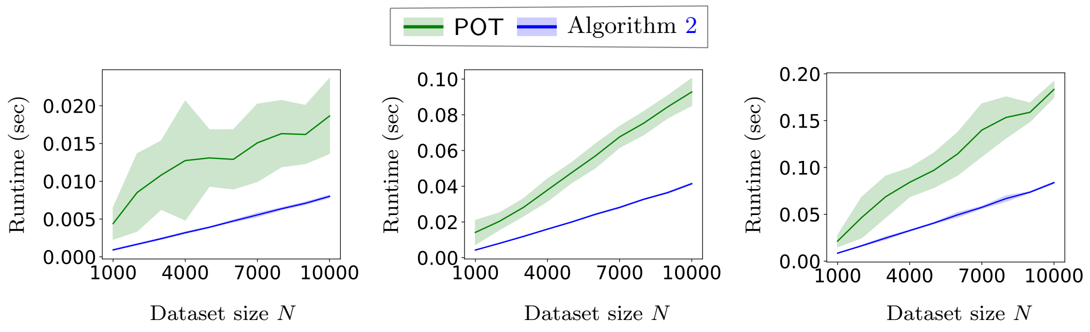
Appendix A. Comparison with POT

The subgradient of Sliced Wasserstein distance can be computed using the POT library.[9] Here, we compare the runtime of Algorithm 2 with POT. We performed the runtime comparison using the point cloud datasets with the dataset sizes ranging from 1000 to 10000. We also varied the number of slices for 10, 50, and 100. Figure 7 shows that the runtime of Algorithm 2 is smaller than POT, regardless of the dataset sizes and the number of slices.



(a) # of slices = 10 (b) # of slices = 50 (c) # of slices = 100

Figure 7: Comparison of runtimes for subgradient computation of Sliced Wasserstein distance using Algorithm 2 and POT.

We also conducted the runtime comparison using the synthetic dataset in Section 6.1. We varied the dataset size from 1000 to 100000, and we fixed the number of slices to 100. Figure 8 shows that Algorithm 2 is again faster than POT.

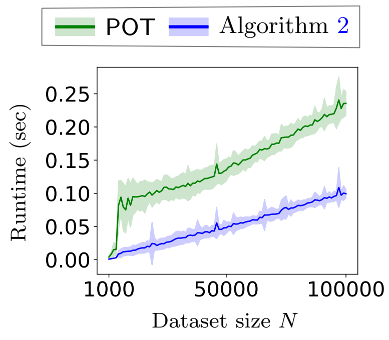


Figure 8: Runtime comparison on the synthetic dataset.

