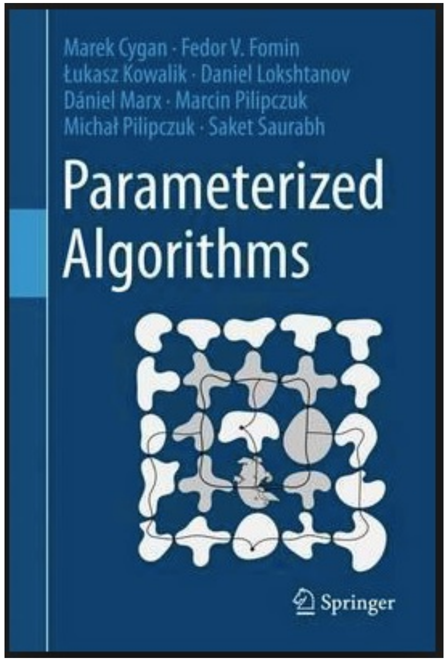


Chapter 3.

(2) Depth-Bounded Search Trees

Zoom out for a moment to think about the big picture

- Previously: kernelization – shrink input to its core (kernel) in polynomial time
- But how do we solve the resulting smaller problem instance?
- We know that the problem is computationally hard (i.e. NP-hard) and, so, at some point we have to do an exhaustive search (unless P=NP)
- Can we come up with strategies to design an exhaustive search that is as \`efficient’ as possible?
- One such technique is depth-bounded search.

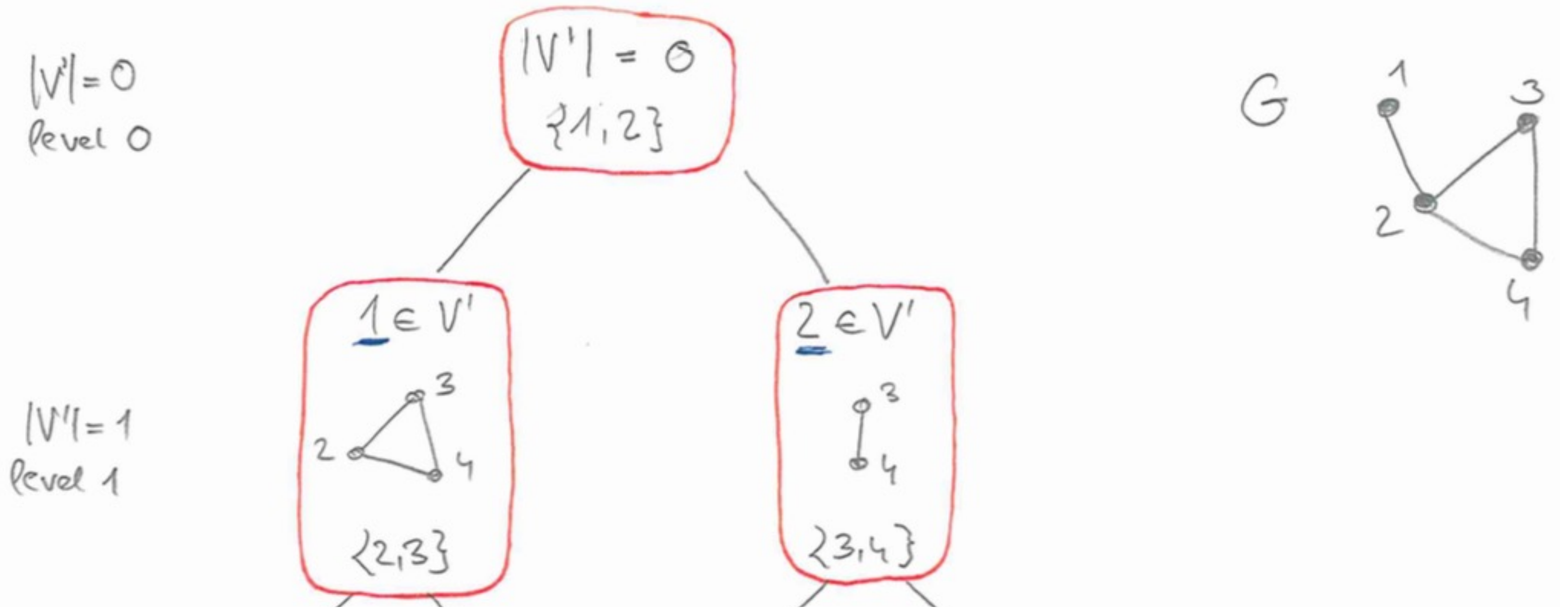
Idea of a bounded search

- Explores the search space to find an optimal solution. E.g. for VC, the search space consists of all $2^{|V(G)|}$ subsets of the vertices of a graph G. Each such subset may or may not be a feasible solution.
- How can we explore the huge search space systematically?
- Turns out that a search can be organized in a tree like fashion where each vertex in the tree corresponds to a possible solutions.
- We explore tricks to bound the size of this tree (e.g. bound the number of children each vertex of the tree can have).

