RWTH Aachen **WS 2011/12**
Lehrgebiet Theoretische Informatik Problem Set 6
Langer—Reidl—Rossmanith—Sikdar 25.11.2011

Parameterized Algorithms Tutorial

Let G be a graph. A set of connected subgraphs $\mathcal{B}$ is called a *bramble* if the following condition holds:
For each $W_1, W_2 \in \mathcal{B}$ either W_1 and W_2 share a vertex *or* they are connected via an edge in G (i.e. there exist two vertices $u \in W_1, v \in W_2$ such that $uv \in E(G)$).
The *order* of $\mathcal{B}$ is the size of its minimum hitting set.

Tutorial Exercise T15

If a graph G has a bramble of order $k+1$, then $tw(G) \geq k$. Prove this by providing a winning strategy for a robber against k cops.
Show that a $r \times r$-grid has a bramble of order $r+1$.

Proposed Solution

After the k cops have been placed, there must be a subgraph $W \in \mathcal{B}$ which does *not* contain a vertex occupied by a cop (otherwise the bramble cannot have order $k+1$). We choose some vertex of that subgraph as the position for the robber. Now, if a cop moves the robber can adhere to the following simple strategy to never get caught: always move to a subgraph $W \in \mathcal{B}$ which is not occupied by a cop. From the definition of a bramble above we can see that such a move is always possible, as each subgraph of a bramble is reachable from every other subgraph.
For the $r \times r$-grid, the following collection of subgraphs is a bramble of order $r+1$:

- The topmost row of the grid
- The leftmost column of the grid excluding the top left vertex
- For each vertex v not in the topmost row and not in the leftmost column, the union of the row and column meeting in v (a cross-shaped subgraph)

The number of subgraphs in the bramble is $2+(r-1)^2$. It is left to show that the size of a minimum hitting set is at least $r+1$. First we convince ourselves that the cross-shaped subgraphs need at least a hitting set of size $r-1$. Assume otherwise, i.e. that there is a hitting set of size $r-2$. But this means that there exists a column and a row which is not hit by any vertex, therefore the cross-shaped subgraph consisting of exactly that column and row is not hit, either. Thus, we need at least $r-1$ vertices to hit all cross-shaped subgraphs. As the remaining two subgraphs (the first row and column) are disjoint to each other and all cross-shaped subgraphs, it follows that proposed bramble has order $r+1$.

Tutorial Exercise T16

Remember how we solved Vertex Cover on graphs of bounded treewidth? Do the same for Dominating Set. Then, think about how the algorithm could be modified in order to run in time $O(4^{tw(G)}\, poly(n))$.

Proposed Solution

As a first approach, let us see how the straightforward method performs. For a nice tree-decomposition of width w, we consider the table of each bag X as a function $f_X \colon \Sigma^w \to \mathbb{N} \cup \{\infty\}$ mapping "coloring" of the vertices of X to a natural number (the solution size). For vertex cover, we needed two symbols for each vertex: either v is in the vertex cover, which we denote by 1, or it is not and we denote this by 0. This means that a table consists of 2^w entries, i.e. possibly all bitstrings of length w over the alhpabet $\{0, 1\}$. For dominating set, however, we need three symbols: either v is in the dominating set (1), or it is dominated but not itself in the set (D), or it is not dominated (0). The size of the tables is therefore 3^w, but we still have to see how long the individual operations on the tree-decomposition (introduce, forget, join) take. For a simple notation, let $S \in \Sigma^*$ denote some string for the current table. Let us write $S[i]$ to denote the character at the ith position of S. We assume that for an introduce-operation, the new vertex will be added to the end of the table, i.e. the last character of the string corresponds to its state. Remember that in a nice tree-decomposition, a join-bag has exactly two children which contain the same vertices. In the following, these two bags will be denoted by L and R. We also need following simple counting function $\#1 \colon \Sigma^c \times \Sigma^c \to \mathbb{N}$ for some constant c, where $\#1(S_1, S_2)$ denotes the number of ones that occur in both strings S_1, S_2 of length c.