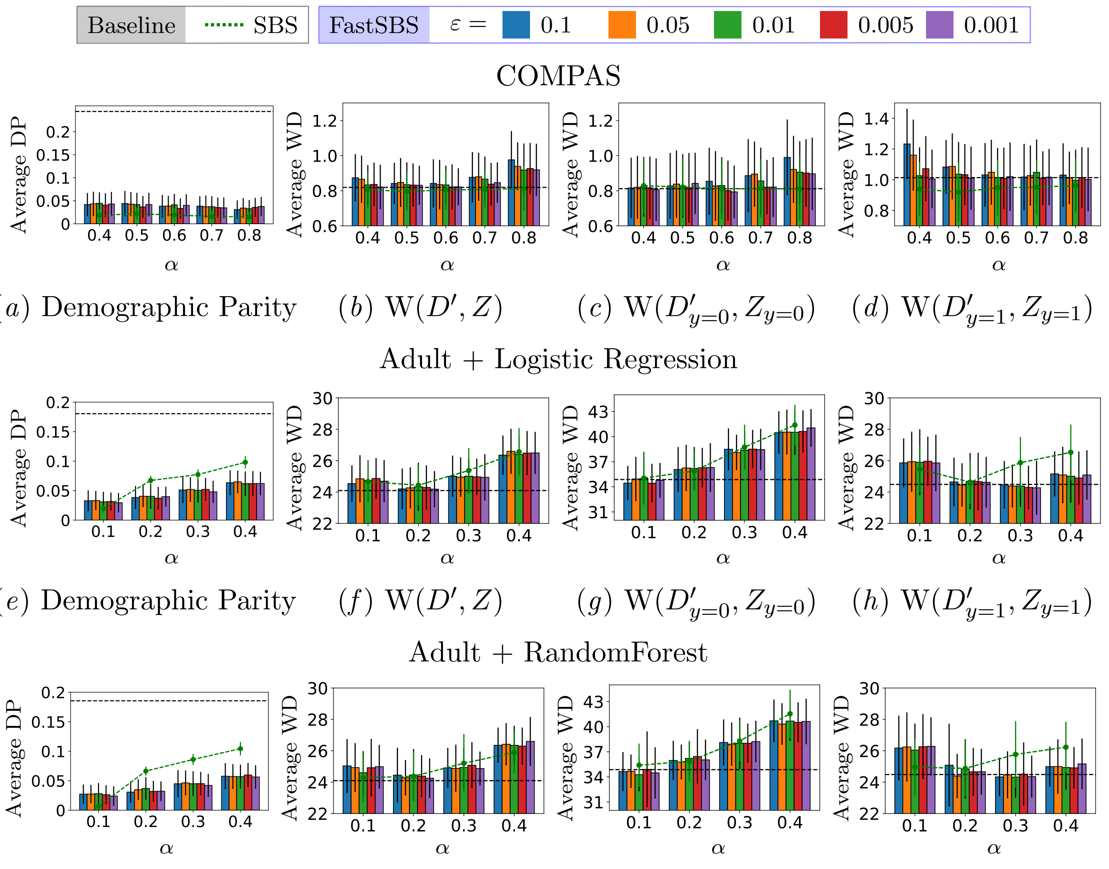
9. https://pythonot.github.io/

Appendix B. Additional Experimental Results

Here, we show the runtime comparisons in Section 6.3 on the COMPAS and Adult datasets.

B.1. The effects of ε

Figure 9 shows the results on the COMPAS and Adult dataset for each method over 10 runs for several different choice of ε. The figures show that the choice of ε does not have any significant impact on the quality of the benchmark Z.



(a) Demographic Parity (b) $\mathrm{W}(D', Z)$ (c) $\mathrm{W}(D'_{y=0}, Z_{y=0})$ (d) $\mathrm{W}(D'_{y=1}, Z_{y=1})$

(e) Demographic Parity (f) $\mathrm{W}(D', Z)$ (g) $\mathrm{W}(D'_{y=0}, Z_{y=0})$ (h) $\mathrm{W}(D'_{y=1}, Z_{y=1})$

(i) Demographic Parity (j) $\mathrm{W}(D', Z)$ (k) $\mathrm{W}(D'_{y=0}, Z_{y=0})$ (l) $\mathrm{W}(D'_{y=1}, Z_{y=1})$

Figure 9: The effects of ε on FastSBS on the COMPAS and Adult dataset experiments.

B.2. The effects of S

Figure 10 shows the results on the COMPAS and Adult dataset for each method over 10 runs for several different choice of S. The figures show that the choice of S does not have any significant impact on the quality of the benchmark Z.

