**CS523: Advanced Algorithms** **Spring 2014**

# Lecture 1: TSP on graphs of bounded branch width

*Lecturer: Glencora Borradaile* *Scribes: Hung Le*

## 1.1 Branch Decomposition

Let $G = (V(G), E(G))$ be an undirected graph. We denote the number of vertices and edges of $G$ by $n$ and $m$, respectively. For a subset $X$ of vertices of $G$, let $G[X]$ be the subgraph induced by $X$.

A *branch decomposition* of $G$ is an unrooted binary tree $\mathcal{T}$ such that each leaf of $\mathcal{T}$ corresponds to an edge of $G$ and removing each edge of $\mathcal{T}$ partitions the edge set $E(G)$ into two parts. For an edge $e$ of $\mathcal{T}$, let $W_e$ be the set of vertices of $G$ such that for each $v \in W_e$, there are two edges $f_1$ and $f_2$ in $G$ incident to $v$ such that the leaves of $\mathcal{T}$ corresponding to $f_1$ and $f_2$ are in different component of $\mathcal{T}\backslash\{e\}$. $|W_e|$ is the *order* of the edge $e$. The maximum order over all edges of $\mathcal{T}$ is the *width* of $\mathcal{T}$. The *branch width* of $G$ is the minimum width over all possible branch decompositions of $G$.

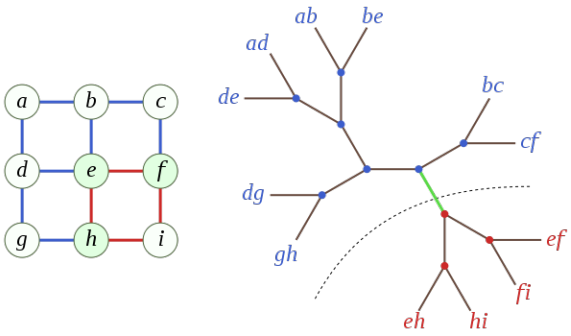


Figure 1.1: Branch decomposition (right ) of the graph (left)[1]

**Note:** It is NP-complete to decide whether a graph $G$ has a branch-decomposition of width at most $k$ [4]. For planar graphs, a minimum-width branch decomposition can be computed in $O(n^3)$ time [2].

[1]Source: `http://en.wikipedia.org/wiki/File:Branch-decomposition.svg`

## 1.2 Dynamic Programming for TSP

Suppose we are given a graph $G = (V(G), E(G))$ with costs on edges and a branch decomposition of $G$ of width $k$. We give a naive dynamic programming algorithm to solve TSP for $G$.

**Theorem 1.1** *There is an algorithm that solves TSP in graphs of branch-width at most $k$ in $O(k^3(2k)^{2k}n)$ time.*

We pick an arbitrary leaf vertex of $\mathcal{T}$ and make it become the root of $\mathcal{T}$. For an edge $e$ of $\mathcal{T}$, we call the partition of $\mathcal{T}\backslash\{e\}$ that does not contain the root of $\mathcal{T}$ the "lower" part of the tree. Let $G_e$ be the subgraph of $G$ containing edges corresponding to the vertices in the lower part $\mathcal{T}\backslash\{e\}$. We call the set $W_e$ the *interface* between $G_e$ and $G$. A *partial solution* of the TSP in $G_e$ is a collection of subgraphs, each with at least one vertex in the interface $W_e$ (see Figure 1.2). We define a labeling scheme $c_e : W_e \to \{0, 1, 2\}$ which assigns labels to vertices in $W_e$ such that for each vertex $v \in W_e$, we interpret as:

- $c_e(v) = 0$ if $v$ is not spanned by the partial solution of TSP in $G_e$.
- $c_e(v) = 1$ if $v$ has odd degree in the partial solution of TSP in $G_e$.
- $c_e(v) = 2$ if $v$ has even degree in the partial solution of TSP in $G_e$.

Let $w = |W_e|$. A *configuration* of the interface $W_e$ is a pair $(c_e, P_e)$ in which:

- $c_e$ is a labeling scheme of $W_e$
- $P_e = \{S_0, S_1, \ldots, S_w\}$ is a partition of vertices in $W_e$ in which $S_0$ is the set of vertices labeled 0. For convenience, we allow some sets $S_i, 0 \leq i \leq w$, of the partition $P_e$ to be empty . Each partition in $P_e$ keeps track of vertices of $W_e$ in the same connected components of the partial solution. (Refer to Figure 1.2(b).)

