Karlsruher Institut für Technologie

Solution to Exercise Sheet 7

Submission due by July 30, 2026

Problem 1: Cycles of length 4

6 points

Consider the following problem. Given a graph $G = (V, E)$, find a smallest possible set $X \subseteq V$ such that $G[V \setminus X]$ contains no cycle of length 4 as a subgraph.

Use Courcelle's Theorem to show that this problem is in FPT when treewidth is used as the parameter.

Solution 1:

Since this is an optimization problem, we apply Courcelle's Theorem for optimization. The predicate $\varphi(X)$ states whether a vertex set X is a valid solution, and we optimize with respect to the solution size $\alpha(X) = |X|$.

A solution X is valid if and only if there is no cycle of length 4 in the graph G after removing all vertices from X. In other words, this is the case if and only if every vertex set $S \subseteq V$ that contains a cycle of length 4 contains at least one vertex from X. Expressed in MSO_2, we obtain:

$$\varphi(X) = \forall S \subseteq V \,:\, \text{has4Cycle}(S) \rightarrow (\exists v \in X \,:\, v \in S)$$

with

$$\text{has4Cycle}(S) = \exists v_1, v_2, v_3, v_4 \in S \,:\, \text{different}(v_1,v_2,v_3,v_4) \wedge \text{connected}(v_1,v_2,v_3,v_4)$$

where

$$\text{different}(v_1,v_2,v_3,v_4) = (v_1 \neq v_2) \wedge (v_1 \neq v_3) \wedge (v_1 \neq v_4) \wedge (v_2 \neq v_3) \wedge (v_2 \neq v_4) \wedge (v_3 \neq v_4),$$

$$\text{connected}(v_1,v_2,v_3,v_4) = \text{adj}(v_1,v_2) \wedge \text{adj}(v_2,v_3) \wedge \text{adj}(v_3,v_4) \wedge \text{adj}(v_4,v_1),$$

and $\text{adj}(x, y) = \exists e \in E \,:\, \text{inc}(x,e) \wedge \text{inc}(y,e)$, which expresses the adjacency of two vertices.

By Courcelle's Theorem, the minimum solution can be found in time $\mathcal{O}(f(|\varphi|, t) \cdot n)$ where t is the treewidth and f is computable. Since the length of φ is constant, f depends only on the treewidth. Hence, the problem is fixed-parameter tractable with respect to the treewidth.

Problem 2: Chordal Graphs

6 + 6 = 12 points

Part (a) Let $G = (V, E)$ be a chordal graph and let $a,b \in V$ be two distinct non-adjacent vertices. Let S be a minimal separator that separates a and b (i.e., no proper subset of S separates a from b). Show that S forms a clique.

Solution 2a:

Let G be a chordal graph. Assume the minimal separator S between a and b is not a clique, i.e., there exist non-adjacent vertices $s_1, s_2 \in S$. Let G_a and G_b be the connected components of a and b in the graph without S. Since S is minimal, there are paths from a (and also from b) to both s_1 and s_2. Let P_a be the shortest path between s_1 and s_2 in G_a and P_b the shortest path between s_1 and s_2 in G_b. Since the components separated by S do not intersect, the union of P_a and P_b forms a cycle. This cycle has length at least 4 and contains no chord, which contradicts the chordality of G.

Part (b) A vertex is called *simplicial* if its neighbourhood forms a clique. Show that every chordal graph is either a clique, or contains two non-adjacent simplicial vertices. You may use part (a).

Solution 2b:

We consider the two cases separately:

Case 1: Let G be a clique. Then G has no non-adjacent vertices, and therefore it does not contain (at least) two non-adjacent simplicial vertices. Furthermore, it is chordal since in every cycle of length at least 4, all chords exist.

Case 2: Assume G is not a clique. Then there exist two vertices $a, b \in V$ that are non-adjacent, and there is a minimal separator S that separates a and b. (*Since a and b are non-adjacent, such a separator always exists. In the worst case, it consists of all vertices of G except a and b themselves. Whenever a separator exists, a minimal separator also exists.*)

Thus, the separator S partitions G into three disjoint subsets A, B, and S with $a \in A$ and $b \in B$, and there are no edges between A and B. From exercise (a), we know that S is a clique.

Consider the vertex set A (the argument for B is analogous). It suffices to find one simplicial vertex in A, because the same reasoning applies to B, yielding two simplicial vertices in total.

