Tree Decompositions

Mahesh Viswanathan

Fall 2018

In this lecture, we will introduce class of "tree-like" graphs that have nice algorithmic properties. This class of tree-like graphs will include the class of series-parallel graphs that we introduced in the previous lecture. We begin this lecture by demonstrating that some NP-complete problems on graphs, can be solved in polynomial time using dynamic programming, when the input graph is a tree. This forms the basis for why "tree-like" also have nice algorithmic properties. We introduce the notion of tree width in graphs, and showing the same dynamic programming technique can be used to solve problems efficiently on graphs having a bounded tree width.

1 Trees and Dynamic Programming

A classical decision problem on graphs is the *independent set* problem. The independent set problem has a close connection with other computational problems including the problem of identifying the optimal number of registers to use during code generation in compiler construction. Given an undirected graph $G = (V, E)$, an *independent set* in G is a subset $I \subseteq V$ of vertices such that for every pair $u, v \in I$, $(u, v) \notin E$, i.e., none of the pairs of vertices in the independent set I are connected by an edge. Notice that $\emptyset$ is trivially an independent set of any graph, and so the interesting computational problem is to find an independent set of largest cardinality. The decision version of this computational problem is the following: given a graph G and a number k, determine if G has an independent set of size $\geq k$. Formally, as a language, we could write

$$\mathsf{IndependentSet} = \{\langle G, k\rangle \mid G \text{ has an independent set } I \text{ with } |I| \geq k\}.$$

This problem is known to be NP-complete.

Theorem 1. $\mathsf{IndependentSet}$ *is NP-complete.*

Proof. It is easy to see that $\mathsf{IndependentSet} \in \mathrm{NP}$. The nondeterministic polynomial time algorithm to check if $\langle G, k\rangle \in \mathsf{IndependentSet}$ simply guess a set I os size k, and checks that no pair of vertices in I are connected by an edge. The running time of this algorithm is, $O(k)$ to guess I and then an extra $O(k^2m)$ (m is the number of edges in G) to check that I is an independent set [1], giving a total running time of $O(k^2m)$.

NP-hardness is established by showing $3{-}\mathsf{SAT} \leq_{\mathrm{P}} \mathsf{IndependentSet}$. So given a 3-CNF formula φ, we will construct a graph G_φ and a number k_φ such that G_φ has an independent set of size k_φ iff φ is satisfiable. To understand the intuition behind the construction, it useful to take an alternate view of the $3{-}\mathsf{SAT}$ problem. The traditional view of $3{-}\mathsf{SAT}$ involves trying to find a way to assign 0/1 to the propositions such that the given formula evaluates to true. The view that we will find convenient for this reduction, is to pick a literal from each clause such that no pair of *complementary* literals are picked; the satisfying truth assignment is one that assigns truth values in such a way that the picked literals evaluate to true.

The reduction is shown in Figure 1 for the formula $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$. The idea behind is as follows. The graph G_φ has one vertex for each literal in each clause. The vertices that belong to the same clause are connected by edges to form a triangle; this ensures that at most one of these

[1] We need to check each pair in I is not an edge in G, which requires in the worst case going through the list of edges for each pair.

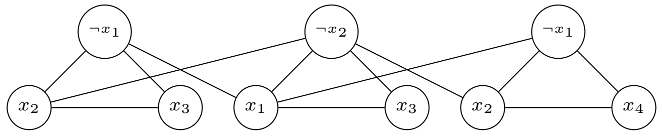


Figure 1: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

literals will be picked in any independent set. Next, 2 vertices that correspond to complementary lieterals are connected by an edge. This ensures that no independent set contains a pair of complementary literals. k_φ is then taken to be the number of clauses in φ. If there is an independent set of size k_φ, then the independent set corresponds to picking exactly one literal from every clause, no two of which are complementary — thus φ is satisfiable.

Formally, let $\varphi = \wedge_{i=1}^{m} (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$. Then $k_\varphi = m$, and $G_\varphi = (V, E)$, where $V = \{\ell_{ij} \mid 1 \leq i \leq m,\ 1 \leq j \leq 3\}$, and $E = \{(\ell ij, \ell i'j') \mid \text{either } i = i' \text{ or } \ell_{ij} = \neq \ell_{i'j'}\}$. Proof for the correctness of the reduction follows from the arguments in the previous paragraph. □

Eventhough IndependentSet is a computationally difficult problem on general graphs, it can be solved efficiently when the input graph is a (rooted) tree. The algorithm to solve IndependentSet on trees uses the dynamic programming paradigm.

Theorem 2. *Given a tree $T = (V, E)$, the size of maximum cardinality independent set can be computed in $O(|V|)$ time.*

Proof. For a vertex $u \in V$, let T_u denote the subtree rooted at u. Let $\mathsf{opt}(u, \emptyset)$ denote the size of maximum independent set in T_u that does not contain u, and $\mathsf{opt}(u, \{u\})$ denote the size of maximum independent set in T_u that contains u. Then,

$$\begin{array}{l}\mathsf{opt}(u, \emptyset) = \sum_{c \text{ child of } u} \max(\mathsf{opt}(c, \emptyset), \mathsf{opt}(c, \{c\})) \\ \mathsf{opt}(u, \{u\}) = 1 + \sum_{c \text{ child of } u} \mathsf{opt}(c, \emptyset)\end{array}$$

Thus, opt can be computed by dynamic programming in time $O(|V|)$ (as any u is child of only one vertex). □

A generalization of IndependentSet is the *maximum weight independent set problem.* Here, we are given a weighted graph $G = (V, E, w)$, where V is the set of vertices, E is the set of edges, and $w : E \to \mathbb{R}$ is a function assigning weights to each vertex. The goal is to compute an independent set in G of maximum weight; the weight of an independent set I is $w(I) = \sum_{u \in I} w(u)$. The decision version of this problem is, given a weighted graph G and k, determine if G has an independent set of weight $\geq k$. The problem is clearly NP-complete — membership in NP follows using the same algorithm as that for IndependentSet in the proof of Theorem 1, and hardness follows because IndependentSet is a special case of this problem where all vertices have weight 1. However, even this generalization of IndependentSet can be solved efficiently on trees; the proof uses the dynamic programming algorithm as the proof of Theorem 2.