**Observation 1** *The number of vertices labeled* 1 *in each set of any partition is even.*

The Observation 1 comes from the fact that the solution of TSP is an Eulerian subgraph, therefore, its intersection with interfaces induces an even number of odd-labelled vertices.

**Lemma 1.2** *For an edge $e$ with interface $W_e$, the number of possible configurations is at most $w(2w)^w$ where $w = |W_e|$.*

**Proof:** We can think of a partition into $w + 1$ sets $\{S_0, S_1, S_2, \ldots, S_w\}$ of a set of $w$ vertices $\{v_1, v_2, \ldots, v_w\}$ as a coloring (it is possible that $S_i = \emptyset$ for some $i$). A vertex $v_i$ has a color $j$ if it is assigned to the partition $S_j$. For each vertex, there are $w + 1$ ways of assigning a color. Therefore, the number of possible partitions of $w$ is $w^{w+1}$. For each partition $S_i, i \neq 0$, there are $2^{|S_i|}$ different labellings, since vertices in $S_i$ can only be assigned labels in $\{1, 2\}$. Vertices in $S_0$ have labels 0 only. Hence, the number of possible configurations of $W_e$ is bounded by:

$$w^{w+1}2^{|S_1|+|S_2|+\ldots+|S_w|} \leq w(2w)^w \tag{1.1}$$

■

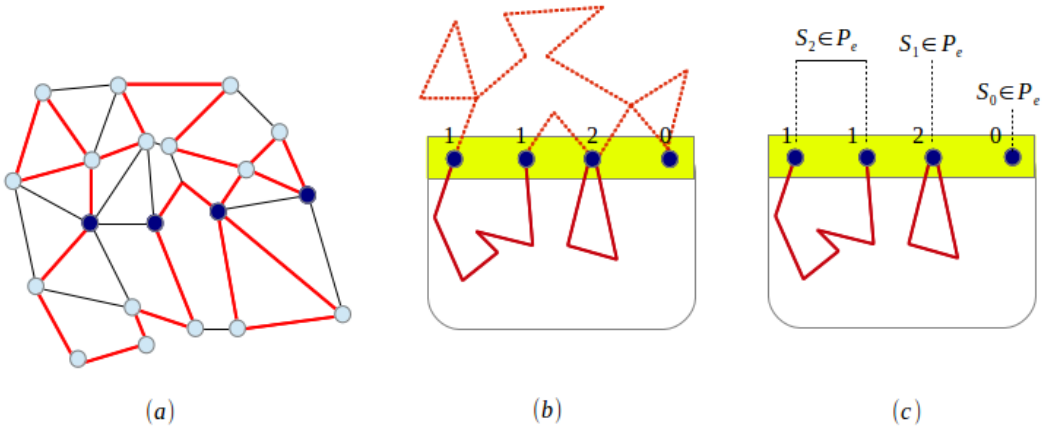


Figure 1.2: (a) The TSP solution are marked by red edges. Verices in the interface $W_e$ are blue. (b) A simplified view of interface and partial solution. (c) Partition and labelling of vertices of the interface $W_e$

A TSP tour $H$ is said to *induce* a configuration $(c_e, P_e)$ if $H \cap G_e$ satisfies the following conditions:

- For a vertex $v \in W_e$:
  - $c_e(v) = 0$ if the degree of $v$ in the subgraph $H \cap G_e$ is 0.
  - $c_e(v) = 1$ if the degree of $v$ in the subgraph $H \cap G_e$ is odd.
  - $c_e(v) = 2$ if the degree of $v$ in the subgraph $H \cap G_e$ is even.
- Vertices in the same set of the partition $P_e$ belong to the same connected component of $H \cap G_e$

Our dynamic programming table is indexed by edges of $\mathcal{T}$ and configurations associated with each edge. Precisely, $A_e[c_e, P_e]$ is the minimum cost subgraph of $G_e$ which induces the configuration $(c_e, P_e)$. We visit the tree $\mathcal{T}$ in post-order and update the dynamic programming table of a node given tables of its children.