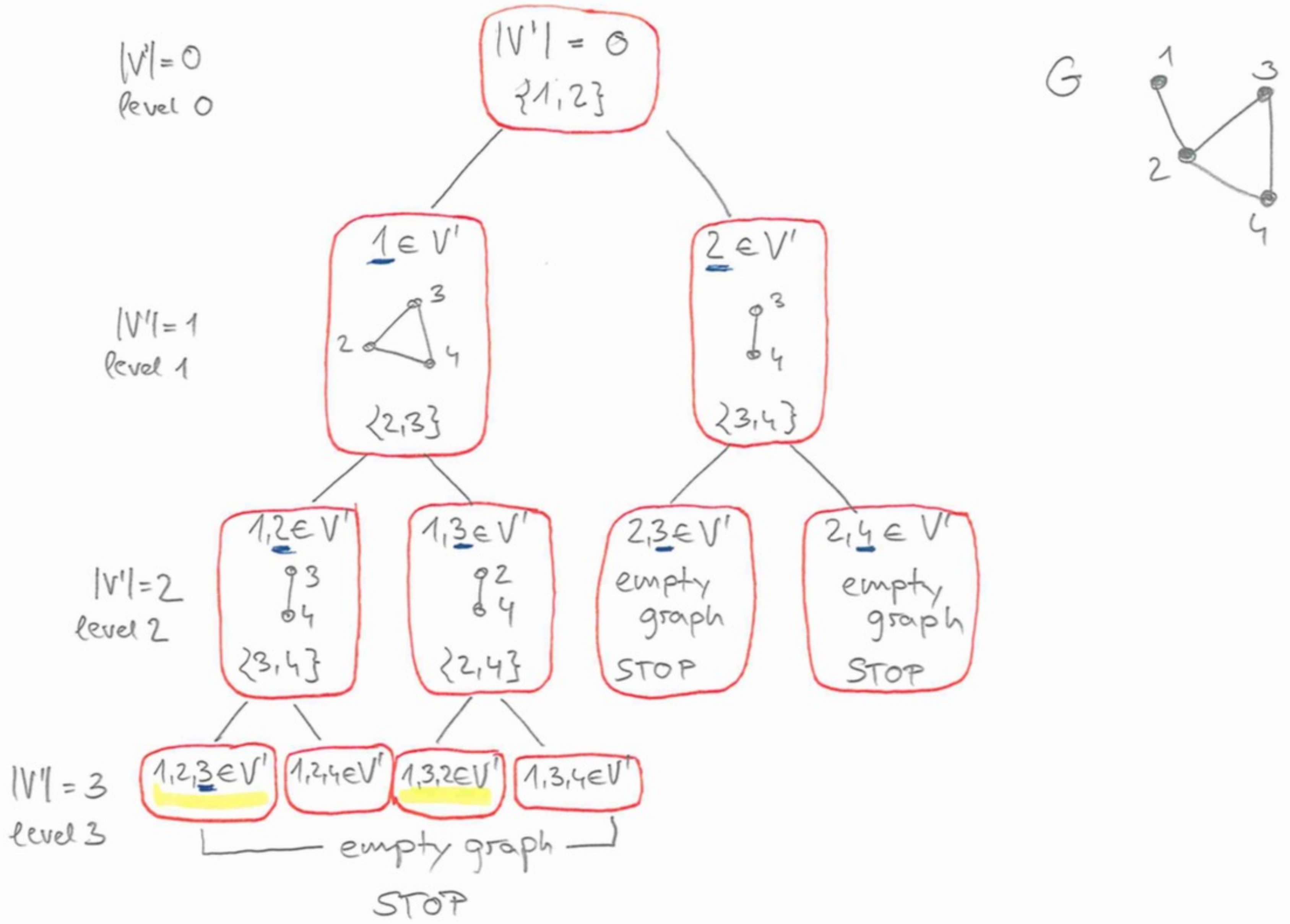
Vertex Cover (VC)

Observations.

Let *G=(V,E)* be a graph, and let *V'* be a vertex cover of *G*. For each edge *{u,v}* in *E*, either *u* or *v* (or both) is in *V'*.



|V'| = 0
level 0
|V'| = 0
{1, 2}
G
1
2
3
4
1 ∈ V'
3
2
4
{2, 3}
2 ∈ V'
3
4
{3, 4}
|V'| = 1
level 1



|V'|=0
level 0
|V'| = 0
{1,2}
G
1
3
2
4
1 ∈ V'
2 ∈ V'
|V'|=1
level 1
3
2
4
{2,3}
3
4
{3,4}
|V'|=2
level 2
1,2 ∈ V'
3
4
{3,4}
1,3 ∈ V'
2
4
{2,4}
2,3 ∈ V'
empty graph
STOP
2,4 ∈ V'
empty graph
STOP
|V'| = 3
level 3
1,2,3 ∈ V'
1,2,4 ∈ V'
1,3,2 ∈ V'
1,3,4 ∈ V'
empty graph
STOP

Bounded-search tree for VC

- Tree lists all possible solutions. (*Is each possible solution considered exactly once?*)
- Level corresponds to the size of a (potential) vertex cover for G.
- If one is interested in deciding if $(G=(V,E),k)$ is a yes-instance of VC, one can stop at level k.
- Level k of the search tree has 2^k vertices. Levels $0,1,\ldots,k-1$ have collectively 2^k-1 vertices.
- Size of the search tree is $O(2^k)$ [(1)].
- Hence, we can solve VC in time $O(2^k |E|)$ which is fpt! Time to remove edges from the graph at each step is $O(|E|)$.
- Ideally, combine kernelization and bounded-search tree: $O(2^k k^2 + |V|^2)$.

[(1)] Has been improved to $O(1.28^k)$.

