# Simplification and Refinement for Speedy Spatio-temporal Hot Spot Detection Using Spark (GIS Cup) *

Shangfu Peng Hong Wei Hao Li Hanan Samet

Center for Automation Research
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742 USA
{shangfu, hyw, haoli, hjs}@cs.umd.edu

## ABSTRACT

This paper describes a spatio-temporal hot spot identification program submitted to the ACM SIGSPATIAL Cup 2016. The advent of large-scale spatio-temporal data (e.g., vehicle tracking data), together with the availability of in-memory distributed computing framework (i.e., Spark), provides an opportunity to quickly identify unusual patterns in a statistically manner, also called hot spots. We propose an efficient approach on Spark to select top-$k$ hot spots using the Getis-Ord statistic. Our method consists of three phases: *simplification*, *detection*, and *refinement*. The simplification phase picks a reasonable number, e.g., $K$, of most important cells; then the detection phase computes the Getis-Ord statistic for the $K$ cells; finally, the refinement phase puts the top $M$ candidates and their neighbors back in the original set of cells to refine the results.

## Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*; H.2.4 [**Database Management**]: Systems—*Query processing*

## Keywords

spatio-temporal, hot spot, distributed processing, Spark

## 1. INTRODUCTION

The problem of spatio-temporal hot spot identification is to locate the top $k$, e.g., 50 in this contest, cells that have the greatest Getis-Ord scores [2] in a tessellated three dimensional space formed by *longitude*, *latitude* and *time*. The Getis-Ord score of each cell is computed by its value and its neighbors' values. A brute-force method on a single processor is to iterate every cell to compute its Getis-Ord score definition. Obviously, the computing process for each cell is independent once the values of its neighbors are at hand. Therefore, identification of hot spots should be sped up by a distributed framework very well.

Spark [6] is a popular open-source distributed framework. In contrast to Hadoop's two-stage disk-based MapReduce framework, Spark's in-memory primitives provide performance up to 100 times faster for certain applications. As the 2015 New York City Taxi and Limousine Commission yellow cab source data [1] is only 24GB in plain text, we can cache all cells with records in memory after scanning them once from HDFS. All subsequent computing takes place in the distributed memory to avoid extra I/O cost.

Our previous experience on Spark [3], although in the context of use in road network problems (e.g., [4, 5]), indicates that the network communication cost during shuffle would greatly exceed the computation or I/O cost, becoming a significant bottleneck if the distributed algorithm is not well designed. To avoid unnecessary shuffle, the key problem is how to load and partition the data to slave nodes more independently. In distributed computing, it is common for a slave machine $A$ to need the data on another slave machine $B$ to finish its own computing. A multiple shuffle strategy or a partition strategy with data redundancy might solve this problem, but both of them result in an increase in network communication cost.

In this paper, we implemented and evaluated three solutions for the contest in Section 4, and our submission is the proposed simplification-refinement approach, termed SR-$K$-$M$, where $K = 2,000,000$ and $M = 500$. The simplification phase is proposed to reduce the runtime, and the refinement phase is proposed to improve the accuracy. As a result, SR-$K$-$M$ can solve any granularity setting within 50 seconds even when the cells with valid records are very sparse. Although there is a theoretical chance that our solution would return a wrong ranking, thanks to the refinement phase, we haven't found a granularity setting that makes our solution failed. In addition, both $K$ and $M$ should be adjusted based on $k$ and the dataset. As $k$ is a fixed number, 50, in this contest, setting $K = 2,000,000$ and $M = 500$ is fair enough.

## 2. PROBLEM FORMULATION

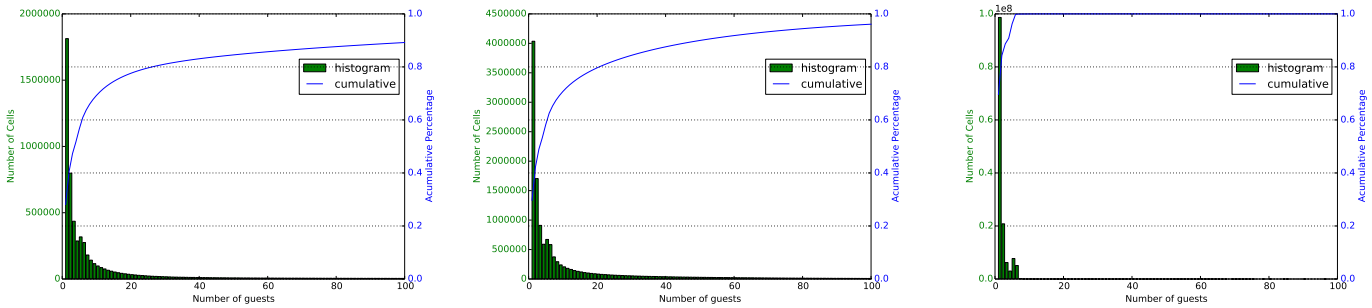
In a uniformly tessellated space, the objective of identi-

*This work was supported in part by the AWS research grant and the NSF under Grants IIS-12-19023 and IIS-13-20791.



