STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 2, 2002

MINIMUM COST PATH IN A HUGE GRAPH

ION COZAC

Abstract. Suppose we have a weighted graph $G = (V, E, c)$, where V is the set of vertices, E is the set of arcs, and $c : E \to \mathbf{R}_+$ is the cost function. Determining a minimum cost path between two given nodes of this graph can take $\boldsymbol{O}(m \log n)$ time, where $n = |V|$ and $m = |E|$. If this graph is huge, say $n \approx 700000$ and $m \approx 2000000$, determining a minimum cost path can be a serious time consuming task. So we must develop an algorithm that quickly determines a path having the cost near the optimum one.

Keywords: minimum cost path, huge graph, strongly connected component

1. Introduction

If we develop a route planning application, it is very important to use efficient algorithms that determine a path between two distinct nodes. But what if the application manages a huge graph? This is the case of a complete roads map of a medium country, like Romania. In this case we simply ask to find a path that has the cost near the optimum one, but this path must be found very quickly. A fast algorithm is very important if the application is running on a server, and must satisfy the requests that come from many users by Internet.

To develop the algorithm proposed in this paper, we need some remarks, such as:

(i) each link (arc) can be either a main road or a secondary road;
(ii) the number of main roads (class MR) is very small as compared to the secondary ones (class SR); suppose the cardinality of MR is 8-10% of the cardinality of $MR \cup SR$;
(iii) the number of vertices that belong to a main road (class MV) is very small as compared to the number of vertices that belongs to a secondary road (class SV); suppose the cardinality of MV is 8-10% of the cardinality of $MV \cup SV$;
(iv) the main roads are uniformly scattered among the secondary ones.

2000 *Mathematics Subject Classification.* 05C40.
1998 *CR Categories and Descriptors.* G.2.2 [**Mathematics of Computing**]: Discrete Mathematics – *Graph Theory*.

We need to exploit the following idea (see figure 1). Given two distinct vertices s and t, each of them being of SV type, we first determine a minimum cost path from s to the nearest vertex s_1 that is of MV type. We also determine a minimum cost path from t to the nearest vertex t_1 (reversing!) that is of MV type. Next we determine a minimum cost path from s_1 to t_1, using only the main roads. The solution of the problem is the union of these three paths. This algorithm is very fast because:

- the paths from s to s_1 and from t_1 to t can be quickly determined: see remark (iv);
- the path from s_1 to t_1 can also be quickly determined: see remarks (ii) and (iii).

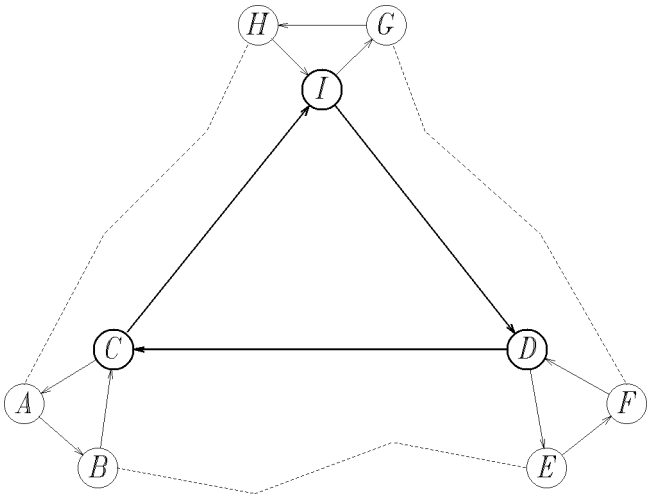

FIGURE 1. Examples of paths determined using the N*earOptimumPath algorithm*. From A to C: A - B - C, from C to D: C - I - D, from B to F: B - C - I - D - E - F

In order to use the algorithm sketched above, we have to prepare two supplementary structures.

We scan the original graph to find all the vertices of MV type. Using these nodes we build a partial subgraph that has only nodes of MV type and arcs of MR type. Let this partial subgraph be $Gs = (Vs, Es)$. We also build the graph $Gi = (V, Ei)$ - the inverse graph of G, where the set Ei is defined as follows:

if $(x, y) \in E$ then $(y, x) \in Ei$, id est, for each arc (x, y) from E we insert the inverse arc (y, x) to Ei.

We describe below the proposed algorithm.

Algorithm *NearOptimumPath*;
Input. The original graph G, the inverse graph Gi, the partial subgraph Gs; two distinct nodes s and t.
Output. Near optimum path from s to t.
begin
* **if** (s is not of MV type) **then**
determine, in graph G, using the algorithm of Dijkstra and selection trees, a minimum cost path D_1 from s to the nearest node s_1 of MV type;
if (t is detected before reaching a node of MV type) **then**
stop: we found the searched path;
else (s is of MV type)
let $s_1 := s$; $D_1 := \emptyset$;
* **if** (t is not of MV type) **then**
determine, in graph Gi, using the algorithm of Dijkstra and selection trees, a minimum cost path D_2 from t to the nearest node t_1 of MV type;
else (**t** is of MV type)
let $t_1 := t$; $D2 := \emptyset$;
* determine, in graph Gs, using the algorithm of Dijkstra and selection trees, a minimum cost path D_3 from s_1 to t_1;
* report the union of these three paths: $D := D_1 \cup$ reverse $(D_2) \cup D_3$;
end (algorithm).