General idea of a bounded-search tree

Find, in polynomial time, a small constant-size subset *S* of the input such that at least one element in *S* is an element of an optimal solution.

What does this mean in the context of VC?

Let *(G=(V,E),k)* be an instance VC.

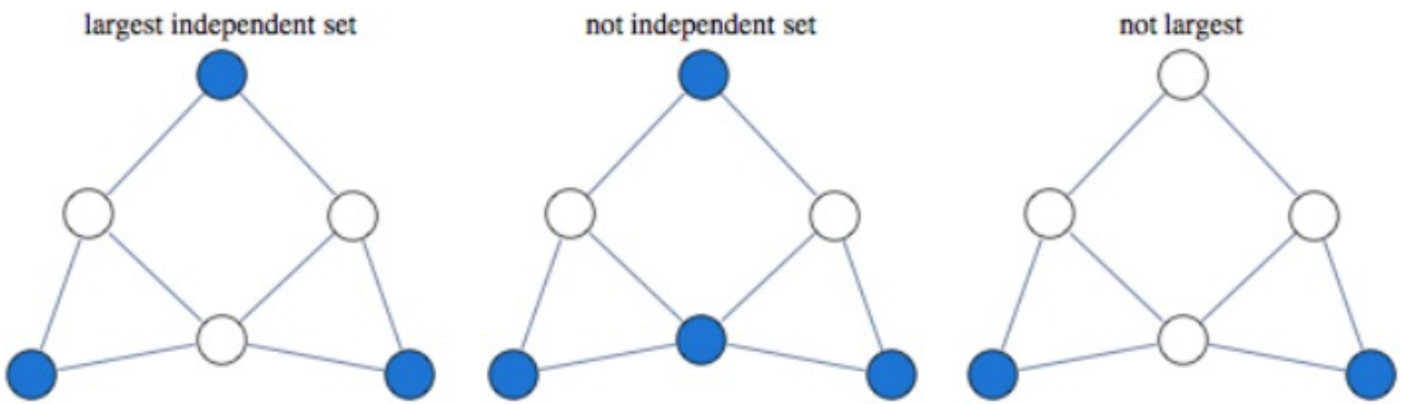
Let *S={u,v}* (i.e. a set of two vertices} such that *{u,v}* ∈ *E*. Then, by our earlier observation, we know that one of *u* and *v* is contained in **any** vertex cover of *G.*

Independent Set (IS) -- a maximization problem

Instance. A graph $G=(V, E)$ and a non-negative integer k.

Question. Is there a subset I of V such that $|I| \geq k$ and no two vertices in I are adjacent to each other?

In other words, no two vertices in I are connected by an edge in E.



Independent Set (IS) vs. Vertex Cover (VC)

VC Instance. A graph $G=(V, E)$ and a non-negative integer k.

VC Question. Is there a subset V' of V such that $|V'| \leq k$ and for each edge $\{u,v\}$ in E, at least one of u and v is in V'?

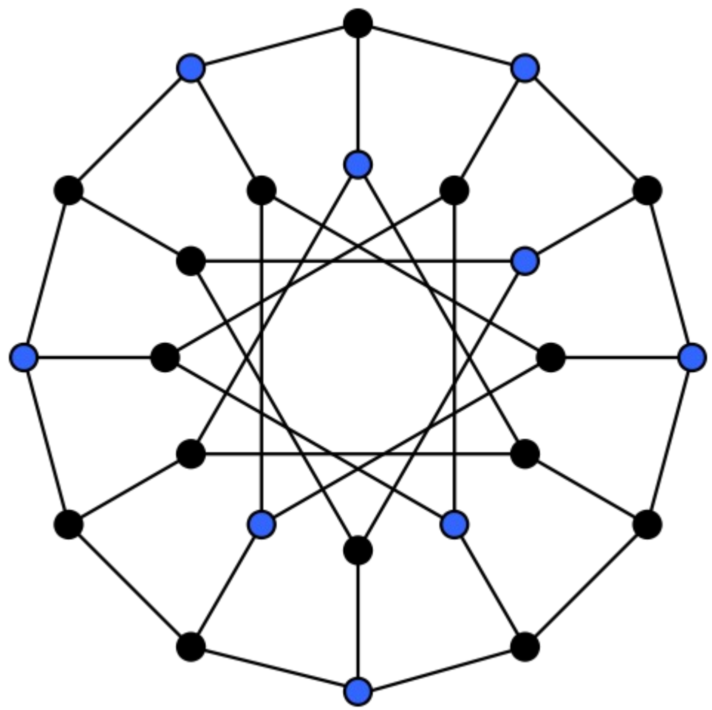
IS Instance. A graph $G=(V, E)$ and a non-negative integer k.

IS Question. Is there a subset I of V such that $|I| \geq k$ and no two vertices in I are adjacent to each other?

A set I is an independent set of G if and only if $V \setminus I$ is a vertex cover of G.

