SpringerPlus
a SpringerOpen Journal

RESEARCH Open Access

Algorithm for shortest path search in Geographic Information Systems by using reduced graphs

Rafael Rodríguez-Puente* and Manuel S Lazo-Cortés

Abstract

The use of Geographic Information Systems has increased considerably since the eighties and nineties. As one of their most demanding applications we can mention shortest paths search. Several studies about shortest path search show the feasibility of using graphs for this purpose. Dijkstra's algorithm is one of the classic shortest path search algorithms. This algorithm is not well suited for shortest path search in large graphs. This is the reason why various modifications to Dijkstra's algorithm have been proposed by several authors using heuristics to reduce the run time of shortest path search. One of the most used heuristic algorithms is the A* algorithm, the main goal is to reduce the run time by reducing the search space. This article proposes a modification of Dijkstra's shortest path search algorithm in reduced graphs. It shows that the cost of the path found in this work, is equal to the cost of the path found using Dijkstra's algorithm in the original graph. The results of finding the shortest path, applying the proposed algorithm, Dijkstra's algorithm and A* algorithm, are compared. This comparison shows that, by applying the approach proposed, it is possible to obtain the optimal path in a similar or even in less time than when using heuristic algorithms.

Keywords: Shortest path search algorithm; Geographic Information Systems; Network analysis; Reduced graphs; Dijkstra's algorithm

Introduction

From a practical point of view, a Geographic Information System (GIS) is a computer system capable of handling georeferenced data. These kinds of data refer to information associated with geographic coordinates (longitude, latitude). A GIS should also facilitate the relationship between socio-economic data (i.e. population density) and geographic data, this can be achieved through the generation of thematic maps (Jiang et al. 2010), a service for generating this kind of maps is described by (Rodríguez-Torres and Rodríguez-Puente 2010). The relevance of a GIS is closely related to the ability of building models or representations coming from the real world. This kind of system is very important because it facilitates the decision-making process and has a high social impact. Among the most demanded features in GIS we can mention those related to the analysis of routes, some examples are as follows:

- What is the shortest path between places x and y?
- What is the optimal path between places x and y considering a certain criterion?
- What is the lowest cost path between x and y via places $x_1, x_2, \ldots, x_n$?

Shortest path search has been widely studied. Many applications can be found in various branches of science, specifically in GIS. The road networks used by GIS to respond to the above requests are usually large and could have thousands of streets, that is why one should pay particular attention to how such information is processed.

One of the classic and most used algorithms for calculating the shortest path from an origin to a destination is Dijkstra's algorithm, it was first enunciated by Edsger Wybe Dijkstra (1959) and is one of the most used and discussed algorithms in the literature of graphs, the temporal complexity is $O(|E| + |V|log|V|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices of the graph. However, this algorithm is not efficient for

*Correspondence: rafaelrp@uci.cu
Universidad de las Ciencias Informáticas, Habana, Cuba

searching shortest path in large graphs (Fuhao and Jiping 2009).

Various modifications to Dijkstra's algorithm have been proposed by several authors. Some of these algorithms use heuristics to reduce the run time of shortest path search and we can classify them as follows:

1. Without data preprocessing, i.e.:
   - A* (A-star) algorithm (Hart et al. 1968). Improved Live long planing A* (Huang et al. 2007).
   - Bidirectional search (Pohl 1969).
   - In (Nazari et al. 2008) an approach based on restrictions on the search space is proposed.
2. With data preprocessing, i.e.:
   - Reach-Based Pruning (Gutman 2004).
   - Landmark-A* (Goldberg and Harrelson 2005; Goldberg and Werneck 2005).
   - Highway Hierarchies (Geisberger et al. 2008; Jagadeesh and Srikanthan 2008; Sanders and Schultes 2005; Song and Wang 2011; Wang et al. 2006).
   - Edge flags (Koehler et al. 2005; Möhring et al. 2006).
   - Geometric containers (Wagner and Willhalm 2007).
   - Precomputed Cluster Distances (PCD) (Maue et al. 2010).

Delling et al. (2009) show an overview of routing algorithms; all approaches show important advances in shortest path search and make possible a low response time in large graphs using heuristics.

One of the most used heuristic algorithms is the A* algorithm, the main goal is to reduce the run time by reducing the search space analyzing only the vertices that have better possibilities to appear in the shortest path. The results obtained by this algorithm depend on the heuristic function used to determine the order in which vertices are visited. If the selected heuristic is optimal the computational complexity is reduced to $O(n)$. That is why the A* algorithm is widely used for shortest path search.

One approach studied for shortest path search on large graphs is related to the use of some properties of the road networks, mainly to reduce the search space of the shortest path.

In the following paragraphs we will be referring to some relevant researches:

- Gutman proposes an approach (Gutman 2004) in which he defines a formal attribute of vertex called *reach*, in order to measure vertex relevance. The reach attribute is precalculated using the graph to reduce the run time of shortest path search.
- A relevant approach that uses a property of a road network is related to the hierarchy present in this kind of network. Many strategies use this approach, for example, Sanders and Schultes propose algorithms for constructing and querying highway hierarchies achieving a small run time and show the feasibility of this approach (Sanders and Schultes 2005).
- Bast et al. define an approach based on relevant nodes (transit nodes) for long-distance travel (Bast et al. 2007). It consists of making precalculations of shortest path between all pairs of transit nodes and from each potential source or destination to its access transit nodes. This approach needs an effective notion of "far away" and the optimal results are guaranteed depending on the local filter selected.
- Gonzalez et al. use the hierarchy of roads for partitioning the network into areas and make precalculations of shortest path in these areas (Gonzalez et al. 2007). This approach uses the fact that some roads are more traveled than others and drivers usually use the largest roads.
- Geisberger et al. propose an approach that uses only edges that are related with "important" nodes (Geisberger et al. 2008). Pfoser et al. present a shortest path algorithm that imitates human driving behavior by exploiting road network hierarchies (Pfoser et al. 2009).

As an important characteristic of the approaches described above, it may be determined that they are based on the idea that for calculation of large paths (in large networks), only high levels roads (highways, roads more traversed, etc.) of the hierarchical road network are needed. This consideration can reduce the run time of shortest path search algorithms, but can not guarantee to return the optimal path.

Various commercial systems use heuristic algorithms with the aim of reducing the run time (Bast et al. 2007). Various authors have defined heuristics for achieving this goal (Fei et al. 2010; Liu and Yang 2009; Nazari et al. 2008; Sun et al. 2008; Xu 2005). Fu et al. show a review of this kind of algorithms for shortest path search in transportation applications (Fu et al. 2006).

Heuristic algorithms are relevant for shortest path search in large graphs, even when an error is introduced, acceptable in most of the situations, but they do not guarantee to obtain the optimal path in all cases.

On the other hand, there are algorithms for reducing a graph (Liu et al. 2010; Lu and Liu 2007; Sadiq and Orlowska 2000). With the application of any algorithm on the reduced graph, obviously, a lower response time is achieved. However, in this case, reduction of data brings loss of information. Thus, obtaining a path that is the optimal in the original graph can not be guaranteed.

Rodríguez-Puente proposes a graph reduction algorithm without loss of information (Rodríguez-Puente 2010). It specifies a mechanism to obtain the original graph from which the reduced graph was obtained. This algorithm can be applied naturally to a GIS because a map is usually divided into: zip code, states, regions, etc. This fragmentation of the map contribute to create a partition according to the algorithm requirements. This algorithm has a computational complexity $O(n^4)$, which is a high cost for a response in real-time environment. However, in the proposed approach we make a graph reduction for each graph, only once, and the execution of the reduction algorithm is done only for data preprocessing. Highlighting that it does not affect the run time of shortest path search.

This article presents a modification of Dijkstra's shortest path search algorithm. It shows that it is possible to obtain the lowest cost path in all cases in a time similar to A* algorithm. Thus, the application of this algorithm in GIS can make improvements in services provided by this kind of systems. The use of the proposed algorithm integrated with the mentioned reduction algorithm will ensure efficiency in shortest path search, while maintaining accuracy.

The paper is organized as follows: first, a brief description of the graph reduction algorithm is provided. Second, the algorithm for finding shortest paths in reduced graphs is presented. Then, correctness of the algorithm is proved. Finally, some experimental results and conclusions are discussed.

Graph reduction

In order to achieve a better understanding of the proposal, certain definitions and notations related with graph theory must be introduced. Then, the selected graph reduction algorithm, used in the proposed approach, is presented.

Definitions and notations

Relevant definitions and notations related to the proposed approach are as follows:

Definition 1. A graph is a pair $G = (V, E)$, where:

- V is a set of vertices.
- E is a set of edges. An edge is an unordered pair of vertices (v_i, v_j) such that $v_i, v_j \in V$.