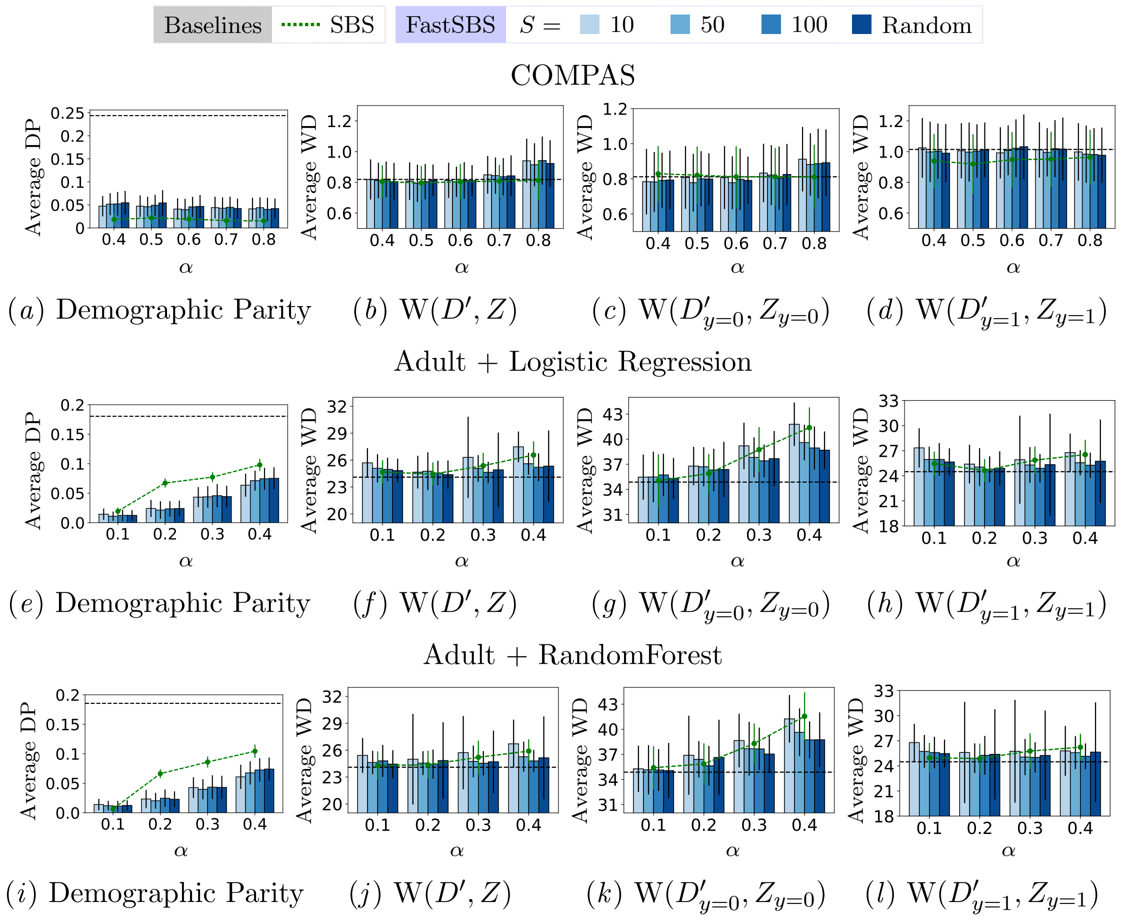


Figure 10: The effects of S on FastSBS on the COMPAS and Adult dataset experiments.

Appendix C. FastSBS with Tree-Sliced Wasserstein Distance

In FastSBS, we used Sliced Wasserstein distance for efficient computation. Here, we consider another variant of FastSBS that uses Tree-Sliced Wasserstein distance (Le et al., 2019).

C.1. Tree-Sliced Wasserstein Distance

Tree Wasserstein distance (Le et al., 2019) represents the distance between distributions on a given tree structure. Let $T = (V, E)$ be a tree with a set of nodes V and a set of edges E. Here, we consider a tree T consisting of N_{in} internal nodes and N leaf nodes so that $|V| = N_{\text{in}} + N$. We assume that each instance of the dataset D corresponds to each leaf node of T.

We use the following notations for the tree. Let $\texttt{leaf}(i)$ be a set of leaf nodes in the subtree rooted at the node i defined by $\texttt{leaf}(i) = \{j \mid j \text{ is the leaf of the subtree rooted at } i\}$. We also define the set of ancestor nodes by $\texttt{ans}(j) = \{i \mid j \in \texttt{leaf}(i)\}$ and the parent node of i by $\texttt{pa}(i)$.

Given the tree T, Tree Wasserstein distance is defined as follows:

$$\mathrm{W}^T(\mu, \nu) = \sum_{i \in V} C_{\texttt{pa}(i), i} |\mu(\texttt{leaf}(i)) - \nu(\texttt{leaf}(i))|, \tag{19}$$

where $\mu(\texttt{leaf}(i)) = \sum_{j \in \texttt{leaf}(i)} \mu_j$ and $\nu(\texttt{leaf}(i)) = \sum_{j \in \texttt{leaf}(i)} \nu_j$. The cost $C_{\texttt{pa}(i),i} = \|x_{\texttt{pa}(i)} - x_i\|$ where $x_{\texttt{pa}(i)}$ is the position of the parent node. No that the one-dimensional Wasserstein distance is a special case of Tree Wasserstein distance.