Independent Set (IS) for general graphs

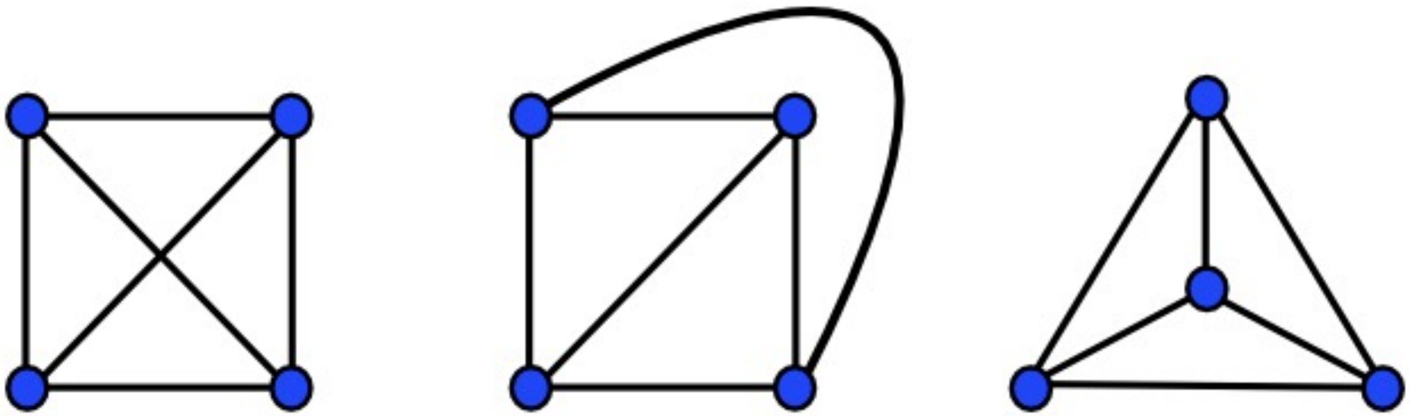
- NP-complete
- intractable from a parameterized point of view (W[1]-complete) with regards to parameter *k* (i.e. size of an independent set)



Independent Set (IS) for planar graphs

- NP-complete
- fixed-parameter tractable in *k*

Planar graphs **can** be drawn in the plane without crossing edges. It can be tested in $O(n)$ if a graph $G=(V,E)$ is planar, where $n=|V|$.



Results on planar graphs

Euler's formula. Let $G=(V,E)$ be a planar graph. Then $|E| \leq 3|V|-6$.

Consequence of Euler's formula. Every planar graph has a vertex of degree at most 5.

Proof. Let $G=(V,E)$ be a planar graph and suppose that every vertex has degree at least 6 (*proof by contradiction*). Then, by the Handshaking Lemma (sum over all vertex degrees is equal to twice the number of edges) $2|E| \geq 6|V|$. Hence $|E| \geq 3|V|$. This gives a contradiction to Euler's formula.

Bounded search for IS on planar graphs

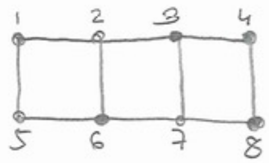
Idea.

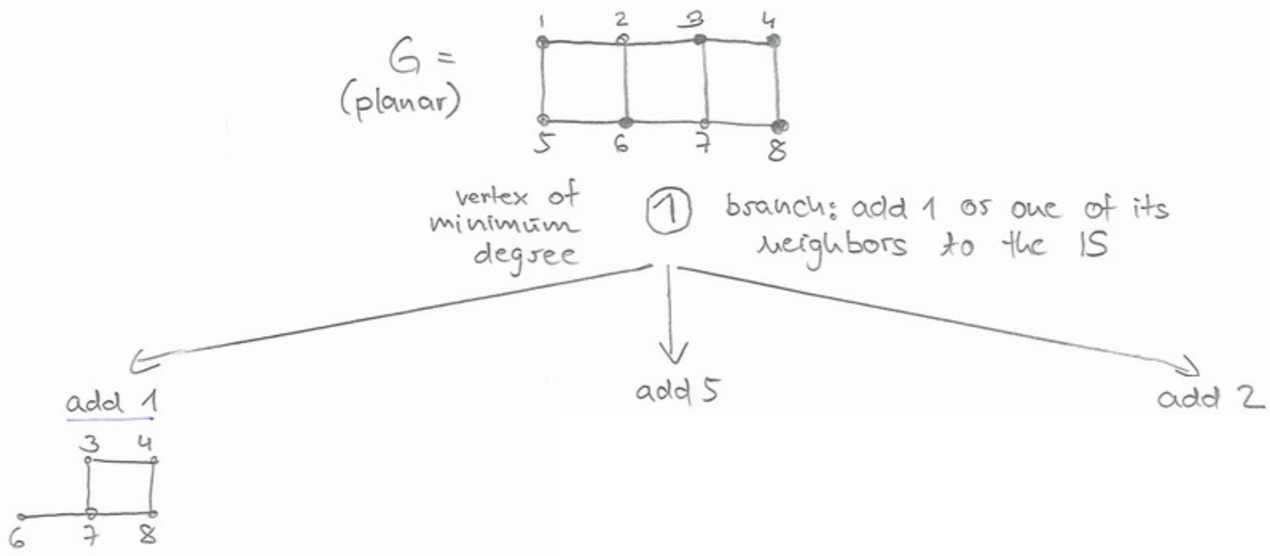
Consider an instance *(G=(V,E),k)* of IS, where *G* is planar.