Definition 2. A weighted graph is defined as a structure $G = (V, E, f_c)$, where:

- V is a set of vertices.
- E is a set of edges.
- The function $f_c : E \to \mathbb{R}^+$ assigns to each edge a positive real value called cost.

Definition 3. A graph rewrite rule $R = (G_i, G_j, \psi_{in}, \psi_{out})$ over a graph $G = (V, E, f_c)$ consists of:

- a graph $G_i = (\{v_i\}, \phi)$, where $v_i \in V$.
- a graph $G_j = (V_j, E_j)$.
- two sets of embedding information ψ_{in}, ψ_{out} of the form $\{(v_m, c_1, c_2, v_n)\}$, where: $c_1, c_2 \in \mathbb{R}^+, v_m \in V_j, v_n \in \{V - V_j\}$; in the case of $\psi_{in}, \exists (v_n, v_i) \in E$, such that $f_c(v_n, v_i) = c_1$. After applying the rewrite rule, a new graph $H = (V_1, E_1, f_{c1})$ is obtained and it holds that $\exists (v_n, v_m) \in E_1$, such that $f_{c1}(v_n, v_m) = c_2$. Analogously to ψ_{in}, we define ψ_{out}, with edges orientation as the only difference.
- $V_1 = \{V - \{v_i\}\} \cup V_j$. $E_1 = E - E_t \cup E_j \cup E_k$, $(v_{t1}, v_{t2}) \in E_t$ if and only if ($v_{t1} = v_i$ and $v_{t2} \in V$) or ($V_{t1} \in V$ and $v_{t2} = v_i$). $(v_m, v_n) \in E_k$ if and only if $(v_m, c_1, c_2) \in (\psi_{in} \cup \psi_{out})$, $c_1, c_2 \in \mathbb{R}^+ . f_{c1} : E \to \mathbb{R}^+$,

$$f_{c1}(u, v) = \begin{cases} f_c(u, v) & \text{if } (u, v) \in Eandu, v \neq v_i \\ f_{cj}(u, v) & \text{if } (u, v) \in E_j \\ c_2 & \text{if } \exists (u, c_1, c_2, v) \in (\psi_{in} \cup \psi_{out}) \end{cases}$$

A graph rewrite rule also can be defined over an undirected graph, in this case, the sets ψ_{in} and ψ_{out} must be represented as an only set called ψ.

The set of edges that join vertex v_i with the vertices of the graph $G - G_i$ are called pre-embedding edges. After applying a rewrite rule, the edges that join a vertex of the graph G_j with a vertex of the graph $G - G_j$ are called post-embedding edges. The function ψ_{in} transforms the set of pre-embedding edges that are incident in a vertex v_i in post-embedding edges that are incident in one or more vertices $v_j \in V_j$. Similarly, the function ψ_{out} transforms pre-embedding outgoing edges from a vertex v_i in one or more post-embedding outgoing edges from several vertices $v_j \in V_j$.

Definition 4. A reduced graph is a tuple $G_r = (V_r, E_r, f, R)$, where:

- V_r is a set of vertices.
- E_r is a set of edges.
- $f : V_r \times V_r \times V_r \to (\mathbb{R}^+ \bigcup \{0, \infty\})$, is a function that for each (v_i, v_j, v_k) returns the cost of going from v_i to v_k through v_j, with v_k adjacent to v_j and v_j adjacent to v_i. Function f is obviously also defined for the cases where $v_i = v_j$ and/or $v_j = v_k$. In the trivial case, $f(v, v, v) = 0$.
- R is a set of rewrite rules over (V_r, E_r, f_c), where f_c is defined as $f_c(v, w) = f(v, v, w)$.

This definition is particularly important when it is associated with another graph, i.e., when a graph is reduced from another graph. We can state that a graph $G_r = (V_r, E_r, f, R)$ is reduced from a graph $G = (V, E, f_c)$, when applying the set of rewrite rules R to the graph G_r, the graph G is obtained.

In the case of function f, for all 3-tuple of vertices $v_i, v_j, v_k \in V_r$ it holds that $f(v_i, v_j, v_k) = f_c(v_i, v_j) + f_c(v_j, v_k)$. Notice that $f(v_i, v_i, v_j) = f_c(v_i, v_j)$. If v_i and v_j are not adjacent, the image of both functions would be infinite. This is the way in which we specify that two vertices are not adjacent.

Graph reduction algorithm

The reduction algorithm enunciated in (Rodríguez-Puente 2010) has as a key characteristic that it guarantees no loss of information through the incorporation of rewrite rules. However, an improved version is presented here, since it is necessary to differentiate between what are defined as internal and external vertices below.

This algorithm has two variables as input: a reduced graph $G = (V, E, f, R)$ and a partition over the set of vertices of the graph. On the other hand, the algorithm has as output, a reduced graph.

In first place, it is necessary to refine partition P in order to achieve an optimal path having the same cost of the optimal path obtained by Dijkstra's algorithm in the original graph; to do this, we introduce the following definition:

Definition 5. Let a graph $G = (V, E)$ and a partition P on V, a vertex $v_i \in V$ is internal if $\forall v_j \in V$, such that v_i and v_j are adjacent, it holds that v_i and v_j are in the same class of P; i.e. $[v_i] = [v_j]$ otherwise v_i is external.

For refining P, we use the following strategy:

- Two vertices v_i and v_j are in the same class of refined partition if, and only if:
    - v_i and v_j are in the same class in the original partition P.
    - v_i and v_j are internal vertices.
- For each external vertex a new equivalence class is created as a singleton containing only this vertex.

In Figure 1, we show an example of how to refine a partition using definitions of internal and external vertex.

Next, we create a new vertex w_i for each $A_i \in P, |A_i| > 1, i = 1..s$. $V' = w_i$ is a set of reduced vertices and $V - V'$ is the set of unreduced vertices in the reduced graph.

We add a vertex in the reduced graph for each class of the partition calculated in the previous step. If the cardinality of the class is 1, the vertex is considered as an unreduced vertex; in any other case, it is considered as a reduced one (*GetReducedVertices* method). Next, a set of edges is calculated. One edge can be added to the reduced graph if the two vertices of the edge belong to different equivalence classes (*GetEdges* method). With the addition of edges to the reduced graph, the cost function f_r of the reduced graph must be updated.

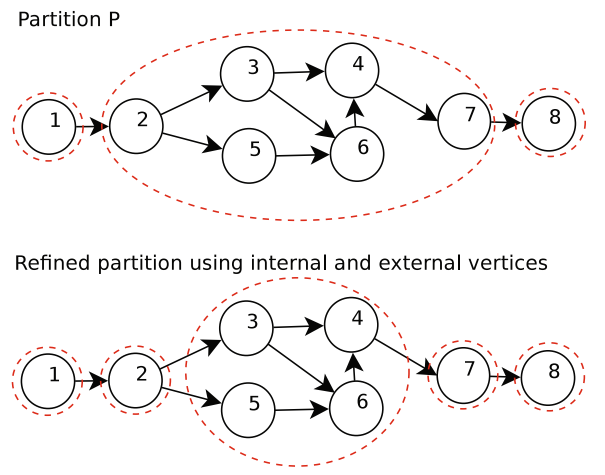

Figure 1 Example of partition refinement.

The creation of the set of rewrite rules is an essential step in the reduction algorithm. With the rewrite rules, the original graph can be obtained from the reduced graph. Therefore, rewrite rules guarantee no loss of information, and so the reduction process is reversible.

According to Definition 3, a graph rewrite rule is a quadruple of the form $(G_i, G_j, \psi_{in}, \psi_{out})$. Then, we create a rewrite rule for each reduced vertex in V', where:

- $G_i = (\{w_i\}, \phi), w_i \in V'$.
- $G_j = (A_i, E_i, f_{cj})$ is a subgraph of $G = (V, E, f_c, R)$, where exists an edge $(u, v) \in E_i$ if and only if $(u, v) \in E$ and $u, v \in A_i$; in addition $f_{cj}(u, v) = f_c(u, v)$.
- ψ_{in} is a set of quadruples of the form (v_m, c_1, c_2, v_n) such that for $v_m \in A_i$ and $v_n \in (V - A_i)$ and $(v_n, v_m) \in E$ and $(v_n, v_i) \in E_r$ it holds that $c_1 = f_{cj}(v_n, v_m)$; and $c_2 = f_{cj}(v_n, v_i)$.
- ψ_{out} is a set of quadruples of the form (v_m, c_1, c_2, v_n) such that for $v_m \in A_j$ and $v_n \in (V - A_j)$ and $(v_m, v_n) \in E$ and $(v_i, v_n) \in E_r$ it holds that $c_1 = f_{cj}(v_m, v_n)$; and $c_2 = f_{cj}(v_i, v_n)$.