Theorem 3. *Given a weighted tree $T = (V, E, w)$, the maximum weight independent set can be computed in $O(|V|)$ time (assuming arithmetic operations take $O(1)$ time).*

Proof. For a vertex $u \in V$, as before (i) let T_u denote the subtree rooted at u; (ii) let $\mathsf{opt}(u, \emptyset)$ denote the size of maximum weight independent set in T_u that does not contain u; and (iii) let $\mathsf{opt}(u, \{u\})$ denote the size of maximum weight independent set in T_u that contains u. Then,

$$\begin{array}{l}\mathsf{opt}(u, \emptyset) = \sum_{c \text{ child of } u} \max(\mathsf{opt}(c, \emptyset), \mathsf{opt}(c, \{c\})) \\ \mathsf{opt}(u, \{u\}) = w(u) + \sum_{c \text{ child of } u} \mathsf{opt}(c, \emptyset)\end{array}$$

Complexity bounds follow as in the case of Theorem 2. □

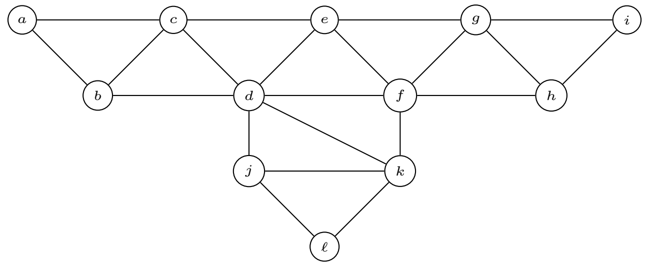


Figure 2: Example “tree-like” graph

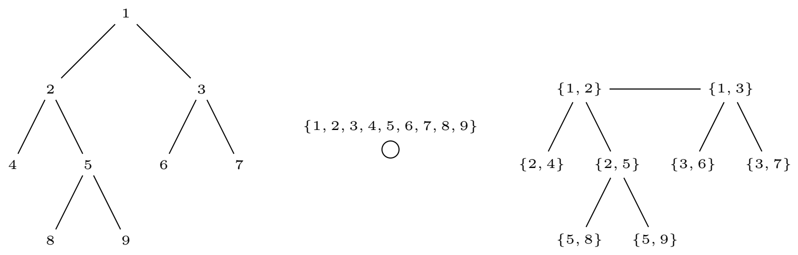


Figure 3: Tree T shown on the left. Two tree decompositions of T are shown to its right. In the middle is a trivial tree decomposition consisting of a single vertex, which on the right is a different tree decomposition.

In the light of Theorems 2 and 3, we could ask two questions that try to generalize the results of these theorems. Can we prove such results for properties other than independent sets in graphs? And, can we get polynomial time results for graphs more general than trees? We will answer both these questions in the affirmative in the following sections.

2 Tree Decomposition and Width

We will introduce the notion of tree width, that is, in some sense, a measure of how close a given graph is to being a tree. For example, consider the graph shown in Figure 2. The graph can be seen to be “tree-like” when we draw a new graph that treats the triangle “subgraphs” as a basic component. This process of identifying the subgraphs that form the basic components that enables viewing the graph as a tree, is called a tree decomposition, and the size of these basic components determines a graphs tree width.

Definition 4 (Tree Decomposition)**.** A *tree decomposition* of $G = (V, E)$ is a labeled tree $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ where $(V_{\tau(G)}, E_{\tau(G)})$ is a tree and $L_{\tau(G)} : V_{\tau(G)} \to 2^V$ is a labeling function such that

Node Coverage For every $u \in V$, there is $t \in V_{\tau(G)}$ such that $u \in L_{\tau(G)}(t)$

Edge Coverage For every $(u, v) \in E$, there is $t \in V_{\tau(G)}$ such that $\{u, v\} \subseteq L_{\tau(G)}(t)$

Coherence If $t_1, t_2 \in V_{\tau(G)}$ and $u \in V$ are such that $u \in L_{\tau(G)}(t_1) \cap L_{\tau(G)}(t_2)$ then $u \in L_{\tau(G)}(t)$ for every t on path from t_1 to t_2.

Let us look at some examples to understand the definition of a tree decomposition.

Example 5. Consider the tree $T = (V, E)$ shown on the left in Figure 3. There is a trivial tree decomposition of T as follows — $\tau(T) = (\{v\}, \emptyset, [v \mapsto \{1, 2, 3, 4, 5, 6, 7, 8, 9\}]$ is a tree decomposition consisting of a single node, and no edges, such that the label of the node is the set of vertices V. This *trivial* tree decomposition is shown in the middle in Figure 3.

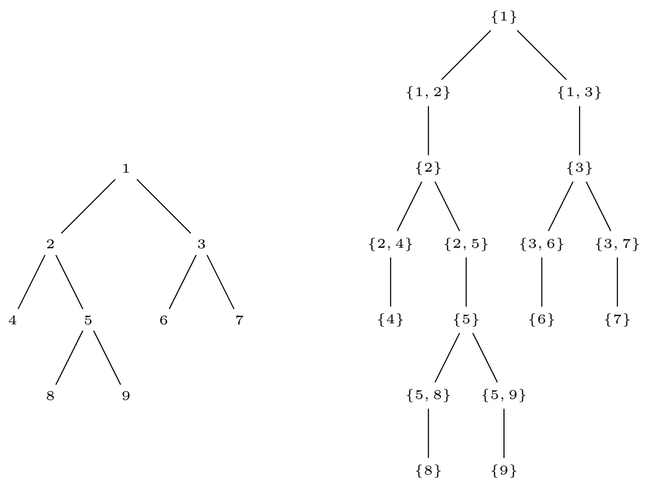


Figure 4: Tree T from Figure 3 on the left and another tree decomposition for it on the right.

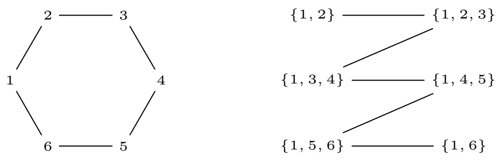


Figure 5: A cycle graph shown on the left, and one tree decomposition shown on the right

A more interesting tree decomposition for T is as follows. $\tau(T) = (V_{\tau(T)}, E_{\tau(T)}, L_{\tau(T)})$ where $V_{\tau(T)} = E$, $E_{\tau(T)} = \{(e_1, e_2) \,|\, e_1 \cap e_2 \neq \emptyset\}$, and $L_{\tau(T)}(e) = \{u, v\}$ where $e = \{u, v\}$. This tree decomposition is shown on the right in Figure 3, and has nodes corresponding to each edge of the tree, with two nodes being adjacent if they correspond to edges that share a common endpoint.

Yet another tree decomposition for tree T is shown in Figure 4. This one has nodes corresponding to each edge and vertex of T, and a node for a vertex is adjacent to a node for an edge if the vertex is an edpoint of the edge. Formally, $\tau(T) = (V \cup E, E_{\tau(T)}, L_{\tau(T)})$ where $E_{\tau(T)} = \{\{u, \{u, v\}\} \,|\, u \in V,\ \{u, v\} \in E\}$, and $L_{\tau(T)}(u) = \{u\}$ and $L_{\tau(T)}(\{u, v\}) = \{u, v\}$.

We can draw a few simple conclusions from Example 5. First, a graph may have multiple tree decompositions. Second, every graph G has a trivial tree decomposition consisting of a single node (with no edges) whose label is the set of all vertices of G. Given this observation, what will be important going forward is to construct tree decompositions of a graph where the maximum size of any label set is small. Example 5 demonstrates that every tree has a tree decomposition where the label of any node in the tree is a set of size at most 2.

Example 6. Consider the cycle of size 6 shown on the left in Figure 5. Apart from the trivial tree decomposition consistenting of a single vertex, a more interesting tree decomposition is shown to the right in Figure 5; in this tree decomposition nodes are labeled by sets of size at most 3.

