Lecture 5:
CS 6306 / INFO 6306:
Advanced Human Computation

Database Perspective

- Class time will remain Tu/Th 11:40-12:55
- Mutual introductions
  - Name
  - What you do
  - What interests you in HC
- Assignment 1: Due Thursday, 15 September 2016
  - Do at least 25 HITs
  - Create at least one HIT that is done by at least 5 workers
- Selecting presenter for next two Thursdays, 15 and 22 September 2016
  - Game theoretical approaches to human computation
  - The human experience in human computation

Week of Sept 20

- Required readings:
  - Martin, D., Hanrahan, B.V., O'Neill, J. and Gupta, N., 2014. "Being a turker." In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*(pp. 224-235). ACM.
  - Irani, L.C. and Silberman, M., 2013. "Turkopticon: interrupting worker invisibility in Amazon Mechanical Turk." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 611-620). ACM.
- Additional readings:
  - Brawley, A.M. and Pury, C.L., 2016. "Work experiences on MTurk: Job satisfaction, turnover, and information sharing." *Computers in Human Behavior*, *54*, pp.531-546.
  - Gray, M.L., Suri, S., Ali, S.S. and Kulkarni, D., 2016. "The crowd is a collaborative network." In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (pp. 134-147). ACM.
  - Gupta, N., Martin, D., Hanrahan, B.V. and O'Neill, J., 2014. "Turk-life in India." In *Proceedings of the 18th International Conference on Supporting Group Work* (pp. 1-11). ACM.
  - Lee, M.K., Kusbit, D., Metsky, E. and Dabbish, L., 2015. "Working with machines: The impact of algorithmic and data-driven management on human workers." In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 1603-1612). ACM.
  - McInnis, B., Cosley, D., Nam, C. and Leshed, G., 2016. "Taking a HIT: Designing around Rejection, Mistrust, Risk, and Workers' Experiences in Amazon Mechanical Turk." In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (pp. 2271-2282). ACM.
  - Salehi, N., Irani, L.C., Bernstein, M.S., Alkhatib, A., Ogbe, E. and Milland, K., 2015. "We are dynamo: Overcoming stalling and friction in collective action for crowd workers." In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 1621-1630). ACM.

Today: Database Perspective

- Required Readings:
  - Franklin, M.J., Kossmann, D., Kraska, T., Ramesh, S. and Xin, R., 2011. "CrowdDB: answering queries with crowdsourcing." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*(pp. 61-72). ACM.
  - Marcus, A., Wu, E., Karger, D.R., Madden, S. and Miller, R.C., 2011. "Crowdsourced databases: Query processing with people." In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR 2011)*.
- Additional Readings:
  - Davidson, S.B., Khanna, S., Milo, T. and Roy, S., 2013. "Using the crowd for top-k and group-by queries." In *Proceedings of the 16th International Conference on Database Theory* (pp. 225-236). ACM.
  - Guo, S., Parameswaran, A. and Garcia-Molina, H., 2012. "So who won?: dynamic max discovery with the crowd." In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 385-396). ACM.
  - Marcus, A., Wu, E., Karger, D., Madden, S. and Miller, R., 2011. "Human-powered sorts and joins." *Proceedings of VLDB*, *5*(1), pp.13-24.
  - Parameswaran, A.G., Park, H., Garcia-Molina, H., Polyzotis, N. and Widom, J., 2012. "Deco: declarative crowdsourcing." In *Proceedings of the 21st ACM international conference on Information and knowledge management* (pp. 1203-1212). ACM.
  - Wang, J., Kraska, T., Franklin, M.J. and Feng, J., 2012. "Crowder: Crowdsourcing entity resolution." *Proceedings of VLDB*,*5*(11), pp.1483-1494.5

CrowdDB

- Use a crowd to overcome limitations of relational databases
- Example:
  - Closed-world assumption:
    - SELECT market_capitalization FROM company WHERE name = "I.B.M.“;
    - Crowd can find items not currently in DB
      (normally would give empty result if I.B.M. is not in the DB)
  - Entity resolution:
    - I.B.M. vs IBM vs International Business Machines
  - Questions that rely on human judgment:
    - SELECT image FROM picture WHERE topic = "Business Success" ORDER BY relevance LIMIT 1;
    - Relevance can be assessed by crowd
- Provides a “clean” way to think about crowds, by relating it to known data operations

CrowdDB: CrowdSQL

```
CREATE TABLE Department (
        university STRING,
        name STRING,
        url CROWD STRING,
        phone STRING
        PRIMARY KEY (university, name) );
```

CrowdDB: CrowdSQL

```
CREATE TABLE Department (
        university STRING,
        name STRING,
        url CROWD STRING,
        phone STRING,
        PRIMARY KEY (university, name) );
```

CrowdDB: CrowdSQL

```
CREATE CROWD TABLE Professor (
        name STRING PRIMARY KEY,
        email STRING UNIQUE,
        university STRING,
        department STRING,
        FOREIGN KEY (university, department) REF Department(university, name) );
```

CrowdDB: CrowdSQL

```
CREATE CROWD TABLE Professor (
        name STRING PRIMARY KEY,
        email STRING UNIQUE,
        university STRING,
        department STRING,
        FOREIGN KEY (university, department) REF Department(university, name) );
```

CrowdDB: CrowdSQL

```
SELECT profile FROM department
        WHERE name ~= "CS";


CREATE TABLE picture
        ( p IMAGE, subject STRING );
SELECT p FROM picture
        WHERE subject = "Golden Gate Bridge“
            ORDER BY CROWDORDER(p, "Which picture visualizes better %subject");
```

CrowdDB: CrowdSQL

```
SELECT profile FROM department
        WHERE name ~= "CS";


CREATE TABLE picture
        ( p IMAGE, subject STRING );
SELECT p FROM picture
        WHERE subject = "Golden Gate Bridge“
            ORDER BY CROWDORDER(p, "Which picture visualizes better %subject");
```

CrowdDB: CrowdSQL

- Other operators
  - LIMIT: max amount of money
  - CrowdProbe: ask workers for answers until a majority is reached
  - CrowdJoin: Computes the join over two tables one of which is crowdsourced
  - CrowdCompare: implements CROWDEQUAL and CROWDORDER

CrowdDB: Assessment

- “Micro Benchmarks” – simple tasks involving filling in missing data
- Time to completion
  - Tradeoff between time to completion and completion percent
- Payment levels
  - Completion rate
  - Answer quality

CrowdDB: Issues

- When do crowds get called?

