AZERBAIJAN REPABLIC AKADEMI OF SCIENCES
INSTITUTE OF CYBERNETICS

G.G.ABDULLAYEVA, M.V.MAMEDOVA

DISCRETE GEODESIC PROBLEM IN LOGISTICS MODELLING

(Preprint)

Published pursuant to the decree of Scientific Council of the Azerbaijan National Academy of Sciences Institute of Cybernetics (protocol № 5 dated 30 may 2007)

BAKU - 2007

Review by M.A. Maharramov, Ph. D. (C.P.M.Sc.)

Numerous transportation problems can be reduced to the problem of optimal path selection in graphs and nets. A large number of solution techniques exist that address the classical problem of optimal path selection in graphs and nets – e.g., Dijkstra, Floyd algorithms, etc.

However, problems of finding optimal paths on classical (two-dimensional and multi-dimensional) surfaces have acquired importance recently. The problem of finding optimal paths on surfaces naturally results in a geodesic problem that can be normally tackled using methods of the Calculus of Variations. The latter methods, however, are significantly less efficient computationally than their graph and net counterparts mentioned earlier.

The objective of this work is to develop a method for solving a discrete geodesic problem that, while still based conceptually on the Calculus of Variations, world be comparable in terms of efficiency with algorithms for graphs.

Results and conclusions of this work have a great significance for information technology applications, and this work will be found of interest by researchers in both theoretical and practical aspects of Probability Theory applications technologies.

Рецензент и научный консультант: к.ф.-м.н. Магеррамов М. А.

Многие транспортные задачи сводятся к задаче выбора оптимальных маршрутов в графах и сетях. Существует большое количество известных методов решения задачи выбора оптимального маршрута в графах и сетях в классической постановке (алгоритмы Дейкстры, Флойда и т.д.).

Однако, в последнее время наибольший интерес представляет постановка задачи выбора оптимального маршрута в терминах классических (двумерных или многомерных) поверхностей. Подобная постановка естественным образом сводится к геодезической задаче, обычно решаемой методами вариационного исчисления, которые значительно уступают в вычислительной эффективности выше упомянутым методам оптимальной маршрутизации на графах и сетях.

Целью настоящей работы является разработка метода решения дискретной геодезической задачи, который, основываясь на подходах вариационного исчисления, был бы сравним по эффективности с алгоритмами на графах.

Результаты работы имеют большое значение для различных информационных приложений и представляют интерес для научных работников, занятых как теоретическими, так и практическими аспектами теоретико-вероятностных методов в информатике.

CHAPTER I

INTRODUCTION

Numerous problems of robotics and navigation can be reduced to various forms of geodesic problems, and, most typically, to the search of a path to connect two given points upon a surface of complicated structure or through a spatial domain encumbered with "obstacles". Mathematical formulation of such problems usually consists of constructing geodesic lines for particular metrics, which can be done through the solution of a boundary-value problem for the system of Lagrange Equations. More particularly, if we are to minimise the functional

$$I = \int_0^T L(\tau, q(\tau), \dot{q}(\tau))d\tau$$

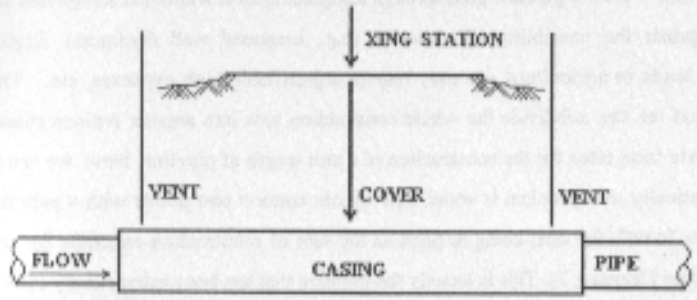
where L denotes a smooth function of three variables (Lagrange function) and $q(t)$ is a smooth vector-function defining a curve with fixed ends, then "optimal paths" -- i.e., curves that minimise the above functional should satisfy the following system of Lagrange Equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) = \frac{\partial L}{\partial q}.$$

