## A. Computing Case 2 Probabilities

We compute $Pr^h(o \wedge C2)$ for all objects $o \in O$ together as a preprocessing step, which is executed only once for each dataset $O$.

In a possible world $pw \in C2$, all the objects are on the same line. Thus, two arbitrary objects $o_a, o_b \in O$ must occur on the line, and therefore this line can only be one of the $m_a \times m_b$ possible lines defined by an instance of $o_a$ and an instance of $o_b$. To minimize the number of lines to check, we choose $a = \arg\min_i\{m_i\}$ and $b = \arg\min_{i \neq a}\{m_i\}$, i.e. $o_a$ (and $o_b$) is the object with the smallest (and second smallest) number of instances.

For each line $\overline{s_a s_b}$ (i.e. under the condition that $s_a$ and $s_b$ occur), we check all the other objects $o_i \in O - \{o_a, o_b\}$ one by one, and meanwhile maintain a pool of key-value pairs.

Assume that we have already checked object instances $s_1 \in o_1$, $s_2 \in o_2$, ..., $s_{i-1} \in o_{i-1}$, then each key-value pair is composed of:

- Key $<s_\ell, s_r>$ that records the leftmost (and upmost) and the rightmost (and lowest) endpoints of $s_1, s_2, \ldots, s_{i-1}$. Note that $s_\ell$ and $s_r$ are instances.
- Value $Pr\{\bigwedge_{t=1}^{i-1}\{s_t \wedge onLine(\overline{s_a s_b}, s_t)\} \mid s_a, s_b\}$, where $onLine(\overline{s_a s_b}, s_t)$ refers to the event that $s_t$ lies on line $\overline{s_a s_b}$.

Initially, there is only one pair $(<s_a, s_b>, 1)$ or $(<s_b, s_a>, 1)$ in the pool. We perform the following operations when checking object $o_i$:

- As long as object $o_i$ is found to have no instance on $\overline{s_a s_b}$, we know that it is impossible for all the objects to be on $\overline{s_a s_b}$, and stop checking $\overline{s_a s_b}$.
- Otherwise, for each $s_i \in o_i$ on $\overline{s_a s_b}$, we expand each pair in the pool by (1) updating the key if $s_i$ becomes a new endpoint, and (2) multiplying the old value by $Pr(s_i)$ to incorporate the event $onLine(\overline{s_a s_b}, s_i)$ into the probability.

In fact, we can merge those pairs with the same key $<s_\ell, s_r>$ after checking each object, by summing up their values, so as to keep the pool size small. This modification, in effect, changes the meaning of the value of a key-value pair to $Pr\{\bigwedge_{t=1}^{i-1}\{onLine(\overline{s_a s_b}, s_t)\} \wedge s_\ell, s_r \text{ are endpoints} \mid s_a, s_b\}$.

Therefore, after all the objects $o_i \in O - \{o_a, o_b\}$ are checked to have instances on $\overline{s_a s_b}$, each pair with key $<s_\ell, s_r>$ has value equal to $Pr\{\text{all objects are collinear} \wedge s_\ell, s_r \text{ are endpoints} \mid s_a, s_b\}$, or equivalently, $Pr\{s_\ell, s_r \text{ are endpoints} \wedge C2 \mid s_a, s_b\}$.

Note that when $s_\ell, s_r$ are endpoints, objects $o_\ell$ and $o_r$ are on the convex hull. Therefore, given the pairs in the pools obtained after all lines are checked, we can compute $Pr^h(o_j \wedge C2)$ by the following formula:

$$Pr^h(o_j \wedge C2) = \left[\sum_{s_a \in o_a} \sum_{s_b \in o_b} \left(\sum_{\ell=j} + \sum_{r=j}\right)\right] Pr(s_a) \cdot$$
$$Pr(s_b) \cdot Pr\{s_\ell, s_r \text{ are endpoints} \wedge C2 \mid s_a, s_b\}. \quad (23)$$

The above equation is formulated according to the additivity of probability, since the events $\{s_a, s_b \text{ occur} \wedge \text{ all objects are on } \overline{s_a s_b} \text{ with endpoints } s_\ell, s_r\}$ are disjoint.

The algorithm of computing $Pr^h(o \wedge C2)$ may take time exponential to the data size in the worst case (e.g. when all instances of all objects are on the same line). Fortunately, for the majority of the real world data, it is very unlikely that all objects are collinear. Line checking usually stops after examining a few objects $o_i \in O - \{o_a, o_b\}$.

