Loke *Hum. Cent. Comput. Inf. Sci.* (2016) 6:4
DOI 10.1186/s13673-016-0061-6

Human-centric Computing and Information Sciences

RESEARCH Open Access

Heuristics for spatial finding using iterative mobile crowdsourcing

Seng W. Loke*

*Correspondence: s.loke@latrobe.edu.au
Department of Computer Science and Information Technology, La Trobe University, Kingsbury Drive, Melbourne, Australia

Abstract
Crowdsourcing has become a popular method for involving humans in socially-aware computational processes. This paper proposes and investigates algorithms for finding regions of interest using mobile crowdsourcing. The algorithms are iterative, using cycles of crowd-querying and feedback till specified targets are found, each time adjusting the query according to the feedback using heuristics. We describe three (computationally simple) heuristics, incorporated into crowdsourcing algorithms, to reducing the costs (the number of questions required) and increasing the efficiency (or reducing the number of rounds required) in using such crowdsourcing: (i) using additional questions in each round in the expectation of failures, (ii) using neighbourhood associations in the case where regions of interest are clustered, and (iii) modelling regions of interest via spatial point processes. We demonstrate the improved performance of using these heuristics using a range of stylised scenarios. Our research suggests that finding in the city is not as difficult as it can be, especially for phenomena that exhibit some degree of clustering.

Keywords: Mobile crowdsourcing, Spatial finding, Heuristics

Background

Crowdsourcing, or crowd computing, is an important powerful approach to problem-solving and critical information gathering, harnessing the power of the crowd, and creatively combining machine and human computations [1–3]. Crowdsourcing can be used to do tasks that either no machine alone can do or where involving humans is better (e.g., CrowdDB [4]). *Mobile crowdsourcing*, i.e. crowdsourcing to mobile users, presents significant new opportunities and challenges, with enormous possibilities for human computations and tasks with spatial and temporal properties [5–12].

One important class of crowdsourcing applications is where information and tasks to be crowdsourced have spatiotemporal properties and are advantageously done by mobile device users (e.g., crowdsourcing for carpark spaces, locations of crowds, maps of areas, transport demand, emergency needs, photos/video at different locations of a parade, location of flora and fauna), different from general tasks that can be crowdsourced such as language translation or copy-editing. Such crowdsourcing may be done over extended periods of time and data centralised for analytics, or can be done in an ad-hoc real-time on-demand manner (e.g., issuing crowdsourced queries to assess a situation in the vicinity within the next few minutes). Real-time crowdsourcing of queries can be useful, even with prior information available (e.g., there is a database of historical traffic or car parking availability for a given area) obtained from some source (e.g., either from stationary sensors or previous crowdsourcing efforts or other means) in order to obtain up-to-date information that can complement or update prior information, especially in the case where information can easily deviate from history (e.g., car parking), or historical data are too coarse-grained (e.g., the database only has average car park availability for a large area over days), or without any prior information, where the queries are to obtain new information about regions in the area, in a just-in-time on-demand manner. Real-time responsiveness in crowdsourcing is a challenge but methods have been explored making it a real possibility. For example, Bernstein has done interesting work in preparing crowds via additional incentives in order to achieve real-time performance [13].

Crowdsourcing involves incentives, and hence, costs, such as monetary costs for payments for answers as well as efficiency costs, e.g., in terms of time taken to achieve an adequate response. Algorithms where humans are viewed as data processors have been explored for finding the maximum [14], filtering [15] and for finding a subset of items with a given property among a given unstructured collection [16], taking into consideration the need to optimise cost and efficiency at the same time.

In this paper, we propose and investigate iterative crowdsourcing processes, based on work from [17], to find regions of particular interest (e.g., regions satisfying particular properties) from among a collection of regions, with all such regions within a given fixed size area, as typically seen in the context of *mobile crowdsourcing*, where the human contributors (or workers) for crowdsourced tasks/queries are people within the area with mobile devices, so that queries or jobs posed to them (and their answers) have spatial properties. We also have in mind real-time crowdsourcing where results are intended for the here-and-now, rather than obtained over a long period of time, but our work does not deal specifically with strict real-time constraints.

In particular, we consider cases where association between regions can be exploited to reduce the costs and increase efficiency in crowdsourcing. Often, information about a region provides clues about information of its neighbouring regions: a region that is crowded might be adjacent to another crowded region or a polluted region might be adjacent to another polluted region, even if this is not always the case. We argue that this is the case for a number of real-world phenomena including car parking, 3G/4G bandwidth, crowded areas, and noise pollution. So, for example, if one wants to use crowdsourced queries to find regions where there are car parks available, regions where there is currently high 3G/4G bandwidth, regions which are crowded, and regions with noise pollution above some threshold, then neighbourhood or proximity associations can be exploited.

In the rest of this paper, we first outline the spatial finding problem in "The spatial finding problem" section, and discuss possible solutions in "A crowdsourcing solution" section. Then, we introduce three heuristics for spatial finding in crowdsourcing and describe experimentation to demonstrate the effectiveness of the heuristics, a heuristic using more queries than the minimal in "A heuristic that embraces failure: redundant questioning" section and a heuristic using immediate neighbourhood associations in "A heuristic for spatial finding: neighbourhood associations" section, and a heuristic using spatial point processes in "Experiments" section. We then review related work in "Related work" section and conclude in "Conclusion" section.

The spatial finding problem

The spatial finding problem is a simple variation of the problem first proposed in [17]. The basic version of the original problem is as follows: given a (large) set of items, a predicate, and a number k, use humans to find k items from the given set that satisfy a given predicate. An example of an instance of this problem is given as follows: "Consider a data set I of images, from which we want to find 10 images that satisfy a predicate or flter f, e.g., whether it is a photo of a cat. We consider each image, and ask humans the question, e.g., 'does this image show a cat?' Suppose on average that 20 % of the photos are of cats. For the purposes of this example, we assume that humans do not make mistakes while answering questions." A solution to the above problem might be sequential (to ask about one image at a time) and stop whenever 10 images are found; this algorithm is cost-optimal asking only as many questions as needed, but could take a long time (requiring many rounds of questioning)—the cost and latency depends on which images are picked. Another solution is to consider asking about all images in parallel; this is fast (requiring only one round of questioning), but is costly since one needs to pay for all the questions asked. A third possible solution which is between the first and second solution in terms of the cost-latency tradeoff is to ask $(10 - x)$ questions at the current round of questioning, if we already found x cat images so far.

This original version does not deal with spatial properties of items as we do in this paper. We define below a spatial version of the above problem and while we explore the above solution ideas, we consider spatial heuristics for picking items to ask about. The general notion of cost-latency tradeoffs, however, also applies here.

Our Problem Assume a a large area R partitioned into n regions $\{r_1, \ldots r_n\}$. The problem is to find a set $S \subseteq R$ of at least $k \leq n$ regions, each of which evaluates to true for a given predicate F representing some criteria, i.e. $F(r) = TRUE$, for each $r \in S$. We also want to solve this problem with the lowest cost (assuming we need to pay to get a question about a region answered) and in a most efficient way (the number of rounds of questions required).

For example, we want to find at least k regions with available car parking spaces, and can divide a large area into a set of regions, about which we can then ask the crowd about, but each time we ask the crowd about a region, we assume that we incur a cost. Another example is to find a not-so-crowded cafe and can issue a query to find at least k regions with a not-so-crowded cafe, answers being given by people near or within the region. A third example is to find a high bandwidth (WiFi, 4G or otherwise) region.

There are two factors to deal with in any solution to the problem. One is the cost, where we assume that each time a query is issued to find out about a region, a cost ϕ is incurred (which includes the cost of issuing the query as well as incentives paid for an answer to the query) so that if we ask about k regions, we incur a total cost of $k. \cdot \phi$. The other is efficiency which we define to be the number of *rounds* of querying required, where in each round, a set of queries is issued in parallel to find out about a particular chosen set of regions (assuming one query per region).

While we do not prescribe the mechanism used by users to issue queries to other users, we assume that some cost is incurred per query issued (or per answer about a region obtained). We assume, for simplicity, here that each query issued is always answered, and answered accurately and truthfully. The cost could be measured in different forms, e.g., to the query issuer, the cost could be a monetary incentive to a user providing an answer about a query (about a region). This means that a general mechanism to ask everyone could incur a high cost if everyone actually answers, as we discuss below.

A crowdsourcing solution

An initial solution to the above problem is to simply adapt the solutions from [16], which was initially developed to find particular items from a database of items: assuming that each region is an item requiring a binary answer YES or NO (TRUE or FALSE), we have the algorithm below which is to find particular regions from a collection of regions. Note that YES/NO questions are very easy for users to respond to (but of course, tend to provide less information than more general responses).

Given a set of regions R and α which denotes the fraction of regions (we call *positive regions*) of R where F evaluates to *TRUE* (and the rest of the regions, F evaluates to *FALSE*, and we assume that $\alpha > 0$), and assuming that we are finding k positive regions from R, where there are at least k regions that can satisfy F, i.e., $k \leq |R| \cdot \alpha$, we have Algorithm 1.

```
ALGORITHM 1: SpatialCrowdsourcing($k, F, R$)
Input: $k$:number of regions to find, $F$:predicate on region, $R$:set of regions
Output: $D$ is the set of found regions, $O$ is the set of observed regions
$D := \emptyset$ (*set of found regions*);
$O := \emptyset$ (*set of observed regions*);
while $(|D| < k)$ and $(O \subset R)$ do
    $C :=$ chooseCandidates($k, D, R \setminus O$);
    askCrowdAbout($C$);
    $(A \cup B) :=$ response from crowd about $C$, where
        $A := \{r \in C \mid F(r) = TRUE\}$ and $B := \{r \in C \mid F(r) = FALSE\}$ and
    $(A \cup B) \subseteq C$;
    $D := D \cup A$;
    $O := O \cup C$;
end
return $D$ and $O$;
```

While we have not found enough regions and while there are still regions from R that are not yet observed, we iteratively choose a subset of regions to observe, and this subset is according to choose Candidates $(k, D, R \backslash O)$, a version of which is shown in Algorithm 2. In Algorithm 1, ask Crowd About(C) issues $|C|$ queries in parallel to ask about regions in C, and in practice, would have a maximum wait time. Algorithm 1 terminates either when k regions satisfying F are found and/or when all regions in R have been observed.

```
ALGORITHM 2: chooseCandidates($k, D, R \setminus O$)
Input: $k$:number of positive regions to find; $D$:set of found positive regions so far; $O$:set of
        observed regions, $R$:set of all regions
Output: $C$ is a set of selected unobserved regions to be queried
$c := k - |D|$;
$C$ := randomly select $c$ regions from $R \setminus O$;
$return\ C$;
```

Depending on how we choose the candidates, three solutions are possible:

1 It has been shown from [16] that a way to minimise total cost is to ask about one region at a time (say, in any order) and stopping whenever k regions are found satisfying the criteria, so that we never ask more questions than required or get more than k positive answers; only in the worst case, this scheme can lead to $|R|$ rounds.

2 A more expensive solution but very efficient (requiring one round) is to ask $|R|$ questions about all the regions in parallel. This could be done, for example, by posting a query like 'where can I find a parking spot?' on a wide-audience medium, such as a (mobile accessible) Website say, and anyone or everyone in any region can answer the question; in effect, we are asking about all regions at the same time. Since we assume that a query about any region is always answered, the cost is then $|R| \cdot \phi$. Now, suppose we want a solution that can achieve a cost less than $|R| \cdot \phi$. To find k positive regions, note that a method to do this might be to issue $|R|$ queries and then wait for a certain fixed period of time for k positive responses and paying for all the first $K \geq k$ answers obtained on a first-come first-serve basis[1]—however, this has already incurred costs in issuing the $|R|$ queries and also paying for what may be largely $K-k$ negative answers; to avoid such costs, we want to select regions to ask about (reducing the cost of issuing queries) and focus search queries to where there is a higher likelihood of getting a positive response. For *each query* issued on a region, the first answer obtained could be used and paid for, or it could be obtained via taking a majority vote of the first z answers (where z is the number of answers that can be paid for from the budget ϕ).

3 A third solution which aims to minimise cost and maximise efficiency at the same time is as follows, which will be the main focus in the rest of this paper. In each round, we ask no more questions than that required if all the answers were positive. More precisely, in round i, if $k_i < k$ regions have already been found where F evaluates to *TRUE*, in parallel, we ask questions about a further $k - k_i$ regions which we have not asked the crowd about previously. It can be seen that this solution never asks more questions than required in this case, and hence, minimises cost, but at the same time, would provide a means to finish in fewer rounds than solution (1). More precisely, chooseCandidates($k, D, R \setminus O$) is as in Algorithm 2.

This algorithm is essentially that in [16] but tailored to spatial finding, where it was shown that the total number of questions it requires is comparable to solution (1), when both are operating on the same input.

[1] $K-k$ if we have that only positive responses need reply. Sometimes, no reply could be taken as a negative response.

A key feature of the algorithm is how $chooseCandidates(\cdot)$ actually selects the regions to ask about. In contrast to Algorithm 2 which, in each round, randomly selects a region to ask about (which we call the *random spatial crowdsourcing*), the algorithm we propose later will use a neighbourhood association heuristic to select regions to ask about which will be called *associative spatial crowdsourcing.* We first discuss the performance of two versions of random spatial crowdsourcing below, one without a heuristic as above and another with a heuristic that embraces failure.

Analysis of spatial crowdsourcing via solution (3) In the worst case, the total cost is $|R| \cdot \phi$ with the largest number of rounds $|R|$. The best case total cost is $k \cdot \phi$ with the least number of rounds being 1.

Let us consider the average case. We describe the typical case of the algorithm via a *success factor* $0 < \sigma \leq 1$, where we assume that in each round, the fraction of queries answered positively with F evaluating to *TRUE* is σ. In random spatial crowdsourcing, we would have $\sigma = \alpha$, where α, as given earlier, is the proportion of regions in R where F evaluates to *TRUE*, since our choice of regions to ask about in each round is random.

