Parameterized Algorithm

Input: $G = (V, E), k$

Parameter: k

Output: A vertex cover $VC(G, k)$ of size k or smaller, if it exists.

if $E = \emptyset$ **then return** $\emptyset$
if $k = 0$ **the**
Choose som
$G_1 := (V -$
$G_2 := (V -$
if $|\{v_1\} \cup V$
then return
else return

Questions:

1. What does "no solution" mean?
2. Why is the running time $O(f(k)n^c)$?
3. What exactly is $f(k)$?
4. Do we always find an optimal vertex cover?
5. Can we simplify the last lines of the algorithm?

Parameterized Algorithm

Input: $G = (V, E), k$

Output: A vertex cover $VC(G, k)$ of size k or smaller, if it exists.

if $E = \emptyset$ **then return** $\emptyset$
if $k = 0$ **then return** "no solution"
Choose some edge $\{v_1, v_2\} \in E$
$G_1 := (V - \{v_1\}, \{ e \in E \mid v_1 \notin e \})$
$G_2 := (V - \{v_2\}, \{ e \in E \mid v_2 \notin e \})$
if $VC(G_1, k - 1) \neq$ "no solution"
then return $\{v_1\} \cup VC(G_1, k - 1)$
else return $\{v_2\} \cup VC(G_2, k - 1)$

Parameterized Algorithm — Running Time

Every recursive call requires only polynomial time.

How many recursive calls are there?

Every incarnation is a leaf in the recursion tree or has two children.

- The root has parameter k
- The parameter of a child is at least one smaller compared to the parent
- The parameter never becomes negative

Therefore the height of the recursion tree is at most k

Its size is then at most 2^k.

The Long Road to Vertex Cover

- Fellows & Langston (1986): $O(f(k)n^3)$
- Robson (1986): $O(1.211^n)$
- Johnson (1987): $O(f(k)n^2)$
- Fellows (1988): $O(2^k n)$
- Buss (1989): $O(kn + 2^k k^{2k+2})$
- Downey, Fellows, & Raman (1992): $O(kn + 2^k k^2)$
- Balasubramanian, Fellows, & Raman (1996): $O(kn + 1.3333^k k^2)$
- Balasubramanian, Fellows, & Raman (1998): $O(kn + 1.32472^k k^2)$

The Long Road to Vertex Cover

- Downey, Fellows, Stege (1998): $O(kn + 1.31951^k k^2)$
- Niedermeier & R. (1998): $O(kn + 1.292^k)$
- Chen, Kanj, & Jia (1999): $O(kn + 1.271^k k^2)$
- Chen, Kanj, & Jia (2001): $O(kn + 1.285^k)$
- Niedermeier & R. (2001): $O(kn + 1.283^k)$
- Chandran & Grandoni (2004): $O(kn + 1.275^k k^{1.5})$
- Chen, Kanj, & Xia (2005): $O(kn + 1.274^k)$

Bounded Search Trees

A Bounded search tree algorithm must fulfil these condition on its recursion tree:

- Every node is labeled by some natural number
- The root is labeled by some function of the parameter
- The number of children of a node is limited by some function of the parent's label
- Children are labeled by smaller numbers than the parent

Correctness

Theorem

Let an algorithm be a bounded search tree algorithm.
Then there is a function f, such that every search tree for an input with parameter k has at most $f(k)$ many nodes.

Proof of Correctness

Proof

We define a function $S(k)$ that is an upper bound on the number of leaves in a subtree whose root is labeled by k.

- Assume that the root is labeled with at most $w(k)$
- Assume that every node with label k hat at most $b(k)$ many children
- The existence of w and b is guaranteed by the definition of bounded search trees.

Proof of Correctness (cont.)

Proof

$$S(k) \leq b(k)S(k-1),$$

because there are at most $b(k)$ children whose subtrees have each at most $S(k-1)$ many leaves.
With $S(0) = 1$ the solution of this recurrence is

$$S(k) \leq \prod_{i=1}^{k} b(i).$$

The total number of leaves consequently is at most $S(w(k))$.

Example Closest String

Let u and v be two strings of length n.

We define $h(u, v)$, called Hamming distance of u and v, as the number of positions on which u and v differ.

Example:

$$h(agctcagtaccc, agctcataacgc) = 3$$

Example Closest String

The Closest string problem is defined as follows:

Input: k strings $s_1, \ldots, s_k$ of length n, a number m

Question: Is there a string s with $h(s, s_i) \leq m$ for all $1 \leq i \leq k$?