The previous explanation corresponds to the implementation of *GetRewriteRules* method.

Another step that contributes to obtain the optimal path is the calculation of function f_r. Function f_r stores the cost of the shortest path from one vertex to another, traversing a reduced one.

Function f_r is calculated, initially, (*Updatefr* method) for each reduced vertex. This step is made in this way:

- Create an auxiliary graph. First, this graph is equal to the graph $G_j = (V_j, E_j, f_{cj})$ of the rewrite rule. Second, we add to this graph, vertices that are adjacent (in the original graph) to vertices of graph G_j (notice that these vertices are internal taking into account original graph and set V_j), and the edges that connect them.
- We apply MDijkstra algorithm (see next section) using all pairs of related vertices, identified in the previous step, as origin and destination vertices.
- The obtained costs and path are stored in f_r.

Additionally, for all 3-tuples of vertices $v_i, v_j, v_k \in V$, where v_j is a non-reduced vertex, $f_r(v_i, v_j, v_k) = f(v_i, v_j, v_k)$.

Path from v_i to v_k is also stored, with the goal of avoiding additional run time, when the shortest path search in a reduced graph is retrieved.

Algorithm *1* provides the detailed pseudo-code of the graph reduction algorithm.

Algorithm *1* *GraphReduction*

```
Input: A reduced graph $G = (V, E, f, R)$, where $R$ is a set of rewrite rules. A partition $P$ on $V$.
Output: A reduced graph $G_r = (V_r, E_r, f_r, R_r)$.
  $P = GetPartition(V, RE)$ {$P = V/RE = \{A_1, A_2, \ldots, A_s\}$, where $A_i = [a_i], a_i \in V, i = 1..s$}
  $GetReducedVertices(P)$
  $GetEdges(P)$
  $GetRewriteRules(P, G)$
  Create the reduced graph $G_r = (V_r, E_r, f_r, R_r)$
  for all $A_i \in P$, $|A_i| > 1$ do
    $Updatef_r(G, R_{ri}.G_i, R_{ri}.G_j, f_r)$ {$R_{ri}$ is the rewrite rule associated with the class $A_i$}
  end for
  return $G_r$
```

The complexity of the reduction algorithm would be determined by steps 6-8. According to the above description of *Updatefr*, this method calculates shortest path from all external vertices (taking into account the original graph) of V_j to all vertices of the auxiliary graph.

In a graph obtained from a network in a map, a vertex represents the intersection of two or more lines and an edge represents the connection between two intersections. That is why, in this kind of graph, there are no edges that intersect among them. Thus, we can assume that graphs representing the modeled network through a map are planar.

Moreover, in a graph with these characteristics, the degree of a vertex is generally equal to 4, except in a few cases. Thus it is assumed, without loss of generalization, that the degree of a graph that represents a network of this type is less than or equal to 10. Let $\Delta(G^+)$ the degree of G, the auxiliary graph has, at most, $a \cdot \Delta(G^+)$ vertices. In *Updatefr* method, MDijkstra algorithm is called for each adjacent vertex to any vertex of V_j (see Shortest path search algorithm section for temporal complexity of this algorithm), so the temporal complexity, in the worst case, is: $O(a \cdot \Delta(G^+) \cdot a \cdot \Delta(G^+) \log(a \cdot \Delta(G^+))) = O(\Delta(G^+)^2 \cdot a^2 \cdot \log(a) + log(\Delta(G^+)))$

The terms involving $\Delta(G^+)$ are constant, so the temporal complexity is $O(a^2 \cdot \log(a))$.

As a conclusion, the temporal complexity of Algorithm *1* is of polynomial order. The reduction process is made only once, as data preprocessing. This preprocessing task causes an increased in the spatial complexity but, with this approach, we can obtain lower run time in every shortest path computation over the reduced graph.

Reduction example

In this section we explain a very simple example to show the reduction process.

Let:

- G the graph of Figure 2(a).
- $P = (\{v_1, v_2, v_3\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_7\})$ a partition over the set of vertices of G.

In first place, we create the reduced vertices, one per each equivalence class of P. Thus, after this step, $G_r = (\{v_{r1}, v_4, v_6, v_5, v_7\}, \{\}, \{\}, \{\})$. Notice that $V_r = \{v_{r1}, v_4, v_6, v_5, v_7\}, E_r = \{\}, f_r = \{\}, R_r = \{\}$.

Then, we need to calculate the edges of G_r as is specified in the description of Algorithm *1*. If there is an edge between two vertices of G, and these vertices are unreduced in G_r, this edge is added to the reduced graph; for example the edge (v_5, v_7) in G is added to G_r. Additionally, if there is a vertex $v \in P_i$ in a class of $P(v \in V_r)$, and there exists an edge from v to other vertex u of G (u is unreduced vertex in G_r), the edge from the reduced vertex, that represents the class P_i of P, to the vertex u is added to G_r; for example the edge (v_2, v_4) in G is added to G_r as the edge (v_{r1}, v_4), v_2 is in the class of P represented by v_{r1}.

Therefore, the graph of Figure 2(b) is obtained. In addition, the rewrite rules are created. The graph G_i of the rewrite rule is $G_i = (\{v_{r1}\}, \{\})$ (see left of Figure 3), the graph G_j is created with the vertices of the class of P_i, represented by v_{ri}, and edges among them on G, as is presented on the right side of Figure 3. Once we created graphs G_i and G_j, the embedding information (ψ_{in} and ψ_{out}) must be specified, as is described in the specification of the reduction algorithm.

Finally, the function f_r is calculated. In the example of the reduced graph of Figure 2(b), we need to store the path from v_5 to v_4 and the path from v_5 to v_6, both through v_{r1}. In this case, $f_r(v_5, v_{r1}, v_4) = 6, f_r(v_5, v_{r1}, v_6) = 9$.

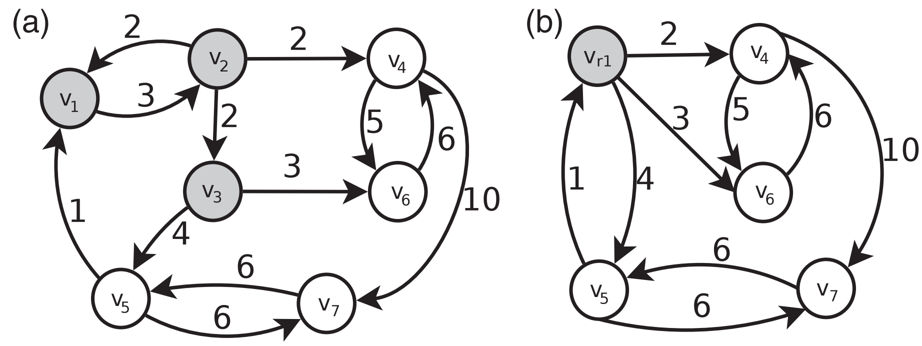

Figure 2 **Two graph examples: (a) is a graph and (b) is a reduced graph from (a).**

The application of the rewrite rules obtained (Figure 3) to G_r (Figure 2(b)) allows us to obtain the original graph G (Figure 2(a)). For this purpose, we enunciated Algorithm *2* based on Definition 3.

This algorithm has as input a reduced graph and a rewrite rule. If a reduced graph has more than one reduced vertex, the application of this algorithm for each reduced vertex would be sufficient to obtain the original graph.

Algorithm *2* Graph Rewrite Rule Application

```
Input: A reduced graph $G_r = (V_r, E_r, f_r, R_r)(R_r i = G_i, G_j, \psi_i n, \psi_o ut$ is a rewrite rule in $R_r$ associated to a reduced vertex $v_r$ of $G_r$).
Output: A reduced graph $G = (V, E, f, R)$.
  for all $e \in E_j$ do
    $AddEdge(G_r, e)$ {Add edge $e$ to graph $G_r$}
  end for
  for all $(u_1, c_1, c_2, u_2) \in \psi_{in}$ do
    $AddEdge(G_r, (u_2, u_1, costo = c_2))$ {Add an edge from $u_2$ to $u_1$ of cost $c_2$}
  end for
  for all $(u_1, c_1, c_2, u_2) \in \psi_{out}$ do
    $AddEdge(G_r, (u_1, u_2, costo = c_2))$ {Add an edge from $u_1$ to $u_2$ of cost $c_2$}
  end for
  $DeleteVertex(G_r, v_r)$ {Delete vertex $v_r$ from $G_r$}
  return $G_r$
```

Following, we show an example of application of the rewrite rule of Figure 3, using Algorithm *2*:

- Add to G_r (Figure 2(b)) the graph G_j of the rewrite rule (G_j is the right side graph of the rewrite rule).
- The pre-embedding edge (v_{r1}, v_5) of cost 1 is transformed in post-embedding edge (v_1, v_5) of cost 1.
- The pre-embedding edge (v_{r1}, v_4) of cost 2 is transformed in post-embedding edge (v_2, v_4) of cost 2.
- The pre-embedding edge (v_{r1}, v_6) of cost 3 is transformed in post-embedding edge (v_3, v_6) of cost 3.
- The pre-embedding edge (v_{r1}, v_5) of cost 1 is transformed in post-embedding edge (v_3, v_5) of cost 4.
- The vertex v_{r1} is eliminated from G_3.