Operation	Recurrence
Introduce v	$f_{X+v}(S+1) = \begin{cases} f_X(S) + 1 \text{ if each neighbour is is either } D \text{ or } 1 \\ \infty \text{ otherwise} \end{cases}$
	$f_{X+v}(S+D) = \begin{cases} f_X(S) \text{ if there exists a neighbour marked 1} \\ \infty \text{ otherwise} \end{cases}$
	$f_{X+v}(S+0) = \begin{cases} f_X(S) \text{ if there is no neighbour marked 1} \\ \infty \text{ otherwise} \end{cases}$
Forget v	$f_{X-v}(S) = \min\{f_X(S+1), f_X(S+D), f_X(S+0)\}$
Join X	$f_X(S) = \min_{S_L \oplus S_R = S}\{f_L(S_L) + f_R(S_R) - \#1(S_1, S_2)\}$

Where the string operator $\oplus$ in the above table is defined character-wise as

$\oplus$	1	D	0
1	1	1	$\perp$
D	1	D	D
0	$\perp$	D	0

where we introduced a new symbol $\perp \notin \Sigma$ to denote an invalid combination of strings. Now, introduce and forget clearly only take time polynomial in w. What about the join? It now looks as if we have to regard each pair of strings from the table of L and R! This would yield an upper bound of $O((3^w)^2 n^{O(1)}) = O(9^w n^{O(1)})$, far above what we want to achieve. How can we speed up the computation of the join?

The resulting join table has at most 3^w entries, and yet we are taking 9^w time to compute it. Therefore, let us get away from the usual bottom-up computation and approach this from top-down: for a certain entry S of the join-table we want to compute its value $f_X(S)$ by using the values given in the tables f_L and f_R.

To this end, we should ask ourselves what strings S_L, S_R are needed to compute a string S in the join bag.

Regard the $\oplus$-table again: note that a 0 in S can only be the result of 0s in both S_L and S_R at the same position. Therefore, given a string S, all positions that are 0 are 'fixed' for the pair of strings S_L, S_R we have to regard. In particular, if S would contain only zeros we would have to look at only one entry from each table!

For all positions in S that are 1 we can argue differently: we could, by the above operation, obtain a 1 by the operations $1 \oplus 1$, $1 \oplus D$ and $D \oplus 1$. However, we can restrict ourselves to entries S_L, S_R where the entry in question is 1 without losing anything. In other words: we can modify our operation $\oplus$ as follows.

$\oplus$	1	D	0
1	1	$\perp$	$\perp$
D	$\perp$	D	D
0	$\perp$	D	0

How can we use this fact? Let us introduce a slightly larger alphabet for the computation of the join table: we generate all strings of length w over $\{0, 1, D_L, D_R, D_B\}$ where for an index i the symbols indicate the following:

- $S[i] = D_L$ means that $S_L[i] = D$ and $S_R[i] = 0$
- $S[i] = D_R$ means that $S_R[i] = D$ and $S_L[i] = 0$
- $S[i] = D_B$ means that $S_L[i] = S_R[i] = D$

Consider the following example, containing our target string S for which we want to calculate the join-table entry and the strings from the left and right table which we need for this calculation:

$$\begin{array}{ll} S & 10D_L101D_B00D_R \\ S_L & 10D101D000 \\ S_R & 100101D00D \end{array}$$

Now, the main point here is that given such a string over the extended alphabet, we know *exactly* which entries we need to calculate the join-table entry—compare this to the situation above, where we tried out all 9^w combinations of strings.

Of course, after generating the table of all 5^w such strings we need to reduce it to the 3^w strings we need in the end, but this is easily done by mapping the symbols D_L, D_R, D_B to D and taking the minimum of all extended strings that get mapped to the same final string. This approach will therefore give us a running time of $O(5^w n^{O(1)})$, quite good already, but still not the goal we want to achieve.