*SIGSPATIAL'16 October 31 - November 03, 2016, Burlingame, CA, USA*

DOI: http://dx.doi.org/10.1145/2996913.3004064




(a) $r_s = 0.001, r_t = 1$ (b) $r_s = 0.001, r_t = 0.25$ (c) $r_s = 0.00001, r_t = 0.25$

**Figure 1: The histogram and cumulative histogram of all $x_i$ in 2015. The data follows the long-tail distribution. In order to keep the top two million cells, we prune cells with $x \leq 10$ in (a) to cut off 70% data, prune cells with $x \leq 30$ in (b) to cut off 85% data, and prune cells with $x \leq 5$ in (c) to cut off 98% data.**

fying hot spots is to find the top-$k$ cells for a given metric. In this contest, the space, which comprises of *longitude*, *latitude* and *time*, is uniformly divided into equal-sized cube cells. And the metric used to sort and select the top-$k$ cells is the Getis-Ord statistic [2]. Formally, given a cube cell $c_i$, its Getis-Ord value $G_i^*$ is defined as follows:

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j}x_j - \bar{X}\sum_{j=1}^{n} w_{i,j}}{S\sqrt{\frac{n\sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2}{n-1}}} \quad (1)$$

where $n$ refers to the total number of cube cells, and $x_j$ is the attribute value of cell $c_j$; $w_{i,j}$ is the spatial weight between cell $c_i$ and cell $c_j$, which is defined as follows:

$$w_{i,j} = \begin{cases} 1, & \text{if } c_i \text{ and } c_j \text{ are neighbors} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Here we define two cells as neighbors if they share at least a vertex, an edge or a face, or they are the same cube cell. $\bar{X}$ and $S$ are defined as

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n}, \qquad S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2} \quad (3)$$

To simplify the problem, first, according to the definition of $w_{i,j}$, it is easy to prove that: **top-$k$ selection by $G_i^*$ is equivalent to top-$k$ selection by $X_i = \sum_{j=1}^{n} w_{i,j}x_j$, i.e., the sum of a cell's neighbors' attribute values** $x_j$. Without causing confusion, we use the term "top-$k$" to denote "top-$k$ (cells) by the metric of $G_i^*$, or $X_i$" later.

Second, although the cell size and time step size, termed granularity $(r_s, r_t)$[1], are arbitrary in this contest, we claim that **only the fine granularity matters in choosing what the distributed solution**, where a fine granularity means that the cells with records are very sparse in the cube space. It is because that any distributed solution, including the brute-force one, on a coarse granularity, e.g., 0.1 and seven days, would take negligible time compared with the runtime of the inevitable preparation phase, which needs $18 \sim 20$ seconds as illustrated in Section 4. To summarize, the problem is how to find the top-$k$ cells by $X_i$ in a fine granularity.

[1]The unit of $r_s$, by default, is *degree* in longitude and latitude, and the unit of $r_t$ is 1 *day* in time.

# 3. APPROACH

As the cells with values are sparse in a fine granularity cube, a baseline solution, termed BASIC, copies each valued cell 27 times, sends one copy to each neighbor cell in the mapper process, and computes the $X_i$ for cells in the reducer process. BASIC only performs very well when the number of cells with values is less than two million, as it increases the workload of the shuffle process by a factor of 27.

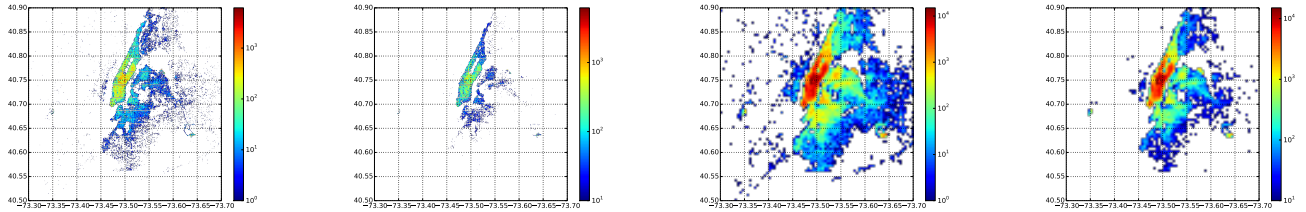
To reduce the shuffle workload, our SR-$K$-$M$ solution (see Algorithm 1) inserts the simplification phase before BASIC, and appends the refinement phase after BASIC.