CrowdDB: Interfaces

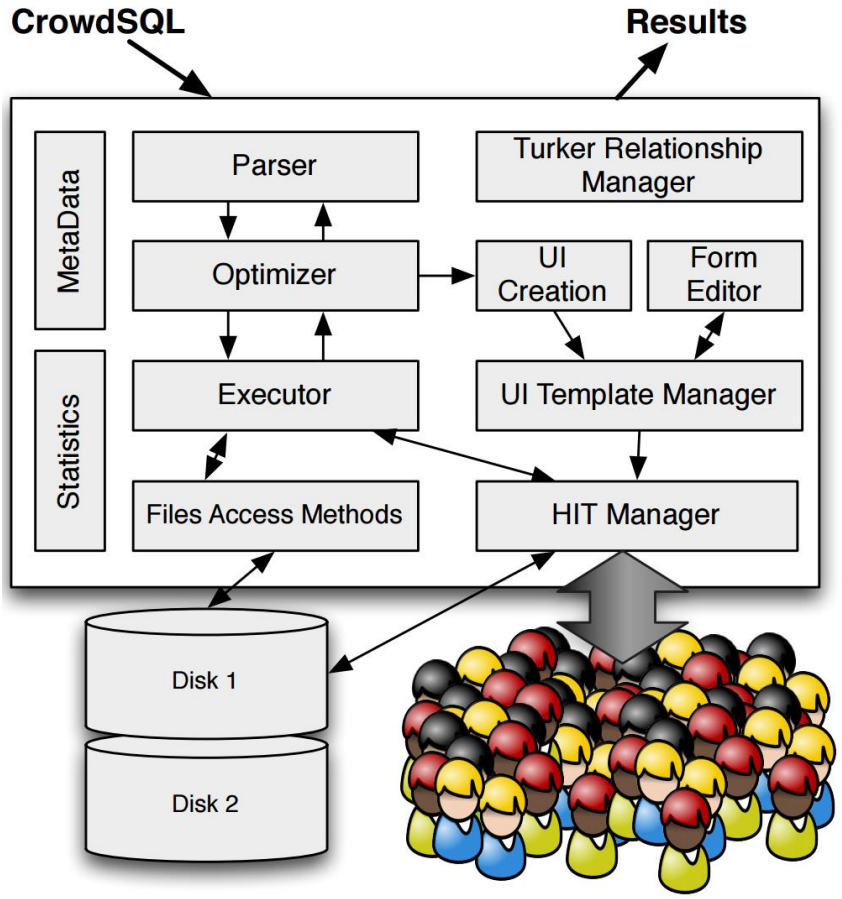
- Known field values are copied over, null values prompt asking user
- Uses javascript to check that responses are of the correct datatype

CrowdDB: Issues

- “If a query involves a CROWD table, then it is unclear how many tuples need to be crowdsourced in order to fully process the query.”
- “In order to be practical, CrowdSQL should provide a way to define a budget for a query”
- Built-in quality testing

CrowdDB: Issues

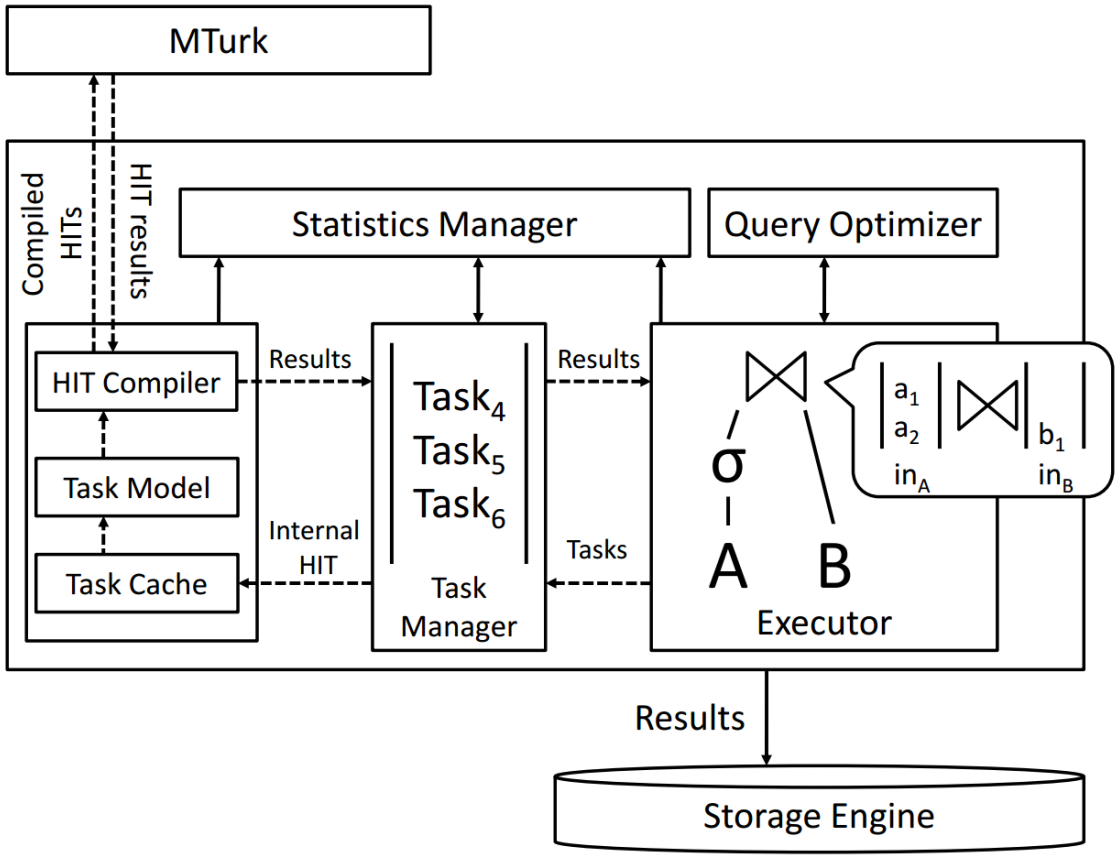
- Architecture:



