# Crowdsourcing computing resources for shortest-path computation

Alexandros Efentakis
Research Center "Athena"
Artemidos 6
Marousi 15125, Greece
efentakis@imis.athena-innovation.gr

Dimitris Theodorakis
National Technical University of Athens
15780 Athens, Greece
dth@atrion.gr

Dieter Pfoser
Research Center "Athena"
Artemidos 6
Marousi 15125, Greece
pfoser@imis.athena-innovation.gr

## ABSTRACT

Crowdsourcing road network data, i.e., involving users to collect data including the detection and assessment of changes to the road network graph, poses a challenge to shortest-path algorithms that rely on preprocessing. Hence, current research challenges lie with improving performance by adequately balancing preprocessing with respect to fast-changing road networks. In this work, we take the crowdsourcing approach further in that we solicit the help of users not only for data collection, but also to provide us their computing resources. A promising approach is parallelization, which splits the graph into chunks of data that may be processed separately. This work extends this approach in that small-enough chunks allow us to use browser-based computing to solve the pre-computation problem. Essentially, we aim for a Web-based navigation service that whenever users request a route, the service uses their browsers for partially preprocessing a large, but changing road network. The paper gives performance studies that highlight the potential of the browser as a computing platform and showcases a scalable approach, which almost eliminates the computing load on the server.

## Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: Graph algorithms

## General Terms

Algorithms

## Keywords

Graph Separators, Shortest Path, Preprocessing, Crowdsourcing

## 1. INTRODUCTION

Crowdsourcing is a process that outsources tasks to a distributed group of people. It affects the present context in two ways. For once, the focus of this work is how to deal with constantly changing road networks in the context of shortest-path (SP) computation, and, secondly, on how to utilize the computing resources controlled by the crowd in the process.

The single-pair shortest path (SPSP) problem of finding an exact shortest path of length $d(s,t)$ between a source $s$ and target $t$ in a graph $G = (V,E)$ is addressed by the classic Dijkstra algorithm [3], which unfortunately requires few seconds on continental sized road networks. More efficient algorithms involve a *preprocessing stage*, which produces a (linear) amount of auxiliary data that is then used to accelerate SP queries. While many effective techniques exist, an important category of SP algorithms is based on *graph separators* (GS). Examples of this category are HiTi [5] and Customizable Route Planning (CRP) [1]. During preprocessing, a multilevel partition of the graph is computed, in order to create a series of interconnected overlay graphs. A query starts at the lowest (local) level and moves to higher (global) levels as it progresses. When considering frequent updates to the road network (e.g., OpenStreetMap), GS methods have the advantage that changes remain local and therefore have limited impact on the overall SP computation.

Another advantage of GS approaches is that their *preprocessing may be easily parallelized*, since during preprocessing each thread needs to have access only to a limited portion of the road network. Hence, CRP significantly reduced its preprocessing time, when tested on a typical multicore workstation.

The objective of this work is to embrace change and trivialize the preprocessing stage. By defining an architecture that would allow us to use Web-browsers as a computing platform, we effectively delegate the task of incorporating the change back to the user and crowdsource not only the updates to our dataset but also the respective computing resources needed to deal with the consequences of such a change. Our system architecture is able to distribute typical GS preprocessing to an unlimited number of Web-clients. All preprocessing takes place on the clients, with the server only distributing data and gathering results from clients. What is more impressive, instead of setting up dozens of clients running custom OSes, we use Web-browsers and Javascript as our computing platform. To the best of our knowledge, this is the first work that advocates such an approach. Our extensive experimentation will show that even with a limited number of clients, we require significantly less preprocessing time than most SP algorithms and still are able to answer SP queries on continental sized road networks in almost 2ms.

The outline of this work is as follows. Section 2 describes our system architecture and design. Experiments establishing the performance characteristics of our system are given in Section 3. Finally, Section 4 gives conclusions and directions for future work.

## 2. CROWDSOURCING SHORTEST-PATH PREPROCESSING

The contribution of this work is to showcase a system, which efficiently distributes GS preprocessing to multiple Web-clients. This section provides the implemented architecture details and the nec-


*ACM SIGSPATIAL GIS,* November 6-9, 2012. Redondo Beach, CA, USA

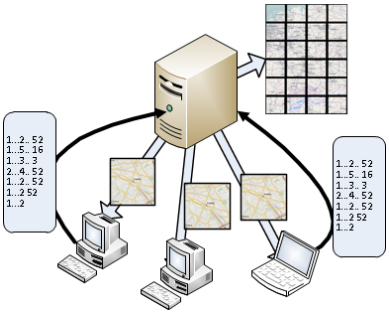

**Figure 1: System architecture. The server sends the cells' graph to the Web-clients and Web-clients return results to the server once they finish clique calculation**

essary optimizations added to typical GS preprocessing in order to adapt it to the distributed nature of such a system.

