Minimizing Multi-Hop Wireless Routing State under Application-Based Accuracy Constraints

Mustafa Omer Kilavuz and Murat Yuksel
Computer Science and Engineering Department
University of Nevada - Reno, 1664 N. Virginia Street, Reno, NV 89557
mkilavuz@cse.unr.edu, yuksem@cse.unr.edu

Abstract

Provisioning of rich routing building blocks to mobile ad-hoc networking applications has been of high interest. Several MANET applications need flexibility in describing paths their traffic will follow. To accommodate this need, previous work has proposed several viable routing schemes such as Dynamic Source Routing (DSR) and Trajectory-Based Routing (TBR). However, tradeoffs involved in the interaction of these routing schemes and the application-specific requirements have not been explored. Especially, techniques to help the application to do the right routing choices are much needed. We consider techniques that minimize routing protocol state costs under application-based constraints. We study the constraint of "accuracy" of the application's desired route, as this constraint provides a range of choices to the applications. As a crucial part of this optimization framework, we investigate the tradeoff between the packet header size and the network state. We, then, apply our framework to the case of TBR with application-based accuracy constraints in obeying a given trajectory. We begin with simple discrete models to clarify the tradeoff between packet header size and network state. We show that the problem of accurate representation of a trajectory with the objective of minimizing the cost incurred due to header size and network state is difficult to solve optimally. Finally, we develop heuristics solving this problem and illustrate their performance.

1. Introduction

As the reach of networked devices increases, the network infrastructure needs to support *application-specific* designs due to the increased variety of reached applications. This need emphasized in sensor networks, especially in routing functions provided by the network. To accommodate various application-specific routing needs, the *expressiveness of the routing interface* must be at sufficient level. The typical wireless routing interface has been a shortest-path interface with simplistic primitives: `Send(src,dst,data)` and `Receive(src,dst,data)`. Recently, there have been a lot of efforts in improving this interface with an "options" argument in the primitives, i.e., `Send(src,dst,data,options)` and `Receive(src,dst,data,options)`. Recent work [2, 6] tackled this problem in the general routing context without customizing it for multi-hop wireless routing.

In terms of application-specific routing functions, previous work showed that very flexible routing functions can be implemented [7, 32] by using network-specific properties such as geographic routing [13] capability. Such routing functions enabled application-specific traffic engineering [15], e.g., load-balancing among multiple paths instead of a single shortest-path. On the other side, over a network where several routing options are provided, applications face the problem of selecting the right options and appropriately identifying its constraints, such that applications' goals are met. Some recent work pinpointed the complexity of this issue within limited contexts, e.g., mimimal/maximal exposure path selection [9].

Though provisioning of rich routing building blocks to multi-hop wireless networking applications is of high interest, application-specific requirements and constraints emphasize the challenge of designing routing schemes. It is a major challenge to include application-based "constraints" (which can be of various type such as path quality, path accuracy, and path cost/price) as an additional argument to the routing primitives, which we investigate in this paper. In particular, we investigate tradeoffs involved in the interaction between wireless routing and the application-specific constraints. Especially, low-complexity techniques to help the application to do the right routing choices are much needed, as the time to make such routing decisions is very minimal for mobile nodes [27]. We outline an optimization framework where various inter-layer design problems can be formulated, e.g., minimization of routing state un-

1-4244-2575-4/08/$20.00 ©2008 IEEE
This work was supported in part by NSF under award 0627039.

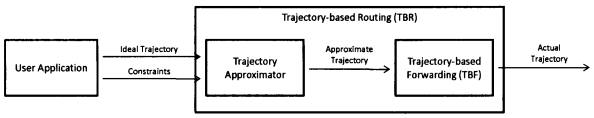


Figure 1. Trajectory-Based Routing (TBR) framework.

der application-specific path constraints. Our optimization framework opens the doors for finding the right tradeoffs in function placement among the application and the routing layers, such that the overall cost of two conflicting goals is minimized: (i) accommodating application's end-to-end requirements with maximum quality and (ii) performing routing with minimum state and resource costs.

We base our study to the key optimization problem of minimizing routing state costs under application-specific constraints. Specifically, we formulate the problem of minimum cost (i.e., the state to be stored) trajectory-based routing (TBR) [7, 32] under application-specific constraint of "path accuracy". That is, we consider an application which desires its packets to follow a trajectory with a bounded error in obeying the trajectory. We illustrate how such a constraint can be quantified and be used in our optimization framework. In TBR, as shown in Figure 1, the application at the source node embeds a desired "ideal trajectory" into packets' headers. This ideal trajectory is to be followed by the packets. The intermediate nodes are assumed to be able to decode the ideal trajectory from the packet header and decide which neighbor to forward the packet next such that the packet obeys the ideal trajectory as much as possible. Though TBR is a good routing option, the decision to select the next neighbor optimally can be time consuming which breaks the basic premise of simple packet forwarding. Thus, approximating the ideal trajectory is needed so that the nodes can work on the "approximate trajectory" rather than the ideal trajectory which can be quite complicated depending on the application needs. The approximate trajectories can be a concatenation of several pieces of "easy to handle" trajectories such as a line, a third order polynomial curve, or a Bézier curve. Then, this approximate trajectory is used to make forwarding decisions for the "actual trajectory/route", as illustrated in Figure 2.