Qurk
("Crowdsourced Databases: Query Processing with People")

- Crowds as user-defined functions
- Motivations:
  - HITs take a (relatively) long time
  - Query optimization should include financial cost and accuracy
  - Can't know ahead of time characteristics of operators necessary for optimization
- Example tasks:
  - Return CEO and contact number for each company in a table
  - "Join" a table of disaster victim photos with submitted photos from family members
  - Return the sentiment of each tweet in a table of Twitter tweets
  - Rank the items in a table of products using Amazon reviews

Qurk: Architecture



Qurk: Filter Example

```
SELECT * FROM images WHERE isFlower(img)

TASK isFlower(Image img) RETURN Bool:
        TaskType: Question
        Text: ‘‘Does this image: <img src=‘%s’>
                contain a flower?’’, URLify(img)
        Response: Choice(‘‘YES’’,‘‘NO’’)
```

Qurk: Filter Example

SELECT * FROM images WHERE isFlower(img)

TASK isFlower(Image img) RETURN Bool:

TaskType: Question

Text: ‘‘Does this image: <img src=‘%s’>
contain a flower?’’, URLify(img)

Response: Choice(‘‘YES’’,‘‘NO’’)

Qurk: Get Values Example

```
SELECT companyName, findCEO(companyName).CEO,
        findCEO(companyName).Phone
FROM companies


TASK findCEO(String companyName)
RETURNS (String CEO,String Phone):
        TaskType: Question
        Text: ‘‘Find the CEO and the CEO’s phone
                number for the company %s’’, companyName
        Response: Form((‘‘CEO’’,String), (‘‘Phone’’,String))
```

Qurk: Get Values Example

```
SELECT companyName, findCEO(companyName).CEO,
        findCEO(companyName).Phone
FROM companies
```

```
TASK findCEO(String companyName)
RETURNS (String CEO,String Phone):
        TaskType: Question
        Text: ‘‘Find the CEO and the CEO’s phone
                number for the company %s’’, companyName
        Response: Form((‘‘CEO’’,String), (‘‘Phone’’,String))
```

Qurk: Ranking Example

```
SELECT productID, productName FROM products
ORDER BY rankProducts(productName)
```

```
TASK rankProducts(String[] prodNames) RETURNS String[]:
        TaskType: Rank
        Text: ‘‘Sort the following list of products
                by their user reviews on Amazon.com’’
        List: prodNames
```

Qurk: Ranking Example

```
SELECT productID, productName FROM products
ORDER BY rankProducts(productName)
```

```
TASK rankProducts(String[] prodNames) RETURNS String[]:
        TaskType: Rank
        Text: ‘‘Sort the following list of products
                by their user reviews on Amazon.com’’
        List: prodNames
```

Qurk: Join Example

```
SELECT survivors.location, survivors.name
FROM survivors, missing
WHERE imgContains(survivors.image, missing.image)


TASK imgContains(Image[] survivors, Image[] missing)
RETURNS Bool:
        TaskType: JoinPredicate
        Text: “Drag a picture of any <b>Survivors</b>
              in the left column to their matching
              picture in the <b>Missing People</b>
              column to the right.”
Response: DragColumns("Survivors", survivors,
                      "Missing People", missing)
```

Qurk: Join Example

```
SELECT survivors.location, survivors.name
FROM survivors, missing
WHERE imgContains(survivors.image, missing.image)


TASK imgContains(Image[] survivors, Image[] missing)
RETURNS Bool:
        TaskType: JoinPredicate
        Text: “Drag a picture of any <b>Survivors</b>
              in the left column to their matching
              picture in the <b>Missing People</b>
              column to the right.”
Response: DragColumns("Survivors", survivors,
                      "Missing People", missing)
```

Qurk: Features

- Results are multi-valued to reflect that different workers might give different answers
- “convenience functions” (example: majorityVote) to collapse values to a single value

Qurk: Query Optimization

- Use Qurk query annotations:
  - maxCost: maximum $ willing to pay
  - minConfidence: minimum number of workers who must agree
  - maxLatency: how long willing to wait on a HIT
  - NecessaryConditions statement (example: photos must have same race and gender)
- Optimizations that can be made:
  - Adjust pricing at runtime
  - Uniformly sample input table
  - Combine multiple Tasks into single HITs
  - Operator implementaions (example: rank by comparison vs scores)
  - TurkIt-like caching
  - Replace large set of HITs with fewer HITS and apply machine learning

Qurk: Issues

Qurk: Issues


I THINK YOU SHOULD BE MORE SPECIFIC HERE IN STEP TWO
THEN A MIRACLE OCCURS

Qurk
("Human-powered Sorts and Joins")

- Describes (some of the) implementation
- Additional syntax element: POSSIBLY
- Evaluated performance on simple joins and sorts and their combination
  - Sort: Tried different human computation algorithms, different UIs
  - SELECT name, scene.img
    FROM actors JOIN scenes
        ON inScene(actors.img, scenes.img)
        AND POSSIBLY numInScene(scenes.img) > 1
    ORDER BY name, quality(scenes.img)

Qurk
("Human-powered Sorts and Joins")

- Describes (some of the) implementation
- Additional syntax element: POSSIBLY
- Evaluated performance on simple joins and sorts and their combination
  - Sort: Tried different human computation algorithms, different UIs
  - SELECT name, scene.img
    FROM actors JOIN scenes
        ON inScene(actors.img, scenes.img)
        AND POSSIBLY numInScene(scenes.img) > 1
    ORDER BY name, quality(scenes.img)

Qurk
("Human-powered Sorts and Joins")

Operator	Optimization	# HITs
Join	Filter	43
Join	Filter + Simple	628
Join	Filter + Naive	160
Join	Filter + Smart 3x3	108
Join	Filter + Smart 5x5	66
Join	No Filter + Simple	1055
Join	No Filter + Naive	211
Join	No Filter + Smart 5x5	43
Order By	Compare	61
Order By	Rate	11
Total (unoptimized) Total (optimized)		1055 + 61 = 1116 66 + 11 = 77

Table 5: Number of HITs for each operator optimization.

Deco
(“Deco: Declarative Crowdsourcing”)

- Motivation:
  - Handle worker disagreement
  - What is the right data model and query language (= how to extend SQL)
  - How to handle crowdsourced data in a database:
    - Do you store all answers, or just cleaned answers?
      (And if all answers, how is it stored?)
    - How does it get updated with new answers?
    - When does data go stale?
    - How do queries get executed?