Generally, consider the cycle graph with n vertices $C_n = (V = \{1, 2, \ldots n\}, E = \{\{i, j\} | (j-1) \equiv i \bmod n\}$. A tree decomposition for C_n is $\tau(C_n) = (V_{\tau(C_n)}, E_{\tau(C_n)}, L_{\tau(C_n)})$ where $V_{\tau(C_n)} = E$, $E_{\tau(C_n)} = \{\{e_1, e_2\} \,|\, (e_1 \cap e_2) \setminus \{1\} \neq \emptyset\}$, and $L_{\tau(C_n)}(\{u, v\}) = \{1, u, v\}$.

As observed in Examples 5 and 6, what is interesting is to construct tree decompositions where the label set of nodes is small. This leads to the notion of the tree width of a graph.

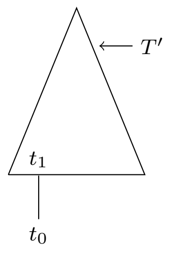


Figure 6: Picture showing the inductive step

Definition 7 (Tree Width)**.** A graph G has *tree width* w if there is a tree decomposition $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ such that for every vertex u, $|L_{\tau(G)}(u)| \leq w + 1$.

A graph with tree width w has a tree decomposition where every vertex is labeled by a set of size $\leq w+1$; the reason for taking the size of the label set to be $w+1$ is to ensure that trees have tree width 1, which follows from the observations in Example 5. The tree width of an arbitrary graph can be computed (parametric) polynomial time. The proof of this is beyond the scope of this course.

Theorem 8 (Bodlaender)**.** *There is an algorithm $\mathcal{A}$ and a polynomial p such that given any graph $G = (V, E)$, $\mathcal{A}$ computes a tree decomposition of G of width $k =$ tree-width(G) in time $O(2^{p(k)}|V|)$.*

Based on Example 5, we observed that trees have tree width 1. It turns out that they are the only (connected) graphs that have tree width 1.

Theorem 9. *A connected graph G has tree width* 1 *if and only if G is a tree.*

Proof. We already observed, based on the construction in Example 5, that trees have tree width 1. We will establish the converse here. Let $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ be a tree decomposition of width 1 for G. We will prove by induction on $|V_{\tau(G)}|$ that G is a tree.

For the base case, suppose $|V_{\tau(G)}| = 1$. Then G is either single vertex or an edge, and so G is a tree. For the induction step, let t_0 be a leaf of $\tau(G)$ with parent t_1. This situation is illustrated in Figure 6. Let the label of t_0 be $\{u, v\}$. Let $T' = \tau(G) \setminus t_0$ and $G_{T'}$ be the subgraph of G induced by the vertices labeling T'. By induction hypothesis, $G_{T'}$ is a tree. Suppose adding the edge $\{u, v\}$ to $G_{T'}$ introduces a cycle. Then there is $x \in T'$ such that $u \in L_{\tau(G)}(x)$. By coherence, all vertices on path from t_0 to x have u in their label. Thus, $u \in L_{\tau(G)}(t_1)$. By a similar reasoning, $v \in L_{\tau(G)}(t_1)$. Hence, $G = G_{T'}$ is a tree. $\square$

2.1 Special Tree Decompositions

Tree decompositions as defined in Definition 4 have very few restrictions on the structure of the labeled tree. For example, the tree may have "redundant" nodes and may be a tree of any arity. We show that we can always restrict our attention to trees that are binary and don't have reducndant nodes. We begin by defining what we mean by a tree decomposition that does not have any redundant nodes.

Definition 10. A tree decomposition $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ is said to be *non-redundant* if there is no edge $(x, y) \in E_{\tau(G)}$ such that $L_{\tau(G)}(x) \subseteq L_{\tau(G)}(y)$.

One can always obtain a non-redundant tree decomposition (of the same width) by "contracting" redundant edges.

Example 11. Consider the tree decomposition shown on the right in Figure 4 for the tree T shown on the left. This tree decomposition is redundant. However, we can make it non-redundant by contracting edges $\{u, v\}$ where $L(u) \subseteq L(v)$. Carry out such contraction of all reducndant edges will result in the tree decomposition shown on the right in Figure 3.

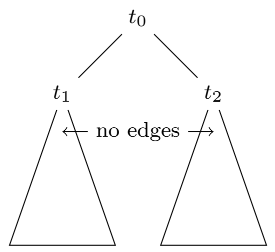


Figure 7: Dynamic programming yields a solution to the maximum weight independent set problem because there are no edges between vertices in subtrees rooted at different siblings.

Non-redundant tree decompositions have a small size when compared with the number of vertices in the original graph.

Proposition 12. *Any non-redundant tree decomposition* $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ *of graph* $G = (V, T)$ *has size* $|V_{\tau(G)}| \leq |V|$.

Proof. We prove this result by induction on $|V_{\tau(G)}|$. Consider the base case when $|V_{\tau(G)}| = 1$. In this case, the proposition holds trivially.

For the induction step, let t_0 be a leaf of $\tau(G)$ with parent t_1. Observe that $U = L_{\tau(G)}(t_0) \setminus L_{\tau(G)}(t_1) \neq \emptyset$, since $\tau(G)$ is non-redundant. By coherence, none of the vertices in U appear as labels in $T' = \tau(G) \setminus \{t_0\}$. Thus, T' is a non-redundant tree decomposition of $G \setminus U$. By induction hypothesis, $|T'| = |V_{\tau(G)}| - 1 \leq |G \setminus U| = |V| - |U| < |V| - 1$, as U is non-empty. Therefore, we have $|V_{\tau(G)}| < |V|$. □

The labeled tree of a tree decomposition could have any arity. However, without loss of generality, we could assume it is binary. Given a tree decomposition $\tau(G)$, we can construct another tree decomposition $\beta(G)$ that is a binary tree and has the same width as $\tau(G)$ as follows. Let $\tau(G)$ be a tree with root v and $t_1, t_2, \ldots t_n$ as subtrees of v; we will denote this as $\tau(G) = v(t_1, \ldots t_n)$. We will use $T(\cdot)$ to denote the recursive translation that will translate $\tau(G)$ to the binary tree $\beta(G)$. We will define $\beta(G) = v(T(t_1), T(v(t_2, \ldots v_n)))$, where the labeling function of $\beta(G)$ is "essentially the same" as that for $\tau(G)$.

The binary tree decomposition constructed in the previous paragraph has the following properties. If $\tau(G)$ has n nodes then $\beta(G)$ has $2n$ nodes. Thus, if the tree decomposition $\tau(G)$ to begin with is non-redundant, using properties of Proposition 12, we say that for any graph G with n vertices, has a binary tree decomposition of minimum width having at most $O(n)$ nodes. Therefore, we will henceforth assume that all tree decompositions are binary, and of the same size as G.

3 Dynamic Programming in Bounded Tree Width Graphs

Recall that the maximum weight indepedent set problem can be solved efficiently on trees using dynamic programming. The key reason why dynamic programming is effective in computing independent sets on trees is because of a property shown in Figure 7 — there are no edges between vertices in the subtree rooted at t_1 and the subtree rooted att_2, where t_1 and t_2 are siblings. Therefore, computing independent sets for the subtree rooted at t_1 and the independent set for the subtree rooted at t_2, independently, yields an independent set for the subtree rooted at t_0.