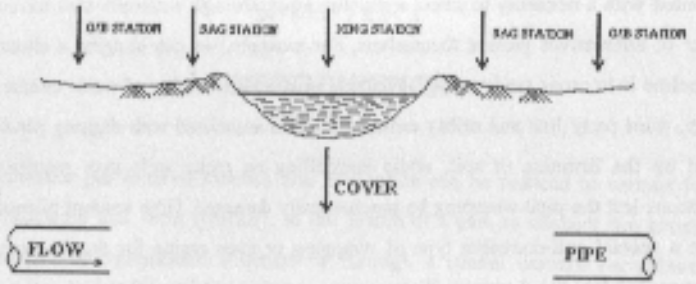
This approach, however, has the disadvantage of requiring a smooth Lagrange function, and bars from consideration some important cases where "penetrability" of various areas is given by a set of discrete values. The simplest of such problems consists in finding the shortest path between two points on the surface of a polyhedron. Polynomial algorithms for the solution of this problem have been recently presented in numerous works, and in this work we will develop new solution techniques for a new kind of geodesic problem, even more valuable from the standpoint of applications, utilising some of the existing algorithms for polyhedrons and graphs.

We will exemplify the usefulness of the proposed algorithms on the basis of applications to Pipeline Engineering and Construction. Pipeline engineers are often

confronted with a necessity to select a pipeline route through a complicated terrain, where a number of alternatives present themselves. For example, we can imagine a situation where the pipeline is to cross sandy, rocky, swampy soils with a number of water course, highway, railway, third party line and utility crossings. Costs associated with digging pipeline trench depend on the firmness of soil, while backfilling on rocky soils may require additional precautions lest the pipe wrapping be mechanically damaged. High content of moisture may require a special anti-corrosive type of wrapping or even casing for the pipeline, meaning extra expenses (see the drawing). Water course crossing requires either laying the pipe under the bottom of the water course, or installation of a metal casing. Railway and highway crossings are also usually installed in casings in order to prevent mechanical damage to the pipe. Proximity to sources of water entails precautions against washout, including sack breakers and off-drains. Installation of water course crossings generally requires placing rip-rap along the banks so that the extremities of the casing will not be exposed.

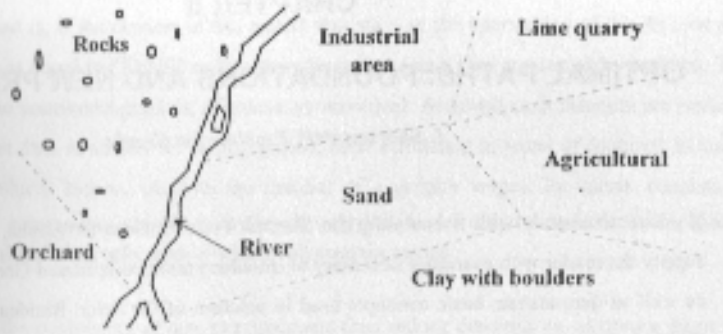


Drawing 1. Cased highway or railway crossing.



Drawing 2. Pipeline under the bottom of a water course

In addition to physical factors directly affecting construction works, ownership and use of lands along the proposed pipeline route affect both the costs and methods of construction -- i. e., if pipeline goes through a populated area, additional safeguards should be taken against the possibility of explosion (e.g., increased wall thickness). Right-of-way through lands in agricultural use may require unjustifiable high expenses, etc.. The above shows that we can subdivide the whole construction area into smaller regions characterised by separate base rates for the construction of a unit length of pipeline. Now, we can say that, mathematically, our problem is about how we can connect two points with a path that has a minimum "cost", the cost being defined as the sum of construction expenses for respective regions (see Drawing 3). This is exactly the problem that has been exhaustively solved in this work.



Drawing 3. Typical Soil Quality and Land Use situation along pipeline Right-of-way

CHAPTER II

OPTIMAL PATHS: FOUNDATIONS AND NEW PROBLEMS

1. Shortest Path Problem for Graphs

We start this study with formulating the Shortest Path Problem for graphs, so that we can supply the reader with examples necessary to introduce more complicated Geodesic Problems as well as demonstrate basic concepts used in solution of the latter. Besides, Graph Theory Problems have a value of their own from the standpoint of applications, and among other things we will demonstrate an application of Graph Theory to an obstacle (barrier) penetration problem that arise in transportation.

