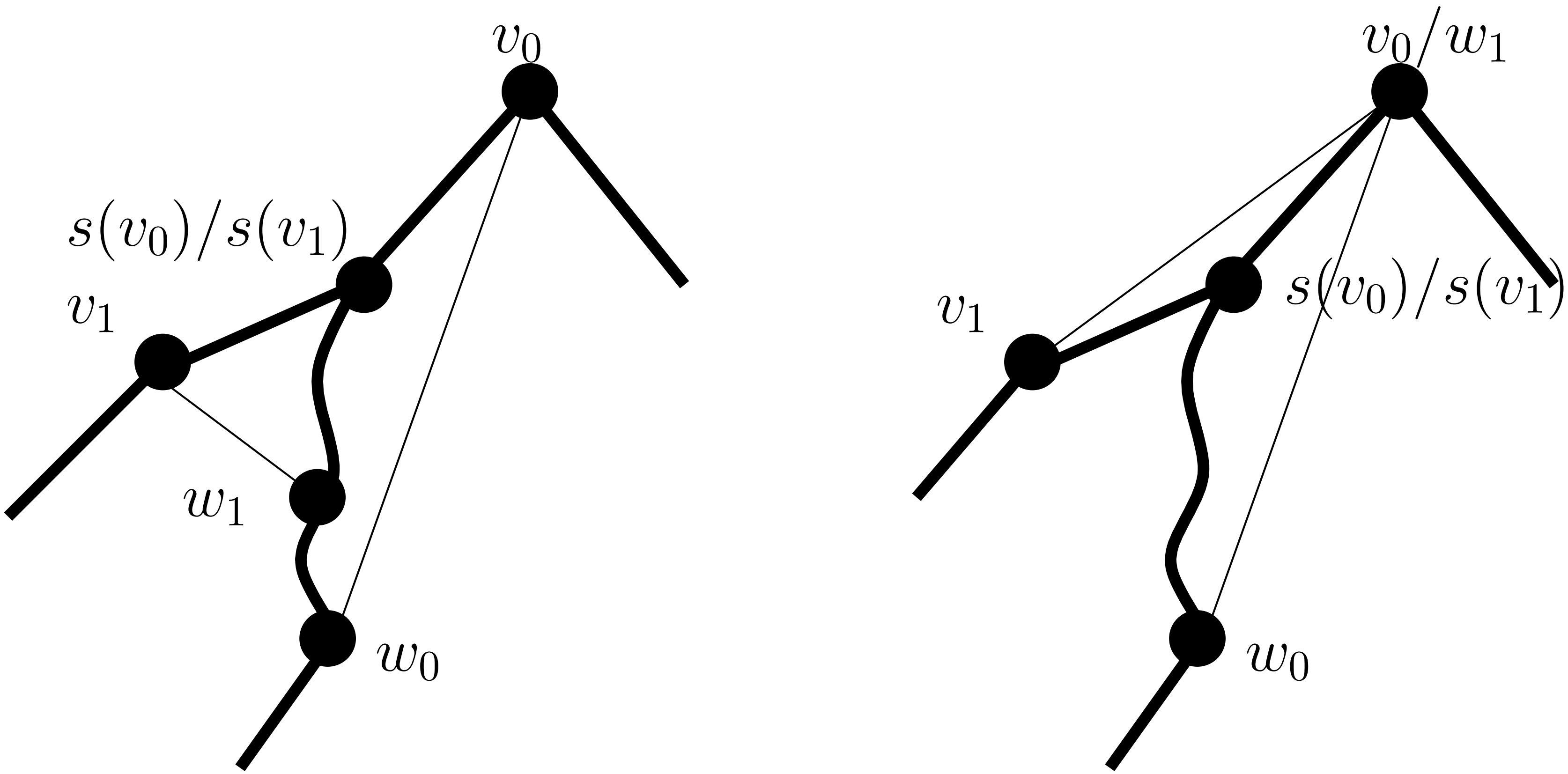
Thus there are two options for v_1: either it is a child of $s(v_0)$ in T which does not lie on the path from $s(v_0)$ to w_0; or v_1 and w_0 are the same vertex.

Assume the first of these is true.



The path from v_1 to w_1 in T cannot internally contain v_0, since v_0 is of degree 2 in T.