## 2.1 Graph separators

In GS approaches [1, 5] a partition C of the graph is computed. Then, the preprocessing stage builds a graph $H$ containing all boundary nodes and boundary arcs of $G$. It also contains a clique for each cell $C$: for every pair $(u, v)$ of boundary nodes in C, a clique arc $(u, v)$ is created whose cost is the same as the shortest path (restricted to $C$) between $u$ and $v$. Note that $H$ is an overlay: the distance between any two nodes in $H$ is the same as in $G$. In order to perform a SP query between $s$ and $t$, a unidirectional or bidirectional version of Dijkstra's algorithm must be run on the graph consisting of the union of $H$, $C(s)$ and $C(t)$. To accelerate queries, multiple levels of overlay graphs may be used, which is a common accelerating technique for most partition-based approaches.

Since each clique is calculated by using only the inner edges of $C$, GS preprocessing may be easily parallelized and each clique calculation may be assigned to a different process, a fact which will be exploited in our system architecture. The best SP query times possible with pure GS was achieved by CRP, which expanded on the ideas initially proposed in HiTi [5] and managed to achieve query times of 0.72ms for the continental road network of Europe.

## 2.2 System architecture

In our simple system architecture, the road network graph is stored on a server. Whenever a new Web-client connects, the server sends one or more cells (the cell's graph restricted to inner arcs of the cell) to the client for processing. When the client finishes calculating the clique for the specific cell(s) assigned to it, it returns the results to the server through AJAX. The server always keeps track of the either calculated or already assigned cells, so that when the next client connects, it is assigned some unprocessed and unassigned cells for clique calculation (Fig. 1).

In terms of the server, the cells' road network graphs are stored in a database. We also use a key-value store as a caching layer, so cells are preloaded in the cache to minimize accesses to the DB server. We additionally use the combination of a Web/application server, which receives requests from clients, assigns cells to them and gathers their results. Results are also stored on the caching layer for speed and efficiency. For a system with live traffic updates, results will never have to be stored permanently on the database, since they are valid for a very limited time (≈15min).

In terms of technical details, we exclusively used free tools for our server implementation: MySQL as the back-end DB server, REDIS as the caching layer and the combination of Nginx and Phusion Passenger for our Web/application server combination. The server application was written in Ruby-on-Rails.

## 2.3 Adapting road network preprocessing

Initially, to partition the graph $G$, we used METIS [6], a graph partitioning tool used frequently in the context of SP computation [4]. Since we will use multiple levels of overlay graphs, we create nested partitions by using METIS in a top-down fashion.

In typical GS preprocessing [1], to calculate cliques we must run a Dijkstra algorithm (restricted to each cell) from every boundary node. Clique calculation here is done entirely in the client's browser with Javascript and the server only sends cells to the browser and gathers the clique arcs calculated from clients once calculation is finished. Therefore, to reduce network communication time, we must minimize the data moved between the server and the Web-clients. This can be done either by (a) algorithmic optimizations and (b) network optimizations and batch grouping of data.

**Algorithmic optimizations**. The first necessary optimization was *nodes reordering*, i.e., we reordered nodes so that nodeIDs within a cell are consecutive. This way, we effectively minimize the cell's graph size sent to the client and the clique arcs size returned back .

In GS approaches, *overlay graphs of higher level partitions may be computed by using the overlay graphs of lower levels* to dramatically reduce preprocessing time. In our case, this technique not only reduces computation time on the Web-client but also significantly reduces network traffic as well.

Contrary to CRP, which uses an adjacency matrix representation of cliques for SP query efficiency, we resort to a adjacency list representation of clique arcs and we report only distances of boundary nodes that are direct descendants of the root of each Dijkstra algorithm we run, leading to a 56-71% reduction of the number of clique arcs created.