**Leaf Edge of $\mathcal{T}$** Let $e$ be a leaf edge and let $(u, v)$ be a single edge in $G_e$. The interface $W_e$ consists of two vertices $W_e = \{u, v\}$. There are only two valid configurations $(c_e = [1, 1], P_e = \{\{\emptyset\}, \{u, v\}\})$ and $(c_e = [0, 0], P_e = \{\{u, v\}\})$. We have:

$$\begin{aligned} A_e[c_e = [1,1], P_e = \{\{\emptyset\}, \{u,v\}\}] &= cost(e(u,v)) \\ A_e[c_e = [0,0], P_e = \{\{u,v\}\}] &= 0 \\ A_e[c_e, P_e] &= +\infty \qquad \text{for all other configurations} \end{aligned} \tag{1.2}$$

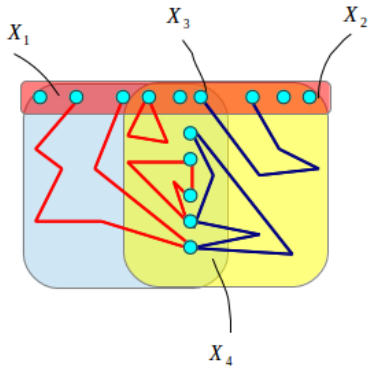


Figure 1.3: A possible combination of partitions. The red edges are in the partial solution restricted to $G_{e_1}$ and the blue edges are in the partial solution restricted to $G_{e_2}$

**Internal Edge of $\mathcal{T}$** Let $e$ be an internal edge with two children $e_1$ and $e_2$. Initially, we set $A_e[c_e, P_e] = +\infty$ for all possible configurations of $W_e$, and then combine two tables $A_{e_1}$ and $A_{e_2}$ to update $A_e$. Let $X_1 = W_e - W_{e_2}$, $X_2 = W_e - W_{e_1}$, $X_3 = W_e \cap W_{e_1} \cap W_{e_2}$ and $X_4 = W_{e_1} \cap W_{e_2} - W_e$ (Refer to Figure 1.3).

**Observation 2**

$$\begin{aligned} W_e &= X_1 \cup X_2 \cup X_3 \\ W_{e_1} &= X_1 \cup X_3 \cup X_4 \\ W_{e_2} &= X_2 \cup X_3 \cup X_4 \end{aligned} \tag{1.3}$$

Let $(c_1, P_1)$ and $(c_2, P_2)$ be two configurations associated with interfaces $W_{e_1}$ and $W_{e_2}$, respectively and let $(c_e, P_e)$ be a configuration of $W_e$ which is the combination of $(c_1, P_1)$ and $(c_2, P_2)$. Note that not all pairs of configurations $(c_1, P_1)$ and $(c_2, P_2)$ can be combined. If they can, we say $(c_1, P_1)$ *compatible* with $(c_2, P_2)$. One example of a pair of incompatible configurations is when merging two partitions $P_1$ and $P_2$ results in an *isolated* connected component, which is a connected component containing no vertex of $W_e$. We define compatibility more formally by showing how to merge two partitions $P_1$ and $P_2$. Initially, we set $P = \emptyset$. For any $u \in W_e$, we have following *color assignment rule*:

- if $u \in X_1$, then $c(u) = c_1(u)$
- if $u \in X_2$, then $c(u) = c_2(u)$
- if $u \in X_3$ and
  - if $c_1(u) = c_2(u) \neq 0$, then $c(u) = 2$
  - if $c_1(u) = 1 \wedge c_2(u) = 2$ or $c_1(u) = 2 \wedge c_2(u) = 1$, then $c(u) = 1$
  - otherwise, $c(u) = max\{c_1(u), c_2(u)\}$