After applying the rewrite rule we have obtained the graph G (Figure 2(a)). Thus, in the reduction process does not exist loss of information, that is, the reduction is reversible.

Shortest path search algorithm

In this section, a modification of Dijkstra's shortest path search algorithm is shown. The goal of the proposal is to obtain an optimal path with the same cost as the path returned by Dijkstra's algorithm, for the same origin and destination, but using a reduced graph.

Both, Dijkstra's algorithm and the one proposed, are based on iterations over the set of vertices. At each iteration, the algorithm will find a vertex so that the distance from the origin vertex to the selected vertex is minimal.

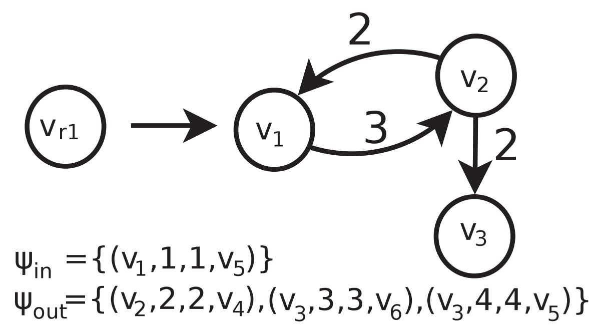

Figure 3 **Rewrite rule example.** On the left side is the graph $G_i = (\{v_{r1}\}, \{\})$, on the right side is the graph $G_j = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_1), (v_2, v_3)\})$ and on the bottom is the embedding information ψ_{out}.

This vertex is called pivot. Usually, the vertices are stored in a priority queue considering, as priority, the distance from the origin vertex. This data structure is used to facilitate the selection of the pivot. Besides, two vectors are updated during the execution of the algorithm. One of them (vector D) is updated with the lowest distance from the origin vertex to each vertex v_i (we refer to this distance as $D[v_i]$). The other one (vector P_r) is updated with the predecessor of each vertex in the shortest path from the origin vertex.

Every time that a pivot w_n is selected, the distances to its adjacent vertices are updated. If the distance from the origin vertex to the pivot ($D[w_n]$) plus the distance from the pivot to vertex v_i is lower than the distance from the origin vertex to v_i ($D[v_i]$), $D[v_i]$ is updated.

Additionally, there are two differences between Dijkstra's algorithm and the proposed one.

In the first place, a cost function $f : V \times V \times V \rightarrow (R^+ \cup \{0, \infty\})$ is used for calculating the cost from one unreduced vertex to another one, traversing a reduced vertex. Notice that, traditionally, the cost function of a graph has the cost of an edge.

The other difference in the proposed algorithm, is related to the actualization of distances to a reduced vertex. Let us consider an unreduced vertex w_n as pivot, it is necessary to update the distances to all adjacent vertices as described above. If a reduced vertex v_r is adjacent to the pivot, we have to update the distances to all vertices that are adjacent to v_r (see lines 15-22 of Algorithm *3*) using the cost function *f*, for guaranteeing the optimal result.

When analyzing the temporal complexity of the proposed algorithm, there are two differences with respect to Dijkstra's algorithm. The first one is the use of function *f*, this function is calculated at preprocessing time, so it does not affect the temporal complexity.

The second one implies the execution of one cycle. However, it should be noted that this cycle is repeated $\Delta(G^+)$ (constant, $\Delta(G^+) < 10$) times for each vertex that is stored in the queue.

Thus, $\Delta(G^+) < \log(|V|)$ for large graphs, this new cycle does not affect the temporal complexity. Concluding, temporal complexity of Dijkstra and MDijkstra algorithms are the same order. Also notice that, in a planar graph, we can establish a linear relation between vertices and edges. From the Euler's formula (Diestel 2010), it follows that $|E| \leq 3|V| - 6$ if $|V| \geq 3$. So, in the case of Dijkstra's algorithm in planar graphs, we can state that the temporal complexity is $O(|E| + |V|\log(|V|)) = O(|V|\log(|V|))$.

For applying the proposed approach, we need to reduce a graph only once. Then, we can make several shortest path search computations. In other words, we propose to make a data preprocessing for achieving a performance improvement in shortest path search.

This approach brings us the benefit of performing shortest path search in graphs with less vertices than other algorithms use, for instance, Dijkstra and A*. Therefore, it is logical for the proposal to achieve a lower run time. Nevertheless, it is necessary to demonstrate, that the path obtained by this proposal is optimal and equal (in terms of cost) to the one obtained by Dijkstra's algorithm. These demonstrations are shown in the following section.

The detailed pseudo-code of the proposed modification is presented in Algorithm *3*.

Table 1 shows a comparison of temporal complexity of Dijkstra, A* and MDijsktra algorithms. When analyzing A* algorithm considering optimal heuristics, it can be stated that its temporal complexity is $O(n)$, where n is the number of vertices of the graph. Besides, the temporal complexity of Algorithm *3* (MDijkstra) is $O(n_1 \log(n_1)) < O(n_1^2)$, where n_1 is the number of vertices of the reduced graph. Thus, if in the reduction process we obtain a graph $G = (V_r, E_r)$, such that $n_1 = |Vr| = \sqrt{n}$, the temporal complexity of both algorithms must be similar.

Algorithm *3 mDijkstra*

```
Input: A reduced and weighted graph $G = (V, E, f, R)$ and an origin vertex $v_{origin}$.
Output: A vector $D$ of minimum distances, a vector $P$ of predecessors.
1:  $C_n = \{\}$, $queue = PriorityQueue()$
2:  for all $v \in V$ do
3:      $D_n[v] = f(v_{origin}, v_{origin}, v)$, $Pr_n[v] = v_{origin}$
4:      $queue.add(D_n[v], v)$
5:  end for
6:  while not $queue.empty()$ do
7:      $w_n = queue.pop()$ {The vertex $w_n$ is the pivot}
8:      $C_n = C_n \cup \{w_n\}$
9:      for all $v \in adjacents(w_n)$ do
10:         if $D_n[v] > D_n[w_n] + f(w_n, w_n, v)$ then
11:             $D_n[v] = D_n[w_n] + f(w_n, w_n, v)$
12:             $Pr_n[v] = w_n$
13:             $queue.decreaseKey(D_n[v], v)$
14:         end if
15:         if $v$ is a reduced vertex then
16:             for all $s \in adjacent(G_r, v)$ do
17:                 if $D_n[s] > D_n[w_n] + f_r(w_n, v, s)$ then
18:                     $D_n[s] = D_n[w_n] + f_r(w_n, v, s)$
19:                     $P_{rn}[s] = v$
20:                 end if
21:             end for
22:         end if
23:     end for
24: end while
```

Table 1 Temporal and spatial complexity of Dijkstra, A* and MDijkstra algorithms

Algorithm	Temporal complexity	Temporal complexity (using Heap data structure)	Spatial complexity
Dijkstra	$O(\|E\| + \|V\|^2)$	$O(\|V\| + \log(\|V\|))$	$O(\|E\| + \|V\|)$
A*	$O(\|V\|)$, if the selected heuristic is optimal	$O(\|V\|)$, if the selected heuristic is optimal	$O(\|E\| + \|V\|)$
MDijkstra	$O(\|E\| + \|V\|^2)$	$O(\|V\| + \log(\|V\|))$	$O(\|E\| + \|V\|) + \|R\|$

However, as is impractical to obtain an optimal heuristics for this purpose, we can state that the proposal obtains a response in a lower run time than Dijkstra and A* algorithm if a condition $n_1 = |V_r| \leq \sqrt(n)$ is satisfied.

Generally, there is a trade off between efficiency and accuracy in algorithms that have large amount of data as input. The main result of the present work is the efficiency improvement of shortest path search in large graphs without affecting accuracy.

We have the possibility to make a shortest path search in the reduced graph between any pair of vertices of the original graph. It can be achieved by applying a rewrite rule to a proper reduced vertex. However, this involves an additional cost to shortest path search.