To define Tree Wasserstein distance in the Euclidean space, one needs to construct a tree T from the dataset D. Typical methods for constructing the tree include hierarchical clustering (Johnson, 1967), farthest-point clustering (Gonzalez, 1985), and the Quadtree algorithm (Shaffer and Samet, 1987). Once the tree T is constructed, Tree Wasserstein distance can be computed in $O(|V|) = O(N_{\text{in}} + N)$ time (Le et al., 2019).

Tree-Sliced Wasserstein distance is defined as the expectation of Tree Wasserstein distances over randomly generated trees:

$$\mathrm{TSW}(\mu, \nu) = \mathbb{E}_T[\mathrm{W}^T(\mu, \nu)]. \tag{20}$$

C.2. FastSBS with Tree-Sliced Wasserstein Distance

Here, we consider FastSBS$_{\text{Tree}}$, a variant of FastSBS that uses Tree-Sliced Wasserstein distance instead of Sliced Wasserstein distance, which is formulated as follows:

$$(\text{FastSBS}_{\text{Tree}}) \quad \min_{\mu} \mathrm{TSW}(\mu, \nu), \quad \text{s.t.} \quad \mu \in P(k),. \tag{21}$$

For optimization, we can use Algorithm 1 also for FastSBS$_{\text{Tree}}$ once we can computed an unbiased subgradient of $\mathrm{TSW}(\mu, \nu)$. Here, a subgradient of the Tree-Wasserstein distance for a randomly generated tree T is an unbiased subgradient of the Tree-Sliced Wasserstein distance. Thus, it is sufficient to compute the subgradient of $\mathrm{W}^T(\mu, \nu)$ for a given T.

Takezawa et al. (2021) has shown that one can rewrite the Tree-Wasserstein distance $\mathrm{W}^T(\mu, \nu)$ in (19) as

$$\mathrm{W}^T(\mu, \nu) = \|\mathrm{diag}(w) B(\mu - \nu)\|_1, \tag{22}$$

Algorithm 3: Subroutines

```
Procedure Rec$_\Delta(\mu, \nu \in \mathbb{R}_+^N, \mathsf{node})$        ▷ Compute $\Delta_{\mu,\nu}(\texttt{leaf}(i))$ for all the nodes.
    if node is leaf then node.$\Delta \leftarrow \mu_{\mathsf{node}} - \nu_{\mathsf{node}}$ ;
    else
        node.$\Delta \leftarrow 0$
        for $i \in$ {child of node} do node.$\Delta \leftarrow$ node.$\Delta + \mathrm{Rec}_\Delta(\mu, \nu, i)$ ;
    end
    return node.$\Delta$
Procedure Rec$_g(\delta \in \mathbb{R}, \mathsf{node})$        ▷ Compute $g_i$ for all the leaf nodes.
    if node is leaf then node.$g \leftarrow \delta + C_{\mathsf{pa(node),node}} \times \mathrm{sign}[\mathsf{node}.\Delta]$ ;
    else
        for $i \in$ {child of node} do Rec$_g(\delta + C_{\mathsf{pa(node),node}} \times \mathrm{sign}[\mathsf{node}.\Delta], i)$ ;
    end
```

Algorithm 4: Computing the subgradient g in (24)

```
Procedure Subgrad_Tree$(\mu, \nu \in \mathbb{R}_+^N, T)$
    Rec$_\Delta(\mu, \nu, T.\mathsf{root})$ ;        ▷ $T$.root denotes the root node of $T$.
    Rec$_g(0, T.\mathsf{root})$ ;
return $g \leftarrow (\mathsf{node}.g \mid \mathsf{node} \in \texttt{leaf}(T.\mathsf{root}))$
```

where $w \in \mathbb{R}^{N_{\text{in}}+N}$ with $w_i = 0$ if i is the root of T and $w_i = C_{\mathtt{pa}(i),i}$ otherwise, and

$$B = \begin{bmatrix} (I - D_1)^{-1} D_2 \\ I \end{bmatrix}, \quad \begin{array}{ll} D_1 \in \mathbb{R}^{N_{\text{in}} \times N_{\text{in}}}, & (D_1)_{ij} = \mathbb{I}[\text{node } j \text{ is a child of node } i], \\ D_2 \in \mathbb{R}^{N_{\text{in}} \times N}, & (D_2)_{k\ell} = \mathbb{I}[\text{node } \ell \text{ is a leaf of node } k]. \end{array}$$

Here, the indicator function $\mathbb{I}$ returns 1 if the condition is satisfied, and 0 otherwise.