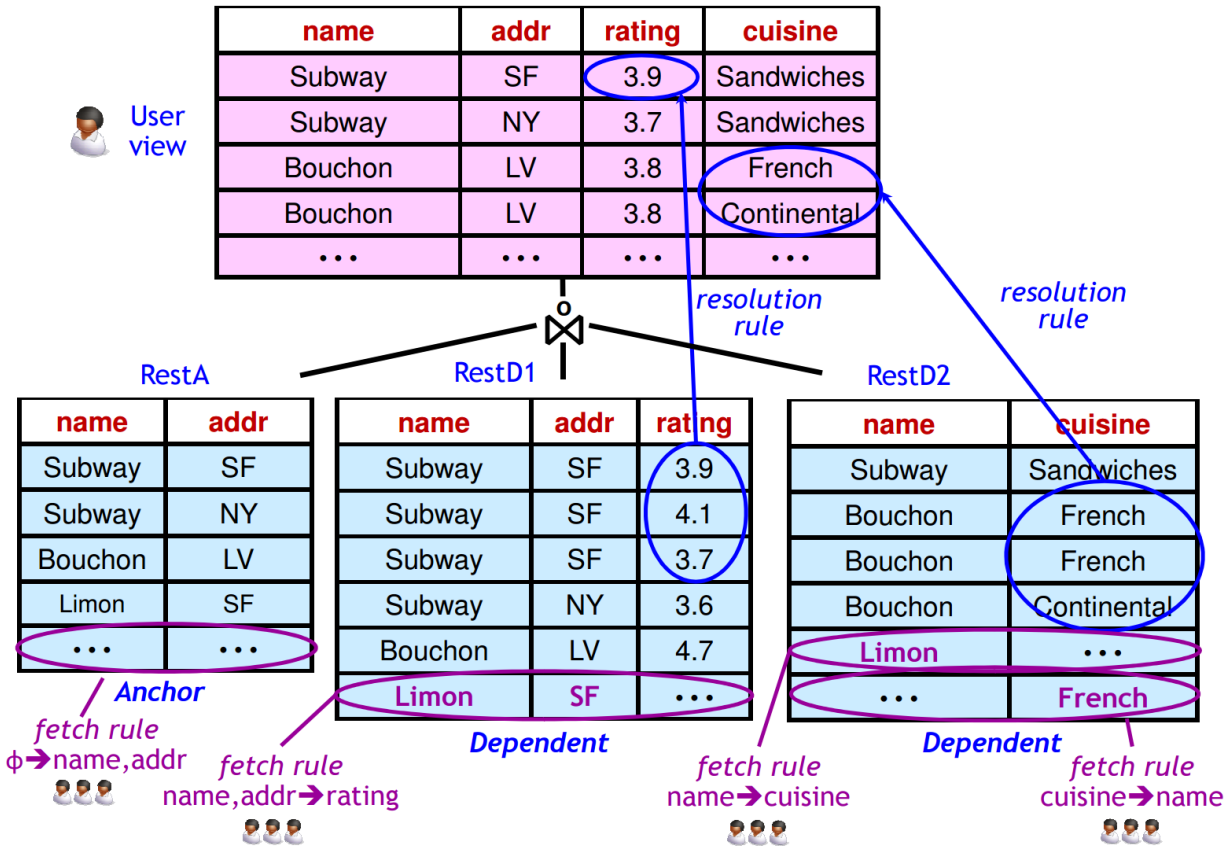
Deco
(“Deco: Declarative Crowdsourcing”)

- Provides a relational data model with well-defined semantics

  SELECT name,address,rating,cuisine
  FROM Restaurant WHERE rating > 4 ATLEAST 5

- Provides a query language that stays close to SQL
- Describes push-pull execution model
  - Ask for one or more restaurant name-address pairs
  - Ask for a rating given a restaurant name and an address
    Ask for a cuisine given a restaurant name
  - Ask for a restaurant name given a cuisine

Deco
(“Deco: Declarative Crowdsourcing”)



Deco
(“Deco: Declarative Crowdsourcing”)

