CrowdDB: Answering Queries using Crowdsourcing

Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, Reynold Xin

Slides by Anil Shanbhag

Outline

- Crowdsourcing
- Motivation
- Design Considerations
- Overview of CrowdDB
- CrowdSQL
- User Interface Generation
- Query Processing
- Experimental Results

Introduction to Crowdsourcing

Crowdsourcing

Let us consider a few simple examples

Example 1 : Given an image containing text, get the text from it (aka CAPTCHA)

Example 2 : Given a university and department name, find the link to department webpage

Example 3 : Given a set of photographs of people from a disaster, and pictures submitted by family members of lost individuals, perform a fuzzy join across both sets, using humans to determine equality

All these examples are trivial for a human to do but expensive to do as a computer

Crowdsourcing

- Platform

- A crowdsourcing platform creates a marketplace on which requesters offer tasks and workers accept and work on the tasks
- The largest among these is Amazon Mechanical Turk

- Platform had 500,000 workers in 2011 and continuously growing.

Crowdsourcing

- Platform

- Definitions

- *HIT* : A Human Intelligent Task, or HIT, is the smallest entity of work a worker can accept to do. HITs contain one or more jobs.

  Eg: Given a university and department name, find the url of department webpage

- *Assignment* : Every HIT can be replicated into multiple assignments.

- *HIT Group* : Group of similar HITs. Grouped for convenience of 'turkers'

  Eg: We can post 1000 HITS to get the URL of 1000 different departments spread across universities.

Crowdsourcing

- Platform

- Definitions

- Workflow

- Package the jobs comprising of information into HITs, determines the number of assignments required for each HIT and posts the HITs
- Optionally specify requirements that workers must meet in order to be able to accept the HIT
- AMT Groups compatible HITs into HIT Groups and posts them so that they are searchable by workers
- A worker accepts and processes assignments
- Requesters then collect all the completed assignments for their HITs and apply whatever quality control methods they deem necessary. More on this later.

Motivation

Hybrid Human-Machine DBMS

Hard Database Problems

- Missing Data

  A key limitation of relational technology stems from the Closed World Assumption. People, aided by tools such as search engines and reference sources, are quite capable of finding information that they do not have readily at hand.

- Fuzzy Comparisons

  People are skilled at making comparisons that are difficult or impossible to encode in a computer algorithm

Harness Human Computation for solving problems that are impossible or too expensive to answer correctly using computers.

Is it possible leverage such human resources to extend the capabilities of database systems ?

Design Considerations

Design Considerations

- **Performance and Variability**

  Humans and machines differ greatly in type, speed and cost of work done
  People show tremendous variability from person to person and over time
  Need appropriate query planning, fault tolerance and answer quality

- **Task Design and Ambiguity**

  Carefully designed user interface with human readable instructions are needed.

Design Considerations

- **Affinity and Learning**

  Workers develop relationship with requesters and skills for certain types of HITs. Not uncommon to find workers doing only image classification. Hesitant to do tasks from requesters who don't provide well-defined tasks / pay appropriately. CrowdDB design to take longer-term view on task and worker community development.