Furthermore, in Section IV-B, we will present an *object-level four-corner pruning* technique, and show that as long as the method prunes some object, $Pr^h(o \wedge C2)$ are guaranteed to be 0 for all $o \in O$, and the preprocessing step described here does not even need to be executed.

## B. CCW Indicator for MBRs

We extend the concept of ccw indicator to deal with the *minimum bounding rectangle* (MBR) of a set of points, which is used in our algorithms, especially for R-tree node pruning.

*Definition 5:* Given two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, and an MBR $M$ with four vertices $a$, $b$, $c$ and $d$, the *ccw indicator* of $p_1$, $p_2$ and $M$ is defined as follows:

- $ccw(p_1, p_2, M) = 1$, if $ccw(p_1, p_2, v) > 0$ for all $v \in \{a, b, c, d\}$. We denote this case as $ccw(\overrightarrow{p_1 p_2}, M)$.
- $ccw(p_1, p_2, M) = -1$, if $ccw(p_1, p_2, v) < 0$ for all $v \in \{a, b, c, d\}$. We denote this case as $cw(\overrightarrow{p_1 p_2}, M)$.
- Otherwise, $ccw(p_1, p_2, M) = 0$, which we denote as $cut(\overrightarrow{p_1 p_2}, M)$.

Figure 6(b) illustrates the idea of Definition 5: for an MBR $M = \square abcd$ totally contained in the upper half-plane, $ccw(p_1, p_2, M) = 1 > 0$; for an MBR $M = \square a'b'c'd'$ totally contained in the lower half-plane, $ccw(p_1, p_2, M) = -1 < 0$; for an MBR $M = \square a''b''c''d''$ intersecting with line $\overline{p_1 p_2}$, $ccw(p_1, p_2, M) = 0$.

Therefore, we can determine whether $ccw(\overrightarrow{p_1 p_2}, M)$, $cw(\overrightarrow{p_1 p_2}, M)$ or $cut(\overrightarrow{p_1 p_2}, M)$ by checking the sign of the ccw indicator.

## C. R-Tree Traversal Operations

Let us denote the MBR of node $\mathcal{N}$ as $\mathcal{N}.M$. Following are several important details during the traversal of $T_O$:

- We can prune a node $\mathcal{N}$ if we find $ccw(\overrightarrow{s_j s_i}, \mathcal{N}.M) \wedge ccw(\overrightarrow{s_i s_k}, \mathcal{N}.M)$, since $\forall o \in \mathcal{N}$, $V(o)$-full holds.
- If we traverse to a node $\mathcal{N}$ satisfying $cw(\overrightarrow{s_j s_i}, \mathcal{N}.M) \vee cw(\overrightarrow{s_i s_k}, \mathcal{N}.M)$, then $\forall o \in \mathcal{N}$, $V(o)$-empty holds. Thus, we terminate and return $p(\widehat{s_j s_i s_k}) = 0$.
- When we reach a leaf node, we filter out entries $o_i.M$, $o_j.M$ and $o_k.M$, since we only consider $o_t \in O - \{o_i, o_j, o_k\}$ in Equations (8) and (9).

We traverse $aR_{o_t}$ to compute the result $\sum_{s_t \in V(o_t)} Pr(s_t)$ (or $|V(o_t)|$), which is initialized to 0. Following are several important details when traversing $aR_{o_t}$:

- If we traverse to a node $\mathcal{N}$ satisfying $ccw(\overrightarrow{s_j s_i}, \mathcal{N}.M) \wedge ccw(\overrightarrow{s_i s_k}, \mathcal{N}.M)$, then $\forall s_t \in \mathcal{N}$, $s_t \in V(o_t)$. Thus, we directly add the aggregate value of $\mathcal{N}$ to the result.
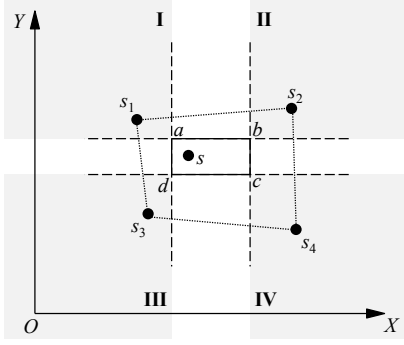
Fig. 15. Proof of Theorem 2

- If we traverse to a node $\mathcal{N}$ satisfying $cw(\overrightarrow{s_j s_i}, \mathcal{N}.M) \vee cw(\overrightarrow{s_i s_k}, \mathcal{N}.M)$, we prune $\mathcal{N}$ since $\forall s_t \in \mathcal{N}$, $s_t \in \overline{V(o_t)}$.
- When we reach a leaf node and process instance $s_t$, we add $Pr(s_t)$ (or 1 when all instances of any object are equally likely to occur) to the result if we find $s_t \in V(o_t)$ after checking Equation (7).