## 3.1 Simplification

Simplification selects only a fixed number $K$ of cells with the largest $x_i$. These $K$ cells, denoted by $\mathcal{C}^K$, are the candidates from which we identify the top-$k$ cells as the program's output. Selecting $\mathcal{C}^K$ in an un-ordered list is implemented by finding the minimum value $x_l$ such that $|c_i : x_i \geq x_l| \leq K$. This step is implemented in a *binary search* strategy with $O(\log n)$ scans on Spark, which finishes in instant time.

Pruning the cells to $\mathcal{C}^K$ is inspired by our observation and analysis on the given taxi data: i) the cells with small $x_i$ comprise the majority but they are less likely to be in top-$k$, see Figure 1; ii) from both spatial and temporal perspectives, the input data presents a quasi-continuous pattern, i.e. a hot spot's spatio-temporal neighbors also tend to be "hot", and it is unlikely for a hot spot to locate at a place where its neighbors are of small values, see Figure 2, 3. These observations give us the idea that removing insignificant cells might not throw away any cells in the top-$k$. And in our experiments, we found that with a reasonable value of $K$, the *simplification* step ensures all the top-$k$ cells fall into $\mathcal{C}^K$, while at the same time, significantly speeds up our program. For example, in Figure 1(a), a setting of $x_l = 10$ with corresponding $K = 2,000,000$ filters away 70% of all cells while still succeeding in preserving the correct top-50 cells.

Furthermore, through simplification, we avoid the necessity of computing $G_i^*$ (or equivalently $X_i$) for every cell in order to perform the top-$k$ selection. This gives us a significant speedup, because computing $G_i^*$ or $X_i$ requires a cell to collect all its neighbors' attribute values $x_j$. This, in a distributed system, brings about costly messaging operations between machine nodes. Therefore, reducing the number of cells significantly reduces the communication overhead and



(a) $r_s = 0.001$ (b) $x_i \geq 10$, $r_s = 0.001$ (c) $r_s = 0.005$ (d) $x_i \geq 10$, $r_s = 0.005$

**Figure 2: Spatial consistency. The heat-map of cells on Jan.1, 2015 ($r_t = 1$) with cell size $r_s = 0.001$ and $r_s = 0.005$. Removing cells with small $x_i$ does not influence the hot spot area, especially for the coarser one.**

computation cost, thus speeding up the program.

In the implementation, a reasonable value of $K$ is of great importance to the program: "too small" might cause some of the top-$k$ to fall outside $\mathcal{C}^K$; however, "too large" will not yield evident speedup. For the data provided in the 2016 ACM SIGSPATIAL CUP, we found $K = 2,000,000$, a threshold that gives us significant speedup, and at the same ensures the accuracy of our program on the data.

## 3.2 Refinement

Note that when we compute $X_i$ of a given cell $c_i$ in the set $\mathcal{C}^K$, its spatial weight function $w_{i,j}$ updates to:

$$w'_{i,j} = \begin{cases} 1, & \text{if } c_i, c_j \text{ are neighbors, and } c_j \in \mathcal{C}^K \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Now using this updated spatial weight function, we find that the greatest $k$ cells are not necessarily identical to our initial top-$k$ cells. For example, a cell that is supposed to rank between 45 to 50 now might fall outside 50 because some of its neighbors were thrown away in the step of simplification. We therefore propose the *refinement* step to remedy this problem.

Refinement is to i) choose the greatest $M$ cells, denoted by $\mathcal{C}^M$, from pre-selected cells $\mathcal{C}^K$ under the metric of $X_i$ by using the updated spatial weight function $w'_{i,j}$, ii) for each cell $c_i \in \mathcal{C}^M \cup neighbor(\mathcal{C}^M)$, re-compute its $X_i$ (equivalent to $G_i^*$) using the original $w_{i,j}$ and select the greatest $k$ cells as the top-$k$ in the results. $neighbor(\mathcal{C}^M)$ is the set containing all neighbors of the cells in $\mathcal{C}^M$.