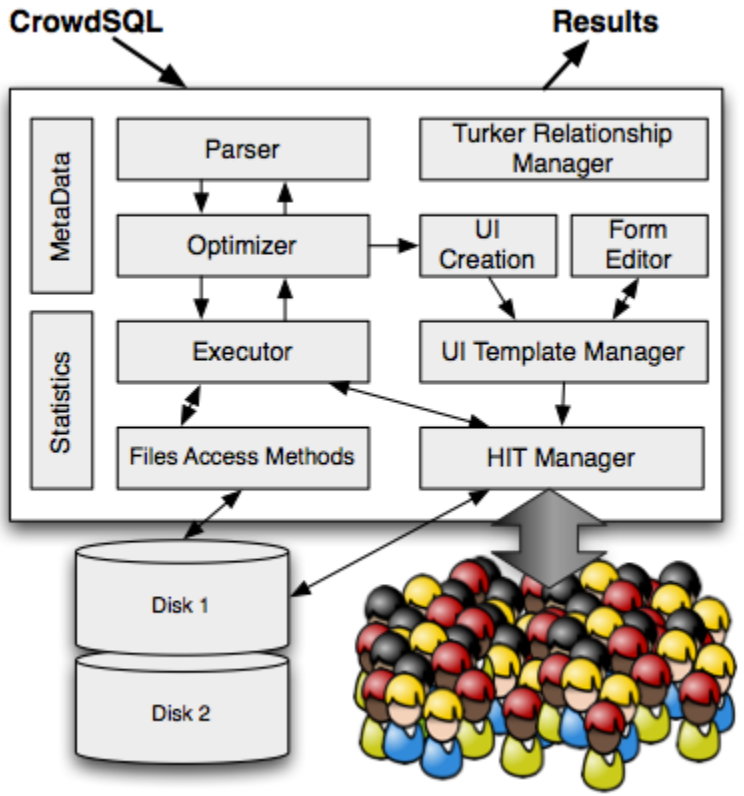
- **Relatively Small Worker Pool**

- **Open vs Closed World**

  In CrowdDB closed world assumption doesn't hold. Any one query operator could conceivably return an unlimited number of answers. Implications on query planning, cost and answer quality.

Overview of CrowdDB

Overview of CrowdDB



Left side of the figure are traditional QO parts: parsing, optimization and execution. Right side contain components used to extend the traditional DB system to get human generated input.

CrowdSQL

CrowdSQL

CrowdSQL is a SQL extension that supports crowdsourcing.

Supports two main use cases:

- allow crowdsourcing missing data
- support subjective comparisons

CrowdSQL

Incomplete Data

SQL DDL Extensions

Two scenarios:

- Specific attributes could be crowdsourced
- Entire tuples could be crowdsourced

Introduce a CROWD keyword to solve both.

Let us revisit the initial example of finding the department webpage url. In CrowdSQL it translates into

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

The crowd keyword indicates that the url attribute will be got using crowdsourcing.

CrowdSQL

Incomplete Data

Let's go one step further. We want to get the details of all the professors in a department. This translates into:

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

Notice the CROWD keyword on table. It indicates that all tuples in this relation will be crowdsourced.

There are no constraints placed and tables both crowd / non-crowd treated same way w.r.t referential integrity constraints.

However there is one exception: CROWD tables must have a primary key so that CROWDDB can infer if two workers input same tuple.

CrowdSQL

Incomplete Data

SQL DML Extensions

A special CNULL value indicates data in CROWD columns that should be crowd-sourced when needed as part of processing a query.

During INSERT/UPDATE, crowd columns can also be initialised. All non-initialised crowd columns are set to CNULL.

```
INSERT INTO Department(university, name)
VALUES ("UC Berkeley", "EECS");
```

Consider example above, it sets `url` to CNULL

Query Semantics

CrowdDB supports any kind of SQL query on CROWD tables and columns

CrowdSQL specifies that tables are updated as a side-effect of crowdsourcing.

Let us take two examples based on the tables created previously:

```
SELECT url FROM Department
WHERE name = "Math";
```

In this, the url column would be implicitly updated with the crowdsourced URL.

```
SELECT * FROM Professor
WHERE email LIKE "%berkeley%" AND dept = "Math";
```

In this query missing values in the email column would be implicitly populated and new professors of math department would be implicitly inserted as a side-effect of processing.

CrowdSQL

Incomplete Data

Subjective Comparisons

Beyond finding missing data, subjective comparisons important use of CrowdDB

Two new operators: CROWDEQUAL and CROWDORDER

CROWDEQUAL(~=) takes two parameters (an lvalue and an rvalue) and asks the crowd to decide whether the two values are equal

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

Here the query writer asks the crowd to do entity resolution with the possibly different names given for Computer Science in the database like 'Computer Science', 'CSE', etc.

CrowdSQL

Incomplete Data

Subjective Comparisons

CROWDORDER is used whenever the help of the crowd is needed to rank or order results

The CrowdSQL query below asks for a ranking of pictures with regard to how well these pictures depict the Golden Gate Bridge

```
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");
```

As with missing data, CrowdDB stores the results of CROWDEQUAL and CROWDORDER calls so that the crowd is only asked once for each comparison. This caching is equivalent to the caching of expensive functions in traditional SQL databases

User Interface Generation

User Interface Generation

- Overview

UI is important!

A clear, unambiguous user interface helps greatly in improving accuracy.

Two step process:

- **Compile-time**

  CrowdDB creates templates to crowd-source missing information from all CROWD tables and all regular tables which have CROWD columns. JS is generated in additon to HTML to do type checking.