The algorithm uses k queries in round 1, $k - \lceil k \cdot \sigma \rceil$ queries in round 2, given an expected $\lceil k \cdot \sigma \rceil$ successes from round 1, $(k - \lceil k \cdot \sigma \rceil) - \lceil (k - \lceil k \cdot \sigma \rceil) \cdot \sigma \rceil$ queries in round 3, given $\lceil (k - \lceil k \cdot \sigma \rceil) \cdot \sigma \rceil$ successes from round 2, $(k - \lceil k \cdot \sigma \rceil) - \lceil (k - \lceil k \cdot \sigma \rceil) \cdot \sigma \rceil - \lceil ((k - \lceil k \cdot \sigma \rceil) - \lceil (k - \lceil k \cdot \sigma \rceil) \cdot \sigma \rceil) \cdot \sigma \rceil$ queries in round 4 and so on. In general, let $Q(k)$ denote the total number of queries used to find k positive regions using the algorithm. Then, Q is given by:

$$Q(0) = 0$$
$$Q(n) = n + Q(n - \lceil n \cdot \sigma \rceil)$$

The reason for this is that with a success rate of σ, to find n positive regions, we first use n queries to find $\lceil n \cdot \sigma \rceil$ successes, and then to find the remaining $n - \lceil n \cdot \sigma \rceil$ positive regions, we use $Q(n - \lceil n \cdot \sigma \rceil)$ questions.

Thus, to look for k positive regions, the total cost of the algorithm is $Cost(k) = \phi \cdot Q(k)$.

The number of rounds taken by the algorithm is $T(k)$, where the function T is as given by the following:

$$T(0) = 0$$
$$T(n) = T(n - \lceil n \cdot \sigma \rceil) + 1$$

We will use the following lemma later. (The Appendix contains proofs of all lemmas and theorems.)

Lemma 1 *The function T is monotonically increasing, i.e. for any m, $T(n) \geq T(m)$ for all $n \geq m$.*

Intuitively, a larger σ can improve performance both of the cost and number of rounds of the algorithm. But α is assumed fixed, and so, we introduce heuristics to increase the success rate of queries in each round.

A heuristic that embraces failure: redundant questioning

We set $\sigma = \alpha$ which means that we use the proportion of regions in R where F evaluates to *TRUE* as an estimate of the success factor. Now, since each query only has an α chance of success, we can improve performance by having more queries in each round as also noted in 17]: we will have γ times more, where $1 \leq \gamma \leq 1/\alpha$. Note that if $\gamma = 1$ means zero redundancy as in the solution above in "A crowdsourcing solution" section. That is, we can obtain k regions faster by having $\gamma \cdot (k - k_i)$ queries in the next round $i+1$, where k_i is the number of regions found to be *TRUE* so far, up to and including round i. In round $i+1$, by asking more queries, the number of successes is then $\lceil (\mathit{number\ of\ queries}) \cdot \alpha \rceil = \lceil (\gamma \cdot (k - k_i)) \cdot \alpha \rceil$.

This slight variation to solution (3) above is given by the definition of $\mathit{chooseCandidates}(\cdot)$ in Algorithm 3.

```
ALGORITHM 3: chooseCandidates($k, D, R \setminus O$)
Input: $k$:number of positive regions to find; $D$:set of found positive regions so far; $O$:set of
        observed regions, $R$:set of all regions
Output: $C$ is a set of selected unobserved regions to be queried
$c := \lceil \gamma \cdot (k - |D|) \rceil$;
$C$ := randomly select $c$ regions from $R \setminus O$;
$return\ C$;
```

We call this *random spatial crowdsourcing with redundancy (RSC-R)*, when $\gamma > 1$, and *random spatial crowdsourcing with no redundancy (RSC-NR)* when $\gamma = 1$ (the algorithm given earlier). In general, let $Q'_\gamma(k)$ denote the total number of queries used to find k positive regions using this algorithm. Then, Q'_γ is given by:

$$Q'_\gamma(0) = 0$$
$$Q'_\gamma(n) = \lceil \gamma \cdot n \rceil + Q'_\gamma(n - \lceil \lceil \gamma \cdot n \rceil \cdot \sigma \rceil)$$

To find n positive regions, RSC-R starts with $\lceil \gamma \cdot n \rceil$ queries, finding $\lceil \lceil \gamma \cdot n \rceil \cdot \sigma \rceil$ positive regions, and then to find the remaining $(n - \lceil \lceil \gamma \cdot n \rceil \cdot \sigma \rceil)$ regions, it uses $Q'_\gamma(n - \lceil \lceil \gamma \cdot n \rceil \cdot \sigma \rceil)$ queries.

Lemma 2 *Given γ, the function Q'_γ is monotonically increasing, i.e. for any m, $Q'_\gamma(n) \geq Q'_\gamma(m)$ for all $n \geq m$.*

Thus, to look for k positive regions, the total cost of the algorithm RSC-R is $Cost'_\gamma(k) = \phi \cdot Q'_\gamma(k)$.

For a given $\gamma \geq 1$, and a given requirement k, the number of rounds is $T'_\gamma(k)$, where T'_γ is given by the function:

$$T'_\gamma(0) = 0$$
$$T'_\gamma(n) = T'_\gamma(n - \lceil \lceil \gamma \cdot n \rceil \cdot \sigma \rceil) + 1$$

We also have the following relationship between the number of rounds taken by RSC-R (denoted by T') and the number of rounds taken by RSC-NR (denoted T).

Theorem 1 *For any* $\frac{1}{\sigma} \geq \gamma \geq 1$ *and a given non-negative integer* n*, we have* $T'_{\gamma}(n) \leq T(n)$.

And the following relationship between the cost of RSC-R (denoted by $Cost'$*) and the cost taken by RSC-NR (denoted* $Cost$*).*

Theorem 2 *For any* $\frac{1}{\sigma} \geq \gamma \geq 1$ *and a given non-negative integer* n*, we have* $Q'_{\gamma}(n) \leq \lceil \gamma \rceil \cdot Q(n)$, i.e. $Cost'_{\gamma}(n) \leq \lceil \gamma \rceil \cdot Cost(n)$.

Theorem 1 means that RSC-R can result in fewer rounds than RSC-NR, but according to Theorem 2, it is no worse than a factor of $\lceil \gamma \rceil$ *in terms of costs.*

We conducted experiments with this technique of posing more queries (with the expectation of $\sigma = \alpha$ *proportion of successes) in each round to see how it helps the performance and how much extra costs it incurs.*

Experiments

Typical run

In comparing RSC-NR and RSC-R, we use the area illustrated in Fig. 1a, where the '1's represent regions where F evaluates to *TRUE* and the '0's represent regions where F evaluates to *FALSE*, with $\sigma = \alpha = 0.2075$. In a run with $k = 20$, and the total number of regions is 1600 (40 × 40 grid), Fig. 1b shows the observed regions as a result of running RSC-NR which completed in nine rounds with 73 questions (73 regions observed). The execution proceeded as follows in this run, showing the number of questions asked in each round, and the number of '1' regions found from asking those questions in that round:

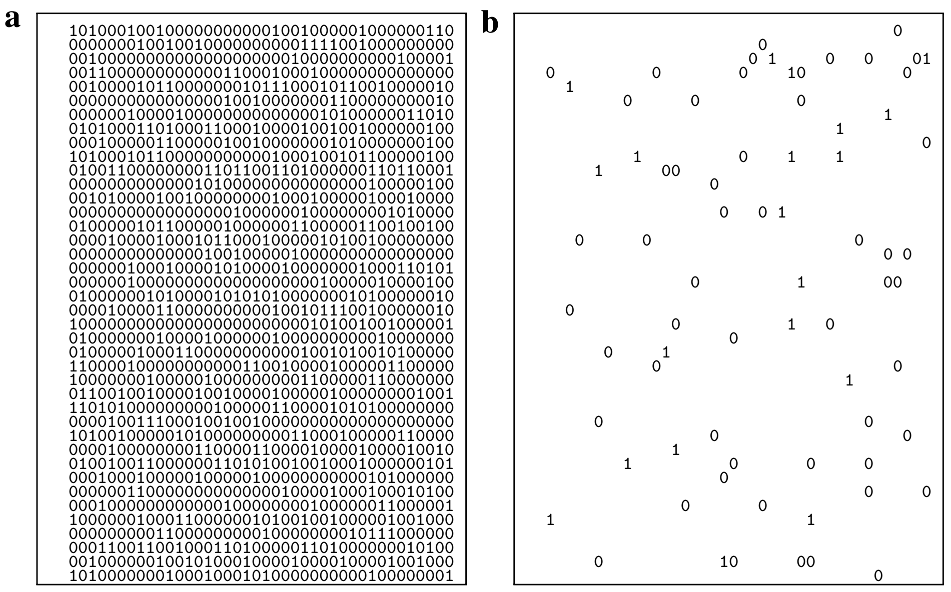


Fig. 1 A test scenario with RSC-NR, and the observed regions after finding $k = 20$ positive regions. **a** Test Scenario: 1600 regions, the number of '0's is 1268 (79.25 %); the number of '1's is 332 (20.75 %). **b** Observed regions from RSC-NR; after 73 questions, nine rounds

```
20 qns in round 0, found:6
14 qns in round 1, found:4
10 qns in round 2, found:3
7 qns in round 3, found:0
7 qns in round 4, found:1
6 qns in round 5, found:1
5 qns in round 6, found:2
3 qns in round 7, found:2
1 qns in round 8, found:1
```

In a run from RSC-R with $\gamma = 2.41$, we get:

```
49 qns in round 0, found:9
27 qns in round 1, found:6
13 qns in round 2, found:1
10 qns in round 3, found:1
8 qns in round 4, found:3
```

It can be seen that asking more (redundant) questions each round results in a larger number of '1's found resulting in a faster convergence, in five rounds, instead of nine rounds as in RSC-NR. But the total number of questions asked is 107 questions instead of 73. In an extreme case, in a run from RSC-R with $\gamma = 4.82$, only one round is enough but with 97 questions asked.

Comparing RSC-NR and RSC-R

We compare the performance of RSC-NR and RSC-R using different values of $k = 20$ and different kinds of distributions of positive regions, but with the total number of regions being 1600 (40×40 grid). Below, we give the average number of rounds and average number of question asked over 1000 runs. Note that RSC-NR is the case of RSC-R with $\gamma = 1$.

With $\sigma = \alpha = 0.315625$ (i.e., 31.5625 % of positive regions represented as '1's), we generated a scenario similar to that in Fig. 1a but with more 1s. Setting $k = 20$, and averaging over 1000 runs, the results for 31.5625 % are shown in Fig. 2b, where the horizontal axis is labelled with different values of γ. Similar experiments were carried out with different values of α on 40×40 scenarios with randomly distributed positive regions, but with varying percentages of positive regions, with results shown in Fig. 2a, c–f. We make the following observations:

- It can be seen that even for all values of α tested, using additional questions in each round can substantially reduce the number of rounds even to 1, without substantially increasing the number of questions asked, since convergence is quick. For example, averaging over 1000 runs, with $\alpha = 0.40125$ in Fig. 2a, with $\gamma = 1.246$, we use a total of four rounds on average with 50 questions on average, compared to using seven rounds on average and 49 questions on average with $\gamma = 1$ (or no redundant questions), and with $\gamma = 2.492$, only 1 round is used on average, with 53 questions used on average.
- When α is smaller, the cases where '1's are sparse and harder to find, we see that more questions are asked with RSC-R but resulting in much greater reduction in the number of rounds. For example, with $\alpha = 0.110625, 0.050625, 0.034375$ in Fig. 2d–f, increasing γ results in a substantial drop in the number of rounds (e.g., 31 to 11,

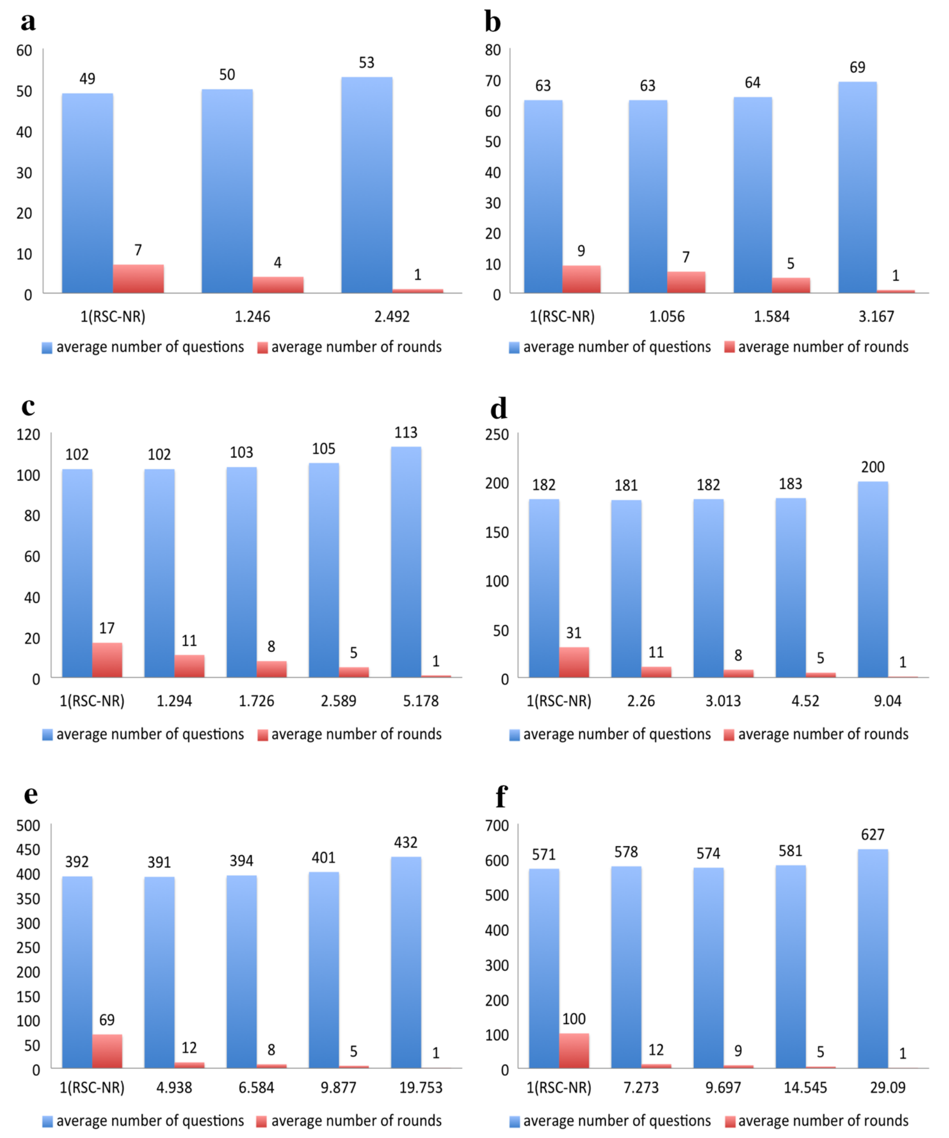


Fig. 2 Comparing RSC-R and RSC-NR: the number of rounds and number of questions averaged over 1000 runs (*vertical axis*) for varying γ (*horizontal axis*) for different values of α. **a** Results with $\alpha = 0.315625$. **b** Results with $\alpha = 0.315625$. **c** Results with $\alpha = 0.193125$. **d** Results with $\alpha = 0.110625$. **e** Results with $\alpha = 0.050625$. **f** Results with $\alpha = 0.034375$.