For two partitions $P_1 = \{S_0^1, S_1^1, \ldots, S_w^1\}$ and $P_2 = \{S_0^2, S_1^2, \ldots, S_w^2\}$, we build a bipartite graph $G_b = A \cup B$ in which $A$ is a set of $w$ vertices corresponding to $w$ sets $\{S_1^1, \ldots, S_w^1\}$ of $P_1$ and $B$ is

a set of $w$ vertices corresponding to $w$ sets $\{S_1^2, \ldots, S_w^2\}$ of $P_2$. Two vertices $u_i \in A$ and $v_j \in B$ have an edge if the two corresponding sets $S_i^1$ and $S_j^2$ satisfy $S_i^1 \cap S_j^2 \neq \emptyset$. We decompose $G_b$ into a collection of connected components $\{C_k\}$ and sets from $P_1$ and $P_2$ in each component $C_k$ will be merged into a single set of $P_e$. Consider a pair of two sets $S_i^1$ and $S_j^2$ of a component $C_k$ such that $S_i^1 \cap S_j^2 \neq \emptyset$ and let $v$ be a vertex in $S_i^1 \cap S_j^2$. If $v \in X_4$ and $c_1(v) \neq c_2(v)$, then $P_1$ and $P_2$ are not compatible. Two sets $S_0^1$ and $S_0^2$ in $P_1$ and $P_2$ are rather special and we need to treat them differently. For $S_0^1$, if $S_0^1 \cap S_t^2 \neq \emptyset$ for some $t \in \{1, 2, \ldots, w\}$, we just include vertices in $S_0^1 \cap S_t^2$ to the partition of $P_e$ which contains $S_t^2$ and assign colors following color assignment rule above. For $S_0^2$, we assign colors and merge vertices in $S_0^2$ similarly. The vertices in $S_0^1 \cap S_0^2$ that have not been assigned to any set of $P_e$ will be $S_0$ of $P_e$. Finally, we check that combination of two partitions $(c_1, P_1)$ and $(c_2, P_2)$ does not lead to an isolated component, that can be done by checking whether a new set in $P_e$ formed by merging sets in $P_1$ and $P_2$ containing a vertex in $W_e$ or not. Clearly, building bipartite graph, finding connected components and checking compatibility can be done in $O(w)$ time.

If $(c_1, P_1)$ and $(c_2, P_2)$ are compatible, we update:

$$A_e[c_e, P_e] = min(A_e[c_e, P_e], A_{e_1}[c_1, P_1] + A_{e_2}[c_2, P_2])$$

**Obtaining Solution** For the root edge $r$, let $(u, v)$ be the only edge of $G \backslash G_r$. The minimum cost of TSP is given by

$$cost(TSP) = min(A[c_r = [2, 2], P_r = \{\emptyset, \{u, v\}\}], A[c_r = [1, 1], P_r = \{\emptyset, \{u, v\}\}] + cost(e(u, v)))$$

**Proof of Theorem 1.1** For each pair of configurations $(c_1, P_1)$ and $(c_2, P_2)$, checking the compatibility and update $(c, P)$ takes at most $O(w)$ time. There are at most $O(w^2(2w)^{2w})$ possible pairs of configurations, therefore, for each edge $e$ of $\mathcal{T}$, the time required to update the dynamic programming table is $O(w^3(2w)^{2w})$. Since $w \leq k$ for all interfaces $W_e$ and the size of $\mathcal{T}$ is $O(n)$, the total running time of the algorithm is bounded by $O(k^3(2k)^{2k}n)$

**TSP for planar graphs of bounded branch width** If the given graph is planar, we can exploit planarity to speed up the dynamic programming algorithm. Since the graph is planar, the partition $P$ in the configuration $(c, P)$ is *non-crossing* [1]. The number of non-crossing partitions from a set of $n$ points arranged around a circle is the Catalan number:

$$C_n = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{n^{\frac{3}{2}}\sqrt{\pi}}$$

Therefore, the number of possible configurations in the Lemma 1.2 is at most:

$$O(w\frac{4^w}{w^{\frac{3}{2}}\sqrt{\pi}}) \sim O(\frac{4^w}{\sqrt{w}}) \tag{1.4}$$

and the total running time of the algorithm is reduced to $O(k\left(\frac{4^k}{\sqrt{k}}\right)^2 n) = O(16^k n)$. By more clever combination, we are able to further improve the running time of the algorithm, see [1].

# References

[1] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Algorithms ESA 2005*, volume 3669 of *Lecture Notes in Computer Science*, pages 95–106. Springer Berlin Heidelberg, 2005.

[2] Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 373–384. Springer Berlin Heidelberg, 2008.

[3] Neil Robertson and P.D Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153 – 190, 1991.

[4] P. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.