We formulate the *trajectory approximation* problem as a combinatorial optimization problem, and show that it is NP-hard. In our formulation, we allow a number of representations to be used to approximate a portion of the trajectory, such as straight lines, polynomial curves, or Bézier curves. We carry the trajectory approximation problem on a discrete space and outline an exhaustive search algorithm (with pruning) that guarantees finding the global optimum. Since this trajectory approximation is supposed to be frequently performed (i.e. whenever there is data to be sent with TBR) at the source node where the application resides, the computational complexity of the solution is of high importance.

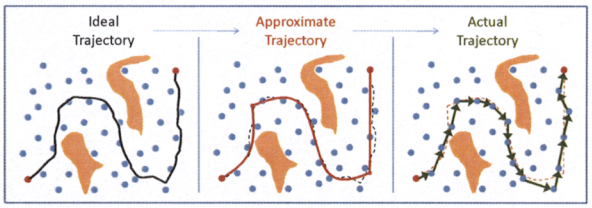


Figure 2. Ideal, approximate, and actual trajectories.

Thus, to reduce the computational time requirements of our solution, we map the problem to a genetic algorithm and design two heuristics as well.

In brief, contributions of this paper include:

- *An optimization framework to minimize routing state under application-based constraints.* We applied this framework on the particular problem of minimizing TBR state under application-based accuracy constraints.
- *Formulation of the trajectory approximation problem.* We formulated the problem of generating an approximate trajectory within an application-defined error bound such that TBR state (i.e., the aggregate of packet header length and network/router state) is minimized.
- *Proof that the trajectory approximation problem is NP-hard.*
- *Solutions to solve the trajectory approximation.* We designed an exhaustive search algorithm, a genetic algorithm, and two heuristics to solve the trajectory approximation problem.
- *Customized the trajectory approximation problem for power-scarce networks.* We increased the perceived cost of packet header length and illustrated how the approximate trajectories should be changed in a power-scarce network.

The rest of the paper is organized as follows: First in Section 3, we describe the TBR in detail. We outline the problem of approximating an ideal trajectory under application-specific accuracy constraint in Section 4. Then, Section 5 details four methods to solve the approximation problem. We present results of our simulation experiments in Section 6 and summarize our work in Section 7.

2. Related Work

There has been a lot of recent interest mobile ad-hoc networks (MANETs) [11, 20, 24, 3, 13, 14], and sensor networks. Main focus of MANET research has been on scalability and complexity issues raised due to underlying dynamism, specifically on routing. Routing tackles the problem of establishing an *indirection* from a *persistent name (or ID)* to a *locator*. In today's routing, this indirection

translates to IP address prefix being the ID and next hop router being the locator. Routing protocols such as AODV [20], DSDV [21], and others, facilitate this indirection by building and maintaining routing tables that map *destination IDs* to *next-hop IDs* based on node or link state information. Dealing with such dynamic indirections usually involves *delaying* the creation of the ID-to-location mapping (a.k.a *"late-binding"*), and reducing the *number and dynamism* of bindings (e.g., using hierarchy [24, 19] or consistent hashing [25, 14] or filtering [23]). MANET routing has grown into two subclasses: *proactive* routing [21] (using early binding) and *reactive or on-demand* routing [11, 20] (involving late-binding).

Application-based routing and network design, the most relevant to our work, has been of a key focus area in sensor networks research. Providing routing expressiveness and flexibility to user applications has been of a great interest [8]. In the general context of routing, defining application-specific custom routes through user-defined routes [29], through a declarative language [6, 26], or through concatenations of several user-chosen contracts [31] have attracted interest. However, these efforts either require significant router computation or autonomous system level route descriptions, both are impractical in wireless routing.

The very first application-based wireless routing work was an extreme scheme of Dynamic Source Routing (DSR) [11], which gives a full flexibility to the user application to define the routes. Since DSR could not scale to sufficient number of nodes, schemes like Trajectory-Based Routing (TBR) [7, 30] and landmark selection [18] were proposed to reduce the packet header costs and yet still give reasonable flexibility for users to define their desired routes as trajectories. Recent wireless routing studies focused on customizing routing metric for applications [16] and centralized optimization of routing for application-based goals [1, 4]. Our work introduces a new knob of "application-specific constraints" into the routing interface, while considering routing state scalability issues.

3. Trajectory-based Routing and Accuracy Constraint

Trajectory-based Routing (TBR) [7, 32] suggests that the source node encodes the trajectory into packet headers, and then intermediate nodes forward the packets according to the trajectory decoded from their headers so as to make them traverse the source-defined trajectory as much as possible. The "ideal trajectory" is received from the user application which is the demanded path of the traffic flow. In Figure 2, the ideal trajectory is shown as a hand-drawn curve. This ideal trajectory can be formed based upon the user application's goals, but the network routing still has to determine the best way of attaining the goal of forwarding the packets along this ideal trajectory.

Since the trajectory will be encoded into the packet headers, the ideal trajectory needs to be represented by formulations which can be understood by all nodes in the network. Implementing very complex trajectory decoding hardware at every node is not practical, and thus these trajectory representations must be of simple type such as a straight line, a polynomial curve, or a Bézier curve. This means that the ideal trajectory may not be representable exactly, depending on how complex the ideal trajectory is. Thus, instead of representing the exact ideal trajectory, we focus on generating an "approximate trajectory" which is easier to represent, however it has slight differences than the ideal trajectory. To give the user application a knob on how the routing function generates this trajectory, we consider the limits of the difference between the ideal trajectory and the approximate trajectory is given by the user application as an "accuracy constraint".