69–12, and 100–12), with only a small increase in the number of questions asked on average.

- In all cases of α tested, for a large enough γ, on average, asking 11 % more questions in total can lead to a reduction to only 1 round required.

In summary, the above results show that additional questions in each round lead to faster convergence towards the required number of positive regions (i.e., taking fewer rounds), though asking more questions in earlier rounds. And the faster convergence offsets the larger number of questions asked in earlier rounds so that, overall, only a small percentage increase in the total number of questions is required, i.e. there is only a relatively small price to pay for faster results.

Discussion

The results show that if we ask $1 < \gamma \leq \frac{1}{\alpha}$ more questions (the additional questions are so-called *redundant* questions) than the minimum required in each round, we can significantly reduce the number of rounds, and because we reduce the number of rounds, we end up asking only a small number of additional questions than we needed to in total, compared to no redundancy. We see that this result holds for a large range of scenarios. However, it must be noted that the gains are greater only when α is small and large enough γ, and the value of γ relies on some prior knowledge (or estimate) of α, which might be difficult to obtain in practice; using too small a γ does not result in much improvements in efficiency but using too large a γ wastes questions. The result of average 1 round in Fig. 2a–f are all obtained with γ having the value approximately $\frac{1}{\alpha}$, as each of the $k \cdot \frac{1}{\alpha}$ questions randomly chooses a region with $\frac{1}{\alpha}$ chance of being positive.

RSC-R above fails to take into account the clustering of positive regions. For example, Fig. 3a, b have similar results, as shown in Table 1. We examine a heuristic that takes advantage of such clustering in the next section.

a

```
0010000001000000100000000000000000001000000000
0000000000100100010000100100000000000000001
0011000000000101000010011000000010000000000
1000000000000000000010111000000000000100010
0010010000000000111100000010000100111110110
10001110000110101110010000000000101111110
1001111000111100011000000100000011111110
0101111001111110000000000011001010011111111
0001111000111100010000000000000000001111111
0000001001111110000110100000000000101111110
0010000000111100000000000000000000000111110
0000000000111100000010010000000001001111110
1000010001111000010001000100000000111111
1100100000111000010000000000000000000111111
1110000000110001100110000000001000000011111
0110000001000001100000000000000000001010110
0111100010000011000010100101000111011111
0001110100111111100000000110001110001110
0000111100111111100000100010010000001000
0000011100110011100000000011100001100000
1000000000000000000000000111110000000001
1111000000000100000000000111111000101000
1110001100100110000000001111111111111100
0000001100000110000010011111111111111110
0000100110000011100011111111111111111110
0000001111000000110001111111111111111110
1000011111110000000001111111111111111110
0000001111111000000110001111111111111110
0000000111111001000100011111111111111110
0110000011111111000000000111111111111110
1100000111111110000100000111111111111110
0000001111111111000000000000111111111110
0000011111111111100000000100000111111110
0000001111100000000100000001100011111110
0001001110000000010001000000000111111110
0000000100000001100010000000000111111110
0110110000010011000000010000100000111111
0000000001110000100110000010000000111110
0000001011110000000000000000000100101110
0000000000000001000000010100000000000011010
```

b

```
1101010101011010100001111001110010111100
0100010110011000011011000001101000100011
1100110100110000011001010100100000000000
00010010010000001101100010000000011100110
00100000001010000001100111001001000010011
0101000110100100101010001010010010010110
01111010100000001101010001010110000100101
1100100011011010001000101100010001101100
1010110001000010001100000001011100010100
1010010100010110100110100100111111000100
0010101101000100100101110011101111100001
1000010100001111100001000000000110000000
1110110000100101011111100000100001001010
0010101111010100010011000000001100000010
0000000001100101110111010011000001000010
11100011011111110000111001100000011001100
0100100010101101000111101000100000000100
1010110011001001010110111011001000101010
0000101001100000010010011110011010011111
1000001000011010011011111000100011111110
101010110001000000010001000000001000011101
000111011010000001110000101000000000011110
1110100111001011100000100010110000011111
11010000001111000011111010000000101010000
01011111000000010001101100110100101110001
11000000001101100000010010010101111111101
00001000000011100110011110101000011101010
0110100010001100001001100100101001100010
0111100000000000010000010000111100011001
11001111000100000001011001010110000000000
0101011011010000001010000111101100101100
1111111000011110010010011010100100010010
0111101001010100011100010000100001011000
1101001100100101011000011000001100101100
0100001010110000011101111110100100000010
0110000101101011000000010111010011100000
1110101010010110000101101111010010100101
1110001100011010111111000101000100001010
1110010000000111000011110100010101110010
0110001110010110010101010110000011010000
```

Fig. 3 Two regions with roughly 40 % of positive regions but different in the amount of clustering. **a** Region with $\alpha = 0.40125$ but more clustering. **b** Region with $\alpha = 0.418125$ but less clustering

Table 1 Results for $\alpha = 0.418125$ ($\alpha = 0.40125$) with less clustering (more clustering), averaged over 1000 runs

γ	1 (1)	1.196 (1.246)	2.392 (2.492)
Avg. no. of questions	47 (49)	48 (50)	52 (53)
Avg. no. of rounds	6 (7)	4 (4)	1 (1)

A heuristic for spatial finding: neighbourhood associations

In the case where we know nothing about the unobserved regions, any selection is as good as any other. The solutions have not so far considered the case where knowing something about a region tells us something about another region. For example, we assume initially no information, i.e. $F(r) = TRUE$ with probability α (and $F(r) = FALSE$ with probability $1 - \alpha$). The value of α may be estimated from some initial density measure if there is some a priori information, but here, we take α to be the true proportion of regions in R that satisfy F. Hence, the success factor σ has been approximated via α in the experiments above.

It can be seen from the definitions of Q, Q', T and T' above and their monotonically increasing properties that if σ was to increase, we can reduce the number of questions and the number of rounds. In this section and the next section, we consider heuristics that can improve the success factor in each round of querying.

Given direct observation of a region, then $F(r)$ must evaluate to true or false, but without direct observation of a region, we can only compute the probability of $F(r)$ being true or false in some way. Note that we say we *observe* a region whenever we ask the crowd a question about it.

To estimate $Pr(F(r) = TRUE)$ given that r has not been observed, we introduce the neighbourhood association factor δ (>0) which represents the informational relationship between neighbouring regions, where knowing something certain about a region q tells us something about its neighbouring region, i.e., if r and p are two neighbouring regions, then if we observed that $F(q) = TRUE$, but have not observed r, then, we set:

$$Pr(F(r) = TRUE \mid F(q) = TRUE) = \alpha \cdot (1 + \delta)$$
$$Pr(F(r) = FALSE \mid F(q) = TRUE) = 1 - \alpha \cdot (1 + \delta)$$

where δ is chosen so that $0 \leq \alpha \cdot (1 + \delta) \leq 1$, and also, if we directly observed that $F(q) = FALSE$, using Baye's rule:

$$\begin{aligned} &Pr(F(r) = TRUE \mid F(q) = FALSE) \\ &\qquad = Pr(F(r) = TRUE) \cdot \frac{Pr(F(q) = FALSE \mid F(r) = TRUE)}{Pr(F(q) = FALSE)} \\ &\qquad = \alpha \cdot \frac{1 - \alpha \cdot (1 + \delta)}{1 - \alpha} \\ &Pr(F(r) = FALSE \mid F(q) = FALSE) = 1 - \alpha \cdot \frac{1 - \alpha \cdot (1 + \delta)}{1 - \alpha} \end{aligned}$$

For example, if α is 0.5 and δ is 0.1, then $Pr(F(r) = TRUE \mid F(q) = TRUE) = 0.55 > 0.5$. In other words, as we observe more regions, given the association among regions, we might be able to do better than randomly selecting a set of regions to ask about in each round; we can select regions with a higher probability of evaluating F to $TRUE$ based on such association information. Also, if a region should be false with probability $1 - \alpha$, on observing that its neighbour is $TRUE$, its probability of being $FALSE$ is reduced.

More precisely, let N be a function that returns the immediate neighbours of a region, i.e. $N(r) \subseteq R$ is the set of regions sharing a boundary with r defined in some way. $N(r)$ would have eight members at most if R is divided into a grid of rectangular regions (including diagonally adjacent regions).

From the point of view of the unobserved region r, it is possible that multiple neighbouring regions have been observed, and so, we need to combine the influence from multiple observed neighbours.

For example, for a given region, if one of its neighbours q_1 is found that $F(q_1) = FALSE$ and another two neighbours q_2 and q_3 are such that $F(q_2) = F(q_3) = TRUE$, then by Bayes' rule (where $H = \alpha \cdot (1 - \alpha \cdot (1 + \delta)) \cdot (\alpha \cdot (1 + \delta)) \cdot (\alpha \cdot (1 + \delta))$):

$$\begin{aligned} &Pr(F(r) = TRUE \mid (F(q_1) = FALSE \wedge F(q_2) = TRUE \wedge F(q_3) = TRUE)) \\ &= \frac{H}{H + (1-\alpha) \cdot \left(1 - \alpha \cdot \frac{1-\alpha\cdot(1+\delta)}{1-\alpha}\right) \cdot \left(\alpha \cdot \frac{1-\alpha\cdot(1+\delta)}{1-\alpha}\right) \cdot \left(\alpha \cdot \frac{1-\alpha\cdot(1+\delta)}{1-\alpha}\right)} \end{aligned}$$

Given a region r, and that we observed some subset of the neighbours of r, say $(A \cup B) \subseteq N(r)$, where A are neighbours where F evaluated to FALSE and B are neighbours where F evaluated to TRUE, then using $obs(N(r))$ to denote the observed neighbours of r, by a Bayesian approach of combining information, we have what we call the *neighbourhood formula*, where $H' = \alpha \cdot (1 - \alpha \cdot (1 + \delta))^{|A|} \cdot (\alpha \cdot (1 + \delta))^{|B|}$:

$$\begin{aligned} &Pr\left(F(r) = TRUE \mid \bigwedge\nolimits_{p\in A}(\mathrm{F(p) = FALSE}) \wedge \bigwedge\nolimits_{q\in B}(\mathrm{F(q) = TRUE}) \wedge \mathrm{(A \cup B) = obs(N(r))}\right) \\ &= \frac{H'}{H' + (1-\alpha) \cdot (1 - \alpha \cdot \frac{1-\alpha\cdot(1+\delta)}{1-\alpha})^{|A|} \cdot (\alpha \cdot \frac{1-\alpha\cdot(1+\delta)}{1-\alpha})^{|B|}} \end{aligned}$$

Note that the above is merely a heuristic for estimating the probability of a region satisfying F; our guess could turn out completely wrong upon observation, i.e. given current observations *obs*, we estimate that $Pr(F(r) = TRUE \mid obs) > 0.5$ but we later may observe that $F(r) = FALSE$. Also, for simplicity, we have taken a Markov-inspired assumption in that we compute the probability based only on observed regions in the neighbourhood of r, and do not consider any influence from regions beyond the neighbourhood, i.e., using *obs* (R) to denote observed regions in the entire area R:

$$Pr(F(r) = TRUE \mid obs(N(r))\ ) = Pr(\ F(r) = TRUE \mid obs(R)).$$

If we are using solution (3), in each round i, for simplicity, we compute probabilities only for regions not yet observed, with the aim of choosing the $k-k_i$ regions most likely to evaluate F to *TRUE*, and we use only observed information. For example, an unobserved region r that has no observed neighbours will have $Pr(F(r) = TRUE) = \alpha$ even if all its unobserved neighbours q have estimated $Pr(F(q) = TRUE \mid obs) > \alpha$ given some observations *obs*.

In the previous random spatial crowdsourcing algorithm, in *SpatialCrowdsourcing (k, F, R)* given above, chooseCandidates($c, R\backslash O$) chooses c candidates from $R\backslash O$ in a random way, and in the associative spatial crowdsourcing algorithm, chooseCandidates($c, R\backslash O$) chooses c candidates from $R\backslash O$ by selecting the c regions with the highest probability of F evaluating to *TRUE*, i.e., for each region $r \in R\backslash O$, we compute the probability of $F(r) = TRUE$ using the neighbourhood formula above and select c regions with the highest probabilities according to the formula, randomly selecting among equal probability regions.