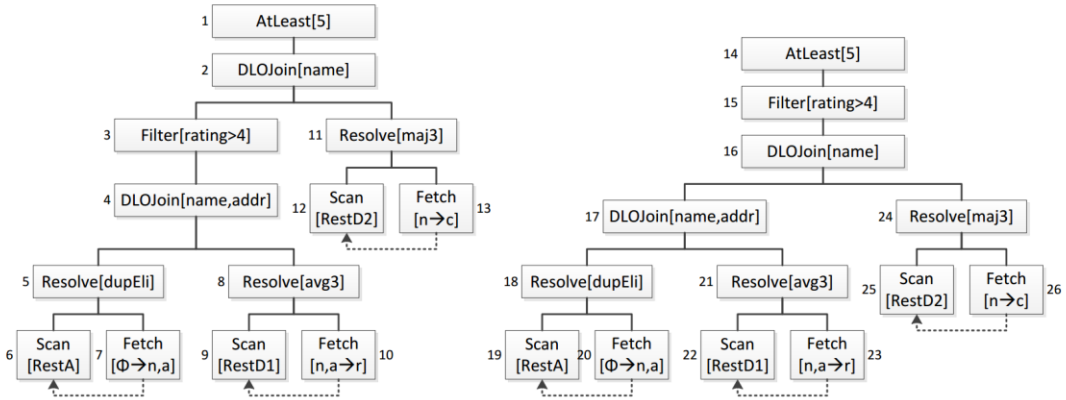


Figure 2: (a) Basic Plan (b) Filter Later

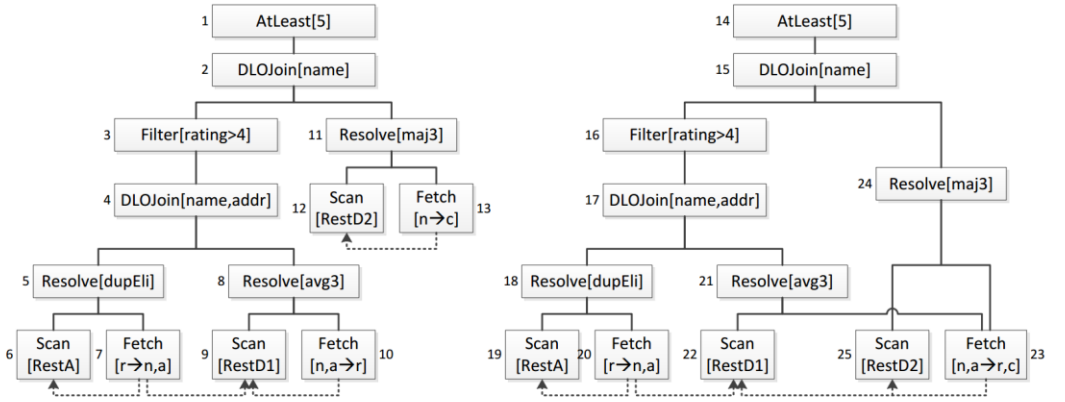
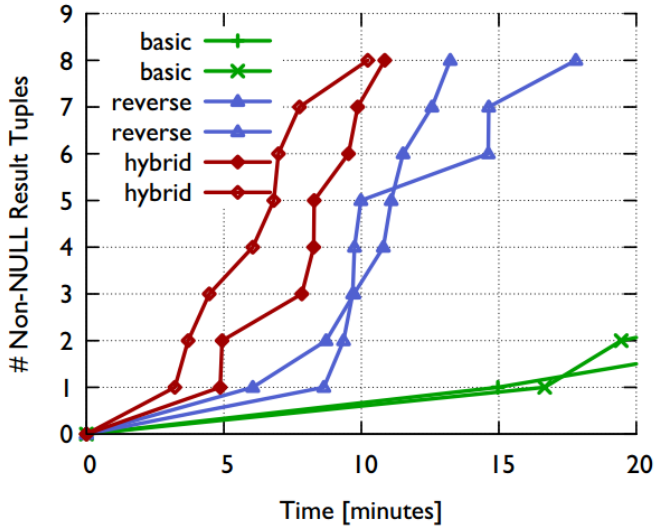
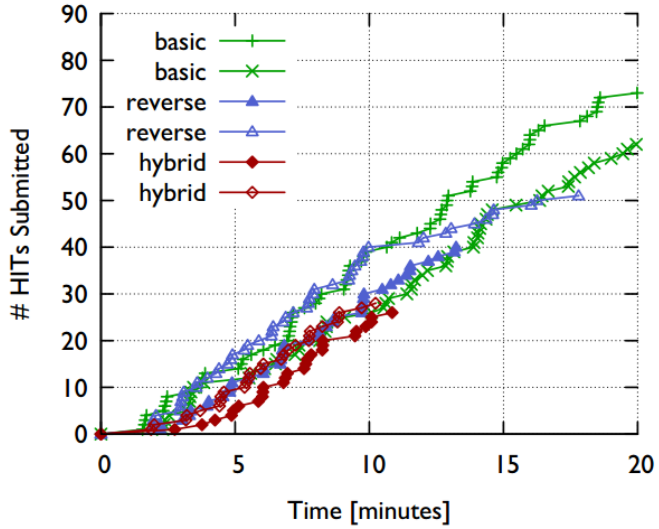


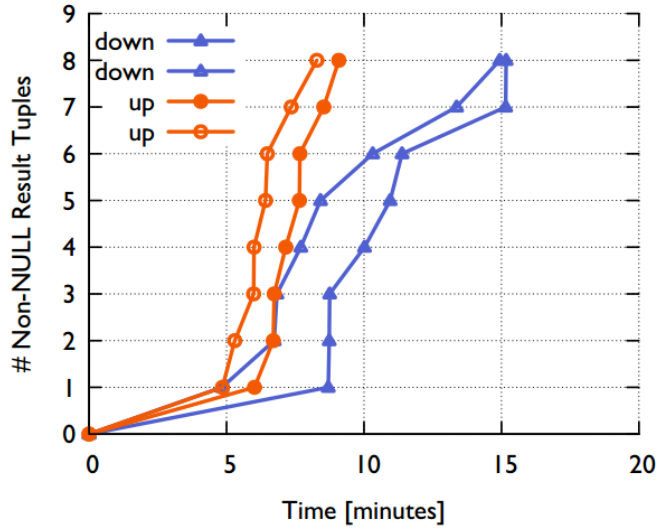
Figure 3: (a) Reverse Fetch (b) Combined Fetch

Deco
(“Deco: Declarative Crowdsourcing”)

- Assessment
  - Experiments







  - Compared expressiveness to CrowdDB

CrowdER

- Entity resolution: I.B.M. vs IBM vs International Business Machines

  SELECT p.id, q.id FROM product p, product q
  WHERE p.product_name ~= q.product_name;

- Pure crowdsourcing infeasible given number of possible matches

ID	Product Name	Price
r_1	iPad Two 16GB WiFi White	$490
r_2	iPad 2nd generation 16GB WiFi White	$469
r_3	iPhone 4th generation White 16GB	$545
r_4	Apple iPhone 4 16GB White	$520
r_5	Apple iPhone 3rd generation Black 16GB	$375
r_6	iPhone 4 32GB White	$599
r_7	Apple iPad2 16GB WiFi White	$499
r_8	Apple iPod shuffle 2GB Blue	$49
r_9	Apple iPod shuffle USB Cable	$19

- Approach:
  - Machine does initial crude pass
  - People verify most likely matches

CrowdER

DEFINITION 1 (CLUSTER-BASED HIT GENERATION). *Given a set of pairs of records, $\mathcal{P}$, and a cluster-size threshold, k, the cluster-based HIT generation problem is to generate the minimum number of cluster-based HITs, $H_1, H_2, \cdots, H_h$, that satisfy two requirements: (1) $|H_\ell| \leq k$ for any $\ell \in [1, h]$, where $|H_\ell|$ denotes the number of records in H_ℓ; (2) for any $(r_i, r_j) \in \mathcal{P}$, there exists H_ℓ ($\ell \in [1, h]$) s.t. $r_i \in H_\ell$ and $r_j \in H_\ell$.*

THEOREM 1. *The cluster-based HIT generation problem is NP-Hard.*

CrowdER

Algorithm 1: TWO-TIERED($\mathcal{P}$, k)

Input: $\mathcal{P}$: a set of pairs of records
k : a cluster-size threshold
Output: $H_1, H_2, \cdots, H_h$: cluster-based HITs

```
1 begin
2     Let CC denote the connected components of the graph
      that is built based on P;
3     SCC = {cc ∈ CC | |cc| ≤ k};  //Small Connected Components
4     LCC = {cc ∈ CC | |cc| > k};  //Large Connected Components
5     SCC ∪ = PARTITIONING(LCC, k);  //Top Tier
6     H_1, H_2, ··· , H_h = PACKING(SCC, k);  //Bottom Tier
7 end
```

Figure 6: An overview of two-tiered approach.