Although the approximate trajectory is optimized in the best way under the constraints, still sometimes it is too complex to encode the whole trajectory into the packet headers. Hence, the approximate trajectory is divided into small pieces that each piece can be encoded into packet headers separately. To manage this, some Special Intermediate Nodes (SINs) [32] are selected around the points where the approximate trajectory is divided. The SINs store the information of the trajectory piece starting from that point only. The packets departing from the source receives only the information about the first piece. When they reach the end of this piece, they acquire the next piece's routing information from the corresponding SIN. They keep changing the routing information in their headers whenever they visit another SIN until they reach the destination. For example, in Figure 3, the approximate trajectory is divided into two pieces, the SIN is located at the point which we call the *split point*. The routing information stored in the packet headers and the routing tables in the SINs are "costs" of maintaining this TBR session. This cost can be translated into other dimensions such as power consumption of the TBR session. We call the cost caused by the routing information (source, destination, next hop, trajectory representation etc.) stored in the packet headers as *packet header cost* and by the routing information stored in the SINs as *network state cost*.

Various Trajectory-based Forwarding (TBF) techniques were proposed [7, 32] to manage the packet traffic over the approximate trajectory. Depending on the density and placement of the nodes, the packets may not follow the approximate trajectory exactly. Thus , the "actual trajectory" the packets follow may be different than the approximate trajectory as shown by arrows in Figure 2. In this paper, we do not focus on the actual trajectory the packets take, but rather focus on generating the best possible approximation of the ideal trajectory such that the actual trajectory

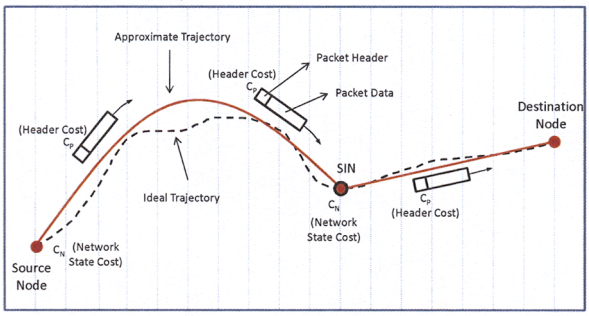


Figure 3. Packet forwarding in TBR.

will have the highest likelihood of being close to the ideal trajectory with a minimal routing state cost.

4. Trajectory Approximation Problem

4.1. System Model

When a trajectory is approximated by a series of representations, each piece of the trajectory is associated with some cost in terms of packet header length and network state. We construct the problem of minimizing this aggregate state cost of the whole trajectory while satisfying the application-defined accuracy constraint. We define the constraint as the error by which the approximate trajectory can deviate from the ideal trajectory. Each piece of the approximate trajectory has an error value that it contributes to the total error of the whole approximate trajectory. The accuracy constraint plays the role of restricting the total error of the whole approximate trajectory. We make the following modeling assumptions for the problem:

- There are k choices for representing a given piece of the trajectory. These are denoted as $r_1, r_2, ..., r_k$. Here r_i is a straight line, a polynomial curve, or a Bézier curve, etc.
- The space in which the trajectory exists is discretized, and there are a maximum of m points (excluding source and destination), or $m+1$ pieces into which the trajectory can be split into.
- Since each piece of the trajectory is represented using a single type of representation, we construct a matrix Q of dimensions (m, k), where Q_{ij} is a binary variable, which denotes which representation is used for a particular piece of the trajectory. This implies that at most a single entry in a row is equal to 1.
- If the algorithm selects a particular representation r_i for a piece of the trajectory, then it makes use of a subroutine to compute the error associated with the representation. This error is calculated such that r_i is fit to its corresponding part of the ideal trajectory with the least error possible by using algebraic equation solving methods [28].
- There is a packet header cost C_P and a network state cost C_N for each piece of the trajectory, depending on the representation used for the piece of the trajectory being considered.

We can now formulate the following binary program:

$$\min \sum_{i=1}^{m}\sum_{j=1}^{k} C_P(j)Q_{ij} + C_N(j)Q_{ij} \quad (1)$$

$$\text{Subject to:} \quad (2)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{k} e(Q_{ij}) \leq E \quad (3)$$

$$\sum_{j=1}^{k} Q_{ij} \leq 1 \;\; \forall\, i = 1 \cdots m \quad (4)$$

$$Q_{ij} \in \{0, 1\} \quad (5)$$

In the formulation (1-5), the constraints (3) denote the error associated with the representation of each piece of the trajectory. Note that, of the m points available, we do not necessarily select all of them, which is captured in constraints (4). In constraints (3), the sum of the errors must not exceed an application-defined error E which we assume is an input to the problem. Constraint (5) states that Q_{ij} is a binary variable.

NP-hardness: The formulation above can be modeled using a graph on $m+2$ vertices (including source and destination nodes), with edges between all nodes (complete graph on $m+2$ vertices) implying that any of these edges can be chosen in an approximation of the trajectory, subject to the error constraint. Next, we allow multiple edges between two vertices, each edge corresponding to one type of approximation for the corresponding portion of the trajectory. The edges are associated with an error measure as well as a cost due to C_p and C_N. While the formulation in $(1 \cdots 5)$ is *node* based, the edge formulation is implicit. For example a solution which selects some $l \leq k$ equal to 1 for some node i, and all other entries in the matrix as 0, implies that the approximation are the edges (s, i) and (i, t) where s and t are source and destination nodes respectively.