Note that there may still be an object $o_t \in C$ such that $V(o_t)$-empty holds, since we only check $o_t.M$ when traversing $T_O$, and we terminate and return $p(\widehat{s_j s_i s_k}) = 0$ once we find such an object $o_t$ after traversing $aR_{o_t}$.

### D. Proof of Theorem 2

*Proof:* Given an object $o$, its MBR $o.M$ determines four regions as shown in Figure 15. Suppose that there exists four objects $o_1$, $o_2$, $o_3$ and $o_4$, such that $o_1.M$, $o_2.M$, $o_3.M$ and $o_4.M$ are totally contained in Regions I, II, III, IV, respectively. Then, in any possible world $pw$, $s_1$, $s_2$, $s_3$ and $s_4$ are totally contained in Regions I, II, III, IV, respectively.

We now prove that the convex hull in $pw$ is a polygon that contains $o.M$, so that any instance $s \in o$ is not on the convex hull. Due to the arbitrary of $pw$, we would have $Pr^h(o) = 0$.

Let us define $O' = \{o_1, o_2, o_3, o_4, o\}$. Since $O' \subseteq O$, the convex hull of $O'$ in $pw$ is a polygon that is contained in the polygon defined by the convex hull of $O$ in $pw$. Therefore, it is sufficient if we can prove that the polygon defined by the convex hull of $O'$ in $pw$ contains $o.M$.

We now prove that the convex hull of $O'$ in $pw$, i.e. polygon $s_1 s_2 s_3 s_4$ in Figure 15, contains $o.M$, i.e. rectangle $abcd$ in Figure 15. Without loss of generality, we only need to prove that polygon edge $s_1 s_2$ is above $o.M$, or equivalently, above rectangle edge $ab$. This holds because both $s_1.y$ and $s_2.y$ are at least $y_2$, which accomplishes the proof.

The case of instance-level four-corner pruning can be similarly proved. ∎

### E. Containment Query Operation

Our four-corner pruning operation is based on four range queries on the object R-tree $T_O$. We call such range queries the *containment queries*, which return whether there exists an object whose MBR is totally contained in the query region defined by query object $o_q$ or query instance $s_q$.

We now illustrate how a *containment query* with Region I $= \{(x, y) \in \mathbb{R}^2 | x \leq x_1 \wedge y \geq y_2\}$ works. The most important details are listed below:

- During the traversal of $T_O$, if a node $\mathcal{N}$ has MBR $\mathcal{N}.M$ totally contained in region $R_1 = \{(x, y) \in \mathbb{R}^2 | x < x_1 \wedge y > y_2\}$, then $\forall o \in \mathcal{N}$, we have $o \neq o_q$ and $o.M$ is totally contained in $R_1 \subset$ Region I. Thus, we terminate the *containment query* immediately and return a positive answer.
- When we reach a leaf node and check whether entry $o.M$ is totally contained in Region I, we need to filter out object $o_q$.

As long as one of the four *containment queries* gives a negative answer, we can return a negative answer immediately without evaluating the remaining queries.

### F. Operations for Upper Bounding

In order to make $UB^h(s)$ tight, for each Region $t \in \{$I, II, III, IV$\}$, we choose objects $o_t$ to maximize $\sum_{s_t \in \text{Region } t} Pr(s_t)$ (or $n_t/m_t$ when all the instances of any object are equally likely to occur), which is achieved by the following three steps:

1) Perform a range query on the object R-tree $T_O$ with Region $t$ as the query window, to find the set of (intersecting) objects $C_t(s) = \{o_r \in O - \{o\} | o_r.M \cap \text{Region } t \neq \emptyset\}$;
2) For each object $o_r \in C_t(s)$, compute $\sum_{s_r \in \text{Region } t} Pr(s_r)$ (or $n_r/m_r$) by performing a range query on the instance aR-tree $aR_{o_r}$ with Region $t$ as the query window;
3) Return the maximum of the computed values.