It is hard to state that an algorithm for shortest path search is better than other in all cases. In this case, our proposal need a higher space, associated to a preprocessing stage to calculate function f (see Definition 4), than classical Dijkstra's and A* algorithms (nevertheless, it should be highlighted that the preprocessing is made only once, but shortest path searches are made several times). However, MDijkstra algorithm gives a response in a lower run time.

Below, we prove the correctness of MDijkstra algorithm, with the aim of establishing that the proposed algorithm obtains an optimal path, and the cost of this path is the same as the cost of the path obtained by Dijkstra's algorithm. Next, we state a theoretical measure to ensure that the response time is lower than A* algorithm. This is the algorithm selected in the literature of shortest path search, to compare run times.

Correctness proof

In this paper, a new shortest path search algorithm is proposed. Therefore, it is necessary to prove that the path obtained by the proposal is optimal in all cases.

With the aim of facilitating the understanding of this section, the correctness proof of several lemmas is presented in Appendix A.

By Lemma 3, $D_{N-1}(v)$ has the minimum distance from vertex v_o to vertex v.

To prove the correctness of Algorithm 3, we shall prove that for any path $Ca = (v_o, v_1, v_2, ..., v_d)$ with distance vector Dc and predecessors vector P, it holds that $\forall v \in V, D_{N-1}(v) \leq Dc_{N-1}(v)$, where v is an unreduced vertex.

Theorem 1. $\forall n \in \{1, 2, .., N-1\}[\, Ca(0) = 0 \rightarrow \forall m < n+1 (Ca(m) < N) \rightarrow \forall m < n+1 [\, Dc(Ca(m)) + f_c(Ca(m), Ca(m+1)) = Dc(Ca(m+1))] \rightarrow D_{N-1}(Ca(n)) \leq Dc(Ca(n))]$

Proof. (By induction on n)
Base case $n = 0$ immediate by Lemma 2,
For $n = k + 1$:
By Lemma 1 in step $N-1$ all vertices have been visited.

$$D_{N-1}(Ca(k+1)) \leq D_{N-1}(Ca(k-1)) + f(Ca(k-1), Ca(k), Ca(k+1)) \tag{1}$$

The distance to a vertex v_i is less than or equal to the distance to a visited vertex v_j plus the distance from v_j to v_i, by Lemma 5

By induction hypothesis, $D_{N-1}(Ca(k-1)) \leq Dc(Ca(k-1))$, replacing $D_{N-1}(Ca(k-1))$ in (1),

$$D_{N-1}(C(k+1)) \leq Dc(Ca(k-1)) + f(Ca(k-1), Ca(k), Ca(k+1)) \tag{2}$$

Note that $Ca(k) = Pc_{N+1}(Ca(k+1))$ and $Ca(k-1) = Pc_{N-1}(Pc_{N-1}(Ca(k+1)))$, replacing $Ca(k)$ y $Ca(k-1)$ in (2),

$$D_{N-1}(Ca(k+1)) \leq Dc(Pc_{N-1}(Pc_{N-1}(Ca(k+1)))) + f(Pc_{N-1}(Pc_{N-1}(Ca(k+1))), Pc_{N-1}(Ca(k+1)), Ca(k+1)) \tag{3}$$

By Lemma 3, $Dc(Ca(k+1)) = Dc(Pc_{N-1}(Pc_{N-1}(Ca(k+1)))) + f(Pc_{N-1}(Pc_{N-1}(Ca(k+1))), Pc_{N-1}(Ca(k+1)), Ca(k+1))$, replacing in (3), $D_{N-1}(Ca(k+1)) \leq Dc(Ca(k+1))$. □

We can prove the correctness of Dijkstra's algorithm with a similar reasoning because the same invariants are satisfied. Thus, for the next proof we assume that

Dijkstra's algorithm is correct and satisfies invariants analogous to those defined for Algorithm *3*.

As demonstrated before, Algorithm *3* returns the shortest path in the reduced graph. However, it remains to prove that the cost of the shortest path obtained by the proposed algorithm and the one obtained by Dijkstra's algorithm (in the original graph without reducing it) are the same.

Let:

- $G = (V, E, f_c)$ a graph.
- $G_r = (V_r, E_r, f)$ a reduced graph obtained from the graph G.

Theorem 2. *Let* $Ca = (v_1, ..., v_n)$ *be a path of cost* c *obtained by applying Dijkstra's algorithm on the graph G, where* v_1 *and* v_n *are unreduced vertices on the graph* G_r*, then* $\exists Ca' = (u_1, u_2, \ldots, u_t)$ *with cost* c, $u_1 = v_1, u_t = v_n$*, such that* Ca' *is an optimal path on* G_r*.*

Proof. From Ca we can build a path Ca' of cost c on the graph G_r as follows:

- Substitute each sub-path $v_i, v_{i+1}, \ldots, v_{i+m}$ for a path v_i, v_k, v_{i+m} where:
  - $v_{i+j} \in [\,v_i]\,, j = 1..m$
  - v_i, v_{i+m} are external vertices. The other vertices are internal
  - v_k is the reduced vertex (in the graph G_r) that represents the equivalence class $[\,v_i]$

The cost of the path v_i, v_k, v_{i+m} is equal to the cost of the path $v_i, v_{i+1}, \ldots, v_{i+m}$, by definition of function f. Thus, the paths Ca and Ca' have the same cost.

Suppose that exists a path $Cb' = (u_1, u_2, \ldots, u_p)$ of cost $c_1 < c$ in the graph G_r, where $u_i \in V_r, i = 1..p$. Then we can obtain a path Cb of cost c_1 on the graph G as follows:

- Substitute each sub-path u_{i-1}, u_i, u_{i+1} by a path $u_{i-1}, u_j, u_{j+1}, u_{j+m}, \ldots, u_{i+1}$ of cost c_3 where:
  - u_{i-1}, u_{i+1} are unreduced vertices
  - $u_{j+t} \in [\,u_i]\,, j = 1..m$
  - $c_3 = f(u_{i-1}, u_i, u_{i+1})$

Therefore paths Cb and Cb' have the same cost (c_1), this leads a contradiction. Thus, there is no path that has less cost than Ca. □

Corollary 1. *Let* $Ca = (v_1, ..., v_n)$ *a path obtained by applying Dijkstra's algorithm on graph G,* $\forall i \in \{1, 2, ..., n\}$ *such that* $Ca[\,i]$ *is an unreduced vertex in* G_r*, it holds that the distance to* $Ca[\,i]$ *is equal to the distance obtained by MDijkstra algorithm on the reduced graph from* v_1 *to* $Ca[\,i]$*.*

Theorem 2 establishes that the cost of the shortest path from a vertex v_i to any vertex v_j (v_i and v_j being unreduced vertices in G_r) obtained by applying Algorithm *3* is the same as the cost of the shortest path calculated by Dijkstra's algorithm in the original graph (without reduction).

The fact that both source and destination must be unreduced vertices could be a limiting factor (in terms of the number of vertices to which one can calculate the shortest path) if one does not have a mechanism that allows obtaining a reduced graph G_{ri} from G_r where $v_i \in V$ (v_i is a vertex in the original graph $G = (V, E)$) is an unreduced vertex on G_{ri}. This can be accomplished by one or more expansions applying rewrite rules to the reduced vertex that contains vertex v_i.

Experimental results

The comparison of the results of shortest path search, applying Algorithm *3* (MDijkstra), Dijkstra's algorithm and A* algorithm, provides elements emphasizing the advantages of the proposed approach. Besides, correctness proof of the proposed shortest path search algorithm is made.

Algorithm *3* was coded in Python, using the NetworkX library (Hagberg et al. 2008). This library provides an implementation of Dijkstra's and A* algorithms, allowing to compare the three algorithms on the same technology and with efficient data structures. NetworkX uses a priority queue, implemented with a Heap, to find the shortest path using Dijkstra and A* algorithms. With this implementation, the complexity is $O(|E| + |V| \log(|V|))$.

It is well-known that there are several techniques to make performance improvement on shortest path search, based on Dijkstra's and A* algorithms; Zeng and Church compare some of them (Zeng and Church 2009). This performance improvement depends on several things, for example: programming language, data structures used in the implementation of algorithms, among others. Therefore, in order to be impartial with the proposal, we compare the proposed algorithm only with the implementation of Dijkstra's and A* algorithms in the NetworkX library.

The algorithms were run on a Pentium 4 (3.2 GHz) with 1.5 Gb of RAM and the Kubuntu 11.10 operating system.