By considering the errors associated with edges as "weights" and the state costs as "costs", we note that the problem of finding the approximate trajectory giving minimum state cost (while satisfying the accuracy constraint) is identical to the *shortest weight-constrained path*, which is a well known NP-Complete problem [22]. This is represented diagrammatically in the Figure 4. In Figure 4, each edge is associated with a cost as well as a representation error. This is represented on the edges (S,1) as (C_{S1}, E_{S1}) and (1,2) as (C_{12}, E_{12}) , though all edges are associated with such numbers.

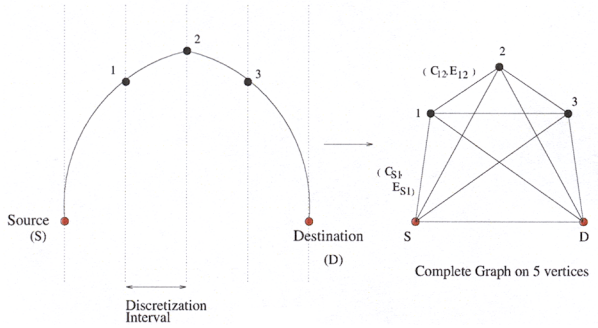


Figure 4. A sample graph formulation of the discretized trajectory approximation problem.

4.2. Cost Calculations

The costs C_P and C_N in the objective (1) represent the cost incurred for selecting an approximation for a given portion of the trajectory. We devised a realistic method to calculate these costs in terms of actual bytes. For each representation we calculate how many bytes we have to store additionally in the packet header (i.e, C_P) and in the intermediate nodes (i.e., C_N). In the rest of the paper, we consider three different representations. The more complex the representations are, the more costly they will be. Below are the three representations we use:

- *Line,* $y = ax+b$*:* Two end points (x_1, y_1) and (x_2, y_2) are enough to express a line. We assume that each parameter is a double. Considering each double takes 8 bytes, the space needed to express a line is 4×8=32 bytes.
- 2^{nd} *degree polynomial curve,* $y = ax^2+bx+c$*:* For a 2^{nd} degree curve, we have to store x_1, x_2, a, b, and c. We do not need y_1 and y_2 since they can be calculated by putting x_1 and x_2 into the equation. Thus, the space needed for a 2^{nd} degree polynomial curve is 5×8=40 bytes.
- 3^{rd} *degree polynomial curve,* $y = ax^3+bx^2+cx+d$*:* Similarly, for a 3^{rd} degree curve, we have to store x_1, x_2, a, b, c and d. Again, we do not need to store y_1 and y_2. So, the space needed is 6×8=48 bytes.

For each segment we calculate the total cost as the sum of C_P and C_N, both of which are dependent to the cost of the representation used. Let the representation cost of r_j be R_j, i.e., 32 or 40 or 48 bytes. Then, we can rewrite the aggregate cost as $c_pR_j + c_NR_j$, where c_P and c_N are the weight constants of the packet header and the network state costs respectively. Depending on the application and the context in which TBR session is taking place, these constants *may* depend on the lengths of the trajectories, the average traffic flow over the network, number of connections etc. Unless otherwise said, we assume that $c_P = c_N = 1$. We will also show how different weights effect the structure of the approximate trajectory.

4.3. Error Measures for Trajectory Approximation

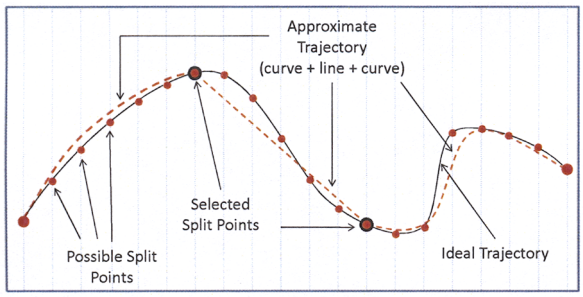
In order to let the user application to express its accuracy constraint as well as to determine the quality of our trajectory approximation, we define a way of quantifying the trajectory approximation error. We assess the approximate trajectory, by means of an aggregate error, which is the sum of the errors in representation of each piece. We define the error in terms of the *deviation area*, i.e., the total area between the ideal trajectory and the approximate trajectory. To make it more generic we defined the error in percentage instead of any unit of area. Of course, for that, we have to define what "100% error" is, which we think that should be something intuitive. For that, we draw projections of the ideal trajectory above and below it by leaving a constant distance in between. We, then, define "100% error" as the area between the two projections. We defined the distance between the ideal trajectory and a projection as 50 pixels in our experiments. So, 100% error is $(50 + 50) * width$ unit area. This distance can be different according to the size of the space and might be defined as a percentage of the height of it. For instance, 1/4 or 1/8 of the height of the space.

5. Methods to Solve the Problem

As the trajectory approximation is an NP-hard problem, fast solutions are needed. In practice, this problem will be solved pretty frequently by the nodes where applications reside. In general, before data transmission starts in an end-to-end session using TBR, this trajectory approximation problem must be solved. We now outline four techniques to perform trajectory approximation: An exhaustive search technique guaranteeing the best solution, a genetic algorithm providing very good solutions in shorter running times, and two heuristics providing solutions in very short running times.