This slight variation to solution (3) above using neighbourhood association is given by this definition of *chooseCandidates*$(\cdot)$ in Algorithm 4.

```
ALGORITHM 4: chooseCandidates($k, D, R \setminus O$)
Input: $k$:number of positive regions to find; $D$:set of found positive regions so far; $O$:set of
       observed regions, $R$:set of all regions
Output: $C$ is a set of selected unobserved regions to be queried
$c := k - |D|$;
$C$ := select $c$ regions from $R \setminus O$ with the highest $Pr(\ F(r) = TRUE \mid obs(N(r))\ )$
computed on each $r \in R \setminus O$, choosing randomly among equal probability regions;
$return\ C$;
```

Experiments with randomly generated area maps

We study the effect that the extent of clustering has with the use of this heuristic as *k* varies and as δ is varied. In the first set of experiments, we generate area maps with α set to values within the range [0.15, 0.20] and clustering introduced so that where whenever there are three '1's surrounding a region, the region will be a '1' (otherwise the region is either '1' or '0' with equal probability).

Figure 4 shows the results of associative spatial crowdsourcing compared with random spatial crowdsourcing (RSC-NR) as *k* is varied for a range of δ values—the number of questions used and the number of rounds used are averages over 1000 runs with the same region map. It can be seen that with even small $\delta(=0.1)$, associative spatial crowdsourcing yields, on average, both a significant reduction in both the number of questions used (up to 30–40 %) and the number of rounds required to find the *k* positive regions (as low as a third or half of the number of rounds required with RSC-NR). The reductions are proportionately larger with larger *k*. Larger values of $\delta(>0.1)$ do not seem to yield much improvement.

Note, however, that with little clustering, associative spatial crowdsourcing provides little to nor advantage, and can even do slightly worse in case it assumed clustering when there wasn't any. However, as we show in the following examples, contiguous and clustered regions (fortunately) occur in a range of real-world scenarios. Below, we use maps sourced from real-world applications as a starting point representing the current state of the world from which we want to find regions of interest.

Experiments on finding parking

We consider using spatial crowdsourcing to look for regions with parking spaces. For our experiments, we use a parking map abstracted from a San Francisco parking census data, dividing an area into 26×20 regions, as illustrated in Fig. 5, which shows the location of parking lots. The problem we address here is then: given the parking map, which we assume here captures the current state of the world with regard to parking in that area, we want to find $k = 5$ or $k = 40$ regions where there is parking available, using crowdsourcing. (Note that, in reality, there could be fewer regions with available parking since some of the parking spaces would have been taken up.) Hence, a query will ask if

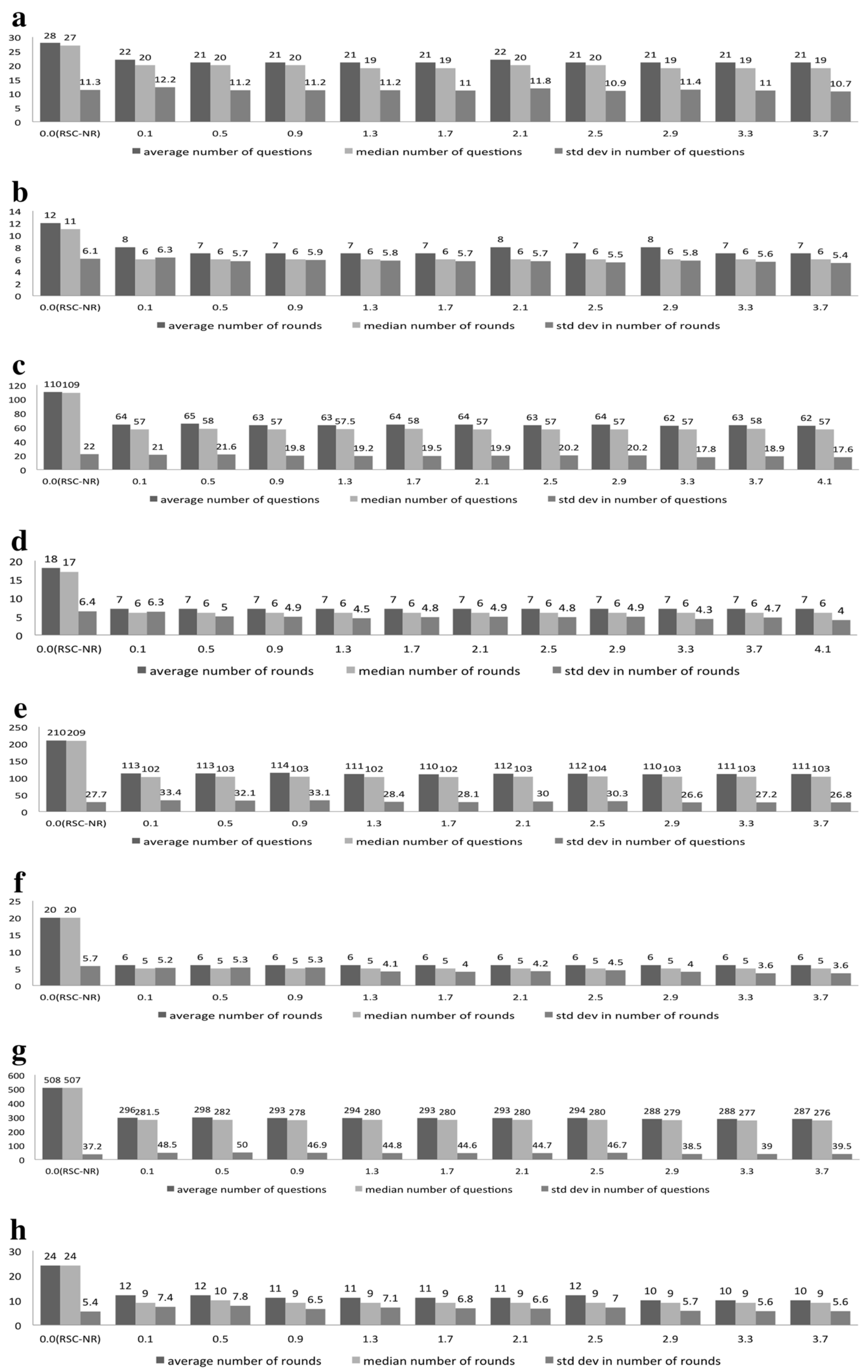
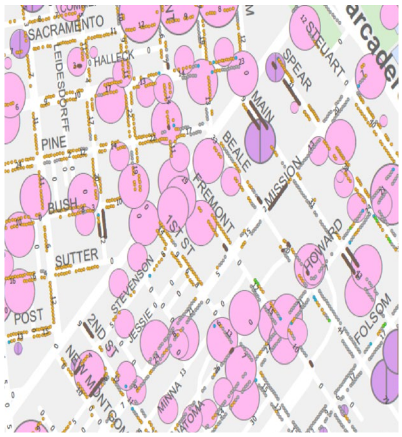


Fig. 4 The number of rounds and number of questions averaged over 1000 runs (*vertical axis*) for varying δ (*horizontal axis*) for different values of k while maintaining a similar α. **a** Average, median and std. dev. for number of questions with $k = 5$ and $\alpha = 0.175$ as δ varies. **b** Average, median and std. dev. for number of rounds with $k = 5$ and $\alpha = 0.175$ as δ varies. **c** Average, median and std. dev. for number of questions with $k = 20$ and $\alpha = 0.179375$ as δ varies. **d** Average, median and std. dev. for number of rounds with $k = 20$ and $\alpha = 0.179375$ as δ varies. **e** Average, median and std. dev. for number of questions with $k = 40$ and $\alpha = 0.189375$ as δ varies. **f** Average, median and std. dev. for number of rounds with $k = 40$ and $\alpha = 0.189375$ as δ varies. **g** Average, median and std. dev. for number of questions with $k = 100$ and $\alpha = 0.195625$ as δ varies. **h** Average, median and std. dev. for number of rounds with $k = 100$ and $\alpha = 0.195625$ as δ varies

a



b

```
00000000111101100001000000
00000000000000000001000000
00001000000010011000000100
10000001001011010000000110
10000011011000000000000000
10000000001000000000010000
00000001000101001000111000
00000001101101100000000010
01101100101100000000001010
01000100111111000000000000
11000000110010000000000000
10000000100000000001000100
11000001000000000000001100
11000000000000011010001100
00000001010100111111000000
00000000010101100010000000
00000101100000110000000010
00000000100000110000000010
00000000001011010000000000
00000000000000000000000000
```

Fig. 5 Parking map and its abstract version. **a** Parking map from http://sfpark.org/resources/parking-census-data-context-and-map-april-2014/, of size roughly 1 by 0.7 km; pink regions have parking spaces. **b** Abstracted discretized view of parking map with $\alpha = 0.2076923076923077$ with 26×20 regions, each region corresponding to roughly 37 by 37 m; '1's representing regions with parking spaces.

there are parking spaces in a region of size 37 by 37 m, and for simplicity, answers are binary, YES or NO, and we assume truthfulness in answers given.

Figure 6 shows the average over 1000 runs of results (number of questions and number of rounds) with two values of k (5 and 40). The median and standard deviation are included to indicate there is a fair amount of variability between runs. With $k = 5$ in Fig. 6a, b, we see that with large enough δ (e.g., 1.7), i.e., using a strong association between neighbouring positive regions), the algorithm can effectively zoom in on positive regions faster than a random approach (RSC-NR, i.e., $\delta = 0$), resulting, on average, with both 40 % reduction in the number of rounds and 25 % reduction in the number of questions used at the same time, i.e., it is not a trading off rounds with questions but reduction in both. However, with the standard deviation of sometimes over 40 % of the average rounds and questions, there is substantial variability among runs so that gains can be small. A similar result is observed for $k = 40$, 100 in Fig. 6c–h with proportionate reductions in the number of rounds and questions, on average. The type of clustering observed in the parking map made it susceptible to gains using our neighbourhood association heuristic. As before, gains can be obtained just with $\delta = 0.1$, with little improvements for $\delta > 0.1$.

Experiments on finding crowds

In this experiment, we are simulating the use of crowdsourcing to find where the crowds are in a city or urban setting. We use a crowd map obtained from the MIT Citysense project,[2] abstracted into 126×148 regions, each region corresponding to roughly 28.5×28.5m in size. Figure 7 illustrates the map we use that, we assume here, represents the current real state of the urban area, and the problem is then, given this state of the world, to find $k = 5$, 40, 100 or 3000 regions where there are crowds, using

[2] http://www.sensysmag.com/spatialsustain/citysense-app-aims-to-connect-tribes.html.

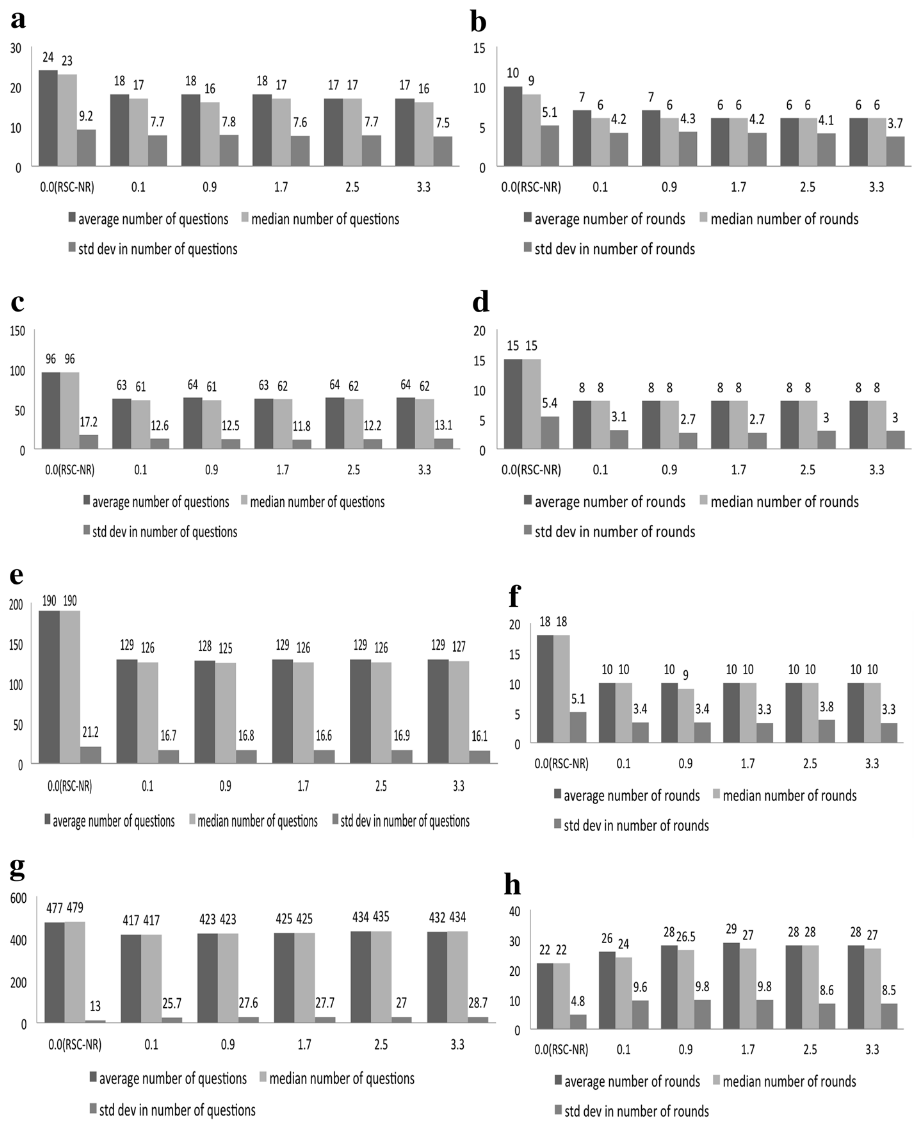


Fig. 6 Results averaged over 1000 runs (*vertical axis*) for varying δ (*horizontal axis*) for four different values of k for the parking map scenario. **a** Average, median and standard deviation for number of questions with $k = 5$ as δ varies. **b** Average, median and standard deviation for number of rounds with $k = 5$ as δ varies. (c) Average, median and standard deviation for number of questions with $k = 20$ as δ varies. **d** Average, median and standard deviation for number of rounds with $k = 20$ as δ varies. (e)Average, median and standard deviation for number of questions with $k = 40$ as δ varies. **f** Average, median and standard deviation for number of rounds with $k = 40$ as δ varies. **g** Average, median and standard deviation for number of questions with $k = 100$ as δ varies. **h** Average, median and standard deviation for number of rounds with $k = 100$ as δ varies.

crowdsourcing. Again, for simplicity, we assume binary answers to a query on each region: is there a crowd here or not?