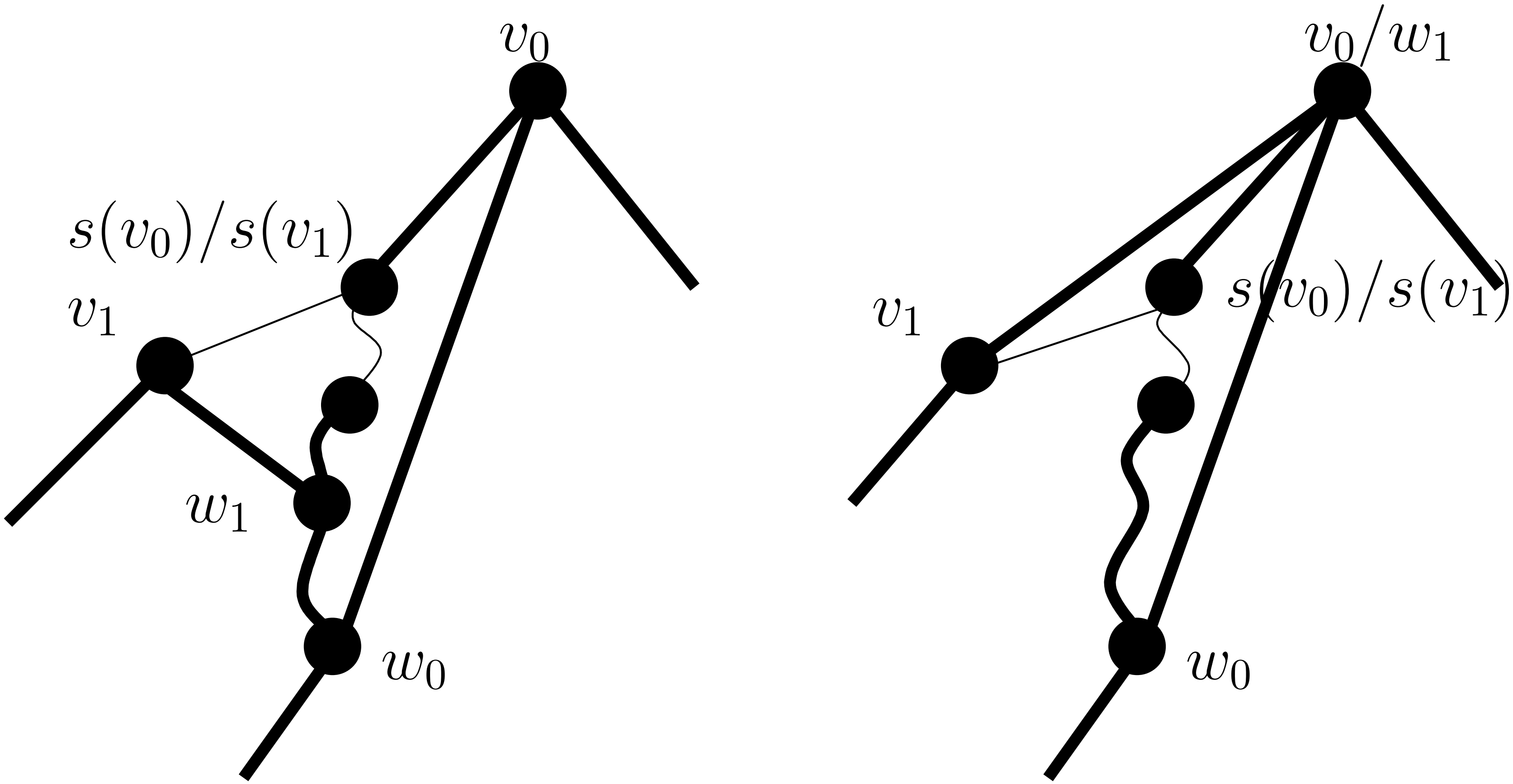
Vertex w_1 can be positioned in the following ways:



In each of these cases, T can be changed into a new tree T' as follows:

