Fixed-Parameter Algorithms, IA166

Sebastian Ordyniak

Faculty of Informatics
Masaryk University Brno

Spring Semester 2013

Bounded Search Tree

Outline

Definitions – digraphs

- A directed graph or digraph is a tuple $G = (V, A)$ such that A is a set of pairs (u, v) with $u, v \in V$;
- Elements of V and A are called the vertices and arcs of G, respectively;
- For a digraph G, $V(G)$ denotes its vertex set and $A(G)$ its arc set.
- If $(u, v) \in A(G)$, then v is an out-neighbor of u and u is an in-neighbor of v;
- The out-degree(in-degree) of a vertex u is the number of out-neighbors (in-neighbors) of u.

Definitions – Paths, cuts, reachability

Let G be a digraph, $S \subseteq V(G)$, and $T \subseteq V(G)$ such that S and T are disjoint.

- An (S, T)-path is a sequence of distinct vertices $v_0, \ldots, v_l$ such that $v_0 \in S$, $v_l \in T$, and $(v_i, v_{i+1}) \in A(G)$ for all $0 \leq i < l$;
- T is reachable from S if G contains an (S, T)-path;
- A set $C \subseteq V(G) \setminus (S \cup T)$ is an (S, T)-cut if T is not reachable from S in $G \setminus C$;
- $\mathrm{MinCut}_G(S, T)$ denotes the size of a minimum (S, T)-cut in G where $\mathrm{MinCut}_G(S, T) = \infty$ if no such cut exists.

Theorem

$\mathrm{MinCut}_G(S, T)$ can be computed in polynomial time for all $S, T \subseteq V(G)$ (e.g., using Flows).

MINIMUM SKEW SEPARATOR (k-MSS) **Parameter:** k

Input: Digraph G, vertex sequences $S = (s_1, \ldots, s_l)$ and $T = (t_1, \ldots, t_l)$ where all s_i have in-degree 0 and all t_i have out-degree 0, and an integer k.
Question: Does G have a skew-separator (SS) C of size at most k, i.e., is there a $C \subseteq V(G) \setminus (S \cup T)$ such that $|C| \leq k$ and for all $j \leq i$, t_j is not reachable from s_i in $G \setminus C$?

Why is this problem interesting?

- It will demonstrate another non-trivial technique of bounding the search tree size.
- The algorithm for this problem is part of the recent FPT algorithm for directed feedback vertex set (Major Breakthrough!).

Reduction rules and easy cases

Let $\mathcal{I} = (G, S, T, k)$ be a k-MSS instance where $S = (s_1, \ldots, s_l)$ and $T = (t_1, \ldots, t_l)$.

Observation (Rule 1)

If there is an arc from s_l to some t_j then $\mathcal{I}$ has no SS.

Observation (Rule 2)

If T is not reachable from s_l and $l \geq 2$ then $(G \setminus \{s_l, t_l\}, S \setminus \{s_l\}, T \setminus \{t_l\})$ has a k-MSS iff $\mathcal{I}$ has a k-MSS.

Observation (Rule 3)

If s_l has an out-neighbor u that has out-neighbors in T then $G \setminus \{u\}$ has a $(k-1)$-MSS iff $\mathcal{I}$ has a k-MSS.

Reduction rules and easy cases

Let $\mathcal{I} = (G, S, T, k)$ be a k-MSS instance where $S = (s_1, \ldots, s_l)$ and $T = (t_1, \ldots, t_l)$.

Observation (Rule 4)

If $\text{MinCut}_G(S, T) > k$ then $\mathcal{I}$ has no k-MSS.

Observation (Rule 5)

If $l = 1$ then G has a k-MSS iff $\text{MinCut}_G(S, T) \leq k$.

Observation

Rules 1–5 can be applied in polynomial time and at most $|V(G)|$ times.

A less trivial rule

Definition

For an out-neighbor u of s_l we denote by $G/(s_l, u)$ the graph obtained from G by contracting (s_l, u) into s_l and subsequently removing all arcs coming into s_l.