5.1. Exhaustive Search with Pruning

This algorithm tries all possible combinations of split points, representations, and chooses the best possible solution yielding the minimum routing state cost within the accuracy constraint. With sufficient resolution in the discrete space, this algorithm is guaranteed to find the optimum solution. First we select the possible split points on the trajectory. The distance between any consecutive split points must be the same on the x-axis and should be small enough to make it possible to find the best solution. However, the number of these points should be small enough to have a reasonably short running time. The exponential



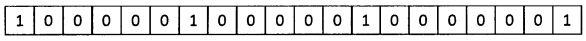


Figure 5. Split points, trajectories, and selection bits.

growth of the running time prevents us to have too many of these points.

The algorithm selects a subset of all possible split points, where the selection is denoted by a binary value (1: selected, 0: ignored) for each possible split point. A representation that is fit between two consecutive split points forms a segment of the approximate trajectory. For every segment, a representation with its own error and cost values will be chosen. The approximate trajectory formed by these representations which is below the tolerated error bound and has the minimum aggregate cost will be chosen as the best solution.

An example is shown in Figure 5. 2 of 19 possible split points are selected. Including the source and destination points we have 4 points which are the end points of 3 segments that we will represent by one of the 3 representations, i.e. line, or a second or third degree curve. Here, a 2^{nd} degree curve, a line and a 3^{rd} degree curve forms the approximate trajectory.

We applied pruning [17] method to reduce the running time by pruning non-optimal solutions whenever possible. Specifically, we begin from the source point and go up to the destination point by giving 0 or 1 to each bit. When we give 1, we branch our searching tree for every type of representation. When we put a representation, we calculate the total error and the total cost so far (including the preceding segments). If we have already exceeded the error bound or the cost of the current best solution (if there is any), we stop looking for a solution having the current bit sequence. Because it is obvious that whatever value the rest of the bits get, we will not be able to find a good solution.

5.2. Genetic Algorithm

Genetic algorithms (GAs) are search algorithms based on the mechanics of natural selection and genetics [10]. We have a population of members that each member has a chromosome which stores an approximate trajectory with the representations to be used and a fitness value showing the

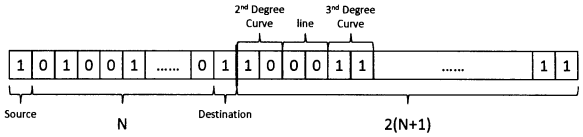


Figure 6. A sample chromosome in GA solution to the trajectory approximation problem.

quality of the member. In every step, a new generation is generated by coupling the current members. After some time, the best member is selected as the best solution.

Assume we have N possible split points, as shown in Figure 6, each of which can be selected for approximating the trajectory. Similar to the exhaustive search approach, the first bit is the source point's followed by N bits for N possible split points, and another bit follows them for the destination point. The bits for source and destination are always 1. We included them in the chromosome for the ease of computation and implementation purposes. The remaining part of the chromosome has 2 bits for each of the possible split points and the source point, which makes total of $2(N+1)$ bits. These 2 bits define which representation will be used between the consecutive split points. 00 and 01 represent a line, while 10 represents a 2^{nd} degree curve and 11 represents 3^{rd} degree curve. If there were more than four possible representations, then we would need more than 2 bits for each point. For example, in Figure 6, the first segment of the approximate trajectory is a 2^{nd} degree curve between the source point and the first split point. A 3^{rd} degree curve follows it between the first and the second split points. Line is omitted.

The initial generation is filled with members who have a chromosome of $3N+4$ random binary numbers. The members of the next generation are generated as follows: We choose 2 parents with roulette selection [10], i.e., the probability of the members to be chosen is proportional to their fitness values which are the aggregate cost of that member + handicap for the members exceeding the error bound. This way, better members are more likely to be chosen for crossover. We apply single point crossover and obtain two child members, and the best two among the parents and children are selected for the next generation. We keep doing this until the next generation is fully generated. Furthermore, we applied one bit mutation after the crossover to increase diversity.

The feasible members in the population are the ones which satisfy the error bound. There are also unfeasible members which exceeds. These have a handicap in their fitness values which makes them less likely to be chosen for the next generation. We keep them in the population because they might lead to a very good solution with only very little changes. Besides, in some of the generations (especially the initial ones) there might not even be any feasible

solutions. When the stopping condition is satisfied, the best feasible member having the least cost is selected as the best solution.

5.3. Greedy Heuristic 1: Equal Error

This algorithm is the fastest and the blindest one. First, we find a representation for the whole trajectory. If the trajectory is above the error bound, we divide the trajectory into two or more pieces, where the number of pieces might be decided according to the error of that segment. For example, a segment with a very high error might be divided into more than 2 pieces at once. Every piece is equal in size on the x-axis. Then, we find the best representations to fit on each segment. We keep dividing the segments recursively that are over the error bound which is the same for all segments in percentage.

5.4. Greedy Heuristic 2: Longest Representation

This algorithm places the segments of the approximate trajectory one by one. The main idea is to choose the longest possible representation below the error bound for the next segment until we reach the end. Starting from the shortest interval (in terms of number of split points) for the next segment, the interval is increased every time until we cannot find a suitable fit anymore. The last suitable one is chosen for that segment. Then, we try to find a representation for the segment starting from the end point of the last segment we have decided.

For example, for the first segment we initialize the interval to [0,1]. As long as we find a representation to fit into that interval we increase it one by one, i.e., [0,2], [0,3] and [0,4]. Let's say we could not find a suitable representation for the interval [0,89] and the last one that we could find was a 3^{rd} degree curve for the interval [0,88]. This means we choose 3^{rd} degree curve for the interval [0,88] as the first segment and we begin to search for the next segment starting from 88. This time, the intervals will be [88,89], [88,90], [88,91] and so on. The step size of 1 for this example can be larger. This way the algorithm will run faster, but since the resolution will be reduced the algorithm will less likely to find better solutions.