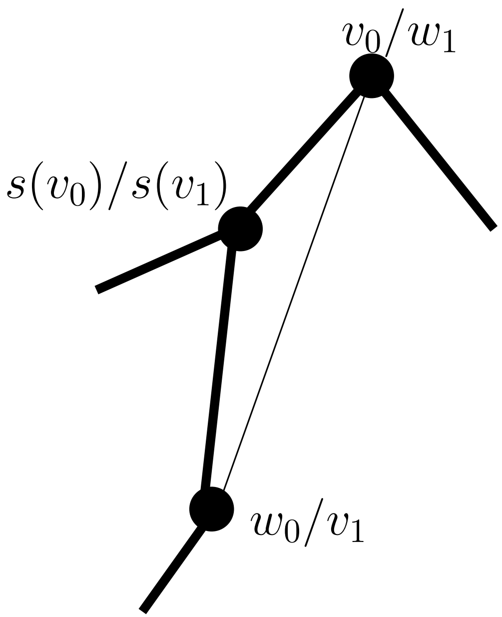
Make $s(v_0)$ a leaf by deleting the edge $v_1 s(v_0)$ and all edges incident to $s(v_0)$ except $s(v_0)v_0$.

Add the new edges $v_0 v_1$ and $v_0 w_0$ to the tree.



This increases the net number of leaves in the tree, which contradicts our initial condition that T have maximum number of leaves.

Thus v_1 must be w_0, and w_1 must be v_0.

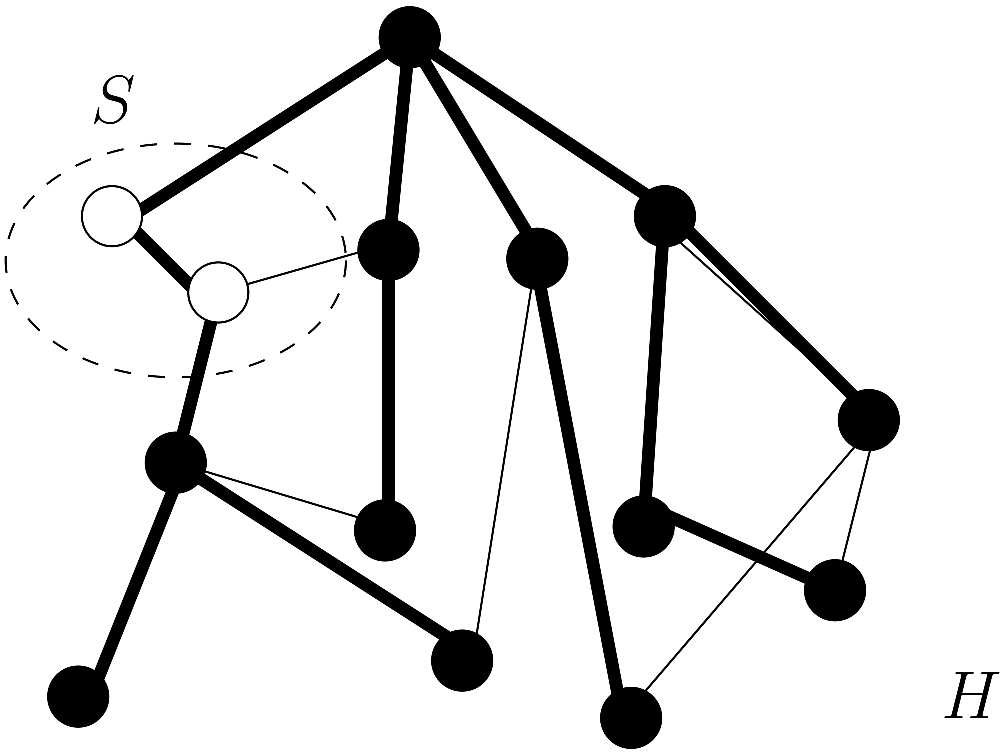


This means that each possible vertex $s(v)$ can only be an s-value for at most two choices of v. So the minimum number of s-values in T is $\frac{2k-4}{2} = k - 2$.

Thus there are at least $k - 2$ internal vertices of degree ≥ 3 in T. But this means from (*) that there are k leaves, which contradicts our initial condition that $l \leq k - 1$.

Case 2

Suppose at most $2k-5$ *of the* $3(k+1)(k+3)+2$ *vertices in* S *have degree* ≥ 3 *in* H.



Each of the remaining $3k^2 + 10k + 16$ vertices in S which are of degree 2 in both T and H are connected to at least one vertex of degree ≥ 3 in H (otherwise they are useless vertices).

If every one of these adjacent vertices of degree ≥ 3 in H is distinct, then there are $3k^2 + 10k + 16$ such vertices. However, this may not be the case.

