HeuristicIoT: A Framework for Augmenting Heuristic Search Algorithms by Internet-of-Things Data

Mohammed Anas, Sherif Khatab and Amr Badr
Faculty of Computers and Information Computer Science Department
Cairo University
Giza-Egypt
harouna90@gmail.com, s.khattab@fci-cu.edu.eg, amr.badr@fci-cu.edu.eg

Abstract: One of Internet of Things (IoT) big opportunities is the huge data that can be collected. As human heuristics have been shown to improve the performance of complex problems and deliver more accurate results, IoT data can be used to extract these heuristics. In this paper, we present a framework to capture human heuristics from taxi drivers using data collected from sensors deployed in the taxis. The captured heuristics are used to create initial chromosomes for a Genetic Algorithm to solve the Travelling Salesman Problem (TSP). A dataset collected from 10,357 taxis in Beijing for 18 months was used. The quality of the TSP solutions was improved by up to 49% as compared to the same genetic algorithm with randomly initialised chromosomes. Our results show a promising potential for augmenting heuristic search algorithms by data collected from the IoT.

Keywords: Internet of Things (IoT), Genetic Algorithm (GA), Initial Population, Taxi Trajectories, Taxi Sensors, T-Drive Data.

1. Introduction

Number of Research studies on how the human mind solves NP-complete problems showed that the human mind capable of giving more accurate solutions faster [1]. As humans face a lot of complex problems, the human mind does not take the hard way of trying all possible combinations in the solution space but uses that mysterious factor called human heuristics [2], [3].

This work is focused on capturing the human decision when facing multiple choices, especially if they affect each other. An example is when a human has to choose between two paths both could get him to the same destination but with different trade-offs in time, distance or cost in general. With this class of problems, a person mind shows great solutions by processing the complete picture.

Lately, a small sensor, such as the active RFID tag, have the ability of converting daily-life dumb objects into smart ones [4]. These types of smart objects can gather data about human conduct, habits and behavior. Typically the human heuristics are in some way embedded within the data gathered simply by these smart objects. As an example, a GPS sensor can convert a taxi (or a public transportation vehicle) into a smart vehicle, whereby the collected data can enhance routing and pick-up strategies [5], [6]. Also, data are available about the sequence of people visiting shopping malls [7], people traversing roads [8], and even about trajectories of animals [9]. The possibilities are unlimited and the combinations are infinite to find creative and inspiring ways to use the collected data. However, these data can be wasted or deleted without making the most of it [10], [11]. The problem of how to capture and use human heuristics is a point of focus in this research [12].

In this paper we present a framework, namely HeuristicIoT, for augmenting heuristic search algorithms, such as the Genetic Algorithm (GA) [13] and Ant Colony Optimization [14], using data collected using IoT. A hook points to which IoT data can be plugged are identified. For these algorithms, the initial solutions are critical step to begin with. IoT data can provide these initial solutions.
number of target points on real map and a number of
taxi trajectories, the framework can generate smarter
initial solutions for the GA optimization algorithm. The
generated initial solutions using HeuristicIoT resulted
in the improvement of the final results.

We apply HeuristicIoT on data collected from Taxi
drivers to solve the Travelling Salesman Problem
(TSP) using the Genetic Algorithm (GA) (Fig. 1).
Instead of starting the GA with random initial
solutions, we generate the initial solutions or parts of
them by using the trajectories collected from taxi
drivers. The Heuristic based initial solutions have the
same order in visiting problem Target points that the
taxi drivers took based on their heuristic thinking.
Because HeuristicIoT works on real-world map
problems, we present a modified TSP with hidden edge
costs.

2. RELATED WORK

In this section, we present some related work on
human heuristics modeling and applications, using
IoT data for problem solving, and the TSP problem.

The human ability of solving problems without an
exhaustive search was the point of interest of many
scientists. For example, a research has been held
where test subjects received some problems and they
started solving it on paper. Pizalo has demonstrated
that human solutions for some problems (e.g.,
Traveling Salesman Problem (TSP)) are better than
most AI solutions [15]. Another research has sought
for a better understanding of human approach in
solving complex problems [16]. The research was a
review on the TSP and related problems.

Understanding the human mind and humans
heuristics in solving problems was a target for many
researchers. A new heuristic model that mimics
human decision-making was proposed and tested on
bandit problems, in which a decision-maker gets
reward-or- failure feedback when choosing
repeatedly between two alternatives [17]. One of the
methods that has been used in generating and finding
human heuristics is hyper-heuristics [18]. Moreover,
some papers were focused on solving problems (e.g.,
fastest route) based on human intelligence extracted
from historical taxi trajectories [19]. The paper has
used GPS logs to build a landmark graph and to
estimate its edges costs. Additionally, they introduce
a two-stage routing algorithm based on the previous
graph to get the fastest route. In general, raw big data
(e.g., taxi trajectory data) consisted of high-valued
data mixed with noisy and erroneous data. These
high-valued data can be sensed using different
technologies in order to generate more revenue,
reduce risk, and predict future outcomes with greater
confidence in low cost [20].