6. Performance Evaluation

We performed performance evaluation of the four algorithms from Section 5 by applying them on several trajectories with varying complexity. The goal of our experiments is twofold:

- *Algorithm performance comparison:* Observe performance of our heuristics and the genetic algorithm in terms of quality of their trajectory approximation and their running time.
- *Customization to a power-scarce network:* Illustrate that our optimization framework can be customized for a power-scarce network by changing the weight of the packet header cost in the aggregate routing state cost.

6.1 Experimental Setup

We used a 400×400 pixel area in our experiments. The source node is at a random place on the y-axis while the destination node is at a random place on the $x = 400$ line. The trajectory between the source and destination nodes made up of small line segments which has a *complexity* parameter in [0,180]. The complexity parameter defines the maximum value in "degrees", the maximum angle between each pair of consecutive line segments can get. The higher this parameter gets the more complex (zigzag) the trajectory will be. If this parameter is set to 0 then the angle between the line segments will be 0, which will produce a straight line.

In the experiments we have tried different error bound and trajectory complexity parameters. We used 4 different error bound values (i.e., 5, 10, 25 and 50) and 18 different trajectory complexity values (i.e., from 10 to 180 increasing with a step of 10). For each error bound and trajectory complexity pair, we produced 32 different trajectories with random seeds and applied the four trajectory approximation algorithms on each of them. We ran the genetic algorithm 32 times on each trajectory and report the average result of those 32 realizations. The other three algorithms do not need this since they do not have a random factor. We now present the rest of our experimental setup specific to each of the four algorithms.

6.1.1 Exhaustive Search

In the exhaustive search, we set the number of possible split points to 19. So, the maximum number of segments that the trajectory can have is 20, which means that the shortest segment can be 20 pixels wide. We could not increase the number of split points because of the time limitation. Even with this set up it sometimes took many hours just for one run.

6.1.2 Genetic Algorithm (GA)

We used the following parameters to tune our GA implementation:

- Population size: 300
- Crossover probability: 0.99
- Mutation probability: 1

These are values that we found out after running many experiments and believe are the near optimum for our problem. The mutation probability we found to be the best is surprisingly high, which yields more diverse solutions faster

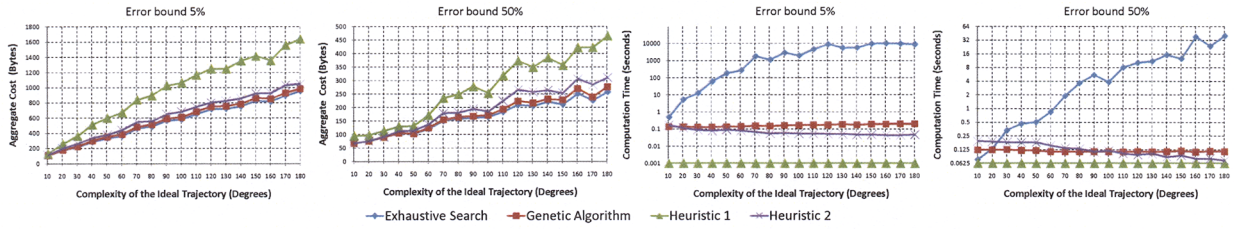


Figure 7. Results of trajectory approximation. Left two: Aggregate routing state cost vs. Trajectory complexity. Right two: Calculation time vs. Trajectory complexity.

(i.e. better exploration of the search space). To make our GA implementation comparable with the exhaustive search method, we used 19 possible split points. However, as will be seen in the results, it is possible to use a lot more splitting points (e.g., 200) and get good solutions in a very short time unlike the exhaustive search.

We have observed the GA results for different stopping conditions. We ran the experiments for up to 20, 50, 100, 200, 300, 1000, and 10000 generations. In most of the cases the GA finds the best solution before 50 generations. After 100 generations we observed that almost nothing changes. So, we decided to define the stopping condition in a way that it will stop if a solution better than the current best is not generated for the last 100 generations.

6.1.3 Greedy Heuristic 1: Equal Error

We divided the segments into 2 pieces every time and used the representation which gives the smallest error for the corresponding piece of the trajectory.

6.1.4 Greedy Heuristic 2: Longest Representation

We set the step size is set to 1 to get the best result. Since, running time is not that much of an issue for this heuristic, we kept the step size small. In larger areas we might need to increase the step size.

6.2. Results & Discussion

6.2.1 Comparison of Exhaustive Search, GA, and Heuristics

We compared the algorithms in two measures: the aggregate cost and the running (calculation) time. We present the results for error bounds 5% and 50% only as the results for the other two (i.e., 10% and 25%) exhibit similar trends. In Figure 7, the plots at the left column show the aggregate cost for trajectories with different complexity values. We see that the Exhaustive Search and Genetic Algorithm (GA) give very close results. We know that, for the given resolution, Exhaustive Search finds the optimum solution. Since GA has the same resolution with Exhaustive Search, we can say that GA performs pretty well. Also, Heuristic 2 is not so bad; however, the same is not true for Heuristic 1. Another point we can see is that the complexity of the trajectory increases the cost; and so the low error bound. The reason is that the trajectory cannot be divided into small number of simple representations. More complex representations increase the aggregate cost.