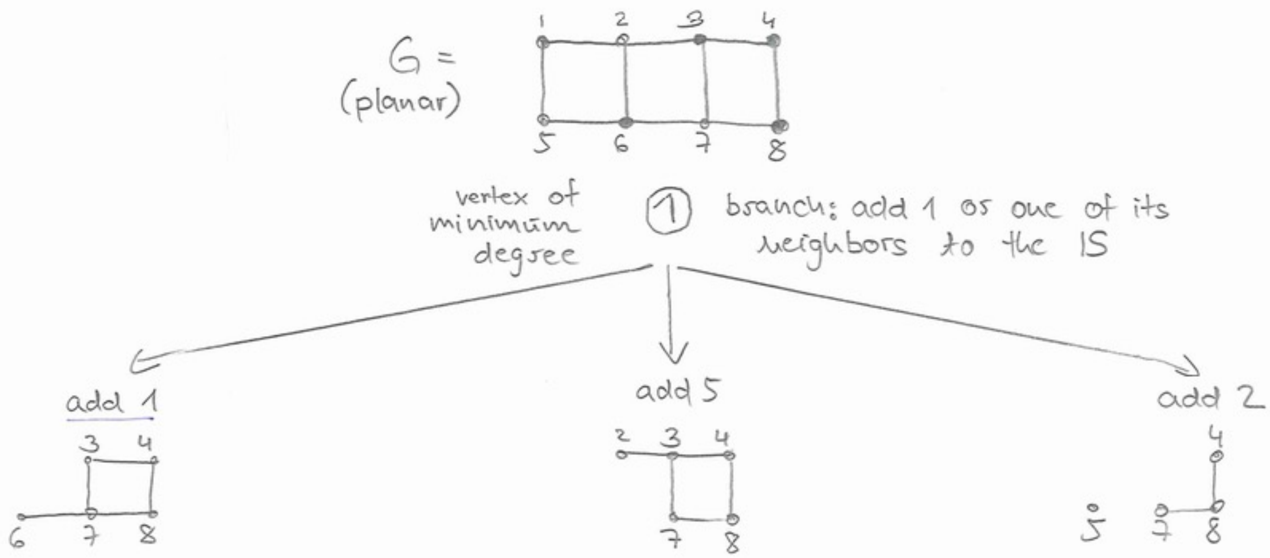
Select a vertex *v* in *G* of minimum degree. We know that *v* has degree at most 5. A **maximum independent set** for *G* contains either *v* or one of its 5 neighbors.

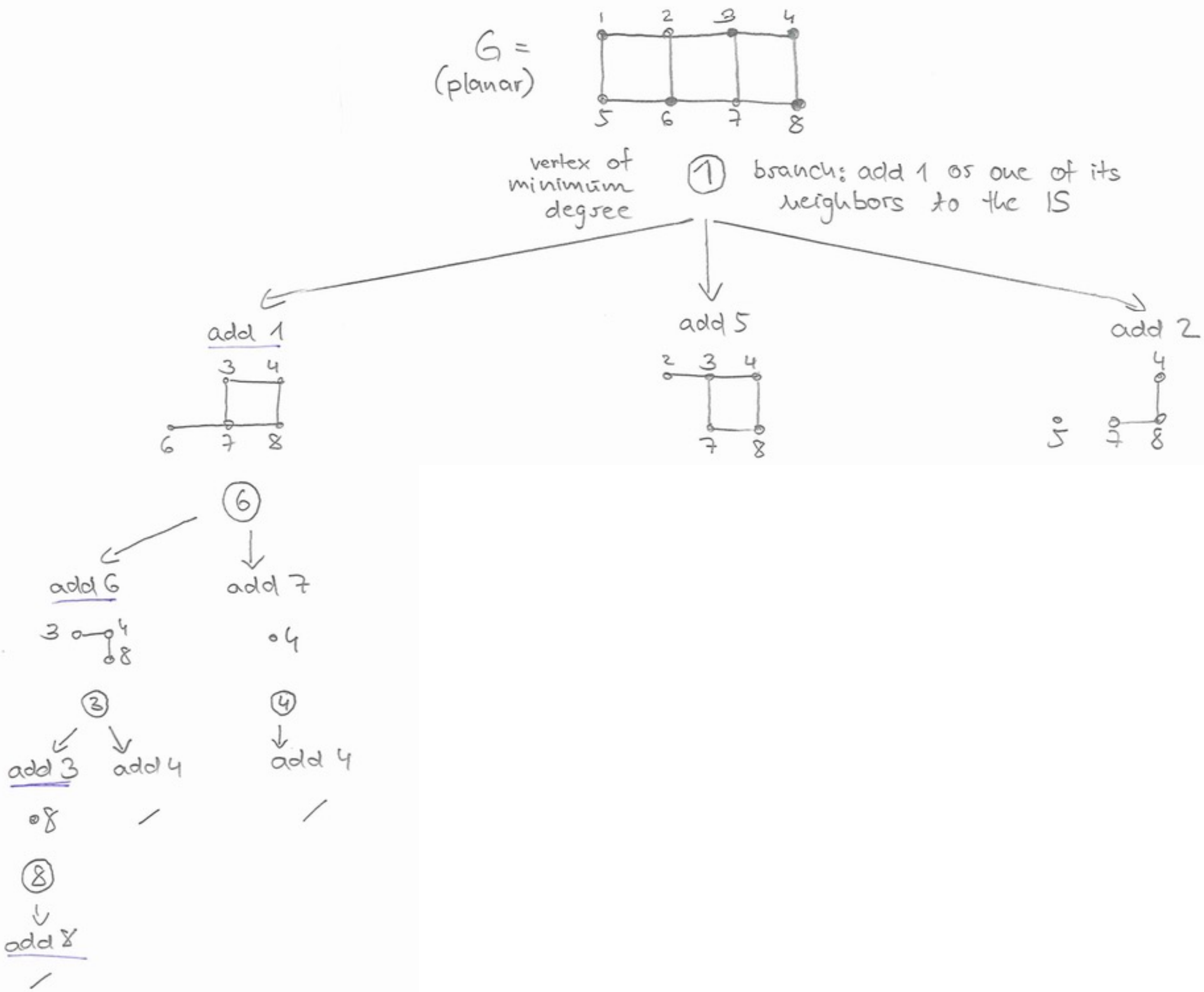
We branch into at most 5+1=6 cases. For each case, delete the corresponding vertex with all its adjacent vertices and edges to obtain a smaller graph *G'*. Recursively solve IS for *(G',k-1).*