The research in [21] presents a TSP problem with
fuzzy cost coefficients (FRTTSP) to develop realistic
solutions for real problems on real-life map. Eugenic
bacterial memetic algorithm (EBMA) has been
chosen to solve the given problem. We used our
proposed HeuristicIoT framework to solve variant of
TSP problems in which the edge costs are not known.

3. Travelling Salesman Problem (TSP)
with Hidden Edge Costs

The paper aims at augmenting heuristic search
algorithm by IoT data to solve complex problems.
This section presents a formulation of a specific
complex problem and presents an attempt to solve it
which goes under the name TSP with hidden edge
costs. We start by formulating the traditional TSP
then we describe the modified problem.
3.1 Travelling Salesman Problem (TSP)

The work aims at extracting full or partial solutions from the way humans solve complex problems. For concreteness, we selected the TSP as a case study. The TSP is an NP-complete problem and is a frequently solved problem by humans. TSP has characteristics that get along with the human way of thinking. Finally, the inputs, outputs, and structure of the TSP have direct correspondence with the structure of the data that can be collected from taxis (e.g., routes, locations, durations).

A solution to the TSP is a trip that begins with a target node and completed by coming back to the start target node while passing every target node exactly once. Another way to describe the problem as an Integer Linear Program is as follow [22].

\[
x_{ij} = \begin{cases} 
1, & \text{the solution goes from target node } i \\
0, & \text{to target node } j \\
0, & \text{Otherwise}
\end{cases}
\]

\[
\min \sum_{i=1}^{n} \sum_{j=0,j\neq i}^{n} c_{ij} * x_{ij}
\]

\[
0 \leq x_{ij} \leq 1 \quad i,j = 0,...,n
\]

\[
u_i \in \mathbb{Z} \quad i = 0,...,m
\]

\[
\sum_{i=0,j\neq i}^{n} x_{ij} = 1 \quad j = 0,...,n
\]

\[
\sum_{j=0,j\neq i}^{n} x_{ij} = 1 \quad i = 0,...,n
\]

\[
u_i - u_j + nx_{ij} \leq n - 1 \quad 1 \leq i \neq j \leq n
\]

The inputs to the TSP problem are the set of n cities and the edge costs $c_{ij}$ between each two cities i and j. The output of the TSP is the minimum tour cost and the set of integers $x_{ij}$, representing the visiting order of the cities.

3.2 TSP with Hidden Edge Costs

The problem that we tackle in this work is a slightly modified version of the TSP. Our goal is still finding the shortest route for a trip that achieves the previous conditions. However, the inputs of the problem are modified. Instead of providing the edge costs, a set of time stamped trajectory points are provided. The trajectories represent moving objects (e.g., taxis) and are modeled as a time-ordered sequence of points, whereby each point is the geographic coordinate of each taxi at a given time-stamp.

In our problem, the graph nodes are points on the map, represented by the longitude and latitude. If we fix two locations i and j on the map and consider only a particular time window $T$, it is possible that a number of taxis passed between the two locations with potentially different trip durations. The edge cost ($C_{ij}^{T}$), which is hidden in our problem, is time-dependent. $C_{ij}^{T}$ is the minimum trip-time over all taxis starting from location i to location j during the time window $T$.

Based on the previous definitions, our modified TSP problem can be defined as follows; given a set of locations L with $|L| = n$ and a set of trajectory points of taxi drivers, find the shortest tour to visit all L locations exactly once, where edge costs are computed for a time window $T$, which is not given as input. We note that the edge costs of the TSP
problem are hidden because the time window, on which the edge costs depend, is not given.

Figure 4. The process of solving complex problems. IoT data can be used in the components marked with a black dot.

4. The HeuristicIoT Framework

In this section, we give an overview of our proposed framework, which contains three building blocks. Then, we explain the process of mining the IoT data and accordingly generating the GA’s initial chromosomes (which is the first building block) and fitness table (the second building block). Finally, we describe the GA, (the third building block) of the proposed framework.

4.1 Framework Overview

Figure 4 displays the process of solving a complex problem using a heuristic search algorithm. The dotted blocks represent the steps in which IoT data can be used. First, there is an instance of a complex problem. Next step is the filtering and transformation of the problem instance. The instance is accepted or rejected based on whether or not the search algorithm can solve it accurately enough. Then, the instance transformed into a representation that suits the search algorithm, for instance, a chromosome in the GA search algorithm. After that comes the search algorithm (e.g., GA, Ant colony optimisation or greedy algorithm). An algorithm needs the setting of its configuration parameters that affects its results. Before giving the final solution a reverse transformation may be needed.