By oriented graph with n nodes and m verges we mean a set of ordered pairs $G = \{(i_k, j_k), k = 1, ..., m, i_k, j_k \in \{1, ..., n\}\}$ such that elements of respective pairs are ordinal numbers in the range $1, 2, ...n$. Unoriented graph is a similar set such that the order of elements in pairs does not matter -- i. e., pairs (i, j) and (j, i) denote one and the same verge. Unless otherwise stipulated, all the graphs throughout this study shall be assumed to be unoriented. The pairs are referred to as "verges" and the ordinal numbers $1,2,...,n$, "nodes" of the graph. If we let each of these ordinal numbers denote a point on the plane, and depict verges as straight line segments (directed if the graph is oriented) that connect corresponding nodes, the above purely algebraic definition will take form of a simple geometrical object that is usually associated with graphs. The term "path" as applied to a graph is used to denote a series of the graph's verges such that each verge connects right on to the beginning of its successor, if any, in the series.

We can now define the shortest path between a graph's two nodes as one that consists of the minimal number of verges among those that connect the nodes, for the number of verges is as yet the only optimality factor that we can use for the want of any other criterion.

Prior to proceeding with solution algorithms, let us briefly outline convenient data structures that can electronically represent graphs. The simplest data structure for storing an n-node

graph probably is a square matrix with Boolean components, or 2-dimensional array of zeroes or ones. That is, if the element of this matrix that stays at the intersection of the ith row and jth column is equal to "TRUE" or "1", then the nodes i and j are connected by a verge. The matrix of an unoriented graph is, of course, symmetrical. Although such matrices are perhaps the simplest data structures for storing graphs, their efficiency in terms of required memory can be doubted. Indeed, whatever the number of a graph's verges, its matrix consists of exactly n^2 components, so that even though only few nodes may be connected, the matrix will still hold n^2 data cells most of which will contain zeroes.

To reduce the complexity of data structures and thus reduce demands on computer memory, we can use a different type of structure to store a graph commonly known as "linked lists". Such lists consist of data records of the following form:

```
type ListElement =
        record
                Node: Integer;
                Next: Pointer to ListElement;
        end;
```

where *Node* is a node of the graph, and contents of the field *Next* point to the next element of the list. Each node of the graph has a list associated with it that holds all of the node's immediate neighbours -- that is, the *Node* field of the list's first element contains one of the node's neighbours while contents of the *Next* field point to the second element that holds another neighbour, etc. until al the immediate neighbours have been exhausted. The *Next* field of the last element in the list contains a *nil* pointer. It is now obvious that despite list elements' being data items of a greater size than mere 1-bit Boolean components of the graph's matrix, the use of linked lists may result in a great economy of computer memory if the number of verges is significantly less than n^2 (if nodes do not connect to themselves then the maximum number of verges is $n^2 - n$ for oriented graphs and half that number otherwise.) Note that both matrices and linked lists suit oriented and unoriented graphs equally well except that matrices of unoriented graphs still hold unnecessary $\frac{n(n-1)}{2}$ elements below (or above) the diagonal, and each verge of unoriented graph has to be entered in two linked lists.

While providing sizeable memory reductions, the use of linked lists may, however, increase the overall computing time, for to find out if a given node is directly connected to another one, we may have to look through the whole of the latter's list (which may contain as many as $n-1$ nodes in the worst case) whilst the use of graph matrix lets us do that merely by checking the value of the corresponding matrix component.

Let us now give a short summary of the most important problems and algorithms for simple graphs (i.e., graphs with no "weights" assigned to their verges). The most obvious problem in connection with graphs is searching a path to connect two given nodes. As was previously noted, we may require that the path be optimal -- e.g., that the number of verges forming the path be minimal.