**Network optimizations**. Although the previous algorithmic optimizations reduce the overall size of cells and clique arcs, we can further reduce network traffic by *batch grouping cells and results*. Instead of the server sending a single cell to each client and then collecting a single result, the server now sends more cells (in a batch) to each Web-client. Moreover, by using JSON as our exchange format and enabling GZIP compression in our Web-server we can further reduce network communication time, which was necessary for adapting typical GS preprocessing to a crowdsourcing context. Consequently by using all those optimizations, our experiments will show that even with a limited nof clients, we can achieve smaller preprocessing times than most other SP algorithms and still achieve reasonable SP query times of almost 2ms.

# 3. EXPERIMENTS

The experimentation that follows assesses the performance of our distributed browser-based approach for various partition sizes (number of cells). We have experimented with 1-8 Web-clients to showcase scalability. Our experiments will report total, network communication and pure computation time, along with the Web-clients' memory usage and data sizes transmitted over the network. Finally, we show SP query times to compare our method with state-of-the-art GS techniques.

Our benchmark road network instance is the the European road network with 18 million nodes and 42 million arcs made available by PTV AG [2]. We experimented with, both, *travel times* and *travel distances*. During our experiments, the total number of cells per partition ranged from 32 to 131072.

Our experiments were conducted in a cloud environment using Amazon Web Services and a total of 5 virtual machines. One of them is the server and four of them are simulating Web-clients. All VMs are Medium High-CPU 64-bit instances and employ 1.7GB of memory, 350GB of storage and 2 virtual cores. The server VM

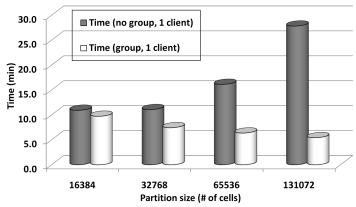


**Figure 2: Effect of batch grouping of cells for varying partition sizes and the whole road network (Travel times)**

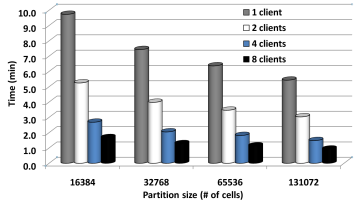


**Figure 3: Effect of multiple Web-clients for varying partition sizes for the whole road network and batch grouping of cells (Travel times)**

is running Ubuntu 11.10 and Web-clients are running Windows Server 2008 R2 with two Google Chrome processes each, summing up to 8 simultaneous Web-clients.

## 3.1 Overall performance

The first implementation we tested was the most straightforward one in which we *use the entire road network for every overlay graph calculation* for varying partition sizes. We used a single Web client and no batch-grouping of cells. Results showed that without batch grouping of cells or using intermediate overlay graphs, clique calculation of high-level partitions (i.e., partitions with small number of cells), requires a total *preprocessing time of more than 4h*. As far as the distribution of total time is concerned, for low-level partitions, network communication is the main bottleneck of our distributed browser-based approach. For medium to high-level partitions a small fraction of total time is devoted to network traffic and computation time on the client remains the main bottleneck.

**Batch grouping**. To minimize network traffic time, we repeated the above experiments for the lower-level partitions (since on those partitions network traffic is the bottleneck), once again for the whole road network, but this time batching cells so that each group of cells fits within a 300KB implicit limit (see Fig. 2).

Results showed that for the lowest-level partition used (the 131072 partition) we can batch-group 64 cells within the 300KB limit (per HTTP response from server). Therefore for this particular partition, through batch grouping of cells we were able to reduce total time from 27.8 min to *7.6 min*, for a total speedup of 3.6. We also see that through batch grouping of cells, lower-level partitions are now faster to calculate than higher level partitions.

**Multiple Web-clients**. Using multiple Web-clients, we repeated the above experiments using the previous batch-group settings for 1, 2, 4 and 8 Web-clients (see Fig. 3). It is evident that our approach scales very well for up to 8 clients. For 4 Web-clients we get a speedup of 3.6 and adding another 4 Web-clients results in an total speedup of 5.8. Combining, both, batch grouping and multiple Web-clients, we can calculate any low-level partition clique, in *less than 2min* by using 8 Web-clients, which is vast improvement over the 27.8min worst initial case.

**Using overlay graphs of lower level partitions**. In those experiments, we used four Web-clients and batched as many cells as possible within the implicit limit of 300KB. We experimented with four intermediate "helper" levels: The 131072 partition overlay graph was used for calculating the overlay graph of the 16384 partition, the 16384 partition overlay graph was used for calculating all overlay graphs for the 8192, 4096 and 2048 partitions, the 1024 partition overlay graph was used for calculating overlay graphs for the 128, 256 and 512 partitions and the 128 partition overlay graph was used for calculating the highest-level overlay graph of the 32 partition (Table 1 and Fig. 4).