CrowdER

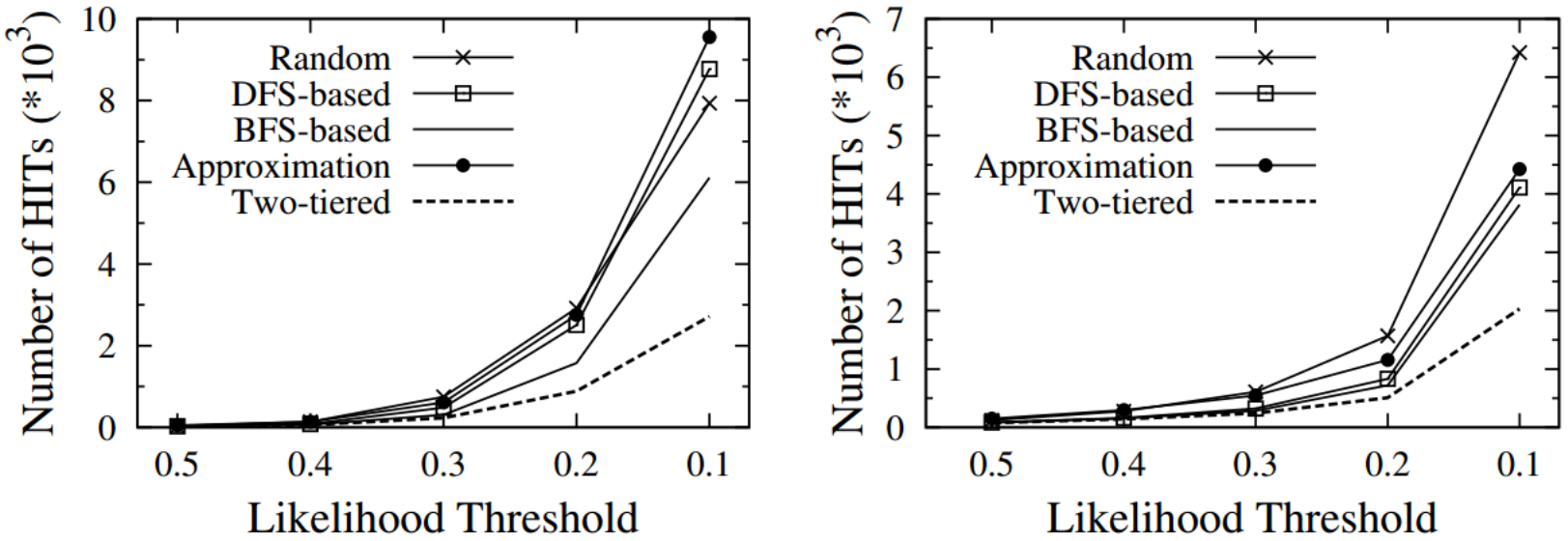


Figure 10: Comparison of the number of cluster-based HITs for various likelihood thresholds (cluster-size=10).

CrowdER

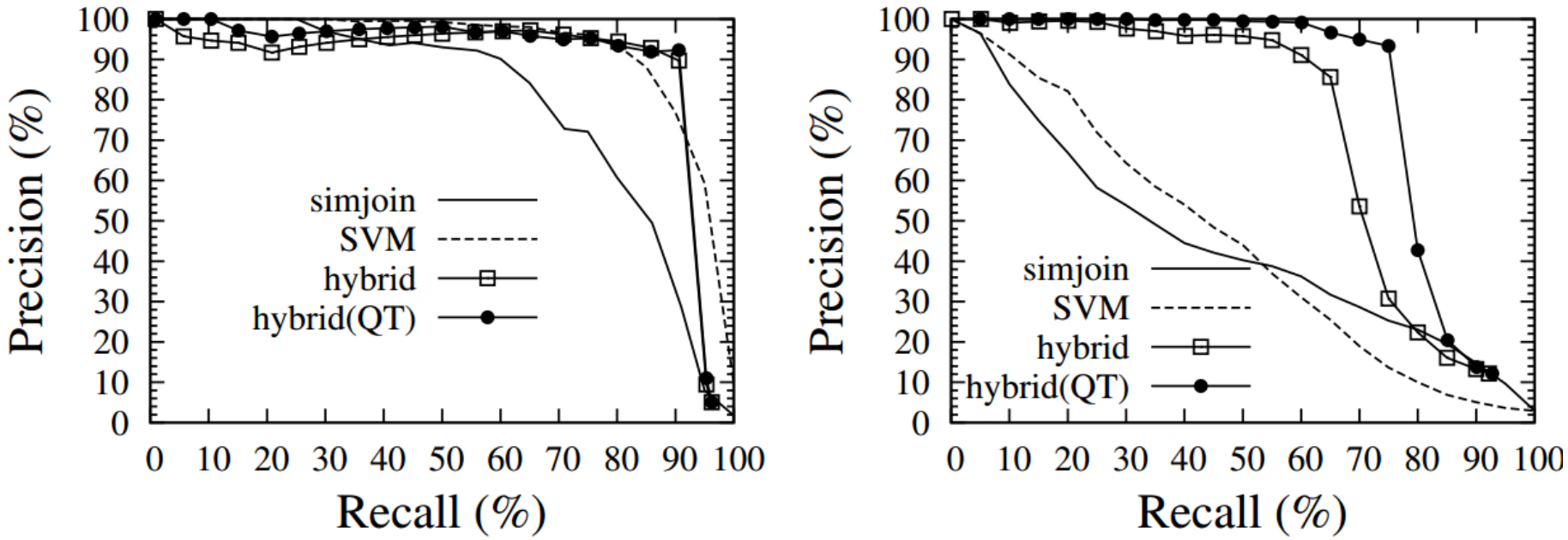


Figure 12: Comparing hybrid human-machine workflow with existing machine-based techniques.