A similar property holds for a tree decomposition of a graph as well. Let the tree shown in Figure 7 be a tree decomposition of a graph G. Let us denote by $G(t_1)$ the subgraph induced by the vertices labeling the nodes in subtree t_1; $G(t_2)$ is defined similarly. Conherence ensures that an edge between vertices in $G(t_1)$ and $G(t_2)$ must involve a vertex labeling t_0. Thus the interconnection between $G(t_1)$ and $G(t_2)$ is captured by the label of t_0, and this can be used to solve maximum weight independent set on graphs with small tree width.

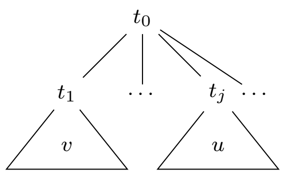


Figure 8: Schematic view of tree decomposition $\tau(G)$.

We begin by observing this technical property of tree decompositions. Let $G = (V, E, w)$ be a weighted graph and let $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ be a tree decomposition of G of width k. For a node $t \in V_{\tau(G)}$, let $T(t)$ denote the subtree rooted at t, and $G(t)$ be the subgraph of G induced by the vertices in the label set of nodes in $T(t)$. For an independent set $U \subseteq L_{\tau(G)}(t)$, let $I(t, U)$ denote the maximum weight independent set in $G(t)$ such that $I(t, U) \cap L_{\tau(G)}(t) = U$.

Lemma 13. *Let $t_0 \in V_{\tau(G)}$ with children $t_1, \dots t_n$. Let $I_j = I(t_0, U) \cap G(t_j)$. Then I_j is a maximum weight independent set in $G(t_j)$ with the property $I_j \cap L_{\tau(G)}(t_0) = U \cap L_{\tau(G)}(t_j)$.*

Proof. The tree decomposition in the lemma is shown in Figure 8. Let I'_j be the maximum weight independent set in $G(t_j)$ with the property $I'_j \cap L_{\tau(G)}(t_0) = U \cap L_{\tau(G)}(t_j)$. Suppose for contradiction $w(I'_j) > w(I_j)$. Consider $I' = (I(t_0, U) \setminus I_j) \cup I'_j$. Observe that $w(I') > w(I(t_0, U))$ and $I' \cap L_{\tau(G)}(t_0) = U$. If we prove that I' is an independent set, then we get a contradiction, which means that $w(I'_j) = w(I_j)$, establishing the lemma.

Suppose for contradiction, I' is not an independent set. Then there is an edge $\{u, v\}$ such that $\{u, v\} \subseteq I'$. Since I'_j is an independent set, $\{u, v\}$ is not a subset of I'_j. Further, since $I(t_0, U) \setminus I_j$ is also an independent set, $\{u, v\}$ is not a subset of $I(t_0, U) \setminus I_j$. Therefore, it must be the case that one endpoint of $\{u, v\}$ belongs to I'_j and one belongs to $I(t_0, U) \setminus I_j$. Without loss of generality, let us assume that $u \in I'_j$ and $v \in I(t_0, U) \setminus I_j$. This is shown in Figure 8. Since $\{u, v\} \in E$ by edge coverage, there is $t \in V_{\tau(G)}$ such that $\{u, v\} \subseteq L_{\tau(G)}(t)$. Suppose $t \in T(t_j)$. Then by coherence, $v \in L_{\tau(G)}(t_0)$ and so $v \in U$. By a similar reasoning, if $t \in T(t_0) \setminus T(t_j)$, then $u \in L_{\tau(G)}(t_0)$ and $u \in U$. Thus, either way, $\{u, v\} \subseteq I(t_0, U)$ which contradicts the fact that it is an independent set. □

Armed with Lemma 13, we are ready to present an efficient dynamic programming algorithm to compute the maximum weight independent set in graphs with bounded tree width.

Theorem 14. *Given a weighted graph $G = (V, E, w)$ of tree width k, the maximum weight independent set in G can be computed in $O(2^k|V|)$ time.*

Proof. Compute a non-redundant tree decomposition $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ of width k, using Bodlaender's theorem (Theorem 8). For $t \in V_{\tau(G)}$ and $U \subseteq L_{\tau(G)}(t)$ (an independent set), let $\mathsf{opt}(t, U)$ be the weight of maximum weight independent set I with $I \cap L_{\tau(G)}(t) = U$. Then,

$$\mathsf{opt}(t, U) = \sum_{c \text{ child of } t} \left\{ \left(\max_{V : V \cap L_{\tau(G)}(t) = U \cap L_{\tau(G)}(c)} \mathsf{opt}(c, V) \right) - w(U \cap L_{\tau(G)}(c) \right\} + w(U)$$

Let us analyze the running time of the algorithm. Since $\tau(G)$ is non-redundant, $|V_{\tau(G)}| \leq |V|$. Thus, the number of sub-problems is $\leq 2^k|V|$. For each subproblem, $\mathsf{opt}(t, U)$ is accessed at most $2^K + 1$ times. Thus, the total running time is $O(2^{k+1}2^k|V|) = O(2^k|V|)$. □

4 MSO on Graphs

We have studied monadic second order logic on words and trees, and demonstrated that they are decidable on such restricted classes of structures. We now consider MSO over the signature of graphs. We begin by studies their expressive power, before present the main result of this section, which demonstrates that MSO is efficiently decidable on graphs of small tree width.

4.1 Expressive Power

Monadic second order logic over the signature of graphs ($\tau_G = \{E\}$) can expressive a variety of interesting properties. MSO can be thought of as describing decision problems — a sentence φ describes the problem where given a graph G, the goal is to determine if $G \models \varphi$. We can see that MSO sentences are rich enough to describe computationally rich problems.

- "Graph is 3-colorable."

$$\begin{aligned}\exists X_1 \exists X_2 \exists X_3 (\forall x (X_1 x \vee X_2 x \vee X_3 x) \wedge \\ \forall x \forall y (Exy \rightarrow (\neg(X_1 x \wedge X_1 y) \wedge \neg(X_2 x \wedge X_2 y) \wedge \neg(X_3 x \wedge X_3 y))))\end{aligned}$$

- "Graph has independent set of size at least k"

$$\exists x_1 \cdots \exists x_k (\wedge_{i \neq j} \neg(x_i = x_j) \wedge \wedge_{i \neq j} \neg Ex_i x_j)$$

- "Graph has a vertex cover of size at most k"

$$\exists x_1 \cdots \exists x_k \forall x \forall y (Exy \rightarrow (\vee_i ((x = x_i) \vee (y = x_i))))$$

All the properties written above define computational problems that are NP-complete. We saw the theorem by Fagin that NP can be characterized exactly by the decision problems that can be described in existential second order logic. However, MSO is not as rich as existential second order logic. Though we can express NP-complete problems, it fails to describe all problems in NP; some problems that computationally very simple, cannot be expressed in MSO. We give an example of such a property.