**Table 1: Number of batched cells for varying partition sizes and helper partitions**

| # helper partition cells | # partition cells | # grouped cells |
|---|---|---|
| 128 | 32 | 1 |
| 1024 | 128 | 1 |
| 1024 | 256 | 1 |
| 1024 | 512 | 1 |
| 16384 | 1024 | 1 |
| 16384 | 2048 | 2 |
| 16384 | 4096 | 4 |
| 16384 | 8192 | 8 |
| 131072 | 16384 | 32 |

Table 1 shows, that even when we use "helper" partitions, we can no longer batch group cells for high-level partitions with less than 2048 cells. Fortunately for all those high-level partitions, network time is less than 13s for all sizes. Figure 4 also shows that *by using "helper partitions", the total time is close to 1min* for all available partition sizes. Evidently, even with our distributed browser-based GS preprocessing, computing the overlay graphs for the lowest level is still the most time consuming process.

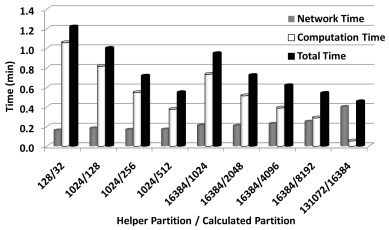


**Figure 4: Total, computation and network communication time for various partition sizes, using intermediate helper partitions, four Web-clients and batch grouping of cells (travel times)**

Conclusively, by using three intermediate "helper" partitions and four Web-clients, we may calculate the 128 partition in 4min. To the best of our knowledge, except CRP which has the fastest preprocessing time of 1min, no other SP algorithm implementation needs this little preprocessing time to provide an *SP query time in the range of 2ms*. As such, our solution of using browser-based computing provides comparable SP computation times.

**Memory usage**. During our experiments, when monitoring our Web-clients VMs, memory usage never exceeded 150MB. Therefore, our browser-based implementation has modest memory requirements and hence may run on any commodity workstation. Moreover, even our server VM with only 1.7Gb of main memory could easily handle our continent-sized road network, since it distributed its computation load entirely to the Web-clients.

**Network packets' sizes**. An important aspect of our distributed browser-based SP preprocessing is the amount of data transmitted over the network. The server sends the cells' graph to the clients and the clients return an adjacency list of clique arcs calculated for their assigned cells. Both network communications use a JSON representation of cells and results. As expected, each cell's size increases for higher-level partitions. Figure 5 shows the average size in KB of a network packet sent from the server to the client for varying partition sizes before and after enabling GZIP compression on the Web-server.

For high-level partitions, an uncompressed JSON representation

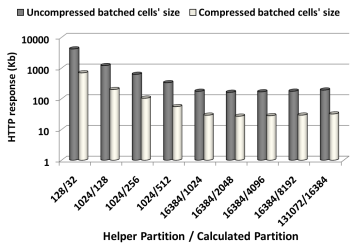


**Figure 5: A network packet's size (batched cells) sent from the server to the client for varying partition sizes using intermediate overlay graphs (before and after GZIP compression)**

**Table 2: SP query performance for 3 levels of overlay graphs.**

| | | | Travel times | | Travel distances | |
|---|---|---|---|---|---|---|
| | | | SP Query | Prepr.. | SP Query | Prepr. |
| # cells level 3 | # cells level 2 | # cells level 1 | time (ms) | time (min) | time (ms) | time (min) |
| 32 | 512 | 16384 | 1.812 | 5.7 | 2.811 | 6.2 |
| 32 | 1024 | 16384 | 1.909 | 5.1 | 2.970 | 5.6 |
| 32 | 2048 | 16384 | 2.122 | 5.8 | 3.250 | 6.4 |
| 128 | 512 | 16384 | 2.191 | 4.4 | 3.598 | 4.9 |
| 128 | 1024 | 16384 | 2.145 | 3.9 | 3.557 | 4.3 |
| 128 | 2048 | 16384 | 2.176 | 4.6 | 3.602 | 5.1 |

of a cell's graph has a size of almost 4MB for the 32 partition, and 1MB for the 128 partition, which is above our 300KB implicit limit. Fortunately, after enabling GZIP compression on the Web server, we can further compress network packets sizes by a factor of 6. With GZIP compression enabled, network packets (i.e., size of each batched cell group) for all partitions up-until the 128 partition are below 200KB.