CrowdER

- Theoretical analysis
  - Hardness
  - “Back of the envelope” algorithmic analysis
  - Experiments with AMT
  - Compared human-powered algorithm with no-human algorithm

Using the Crowd for Top-k and Group-by Queries

- Motivating example

```
SELECT most-recent(photo)
FROM photoDB
WHERE singlePerson(photo)
GROUP BY Person(photo)
```

Using the Crowd for Top-k and Group-by Queries

- Motivating example

```
SELECT most-recent(photo)
FROM photoDB
WHERE singlePerson(photo)
GROUP BY Person(photo)
```

Using the Crowd for Top-k and Group-by Queries

- Motivating example

```
SELECT most-recent(photo)
FROM photoDB
WHERE singlePerson(photo)
GROUP BY Person(photo)
```

- Goal: Use the “crowd” to answer *type* and *value* questions
  - Both take two items as input
  - type: are they of the same type
  - value: which comes first

Using the Crowd for Top-k and Group-by Queries

- Algorithms for and mathematical analysis of:
  - Max and top-k
  - Cluster on type
  - Cluster on type and values
- Error models:
  - Questions answered correctly with probability $\geq ½ + \varepsilon$ for constant ε $0 < \varepsilon \leq ½$
  - If $x_i > x_j$ Pr[x_j is returned as the larger element] $\leq \frac{1}{f(j-i)} - \varepsilon$ where
    - f is monotone
    - $f(1) \geq 2$
    - $\varepsilon > 0$ is a constant

Using the Crowd for Top-k and Group-by Queries

Algorithm 1 Algorithm for finding the maximum element.

1: – Choose a random permutation Π of the elements $x_1, \cdots, x_n$.
2: **for** levels $L = 1$ to $\log n$ in the comparison tree **do** { *leaves are in level 0, the root is in level* $\log n$* }
3: – If $L \leq \log X$ (lower $\log X$ levels), do one comparison at each internal node. Propagate the winners to the level above.
4: – If $L > \log X$ (upper $\log \frac{n}{X}$ levels), do N_L comparison at each internal node. Take majority vote and propagate the winners to the level above.
5: **end for**
6: **return** The element at the root node of the comparison tree.

Using the Crowd for Top-k and Group-by Queries

THEOREM 2. *For all strictly growing functions f and constant $\epsilon, \delta > 0$, $n + o(\frac{n}{\delta} \log \frac{1}{\delta})$ value questions are sufficient to output the maximum element x_1 with probability $\geq 1 - \delta$ in the variable error model.*

Further, if $f(\Delta) = \Omega(\Delta)$, then $n + O(\frac{\log \log n}{\delta^2} \log \frac{1}{\delta})$ questions are sufficient. If $f(\Delta) = 2^{\Delta}$, then $n + O(\log^2 \frac{1}{\delta})$ questions are sufficient.

Using the Crowd for Top-k and Group-by Queries

Algorithm 2 Algorithm for clustering with only type questions (given n elements, and the values of $\epsilon, \delta > 0$))

1: – List the elements in arbitrary order L.
2: – Initialize a set for clusters $P = \emptyset$.
3: **while** L is not empty **do**
4: Let y be the first element in L.
5: Find elements with the same type as y among the remaining elements in L as follows: For each remaining element x in L, ask the type question $\texttt{type}(x) = \texttt{type}(y)$ $O(\frac{1}{\epsilon^2}(\log \frac{n}{\delta}))$ times. If the majority of the answers are "yes", x, y are decided to have the same type; otherwise they are decided to have different types.
6: Collect all elements of the same type, make a cluster C, add to P, and delete these elements from L.
7: **end while**
8: **return** the clusters in P.

Using the Crowd for Top-k and Group-by Queries

THEOREM 3. *For all $\delta > 0$, to group n elements into J clusters with probability $\geq 1 - \delta$, $O(nJ \log \frac{n}{\delta})$ type questions in expectation are sufficient in the constant error model.*

On the other hand, $\Omega(nJ)$ type questions are necessary (i) even if the algorithm is randomized, (ii) even when answers to all type questions are exact, and (iii) even when the value of J is known.

Using the Crowd for Top-k and Group-by Queries

Algorithm 3 Algorithm for clustering in the full correlation case (given $\epsilon, \delta > 0$)

– List all elements in L in an arbitrary order.
– Initialize $\texttt{link}(y) = null$ for each element y.
– Set `repeat_loop` = true.
while `repeat_loop` is true **do**
  – Let $s = |L|$.
  – Initially, the entire L forms a single interval.
  while $|L| > s/2$ **do** {*/*The total number of elements in L is not halved*/*}
    if each interval has exactly one element **then**
      – `repeat_loop` = false
    else
      /* Divide each interval in half to form two smaller intervals*/
      for each interval B with two or more elements **do**
        – Find the median of the elements in B.
        – Partition the elements in B in two halves comparing with the median by value questions.
        – Each of these two halves forms a new interval, say B_1 and B_2.
        for both $B_i, i \in \{1, 2\}$ **do**
          – Check if B_i has at least two types: The first element y in B_i is compared with each of the other elements z in B_i to check if there is a z such that $\texttt{type}(y) \neq \texttt{type}(z)$.
          – If B_i has at least two types, B_i is called an *active interval*. Do nothing.
          – If B_i is not active (all elements have the same type), choose an arbitrary element y from B_i. For the other elements z in the interval, set $\texttt{link}(z) = y$. Delete all elements in B_i from L except y.
        end for
      end for
    end if
  end while
end while
return all elements y with their link $\texttt{link}(y)$.

Using the Crowd for Top-k and Group-by Queries

THEOREM 4. *Given any $\delta > 0$, it is sufficient to ask $O\left((n \log(\alpha J) + \alpha J) \log \frac{n}{\delta}\right)$ type and value questions in expectation to cluster n elements into J clusters with probability $\geq 1 - \delta$.*

Using the Crowd for Top-k and Group-by Queries

- Assessment:
  - Theorems

So Who Won? Dynamic Max Discovery with the Crowd

- Algorithms for and mathematical analysis of computing Max without pre-set (“structured”) algorithm:
  - Judgment Problem: “What’s the best estimate so far?”
  - Next Votes Problem: “If I spend a little more money, what do I spend it on?”