The plots at the right column of Figure 7 show the running time of the algorithms. It is clear that Exhaustive Search is not practical with several seconds of running time, though it finds the optimum solution in terms of routing state cost. Under 5% error, the Exhaustive Search takes several hours to run in some cases. Heuristic 1 runs in almost no time. It is not a suitable choice because its performance is too low. It might be a choice only where time is very crucial and routing state cost is not much of an issue. Overall, GA and Heuristic 2 run in reasonably short time, while giving pretty good performance in minimizing the routing state cost. With this set up, GA runs in less than 0.2 seconds, and it is quite useful for the initial approximation of the trajectory.

6.2.2 Customization to Power-Scarce Networks

We also ran experiments for a customized objective (1) of cost and observed how the solution is adjusted to the new definitions. Specifically, we focused on the trade-off between packet header cost and network state cost. It is well-known that data transmission consumes larger power than storing the data [5, 12]. Thus, we aim to customize our framework such that the approximate trajectory is calculated while trying to reduce the amount of packet header state. We include the length of the trajectory piece the packets have to travel as a weighting factor in our objective (1).

This means that we have two cases to compare in calculating our routing state cost objective: *length independent calculation* and *length dependent calculation*. The length independent calculation is what we used in the above experiments, i.e., for every piece of trajectory, the packet header cost is the same with the base cost of the representation no matter what the length of that piece is. If it is a straight line then $C_P = 32$ bytes. In the length dependent calculation,

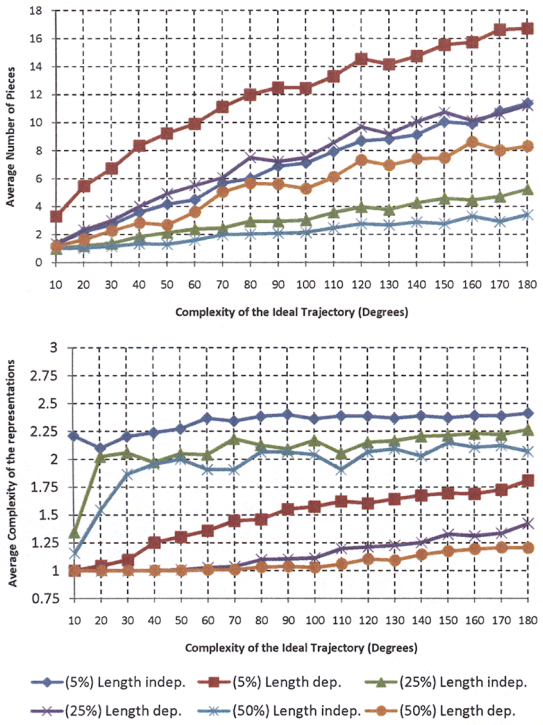


Figure 8. Customization of the routing state costs for a power-scarce network. Cost of packet header state can be made dependent on the length of trajectory piece the packets have to travel.

we assume that for long pieces of trajectories the packets has to go through more than one node and we calculate cost for every node passed through. This cost is dependent to the density of the nodes so we made another assumption. We assumed that for every 20pixels the packets visit another node. For example, for a 2^{nd} degree curve which is 148 pixels long, 8 nodes will be passed, and so C_P will be 32×8=256 bytes.

Comparing both cases, we observed the number of pieces that the trajectory was split into and the average complexity of the representations used. We used our GA implementation only. For simplicity we used 1 for straight line, 2 for 2^{nd} degree curve and 3 for 3^{rd} degree curve for the complexity values of the representations. We plotted these measures for trajectories with different complexities and different error bounds as in Figure 8.

We see that whatever the error bound is, the length dependent calculation causes the trajectories to have more pieces and be formed by less complex representations. The reason is that, in this calculation the weight of packet header cost is increased and the optimization framework tends to reduce the packet header cost by increasing the network state cost. As an example, in Figure 9, the left picture shows

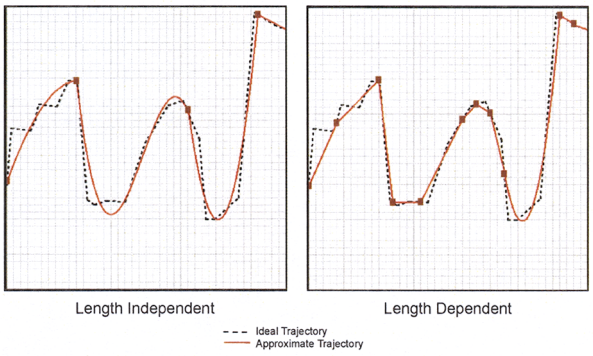


Figure 9. Sample runs for different cost calculations. Trajectory complexity: 120, Error bound: 25%

the approximate trajectory generated with length independent calculations. It is made up of 3 curves and 1 line. On the other hand the approximate trajectory generated with the length dependent calculations, on the right, has more pieces and most of them are lines. We can conclude that the approximate trajectory generated by GA is completely flexible and is adjusted for different customizations of ideal trajectory and the constraints.

7. Summary and Future Work

In this paper we presented an optimization framework minimizing routing state under application-based constraints. We formulated the problem of generating an approximate trajectory within an application-defined error bound such that Trajectory-Based Routing state is minimized. We showed that this approximation problem is NP-complete and hard to solve with regular brute force methods. To solve the problem, we devised four algorithms each having its advantages and disadvantages, and compared their performance and applicability in a real MANET setting. We also showed that our problem is customizable for different contexts such as a power-scarce sensor network.