The best-known path-searching algorithms for simple graphs are Breadth-First and Depth-first Searches. The Depth-First Search basically consists in the following: first we define a stack which is to hold up to n graph nodes, and a Boolean array WasInStack[*1..n*] such that its element WasInStack[*i*] indicates whether or not the node i has been previously entered in the stack. Then, on the assumption that we are to find a path connecting two nodes named Source and Destination, we execute the following algorithm:

Algorithm 1. (Depth-First Search)

for any node i WasInStack[i] := FALSE;
STACK <= Source;
WasInStack[Source] := TRUE;

while *STACK* ≠ ∅ do
begin
    t := top of STACK;
    If t = Destination then a path has been found and is held in the STACK, so set a flag
    else   if there exists a node i such that it t and i are connected
                and not WasInStack[i] then
                begin

                place i on top of the STACK;
                WasInStack[i] := TRUE;
            end;
        else remove t from the STACK;
end; {back on your tracks}
if the flag has been set, path was found;
End of the Algorithm.

Correctness of the algorithm can be proved by Mathematical Induction.

❑

Breadth-first search as opposed to Depth-first search places the graph's nodes in a queue rather than in a stack. Whenever a new node is encountered, all of its neighbours that have not been previously placed in the queue are placed there. A node is then picked off the queue's head and all of its neighbours processed in the manner indicated above until the destination has been reached. At this moment, Depth-First Search algorithm would have had a route stored in its stack. In case of Breadth-first Search, a special array Predecessor[1..n] should hold the nodes' predecessors on a path from the Source to Destination. Despite the obvious disadvantage of requiring an extra $O(n)$-sized structure, the Breadth-first Search has an advantage of producing, as a first solution, a path which is shortest in terms of number of verges. In the following algorithm, the notation s<=*QUEUE* means that the node s is picked off the queue's head, and *QUEUE*<=s means that s is added to the queue's tail.

Algorithm 2. (Breadth-First Search)

for i := 1 to n do WasInQueue[i]:=FALSE;
WasInQueue[Source]:=TRUE;
QUEUE:= {Source}; Predecerssor[Source] := Source;

while *QUEUE* ≠ ∅ do
begin
    s <= *QUEUE*;

if s = Destination then

{the shortest path has been found and can be recovered and is stored in Predecessor[1..n]}

else {place at the tail of the QUEUE all the nodes $\{i\}$ directly connected to s and such that WasInQueue[i] is FALSE;

for all such nodes assign Predecessor[i] := s and WasInQueue[i]:=TRUE}

end;

Let us prove that Breadth-first Search generates the shortest path between the Source and Destination. We will use mathematical induction and prove that the algorithm generates the shortest path in a graph that contains $n+1$ nodes on the assumption that it generates shortest paths for any graphs with the number of nodes below or equal to n. If the Destination is connected only to a single node (or is isolated), the statement readily follows from the assumption of Induction. If we suppose the Destination to be connected to more than one node – e.g., m nodes – we have to consider shortest path problem for m Destinations and the same Source in the subgraph derived from the original graph by the deletion of the Destination and adjacent verges. The optimal path from the Source to Destination goes through that one of the above m nodes which is closest to the Source. As we go deeper inside the graph heading for m different nodes, number of verges of the shortest paths from the Source to nodes stored in the QUEUE increases, the increment never exceeding 1. We can also note that a node that is k verges away from the Source quits the QUEUE (and is processed in the algorithm's while loop) before any other node which is more than k verges away from the Source. So, among the m nodes that lead to the original Destination, the one which is closest to the Source shall be encountered first of all and so the shortest path from the Source to Destination shall be found.

We have formulated the Shortest Path Problem in terms of the minimal number of verges. Now we move on to formulate and solve the Shortest Path Problem for graphs with positive "weights" assigned to their verges.

We make one important addition to the previous definition of graph: a positive real number a_i, $i = 1, ..., m$ is assigned to each verge i of the graph, where m is the number of the graph's

verges. To any path l that connects any two nodes of the graph we assign the path's weight which is defined as follows:

$$W(l) = \sum_{k=1}^{s} a_{i_k} \qquad (1)$$

where the path l consists of the verges $\{i_k : k = 1...s\}$. The Shortest Path Problem now formulates as the search for a path that connects any two given nodes and brings the function (1) to its minimum among all the paths connecting the same points.