So Who Won? Dynamic Max Discovery with the Crowd

ML FORMULATION 1 (JUDGMENT). *Given* W *and* p, *determine:* $\arg\max_j P(\boldsymbol{\pi}^{-1}(1) = j|W)$.

(where *W* is the matrix of votes, π^{-1} is a permutation over the items)

THEOREM 2. *(Hardness of the Judgment Problem) Finding the maximum object given evidence is* NP*-Hard.*

THEOREM 3. *(*#P*-Hardness of Probability Computation) Computing* $P(\boldsymbol{\pi}^{-1}(1) = j, W)$ *is* #P*-Hard.*

So Who Won? Dynamic Max Discovery with the Crowd

Strategy 5 Iterative

Require: n objects, vote matrix W
Ensure: *ans* = predicted maximum object
  $dif[\cdot] \leftarrow 0$ {$dif[\cdot]$ is the scoring metric}
  for $i : 1 \ldots n$ **do**
    for $j : 1 \ldots n, j \neq i$ **do**
      $dif[j] \leftarrow dif[j] + w_{ij}; dif[i] \leftarrow dif[i] - w_{ij}$
    end for
  end for
  initialize set Q {which stores objects}
  for $i : 1 \ldots n$ **do**
    $Q \leftarrow Q \cup i$
  end for
  while $|Q| > 1$ **do**
    sort objects in Q by $dif[\cdot]$
    for $r : (\frac{|Q|}{2} + 1) \ldots |Q|$ **do**
      remove object i (with rank r) from Q
      for $j : j \in Q$ **do**
        if $w_{ij} > 0$ **then**
          $dif[j] \leftarrow dif[j] - w_{ij}; dif[i] \leftarrow dif[i] + w_{ij}$
        end if
        if $w_{ji} > 0$ **then**
          $dif[i] \leftarrow dif[i] - w_{ji}; dif[j] \leftarrow dif[j] + w_{ji}$
        end if
      end for
    end for
  end while
  $ans \leftarrow S[1]$ {$S[1]$ is the final object in S}

So Who Won? Dynamic Max Discovery with the Crowd

- Iterative is the best when the number of votes sampled is n(n-1)/2
- PageRank is the best when there are few votes and worker accuracy is high
- PageRank is poor when worker accuracy is low

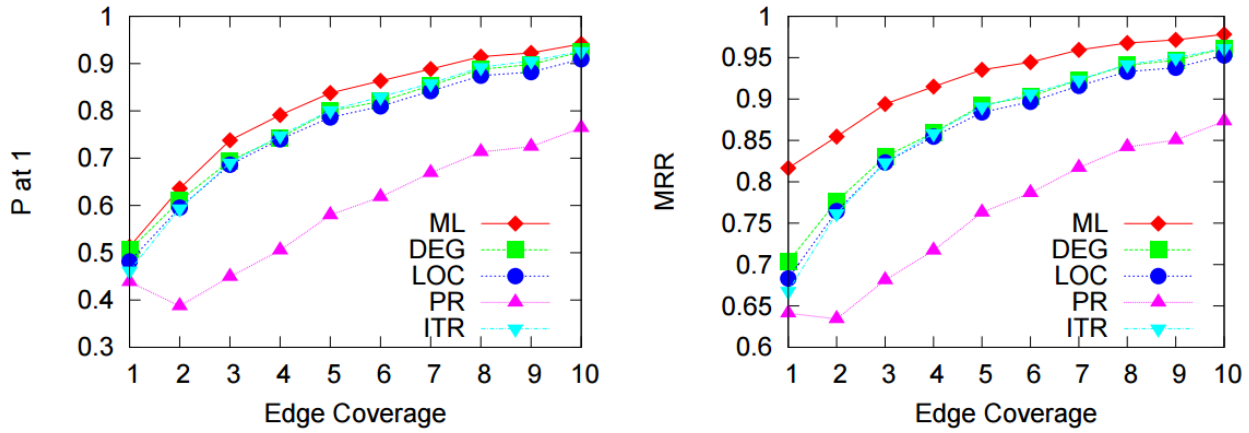


Figure 2: Comparison of ML and heuristics. Prediction performance versus Edge Coverage. 5 objects, p=0.75. P@1 (left), MRR (right).

So Who Won? Dynamic Max Discovery with the Crowd

- Assessment:
  - Theorems about hardness of exact solution
  - Experiments about approximation methods

So Who Won? Dynamic Max Discovery with the Crowd

- Assessment:
  - Theorems about hardness of exact solution
  - Experiments about approximation methods

  - Nothing with humans

Week of Sept 20

- Required readings:
  - Martin, D., Hanrahan, B.V., O'Neill, J. and Gupta, N., 2014. "Being a turker." In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*(pp. 224-235). ACM.
  - Irani, L.C. and Silberman, M., 2013. "Turkopticon: interrupting worker invisibility in Amazon Mechanical Turk." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 611-620). ACM.
- Additional readings:
  - Brawley, A.M. and Pury, C.L., 2016. "Work experiences on MTurk: Job satisfaction, turnover, and information sharing." *Computers in Human Behavior*, *54*, pp.531-546.
  - Gray, M.L., Suri, S., Ali, S.S. and Kulkarni, D., 2016. "The crowd is a collaborative network." In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (pp. 134-147). ACM.
  - Gupta, N., Martin, D., Hanrahan, B.V. and O'Neill, J., 2014. "Turk-life in India." In *Proceedings of the 18th International Conference on Supporting Group Work* (pp. 1-11). ACM.
  - Lee, M.K., Kusbit, D., Metsky, E. and Dabbish, L., 2015. "Working with machines: The impact of algorithmic and data-driven management on human workers." In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 1603-1612). ACM.
  - McInnis, B., Cosley, D., Nam, C. and Leshed, G., 2016. "Taking a HIT: Designing around Rejection, Mistrust, Risk, and Workers' Experiences in Amazon Mechanical Turk." In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (pp. 2271-2282). ACM.
  - Salehi, N., Irani, L.C., Bernstein, M.S., Alkhatib, A., Ogbe, E. and Milland, K., 2015. "We are dynamo: Overcoming stalling and friction in collective action for crowd workers." In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 1621-1630). ACM.