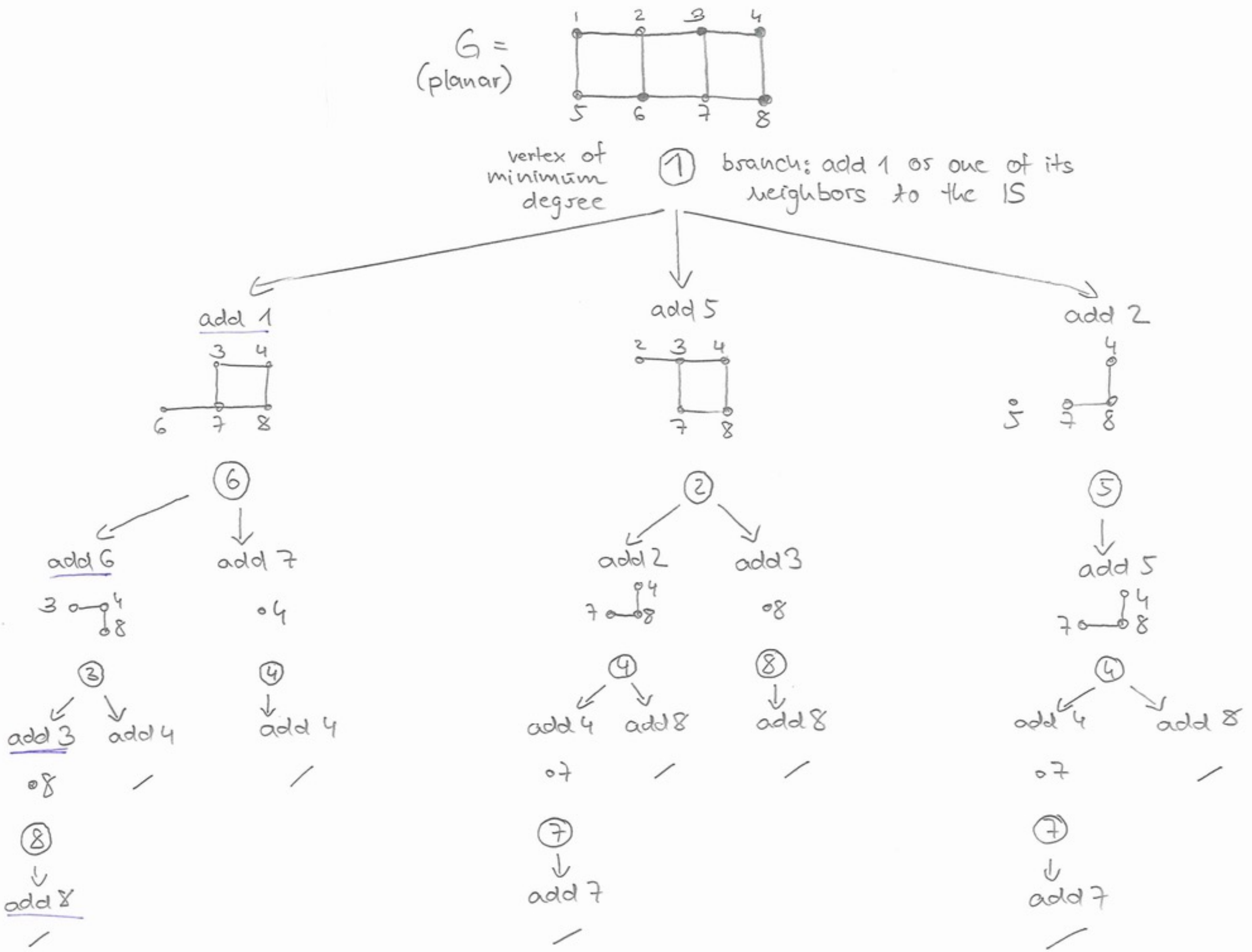
$G =$ (planar)




G =
(planar)
1 2 3 4
5 6 7 8
vertex of minimum degree
1
branch: add 1 or one of its neighbors to the IS
add 1
add 5
add 2
3 4
6 7 8


G =
(planar)
1 2 3 4
5 6 7 8
vertex of minimum degree
1
branch: add 1 or one of its neighbors to the IS
add 1
3 4
6 7 8
add 5
2 3 4
7 8
add 2
4
5 7 8


G =
(planar)
1 2 3 4
5 6 7 8
vertex of minimum degree
1
branch: add 1 or one of its neighbors to the IS
add 1
3 4
6 7 8
add 5
2 3 4
7 8
add 2
4
5 7 8
6
add 6
add 7
3 4
8
4
3
4
add 3
add 4
add 4
8
8
add 8


G =
(planar)
1 2 3 4
5 6 7 8
vertex of minimum degree
1
branch: add 1 or one of its neighbors to the IS
add 1
3 4
6 7 8
add 5
2 3 4
7 8
add 2
4
5 7 8
6
add 6
add 7
3 4
8
4
3
add 3
add 4
8
8
add 8
4
add 4
2
add 2
add 3
4
7 8
8
4
add 4
add 8
7
7
add 7
8
add 8
5
add 5
4
7 8
4
add 4
add 8
7
7
add 7

Bounded search for IS on planar graphs

Theorem 7.