❑

Square matrix undoubtedly is the most convenient and natural data structure for storing "weighed" graphs. Indeed, if a weighed graph G has n nodes, its weights may be stored in a square matrix with n^2 elements, the element staying at the intersection of the ith row and jth column being equal to the weight of the graph's verge that leads from the ith to jth node, if any, or infinity if no such verge exists. We are going to describe an algorithm that will later provide us with a template for solving Discrete Generalised Preblem.

Algorithm 3. (Dijkstra's Algorithm)

Let s and d be nodes of the above graph G, source and destination of the optimal path, respectively. Upon the termination of the algorithm, an array $D[i] = D_i$, $i = 1, 2, ..., n$ is supposed to hold the weights of optimal paths from nodes $i = 1, 2, ..., n$ to d, and array Successor[i] is to hold the node that follows i on the optimal path from i to d.

1. for j := 1 to n do
   begin D[j]:= a[j,d]; Successor [j] := d;
   end;
   Optimum := {d}

repeat

2. MinDist := ∞. Nearest := *any element of* {1,2,...,n} \ **Optimum**
   for j ∈ {1,2,...,n} \ **Optimum** do if D[j] <= MinDist then
   begin MinDist := D[j]; Nearest := j;

end;

3. **Optimum** := **Optimum** ∪ {Nearest}

4. for j ∈ {1,2,...,n} \ **Optimum** do
   if D[j] > a[j, Nearest] + D[Nearest] then
   begin Successor[j]:= Nearest; D[j] := a[j, Nearest] + D[Nearest];
   end;

5. repeat steps 2-4 until **Optimum** = {1,2,3,...,n} or Nearest = s if you only want the path from s to d.

Here is the main statement of Mathematical Induction in respect of the algorithm: at the end of the *k*-th iteration of the repeat-loop, the set **Optimum** contains the graph's *k* nodes closest to the destination, and contents of the array *D* are such that *D*[*i*] is equal to the weight of a path that has the minimal weight among all the paths from *i* to *d* that go only through nodes contained in the set **Optimum**, apart from the initial node *i*. In other words, if *i* ∈ **Optimum**, then *D*[*i*] is the weight of the optimal path from *i* to *d*, and so after the *n-1* iterations the set will contain all the graph's nodes, and the array -- all the optimal weights. To prove the correctness of Dijkstra's algorithm we only need to prove the validity of the above statement for the *k+1*-th iteration on assumption of its validity for the *k*-th iteration. The statement's correctness after the first step is obvious.

Indeed, a *k+1*-st closest node (there may be a few nodes equally distanced from the destination) is necessarily connected to a closer node or the destination itself. Consequently, the weight of the shortest path from this node to the destination is stored in the corresponding element of the array *D*. If we now pick up the node *i* outside the set **Optimum** such that

$$D[i] = \min_{k \notin Optimum} D[k]$$

this node will be a *k+1*-st closest node -- the one stored in the variable *Nearest*. To modify contents of the array *D* so that the addition of a new element to the set **Optimum** can be reflected, we only need to check if routing through the newly added Nearest point can reduce the weight. So the step of induction has been proved, base and assumption being valid, so the algorithm's correctness has been shown.

Please note that Dijkstra's algorithm is applicable to graphs with positively weighted verges: if, indeed, some of the graph's verges have negative weights, closest among the nodes immediately connected to destination may not be closest to the destination among all the graph's nodes.