Recall that a graph $G = (V, E)$ is said to *bipartite* if there is a partition of V into two sets (U_1, U_2) such that $E \subseteq (U_1 \times U_2) \cup (U_2 \times U_1)$, i.e., G has no edges between vertices belonging to the same partition. We could express the fact that a graph is bipartite in MSO as follows.

$$\exists U_1 \exists U_2 (\forall x (U_1 x \leftrightarrow \neg U_2 x) \wedge \forall x \forall y (Exy \rightarrow ((U_1 x \wedge U_2 y) \vee (U_2 x \wedge U_1 y))))$$

A bipartite graph $G = (V, E)$ with partites U_1 and U_2 is said to be *balanced* if the partites are of equal size, i.e., $|U_1| = |U_2|$.

Proposition 15. *Balanced bipartite graphs are not expressible in MSO. That is, there is no MSO sentence φ_b such that $G \models \varphi_b$ iff G is a balanced bipartite graph.*

Proof. Our proof will show that if balanced bipartite graphs were expressible in MSO then there is a non-regular (word) language L that is expressible in MSO (over the signature of words). Since we know that over words, MSO is expressively equivalent to regularity, we will have our desired result.

Without loss of generality, assume that the vertices of the graph are $\{0, 2, \ldots n - 1\}$. Consider the mapping f that maps (bipartite) graphs to words over $\{a, b\}$ such that symbol at position i in word $f(G)$ is a if $i \in U_1$ and is b otherwise. Observe that$G$ is balanced iff $f(G)$ has equal number of as and bs. Suppose φ_b expresses balanced bipartite graphs. Then $\varphi_b[Exy \mapsto (Q_a x \leftrightarrow Q_b y)]$ expresses the non-regular language $L = \{w \in \{a, b\}^* \mid \text{number of } a\text{s equals number of } b\text{s in } w\}$. □

Thus, while many NP-complete problems can be "expressed" in MSO (over graphs) but some very simple properties cannot. Note that the inexpressivity proof of Proposition 15, in fact shows that balanced bipartite graphs cannot be expressed in the signature $\{<, E\}$.

4.2 Courcelle's Theorem

In this section, we will prove Courcelle's theorem, which is as follows.

Theorem 16. *Given a graph $G = (V, E)$ of tree width w, and an MSO sentence φ, $G \models \varphi$ can be decided in time $O(f(|\varphi|, w)|V|)$.*

Thus, if a decision problem is defined by an MSO sentence, then Theorem 16 says that the problem be decided in linear time in the size of the input graph, provided the input has bounded tree width, Courcelle's theorem, therefore, generalizes previous observations about NP-complete problems like independent set, being solvable in polynomial time on graphs with bounded tree width.

We will prove Courcelle's theorem in the rest of this section through a series of observations. Our proof will rely on the observation that MSO can be decided on trees efficiently. We recall this observation here for completeness.

Theorem 17. *Given a Σ-labeled tree T (where Σ is a finite set) and an MSO sentence φ, $T \models \varphi$ can be decided in time that is linear in $|T|$.*

Proof. Construct the tree automaton $\mathcal{A}$ recognizing $[\![\varphi]\!]$ and check if $T \in L(\mathcal{A})$. □

The idea behind proving Theorem 16, is the following. Given a graph G, we will will reduce the question of determining if $G \models \varphi$ to a question of checking the satisfaction of an MSO sentence on a *tree*, and then use Theorem 17. To carry this plan out, we need to construct a tree labeled by elements of a finite alphabet, and transform φ to another sentence such that satisfaction is preserved. Using Theorem 8, we can construct a tree decomposition $\tau(G)$ of G; we can assume that $\tau(G)$ is a binary tree of the same size as G. However, $\tau(G)$ is not a tree with a finite label set; its nodes are labeled by subsets of vertices of G, and so the label set depends on the size of G. Thus, we cannot directly use Theorem 17 on this tree. Instead, we will use $\tau(G)$ to construct another tree $\mathcal{T}_{G,\tau(G)}$ of the same size. Further, the nodes of $\mathcal{T}_{G,\tau(G)}$ will be labeled by elements from a *finite* set. The construction of the tree $\mathcal{T}_{G,\tau(G)}$ will crucially rely on the fact that the width of $\tau(G)$ is bounded. We will also construct an MSO sentence φ' such that $G \models \varphi$ iff $\mathcal{T}_{G,\tau(G)} \models \varphi'$. Theorem 16 then follows from Theorem 17.

As a first step towards filling out the above proof outline, let us establish some conventions about tree decompositions of width w. Every $S \subseteq V$ of size at most $w + 1$ can be encoded as a string of length $w + 1$ — list the elements of S in some (canonical) order, and repeat some to ensure that the string is of length exactly $w + 1$. For example, if $w = 3$, the set $\{1, 2, 3\}$ can be thought of as string 1231 or 3211, etc. Thus, we can assume that a tree decomposition $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ of width w for graph $G = (V, E)$ will be such that $L_{\tau(G)} : V_{\tau(G)} \to V^{w+1}$

Let us now describe the tree $\mathcal{T}_{G,\tau(G)}$ that is labeled by elements from a finite alphabet. We have $G = (V, E)$ be a graph, $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ of width w. Define $\mathcal{T}_{G,\tau(G)} = (V_{\tau(G)}, E_{\tau(G)}, L)$, with $L(u) = (\lambda_1, \lambda_2, \lambda_3)$ for any $u \in V_{\tau(G)}$, where

- $\lambda_i \subseteq \{1, 2, \ldots w + 1\} \times \{1, 2, \ldots w + 1\}$, for $i = 1, 2$, or 3.
- $\lambda_1 = \{(i, j) \mid (L_{\tau(G)}(u)[i], L_{\tau(G)}(u)[j]) \in E\}$
- $\lambda_2 = \{(i, j) \mid L_{\tau(G)}(u)[i] = L_{\tau(G)}(u)[j] \text{ and } i \neq j\}$
- $\lambda_3 = \{(i, j) \mid L_{\tau(G)}(u)[i] = L_{\tau(G)}(v)[j]\}$, where v is the parent of u in $\tau(G)$

Example 18. Consider the cycle graph with 6 vertices $\{A, B, C, D, E, F\}$ shown in Figure 9. Its tree decomposition of width 2 is shown to its right. Notice, that the vertices are labeled by a string, with possibly repeated vertices, as per the convention adopted in this section. Finally, the tree $\mathcal{T}_{G,\tau(G)}$ shown below both of them. Notice that $\mathcal{T}_{G,\tau(G)}$ depends upon how the sets labeling the nodes in the tree decomposition are written as strings in $\tau(G)$.