The natural question to ask now is: can we reduce the size of the extended alphabet from five to four? Regard the domination symbol D again. What we want to express with it is that a vertex is dominated by some other vertex (which we probably cannot see anymore). In the situation of a join-bag and in the contex of this top-down computation,

the symbol D in the join-table means that a vertex must be dominated by some vertex in a bag somewhere below, but we do not really care whether this occured in the left subtree, the right subtree, or even in both. Say we *know* it the vertex marked by D was dominated in the left subtree: why should we consider both possibilities (not dominated vs. dominated) for the right subtree? We already have what we want! The entry in the right subtree where this vertex is *not* dominated *can only be better.*

With this consideration, let us leave out the symbol D_B and use the remaining extended alphabet $\{0, 1, D_L, D_R\}$ with the following semantic:

- $S[i] = D_L$ means that $S_L[i] = D$ and $S_R[i] = 0$
- $S[i] = D_R$ means that $S_R[i] = D$ and $S_L[i] = 0$

This, finally, gives us the algorithm running in time $O(4^w n^{O(1)})$.

Tutorial Exercise T17

Prove the following: if a graph G has treewidth k, there exists a vertex set $S \subseteq V(G)$ such that

1. $|S| \leq k$
2. The connected component of $G - S$ can be divided into two sets such that the number of vertices on each side is at most $\frac{2}{3}n$

Note: this exercise was wrongly stated in the original version of this sheet

Proposed Solution

Consider a nice tree-decomposition $\mathscr{T} = (T, \mathcal{X})$ of G. Take an arbitrary bag X of $\mathscr{T}$. Regard the bags adjacent to X in $\mathscr{T}$, as we assumed a nice tree-decomposition, there are at most three such neighbouring bags which we will label X_1, X_2, X_3 which induce subgraphs G_1, G_2, G_3 of $G - X$. If all subgraphs have size $\leq \frac{1}{3}n$ we are done, so assume wlog that G_1 is the largest subgraph, which implies $|G_1| \geq \frac{1}{3}n$. This means that $|G[V(G_1) \cup V(G_2) \cup X]| < \frac{2}{3}n$, therefore we can pick X_1 as a seperator and repeat the previous process, i.e. move in the direction of a too-large subgraph or stop and use the current bag as S.

Note, however, that this process must terminate at some point if we do not back-track: the size of the largest component must decrease with each such step along the tree-decomposition.

Homework H12

Show that graphs on the right have treewidth exactly k by

1. giving a strategy for $k+1$ cops to catch a robber or a tree decomposition of width k
2. providing a bramble of size $k+1$

If the strategy is simple enough you do not have to prove its correctness.

Graph	treewidth k
Tree	1
Cycle	2
Wheel	3
$K_{n,n}$	n

Proposed Solution

Graph	Bramble	Strategy/Decomposition
Tree	Partition the tree into two vertex-disjoint subtrees. Both subgraphs are connected to the other via an edge and we need two vertices to hit both graphs.	Start at the root with the first cop, place the second on the child in whose componend the robber resides, recurse.
Cycle	Partition the cycle into three paths.	Keep one cop in place, let the other two leap-frog through the cycle.
Wheel	Partition the rim of the wheel in three paths and take the center vertex as its own subgraph.	Same as the cycle, but place one more cop at the center.
$K_{n,n}$	Pair the vertices via a matching of size n. Take all matchings except one and use them as subgraphs, take the remaining two vertices and use each as a single subgraph.	Place a cop on each vertex of one side, then place the remaining cop on the vertex where the robber hides.

Homework H13

Proof that a graph G which has a vertex cover of size k also has treewidth at most k.

Proposed Solution

We simply show that we can catch a robber with $k+1$ cops in such a graph. The strategy is as follows: place once cop on each vertex of the vertex cover. The remaining components of the graph all have size one, they form an independent set. This means we can place the remaining cop on exactly the vertex on which the robber resides.

Homework H14

We define a n-loop as a $3n \times 3n$-grid where a $n \times n$-grid was removed from the center. Show that this graph has treewidth at most $2n$.

Proposed Solution

We simply combine the well-known strategies for a cycle and a grid: leave n cops in a row of the n-loop, then comb through the rest of the graph with $n+1$ cops (using the movement pattern we used for the $n \times n$-grid).