Figure 8 shows our results when finding $k = 5$, 20, 40, 100 and 3000 crowded regions. Similar to the previous case study, our results show a considerable reduction (up to 70 %) in the number of rounds required and up to 60 % reduction in the number of questions required, on average, with $k = 5$, 100 and 3000. This is due to the clustering in the crowd map, which is to a higher degree than in the parking map. These results show

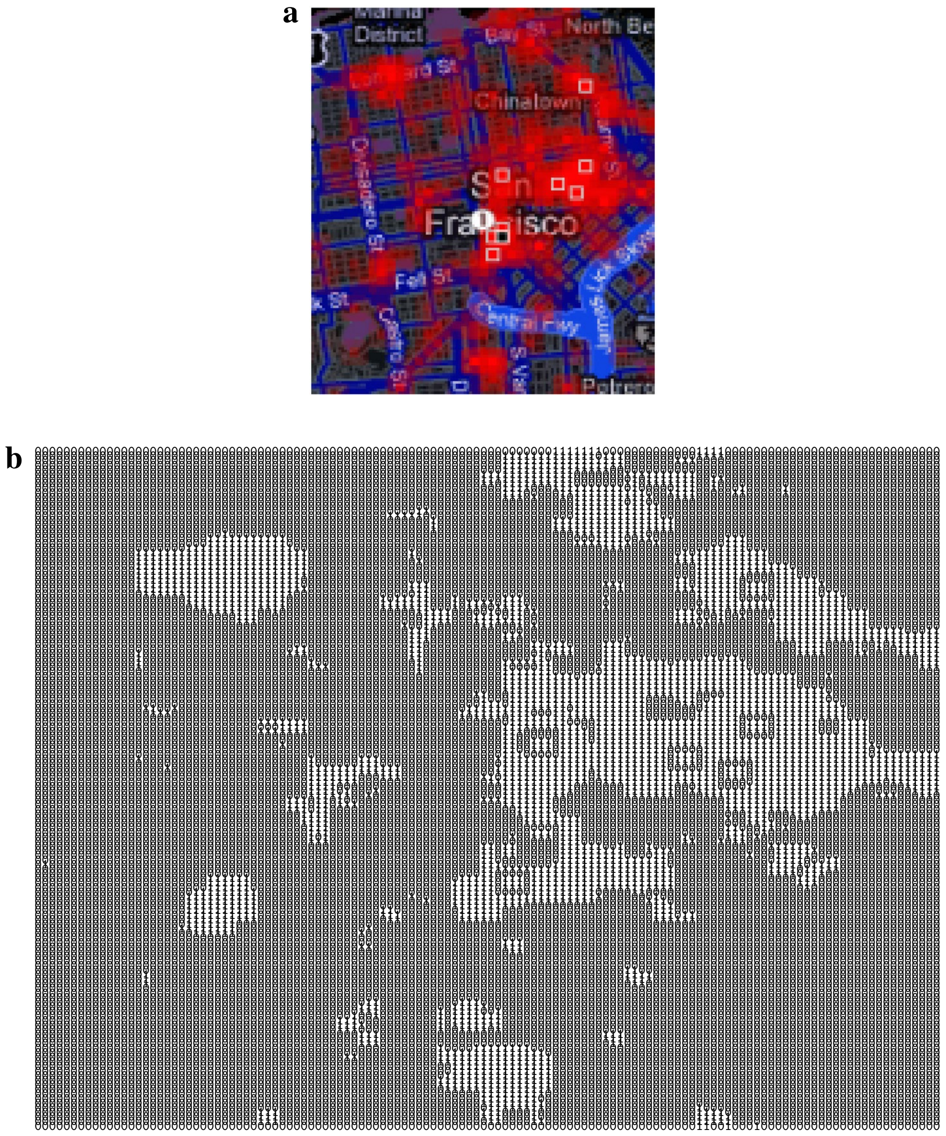


Fig. 7 Crowd map and its abstract version. **a** *Crowd map* from Citysense project (http://www.sensysmag.com/spatialsustain/citysense-app-aims-to-connect-tribes.html) of size roughly 3.6 by 4.2 km; red regions are where the crowds are. **b** Abstracted discretized view of crowd map with $\alpha = 0.1968039468039468$, with 126 × 148 regions, each region corresponding to roughly 28.5 × 28.5m; '1's (or lighter regions) representing regions with crowds

that neighbourhood association can be extremely useful in knowing which regions to ask about when looking for crowded regions—neighbouring regions tend to be crowded.

Experiments on finding coverage/bandwidth

In this experiment, we simulate finding regions where there is coverage (or adequate bandwidth) for 3G/4G networking. The assumed current coverage/bandwidth map is taken from OpenSignal as illustrated in Fig. 9. We want to find $k = 5$, 20, 40, 100 or 1000 regions where there is coverage, using crowdsourcing. We have 103 × 77 regions, each

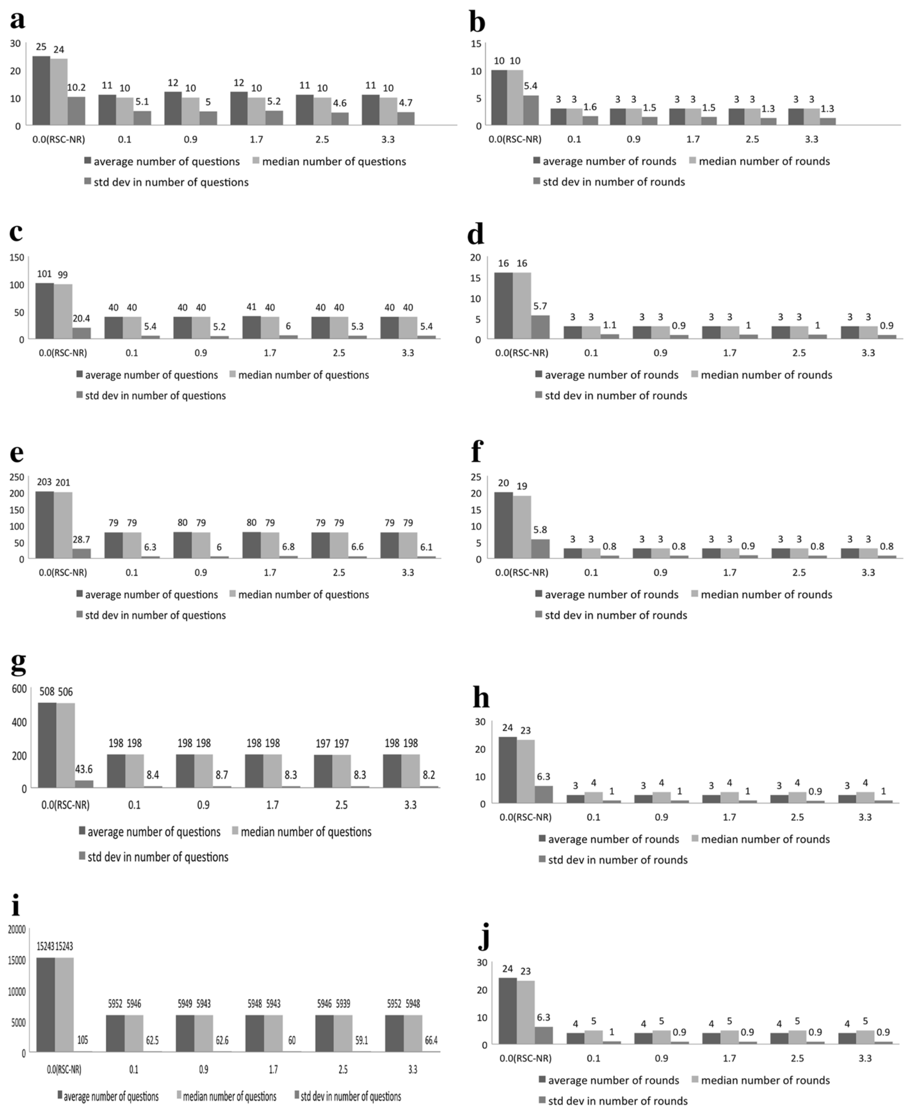


Fig. 8 Results averaged over 1000 runs for varying δ for different values of k for the crowd map scenario. **a** Average, median and standard deviation for number of questions with $k = 5$ as δ varies. **b** Average, median and standard deviation for number of rounds with $k = 5$ as δ varies. **c** Average, median and standard deviation for number of questions with $k = 20$ as δ varies. **d** Average, median and standard deviation for number of rounds with $k = 20$ as δ varies. **e** Average, median and standard deviation for number of questions with $k = 40$ as δ varies. **f** Average, median and standard deviation for number of rounds with $k = 40$ as δ varies. **g** Average, median and standard deviation for number of questions with $k = 100$ as δ varies. **h** Average, median and standard deviation for number of rounds with $k = 100$ as δ varies. **i** Average, median and standard deviation for number of questions with $k = 3000$ as δ varies. **j** Average, median and standard deviation for number of rounds with $k = 3000$ as δ varies

region corresponding to roughly 100 by 100 m in size. Each query will determine if each such region has 3G/4G coverage or adequate bandwidth.

Similar to the previous two experiments, from Fig. 10, a significant reduction (up to 60 %) in the number of rounds required can be achieved and 40–50 % reductions in the number of questions required are observed, with all values of k used.

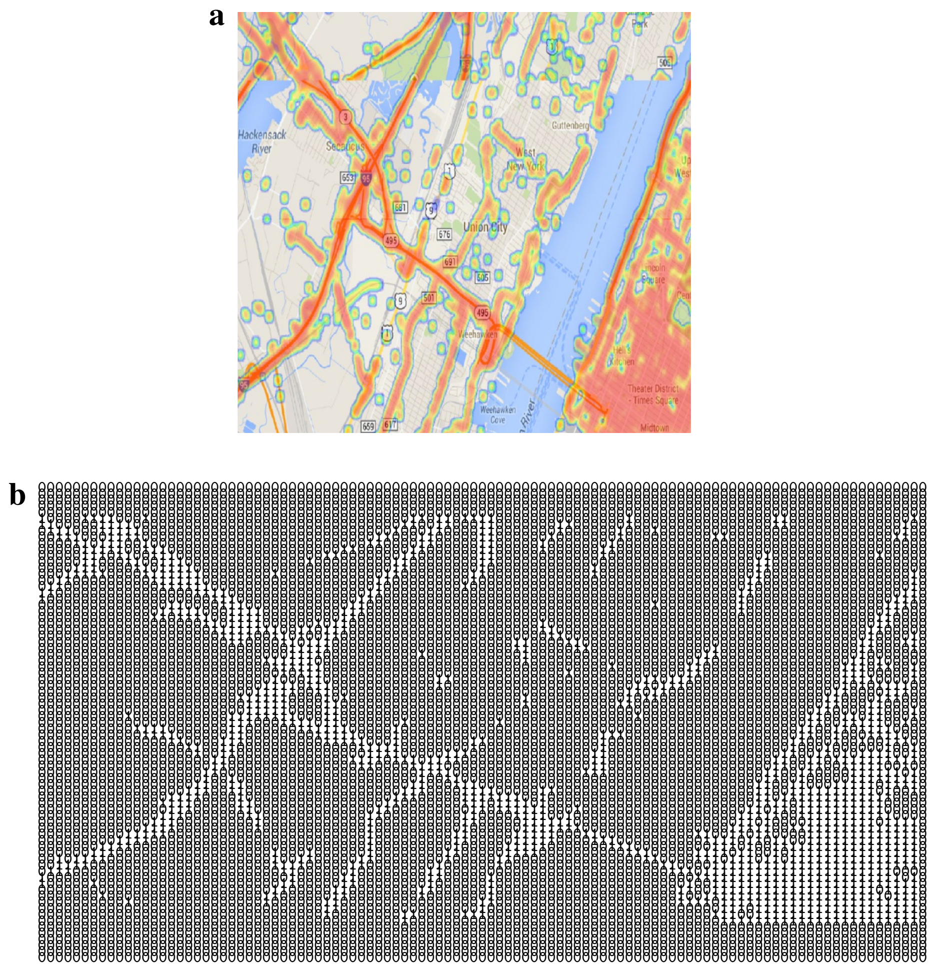


Fig. 9 Coverage map and its abstract version. **a** 3G/4G coverage map from http://opensignal.com/ of a part of New York city, of size roughly 10 by 8 km; orange regions have 3G/4G coverage. **b** Abstracted discretized view of coverage map with $\alpha = 0.1784138191905182$ with 103×77 regions, each region corresponding to roughly 100 by 100 m; 1s representing regions with coverage

A heuristic based on spatial point processes

To go beyond simple immediate neighbourhood influences, we explore an alternative heuristic for choosing potential regions to query about, i.e. to guess the location of positive regions (labelled '1's) within an area, based on modelling the distribution of positive regions using *spatial point processes* [18].