We prove by induction on $|A|$ that A contains a simplicial vertex.

Base case ($|A| = 1$): The only vertex $a \in A$ can only be adjacent to vertices of S, which is a clique. Hence, the vertex a is simplicial.

Induction hypothesis ($|A| \leq n$): Let $n \in \mathbb{N}$. Assume that for every set A with $|A| \leq n$, the set A contains at least one simplicial vertex.

Inductive step ($|A| = n + 1$): Take the vertex $a \in A$ that is separated from $b \in B$ by the separator S.

Case 1: a is simplicial – we are done.

Case 2: a is not simplicial. Then, $A \cup S$ is not a clique. Consequently, there exist two non-adjacent vertices $a', b' \in A \cup S$ and a minimal separator S' that separates a' from b' in the subgraph $G[A \cup S]$, dividing it into disjoint sets A' (with $a' \in A'$) and B' (with $b' \in B'$). At least one of A' or B' is disjoint from the original separator S, because if both intersect S, there would be an edge between A' and B' (since S is a clique), contradicting that S' is a separator. Assume the set A' is the part disjoint from S. It has no edges to B, only to $A \cup S$. Because A', B', and S' are all non-empty, we have $1 \leq |A'| < |A' \cup S'| < n$, and by the induction hypothesis, there exists a simplicial vertex in $A' \subseteq A$.

Since the same argument applies to the set B, the graph contains at least two simplicial vertices.

Problem 3: Induced Matchings **10 points**

Let $G = (V, E)$ be a graph. A matching $M \subseteq E$ is an *induced matching* if there exists a vertex set $V' \subseteq V$ such that V' induces the matching M (i.e., $G[V'] = (V', M)$). The problem Induced Matching asks, given a graph G and a parameter k, whether there exists an induced matching of size k.

Show that Induced Matching is $W[1]$-complete.

Solution 3:

Containment in $W[1]$. To show that Induced Matching lies in $W[1]$, we reduce it to WCS[1]. So given a graph G, we have to construct a Boolean circuit of weft at most 1 such that G has an induced matching with k edges if and only if the circuit can be satisfied with exactly k true inputs.

In the circuit, we create, for each edge e of the graph G, an input gate I_e that is connected to a NOT gate. In the next layer, the NOT gates of two edges e_1 and e_2 are connected to an OR gate if and only if the edges share an endpoint or there exists another edge that connects an endpoint of e_1 with an endpoint of e_2. All OR gates feed into a single AND gate, which finally connects to the output gate.

For an induced matching of size k, we set the k input gates corresponding to the matching edges to true; this yields a satisfying assignment because the OR gates encode precisely the conditions required for an induced matching. Conversely, any satisfying assignment with k true inputs yields an induced matching of size k formed by the edges whose input gates are true.

A	B	C
a	x	f
b	x	e
c	y	f
a	x	h

UNIQUE instance

→

A'	A	B	C
1	a	x	f
2	b	x	e
3	c	y	f
4	a	x	h

FD_{FIXED} instance

Table 1: Example of reduction from UNIQUE to FD_{FIXED} (where A' is the fixed column).

$W[1]$-hardness. We reduce from INDEPENDENT SET, a known $W[1]$-complete problem. Let G be the graph of an INDEPENDENT SET instance. We construct a graph G' for the INDUCED MATCHING instance by copying G and, for every vertex $v_i \in \{v_1, \dots, v_n\} = V(G)$, adding an extra vertex v_i' and connecting v_i' to v_i and to all neighbours of v_i.

If G contains an independent set X of size k, then the set $\{(x,x') \mid x \in X\}$ forms an induced matching of size k in G'. Conversely, let M be an induced matching of size k in G'. For each edge $m \in M$ one endpoint lies in $\{v_1, \dots, v_n\}$. Selecting that endpoint for each edge yields a vertex set $X \subseteq V(G)$ with $|X| = |M|$. Because the matching edges are disjoint and the endpoints of different matching edges are independent in G, the set X is an independent set of size k.

Problem 4: Functional Dependencies

4 + 8 = 12 points

(This exercise builds on Lecture 13 from July 20)

Part (a) Provide a parameterized reduction from UNIQUE to FD_{FIXED}.