Two graphs were used for experimental test: one was obtained from a cartography of the North Carolina State[a] and the other represents the road network of San Francisco[b]. The first graph, obtained from North Carolina cartography, has 41810 vertices. This graph was reduced twice. First, we arbitrarily construct two sets of polygons using zip codes. The first one has 30 polygons. The second one has 5 polygons (the second set of polygons does not depend of the first one). Obviously in the second case polygons are larger. In both reductions we use the equivalence relation "in". If two points are into the same polygon, then they are related through relation "in". We obtain a reduced graph of 1826 using the first set of polygons, and a reduced graph of 250 vertices using the second set.

The second graph, obtained from San Francisco cartography, has 149756 vertices and it was also reduced twice, using the equivalence relation defined above and two new arbitrary sets of polygons. The first set has 10 polygons and the second one has 4 polygons. In the first reduction, using the first set of polygons we obtain a reduced graph of 2617 vertices. Using the second set of polygons, we obtain another reduced graph of 769 vertices.

Dijkstra's and A* algorithms were executed on the original graphs and the proposed algorithm was applied to the reduced ones. Each algorithm was executed 10 times; the highest and lowest values were discarded. Finally, the average time among the remaining 8 values are shown.

Table 2 shows a comparison among the three selected algorithms based on the run time of shortest path search.

Discussion

The results shown in Table 2 confirm the fact that, for large graphs, the run time of shortest path search with the proposed approach would be smaller than the run time obtained with classical approaches.

If in the reduction process we obtain a graph $G = (V_r, E_r)$, such that $n_1 = |Vr| = \sqrt{n}$, the temporal complexity of both algorithms (Dijkstra and MDijkstra) must be similar. However, as is impractical to obtain an optimal heuristics for this purpose, we can state that the proposal obtains a response in a lower run time than Dijkstra's and A* algorithm if a condition $n_1 = |V_r| \leq \sqrt{n}$ is satisfied. Thus, if we assume that we have sufficient memory for storing reduced graphs, the proposed approach is better than Dijkstra's and A* algorithms; taking into account that if we reduce original graph as proposed before, always we can obtain a response in a lower runtime. The proposal is not useful when the available memory is low and does not permit to store reduced graphs.

Table 2 Time of shortest path search with Dijkstra's and A* algorithms in two original graphs (G_1, G_2) and time of shortest path search in four reduced graphs with the proposed approach

Graph	Algorithm	NV[a]	Time (seconds)	Optimal path
G_1	Dijkstra		0.6160	yes
	A* (h=0)	41810	0.4938	yes
	A* (h=Euclidean distance)		0.0200	no
$G_{r1.1}$	Algorithm *3* (MDijkstra)	250	0.0036	yes
$G_{r1.2}$	Algorithm *3* (MDijkstra)	1826	0.0265	yes
G_2	Dijkstra		3.0249	yes
	A* (h=0)	149756	2.2108	yes
	A* (h=Euclidean distance)		0.1011	no
$G_{r2.1}$	Algorithm *3* (MDijkstra)	765	0.0193	yes
$G_{r2.2}$	Algorithm *3* (MDijkstra)	2617	0.0722	yes

[a] Number of vertices of the graph.

In the case of the run time of Algorithm *3* (MDijkstra) on the graph $G_{r1.2}$, the obtained time is higher than the one obtained by A* algorithm. The reason of this result is that the graph $G_{r1.2}$ has a number of vertices considerably higher than the square root of the number of vertices of G_1. Notice that we state that the number of vertices of the reduced graph must be less than or equal to the square root of the number of vertices of the original graph. In the case of the graph $G_{r2.2}$ a lower run time than the one obtained by A* algorithm is achieved, although the number of vertices is higher than the square root of the number of vertices of G_2.

The selection of origin and destination of the shortest path search in a GIS is usually made using a map, i.e. a user selects these points by clicking in the map shown by the GIS. We believe that, at any time that a user selects an origin or a destination point, the GIS can make an expansion of the reduced graph, using the extent of the map that is visualized and the selected point. If a system for shortest path search is implemented in this way, the time needed to expand a reduced vertex would be irrelevant for the shortest path search, considering that the temporal complexity of expanding a reduced vertex is $O(a)$, where $a = max\{|A_i|, A_i \in P\}$.

Most algorithms developed lately for shortest path search make efficiency improvement by reducing the search space, these approaches cause loss in accuracy. The presented approach makes use of a graph reduction algorithm without loss of information, in order to obtain a better run time of the search. This approach maintains the accuracy because the reduction algorithm guarantees no loss of data (see Table 1).

Generally, heuristic algorithms are developed in order to reduce the run time of a specific algorithm, which solves some problems whose optimal solution involves a high computational cost. Many heuristic algorithms are developed for shortest path search in GIS, with the assumption that a low bound of error is admissible in this area. However, with the proposed approach, it is possible to obtain the optimal path in a similar time, and even in less time, than with heuristic algorithms, as shown in Table 2.

Conclusions

In this paper, an algorithm for shortest path search on reduced graphs is developed. Experimental results show that the proposed algorithm is more efficient than Dijkstra's algorithm on large graphs. In addition, we can conclude the following:

- The proposed approach is particularly applicable to GIS, due to the way in which users perform a shortest path search in this kind of systems. This allows us to expand vertices avoiding the influence of the time used in this operation on the shortest path search.
- The use of reduced graphs significantly reduces the response time in the shortest path search. That is one of the two main approaches used in literature to reduce the computational cost of this operation.
- The shortest path search on a reduced graph ensures scalability regarding the size of the graph on which the analysis is performed.
- We prove that the proposed algorithm allows us to obtain an optimal path in a reduced graph. The cost of the obtained path is equal to the cost of the path found using Dijkstra's algorithm on the original graph.
- We have developed a method capable of performing shortest path search in a run time similar to A* algorithm (with h=0 and h=Euclidean distance).

Future work

The modifications made on Dijkstra's algorithm are related to the use of a new function that has the cost of going through a reduced vertex. Therefore, we can modify other algorithms to make shortest path search in reduced graph (like A* algorithm), whenever the cost of going through a reduced vertex is considered as the cost of the path.

Appendix

A Demonstration of cycle invariants of Algorithm *3*

Preconditions that must be met to prove the correctness of Algorithm *3* are expressed by the following definitions and notations:

- $G = (V, E, f_c)$ is a weighted graph. Without loss of generality we assume that $V = \{0, 1, ..., M-1\}$ to make demonstrations less complex.
- $G_r = (V_r, E_r, f, R)$ is a reduced graph from G and the equivalence relation RE. Without loss of generality we assume that $V_r = \{0, 1, ..., N-1\}$. It is important to notice that in each path of a reduced graph, between two reduced vertices there are, at least, two unreduced vertices, as is shown in Figure 4.
- $\forall n < N$, in the execution of Algorithm *3* we define:
    - A vertex w_n, the vertex selected in step n.
    - A set $C_n \subseteq V_r$, the set of vertices visited in step n. $C_0 = \{v_o\}$, $C_{n+1} = C_n \bigcup \{w_n\}$.

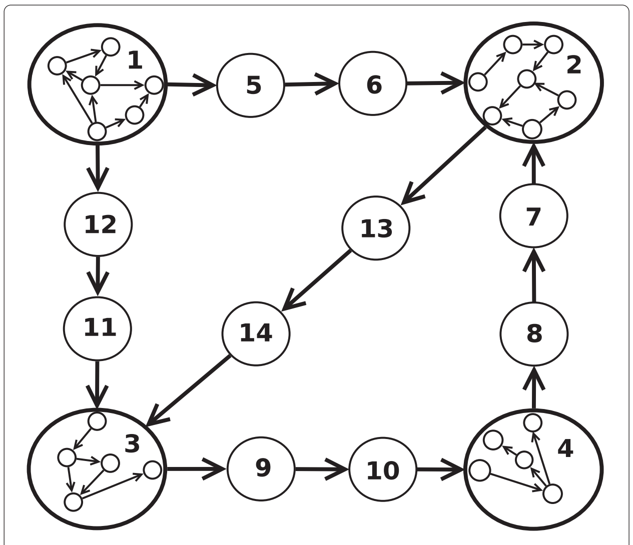

Figure 4 Reduced graph example. Vertices 1,2,3 and 4 are reduced vertices, the rest are unreduced ones.