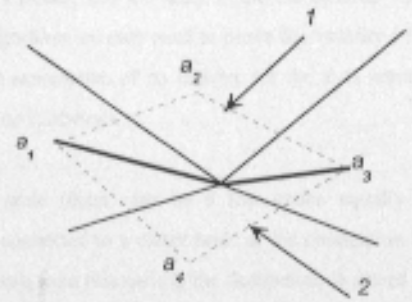
❑

2. A Planar Obstacle (Barrier) Penetration Problem

In the previous section we learned how to optimise paths in graphs that consisted of nodes connected by verges. In many engineering applications, however, a similar problem arises where nodes are separated by linear barriers that are assigned positive weights in same manner as graph verges were in the last section. More formally, we can formulate the problem as follows:

Planar Barrier Penetration Problem

Let P be (an irregular) planar polygon subdivided into a system of n smaller (irregular) polygons $P_s, s = 1, ..., n$, $P = \bigcup P_s$. Let positive real numbers ("weights") be assigned to any segments of straight lines that form the polygons' sides so that common segments of any two adjacent polygons have the same weight. Now, if points a_s belong to the respective polygons' interiors, the weight of a path connecting any two such points is defined as the total weight of all barriers (i.e. segments) intersected by the path. In case the path goes through a point where several segments come together (see the drawings), the path's weight should be deemed equal to the weight of a path derived from the original one by circumventing the said point so that the resulting weight is minimal (the thick line on the drawings represents the original path and the dashed lines show alternatives).



Drawing 1.

The rest of the formulation is not different from what we had in the previous section.❑

In solution of this problem Dijkstra's algorithm comes handiest: we only need to introduce a new graph with nodes a_s such that any two nodes a_i and a_j are connected if and only if they belong to the interiors of two adjacent polygons (that have a common segment[1]). The algorithm for transforming the system of polygons $\{P_s\}$ into a graph will have complexity of $O(m^2)$ where m is the total number of the polygons' nodes.

Later in this work we will present a computer programme that implements this algorithm in application to pipeline route selection. In the context of pipeline construction, the polygons mean homogeneous areas where unforeseeable expenses can be kept to minimum, and barriers (segments) mean either natural or artificial obstacles (rivers, canals and other water courses, ditches, cased or uncased highway and railway crossings, third party pipelines, power transmission lines, etc.)

❑

[1] The polygons' having a common segment is crucial, for we know that any paths that go through points where several segments converge should be mended so as to intersect segments inside.

3. Generalised Discrete Geodesic Problem

Classical formulation of the Geodesic Problem consists in finding the shortest path between two points on the surface of a polyhedron. A solution algorithm for such problem in case of an arbitrary polyhedron was proposed in [DGP] on the basis of Dijkstra's Algorithm. Formulation of the Generalised Discrete Geodesic Problem (GDGP) differs in that the optimal path is sought through a planar domain that consists of a number of areas with varying penetrability. More precisely, the Problem formulates as follows(see [GDGP]):

Generalised Discrete Geodesic Problem. (GDGP)

Let P be a planar polygon subdivided into a system of n smaller polygons $P = \bigcup_{i=1}^{n} P_i$ such that positive real numbers ("weights") $w_1, ..., w_n$ are assigned to the respective polygons. The weight of a rectifiable contour l running inside P is defined as follows:

$$W(l) = \sum_{k=1}^{m} \gamma_k w_{i_k} \qquad (2)$$

where m is the total number of polygons crossed by the curve, γ_k is equal to the length of the curve's intersection with the kth polygon and w_{i_k} is that polygon's weight. The problem now naturally consists in finding a path between two given points inside P such that the weight (2) is brought to minimum.

□

Note that the optimal path is to be selected among a continuum of curves connecting the two points -- and this is a very important feature that makes GDGP different from shortest path problems for graphs. We need to formulate the solution to this problem in terms of discrete mathematics in order to render it executable by a computer. This problem is closely related to the optical problem of calculating the trajectory of light in inhomogeneous environment. We can imagine that the weights w_i represent various refraction coefficients so that the weight (2) is proportionate to the time that light takes to get from the Source to Destination. As we know from Optics, light would always take the quickest route -- i.e., minimise the functional (2). Consequently, we can regard optimal paths of GDGP as trajectories of light within a "mosaic" semi-transparent structure made up of pieces varying in refraction.

Since the weight inside each polygon is constant, we can conclude that the intersections of an optimal path with individual polygons are straight lines. However, the question arises as to how a transition is made from within one rectangle on to an adjacent one, which is answered by

Lemma 1.