Suppose that an area R has been divided into I disjoint subareas $R_1, R_2, \ldots R_I$, i.e. each R_i has a set of regions. At time t, for a subarea R_i, positive regions are assumed to be distributed according to a Poisson process with intensity $\lambda_i(t)$ (where *intensity* here is defined to be the average number of positive regions per unit area, or the potential of an event to appear at any location). The expected number of positive regions in area R_i is given by $\lambda_i(t) \cdot |R_i|$, where $|R_i|$ denotes the size of (or the number of regions in) R_i, and the positive regions in R_i are assumed distributed uniformly within R_i.

Now, for $p > 1$, suppose that after round $(p-1)$, we have observed N_p regions $r_1, \ldots, r_{N_p}$ (some of which are observed to be positive and some negative). We would

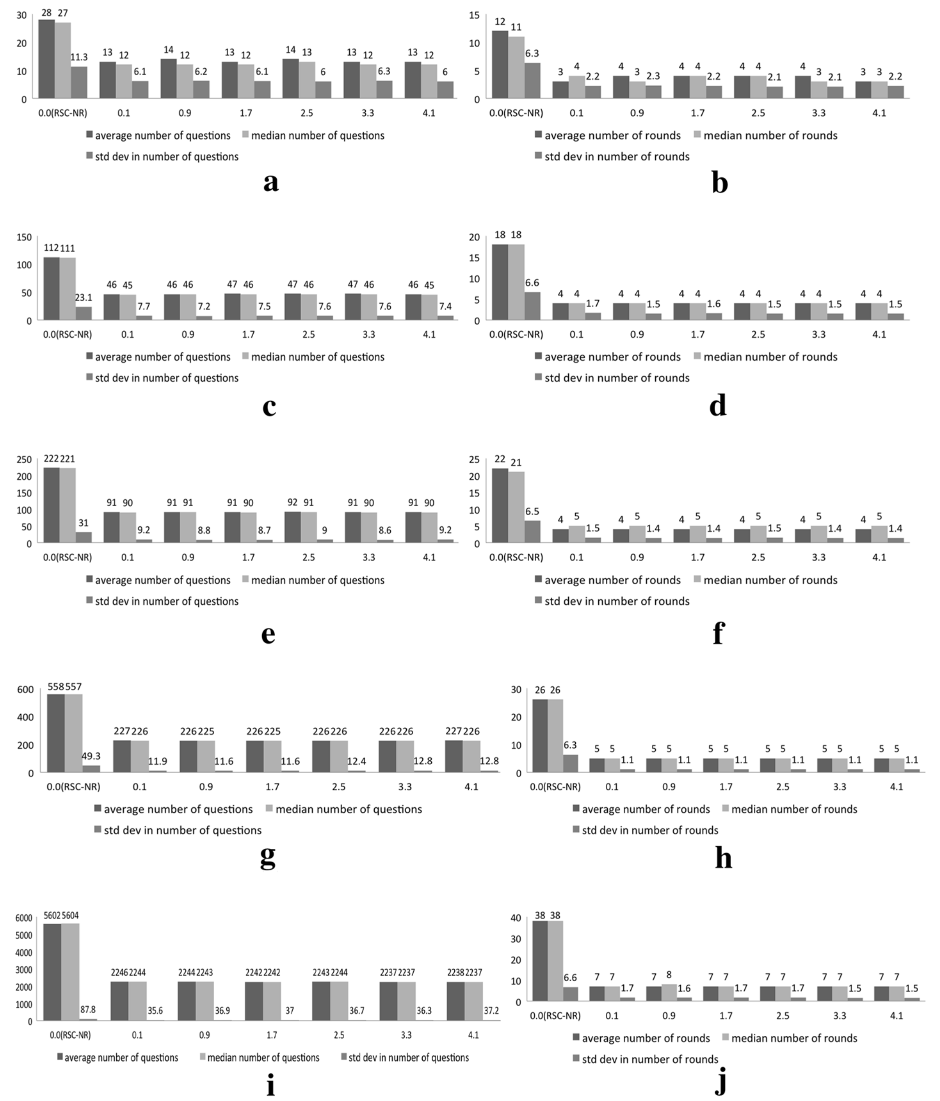


Fig. 10 Results averaged over 1000 runs for varying δ for different values of k for the coverage map scenario. **a** Average, median and standard deviation for number of questions with $k = 5$ as δ varies. **b** Average, median and standard deviation for number of rounds with $k = 5$ as δ varies. **c** Average, median and standard deviation for number of questions with $k = 20$ as δ varies. **d** Average, median and standard deviation for number of rounds with $k = 20$ as δ varies. **e** Average, median and standard deviation for number of questions with $k = 40$ as δ varies. **f** Average, median and standard deviation for number of rounds with $k = 40$ as δ varies. **g** Average, median and standard deviation for number of questions with $k = 100$ as δ varies. **h** Average, median and standard deviation for number of rounds with $k = 100$ as δ varies. **i** Average, median and standard deviation for number of questions with $k = 1000$ as δ varies. **j** Average, median and standard deviation for number of rounds with $k = 1000$ as δ varies

like to obtain the intensity at any region $r \in R$, as an estimate of the probability of the region r being positive. To obtain an estimate of the intensity at any $r \in R$, taking into consideration contributions of both observed positive and negative regions, we use the Nadaraya-Watson kernel weighted average, with bandwidth h:

$$\hat{\lambda}_h(r) = \frac{\sum_{i=1}^{N_p}(K_h(r, r_i) \cdot F(r_i))}{\sum_{i=1}^{N_p} K_h(r, r_i)}$$

with a Gaussian kernel:

$$K_h(r, r_i) = \frac{1}{2\pi} \cdot e^{-\frac{1}{2}\left(\frac{|r-r_i|}{h}\right)^2}$$

where $|r - r_i|$ denotes the Euclidean distance between regions r and r_i (computed using their coordinates), and h is the standard deviation.

Note that we recompute the intensity after each round since we observe more regions after each round and so can improve the model after each round. The method for $\mathit{chooseCandidates}(\cdot)$ in the spatial crowdsourcing algorithm then chooses the c regions with the highest $\lambda_h(\cdot)$.

Below, for short, the associative spatial crowdsourcing algorithm using the immediate neighbourhood heuristic given in the previous section is termed ASC-IN, and the associative spatial crowdsourcing algorithm using the spatial point process modelling is termed ASC-SPP.

Experiments with a randomly generated map

Here, we generated a random scenario with around 25 % of positive regions (or '1's), where there is some clustering but immediate neighbourhoods with contiguous regions of '1's are purposely reduced. We conducted a larger number of runs to compare RSC-NR, ASC-IN (with $\delta = 0.1$) and ASC-SPP, taking average values for questions and rounds. Figure 11 shows examples of observed regions for ASC-CPP and ASC-IN. With $k = 40$, a run of ASC-SPP is shown in Fig. 11a. This can be compared to the run of ASC-IN (with $\delta = 0.1$ only, since larger values do not improve performance as can be seen from the previous section) shown in Fig. 11b. It can be seen that ASC-SPP is able to focus on areas with higher density of '1's without necessary getting 'stuck' at exploring immediate neighbourhoods as in ASC-IN, and as we expect, this has resulted in better performance for this scenario.

Figure 12 shows the results (questions and rounds) for $k = 5$ and 40. It can be seen that ASC-SPP performs better in such a scenario where immediate neighbourhoods of positive regions are mostly not positive. ASC-SPP performs better than ASC-IN, as we would expect, since ASC-IN ends up searching negative neighbourhoods, but ASC-SPP also performs better than the random RSC-NR by directing search via the spatial point process heuristic above (especially for $k = 40$).

Experiments on finding noisy areas

In this experimentation, we consider noise maps, such as in Fig. 13a abstracted as Fig. 13b. In the abstracted map, it can be seen that immediate neighbourhoods of '1's need not be '1's themselves, and we have a case where ASC-SPP might perform better for this type of scenario. Figure 14a and b shows that this is indeed the case, for $k = 40$, with substantial gains over ASC-IN.

But for $k = 5$, as in Fig. 14c and d, there is considerable variation (high standard deviation) and in fact, ASC-SPP has, on average, poorer performance than both ASC-IN and RSC-NR; with $k = 5$, ASC-SPP performs extremely well in some cases but does extremely poorly in particular cases, which raised the average substantially. The reason is as follows: it was found that ASC-SPP can end up exploring large sparse areas. For

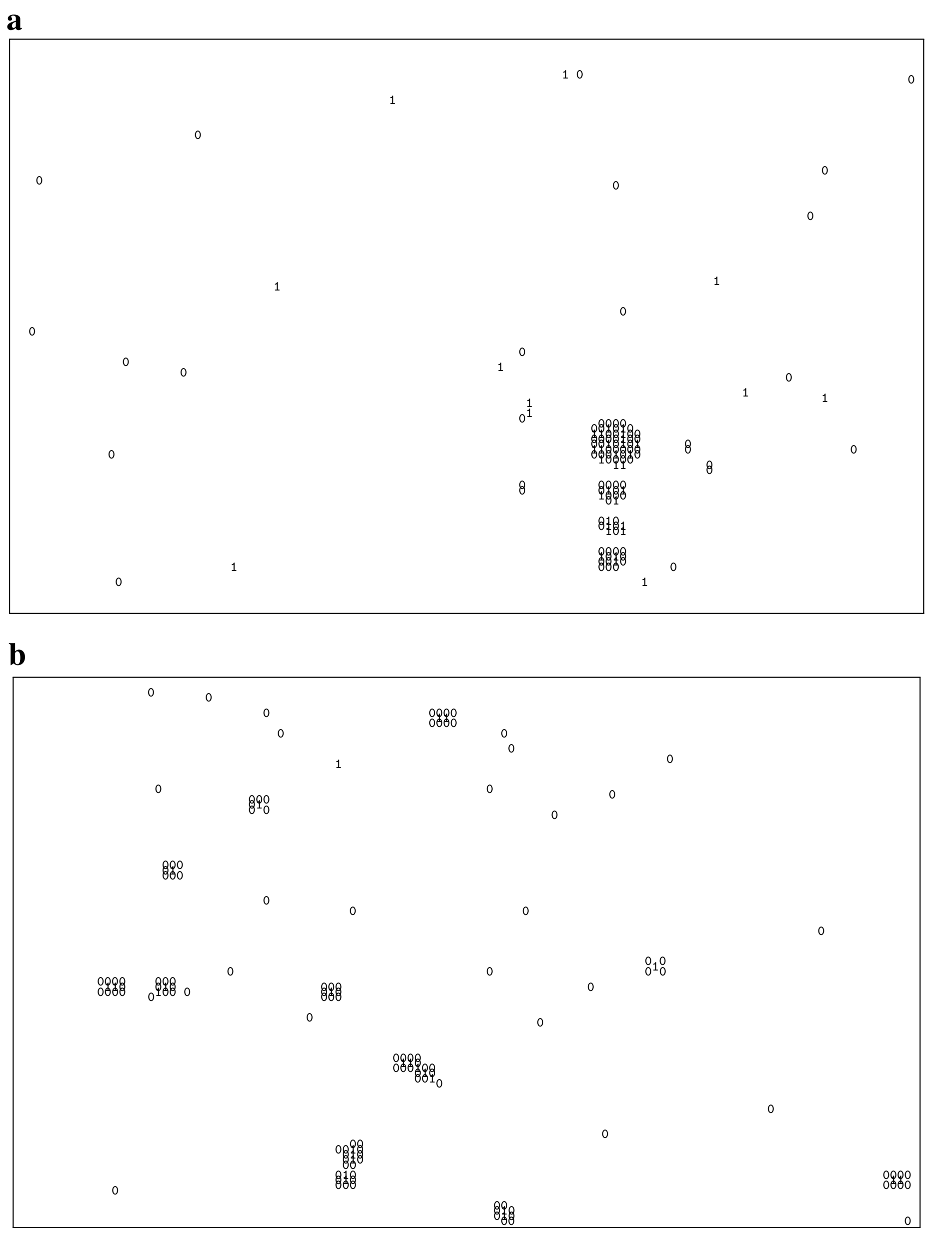


Fig. 11 An ASC-SPP result observing a set of contiguous regions, compared to an ASC-IN result focusing on regions immediately surrounding found '1's (many of which turn out to be '0's in this scenario). **a** Observed regions with ASC-SPP in another run (total qns: 140; total rounds: 15). **b** Observed regions with ASC-IN in one run (total qns: 226; total rounds: 27)

example, dividing the area in Fig. 13b into four subareas: *top-left*, *top-right*, *bottom-left* and *bottom-right*, (i.e., $\frac{TL|TR}{BL|BR}$), ASC-SPP could sometimes have a good start (for all ASC-IN, ASC-SPP and RSC-NR, in the first round, the initial regions to query are random) finding the first few '1's in the *top-left* subarea quickly but end up exploring almost the entire sparse subarea in the *top-left* or *bottom-left* to find the final '1'. But with higher k (e.g., with $k = 40$), as the search continues, ASC-SPP starts to find areas of higher