Solution 4a:

Given a UNIQUE instance, i.e., a database D and a parameter k, we construct an equivalent FD_{FIXED} instance consisting of a database D', a parameter k', and a column A' of D'. To do this, we simply add a new column A' to D, where the entries in A' are all different from one another (see table 1). We leave the parameter the same, i.e., $k' = k$. The FD_{FIXED} problem is now to find a functional dependency $X' \to A'$ with $|X'| \leq k' = k$. The reduction can obviously be carried out in polynomial time ($\mathcal{O}(\#\text{rows})$).

We show that the UNIQUE instance (D,k) has a unique column combination X of size $|X| \leq k$ if and only if the FD_{FIXED} instance (D',A',k') has a functional dependency $X' \to A'$ with $|X'| \leq k'$.

"⇒": Let X be a unique column combination of size at most k in D, i.e., the restriction of D to X contains no duplicate tuples. It follows immediately that $X \to A'$ is a functional dependency in D' (since for all pairs of rows (r_i,r_j) of D, we have: $r_i[X] = r_j[X] \Rightarrow i = j \Rightarrow r_i[A'] = r_j[A']$). Furthermore, $|X| = k = k'$.

A	B	C	D
a	x	c	f
a	x	c	g
b	y	d	g
a	y	e	h

FD_{FIXED} instance

→

A	B	C	D
a	x	c	f
a	x	c	g
b	y	d	g
a	y	e	h
0	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

FD instance

Table 2: Example of reduction from FD_{FIXED} (where A is the fixed column) to FD.

"⇐": Let $X' \to A'$ be a functional dependency in D' with $|X'| \leq k'$. Since the entries in column A' are all different, for every pair of rows (r_i,r_j) we have: $r_i[A] \neq r_j[A]$. By the definition of functional dependencies, it follows from this for every pair of rows (r_i,r_j) that $r_i[X'] \neq r_j[X']$, which means that X' is a unique column combination for D'. Since every unique column combination of D' is also one of D and, furthermore, $|X'| \leq k' = k$, the claim follows.

Part (b) Provide a parameterized reduction from FD_{FIXED} to FD.

Solution 4b:

Given an FD_{FIXED} instance, i.e., a database D, a column A of D, and a parameter k, we construct an FD instance consisting of a database D' and a parameter k'. Here, D' consists of the same columns as D. W.l.o.g., the values 0 and 1 do not occur anywhere in D. To obtain D', we add a new row consisting only of 0 entries to D. Furthermore, for each column $X \neq A$, we add a new row to D that contains a 1 exactly in column X and a 0 in all other columns (see table 2). The parameter remains the same, i.e., $k' = k$. The reduction obviously works in polynomial time ($\mathcal{O}$(#columns)).

We show that the FD_{FIXED} instance (D,A,k) has a functional dependency $X \to A$ with $|X| \leq k$ if and only if the FD instance (D',k') has a functional dependency $X' \to A'$ with $|X'| \leq k'$ for some column A'.

"⇒": Let $X \to A$ be a functional dependency in D with $|X| \leq k$. Then $X \to A$ is also a functional dependency in D' with $|X| \leq k = k'$: For pairs of rows (r_i,r_j) that are also in D, the condition obviously holds. For pairs of rows (r_i, r_j), where one of the two rows is from D and the other is not, the statement also holds, since the values 0 and 1 did not occur anywhere in D and r_i and r_j therefore differ in every entry. If r_i and r_j are both not from D, then by construction $r_i[A] = 0 = r_j[A]$, and thus the condition for a functional dependency is also satisfied.

"⇐": Let $X' \to A'$ be a functional dependency in D with $|X'| \leq k$. Then $A' = A$ must hold, since the newly inserted rows ensure that there can be no functional dependencies to columns other than A, not even if all other columns are selected. This is because, for every $X \neq A$, there are two rows in which all columns except X have the value 0, and X has the value 0 in one of these two rows and the value 1 in the other row.

Since every functional dependency in D' is also a functional dependency in D, it follows that $X' \to A' = A$ is also a solution for the FD_{FIXED} instance with $|X'| \leq k' = k$.

Problem 5: Independent Family

6 + 6 + 6 = 18 bonus points

In the following, we want to prove the $W[3]$-completeness of a natural problem from the database domain. The problem Independent Family can be viewed informally as Independent Set on hypergraphs, except that we select sets of vertices rather than individual (hopefully non-connected) vertices.