The average size of results (i.e., JSON representation of an adjacency list of the clique arcs calculated on the Web-clients) remains stable below 300KB. This is attributed to the arc reduction optimization, nodes reordering and our adjacency list representation.

**Travel distances**. To evaluate the impact of a different metric on our approach, we also experimented with travel distances for the same European road network. Results showed that the total time for computing travel distances is *10-15% higher* than compared to computing travel times, since travel distances overlay graphs in lower and medium level partitions are denser (have 5% more arcs) when compared to travel times graphs. Still, a 10-15% increase translates to only 30s more computation time making our approach also a contender for travel distance computation.

**SP queries**. Although our main focus was to distribute typical GS preprocessing to multiple browsers, we also report how our preprocessing results may be used for typical SP computation.

Our SP query experiments were performed on a 6-core AMD Phenom II (3.3GHz) with 16Gb of main memory, running Ubuntu 12.04 64bit. Our source code was written in Oracle Java 7u4 and we used only one core for SP computation. Query times for a set of 10,000 random SP queries for both travel times and distances are presented in Table 2. Similar to CRP, we report the average time required for outputting the shortest path's length. We used 3 levels of overlay graphs, which proved to provide sufficient acceleration with reasonable preprocessing times. Preprocessing times are calculated using 4 Web-clients. For 8 Web-clients preprocessing time is smaller by a factor of *1.6*.

Results show that with *3min (8 Web-clients) of distributed browser-based preprocessing we can easily achieve SP query times of about* 2ms, which is comparable to CRP, the state-of-the-art GS approach. Our system may also support live updates. If a single arc weight changes, we only need to recompute one cell on each level (taking less than 2s).

# 4. CONCLUSION AND FUTURE WORK

This work introduced a novel approach for distributing SP preprocessing to multiple Web-clients. Instead of using a dedicated cluster of nodes connected to a network infrastructure, we use Web-browsers and Javascript as our computing platform. All the necessary computation work is distributed to the Web-browsers and the server just transmits cells and collects their results. Hence, not only the computation load on our server remains minimal but even the Web-clients hardly experience any load, since each cell's clique calculation takes less than 2s and memory usage remains below 150MB. Therefore, our client-side approach may work on any conventional workstation or current mobile device. Our extensive experimentation with a continent-sized road network showed that our approach is not only feasible, but very fast and efficient as well. *With 8 Web-clients, preprocessing for a continent-sized road network requires 3min and we can answer SP queries in almost 2ms*.

Although our work is the first distributed SP preprocessing effort in which clients do not share any common data structures, the *true novelty of our work is the use of Javascript and Web-browsers in the context of SP preprocessing*. Javascript is still considered a toy language, which is not truly capable of handling computing intensive applications. Our work clearly demonstrated that this is no longer the case. Furthermore, with the popularity of the Web, researchers now have access to an unlimited pool of computing resources and this work showcases plausible and cost effective ways to do that. By setting up a minimal Web server and relying entirely on open-source tools on a public Web service, we were able to create our dedicated cluster of unlimited nodes within minutes. In that spirit, we seriously hope we will encourage other researchers to use Javascript and Web-browsers as a means to parallelize their computing intensive problems.

## Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme under projects Initial Training Network "GEOCROWD" (`http://www.geocrowd.eu`, grant agreement No. FP7-PEOPLE-2010-ITN-264994) and "SimpleFleet" (`http://www.simplefleet.eu`, grant agreement No. FP7-ICT-2011-SME-DCL-296423).

# 5. REFERENCES

[1] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Proceedings of the 10th international conference on Experimental algorithms*, 2011.

[2] C. Demetrescu, A. V. Goldberg, and D. Johnson. *The shortest path problem. Ninth DIMACS implementation challenge, Piscataway, NJ, USA, November 13–14, 2006. Proceedings.* DIMACS Book 74. AMS , 2009.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[4] A. Efentakis, D. Pfoser, and A. Voisard. Efficient data management in support of shortest-path computation. In *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, CTS '11, pages 28–33, New York, NY, USA, 2011. ACM.

[5] S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering*, 14:1029–1046, 2002.

[6] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, December 1998.