In the implementation, we set $M = 10 \times k \ll K$, which makes re-computing $X_i$ for cells in $\mathcal{C}^M \cup neighbor(\mathcal{C}^M)$ very efficient. And also in our experiments, we found that, even if a cell's rank $m'$ falls behind position $k$ by exploiting the updated $w'_{i,j}$, $m'$ is usually only slightly larger than $k$. We therefore set $M = 10 \times k$, which is large enough to satisfy top-$k \in \mathcal{C}^M \cup neighbor(\mathcal{C}^M)$.

# 4. EVALUATION

In this section, we evaluated our following solutions.

1. Our simplification-refinement solution, SR-$K$-$M$, where $K$ is the number of cells that we kept in the simplification phase, and $M$ is the number of results that we need to refine them. Both $K$ and $M$ should be set based on how many hot spots are sought and the dataset. We set $K = 2,000,000$ and $M = 500$ in our submission and evaluation.
2. The baseline solution, BASIC, which copies each cell 27 times, and sends one copy to each neighbor cell by one Map-Reduce task.
3. An coordinate-shift solution, termed SHIFT, which does 27 Map-Reduce tasks, and each Map-Reduce task first does one coordinate shift $(i, j, k) \in \{-1, 0, 1\}^3$ on all cells, then congregates every $3 \times 3$ cube (with 27 cells) to one cell, e.g., the center cell of the cube, and finally computes the top 50 cells in this shift.

**Algorithm 1:** Hot Spot Identification on Spark

**Input**: The number of hot spots to output – $k$; the spatio-temporal data – $\mathbf{z}$; spatio-temporal resolution – $(r_s, r_t)$

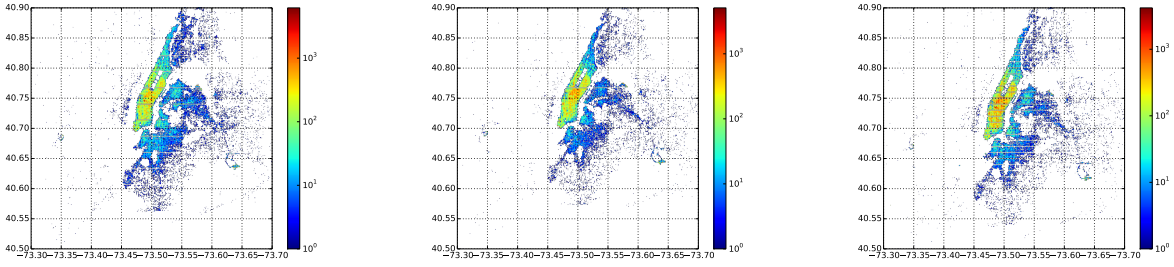
**Output**: top-$k$ hot spots

1: data pre-processing:
   a. tessellate space uniformly with $(r_s, r_t)$ into equal-sized cube cells;
   b. aggregate $\mathbf{z}$ into cube cells according to their *longitude*, *latitude* and *time*;
   c. for each cell, calculate its attribute value $x_i$ by summing the spatio-temporal data in it.
2: $\mathcal{C}^K \leftarrow$ top $K$ cells with largest $x_i$
3: perform a Map-Reduce procedure on $\mathcal{C}^K$ to calculate $X_i$ for each cell, using updated spatial weight $w'_{i,j}$:
   a. Map: copy the current cell value 27 times but with neighbors' coordinates;
   b. Reduce: reduce by using cell's coordinates and sum the attribute value to get $V_i$.
4: $\mathcal{C}^M \leftarrow$ top $M$ cells from $\mathcal{C}^K$ with largest $X_i$
5: perform a Map-Reduce on $\mathcal{C}^M \cup neighbor(\mathcal{C}^M)$ to re-calculate $X_i$ for each cell, but using $w_{i,j}$ with its original cells.
6: top-$k$ hot spots $\leftarrow$ top-$k$ cells from $\mathcal{C}^M$ with largest $V_i$.

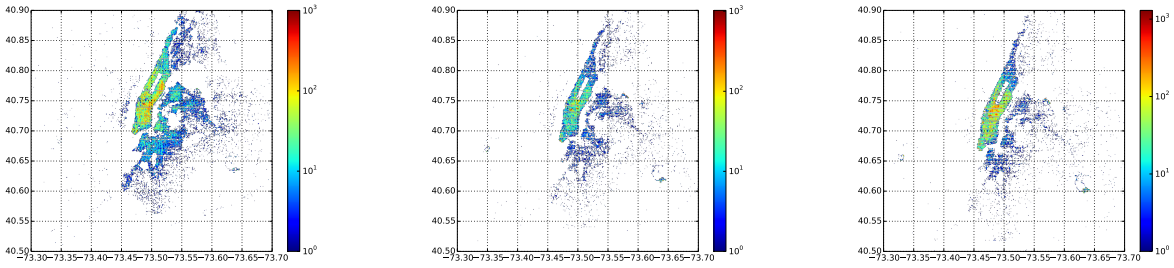
We ran all our evaluations on a Apache Spark 1.6.0 cluster on Amazon EC2. The Spark cluster we created has 15 m3.2xlarge instances, where each m3.2xlarge instance has 8 vCPUs, 30GB memory, and $2 \times 80$GB SSD.