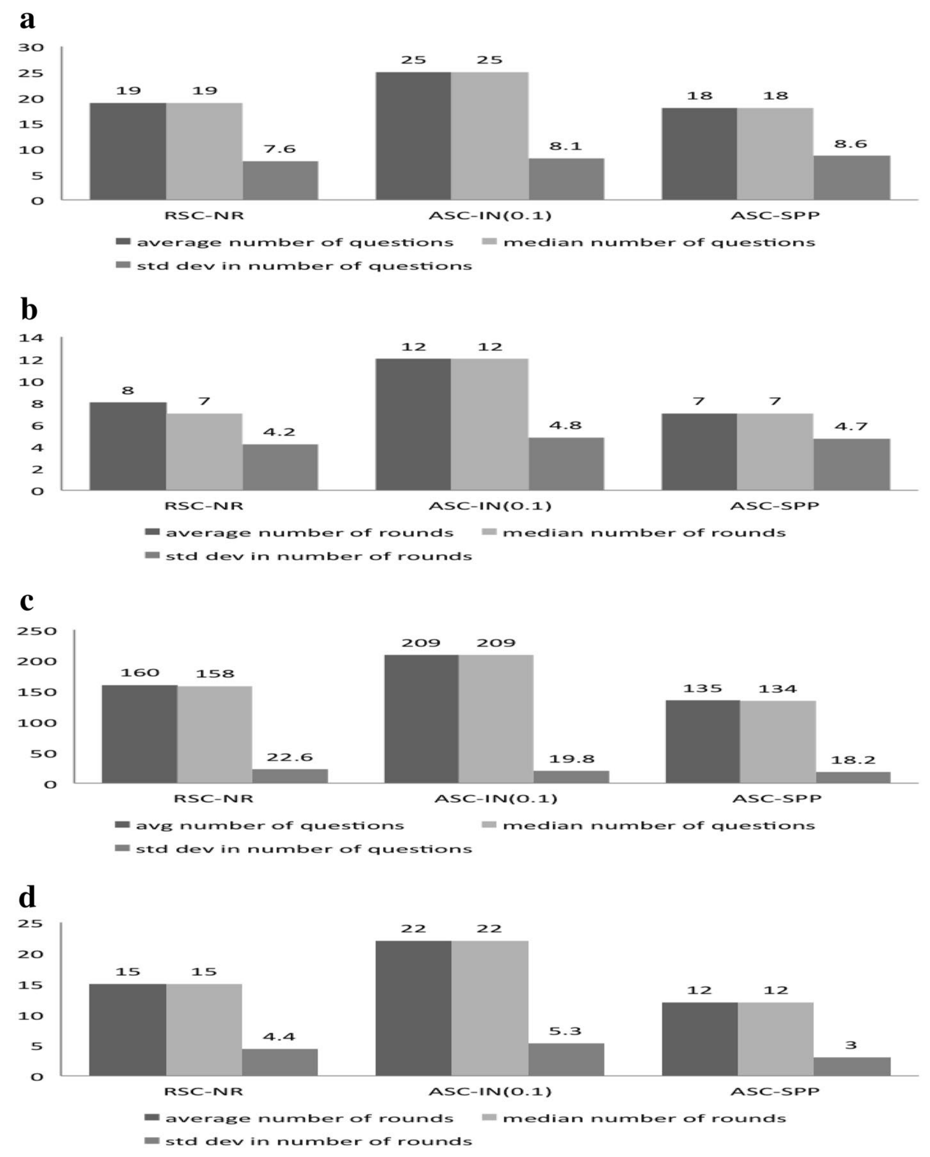
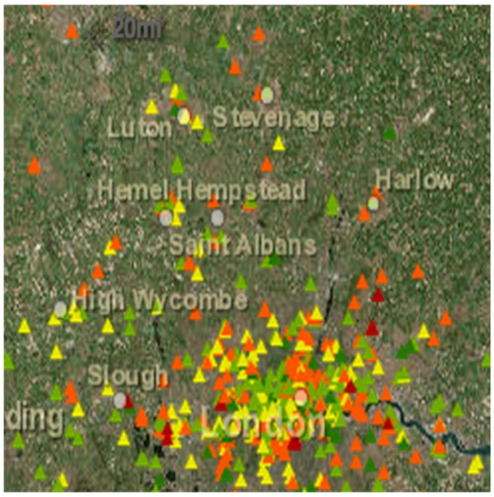


Fig. 12 Results averaged over 1000 runs for two different values of k for the randomly generated scenario, comparing RSC-NR, ASC-IN ($\delta = 0.1$) and ASC-SPP. **a** Average, median and standard deviation for number of questions with $k = 5$. **b** Average, median and standard deviation for number of rounds with $k = 5$. **c** Average, median and standard deviation for number of questions with $k = 40$. **d** Average, median and standard deviation for number of rounds with $k = 40$.

density better than ASC-IN and RSC-NR and then, on average, outperforms ASC-IN and RSC-NR.

To deal with this, for cases of low k, one can employ history to focus the initial search, i.e., assuming historical information tells us that it would be better to start querying regions (randomly selected) from the bottom-right subarea than randomly selecting regions from anywhere to query in the first round, the results are substantially different; in Fig. 14e and f, the advantage of using history at the start of ASC-SPP, called ASC-SPP-His, in exploring regions of higher density (even if not contiguous) is demonstrated.

a



b

```
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000010000000000000000000000000100000000000000000000000000000000
000000000010000000000000000000000000100000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000001100000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000010000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000001000000000000000000000000000000010000000000000000000000000000000
000001000000000000000000000000000000010000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000010000000000000000
000000000000000000000000100000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000001000000000000000000
000000000000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000010000000000000000000000000000000000000000000000000000
000000000000000010000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000001000000000000000000000000000000000000000000
000000000000011000000000000000000000000000000000000001000010000000000
000000000000000000000000000000000000000000000000001000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000010000000000000000000
000000000000000000000000000100000000110000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000011000000
000000000000000000000000000000000000000000100000000000000000100000000
000000000000000000000000000000000000000000010100010000000000100000000
000000000000000000000000000000001000000000000000000000000000000000000
000000000000000000000000110000000000000011000000000000000000000000000
000000000000000000000000000000000000010000111110000000000000000000000
000000000000000000000000000000111000000000111110000000000000000000000
000000000110000000000000000000110000000000000000000001000000000000000
000000000010000000000000000000010110000000000000000011001000000000000
000000000000000000000000100000000000000000001000111000000000000000000
000000000000000000011000000000000000000000000000000001101100000000000
000000000000000000000000001000000000000001000000000000000000010000000
000000000000000000000000000000000000000000000000010001000000000000000
000000000000000000000000000001111000001100000000001000000000010000000
000000000000000000000000000000011000000110000011010000000110000000000
000000000000000000000000000000110000000000000000011001000000000000000
000000000000000000000000000000100000000000000000000001000000000000000
000000000000000000000000000000000000000011000100000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
```

Fig. 13 Noise map and its abstract version. **a** Noise map from http://urbanobservatory.org/compare/index.html (NoiseWatch) around London, *red* and *orange triangles* indicate high noise levels. **b** Abstracted discretized view of noise map with $\alpha = 0.030657748$ with 69×52 regions;1s representing high noise regions

Related work

Algorithms for crowdsourcing has been a relatively new endeavour but currently a very active area of work.[3] The past half decade has seen much development in the area, e.g., the work in [17] in crowdsourcing algorithms, the work in [19] on crowdsourcing for discovery, the work in [20, 13, 21] on achieving real-time results in crowdsourcing, and

[3] For example, see http://crowdwisdom.cc/nips2013/ and http://www.humancomputation.com/2014/.

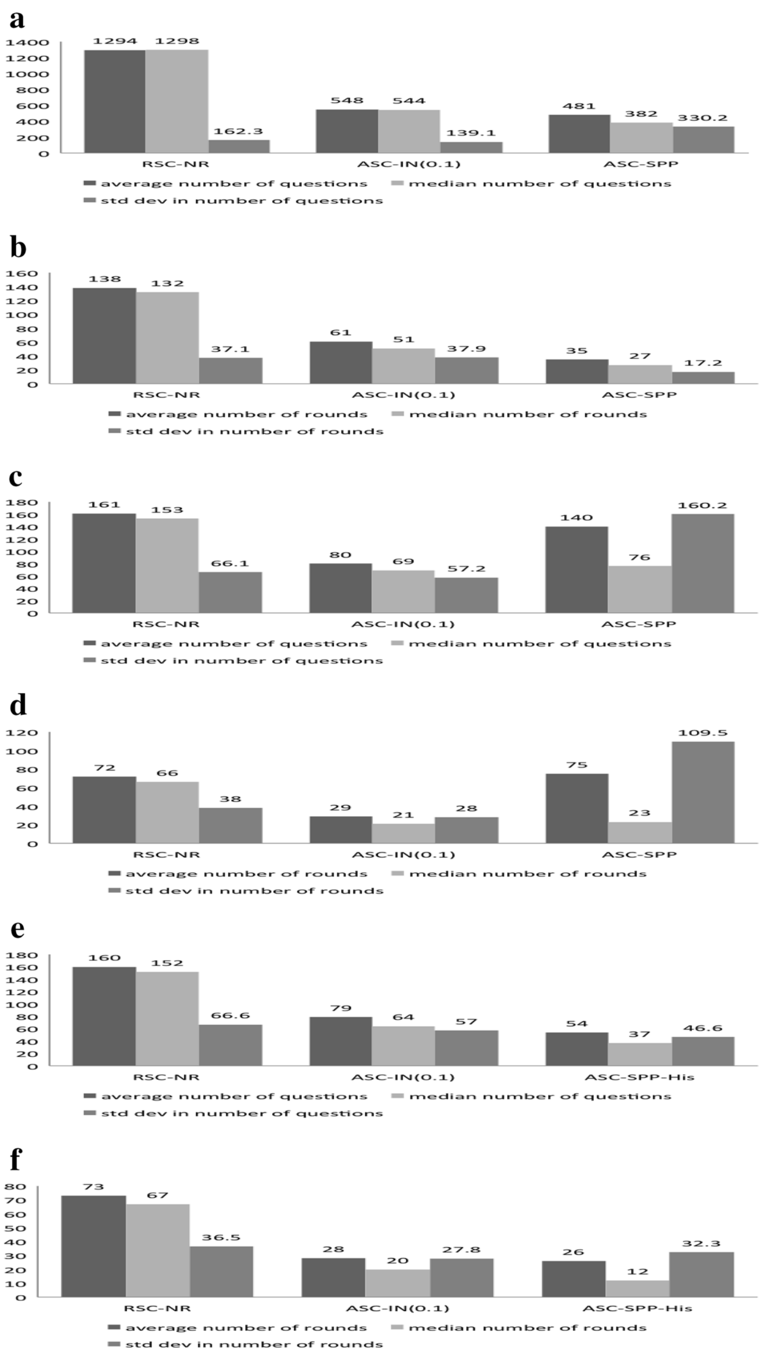


Fig. 14 Results averaged over 1000 runs comparing RSC-NR, ASC-IN ($\delta = 0.1$), and ASC-SPP (and ASC-SPP-His) for different values of k for the noise scenario. **a** Average, median and standard deviation for number of questions with $k = 40$ comparing RSC-NR, ASC-IN ($\delta = 0.1$), and ASC-SPP. **b** Average, median and standard deviation for number of rounds with $k = 40$ comparing RSC-NR, ASC-IN ($\delta = 0.1$), and ASC-SPP. **c** Average, median and standard deviation for number of questions with $k = 5$ comparing RSC-NR, ASC-IN ($\delta = 0.1$), and ASC-SPP. **d** Average, median and standard deviation for number of rounds with $k = 5$ comparing RSC-NR, ASC-IN ($\delta = 0.1$), and ASC-SPP. **e** Average, median and standard deviation for number of questions with $k = 5$ comparing RSC-NR, ASC-IN ($\delta = 0.1$), and ASC-SPP-His. **f** Average, median and standard deviation for number of rounds with $k = 5$ comparing RSC-NR, ASC-IN ($\delta = 0.1$), and ASC-SPP-His

the work in [22] on crowd-selection for microtasks. There is already a range of commercial frameworks such as Amazon Mechanical Turk,[4] CrowdFlower,[5] and CrowdCloud.[6] Focus has been myriad, from user interaction aspects to algorithmic and framework aspects.

There is also the emerging trend of *geo-crowdsourcing*, where ever-expanding groups of users collaboratively and often voluntarily (or paid to) contribute different types of spatial or geographic information [9, 23, 24]. Also, emerging are work on location/context-based crowdsourcing where location is key in distributing jobs to workers [5, 25, 11, 26] with applications in transportation and so on, and work in [27] where different strategies are explored to answer location-based crowdsourcing of queries. However, we believe our work is original in approaching the spatial finding problem. Microsoft Research has an interesting set of *spatial crowdsourcing* projects, focusing on mechanisms to encourage ordinary people to perform tasks at specific locations.[7] The gMission system [28] is a platform to support spatial crowdsourcing and provides a range of features including matching potential workers with tasks, but our work focuses on which areas to query rather than workers.

As mentioned earlier, this work is partly motivated by increasing work on mobile crowdsourcing mentioned earlier and mobile crowdsensing [29–32], where mobile context provides valuable situational knowledge that can be crowdsourced. We did not deal with incentives in this paper but assumed that we will get response about a region whenever a query is asked but how to use incentives to get appropriate responses is also an active area of research.

In [25], a Gaussian process model was used to predict future traffic saturation at junctions with sensors with generalisation to junctions without sensors. A Gaussian approach might be used in modelling the distribution of positive regions, but with too small a k and too small a proportion of observed regions, it is uncertain if meaningful predictions can be made with this approach but it could be investigated. Spatial sampling techniques such as spatial simulated annealing using prior information [33] can be employed in place of our random sampling approach and compared to our heuristics.

The work in [34] reviews mobile crowdsourcing pointing out further challenges such as incentive mechanisms, reputation management, and task allocation.

Conclusion

This paper proposed and investigated finding regions of interest from a set of regions of an area using iterative crowdsourcing processes controlled by the principle of a query-feedback loop interleaved with query adjustment based on responses and heuristics. We have described three simple, though effective, heuristics for reducing the costs (the number of questions required) and increasing the efficiency (or reducing the number of rounds required) in using crowdsourcing for finding regions of interests:

- using a proportionate number of redundant questions in each round in the expectation of failure, as already pointed out in earlier work by [17],

[4] http://aws.amazon.com/mturk/.

[5] http://www.crowdflower.com/.

[6] http://www.crowdcloud.com/.

[7] http://research.microsoft.com/en-us/projects/spatialcrowdsourcing/,

- immediate neighbourhood associations in the case where regions of interest are clustered contiguously, and
- spatial point processes for approximating distribution of positive regions, working even without contiguous positive regions, with approximations improved on each round, with the use of historical information to guide starting queries in cases of low k.