The parameter is m

Motivation: Construct a chemical marker that closely fits to a set of DNA sequences

In practice m is small, e.g. 5

Example Closest String

```
agcacagtacgcaatagtgtcgcaggt
agctcagtagccaatagagtcccaggt
agatcagttcccaatagagtcgcacgt
agctcagtaaaaaatagagtcgcaggt
agcgcagtacacaatagagtcgcaagt
```

Example Closest String

agcacagtacgcaatagtgtcgcaggt
agctcagtagccaatagagtcccaggt
agatcagttcccaatagagtcgcacgt
agctcagtaaaaaatagagtcgcaggt
agcgcagtacacaatagagtcgcaagt

---

agctcagtacccaatagagtcgcaggt

Example Closest String

```
gctaggagtcagaagtaggcgttgcat
gcaatgaatcagaactgggcctagcat
gctagggatcagaactaggcctagcat
gcaaggaatcataactaggcctagcat
gcaaggaattagaaataggcctagcat
gcaagaaatcagaactagccctagcat
```

Example Closest String

gctaggagtcagaagtaggcgttgcat
gcaatgaatcagaactgggcctagcat
gctagggatcagaactaggcctagcat
gcaaggaatcataactaggcctagcat
gcaaggaattagaaataggcctagcat
gcaagaaatcagaactagccctagcat

---

gcaaggagtcagaactaggcctagcat

An Algorithm for Closest String

Input: Strings $s_1, \ldots, s_k$, a number m.

Algorithm center(s, l) finds out, if there is an s', such that

- $h(s, s') \leq l$
- $h(s', s_i) \leq m$ for $1 \leq i \leq k$

With center we can easily solve the closest string problem:

Just call center(s_1, m)!

An Algorithm for Closest String

We can implement center(s, l) as follows:

Choose some string s_i with $h(s, s_i) > m$.

(If no such string exists, then s is a solution and we answer Yes.)

Choose a set P of $m + 1$ positions, where s and s_i differ.

Try all positions $p \in P$. Each time let s' be the same as s except for position p, where s' coincides with s_i.

Each time call center$(s', l - 1)$. If one of them answers Yes, then answer Yes.

An Algorithm for Closest String

The size of the search tree is at most $(m+1)^m$.

- The root is labeled with m
- Children are labeled with smaller numbers than the parent
- If the label is 0, we find a solution in polynomial time.
- Every node has at most $m+1$ children.

This algorithm is efficient and works well in practice.

An Algorithm for Closest String

The size of t

It has been known for a long time that this problem is *fixed parameter tractable*, if *both* k and m are parameters.

- The root is labeled with m
- Children are labeled with smaller numbers than the parent
- If the label is 0, we find a solution in polynomial time.
- Every node has at most $m + 1$ children.

This algorithm is efficient and works well in practice.

An Algorithm for Closest String

The size of t

It has been known for a long time that this problem is *fixed parameter tractable*, if *both* k and m are parameters.

- The root is labeled with m
- Children are labeled with smaller numbers than the parent
- If
- E

This a

For applications m is the crucial parameter.
Nevertheless, it is also interesting to consider the parameter k.
Question: Is Closest String fixed parameter tractable, if k is the parameter?
(Both questions, for k and m, were open for a long time.)

Analysis of Bounded Search Tree Algorithms

If

1. the root of a tree is labeled with k,
2. every node has at most two children,
3. no label is negative,
4. children are labeled with smaller numbers than the parent,

then it is quite clear that the tree has at most 2^k many leaves.

How can we generalize this obvious fact?

Branching vectors

If every inner node has two children and their labels are exactly one smaller, we get the recurrence relation

$$B_k = B_{k-1} + B_{k-1}.$$

The corresponding branching vector is $(1, 1)$.

A recurrence

$$B_k = B_{k-z_1} + B_{k-z_2} + \cdots + B_{k-z_m}$$

corresponds to the branching vector $(z_1, \ldots, z_m)$.

We can succinctly describe bounded search trees with branching vectors.

Branching Vectors

If the two branching vectors $(1, 1)$ and $(2, 2, 3)$ occur in a bounded search tree algorithms, we get the recurrence

$$B_k = \max\{2B_{k-1}, 2B_{k-2} + B_{k-3}\}.$$

We would like to analyse bounded search tree algorithms with multiple branching vectors. For this end we have to solve recurrences as above.

Linear Recurrence Equations with Constant Coefficients

For a branching vector the corresponding recurrence is a linear recurrence equation with constant coefficients.