Theorem (Rule 6)

If s_l has an out-neighbor u such that $\text{MinCut}_G(s_l, T) = \text{MinCut}_G(\{s_l, u\}, T)$ then G has a k-MSS iff $G/(s_l, u)$ has a k-MSS.

A less trivial rule

The correctness of Rule 6 follows from the following proposition and theorem.

Proposition

Let u be an out-neighbor of s_l. Then G has k-MSS that does not contain u iff $G/(s_l, u)$ has a k-MSS.

Theorem 1

If s_l has an out-neighbor u such that $\text{MinCut}_G(s_l, T) = \text{MinCut}_G(\{s_l, u\}, T)$ then G has a MSS that does not contain u.

A less trivial rule

The correctness of Rule 6 follows from the following proposition and theorem.

Proposition

Let u be an out-neighbor of s_l. Then G has k-MSS that does not contain u iff $G/(s_l, u)$ has a k-MSS.

Proof:

Let C be a MSS that does not contain u in G then C is also a MSS in $G/(s_l, u)$. For the reverse direction suppose C is a MSS for $G/(s_l, u)$. Then C is also a MSS for G that does not contain u. □

Proof of Theorem 1

Proof:

Let Y be a minimum $(\{s_I, u\}, T)$-cut in G which is a minimum (s_I, T)-cut in G that does not contain u. Let X be a minimum MSS for I. If $u \notin X$ we are done, so assume $u \in X$. We define:

- $Z = X \cap Y$;
- Y_B: the vertices in $Y \setminus X$ that cannot reach T in $G \setminus X$;
- Y_F: the vertices in $Y \setminus X$ that can reach T in $G \setminus X$;
- X_B: the vertices in $X \setminus Y$ that cannot reach T in $G \setminus Y$;
- X_F: the vertices in $X \setminus Y$ that can reach T in $G \setminus Y$;

Hence: $X = X_B \cup Z \cup X_F$ and $Y = Y_B \cup Z \cup Y_F$.

Proof of Theorem 1, continued

We need the following claims:

Claim 1

$Y' = Y_B \cup Z \cup X_B$ is an (s_I, T)-cut in G.

Claim 2

$X' = X_F \cup Z \cup Y_F$ is a MSS for I.

Proof of Theorem 1, continued

Assuming Claim 1 and 2 hold, we proof Theorem 1 as follows:

Proof (Theorem 1):

- Y is a minimum (s_l, T)-cut, hence $|Y| \leq |Y'|$ and $|Y_B| + |Z| + |Y_F| \leq |Y_B| + |Z| + |X_B|$ and consequently $|Y_F| \leq |X_B|$;
- Therefore, $|X'| = |X_F| + |Z| + |Y_F| \leq |X_F| + |Z| + |X_B| = |X|$, hence X' is a minimum MSS;
- $u \notin X_F$ because otherwise T would be reachable from u in $G \setminus Y$ and hence from s_l, but Y is an (s_l, T)-cut.
- $u \notin Z \cup Y_F \subseteq Y$ be the choice of Y and so $u \notin X'$.

□

Proof of Claim 1

Proof of Claim 1 ($Y' = Y_B \cup Z \cup X_B$ is an (s_l, T)-cut in G):

Suppose not and let P be an (s_l, T)-path in $G \setminus Y'$. Let w be the **first** vertex of P in $X_F \cup Y_F$. Because Y and X are (s_l, T)-cuts, $Y \setminus Y' = Y_F$, and $X \setminus Y' = X_F$ such a vertex must exists.

If $w \in X_F$ then $P' = P_{s_l,w}$ is a path in $G \setminus Y$ and by the definition of X_F there is a path P'' from w to T in $G \setminus Y$. Hence, $P' \circ P''$ is an (s_l, T)-path in $G \setminus Y$, a contradiction.

If $w \in Y_F$ then $P' = P_{s_l,w}$ is a path in $G \setminus X$ and by the definition of Y_F there is a path P'' from w to T in $G \setminus X$. Hence, $P' \circ P''$ is an (s_l, T)-path in $G \setminus X$, a contradiction. □

Proof of Claim 2

Proof of Claim 2 ($X' = X_F \cup Z \cup X_F$ is a MSS for I):

Suppose not and let P be an (s_I, T)-path in $G \setminus X'$. Because X is a MSS and $X \setminus X' = X_B$ the path P contains a vertex in X_B. Let w be the **last** vertex of P in X_B and $P' = P_{w,t_j}$.