Given query instance $s = (x, y) \in o$, we now illustrate how a *range query* with Region I$= \{p \in \mathbb{R}^2 | p.x \leq x \wedge p.y \geq y\}$ works (Step 1). The most important details are listed below:

- During the traversal of $T_O$, if a node $\mathcal{N}$ has MBR $\mathcal{N}.M$ totally contained in region $R_1 = \{p \in \mathbb{R}^2 | p.x < x \wedge p.y > y\}$, then $\forall o_r \in \mathcal{N}$, we have $o_r \neq o$ and $o_r$ is totally contained in $R_1 \subset$ Region I. Therefore, we return $\max_t\{\sum_{s_t \in \text{Region } t} Pr(s_t)\} = 1$ (or $\max_t\{n_t/m_t\} = 1$) immediately without going to the $2^{nd}$ and $3^{rd}$ steps.
- Suppose we traverse to a node $\mathcal{N}$ whose MBR $\mathcal{N}.M$ has lower-left and upper-right corners $(x_1, y_1)$ and $(x_2, y_2)$, repetively. If $x_1 > x$ or $y_2 < y$, we prune $\mathcal{N}$ since it is totally outside of Region I.
- When we reach a leaf node, we need to filter out the object $o$ of query instance $s$, as well as those objects already picked when processing other regions.

The last operation is necessary because we require $o_I$, $o_{II}$, $o_{III}$ and $o_{IV}$ to be different in Equation (15).

After obtaining $C_t(s)$ in the first step, we perform a range query on $aR_{o_r}$ for each $o_r \in C_t(s)$ to compute $\sum_{s_r \in \text{Region } t} Pr(s_r)$ or $n_r/m_r$. This operation is similar to that of computing Equations (10) and (11) in Section IV-A (where $V(o)$ is involved instead of $C_t(s)$), except that only coordinate comparison is required here instead of *ccw indicator* evaluation.
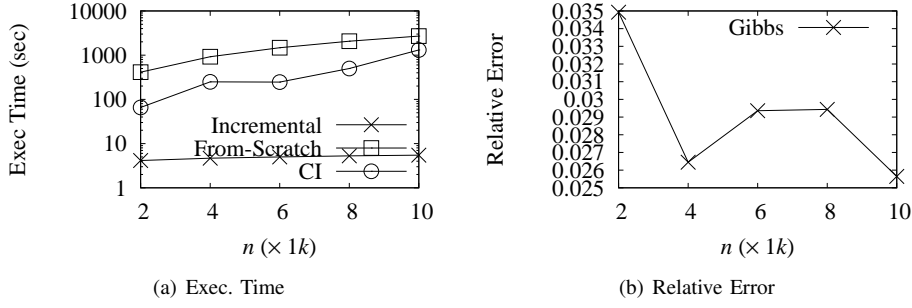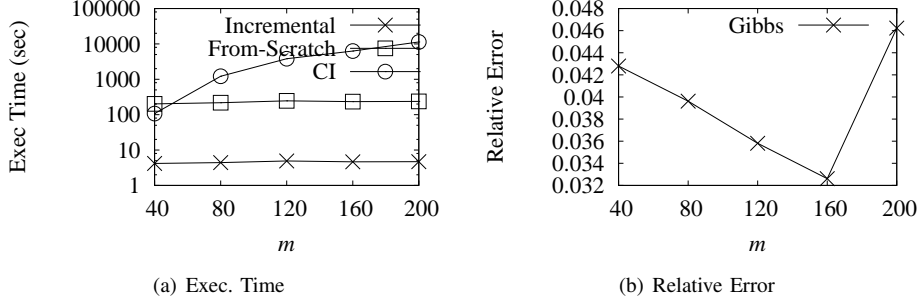
(a) Exec. Time