- **Runtime**

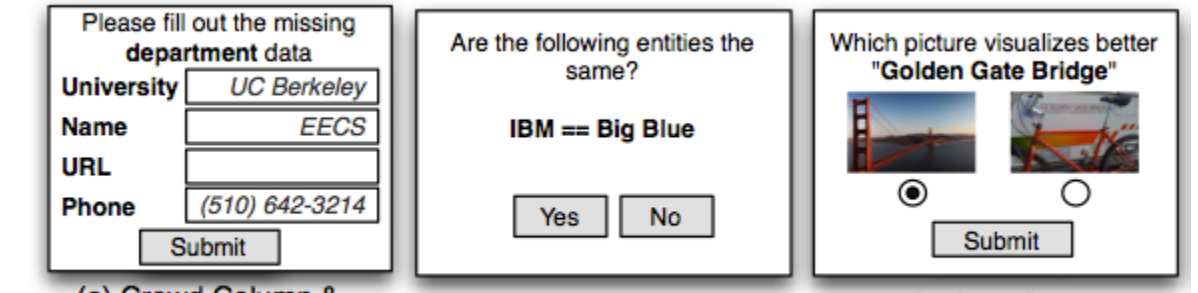
  These templates are instantiated at runtime in order to provide a user interface for a concrete tuple or a set of tuples.

User Interface Generation

- Overview

- Basic Interfaces

Basic UI for crowd tasks



(a) Crowd Column & Crowd Tables w/o Foreign Keys

(b) CROWDEQUAL

(c) CROWDORDER

- (a) is our earlier example where we want to crowdsource url
- (b) does entity resolution using CROWDEQUAL
- (c) is our earlier example to rank a set of images based on how well they visualize subject (here Golden Gate Bridge)

User Interface Generation

- Overview

- Basic Interfaces

The following optimizations are used:

- Batching: Get information of several tuples at once (Eg: URL of Elec, CS, EP of UC-Berkeley). Assumption: cheaper to input two pieces of information of the same kind in a single form rather than separate forms
- Prefetching: Consider, say both the department and email of a professor are unknown, but only the email of that professor is required to process a query, it might make sense to get the department too.

Interfaces for CROWDEQUAL(Fig (b)) and CROWDORDER(Fig (c)) can also be batched.

User Interface Generation

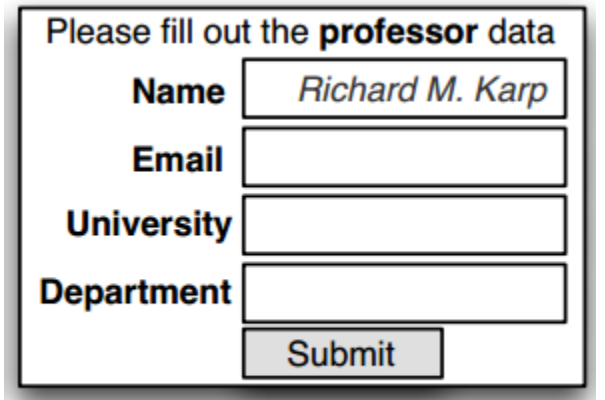
- Overview

- Basic Interfaces

- Multi-Relation Interfaces

Crowdsourcing relations with foreign keys require special considerations

- If foreign key references non-CROWD table, the generated user interface shows a select box and for larger lists a ajax based suggest method
- If foreign key references CROWD table, there are two types of interfaces which are used:
- **Normalised Interface** : The worker inputs the value of foreign key but no other attributes of referenced tuple