Assume that w_1 and w_2 are the weights of two adjacent polygons P_1 and P_2 bordering on a common line L and AOB is a portion of some optimal path which intersects the line L (see Drawing 2). There holds then the following formula:

$$\frac{w_1}{w_2} = \frac{\sin\beta}{\sin\alpha} \qquad (3)$$

where α and β are angles between the segments of the optimal path lying in the respective polygons and the norm to the line L drawn at the point O where the path intersects the line (see Drawing 2 where the directions of angles are indicated).

Drawing 2

Proof.

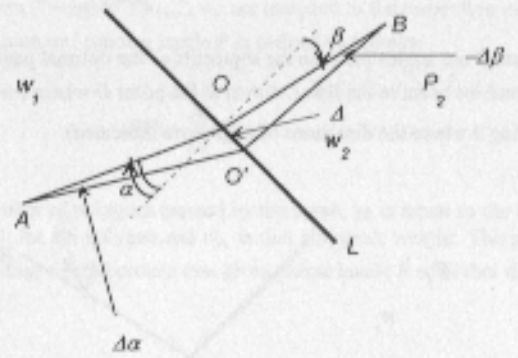
This lemma plays one of the central parts in Geometrical Optics, and can be proved by means of a variational principle. As we know, the derivative of a differentiable function at a point where the function takes its extremum is equal to zero. If we now let the path AOB slightly

variate -- more precisely, let change its point of intersection with the line L to a new point O' -- then the length of the path's segment lying in P_1 will grow by $\Delta\sin\alpha$ (and so will its contribution to the sum (2) by $w_1\Delta\sin\alpha$) and the same for the segment in P_2 will be reduced by $\Delta\sin\beta$ and $-w_2\Delta\sin\beta$, respectively (see Drawing 3 where $\Delta = |OO'|$). To make the derivative of (2) with respect to Δ equal to zero, we need to make it satisfy the following equation:

$$w_1\Delta\sin\alpha = w_2\Delta\sin\beta \quad \Rightarrow \quad \frac{w_1}{w_2} = \frac{\sin\beta}{\sin\alpha}$$

-- the Lemma has been proved.

□



Drawing 3

Now we have all criteria necessary to describe the optimal path locally -- straightness of its intersections with polygons P_i and formula (3) for transition sections -- so we are ready to formulate the global search algorithm.

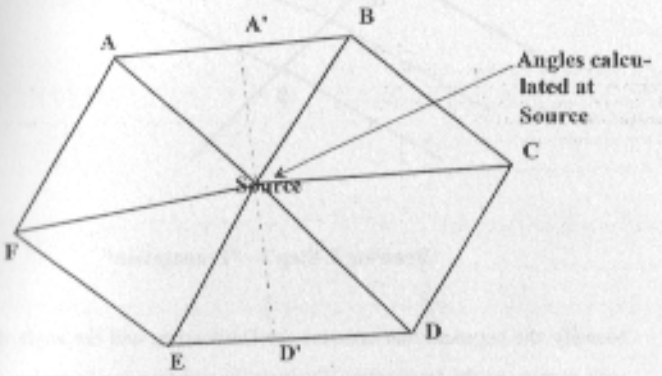
□

4. Solution Algorithm

Solution Algorithm is based on the following principle of the geometrical optics: at the initial moment a signal (circular wave) is emitted from the Source. Once the wave touches upon the boundary of the polygon that contains the Source, part of the wave passes into neighbouring polygon, and from now on is supposed to be detached from the rest of the wave. Waves are propagating in polygons at speeds equal to the inverse values of the corresponding weights -- i.e., the greater is a polygon's weight, the slower is the wave's propagation in that polygon. Subsequently this process is applied to all new "broken" waves until the whole area has been covered. Mathematically, we have the following algorithm.

1. Step 1. Create an empty list of verges (segments) $l=(a, b)$, pairs of angles (α, β), and distances D.

2. Add to the above list all the verges of the polygon containing the Source with angles being calculated at the source with respect to a prefixed direction and the array of distances holding the minimal distances between the verges and the Source. Any verge such that the said minimal distance is not attained at one of the verge's extreme points is split into two (see Drawing 4).