We demonstrated, via a range of maps (synthetic and real-world based), that our heuristics lead to improved performance over randomly choosing regions to ask about. While we use stylised maps based on real-world distributions of parking, crowd, bandwidth coverage, and noise, our research suggests that finding in the city is not as difficult as it can be for phenomena that exhibit some degree of clustering. While our focus has been on spatial problems, we also note that the heuristics are generalisable to non-spatial problems as long as meaningful associations can be defined among items.

Future work involves exploring a combination of the heuristics in real deployments as well as other application-specific spatial and geographically based heuristics, and heuristics that exploit historical information—so, for example, we can include historical information in computing probabilities, i.e. for a region r, we calculate $Pr(\, F(r) = TRUE \mid obs(N(r)) \wedge history(r))$. There are also many applications to explore, from finding vacant/quiet coffee-shops to finding strategic points of interest in emergency situations. Dealing with uncertainties and unresponsive crowds are further issues to consider, e.g., taking into account regions with low density of people. We did not deal with the problem of incentives and strategic sampling will need to be considered in the future.

Appendix

Proof of Lemmas and Theorems

Lemma 1 *The function T is monotonically increasing, i.e. for any m, $T(n) \geq T(m)$ for all $n \geq m$.*

Proof Let $S(n)$ denote the statement: $T(n) \geq T(m)$ for all $0 \leq m \leq n$. Suppose $n = 1$, then, $T(1) \geq T(0)$, i.e., $S(1)$ is true. For a given p, assume that $S(2), \ldots, S(p)$ are all true and we will prove $S(p+1)$. With $S(p)$ being true, for any $m \leq p$, $T(p) \geq T(m)$. So, we just need to show that $T(p+1) \geq T(p)$.

Note that $p \geq 2$. Then, since $((p+1) - \lceil (p+1) \cdot \sigma \rceil) \leq p$, and so, $S((p+1) - \lceil (p+1) \cdot \sigma \rceil)$ is also true, as assumed, and we have $T((p+1) - \lceil (p+1) \cdot \sigma \rceil) \geq T(m)$, for any $m \leq ((p+1) - \lceil (p+1) \cdot \sigma \rceil)$. Now,

$$\begin{aligned} T(p+1) &= T((p+1) - \lceil (p+1) \cdot \sigma \rceil) + 1 \\ &\geq T(p - \lceil p \cdot \sigma \rceil) + 1 \\ &\text{(by induction as } (p - \lceil p \cdot \sigma \rceil) \leq ((p+1) - \lceil (p+1) \cdot \sigma \rceil)) \\ &= T(p) (\text{ by definition of } T) \end{aligned}$$

□

Lemma 2 *Given γ, the function Q'_γ is monotonically increasing, i.e. for any m, $Q'_\gamma(n) \geq Q'_\gamma(m)$ for all $n \geq m$.*

Proof Let $S'(n)$ denote the statement: $Q'_\gamma(n) \geq Q'_\gamma(m)$ for all $0 \leq m \leq n$. Suppose $n = 1$, then, $Q'_\gamma(1) \geq Q'_\gamma(0)$, i.e., $S'(1)$ is true. For a given p, assume that $S'(2), \ldots, S'(p)$ are all true and we will prove $S'(p+1)$. With $S'(p)$ being true, for any $m \leq p$, $Q'_\gamma(p) \geq Q'_\gamma(m)$. So, we just need to show that $Q'_\gamma(p+1) \geq Q'_\gamma(p)$.

$$\begin{aligned}
Q'_\gamma(p+1) &= \lceil \gamma \cdot (p+1) \rceil + Q'_\gamma((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil) \\
&\geq \lceil \gamma \cdot (p+1) \rceil + Q'_\gamma(p - \lceil \lceil \gamma \cdot p \rceil \cdot \sigma \rceil) \\
&\quad \text{(by induction as } ((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil) \geq (p - \lceil \lceil \gamma \cdot p \rceil \cdot \sigma \rceil) \\
&\qquad \text{and } p \geq ((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil)) \\
&\geq \lceil \gamma \cdot p \rceil + Q'_\gamma(p - \lceil \lceil \gamma \cdot p \rceil \cdot \sigma \rceil) \\
&= Q'_\gamma(p) \\
&\quad \text{(by definition of } Q'_\gamma)
\end{aligned}$$

□

Theorem 1 *For any $\frac{1}{\sigma} \geq \gamma \geq 1$ and a given non-negative integer n, we have $T'_\gamma(n) \leq T(n)$.*

Proof Firstly, $T'_\gamma(0) = T(0)$. Now, for all integers $1 \leq m \leq p$, we assume that $T'_\gamma(m) \leq T(m)$, and will prove that $T'_\gamma(p+1) \leq T(p+1)$. We have:

$$\begin{aligned}
T'_\gamma(p+1) &= T'_\gamma((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil) + 1 \\
&\leq T((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil) + 1 \\
&\text{(by induction since } ((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil) \leq p) \\
&\leq T((p+1) - \lceil (p+1) \cdot \sigma \rceil) + 1 \\
&\text{(by Lemma 1 since} \\
&((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil) \leq ((p+1) - \lceil (p+1) \cdot \sigma \rceil)) \\
&= T(p+1) \\
&\text{(by definition of } T)
\end{aligned}$$

□

Theorem 2 *For any $\frac{1}{\sigma} \geq \gamma \geq 1$ and a given non-negative integer n,* we have $Q'_\gamma(n) \leq \lceil \gamma \rceil \cdot Q(n)$, i.e. $Cost'_\gamma(n) \leq \lceil \gamma \rceil \cdot Cost(n)$.

Proof Firstly, $Q'_\gamma(0) \leq \lceil \gamma \rceil \cdot Q(0)$. Now, assume that for any $m \leq p$, we have $Q'_\gamma(m) \leq \lceil \gamma \rceil \cdot Q(m)$. We will show that $Q'_\gamma(p+1) \leq \lceil \gamma \rceil \cdot Q(p+1)$.

$$\begin{aligned}
Q'_\gamma(p+1) &= \lceil \gamma \cdot (p+1) \rceil + Q'_\gamma((p+1) - \lceil \lceil \gamma \cdot (p+1) \rceil \cdot \sigma \rceil) \\
&\le \lceil \gamma \cdot (p+1) \rceil + Q'_\gamma((p+1) - \lceil (p+1) \cdot \sigma \rceil) \\
&(\text{ by Lemma 2 }) \\
&= \lceil \gamma \cdot (p+1) + Q'_\gamma((p+1) - \lceil (p+1) \cdot \sigma \rceil) \rceil \\
&\le \lceil \gamma \cdot (p+1) + \lceil \gamma \rceil \cdot Q((p+1) - \lceil (p+1) \cdot \sigma \rceil) \rceil \\
&(\text{by induction since } ((p+1) - \lceil (p+1) \cdot \sigma \rceil) \le p) \\
&\le \lceil \lceil \gamma \rceil \cdot (p+1) + \lceil \gamma \rceil \cdot Q((p+1) - \lceil (p+1) \cdot \sigma \rceil) \rceil \\
&= \lceil \lceil \gamma \rceil \cdot ((p+1) + Q((p+1) - \lceil (p+1) \cdot \sigma \rceil)) \rceil \\
&= \lceil \lceil \gamma \rceil \cdot Q(p+1) \rceil (\text{by definition of } Q) \\
&= \lceil \gamma \rceil \cdot Q(p+1)
\end{aligned}$$

□

Received: 20 July 2015 Accepted: 24 November 2015
Published online: 17 April 2016

References

1. Brabham DC (2013) Crowdsourcing. The MIT Press, Cambridge
2. Law E, Ahn LV (2011) Human computation. Synthesis lectures on artificial intelligence and machine learning. Morgan & Claypool Publishers, San Rafael
3. Michelucci P (2013) Handbook of Human Computation. Springer Publishing Company, Incorporated, New York
4. Franklin MJ, Kossmann D, Kraska T, Ramesh S, Xin R (2011) CrowdDB: answering queries with crowdsourcing. In: Proc. of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, ACM, New York, pp 61–72
5. Alt F, Shirazi AS, Schmidt A, Kramer U, Nawaz Z (2010) Location-based crowdsourcing: extending crowdsourcing to the real world. In Proceedings of the 6th Nordic Conference on Human-Computer Interaction: extending boundaries, NordiCHI '10, New York, ACM, pp 13–22
6. Georgios G, Konstantinidis A, Christos L, Zeinalipour-Yazti D (2012) Crowdsourcing with smartphones. IEEE Internet Computing 16(5):36–44
7. Gupta A, Thies W, Cutrell E, Balakrishnan R (2012) mClerk: enabling mobile crowdsourcing in developing regions. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '12, ACM, New York, pp 1843–1852
8. Charoy F, Benouare K, Valliyur-Ramalingam R (2013) Answering complex location-based queries with crowdsourcing. In: Proc. of the 9th IEEE Int. Conf. on Collaborative Computing: Netw., App. and Worksharing, IEEE Computer Society
9. Kazemi L, Shahabi C (2012) Geocrowd: enabling query answering with spatial crowdsourcing. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems, SIGSPATIAL '12, ACM, New York, pp 189–198
10. Della Mea V, Maddalena E, Mizzaro S (2013) Crowdsourcing to mobile users: a study of the role of platforms and tasks. In DBCrowd, pp 14–19
11. Tamilin A, Carreras L, Ssebaggala E, Opira A, Conci N (2012) Context-aware mobile crowdsourcing. In: Proceedings of the 2012 ACM Conference on Ubiquitous Computing, UbiComp '12, ACM, New York, pp 717–720
12. Yan T, Marzilli M, Holmes R, Ganesan D, Corner M (2009) mCrowd: a Platform for Mobile Crowdsourcing. In Proc. of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys '09, ACM, New York, pp 347–348
13. Bernstein MS, Brandt J, Miller RC, Karger DR (2011) Crowds in two seconds: enabling realtime crowd-powered interfaces. In UIST, pp 33–42
14. Guo S, Parameswaran A, Garcia-Molina H (2012) So who won?: dynamic max discovery with the crowd. In: SIGMOD Conference, pp 385–396
15. Parameswaran AG, Garcia-Molina H, Park H, Polyzotis N, Ramesh A, Widom J (2012) Crowdscreen: algorithms for filtering data with humans. In SIGMOD Conference, pp 361–372
16. Das Sarma A, Parameswaran A, Garcia-Molina H, Halevy A (2014) Crowd-powered find algorithms. In: Data engineering (ICDE), 2014 IEEE 30th International Conference on, pp 964–975
17. Parameswaran AG (2013) Human-powered data management. PhD thesis, Department of Computer Science, Stanford University, Stanford
18. Cressie N, Wikle CK (2011) Statistics for Spatio-Temporal Data., Wiley Series in Probability and StatisticsWiley, Hoboken
19. Faridani S (2012) Models and Algorithms for crowdsourcing discovery. PhD thesis, Berkeley, University of California
20. Bernstein MS (2013) Crowd-powered systems. KI 27(1):69–73

21. Bigham JP, Jayant C, Ji H, Little G, Miller A, Miller RC, Miller R, Tatarowicz A, White B, White S, Yeh T (2010) Vizwiz: nearly real-time answers to visual questions. In Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology, UIST '10, pp 333–342, New York, ACM
22. Abraham I, Alonso O, Kandylas V, Slivkins A (2013). Adaptive crowdsourcing algorithms for the bandit survey problem. CoRR, abs/1302.3268
23. Shahabi C (2013) Towards a generic framework for trustworthy spatial crowdsourcing. In: Proceedings of the 12th International ACM Workshop on Data Engineering for Wireless and Mobile Acess, MobiDE '13, ACM, New York, pp 1–4
24. Sui D, Elwood S, Goodchild M (2013) Crowdsourcing geographic knowledge: volunteered geographic information (VGI) in theory and practice, Springer
25. Schnitzler F, Liebig T, Mannor S, Morik K (2014) Combining a gauss-markov model and gaussian process for traffic prediction in dublin city center. In EDBT/ICDT Workshops, pp 373–374
26. Wu D, Zhang Y, Bao L, Regan AC (2013) Location-based crowdsourcing for vehicular communication in hybrid networks. IEEE Trans Intell Transp Sys 14(2):837–846
27. Benouaret K, Valliyur-Ramalingam R, Charoy F (2013) Answering complex location-based queries with crowdsourcing. In CollaborateCom, pp 438–447
28. Chen Z, Fu R, Zhao Z, Liu Z, Xia L, Chen L, Cheng P, Cao CC, Tong Y, Zhang CJ (2014) gmission: a general spatial crowdsourcing platform. PVLDB 7(13):1629–1632
29. Ganti RK, Ye F, Lei H (2011) Mobile crowdsensing: current state and future challenges. IEEE Commun Mag 49(11):32–39
30. Hu X, Chu T, Chan H, Leung V (2013) Vita: a crowdsensing-oriented mobile cyber-physical system. IEEE Trans Emerg Top Comput 1(1):148–165
31. Lane ND, Chon Y, Zhou L, Zhang Y, Li F, Kim D, Ding G, Zhao F, Cha H. Piggyback crowdsensing (pcs): energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities. In: Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, SenSys '13, ACM, New York, pp 1–7
32. Sherchan W, Jayaraman PP, Krishnaswamy S, Zaslavsky A, Loke S, Sinha A (2012) Using on-the-move mining for mobile crowdsensing. In: Proceedings of the 2012 IEEE 13th International Conference on Mobile Data Management (mdm 2012), MDM '12, IEEE Computer Society, Washington, pp 115–124
33. Van Groenigen JW (1997) Spatial simulated annealing for optimizing sampling. In: Soares AO, Gmez-Hernandez J, Froidevaux R (eds) geoENV I - Geostatistics for Environmental Applications, vol 9., Quantitative Geology and GeostatisticsSpringer, Netherlands, pp 351–361
34. Ren J, Zhang Y, Zhang K, Shen X (2015) Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. Commun Mag IEEE 53(3):98–105