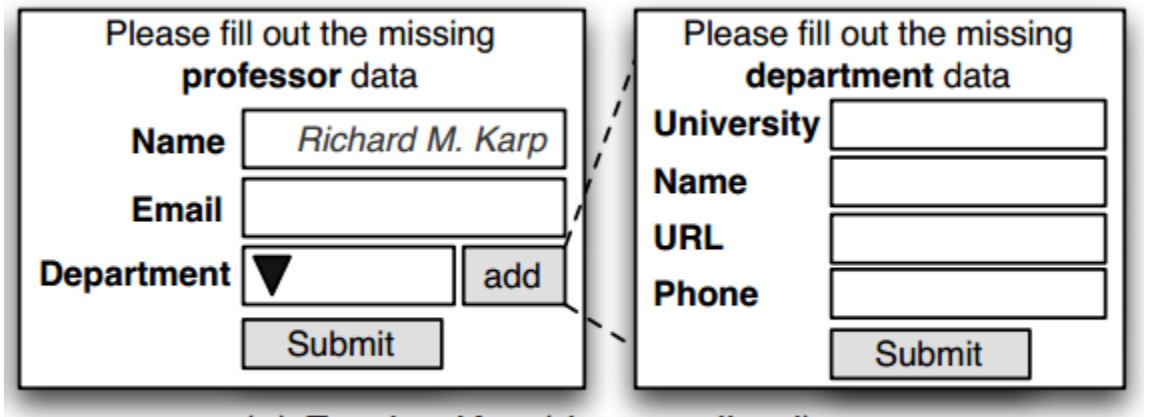


(d) Foreign Key(normalized)

User Interface Generation

- Overview
- Basic Interfaces
- Multi-Relation Interfaces

- **Denormalised Interface** : There is a select box and an add button which allows the worker to input a new department



(e) Foreign Key (denormalized)

- To source entirely new tuples, the non-key attributes can be preset via WHERE clause, autosuggest while typing and an option to say no new professor entry present. If many workers say no new professor entry present, we can stop.

Query Processing

Query Processing

The traditional database model extended:

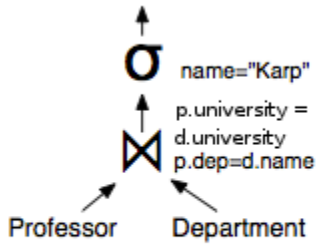
- SQL extended to CrowdSQL
- Crowd Operators for crowdsourcing
- Optimizer that handles crowd operators.

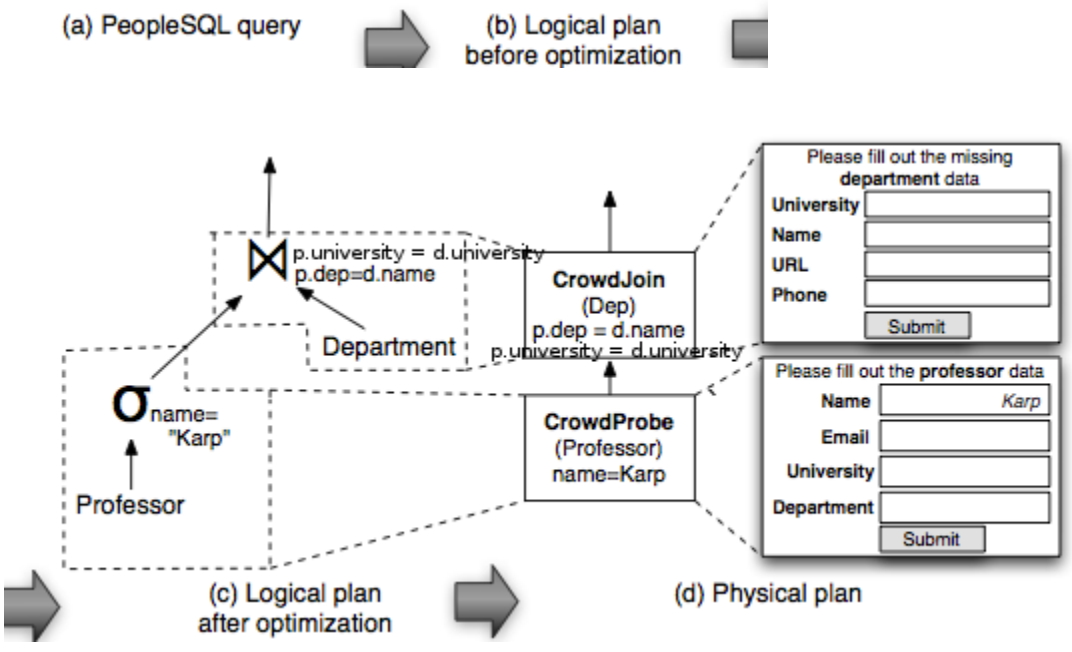
  CPU time taken << time taken by crowd to answer
  => goal of optimizer is to find plan which results in least number of queries to Crowd.