By the chain rule, the subgradient of (22) with respect to μ is given by

$$g = B^\top \mathrm{diag}(w)\, \mathrm{sign}\left[\mathrm{diag}(w) B(\mu - \nu)\right], \tag{23}$$

where $\mathrm{sign}[\cdot]$ is an element-wise sign function that, for a vector $a \in \mathbb{R}^N$, returns $(\mathrm{sign}[a])_i = 1$ if $a_i \geq 0$ and $(\mathrm{sign}[a])_i = -1$ otherwise. While Takezawa et al. (2021) computed the subgradient (23) by explicitly constructing the matrix B, we show that we can compute the subgradient more efficiently. Let us recall that $(D_1^k)_{ij} = 1$ if and only if the node j is reachable from the node i in k-hops. Hence, $((I - D_1)^{-1})_{ij} = (\sum_{k=0}^\infty D_1^k)_{ij} = 1$ indicates that the node j is reachable from the node i. Because we are considering a tree, this is equivalent to the fact that the node j is a descendant of the node i in the tree. We can then conclude that $((I - D_1)^{-1} D_2)_{ij} = 1$ if and only if the leaf node j belongs to the subtree rooted at the node i.

We rewrite the subgradient g by using the property of B above. First, we have

$$\mathrm{sign}\left[w_i \left(B(\mu - \nu)\right)_i\right] = \mathrm{sign}\left[C_{\mathtt{pa}(i),i} \Delta_{\mu,\nu}(\texttt{leaf}(i))\right] = \mathrm{sign}\left[\Delta_{\mu,\nu}(\texttt{leaf}(i))\right],$$

where $\Delta_{\mu,\nu}(\texttt{leaf}(i)) = \sum_{k \in \texttt{leaf}(i)} (\mu_k - \nu_k)$ and because $C_{\mathtt{pa}(i),i} \geq 0$. We then have

$$g_j = \sum_{i \in \texttt{ans}(j)} C_{\mathtt{pa}(i),i}\, \mathrm{sign}\left[\Delta_{\mu,\nu}(\texttt{leaf}(i))\right]. \tag{24}$$

Algorithm 4 computes the subgradient (24) by using recursions. In the algorithm, we first compute $\Delta_{\mu,\nu}(\texttt{leaf}(i))$ for all the node i in the tree by using the recursion

$$\Delta_{\mu,\nu}(\texttt{leaf}(i)) = \sum_{i':\text{child of } i} \Delta_{\mu,\nu}(\texttt{leaf}(i')).$$

We then compute (24) by using the recursion

$$\begin{aligned}\sum_{i \in \texttt{ans}(j)} C_{\texttt{pa}(i),i} \operatorname{sign}\left[\Delta_{\mu,\nu}(\texttt{leaf}(i))\right] = &\sum_{i \in \texttt{ans}(\texttt{pa}(j))} C_{\texttt{pa}(i),i} \operatorname{sign}\left[\Delta_{\mu,\nu}(\texttt{leaf}(i))\right] \\ &+ C_{\texttt{pa}(j),j} \operatorname{sign}\left[\Delta_{\mu,\nu}(\texttt{leaf}(j))\right].\end{aligned}$$

Because Algorithm 4 visits each node only twice, once in Rec_Δ and once in Rec_g, its time complexity is $O(N_{\mathrm{in}} + N)$, which is typically $\tilde{O}(N)$ for a balanced tree.

C.3. Experiments

Here, we show the experimental results on $\mathrm{FastSBS}_{\mathrm{Tree}}$. We found that the quality of the benchmark created by $\mathrm{FastSBS}_{\mathrm{Tree}}$ were comparable with the ones of FastSBS. However, $\mathrm{FastSBS}_{\mathrm{Tree}}$ tends to take longer time than FastSBS because the tree generation is more costly than the projections. We therefore concluded that FastSBS would be more practical because of its smaller runtime.