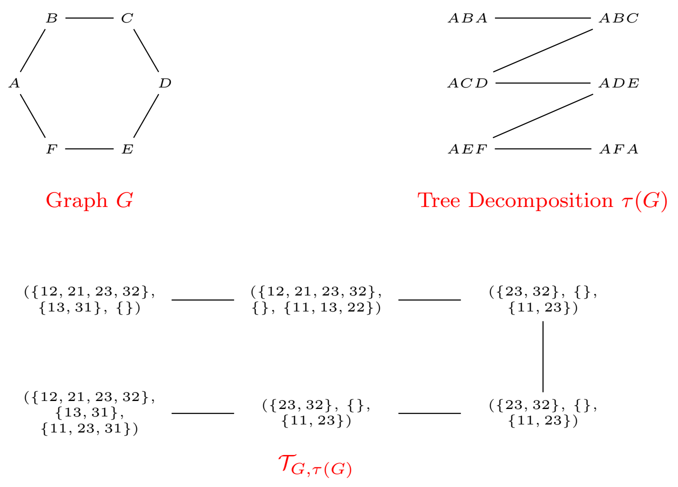


Figure 9: Showing graph G, its tree decomposition $\tau(G)$, and the tree $\mathcal{T}_{G,\tau(G)}$

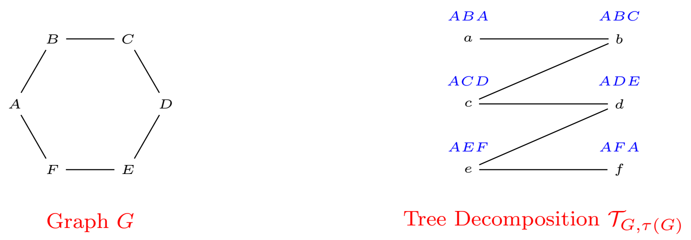


Figure 10: Graph G and its tree decomposition

To describe the translation of the MSO formula φ to a formula φ' for the tree $\mathcal{T}_{G,\tau(G)}$ requires one to interpret vertices and edges of graph G in the tree $\mathcal{T}_{G,\tau(G)}$. We will interpret vertices of G as a tuple of nodes of $\mathcal{T}_{G,\tau(G)}$. Let $G = (V, E)$ be a graph and $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ of width w. A set $S \subseteq V$ will be encoded as $\mathsf{Tree}(S) = (U_1, U_2, \ldots U_{w+1})$, where $U_i \subseteq V_{\tau(G)}$ such that

$$U_i = \{u \in V_{\tau(G)} \mid L_{\tau(G)}(u)[i] \in S\}$$

Example 19. Let us look at an example to understand the interpretation of vertices. Consider the graph and its tree decomposition shown in Figure 10. Here is the interpretation of some subsets of vertices.

- $\mathsf{Tree}(\{A, C, E\}) = (\{a, b, c, d, e, f\}, \{c, e\}, \{a, b, d, f\})$
- $\mathsf{Tree}(\{A\}) = (\{a, b, c, d, e, f\}, \{\}, \{a, f\})$
- Not every tuple of tree nodes corresponds to vertices of the graph. For example, $(\{a\}, \{\}, \{\})$ is not $\mathsf{Tree}(S)$ for any S

In what follows, let us fix a graph $G = (V, E)$, a tree decomposition $\tau(G) = (V_{\tau(G)}, E_{\tau(G)}, L_{\tau(G)})$ of width w, and the tree $\mathcal{T}_{G,\tau(G)} = (V_{\tau(G)}, E_{\tau(G)}, L)$. For any $u \in V_{\tau(G)}$, we will assume that $L(u) = (\lambda_1(u), \lambda_2(u), \lambda_3(u))$.

Proposition 20. *A tuple of sets of nodes of $\mathcal{T}_{G,\tau(G)}$, $\overline{X} = (X_1, \ldots X_{w+1})$ is the encoding* $\mathsf{Tree}(S)$ *for some S (with $|S| > 1$) iff*

(a) *If $(i,j) \in \lambda_2(u)$ then $u \in X_i$ iff $u \in X_j$.*

(b) *If $(i,j) \in \lambda_3(u)$ then $u \in X_i$ iff $v \in X_j$, where v is the parent of u.*

$\overline{X}$ is the encoding $\mathsf{Tree}(n)$ for some $n \in V$ iff conditions (a) and (b) hold and

(c) *If $u \in X_i$ and $u \in X_j$ then $(i,j) \in \lambda_2(u)$.*

(d) *If $u \in X_i$, $v \in X_j$ (where v is parent of u) then $(i,j) \in \lambda_3(u)$.*

(e) *$\cup_{i=1}^{w+1} X_i$ forms a connected component of $\mathcal{T}_{G,\tau(G)}$.*

Proof. Let us first consider the case of S where $|S| > 1$. It is easy to see that for any S (with $|S| > 1$) $\mathsf{Tree}(S)$ satisfies conditions (a) and (b). We need to prove the converse. For a tuple $\overline{X} = (X_1, \ldots X_{w+1})$, let

$$\mathsf{Set}(\overline{X}) = \{v \in V \mid \exists i \exists u.\ u \in X_i \text{ and } L_{\tau(G)}(u)[i] = v\}$$

We will show that $\mathsf{Tree}(\mathrm{Set}(\overline{X})) = \overline{X}$, provided $\overline{X}$ satisfies conditions (a) and (b).

Suppose $\mathsf{Tree}(\mathrm{Set}(\overline{X})) \neq \overline{X}$. Then there are u_1, u_2 (possibly the same) such that $L_{\tau(T)}(u_1)[i] = L_{\tau(T)}(u_2)[j] = v$ such that $u_1 \in X_i$ and $u_2 \notin X_j$. Since $v \in L_{\tau(G)}(u_1)$ and $v \in L_{\tau(G)}(u_2)$, v appears in the label of every vertex on the path from u_1 to u_2. Thus, without loss of generality, we may assume that either $u_1 = u_2$ or u_1 is parent of u_2. But then $\overline{X}$ either violates (a) or (b), which contradicts the fact that $\overline{X}$ satisfies both (a) and (b). Hence, we have established the proposition for S such that $|S| > 1$.

Let us now consider the case when $S = \{v\}$ for $v \in V$. Once again it is easy to see that $\mathrm{Tree}(\{v\})$ for any v satisfies conditions (a), (b), (c), (d), and (e). We need to prove the converse. For tuple $\overline{X}$, once again consider $S = \mathsf{Set}(\overline{X})$; we will argue that $|S| = 1$ if $\overline{X}$ satisfies (a),(b),(c),(d), and (e).

Suppose S has more than one element. Then there are vertices u_1 and u_2 (possibly the same) and i and j such that $u_1 \in X_1$, $u_2 \in X_j$, and $L_{\tau(G)}(u_1)[i] \neq L_{\tau(G)}(u_1)[j]$. Since $\overline{X}$ satisfies condition (e), we can assume that either $u_1 = u_2$ or u_1 is parent of u_2. But, then $\overline{X}$ either violates condition (c) or condition (d). Thus, the proposition is established. □

Observe that each of the conditions (a) through (e) in Proposition 20, can be expressed in MSO.