(b) Relative Error

Fig. 16.  Effect of $n$ on Gibbs Sampling



(a) Exec. Time

(b) Relative Error

Fig. 17.  Effect of $m$ on Gibbs Sampling



(a) Exec. Time
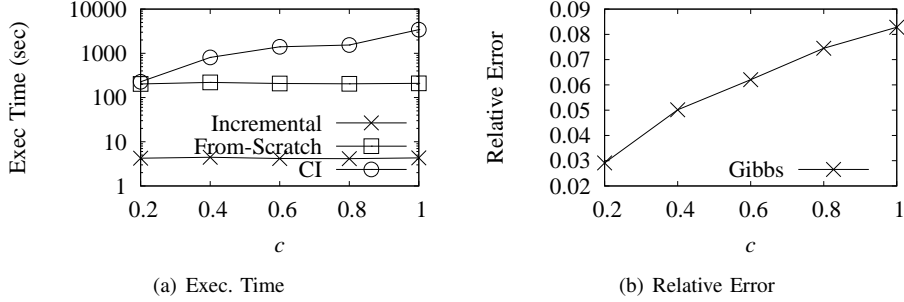
(b) Relative Error

Fig. 18.  Effect of $c$ on Gibbs Sampling

### G. Scalability Results of Gibbs Sampling

We now present the results of our Gibbs sampling algorithms. Besides the data parameter $(n, m, c, \alpha)$, we have another parameter $k$ indicating the number of samples. To evaluate the accuracy of Gibbs sampling, let $\widehat{Pr^h(o)}$ be the value of $Pr^h(o)$ estimated by our Gibbs sampling method, then we define the relative error as

$$|\widehat{Pr^h(o)} - Pr^h(o)|/\max\{Pr^h(o), \delta\}. \quad (24)$$

Note that if $Pr^h(o) \geq \delta$, Equation (24) is exactly the relative error in the traditional sense. We define the relative error as such to avoid the influence of objects with very small $Pr^h(o)$ in error evaluation. For example, consider an object $o$ with $Pr^h(o) = 10^{-10}$, and suppose that it happens to appear on the convex hull in one sample of totally $k = 10^6$ samples. Then, the traditional relative error is $(10^{-6} - 10^{-10})/10^{-10} \approx 10^4$ which is quite large. However, this is due to the small sample problem (only one occurrence) and an object with $Pr^h(o) = 10^{-10}$ is uninteresting. By using Equation (24) and setting $\delta = 0.1\%$, the error contribution of that object is mitigated as $(10^{-6} - 10^{-10})/10^{-3} \approx 10^{-3}$.

We set $\alpha = \varepsilon$ and $\delta = 0.1\%$ in all our experiments (i.e., we are only interested in objects $o$ with $Pr^h(o) \geq 0.1\%$), and the reported relative error is averaged over all non-pruned objects.

Experiments show that the relative error decreases as $k$ increases. For example, when $(n, m, c) = (10^4, 20, 0.2)$, the relative error is 0.06 when $k = 10^5$, 0.026 when $k = 5 \times 10^5$, 0.019 when $k = 10^6$ and 0.007 when $k = 10^7$.

Figure 16(a) (and (b)) shows the running time (and accuracy) of our algorithms when $(m, c, k) = (20, 0.2, 5 \times 10^5)$ and $n$ varies, where we denote the Gibbs sampling method that dynamically maintains the convex hull by *Incremental*, and the one that computes it from scratch by *From-Scratch*. Figure 16(a) shows that *Incremental* always finishes in several seconds, which is orders of magnitude faster than *From-Scratch* and $\mathcal{CI}$. Figure 16(b) shows that the accuracy of Gibbs sampling increases as $n$ increases.

Figure 17(a) (and (b)) shows the running time (and accu-

racy) of our algorithms when $(n, c, k) = (1000, 0.2, 5 \times 10^5)$ and $m$ varies. Figure 17(a) shows that *Incremental* always finishes in several seconds, and is orders of magnitude faster than *From-Scratch* and $\mathcal{CI}$. Figure 17(b) shows that the accuracy of Gibbs sampling decreases as $m$ increases.

Figure 18(a) (and (b)) shows the running time (and accuracy) of our algorithms when $(n, m, k) = (1000, 40, 5 \times 10^5)$ and $c$ varies. Figure 18(a) shows that *Incremental* always finishes in several seconds, and is orders of magnitude faster than *From-Scratch* and $\mathcal{CI}$. Figure 18(b) shows that the accuracy of Gibbs sampling decreases as $c$ increases.

Note that in Figure 16(a), *From-Sketch* is even more expensive than the exact algorithm $\mathcal{CI}$. However, as $m$ and $c$ becomes larger, $\mathcal{CI}$ becomes much more expensive than *From-Sketch*, as shown in Figure 17(a) and Figure 18(a). Since *Incremental* always stops in several seconds and gives accurate probability estimations (the relative error is much smaller than 5%), it is able to support real-time applications such as animal tracking and analysis. On the other hand, *From-Sketch* is a baseline for comparison that is much more expensive than *Incremental*, but returns the same result as *Incremental*, and thus we should never use it.