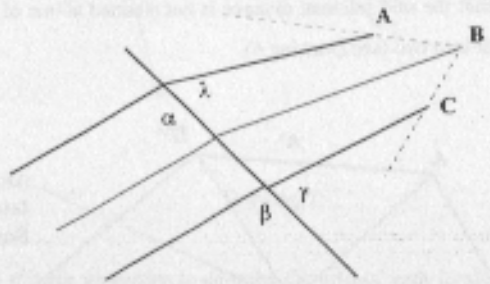


Drawing 4. Step 2 of the Algorithm (angles are calculated with respect to a prefixed direction).

3. An element (segment) is picked out of the list whose distance D to the Source is minimal. Given the values of angles α and β at which the segment is intersected and using the formula (3) we find the "projection" of the selected segment onto the opposite sides of the corresponding polygon (this time - two segments [A, B], but may be any number of segments). Thus obtained segment(s) are then added to the list, along with the corresponding angles[2] and distances. After the new segment(s) have been added, we apply the same operation to them in respect of identifying the minimal distance (see Step 2).

4. If the newly added segments intersect with other elements of the List, discard the intersection, leaving the segments which are closest to the Source. If necessary, make the minimal distance from the Source attainable at the extremeties of segments by dividing them into smaller ones, as described in Step 2..

5. Repeat Steps 3 and 4 until all the verges have been covered twice.



Drawing 5. Step 3 - "Propagation"

6. Identify the segment that contains the Destination, and the angle that generates the path coming to the Destination. Evaluate the weight using formulae (2) and (3).

[2] We keep angles at which the extremeties of these segmennts are viewed from the Source, rather than the angles of intersection with any verges.

Correctness of the proposed algorithm easily follows from the optical and geometrical interpretation of optimal paths. Since the algorithm just emulates the propagation of light through a "mosaic" of glasses having various refraction coefficients, the found optimal path is of necessity the optimum, for it coincides with the trajectory of light which minimises, as is well known, functional (2).

❑

Logistika modelləşdirilməsində diskret geodeziya məsələsi

Xülasə

Bir çox nəqliyyat məsələləri qraflar və şəbəkələrdə optimal marşrutların seçilməsinə gətirib çıxarılır. Klassik qoyuluşda qraflar və şəbəkələrdə optimal marşrutların seçilməsi məsələsinin həlli üçün böyük miqdarda məlum metodlar vardır (Deykstr, Floyd və digər alqoritmlər).

Lakin son zamanlar klassik səthlər (ikiölçülü və ya çoxölçülü) terminlərində optimal marşrutun seçilməsi məsələsinin qoyuluşu böyük maraq doğurur. Məsələnin bu çür qoyuluşu təbii olaraq, adətən variasiya hesabı metodları ilə həll olunan geodeziya məsələsinə gətirib çıxarır. Bu variasiya metodları öz hesablama effektliyinə görə yuxarıda adları göstərilən qraf və şəbəkələrdə optimal marşrutun seçilməsi metodlarından geri qalır.

İşin məqsədi: diskret geodeziya məsələsinin həll metodunun seçilməsidir. Metod variasiya hesabı yanaşmalarına əsaslanaraq öz effektivliyinə görə qraflar üzərinə alqoritmlərlə müqayisə oluna bilər.

Дискретная геодезическая задача в моделировании логистики

Аннотация

Геодезические задачи, как правило, решаются методами вариационного исчисления, которые уступают в вычислительной эффективности методам оптимальной маршрутизации на графах и сетях.

Целью настоящей работы является разработка метода решения дискретной геодезической задачи, который, основываясь на подходах вариационного вычисления, сравним по эффективности с алгоритмами на графах.

Подписано к печати 13.05.2007
Заказ 188 Тираж 100
Участок подготовки информационных материалов Института Кибернетики НАН Азербайджана. Баку, ул. Ф.Агаева, 9.