(a) "If $(i,j) \in \lambda_2(u)$ then $u \in X_i$ iff $u \in X_j$" is

$$\begin{aligned}\varphi_a(X_1, \ldots X_{w+1}) = \forall x \\ \textstyle\bigwedge_{i,j \in \{1,\ldots w+1\}} \bigwedge_{a=(\lambda_1,\lambda_2,\lambda_3):\ (i,j)\in\lambda_2} (Q_a x \to (X_i x \leftrightarrow X_j x))\end{aligned}$$

(b) "If $(i,j) \in \lambda_3(u)$ then $u \in X_i$ iff $v \in X_j$, where v is the parent of u" is

$$\begin{aligned}\varphi_b(X_1, \ldots X_{w+1}) = \forall x \forall y \\ \textstyle\bigwedge_{i,j \in \{1,\ldots w+1\}} \bigwedge_{a=(\lambda_1,\lambda_2,\lambda_3):\ (i,j)\in\lambda_3} ((Q_a x \wedge (S_1 yx \vee S_2 yx)) \to \\ (X_i x \leftrightarrow X_j y))\end{aligned}$$

(c) "If $u \in X_i$ and $u \in X_j$ then $(i,j) \in \lambda_2(u)$" is

$$\varphi_c(X_1, \ldots X_{w+1}) = \forall x ((X_i x \wedge X_j x) \to (\bigvee_{a=(\lambda_1,\lambda_2,\lambda_3):\ (i,j)\in\lambda_2} Q_a x))$$

(d) "If $u \in X_i$, $v \in X_j$ (where v is parent of u) then $(i,j) \in \lambda_3(u)$" is

$$\begin{aligned}\varphi_d(X_1, \ldots X_{w+1}) = \forall x \forall y ((X_i x \wedge X_j y \wedge (S_1 yx \vee S_2 yx)) \to \\ (\textstyle\bigvee_{a=(\lambda_1,\lambda_2,\lambda_3):\ (i,j)\in\lambda_3} Q_a x))\end{aligned}$$

(e) “ $\cup_{i=1}^{w+1} X_i$ forms a connected component of $\mathcal{T}_{G,\tau(G)}$” is

$$\varphi_e(X_1,\dots X_{w+1}) = \mathsf{connected}(\bigcup_i X_i)$$

where

$$\begin{array}{l}\mathsf{connected}(Y) = \forall Y_1 \forall Y_2 \\ ((\text{“}Y_1 \cup Y_2 = Y\text{''} \wedge \text{“}Y_1 \cap Y_2 = \emptyset\text{''} \wedge \text{“}Y_1 \neq \emptyset\text{''} \wedge \text{“}Y_2 \neq \emptyset\text{''}) \rightarrow \\ \qquad (\exists x \exists y Y_1 x \wedge Y_2 y \wedge (S_1 xy \vee S_2 xy \vee S_1 yx \vee S_2 yx))\end{array}$$

Armed with Proposition 20, and formulas $\varphi_a, \varphi_b, \varphi_c, \varphi_d, \varphi_e$ described above, we are to give the translation of sentence φ. Recall that to prove Courcelle's theorem, our goal is the following. Given graph G (with tree decomposition $\tau(G)$ of width w) and formula $\varphi(x_1,\dots x_n, Y_1,\dots Y_m)$ we will construct formula $T(\varphi)(\overline{X_1},\dots \overline{X_n}, \overline{Y_1},\dots \overline{Y_m})$ such that

$$\begin{array}{c} G \models \varphi[a_1,\dots a_n, Y_1, \dots Y_m] \\ \text{iff} \\ \mathcal{T}_{G,\tau(G)} \models T(\varphi)[\mathrm{Tree}(a_1),\dots \mathrm{Tree}(a_n), \mathrm{Tree}(Y_1),\dots \mathrm{Tree}(Y_m)]. \end{array}$$

$T(\varphi)$ will be constructed inductively.

Let start with the base cases.

- *Case* $\varphi = (x = y)$: $T(\varphi) = \text{“}\forall i.\, X_i = Y_i\text{''}$
- *Case* $\varphi = Exy$:
$$T(\varphi) = \exists u \bigvee_{a=(\lambda_1,\lambda_2,\lambda_3):\ (i,j)\in\lambda_1} (X_i u \wedge Y_j u \wedge Q_a u)$$
- *Case* $\varphi = Yx$: $T(\varphi) = \text{“}\forall i.\, X_i \subseteq Y_i\text{''}$

For the induction step, the boolean case of $\varphi = \psi_1 \rightarrow \psi_2$ is straightforward — $T(\varphi) = T(\psi_1) \rightarrow T(\psi_2)$. Let us consider the case of quantification.

- *Case* $\varphi = \exists x \psi$: $T(\varphi) = \exists \overline{X}(T(\psi) \wedge \varphi_a \wedge \varphi_b \wedge \varphi_c \wedge \varphi_d \wedge \varphi_e)$
- *Case* $\varphi = \exists Y \psi$: $T(\varphi) = \exists \overline{Y}(T(\psi) \wedge \varphi_a \wedge \varphi_b)$

5 Applications of Courcelle's Theorem

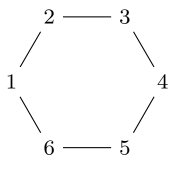
As observed in Section 4.1, *graph coloring*, *vertex cover*, and *independent set* can all be expressed in MSO over the signature of graphs. Therefore, as a consequence of Courcelle's Theorem (Theorem 16), all these problems can be solved efficiently on bounded tree width graphs.

Corollary 21. *Given graph $G = (V, E)$ of tree width w and k (parameter for colorability, vertex cover, and independent set), problem P (which is either graph coloring, vertex cover, or independent set) can be solved in time $O(f(w,k)|V|)$.*

5.1 Courcelle's Theorem for Arbitrary Structures

Another classical NP-complete problem is *Hamiltonian cycle* problem. Recall that a sequence of vertices $v_1, v_2, \dots v_n$ is a *Hamiltonian* cycle in graph $G = (V, E)$ iff (a) $(v_i, v_{i+1}) \in E$ for all $i \in \{1, \dots n-1\}$ and $(v_n, v_1) \in E$, and (b) for every vertex $v \in V$, there is a *unique* i such that $v = v_i$. The Hamiltonian cycle problem is, given a graph G, determine if G has a Hamiltonian cycle.

Unfortunately, the existence of a Hamiltonian cycle in a graph cannot be expressed in MSO over the signature $\tau_G = \{E\}$ of graphs. It can, however, be expressed over a richer vocabulary of hypergraphs



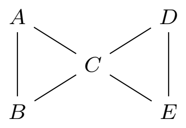


Figure 11: GraphG_1 has a Hamiltonian cycle while G_2 does not.

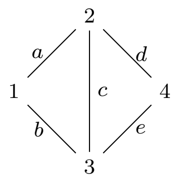


Figure 12: Example graph represented as a structure over $\tau_G = \{E\}$ and over $\tau_{HG} = \{V, E, I\}$.

— $\tau_{HG} = \{V, E, I\}$, where V and E are unary predicates denoting whether an element is a vertex or an edge, and I is a binary predicate encoding the incidence relation. Here is an example to illustrate the difference between encoding a graph over the signature $\tau_G = \{E\}$ and over the signature of hypergraphs $\tau_{HG} = \{V, E, I\}$.