Since each vertex of H has maximum degree $k-1$, the number of such vertices of degree ≥ 3 in H is at least:

$\lceil \frac{3k^2+10k+16}{k-1} \rceil = \lceil \frac{(3k+13)(k-1)+19}{k-1} \rceil \geq 3k+14$

Since vertices in S cannot be adjacent in H to leaves of T, there must be at least $3k+14$ vertices of degree ≥ 3 in H that are internal vertices of T.

At most $2k - 5$ of these internal vertices can have degree 2 in T.

Therefore at least $(3k + 14) - (2k - 5) = k + 19$ of these internal vertices will have degree ≥ 3 in T.

Thus (from (*)), T has at least $k + 21$ leaves. This gives us a spanning tree with more than k leaves.

Claim:

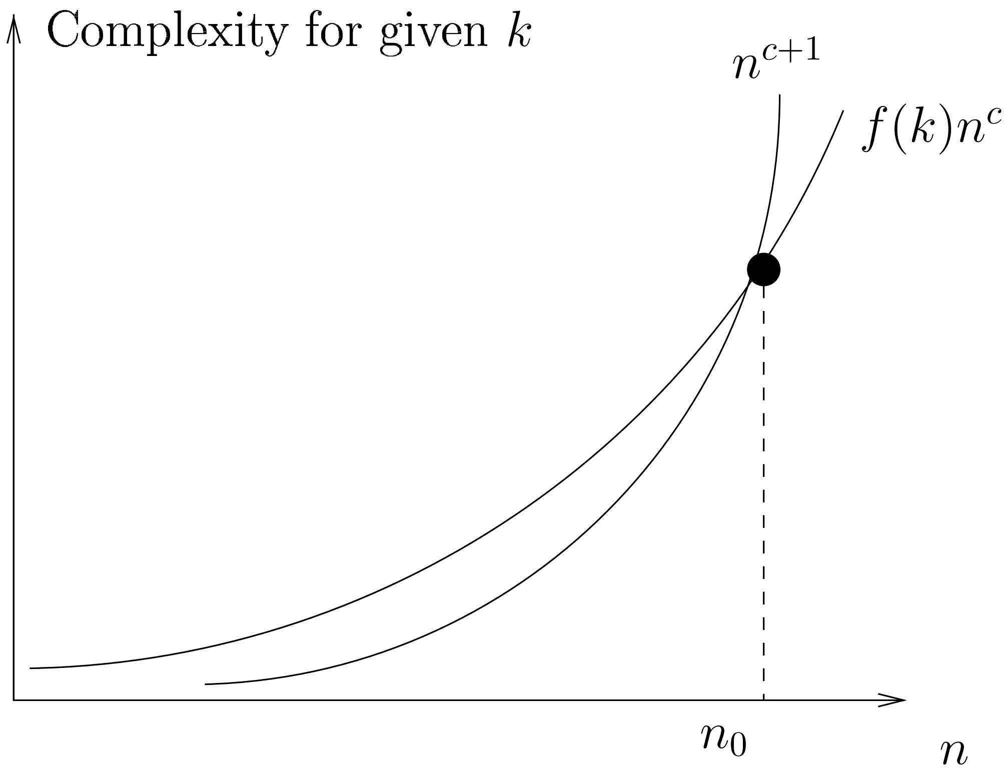
If there exists an algorithm for solving a parameterized problem which runs in time $f(k)n^c$, there also exists an algorithm for solving the same problem that runs in time at most $g(k) + n^{c_1}$.

Let M_k be an algorithm that solves some parameterized problem.

$M_k(x, k)$ accepts iff $(x, k) \in L$.

On input (x, k), algorithm M_k runs in time $f(k)n^c$.

Compare $f(k)n^c$ and n^{c+1}.



For larger values, n^{c+1} will have the greater complexity. Let n_0 be the point at which the two values are the same.

New algorithm:

For all values of n where $f(k)n^c > n^{c+1}$ (i.e., up to n_0), determine if $(x, k) \in L$ using algorithm M_k, and list the results in some table T. The complexity of this step is a function of k, since the size of the table (the value of n_0) is dependent on k.

Then:

If $|x| > n_0$, run M_k on input (x, k). The algorithm runs in time $< |x|^{c+1}$ with this input.

If $|x| \leq n_0$, look up x in table T to find if $(x, k) \in L$. This step runs in a time which is some function of k.

Thus, if there exists an algorithm for solving a parameterized problem which runs in time $f(k)n^c$, there also exists an algorithm for solving the same problem that runs in time at most $g(k) + n^{c_1}$.