The work in this paper is only one important part of the whole framework. Inclusion of this work into the larger framework with trajectory-based forwarding under a real user application is of high interest and deserves a full inspection. Considering the trajectory approximation problem, there are still points to be improved as well. New representations like B-spline and Bézier curves can be used for more flexible and optimum approximations. Some other parameters can be used also, e.g., node density, packet traffic density. The whole framework can be tested where there are more than one connection at a time.

References

[1] A. Alippi and G. Vanini. Application-based routing optimization in static/semi-staticWireless Sensor Networks. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, pages 47–51, 2006.

[2] B. T. Loo and J. M. Hellerstein and I. Stoica and R. Ramakrishnan. Declarative Routing: Extensible Routing with Declarative Queries. In *Proceedings of ACM SIGCOMM*, 2005.

[3] S. Basagni, I. Chlamtac, V. Syrotiuk, and B. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of ACM/IEEE MobiCom 98*, pages 76–84, October 1998.

[4] Chao Gui and Prasant Mohapatra. Virtual patrol: a new power conservation design for surveillance using sensor networks. In *Proceedings of IEEE International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005.

[5] J.-C. Chen, K. M. Sivalingam, and P. Agrawal. Performance comparison of battery power consumption in wireless multiple access protocols. *Wirel. Netw.*, 5(6):445–460, 1999.

[6] David Chu and Lucian Popa and Arsalan Tavakoli and Joseph M. Hellerstein and Philip Levis and Scott Shenker and Ion Stoica. The Design and Implementation of A Declarative Sensor Network System. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.

[7] Dragos Niculescu and Badri Nath. Trajectory-based forwarding and its applications. In *Proceedings of ACM MOBICOM*, September 2003.

[8] J. Follows and D. Straeten. *Application-Driven Networking: Concepts and Architecture for Policy-Based Systems*. IBM Red Book, 1999.

[9] G. Veltri and Q. Huang and G. Qu and M. Potkonjak. Minimal and Maximal Exposure Path Algorithms for Wireless Embedded Sensor Networks. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[10] Goldberg, David E. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, 1989.

[11] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[12] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen. A survey of energy efficient network protocols for wireless networks. *Wirel. Netw.*, 7(4):343–358, 2001.

[13] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of ACM/IEEE MobiCom 2000*, August 2000.

[14] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking*, pages 120–130, August 2000.

[15] Lucian Popa and Afshin Rostami and Richard Karp and Christos Papadimitriou and Ion Stoica. Balancing the Traffic load in Wireless Networks with Curveball Routing. In *Proceedings of ACM MOBIHOC*, 2007.

[16] M. Perillo and W. Heinzelman. DAPR: A protocol for wireless sensor networks utilizing an application-based routing cost. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, June 2004.

[17] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 1999.

[18] Nikola Milosavljevic and An Nguyen and Qing Fang and Jie Gao and Leonidas Guibas. Landmark Selection and Greedy Landmark-descent Routing for Sensor Networks. In *Proceedings of IEEE INFOCOM*, 2007.

[19] G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing in mobile ad hoc networks. In *Proceedings of Workshop on Wireless Networks and Mobile Computing*, pages D71–D78, 2000.

[20] C. Perkins. Ad hoc on demand distance vector (aodv) routing. IETF, Internet Draft, draft-ietf-manet-aodv-00.txt, November 1997, 1997.

[21] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM CCR*, 1994.

[22] E. Pierluigi Crescenzi, Viggo Kann. A compendium of np optimization problems. *http://www.nada.kth.se/ viggo/problemlist*.

[23] C. Santivanez and R. Ramanathan. Hazy sighted link state (hsls) routing: A scalable link state algorithm. Technical Report BBN-TM-1301, BBN Technologies, Cambridge, MA, 2001.

[24] P. Sinha, S. Krishnamurthy, and S. Dao. Scalable unidirectional routing with zone routing protocol. In *Proceedings of Wireless Communications and Networking Conference (WCNC)*, pages 1329–1339, 2000.

[25] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[26] Timothy G. Griffin and Joao L. Sobrinho. Metarouting. In *Proceedings of ACM SIGCOMM*, 2005.

[27] Vinod Muthusamy and Milenko Petrovic and Hans-Arno Jacobsen. Effects of Routing Computations in Content-Based Routing Networks with Mobile Data Sources. In *Proceedings of ACM MOBICOM*, 2005.

[28] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2002.

[29] Xiaowei Yang and David Clark and Arthur Berger. NIRA: A New Inter-Domain Routing Architecture. *IEEE/ACM Transactions on Networking (ToN)*, 15(4):775–788, August 2007.

[30] M. Yuksel, J. Akella, S. Kalyanaraman, and P. Dutta. Free-Space-Optical Mobile Ad Hoc Networks: Auto-Configurable Building. Submitted to Wireless Networks (after major revisions), 2006.

[31] M. Yuksel, A. Gupta, and S. Kalyanaraman. Contract-Switching Paradigm for Internet Value Flows and Risk Management. In *Proceedings of IEEE Global Internet Symposium*, 2008.

[32] M. Yuksel, R. Pradhan, and S. Kalyanaraman. An Implementation Framework for Trajectory-Based Routing in Ad-Hoc Networks. *Ad Hoc Networks, Elsevier Science*, 4(1):125–137, January 2006.