Example 22. Consider graph shown in Figure 12. As a structure over $\tau_G = \{E\}$, the graph has universe $\{1, 2, 3, 4\}$, with relation $\{(1, 2), (1, 3), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (3, 4), (4, 2), (4, 1)\}$ being the interpretation of E.

On the other hand, if the graph is viewed as a structure of $\tau_{HG} = \{V, E, I\}$, the universe of the structure would be $\{1, 2, 3, 4, a, b, c, d, e\}$. The interpretation of the the predicates V, E, I will be $\{1, 2, 3, 4\}$, $\{a, b, c, d, e\}$, and $\{(1, a), (1, b), (2, a), (2, c), (2, d), (3, b), (3, c), (3, e), (4, d), (4, e)\}$, respectively.

Over the signature of hypergraphs, Hamiltonian cycle can be expressed in MSO as follows: "There is a set of edges H such that for every vertex there are exactly two edges in H incident on it and for every cut U, U' (i.e., partition of vertices) there is an edge that crosses the cut." This can be written down as

$$\begin{aligned}\varphi_H = \exists H((Hx \to Ex) \wedge \forall x(Vx \to \mathsf{exactly2}(x, H)) \wedge \\ \forall U \forall U'(\mathsf{part}(U, U') \to \exists x_1 \exists x_2 \exists y(Ux_1 \wedge U'x_2 \wedge Hy \wedge Ix_1y \wedge Ix_2y)))\end{aligned}$$

where

$$\begin{aligned}\mathsf{exactly2}(x, H) &= \exists y_1 \exists y_2(\neg(y_1 = y_2) \wedge Hy_1 \wedge Hy_2 \wedge Ixy_1 \wedge Ixy_2 \wedge \\ &\qquad \forall y((Hy \wedge Ixy) \to ((y = y_1) \vee (y = y_2)))) \\ \mathsf{part}(U_1, U_2) &= \forall x(((U_1x \vee U_2x) \to Vx) \wedge (Vx \to ((U_1x \leftrightarrow \neg U_2x))) \\ &\qquad \wedge((\exists x U_1x) \wedge (\exists x U_2x))\end{aligned}$$

Can the Hamiltonian cycle problem be solved efficiently on graphs of bounded tree width? We will answer this in the affirmative. To do this we will extend Courcelle's theorem for arbitrary structures as opposed to just graphs. To do this, we first extend the definitions of tree decompositions and tree width to arbitrary structures.

Definition 23 (Tree Decompositions of Structures)**.** A *tree decomposition* of a structure $\mathcal{A} = (A, R_1^{\mathcal{A}}, \ldots R_k^{\mathcal{A}})$ is a tree $\tau(\mathcal{A}) = (V_{\tau(\mathcal{A})}, E_{\tau(\mathcal{A})}, L_{\tau(\mathcal{A})})$ where $(V_{\tau(\mathcal{A})}, E_{\tau(\mathcal{A})})$ is a tree and $L_{\tau(\mathcal{A})} : V_{\tau(\mathcal{A})} \to 2^A$ is a labelling function such that

Universe Coverage For every $a \in A$, there is $t \in V_{\tau(\mathcal{A})}$ such that $a \in L_{\tau(\mathcal{A})}(t)$

Relation Coverage For every $(a_1, \ldots a_n) \in R_i^{\mathcal{A}}$, there is $t \in V_{\tau(\mathcal{A})}$ such that $\{a_1, \ldots a_n\} \subseteq L_{\tau(\mathcal{A})}(t)$

Coherence If $t_1, t_2 \in V_{\tau(\mathcal{A})}$ and $a \in A$ are such that $a \in L_{\tau(\mathcal{A})}(t_1) \cap L_{\tau(\mathcal{A})}(t_2)$ then $a \in L_{\tau(\mathcal{A})}(t)$ for every t on path from t_1 to t_2.

Definition 24 (Tree Width of Structures)**.** A structure $\mathcal{A}$ has *tree width* w if there is a tree decomposition $\tau(\mathcal{A}) = (V_{\tau(\mathcal{A})}, E_{\tau(\mathcal{A})}, L_{\tau(\mathcal{A})})$ such that for every vertex t, $|L_{\tau(\mathcal{A})}(t)| \leq w + 1$.

Bodlaender's theorem (Theorem 8) cn also be generalized to structures, and so a small binary tree decomposition of a structure can be computed efficiently. Courcelle's Theorem can be generalized as follows.

Theorem 25 (Courcelle)**.** *Given an MSO sentence φ over signature τ and a structure $\mathcal{A}$ of tree width w, $\mathcal{A} \models \varphi$ can be decided in time $O(f(|\varphi|, w)|\mathcal{A}|)$.*

Proof. The proof is similar to the the one for graphs. Based on the tree decomposition $\tau(\mathcal{A})$ we will construct a tree $\mathcal{T}_{\mathcal{A},\tau(\mathcal{A})}$ and an MSO sentence φ' such that $\mathcal{A} \models \varphi$ iff $\mathcal{T}_{\mathcal{A},\tau(\mathcal{A})} \models \varphi'$.

Using Bodlaeder's Theorem, we will first compute a small tree decomposition $\tau(\mathcal{A})$ of small width, and $\mathcal{T}_{\mathcal{A},\tau(\mathcal{A})}$ will be the same as $\tau(\mathcal{A})$ except in the labelling function, which now maps a vertex u to a tuple $(\lambda_1^1, \lambda_1^2, \ldots \lambda_1^k, \lambda_1, \lambda_3)$ (where $\tau = \{R_1, \ldots R_k\}$) such that

- $\lambda_2 = \{(i,j) \mid L_{\tau(\mathcal{A})}(u)[i,i] = L_{\tau(\mathcal{A})}(u)[j,j]$ and $i \neq j\}$
- $\lambda_3 = \{(i,j) \mid L_{\tau(\mathcal{A})}(u)[i,i] = L_{\tau(\mathcal{A})}(v)[j,j]\}$, where v is the parent of u in $\tau(\mathcal{A})$
- $\lambda_1^i = \{(i_1, \ldots i_n) \mid (L_{\tau(\mathcal{A})}(u)[i_1,i_1], \ldots L_{\tau(\mathcal{A})}(u)[i_n,i_n]) \in R_i^{\mathcal{A}}\}$

Sets of elements of A are encoded in the same manner as for graphs, and the translation of φ to φ' uses similar ideas. $\square$

Corollary 26. *The following problems can be solved in time $O(f(w)|V|)$ on graphs $G = (V, E)$ of tree width w.*

- *Determine if G has a Hamiltonian cycle.*
- *Determine if G is planar.*

Proof. Follows from Theorem **??** and the fact that these properties can be expressed in MSO over τ_{HG}. $\square$

The result about checking planarity can be generalized in the following manner.

Definition 27 (Edge Crossings)**.** The *crossing number* of a graph is the least number of edge crossings required to draw the graph in the plane such that at each point of the plane at most two edges cross.

Theorem 28. *Given a graph G of tree width w, and a number k, there is an algorithm that determines if the crossing number of G is $\leq k$, that runs in time $O(f(k, w)|G|)$.*

Proof. The result can be seen as a consequence of Courcelle's theorem. $\square$