Query Processing

Example

```
SELECT *
FROM professor p,
  department d
WHERE p.department = d.name
  AND p.university = d.university
  AND p.name = "Karp"
```





CROWDPROBE

- Crowdsources missing information of CROWD columns (i.e., CNULL values) and new tuples
- Quality control carried by majority vote. If not majority achieved at max hits, choose randomly from most frequent ones.
- In the case of new tuples, finding majority impossible. The DB reposts the tasks with only primary key filled in.

CROWDJOIN

- Implements a nested-loop join where atleast one of the tables is a CROWD table
- For every tuple of outer relation, creates HIT's to find the inner tuples

CROWDCOMPARE

- Implements the CROWDEQUAL and CROWDCOMPARE functions
- Typically used inside a traditional operator like sorting

The basic functionality of all Crowd operators is the same.

- Initialized with a user interface template and the standard HIT parameters
- At runtime, they consume a set of tuples
- Depending on the Crowd operator, crowdsourcing can be used to source missing values of a tuple or to source new tuples.
- Batch HITs. Create HIT Groups.
- Consume tuples from crowd and do quality control

Quality control is currently carried out by a majority vote. The number of workers assigned to each HIT is controlled by an Assignments parameter.

Experiments and Results

As the HIT group size increases, the time to get first x responses decreases. Larger HIT groups mean more tasks to attempt. HITs are repetitive tasks and there is an initial overhead of learning how to do the task. Hence larger HIT groups give higher payoffs and attract more turkers.

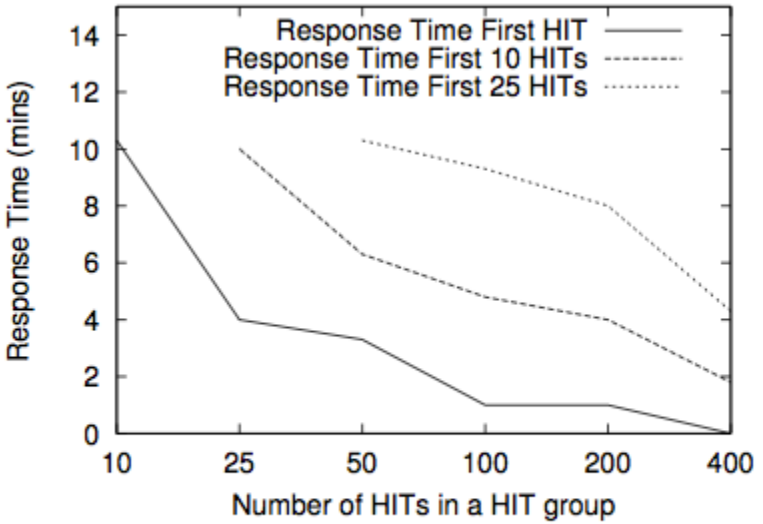


Figure 4: Response Time (min): Vary Hit Group *(1 Asgn/HIT, 1 cent Reward)*

Experiments and Results

However the percentage of HIT's completed in the 30 minutes increases and then decreases. Exhibits tradeoff between throughput and completion %.

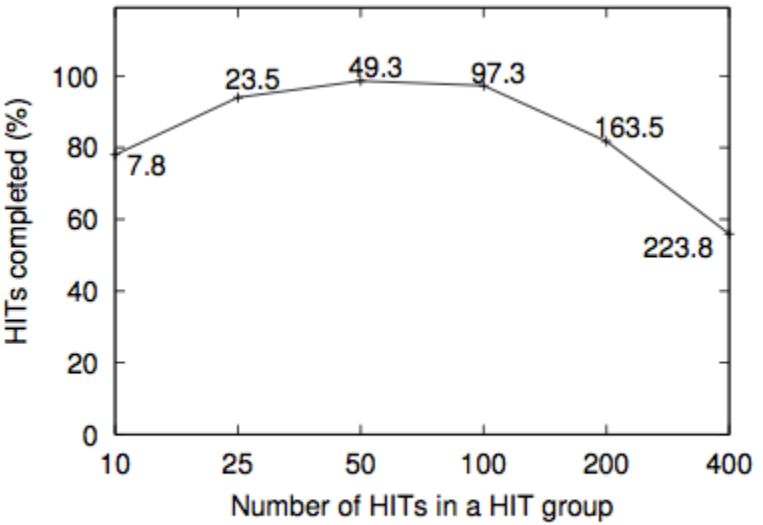


Figure 5: Completion (%): Vary Hit Group *(1 Asgn/HIT, 1 cent Reward)*

Experiments and Results

Paying more than 1 cent per task attracts more workers. However beyond 2 cents, there is barely any difference.