    - D_n represents the minimum distance from v_o to each vertex $v \in V_r$ as far as it is known in step n. $D_0(v_o) = 0$, $D_{n+1}(v) = Min(D_n(w_n) + f_c(w_n, v), D_n(v)) = Min(D_n(P_n(w_n)) + f(P_n(w_n), w_n, v), D_n(v))$.
    - P_n store, for each vertex, the predecessor in the shortest path from v_o to v_d, as far as it is known in step n. $P_0(v_o) = v_o$,

$$P_{n+1}(v) = \begin{cases} w_n & \text{if} D_n(v) > D_n(w_n) + f_c(w_n, v) \\ P_n(v) & \text{in other case} \end{cases}$$

For the correctness proof it is necessary to demonstrate that the following cycle invariants are held:
$\forall n < N$:

1. $|C_n| = n + 1$. In the iteration n, there are $n + 1$ visited vertices.
2. $D_n(v_o) = 0 \wedge P_n(v_o) = v_o$. The distance from origin vertex to itself is 0 at any iteration. The predecessor of the origin vertex is the vertex itself.
3. $D_n(v) = D_n(P_n(P_n(v))) + f(P_n(P_n(v)), P_n(v), v)$. The distance to a vertex depends on the distance to its predecessor in the shortest path.
4. $\forall v \in C_n \; D_{n+1}(v) = D_n(v)$. The distance to a vertex in the step n is the same that the distance in the step $n + 1$, for all visited vertices.
5. $\forall v_i, v_j \in V[\, v_j \in C_{n+1} \rightarrow D_{n+1}(v_i) \leq D_{n+1}(P_n(v_j)) + f(P_n(v_j), v_j, v_i)]$. The distance to any vertex v_i is less than or equal to the distance to a visited vertex v_j plus the distance from v_j to v_i.

Lemma 1. $\forall n < N$, $|C_n| = n + 1$

Proof. (By induction on *n*)
From the definition of the algorithm, at each step a vertex w is visited, in step 0 vertex v_o is visited, thus in the base case we have $C_0 = \{v_o\}$, $|C_0| = 1$,
For $n = k+1$, $C_{k+1} = C_k \bigcup \{v\}$, being v the visited vertex in step $k+1$, therefore $|C_{k+1}| = |C_k| + |\{v\}| = k+2$. □

Lemma 2. $\forall n < N, D_n(v_o) = 0 \wedge P_n(v_o) = v_o$

Proof. First, we visit vertex v_o and update $D_n(v_o) = 0$, i.e., the minimum distance from v_o to itself is 0, the function D_n has its domain in $\mathbb{R}^+ \bigcup \{0, \infty\}$, so the smallest possible value that can be achieved is 0;
Let $cost = D_n(P_n(w_n)) + f(P_n(w_n), w_n, v), \forall w_n, v \in V$, it holds that $0 \leq 0 + cost$, because the image of the function f is $\mathbb{R}^+ \cup \{0, \infty\}$ and the vector $D(V_r)$ is initialized from f.
The condition $D_n(v_o) > D_n(w_n) + f(P_n(w_n), w_n, v_o)$ is never satisfied, thus $D_n[v_o]$ and $P_n[v_o]$ never change. □

Lemma 3. $\forall n < N$, $D_n(v) = D_n(P_n(P_n(v))) + f(P_n(P_n(v)), P_n(v), v)$

Proof. (By induction on *n*)
The base case $n = 0$, $\forall v \in V_r, D_0(v) = f_c(v_o, v)$, by preconditions.
$f(v_o, v_o, v) = f_c(v_o, v_o) + f_c(v_o, v) = f_c(v_o, v)$, by definition of f and f_c, replacing f by f_c:
$D_0(v) = 0 + f(v_o, v_o, v) D_0(v) = D_0(v_o) + f(v_o, v_o, v)$, by Lemma 2
$D_0(v) = D_0(P_0(v_o)) + f(P_0(v_o), v_o, v)$, by Lemma 2
For $n = k+1$:
Choose $w_{k+1} \in V \setminus C_k$ such that $D_{k+1}(w_{k+1})$ is minimal, $C_{k+1} = C_k \bigcup \{w_{k+1}\}$.
Case 1: If $D_{k+1}(v) > D_{k+1}(P_{k+1}(w_{k+1})) + f(P_{k+1}(w_{k+1}), w_{k+1}, v)$, then $D_{k+1}(v) = D_{k+1}(P_{k+1}(w_{k+1})) + f(P_{k+1}(w_{k+1}), w_{k+1}, v) \wedge P_{k+1}(v) = w_{k+1}$
Case 2: If case 1 is not satisfied, $D_{k+1}(v) = D_k(v), P_{k+1}(v) = P_k(v)$, $D_{k+1}(v) = D_k(P_k(P_k(v))) + f(P_k(P_k(v)), P_k(v), v)$, by induction hypothesis, replacing $P_k(v)$ by $P_{k+1}(v)$ $D_{k+1}(v) = D_k(P_{k+1}(P_{k+1}(v))) + f(P_{k+1}(P_{k+1}(v)), P_{k+1}(v), v)$. □

Lemma 4. $\forall v \in C_n\ D_{n+1}(v) = D_n(v)$

Proof. Let $v \in C_n, w_n \in V \setminus C_n$
$w_n \in C_{n+1}$ by definition.
$D_n(v) \leq D_n(w_n)$, otherwise vertex w_n was visited before vertex v,
$D_n(v) \leq D_n(w_n) + f_c(w_n, v) = D_n(P_n(w_n)) + f(P_n(w_n), w_n, v)$,
$D_{n+1}(v) = D_n(v)$, by definition of $D_{n+1}(v)$. □

Lemma 5. $\forall n < N$, $\forall v_i, v_j \in V[v_j \in C_{n+1} \rightarrow D_{n+1}(v_i) \leq D_{n+1}(P_n(v_j)) + f(P_n(v_j), v_j, v_i)]$

Proof. (By induction on *n*)
The base case $n = 0$, $C_0 = \{v_j\}, D_0(v_j) = 0$, by definition, notice that v_j is the only vertex in C_0 (in the base case, if $v_j \in C_0, v_j$ is the origin vertex).
$P_n(v_j) = v_j$ by Lemma 2.

$\forall v_i \in V, f(v_j, v_j, v_i) = f(P_0(v_j), v_j, v_i)$, by definition of f, and $D_0(v_i) = f(v_j, v_j, v_i)$. Thus $D_0(v_i) \leq 0 + f(P_0(v_j), v_j, v_i)$, $D_0(v_j) = 0 = D_0(P_0(V_j))$ $D_0(v_i) \leq D_0(P_0(v_j)) + f(P_0(v_j), v_j, v_i)$
$D_1(v_i) \leq D_0(v_i)$, from the definition $(D_{n+1}(v) = Min(D_n(w_n) + f_c(w_n, v), D_n(v)))$ and $D_1(v_j) = D_0(v_j) = 0$ (notice that v_j is the origin vertex). Replacing D_0 by D_1:
$D_1(v_i) \leq D_1(P_0(v_j)) + f(P_0(v_j), v_j, v_i)$
For $n = k+1$:
Case 1: $v_j \in C_k$ $D_{k+1}(v_i) \leq D_k(v_i)$, by definition $D_{k+1}(v_i) \leq D_k(P_k(v_j)) + f(P_k(v_j), v_j, v_i)$, by induction hypothesis $D_{k+1}(v_i) \leq D_{k+1}(P_k(v_j)) + f(P_k(v_j), v_j, v_i)$ by Lemma 4
Case 2: $v_j = w_k$.
$D_{k+1}(v_i) \leq D_k(P_k(w_k)) + f(P_k(w_k), w_k, v_i)$, by definition $D_{k+1}(v_i) \leq D_{k+1}(P_k(w_k)) + f(P_k(w_k), w_k, v_i)$, by Lemma 4, replacing w_n by v_j:
$D_{k+1}(v_i) \leq D_{k+1}(P_k(v_j)) + f(P_k(v_j), v_j, v_i)$.

□

Endnotes

[a]Available in
http://grass.osgeo.org/sampledata/north_carolina/
[b]Available in
http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm

Competing interests
We declare that we have no conflict of interest.

Authors' contributions
Both authors, viz. RRP and MSLC, were involved in drafting the article and revising it critically, until the final approval of the version to be submitted. Programming and experiments were carried out by RRP. Both authors read and approved the final manuscript.

Acknowledgements
Authors are very grateful to Yvonne Collada Peña and Yoan Martínez Márquez for the detailed revision of the manuscript. Authors also acknowledge the critical, thorough and detailed revision of the anonymous reviewer as well as all the valuable comments and suggestions.

Received: 14 November 2012 Accepted: 14 May 2013
Published: 1 July 2013

References
Bast H, Funke S, Sanders P, Schultes D (2007) Fast routing in road networks with transit nodes. Science 316(5824): 566. http://www.ncbi.nlm.nih.gov/pubmed/17463281
Delling D, Sanders P, Schultes D, Wagner D (2009) Algorithmics of large and complex networks. chap. Engineering Route Planning Algorithms.