When can we use this algorithm? The following theorem below answer this question.

Theorem 1. *The algorithm* NearOptimumPath *can find a path between any two distinct vertices of the graph G if and only if G and G_s are both strongly connected.*

Proof. These two conditions are obviously sufficient, and the graph G must also be strongly connected. We have to prove that the graph Gs must also be strongly connected. Indeed, suppose that the graph Gs is not strongly connected. It is possible that the algorithm wrongly reports that there is no path between two given nodes, even if such a path exists - the graph G is strongly connected. Let examine the figure 2:

- one can find two distinct nodes s and t, each of them being of MV type, but there is no path from s to t having only arcs of MR type;
- one can find two distinct nodes s and t, at least one being of SV type, and the algorithm find two nodes s_1 and t_1, but there is no path from s_1 to t_1 having only arcs of MR type. □

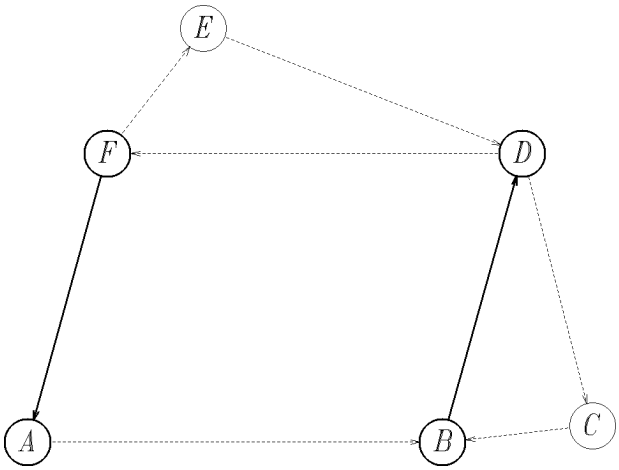

FIGURE 2. The algorithm can not find any path from A to D or from A to E, because the partial subgraph engendered by the arcs (F, A) and (B, D) is not strongly connected

How quickly can we determine a path using this algorithm? The running time of an implementation that uses this algorithm, as compared to the original Dijkstra's algorithm, is proportional to the percentage of the number of the main roads and main vertices.

We saw that this method needs to arrange the information into an organized structure to accelerate searching. This preprocessing phase is necessary because queries will be performed repeatedly on the same graph; these are so called repetitive-mode queries. How much time is needed to arrange the data for searching? To answer this question we present below an algorithm that determines the strongly connected components of a directed graph. This presentation is a review of the algorithm presented in [3].

Algorithm *StronglyConnectedComponents*;
Input. A directed graph $G = (V, E)$.
Output. An array C : the strongly connected components, each vertex being marked with the component number that contains it.
begin
  for (each vertex $x \in V$) **do**
    $Mk[x] := False$;
  $Md := 0$;
  for (each vertex $x \in V$) **do**

```
    if (Mk[x] = False) then
      ScanMark(x);
  Sort D on decreasing order, storing for each mark the associated vertex in X;
  for (each vertex x ∈ V) do
    C[x] := False;
  Build the inverse graph G' corresponding to the graph G;
  nc := 0;
  Warning! The procedure ScanCnx manages the graph G';
    for (each vertex x ∈ X) do
      if (C[x] = 0) then begin
        nc := nc + 1; ScanCnx(x);
        end
end (algorithm).
Procedure ScanMark(vertex x);
begin
  Mk[x] := True;
    for (each vertex y, successor of x) do
    if (Mk[y] = 0) then      ScanMark(y);
  Md := Md + 1; D[x] := Md;
end (procedure).
Procedure ParcCnx(vertex x);
begin
      C[x] := nc;
      for (each vertex y, successor of x) do
    if (C[y] = 0) then
      ScanCnx(y);
end (procedure).
```

The determination of the strongly connected components of a directed graph needs $\mathbf{O}(n \log n + m)$ time, and building the inverse graph Gi and the partial subgraph Gs takes $\mathbf{O}(m)$ time. So we have

Theorem 2. *The supplementary structures used by the algorithm NearOptimumPath are determined in* $\boldsymbol{O}(n \log n + m)$ *preprocessing time.*

We can use the following technique for a parallel architecture. The searching process starts two execution threads: the first thread uses the algorithm *NearOptimumPath*, and the second thread uses the original Dijkstra's algorithm. If one of these two threads finds a path (which may be optimal or not), it must stop the other thread. In this case we don't need to impose a very restrictive condition: it is not necessary that the partial subgraph Gs be strongly connected.

References

[1] Michel Gondran, Michel Minoux, Graphes et algorithmes, Editions Eyrolles, Paris 1979.
[2] Harry R Lewis, Larry Denenberg, Data Structures and Their Algorithms, Harper Collins Publishers, 1991.
[3] Dumitru Dan Burdescu, Analiza complexităţii algoritmilor, Editura Albastră, 1998.
[4] Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest, Introduction to Algorithms, MIT Press, Cambridge, Massachusetts, 1998.

"Petru Maior" University of Tg. Mureş
E-mail address: cozac@uttgm.ro