For $\mathrm{FastSBS}_{\mathrm{Tree}}$, we set the number of fixed slices $S = 10$ in our heuristics. We randomly generated trees using the farthest-point clustering (Gonzalez, 1985). When constructing a tree, we set two hyper-parameters, the depth of the tree b and the number of child nodes c. In each node of the tree, we split the set of instances belonging to the node to c child nodes based on the splitting criterion of the farthest-point clustering. We repeat the procedure until the tree depth reaches b. We set the position of the child nodes as the average of the instances belonging to that node. It takes $O(Nbc)$ time to construct a tree from the dataset of size N. In the experiments, we tried a few candidates of (b, c) and adopted the one that performed well. For the synthetic, COMPAS, and Adult datasets, we set $(b, c) = (5, 3), (5, 5)$, and $(5, 7)$, respectively. The other experimental settings were the same with Sections 6.1 and 6.2.

Results Figure 11 shows the results on the synthetic, COMPAS, and Adult datasets. The figures show that, for both requirements (R1) and (R2), $\mathrm{FastSBS}_{\mathrm{Tree}}$ performed comparable with SBS and FastSBS. Thus, $\mathrm{FastSBS}_{\mathrm{Tree}}$ could successfully generate fake benchmark as expected.

Figures 12 and 13 show the results when we varied the stopping threshold ε and the number of fixed slices S. The figure indicates that the runtime of $\mathrm{FastSBS}_{\mathrm{Tree}}$ is significantly faster than that of SBS for any $S \leq 100$. The results of Random of $\mathrm{FastSBS}_{\mathrm{Tree}}$ is exceptional who randomly generates a tree in every iteration of the optimization. The results on the Adult dataset show that $\mathrm{FastSBS}_{\mathrm{Tree}}$ with Random tends to be slower than the original SBS. This finding shows that the generation of the trees is a dominating step if we naively implement $\mathrm{FastSBS}_{\mathrm{Tree}}$. Thus, the use of the proposed heuristic would be essential to speedup $\mathrm{FastSBS}_{\mathrm{Tree}}$.