Formally, an instance of Independent Family consists of two sets of sets $\mathcal{S}, \mathcal{T} \in 2^U$ over a universe U and a parameter k. Each collection can be seen as the edge set of a hypergraph. The question is whether there exist k hyperedges $S_1, \ldots, S_k \in \mathcal{S}$ such that no hyperedge $t \in \mathcal{T}$ is contained in $\bigcup_{i=1}^{k} S_i$.

We also define Multicolored Independent Family as follows: Given the parameter k and the collections $\mathcal{S}_1, \ldots, \mathcal{S}_k$ as well as $\mathcal{T}$ of sets over a universe U (i.e., $\mathcal{S}$ is partitioned into k colors), decide whether there exist hyperedges $S_1 \in \mathcal{S}_1, \ldots, S_k \in \mathcal{S}_k$ such that no hyperedge $t \in \mathcal{T}$ is contained in $\bigcup_{i=1}^{k} S_i$.

Part (a) Show that Independent Family and Multicolored Independent Family are equivalent under parameterized reduction.

Solution 5a:

We reduce Independent Family from and to Multicolored Indepentent Family:

Independent Family → Multicolored Independent Family: Given an instance $U, \mathcal{S}, \mathcal{T}, k$ of Independent Family, we construct a Multicolored Independent Family instance as follows: Initially, we set $\mathcal{S}_i = \mathcal{S}$ for all $i \in [k]$ and copy over $U, \mathcal{T}$, and k as they are. With this, every solution of the original Independent Family instance directly translates to a solution of the constructed Multicolored Independent Family instance. For the other direction of the equivalence, we have to construct something that forbids two differently colored copies of the same vertex in S to be contained together in a solution of the constructed instance. We do this as follows. We introduce one new unique universe element $u_{e,c}$ for each combination of an original hyperedge $e \in \mathcal{S}$ and a color c between 1 and k. Then, we add the element $u_{e,c}$ to each hyperedge

$e' \in S_c$ that correponds to the edge e. Finally for each pair $e_i \in S_i$,$e_j \in S_j$ that are copies of the same vertex $e \in S$, we add the hyperedge $t = (u_{e,i}, u_{e,j})$ to the set $\mathcal{T}$. So now if these vertices e_i and s_j are chosen together, the hyperedge t is contained in the union $\bigcup_{i=1}^{k} S_i$, and hence, cannot be contained in a solution together. This is exactly what we wanted to achieve.

Multicolored Independent Family → Independent Family: Given an instance $U, (S_1, \dots, S_k), \mathcal{T}, k$ of Multicolored Independent Family, we construct an Independent Family instance as follows: First, we set $S = \bigcup_{i=1}^{k} S_i$ (w.l.o.g. this union is disjoint) and initially copy over $U, \mathcal{T}$, and k without any change. Then we basically do the same trick as before to forbid that a valid solution in the constructed instance contains two hyperedges which correspond to hyperedges of the original instance that had the same color. Namely, we first extend our universe U by new elements: for each color c from 1 to k and each hyperedge e in S_c, we add a unique new universe element $u_{e,c}$. We add to each hyperedge $e' \in S$ one of the new universe elements as follows: if e' corresponds to a hyperedge $e \in S_c$ with color c, we add the element $u_{e,c}$ to the hyperedge e'. Finally, we add a new hyperedges to $\mathcal{T}$ of size 2 for each pair of universe elements $u_{e_1,i}, u_{e_2,i}$ where $i \in [k]$ and $e_1, e_2 \in S_i$. With this, the equivalence of the original instance and the constructed instance is straightforward to show.

Part (b) Show that Independent Family is in $W[3]$.

Solution 5b:

We reduce Independent Family to WCS[3]. Given an Independent Family instance with universe U, hyperedge sets S and $\mathcal{T}$, and parameter k. We describe how to construct an equivalent circuit of depth 4 with weft 3. Figure 1 shows an example of the reduction. First, we create an input node v_e for each hyperedge $e \in S$. In the second level, we create an OR node v_u for each element of the universe U that is contained in at least one hyperedge from $\mathcal{T}$. We connect an input node v_e with a level 2 node v_u if u is contained in the hyperedge e. Such an OR node v_u therefore passes on 1 if and only if at least one input was selected that corresponds to a hyperedge from S that contains u. In level 3, for each hyperedge $t \in \mathcal{T}$, we create an AND node that receives an incoming edge exactly from those level 2 nodes that correspond to a node $u \in U$ that is contained in the corresponding hyperedge t. Such an AND node therefore evaluates to 1 if t is completely covered by the selected hyperedges from S (or if the corresponding input nodes in the circuit were set to true). In a valid solution of Independent Family, precisely this case must not occur for any hyperedge in $\mathcal{T}$, since otherwise $t \subseteq \bigcup_{i=1}^{k} S_i$ would hold, so none of the AND nodes from level 3 may evaluate to 1. Therefore, we OR all the AND nodes from level 3 together in level 4 and then negate. Thus, when setting k input nodes to true, the resulting circuit can evaluate to true overall if and only if there is an Independent Family of size k in the Independent Family instance.