Its general form is

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_t a_{n-t} \text{ for } n \geq t.$$

We develop a simple method to solve such recurrence equations.

Linear Recurrence Equations with Constant Coefficients

Let us assume there is a solution of the form $a_n = \alpha^n$, where $\alpha \in \mathbf{C}$ can be a complex number. If we insert this solution into the recurrence and set $n = t$, we get

$$\alpha^t = c_1\alpha^{t-1} + c_2\alpha^{t-2} + \cdots + c_{t-1}\alpha + c_t$$

meaning that α is a root of the *characteristic polynomial*

$$\chi(z) = z^t - c_1 z^{t-1} - c_2 z^{t-2} - \cdots - c_{t-1}z - c_t.$$

Linear Recurrence Equations with Constant Coefficients

On the other hand, $a_n = \alpha^n$ is a solution of the recurrence, if α is a root of

$$\chi(z) = z^t - c_1 z^{t-1} - c_2 z^{t-2} - \cdots - c_{t-1} z - c_t .$$

This is easy to see if we insert it into the recurrence:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_t a_{n-t}$$

Linear Recurrence Equations with Constant Coefficients

If α is a k-fold root of χ, then $a_n = n^j \alpha^n$ for $0 \leq j < k$ are also solutions of the recurrence. We can check this again by inserting it into the recurrence:

$$n^j \alpha^n = \sum_{r=1}^{t} c_r (n-r)^j \alpha^{n-r} \text{ resp. } n^j \alpha^t - \sum_{r=1}^{t} c_r (n-r)^j \alpha^{t-r} = 0.$$

The left hand side is a linear combination of $\chi(\alpha)$, $\chi'(\alpha)$, $\chi''(\alpha)$, $\ldots$, $\chi^{(j)}(\alpha)$. The first $k-1$ derivatives of χ become 0 at α because α ist a k-fold root of χ.

Linear Recurrence Equations with Constant Coefficients

Theorem

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_t a_{n-t} \; for \, n \geq t$$

has the solutions $a_n = n^j \alpha^n$*, for every root* α *of the characteristic polynomial*

$$\chi(z) = z^t - c_1 z^{t-1} - c_2 z^{t-2} - \cdots - c_{t-1} z - c_t,$$

and for all $j = 0, 1, \ldots, k-1$*, where* k *is the order of the root* α*. All these solutions are linearly independent. They form a base of the vector space of solutions.*

The Size of Search Trees

Theorem

A bounded search tree with branching vector$(r_1, \ldots, r_m)$, whose root is labeled with k, has size

$$k^{O(1)} \alpha^k,$$

where α is the root with biggest absolute value of the characteristic polynomial

$$\chi(z) = z^t - z^{t-r_1} - z^{t-2} - \cdots - z^{t-r_m},$$

where $t = \max\{r_1, \ldots, r_m\}$.

The Size of Search Trees

Example:

The branching vector $(1, 3)$ has the characteristic polynomial

$$z^3 - z^2 - 1.$$

The largest root is approximately 1.465571.

The size of search tree is $O(1.465572^k)$.

The Size of Bounded Search Trees

Another example:

The branching vector $(1, 2, 2, 3, 6)$ has the characteristic polynomial

$$z^6 - z^5 - z^4 - z^4 - z^3 - 1.$$

The largest real root is 2.160912.

The size of the search tree is therefore $O(2.160913^k)$.

The Reflected Characteristic Polynomial

To determine the characteristic polynomial

$$z^6 - z^5 - z^4 - z^4 - z^3 - 1$$

from the branching vector

$$(1, 2, 2, 3, 6)$$

is not easy and error-prone.

The reflected characteristic polynomial is

$$1 - z - z^2 - z^2 - z^3 - z^6.$$

The Reflected Characteristic Polynomial

Theorem

The characteristic polynomial has a root α iff the reflected characteristic polynomial has the root $1/\alpha$.

The Reflected Characteristic Polynomial

Theorem

A search tree with branching vector $(r_1, \ldots, r_m)$, whose root is labeled with k, has the size

$$k^{O(1)}\alpha^{-k},$$

where α is the root with minimum absolute value of the reflected characteristic polynomial

$$\chi(z) = 1 - z^{r_1} - z^{r_2} - \cdots - z^{r_m}.$$

Branching Numbers

Definition

For each branching vector there is a corresponding branching number which is the reciprocal of the smallest root of the characteristic polynomial.

Theorem

A search tree with branching number α whose root is labeled k has size

$$k^{O(1)}\alpha^k.$$

If the root is simple then the size is $O(\alpha^k)$.