Because P' is not a path in $G \setminus Y$ (by the definition of X_B and $Y \setminus X' = Y_B$, P' contains a vertex $x \in Y_B$. Let $P'' = P'_{x,t_j}$ (because $x \neq w$ P'' is strictly shorter than P').

Because P'' is not a path in $G \setminus X$ (by the definition of Y_B) and $X \setminus X' = X_B$, P'' contains a vertex $y \in X_B$. This contradicts the choice of w. □

A branching rule!?

Theorem (Rule 6)

If s_l has an out-neighbor u such that $\text{MinCut}_G(s_l, T) = \text{MinCut}_G(\{s_l, u\}, T)$ then G has a k-MSS iff $G/(s_l, u)$ has a k-MSS.

Branching Rule

If s_l has an out-neighbor u, then G has a k-MSS iff either $G \setminus \{u\}$ has a $(k-1)$-MSS or $G/(s_l, w)$ has a k-MSS.

A branching algorithm?!

Let $I = (G, S, T, k)$ be a k-MSS instance where $S = (s_1, \ldots, s_l)$ and $T = (t_1, \ldots, t_l)$.

Step 1) Apply Rules 1–6 until an irreducible instance is obtained, or the correct answer is returned (In this case output the correct answer). Denote the reduced instance again by $I = (G, S, T, k)$.

Step 2) Consider an out-neighbor u of s_l.
If $G \setminus \{u\}$ has a $(k-1)$-MSS or $G/(s_l, u)$ has a k-MSS then
  return YES
else
  Return NO

A branching algorithm?!

- The algorithm is correct since we have proved the correctness of all reduction rules and the single branching rule;
- Step 1) as well as the construction of the new graphs takes polynomial time.

Problem

Only the number of vertices but not the parameter k decreases in the second branch $(G/(s_l, u))$! This only yields a bound of $2^{O(|V(G)|)}$!

How to analyze the algorithm properly?

Analysis of the algorithm

Proposition (1)

Let G be an irreducible graph and u an out-neighbor of s_l. Then $\text{MinCut}_G(s_l, T) < \text{MinCut}_{G/(s_l,u)}(s_l, T)$.

Proof:

An (s_l, T)-cut in $G/(s_l, u)$ is an (s_l, T)-cut in G that does not contain u, but every minimum (s_l, T)-cut in G contains u.

Idea

Therefore, after choosing the second branch at least k times, $\text{MinCut}(s_l, T) > k$ and NO may be returned.

How to turn this Idea into a proper proof?

Analysis of the algorithm

Idea:

Take $2k - m$ instead of just k as the parameter where $m = m(G, S, T) = \text{MinCut}_G(s_l, T)$. Then do induction over $2k - m$ as normal.

Proposition (2)

For an irreducible instance $\mathcal{I} = (G, S, T, k)$ both branching instances $\mathcal{I}_i = (G_i, S, T, k_i)$ have $2k_i - m_i < 2k - m$ where $m_i = m(G_i, S, T)$.

Analysis of the algorithm

Proposition (2)

For an irreducible instance $\mathcal{I} = (G, S, T, k)$ both branching instances $\mathcal{I}_i = (G_i, S, T, k_i)$ have $2k_i - m_i < 2k - m$ where $m_i = m(G_i, S, T)$.

Proof:

In the first case $G_1 = G \setminus \{u\}$ and $k_1 = k - 1$. Furthermore, by deleting u from G MinCut(s_l, T) decreases by at most 1. Hence, $2k_1 - m_1 \leq 2(k-1) - (m-1) = 2k - m - 1 < 2k - m$.

In the second case $G_2 = G/(s_l, u)$ and $k_2 = k$ and by Proposition (1) $m(G, S, T) < m(G_2, S, T)$. Hence, $2k_2 - m_2 \leq 2k - (m+1) < 2k - m$. □

Analysis of the algorithm

Proposition (3)

For every reduction rule (1–6) that reduces (G, S, T, k) to (G', S', T', k') it holds that $2k' - m' \leq 2k - m$.