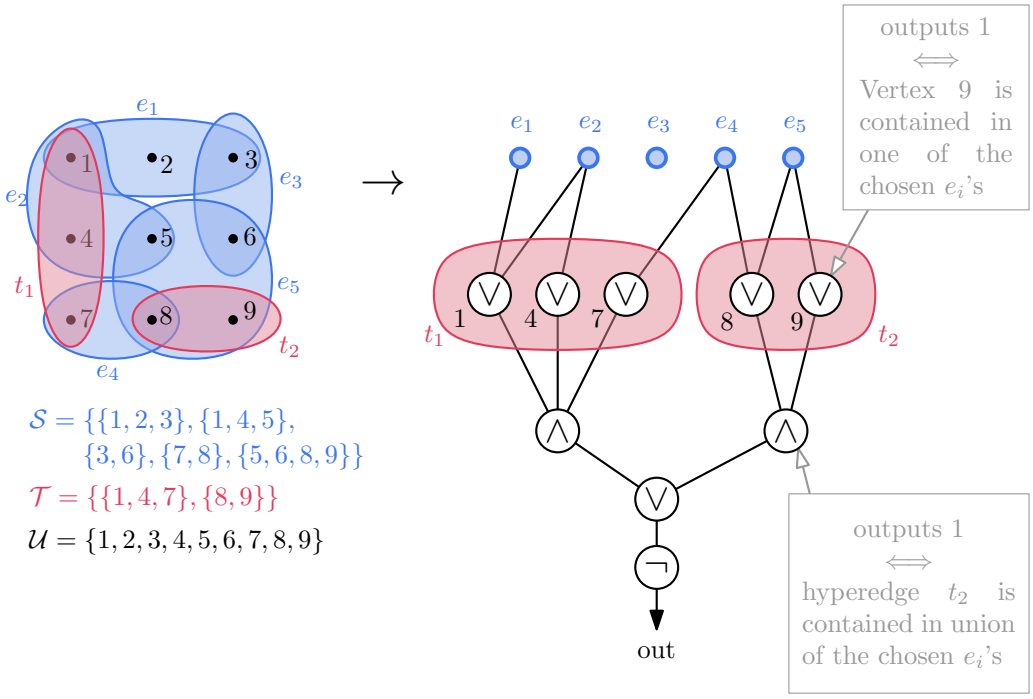

Figure 1: Example of reduction from Independent Family to WCS[3]

Part (c) Show that Multicolored Independent Family is $W[3]$-hard.

Hint: Weighted Antimonotone 3-normalized Satisfiability (see the lecture from July 20) is well suited for the reduction.

Solution 5c:

To show W[3]-hardness of Multicolored Independent Family, we reduce Weighted Antimonotone 3-normalized Satisfiability to Independent Family. This suffices because of part (a). First, we observe that we can transform the constructed circuit from subtask (b) using De Morgan so that we push the negation from the lowest level in each path upward directly below the input level, whereby all logical operations are reversed. We then obtain exactly one Weighted Antimonotone 3-normalized Satisfiability instance. So if we are now given a Weighted Antimonotone 3-normalized Satisfiability, then we can simply apply the construction from subtask (b) analogously in the other direction: For each AND node from level 3 (note that level 2 now contains the negations of the input), we create a universe element. For each input node v, we create a hyperedge in $\mathcal{S}$ consisting exactly of those universe elements which correspond to those AND-nodes to which the negation of v has an edge in the circuit. To make sure that we don't add the same hyperedge multiple times in this step (note that $\mathcal{S}$ is a set, not a multi-set), we use the same trick as in part (a) and add a unique universe element to each hyperedge. Level 4, in turn,

then defines the hyperedges in $\mathcal{T}$.