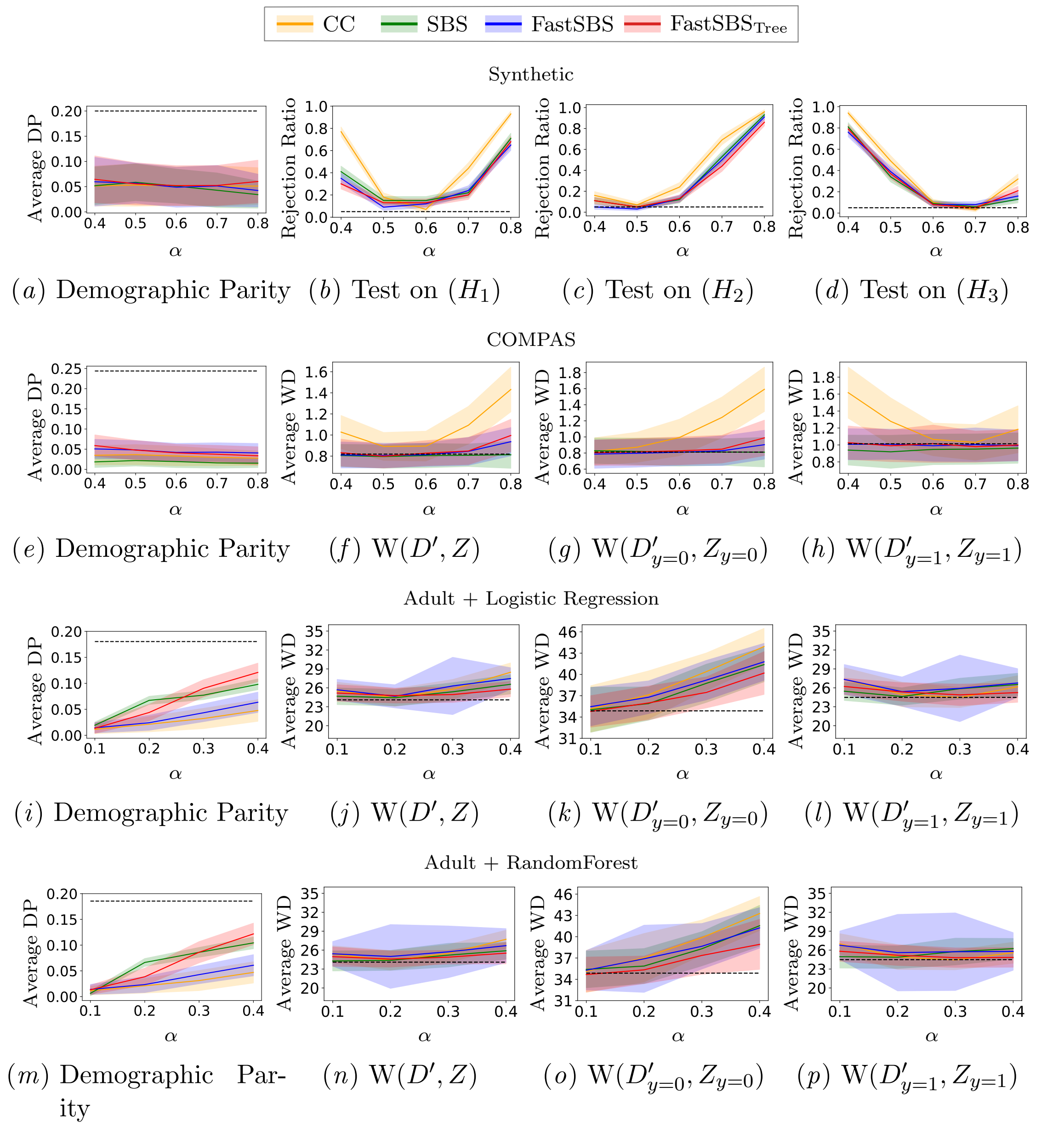


Figure 11: The results on the synthetic dataset (a)–(d), COMPAS (e)–(h), Adult + Logistic Regression (i)–(l), and Adult + RandomForest (m)–(p). The solid lines represent the average over 100 runs, and the shaded areas represent the standard deviation. The black dashed lines are the same as those in Figures 1 and 2.

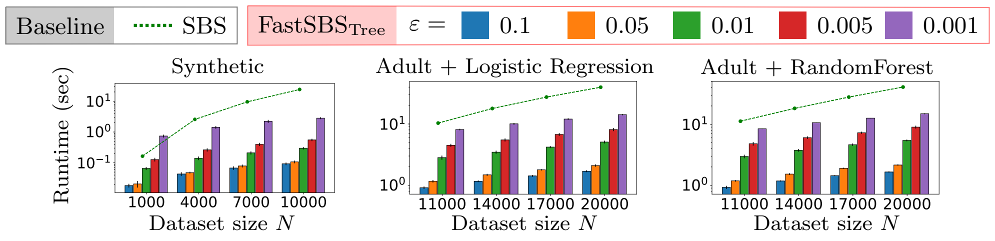


Figure 12: Runtimes of FastSBS$_{\text{Tree}}$ for difference choice of ε on the synthetic and Adult datasets.

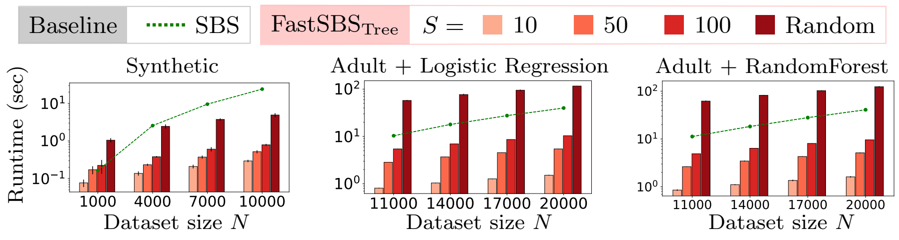


Figure 13: Runtimes of FastSBS$_{\text{Tree}}$ for difference choice of S on the synthetic and Adult datasets.