Proof:

Rules (1), (4) and (5) terminate and need not be considered.

Rule (2) (delete $\{s_l, t_l\}$) only applies if $m = 0$ and does not change k.

Rule (3) (delete a vertex that must be in the MSS) decreases k by 1 and m by at most 1.

Rule (6) (contract an arc out of s_l) does not change k and m by definition. □

Analysis of the algorithm

Theorem

The branching algorithm for k-MSS gives a search tree with at most 4^k leaves.

Proof:

We show that the search tree has at most 2^{2k-m} leaves using induction over $2k - m$.

IB: $2k - m \leq 0$. We show that in this case one of the reduction rules (1)–(6) apply. Because these rules do not increase $2k - m$ (Proposition 3) this shows that the algorithm terminates without branching!

Case 1. ($k > 0$) Then $m \geq 2k > k$. Hence, rule (4) applies.

Case 2. ($m = 0$ and $l \geq 2$) Then rule (2) applies.

Case 3. ($m = 0$ and $l = 1$) Then rule (5) applies.

Analysis of the algorithm

Theorem

The branching algorithm for k-MSS gives a search tree with at most 4^k leaves.

Proof:

IS: $2k - m > 0$.
Because the reduction rules do not increase $2k - m$ it suffices to proof the statement for irreducible G. By Proposition (2) both branching instances G_1 and G_2 have parameter strictly smaller than $2k - m$ and so by induction the number of leaves is at most:

$$2^{2k-m-1} + 2^{2k-m-1} = 2^{2k-m}$$

Outline

Definitions

Let G be an undirected graph and k a natural number.

k-Coloring

A proper k-vertex coloring or k-coloring of G is a function $\alpha : V(G) \to \{1, \ldots, k\}$ such that for all $\{u, v\} \in E(G)$ it holds that $\alpha(u) \neq \alpha(v)$.

Chromatic Number

The chromatic number ($\chi(G)$) of G is the minimum number k such that G admits a k-coloring.

k-Colorability

Input: An undirected graph G and a natural number k.
Question: Is $\chi(G) \leq k$?

Definitions

k-Colorability

Input: An undirected graph G and a natural number k.
Question: Is $\chi(G) \leq k$?

Proposition

k-Colorability is NP-hard even for $k = 3$.

Hence, choosing k as a parameter is hopeless, however, there are some graph classes for which the problem is easy, i.e., solvable in polynomial time!

Chordal Graphs

Definition

A graph G is chordal if the vertices can be ordered $v_1, \ldots, v_n$ such that the neighbors of v_i in $\{v_{i+1}, \ldots, v_n\}$ form a clique. Such an ordering is called a (perfect) elimination order.

Some Facts about chordal graphs:

- It can be decided in linear time whether a graph G is chordal and if so an elimination ordering witnessing this can also be found in linear time.
- If G is chordal then $G \setminus \{v\}$ is also chordal for every $v \in V(G)$.

Chordal Graphs

Alternative Characterization

A graph G is chordal if it contains no induced cycles of length at least 4.

Chordal Graphs and Colorings

Theorem

$\chi(G)$ can be computed in polynomial time if G is chordal.

Proof:

First construct an elimination ordering $v_1, \ldots, v_n$. This can e.g. be achieved by choosing any vertex whose neighborhood induces a clique and then deleting this vertex from G, etc. .
Color the vertices v_i greedily in reverse order, by always assigning the lowest color that is not used for any $v_j \in N(v_i)$ with $j > i$.
This yields a proper k-coloring for some k. Furthermore, if at some step the color k is used for a vertex v_i, then G contains a clique on at least k vertices (consisting of v_i and its higher numbered neighbors), hence, $k \leq \chi(G) \leq k$ and $\chi(G) = k$. □

Chordal Completion

Definition

The chordal completion number of a graph G is the minimum number of edges needed to make G chordal.

Question: Can we parameterize k-coloring by the chordal completion number?

Answer: No, because the chordal completion number can not be computed in polynomial time!