IS on planar graphs has a search tree of size at most $O(6^k)$. Moreover, for a planar graph $G=(V,E)$, a maximum IS can be computed in time $O(6^k n)$, where $n=|V|$.

Correct as from $\{v\} \cup N(v)$ *at least one vertex must be in a maximum independent set of* G.

Closest String (CS)

Instance. A set $\{s_1,s_2,\ldots,s_k\}$ of k strings each of length L and over an alphabet $\sum$; and a non-negative integer d.

Question. Is there a closest string s such that $d_H(s,s_i) \le d$ for all $i \in \{1,2,\ldots,k\}$?

What is $d_H(s,s_i)$?

Hamming distance between the two strings s and s_i.

Has applications in computational biology (e.g. comparison of DNA and protein sequences).

Closest String (CS)

What is $d_H(s,s_i)$?

Hamming distance between the two strings s and s_i.

$d_H(s,s_i)$ is equal to the number of positions $1,2,\ldots,L$ that differ in s and s_i

```
s = 10010110          A = logarithm
s'= 11011011          B = algorithm
```

$d_H(s,s')=4$ $d_H(A,B)=3$

Closest String (CS)

Example 6.

Consider the following four strings:

s_1 = MAUS

s_2 = HAUT

s_3 = RAUS

s_4 = HANS

Does there exist a string s such that $d_H(s,s_i) \leq 1$ for each $i \in \{1,2,3,4\}$?

Closest String (CS)

Example 6.

Consider the following four strings:

s_1 = MAUS

s_2 = HAUT

s_3 = RAUS

s_4 = HANS

Does there exist a string s such that $d_H(s,s_i) \leq 1$ for each $i \in \{1,2,3,4\}$?

s = HAUS

Closest String (CS)

In what follows, we think of the strings $s_1, s_2, \ldots, s_k$ of length L as a character matrix with k rows and L columns.

```
ATCTATAGAAGT
ATCTACAGTAAC
ATCTACAGAAGT
ATCTATAGAGAT
ATCTATAGAAGT
```

Here, $k=5$ and $L=12$.

A column is called dirty if it has at least 2 different symbols from $\Sigma=\{A,C,G,T\}$.

Here, 5 dirty columns.

Closest String (CS)

Lemma 8.

If a matrix has more than kd dirty column, then there is no solution to the associated instance of CS.

Proof. [Notation $i \in \{1,2,\dots,k\}$ and $j \in \{1,2,\dots,L\}$]

Fix a closest string s. For every dirty column j there exists a string s_i such that $s_i[j] \neq s[j]$. Since every string s_i differs from s on at most d positions (if it is a yes-instance), we can have at most kd dirty columns.

What if a matrix has at most kd dirty columns?

Closest String (CS)

Consider an instance of $I=(\{s_1,s_2,\ldots,s_k\}, d)$ of CS.

Lemma 8 gives a simple reduction to a problem kernel: delete all non-dirty column. If there are more than kd columns left, then I is a no-instance. Otherwise, solve CS for the resulting matrix of size at most $O(kd)$.

Kernelization depends on d **and** k.

No kernelization that only depends on d is known.

Next. Bounded-search tree whose size only depends on d.

Closest String (CS)

Lemma 9.

If there exist $i,i' \in \{1,2,\ldots,k\}$ such that $d_H(s_i,s_{i'}) > 2d$, then there is no solution to the associated instance of CS.

Proof.

The Hamming distance is a metric and satisfies the triangle inequality

$$d_H(x,y) \leq d_H(x,z)+d_H(z,y)$$

for arbitrary strings x, y, and z. If $d_H(s_i,s_{i'}) > 2d$, then

$$d_H(s_i,s)+d_H(s,s_{i'}) \geq d_H(s_i,s_{i'}) > 2d$$

for a string s. It follows that $d_H(s_i,s) > d$ or $d_H(s_{i'},s) > d$.

Closest String (CS)

Idea of algorithm. Suppose $I=(\{s_1,s_2,\ldots,s_k\},d)$ of CS is a yes-instance.

Let $\hat{s}$ be a **closest string** (i.e. a solution to a given instance of CS). Select one of the input strings, say s_1, as **candidate string**. As long as there is a string s_i with i $\in\{2,3,\ldots,k\}$ such that $d_H(s_1,s_i)\geq d+1$, then for at least one position p in which s_1 and s_i differ, we have $\hat{s}[p]=s_i[p]$. We can recursively try $d+1$ ways to move the candidate string closer to the closest string by setting $s_1[p]=s_i[p]$.

CS-D.

Kernelize

Check Lemma 8

Call *CSd(s_1,d)*

A recursive call decreases Δd by one

(Niedermeier, 2006, page 105)

Recursive procedure $CSd(s, \Delta d)$:

Global variables: Set of strings $S = \{s_1, s_2, \ldots, s_k\}$, nonnegative integer d.