Springer-Verlag, Berlin, Heidelberg, pp 117–139. doi:http://dx.doi.org/10.1007/978-3-642-02094-0_7
Diestel R (2010) Graph Theory, fourth edn. Graduate Text in Mathematics. Springer-Verlag. URL http://diestel-graph-theory.com/index.html
Dijkstra EW (1959) A note on two problems in connection with graphs. Numerische Mathematik 1: 269–271
Fei S, Wei D, Bing Z (2010) Traffic information management and promulgating system based on gis In: 2010 International Conference on Optoelectronics and Image Processing (ICOIP), *ICOIP '10*, vol. 2. IEEE Computer Society, Los Alamitos, pp 676–679. doi:http://dx.doi.org/10.1109/ICOIP.2010.243
Fu L, Sun D, Rilett LR (2006) Heuristic shortest path algorithms for transportation applications: state of the art. Comput Oper Res 33: 3324–3343. doi:10.1016/j.cor.2005.03.027. URL http://dl.acm.org/citation.cfm?id=1143184.1143201
Fuhao Z, Jiping L (2009) An algorithm of shortest path based on dijkstra for huge data. In: Chen Y, Deng H, Zhang D, Xiao Y (eds) Sixth International Conference on Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09, vol. 4. IEEE Computer Society, Los Alamitos, pp 244–247. http://dl.acm.org/citation.cfm?id=1800875.1800929
Geisberger R, Sanders P, Schultes D, Delling D (2008) Contraction hierarchies: faster and simpler hierarchical routing in road networks. In: Proceedings of the 7th international conference on Experimental algorithms, WEA'08. Springer-Verlag, Berlin, Heidelberg, pp 319–333. http://dl.acm.org/citation.cfm?id=1788888.1788912
Goldberg AV, Harrelson C (2005) Computing the shortest path: A search meets graph theory. In: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '05. Society for Industrial and Applied Mathematics, Philadelphia, pp 156–165. http://dl.acm.org/citation.cfm?id=1070432.1070455
Goldberg AV, Werneck RF (2005) Computing point-to-point shortest paths from external memory. Area. http://www.cs.princeton.edu/courses/archive/spr06/cos423/Handouts/GW05.pdf
Gonzalez H, Han J, Li X, Myslinska M, Sondag JP (2007) Adaptive fastest path computation on a road network: a traffic mining approach. In: Proceedings of the 33rd international conference on Very large data bases, VLDB '07. VLDB Endowment, Vienna, Austria, pp 794–805. http://dl.acm.org/citation.cfm?id=1325851.1325942
Gutman RJ (2004) Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: Arge L, Italiano GF, Sedgewick R (eds) Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics. SIAM, New Orleans, pp 100–111
Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2): 100–107. doi:10.1109/TSSC.1968.300136
Huang B, Wu Q, Zhan FB (2007) A shortest path algorithm with novel heuristics for dynamic transportation networks. Int J Geogr Inf Sci 21(6): 625–644. doi:10.1080/13658810601079759
Jagadeesh G, Srikanthan T (2008) Route computation in large road networks: a hierarchical approach. Intell Transp Syst, IET 2(3): 219–227. doi:10.1049/iet-its:20080012
Jiang L, Qi Q, Zhang A (2010) The thematic mapping system on internet. In: Liu Y, Chen A (eds) 2010 18th International Conference on Geoinformatics. IEEE Computer Society, Piscataway, pp 1–4. doi:10.1109/GEOINFORMATICS.2010.5567802
Koehler E, Moehring RH, Schilling H (2005) Acceleration of shortest path and constrained shortest path computation. Exp Efficient Algorithms 3503(1126): 126–138
Liu YC, Yang DH (2009) A spatial restricted heuristic algorithm of shortest path. In: International conference on artificial intelligence and computational intelligence, 2009. AICI '09, vol. 2. IEEE Computer Society, Los Alamitos, pp pp 36–39. http://www.computer.org/csdl/proceedings/icsssm/2005/8971/02/01500172-abs.html
Liu QX, Cao BX, Zhao YW (2010) An improved verification method for workflow model based on petri net reduction. In: 2010 The 2nd IEEE International Conference on Information Management and Engineering (ICIME), vol. 2. IEEE Computer Society, Piscataway, pp 252–256. doi:10.1109/ICIME.2010.5477436
Lu K, Liu Q (2007) An algorithm combining graph-reduction and graph-search for workflow graphs verification. In: Shen W, Yang Y, Yong J, Hawryszkiewycz I, Lin Z, Barthès JPA, Maher ML, Hao Q, Tran MH (eds) 11th International conference on computer supported cooperative work in design, 2007. CSCWD 2007. IEEE Computer Society, Washington, pp 772–776. doi:10.1109/CSCWD.2007.4281534
Maue J, Sanders P, Matijevic D (2010) Goal-directed shortest-path queries using precomputed cluster distances. J Exp Algorithmics 14: 2:3.2–2:3.27. doi:http://doi.acm.org/10.1145/1498698.1564502
Möhring RH, Schilling H, Schütz B, Wagner D, Willhalm T (2006) Partitioning graphs to speedup dijkstra's algorithm. ACM J Exp Algorithmics 11(2.8): 1–29
Nazari S, Meybodi MR, Salehigh MA, Taghipour S (2008) An advanced algorithm for finding shortest path in car navigation system. In: Zheng H, Li L, Eguchi K, Wang W (eds) First international conference on intelligent networks and intelligent systems, 2008. ICINIS '08. IEEE Computer Society, Los Alamitos, pp 671–674, doi:10.1109/ICINIS.2008.147. http://dl.acm.org/citation.cfm?id=1471609.1472922
Pfoser D, Efentakis A, Voisard A, Wenk C (2009) Exploiting road network properties in efficient shortest path computation. Tech. rep., International Computer Science Institute. http://www.icsi.berkeley.edu/pubs/techreports/TR-09-007.pdf
Pohl IS (1969) Bi-directional and heuristic search in path problems. Ph.D. thesis, Stanford University, Stanford, CA , USA. AAI7001588
Rodríguez-Puente R (2010) Aplicación de las gramáticas de grafo en sistemas de información geográfica. Revista Cubana de Ciencias Informáticas (RCCI) 4(1/2): 5–10
Rodríguez-Torres A, Rodríguez-Puente R (2010) Servicio de mapas temáticos. Mapping: Revista internacional de ciencias de la tierra (139): 36–39
Sadiq W, Orlowska ME (2000) Analyzing process models using graph reduction techniques. Inf Syst 25(2): 117–134. doi:10.1016/S0306-4379(00)00012-0. http://portal.acm.org/citation.cfm?id=344358.344369
Sanders P, Schultes D (2005) Highway hierarchies hasten exact shortest path queries. Europe 3669(October): 568–579. http://www.springerlink.com/index/5XG8NG9CWXEQYVD9.pdf
Song Q, Wang X (2011) Efficient routing on large road networks using hierarchical communities. Intell Transportation Syst, IEEE Trans 12(1): 132–140. doi:10.1109/TITS.2010.2072503
Sun L, Hu X, Li Y, Lu J, Yang D (2008) A heuristic algorithm and a system for vehicle routing with multiple destinations in embedded equipment In: 7th International Conference on Mobile Business, 2008. ICMB '08. IEEE Computer Society, Washinton, pp 1–8, doi:10.1109/ICMB.2008.47. http://dl.acm.org/citation.cfm?id=1439274.1439769
Wagner D, Willhalm T (2007) Speed-up techniques for shortest-path computations. In: Thomas W, Weil P (eds) 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS). Springer, Aachen, pp 23–36
Wang Z, Che O, Chen L, Lim A (2006) An efficient shortest path computation system for real road networks. In: Ali M, Dapoigny R (eds) Advances in applied artificial intelligence, 19th international conference on industrial, engineering and other applications of applied intelligent systems, IEA/AIE 2006, Lecture notes in computer science. Springer-Verlag, Germany, pp 711–720
Xu L (2005) A decision support model based on gis for vehicle routing. In: Chen J (ed) 2005 International conference on services systems and services management, 2005. Proceedings of ICSSSM '05., vol. 2. IEEE Computer Society, Piscataway, pp 1126–1129. doi:http://doi.ieeecomputersociety.org/10.1109/ICSSSM.2005.1500172
Zeng W, Church RL (2009) Finding shortest paths on real road networks: the case for a*. Int J Geograp Inf Sci 23(4): 531–543. doi:10.1080/13658810801949850

doi:10.1186/2193-1801-2-291
Cite this article as: Rodríguez-Puente and Lazo-Cortés: **Algorithm for shortest path search in Geographic Information Systems by using reduced graphs.** *SpringerPlus* 2013 **2**:291.