However, what happens if we restrict the input to graphs that can be made chordal by adding at most l edges?

l-k-Coloring

This leads us to the following problem:

l-k-Colorability	**Parameter:** l
Input: 2 natural numbers l and k and a graph G that can be made chordal by adding at most l edges. **Question:** Is $\chi(G) \leq k$?	

In order to solve this problem we need to be able to:

(1) Find a set A of at most l edges whose addition makes G chordal in FPT time with respect to l.

(2) Given A of size at most l decide k-Colorability in FPT time, again with respect to l.

l-k-Coloring

Hence, we need to be able to solve the following two parameterized problems in FPT time:

k-Chordal Completion — **Parameter:** k

Input: A graph G and a natural number k.
Question: Compute a set A of at most k edges s.t. the graph $(V(G), E(G) \cup A)$ is chordal, if no such set exists answer No.

l-Chordal-Supergraph-k-Colorability (l-CS-k-Col)

Input: A graph G, a set $A \subseteq [V(G)]^2 \setminus E(G)$, and natural number k.
Parameter: $l = |A|$
Question: Is $\chi(G) \leq k$?

Chordal Completion

k-Chordal Completion — **Parameter:** k

Input: A graph G and a natural number k.
Question: Compute a set A of at most k edges s.t. the graph $(V(G), E(G) \cup A)$ is chordal, if no such set exists answer No.

Theorem

k-Chordal Completion is fixed-parameter tractable, i.e., it can be solved in time $O^*(4^k)$ via a simple branching algorithm.

l-Chordal-Supergraph-k-Colorability

Hence, we are left with designing an FPT algorithm for the following problem.

l-Chordal-Supergraph-k-Colorability (l-CS-k-Col)

Input: A graph G, a set $A \subseteq [V(G)]^2 \setminus E(G)$, and natural number k.
Parameter: $l = |A|$
Question: Is $\chi(G) \leq k$?

l-Chordal-Supergraph-k-Colorability

Let G be a graph and $\{u, v\} \in E(G)$.

Definition

$G/\{u, v\}$ is the graph obtained from G after contracting the edge $\{u, v\}$ into a new vertex, i.e., $G/\{u, v\}$ has vertex set $(V(G) \setminus \{u, v\}) \cup \{n\}$ and edge set

$$\{\{x, y\} \in [V(G) \setminus \{u, v\}]^2 : \{x, y\} \in E(G)\} \cup$$
$$\{\{x, n\} : x \in V(G) \setminus \{u, v\} \text{ and } (\{x, u\} \in E(G) \text{ or } \{x, v\} \in E(G))\}.$$

l-Chordal-Supergraph-k-Colorability

Let G be a graph and $u, v \in V(G)$.

Definition

$G \oplus \{u, v\}$ is the graph obtained from G after adding an edge between u and v, i.e., $G \oplus \{u, v\}$ has vertex set $V(G)$ and edge set $E(G) \cup \{u, v\}$.

Definition

$G \oplus_{/} \{u, v\}$ is the graph $(G \oplus \{u, v\})/\{u, v\}$.

l-Chordal-Supergraph-k-Colorability

The following observation follows immediately from the characterization of chordal graphs via long induced cycles.

Observation

If G is chordal then the graph G' obtained from G by contracting an edge is also chordal.

A branching rule

Theorem (1)

Let G be a spanning subgraph of a chordal graph H and $\{u, v\} \in E(H) \setminus E(G)$. Then G has a k-coloring iff either $G \oplus \{u, v\}$ has a k-coloring or $G \oplus_{/} \{u, v\}$ has a k-coloring.

Proof:

Let α be a k-coloring of G. If $\alpha(u) \neq \alpha(v)$ then α is also a k-coloring of $G \oplus \{u, v\}$. Otherwise, setting $\beta(n) = \alpha(u) = \alpha(v)$ and $\beta(x) = \alpha(x)$ for all $x \neq n$ gives a k-coloring β of $G \oplus_{/} \{u, v\}$.