The taxi drop-off dataset [1] in 2015 contains 143, 744, 411 drop-off records restricted in latitude/longitude $[40.5, 40.9] \times [-74.25, -73.7]$. We list the number of cells that contain at least one drop-off record using different granularity in Table 1. As the number of cells in setting 0.00001 and six hours is 141.75M, which is near to the total number of input records, we did not try finer granularity in our evaluation.

In all our implementations, the preparation phase includes reading once all CSV files, grouping drop-off records by cells, and caching the cells in Spark memory. The runtime of this



(a) $r_t = 1$, Jan 1, 2015 (b) $r_t = 1$, Jan 2, 2015 (c) $r_t = 1$, Jan 3, 2015



(d) $r_t = 0.25$, 00:00 - 6:00 (e) $r_t = 0.25$, 06:00 - 12:00 (f) $r_t = 0.25$, 12:00 - 18:00

**Figure 3: Temporal consistency. The first row shows the heat-map of three consecutive days. The second row shows the heat-map of three consecutive 6-hours of Jan 1, 2015. All spatial resolution is $r_s = 0.001$. Hot areas remain to be hot across times slots.**

**Table 1: # of Cells in Granularity Settings**

| Time Step Size / Cell Size | seven days | one day | six hours |
|---|---|---|---|
| 0.1 (10 km) | 1275 | - | - |
| 0.001(100 meters) | 1.93M | 6.45M | 13.59M |
| 0.00001 (1 meter) | - | - | 141.75M |

phase is fixed, about $18 \sim 20$ seconds, where the most time-consuming part is the string operations during reading and grouping. Caching results in Spark memory is very quick.

Table 2 illustrates the runtime of our three solutions in seconds. As the granularity gets finer, the runtime of BASIC and SHIFT is more volatile. It is because many sub-tasks during shuffle fail in this case, which would be restarted. We guess it is caused by the limited size of the network buffer or a glitch in Spark's network transfer module, but not sure. Our SR-$2,000,000$-500 takes around 30 seconds for all settings with fine granularity, as we only choose the top two million cells in the simplification phase. Note that the binary search in the simplification phase and the refinement for top $M$ results take little time on Spark. Thus, the main time-consuming part is still computing the $X_i$ score for the $K$ cells using BASIC.

## 5. CONCLUDING REMARKS

SR-$K$-$M$ is a solution to trade off between the speed and accuracy to detect hot spots in a distributed framework. It reduces the computation and network communication workload by simplification resulting in a huge speedup, and also improves accuracy by refinement. We believe SR-$K$-$M$ can work on other hot spot definitions adjusted by $K$, $M$, and how many expansion neighbors in refinement.

**Table 2: Runtime (seconds) without Preparation**

| Setting / Solution | SR-2M-500 | BASIC | SHIFT |
|---|---|---|---|
| 0.1, seven days | $\leq 1$ | $\leq 1$ | 5 |
| 0.001, seven days | 25 - 26 | 23 - 28 | 40 - 45 |
| 0.001, one day | 27 - 29 | 90 - 120 | 80 - 85 |
| 0.001, six hours | 27 - 30 | 300 - 500 | 100 - 120 |
| 0.00001, six hours | 30 - 33 | $\geq 2000$ | 600 - 700 |

## 6. REFERENCES


[1] New York City Taxi and Limousine Commission (NYC TLC). http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

[2] J. K. Ord and A. Getis. Local spatial autocorrelation statistics: Distributional issues and an application. *Geographical Analysis*, 27(4):286–306, 1995.

[3] S. Peng, J. Sankaranarayanan, and H. Samet. SPDO: High-throughput road distance computations on spark using distance oracles. In *ICDE*, pages 1239–1250, Helsinki, Finland, May 2016.

[4] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *ICDE*, pages 652–663, Shanghai, China, Apr. 2009.

[5] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *TKDE*, 22(8):1158–1175, Aug. 2010.

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *HotCloud*, Boston, MA, Jun 2010.