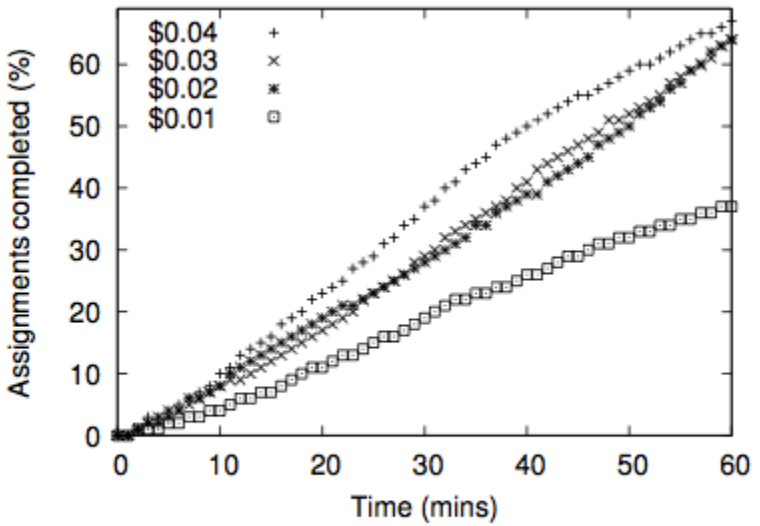


Figure 6: Completion (%): Vary Reward *(100 HITs/Group, 5 Asgn/HIT)*

Experiments and Results

The graph below shows the work distribution. It is highly skewed. Total 750 workers.

- Tasks acquire a community
- The authors thought the ones doing more hits will have lesser error but this behaviour not seen in experiments.

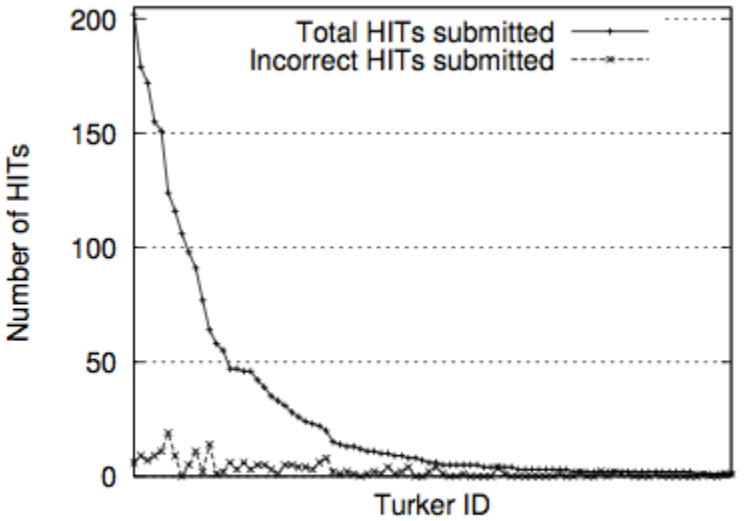


Figure 8: HITs/Quality by Worker *(Any HITs/Group, 5 Asgn/HIT, Any Reward)*

Complex Queries

Query is to sort the pictures for Golden Gate bridge. They rankings are close to ranking by experts.

(a) 15, 1, 1 (b) 15, 1, 2 (c) 14, 3, 4 (d) 13, 4, 5

(e) 10, 5, 6 (f) 9, 6, 3 (g) 4, 7, 7 (h) 4, 7, 8

Figure 10: Pictures of the Golden Gate Bridge [1] ordered by workers. The tuples in the sub-captions is in the following format: {the number of votes by the workers for this picture, rank of the picture ordered by the workers (based on votes), rank of the picture ordered by experts}.

References

CrowdDB: answering queries with crowdsourcing

Michael J. Franklin, Donald Kossmann, Tim Kraska, S. Ramesh, Reynold Xin

Suggested Reading:

CrowdScreen: algorithms for filtering data with humans

Aditya Parameswaran, et al.

Crowdsourced databases: Query processing with people

A Marcus, et al.

Deco: Declarative Crowdsourcing

Aditya PArameswaran, et al.