For the reverse direction first note that a k-coloring of $G \oplus \{u, v\}$ is also a k-coloring of G. Furthermore, a k-coloring α of $G \oplus_{/} \{u, v\}$ gives a k-coloring β of G as follows: $\beta(u) = \beta(v) = \alpha(n)$ and $\beta(x) = \alpha(x)$ for all other x. □

A branching rule

Let (G, H, k) be a l-CS-k-Col instance with parameter $l = |E(H)| - |E(G)|$ and $\{u, v\} \in E(G)$.

Observation (1)

Then $(G \oplus \{u, v\}, H, k)$ is l-CS-k-Col instance with parameter $|E(H)| - |E(G \oplus \{u, v\})| = l - 1$.

Observation (2)

Then $(G \oplus_{/} \{u, v\}, H/\{u, v\}, k)$ is l-CS-k-Col instance with parameter $|E(H/\{u, v\})| - |E(G \oplus_{/} \{u, v\})| = l - 1$.

A branching algorithm

Theorem

The k colorability of a l-CS-k-COL instance (G, H, k) can be decided with a branching algorithm that yields a search tree with at most 2^l leaves where $l = |E(H)| - |E(G)|$.

Proof:

By induction over l. If $l = 0$ then G is chordal and k-colorability can be decided in polynomial time. Hence, the search tree has only $2^l = 2^0 = 1$ node.

A branching algorithm

Theorem

The k colorability of a l-CS-k-Col instance (G, H, k) can be decided with a branching algorithm that yields a search tree with at most 2^l leaves where $l = |E(H)| - |E(G)|$.

Proof, continued:

If $l > 0$ then we can choose an edge $\{u, v\} \in E(H) \setminus E(G)$. Using Theorem (1) we obtain that G is k-colorable iff either $G \oplus \{u, v\}$ is k-colorable or $G \oplus_/ \{u, v\}$ is k-colorable. Because of Observations (1) and (2) we obtain instances of l-CS-k-Col with parameter $\leq l - 1$ which by the induction hypothesis can be decided with a search tree with at most 2^{l-1} leaves. Hence, the search tree has at most $2^{l-1} + 2^{l-1} = 2^l$ leaves.

Outline

Idea:

Take an NP-hard problem and a class of instances for which the problem can be solved in polynomial time. Choose a parameter that expresses “how close’ a particular instance is to this class, i.e., the parameter expresses the “distance to triviality”.

To obtain an FPT algorithm for the NP-hard problem parameterized by this “distance to triviality” you need to:

- Restrict the input instances to those that are close to triviality.
- Find an FPT algorithm to compute the difference to triviality.
- Find an FPT algorithm that given the distance to triviality solves your problem.

Examples

Example 1)

For a CNF formula F the deficiency is the difference between the number of clauses and the number of variables. The maximum deficiency $\delta^*(F)$ is the maximum deficiency over all subformulas of F. Then:

- formulas F with $\delta^*(F) = 0$ are satisfiable.
- An FPT algorithm for satisfiability exists for the parameter δ^*.

Examples

Example 2)

Many graph problems are easy on trees. Later we will see a parameter called treewidth that expresses how close a graph is to being a tree. Almost every problem has an FPT algorithm parameterized by treewidth.

Examples

Example 3)

Take the satisfiability problem of CNF formulas. There exists a number of classes of formulas for which it becomes tractable, such as, 2-CNF, HORN, 0-VALID, q-HORN etc. . Now if you can easily (in FPT time) find a set B of variables of the formula F such that the reduced formula $F[\tau]$ is in such a class for every truth-assignment τ of the variables in B, then you can decide the satisfiability of F in time $O^*(2^{|B|})$ – just go over all $2^{|B|}$ truth assignments of the variables in B and decide whether $F[\tau]$ is satisfiable. Such a set B is called a $\mathcal{C}$-strong backdoor set for some tractable class of formulas $\mathcal{C}$.

Outline

Bounded Search Trees – Summary

- For bounded search trees, both the node degrees (number of branching cases) and the maximum depth should be bounded by a function of k, to obtain an FPT algorithm.
- Bounding the depth can often be done by considering the parameter, but sometimes a new parameter needs to be defined.
- Many of the fastest FPT algorithms are branching algorithms, using elaborate case distinctions. Analyzing these can be done with branching vectors.