Input: Candidate string s and integer Δd.

Output: A string $\hat{s}$ with $\max_{i=1,\ldots,k} d_H(\hat{s}, s_i) \leq d$ and $d_H(\hat{s}, s) \leq \Delta d$, if it exists, and "not found," otherwise.

Method:

(D0) **if** $\Delta d < 0$ **then return** "not found";

(D1) **if** $d_H(s, s_i) > d + \Delta d$ for some $i \in \{1, \ldots, k\}$ **then return** "not found";

(D2) **if** $d_H(s, s_i) \leq d$ for all $i = 1, \ldots, k$ **then return** s;

(D3) choose any $i \in \{1, \ldots, k\}$ such that $d_H(s, s_i) > d$:

  $P := \{ p \mid s[p] \neq s_i[p] \}$;

  choose any $P' \subseteq P$ with $|P'| = d + 1$;

  for all $p \in P'$ **do**

    $s' := s$;

    $s'[p] := s_i[p]$;

    $s_{ret} := CSd(s', \Delta d - 1)$;

    if $s_{ret} \neq$ "not found" **then return** s_{ret};

(D4) **return** "not found"

FIG. 8.4. **Algorithm CS-D.** Inputs are a CLOSEST STRING instance consisting of a set of strings $S = \{s_1, s_2, \ldots, s_k\}$ of length L each, and a nonnegative integer d. The recursion is invoked with $\boxed{CSd(s_1, d)}$. Instead of s_1, we could choose an arbitrary element from S here.

Closest String (CS)

Search tree size.

Consider the recursive part of the algorithm. Each call decreases Δd by one. The algorithm stops when $\Delta d < 0$. Hence, the height of the search tree is at most d.

In each recursive call, the algorithm explores $d+1$ subcases (branchings) for positions at which s and s_i disagree.

An upper bound on the size of the search tree is $O((d+1)^d)=O(d^d)$.

Closest String (CS)

Correctness.

To establish correctness of CS-D, we need to show that the algorithm always finds a string $\hat{s}$ such that $\max_{i=1,2,\ldots,k} d_H(\hat{s}, s_i) \leq d$ if one exists.

We show correctness of the first recursive call (that is *CSd*(s_1, *d*)). Correctness for CS-D then follows by induction.

Closest String (CS)

Correctness (cont.).

If $\max_{i=1,2,\ldots,k} d_H(s_1,s_i) \leq d$, then we already have a solution, namely s_1. If s_1 is not a solution but there exists a closest string with distance at most d to all input strings, then there is a string s_i with $i \in \{2,3,\ldots,k\}$ such that $d_H(s_1,s_i) > d$. [See 1st line of (D3)]

Now consider all positions $1,2,\ldots,L$ where s_1 and s_i differ, i.e.

$$P=\{p \mid s_1[p] \neq s_i[p]\}.$$

By Lemma 9, we may assume that $d+1 \leq |P| \leq 2d$. [See 2nd line of (D3)]

Closest String (CS)

Correctness (cont.).

The algorithm successfully creates $d+1$ branchings (choose $d+1$ elements in P). Each branching creates a new candidate string s' that can be obtained from s_1 by replacing $s_1[p]$ with $s_i[p]$. [See 6th line of (D3)]

Question coming up: Why is this bringing us any closer to a solution?

Closest String (CS)

Correctness (cont.).

Suppose that $\hat{s}$ is a closest string. Hence $\max_{i=1,2,\ldots,k} d_H(\hat{s}, s_i) \leq d$. A change as described on the previous slide is correct if we choose a position p from $P \supset \boxed{P_1 = \{p \mid s_1[p] \neq s_i[p] \text{ and } s_i[p] = \hat{s}[p]\}.}$ [Good guys!]

We next show that at least one of the $d+1$ branchings is correct.

Observe that $P = P_1 \cup P_2$, where $\boxed{P_2 = \{p \mid s_1[p] \neq s_i[p] \text{ and } s_i[p] \neq \hat{s}[p]\}.}$ [Bad guys!]

Since $d_H(\hat{s}, s_i) \leq d$, we can conclude that $|P_2| \leq d$. Hence, at least one of the $d+1$ branchings tries a position that is in P_1.

Thus, $d+1$ branchings are sufficient to change s_1 into a new string s' that is "closer to $\hat{s}$".

Closest String (CS)

We now combine everything to get the following result.

Theorem 10.

CS can be solved in $O(kL\, d^d)$. Hence, CS is fixed-parameter tractable with parameter d.

kL	size of matrix (without kernelization)
d^d	size of bounded search tree

Closest String (CS)

It is an open problem whether or not CS is fixed-parameter tractable when parameterized by k using a bounded-search tree approach.

Closest String (CS) – additional slide

Theorem.

There exists a closest string for $\{s_1,s_2,\ldots,s_k\}$ with distance at most d to each input string if and only if the algorithm returns string $\hat{s}$.

Proof.

Suppose $\hat{s}$ exists. Then the previous correctness proof shows that the algorithm finds $\hat{s}$ because at least one of the $d+1$ branchings brings us closer to $\hat{s}$ in each iteration.

Suppose $\hat{s}$ does not exist. Pick any string s. Then there exists a string s_i such that $d_H(s,s_i)>d$. But the algorithm stops after d iterations and correctly returns “not found”.