In the proposed HeuristicIoT framework, the step of transformation and filtering is changed based on the IoT data. For instance, a problem instance of the TSP can be rejected if one of the cities has an invalid position (e.g., a node in water is impossible to reach using cars). HeuristicIoT changes the algorithm itself. For example, it changes the way of generating the initial chromosomes in GA. In HeuristicIoT, the initial chromosomes are generated from the data. Moreover, the fitness table of the GA, which is used to hold the cost in time to move from one target point to another, is generated from the data as well. This is different than the traditional way of having the fitness table as an input. Figure 1 summarizes the role of IoT data in the HeuristicIoT framework: generating GA’s initial population and generating the fitness table.

The proposed framework has three building blocks, namely generating the initial population, computing the fitness table to be used by the GA, and running the GA. These components are described in detail in the next subsections.

4.2 Generating Initial Chromosomes

First we have number of chromosomes equal to the number of our taxi drivers in the data set. We generate our chromosome by looping over all our taxi driver trajectory points. If a trajectory point has a distance less than 0.2 kilometers between it and a target point we consider this point visited by this driver. We add every visited point to our chromosome (i.e. a possible TSP solution) maintaining in the same time the order of the appearance as shown in Algorithm 1. If a driver did not pass all our target points we complete the remaining ones randomly. The partial passing order from our taxi drivers may effect in a good way our solutions during our genetic programming cross over stage later by the chance of cutting this part and attaching it to another uncompleted solutions.

The framework’s main purpose is to get the order which taxi drivers took to pass our target points as shown in figure 2. As shown in figure 2 we have two kinds of our taxis problem solvers, one who bypassed all our problem nods as a complete chromosome in the our initial population. On the other hand, we have a taxi drivers how traversed number of target points. Although it’s not a complete solution but still there is a probability that his passing target nodes sequence is a good part of the final solution. which could enhance our GA offspring as in Figure 2b.

4.3 Computing The Fitness Table

We calculate the time between every 2 target points by looping over every taxi and check if the distance between a taxi and one of the targets points is less than 0.2 kilometers. If so we start calculating the time from point $target_i$ till we reach another target point $target_j$. We sum the time between every 2 trajectory points between $target_i$ and $target_j$. After calculating the time we store it in our 2d array indexed by the two target points. If another taxis drivers traveled between the same two target points with different times we store the minimum one between them in the 2D array fitness table as shown in figure 6.

4.4 The Genetic Algorithm

The GA has two steps. First generating the initial chromosomes, and in our case we described it in the previous subsection. Second is GA three operators and in our case we worked with the standard ones. We used one-point cross over, bit string mutation and Elitism selection.

In the end we get the fittest chromosome that holds the best sequence of passing our target points in the shortest time.
5. Experiments

![Algorithm 1](image)

**Figure 5.** 3D array holding the time for every taxi passed between every two TSP problem targets. $m_k$ is the number of times Taxi $k$ passed from node $i$ to target node $j$ in the data set.

![Figure 6](image)

**Figure 6.** 2D array holding the $T_{ij}$ the minimum time between every two targets ($i$ and $j$) of the TSP problem.

To test our framework we compared three approaches, brute force (BF), which was only possible in small problems, original genetic algorithm methodology (RAND), which uses an initial population that was generated randomly, and the proposed framework (HeuristicIoT), that uses the minimum time spam in which all taxis travel between every two target points. We used the BF to get an insight of how far our solution was from the optimal solution. In what follows, we describe the basic components for running our experiments (calculating the ground truth for comparing purposes, generating an instance of the travel sales man problem, the GA Parameters, and finally solving the problem with the a GA which generate initial solutions randomly).

**Calculating the Ground Truth.** In order to make sure that the comparison of the studied approaches was done on an equal ground, and to calculate the fitness of the final chosen chromosomes; we created a ground truth of each TSP problem. The ground truth allowed us to compare the three approaches studied.

Creating the ground truth was done on the following stages. First, we determined a specific time spam in which all taxis travel between every two target points during time measured. Second, we stored the minimum of all taxi travel times between every two target points.

**Generating TSP Problem Instances.** Generating an instance of the TSP problem was done in three steps. First, we generated a set of random target points with
longitude and altitude that is guaranteed to be within Beijing boundaries, which are (39.7015338, 115.9230211) at the bottom left and (40.1060691, 116.8609789) at the top right. To create each target point, we used the following formula:

\[ X = \text{Lat\_Range\_min} + (\text{Lat\_Range\_Max} - \text{Lat\_Range\_min}) \times \text{UniformRandom}[0,1] \]

\[ Y = \text{Long\_Range\_min} + (\text{Long\_Range\_Max} - \text{Long\_Range\_min}) \times \text{UniformRandom}[0,1] \]

In the equations above, \text{Lat\_Range\_min} = 39 and \text{Lat\_Range\_Max} = 40 because Beijing latitude is between 39.701538 and 40.1060691 and \text{Long\_Range\_min} = 116.09 and \text{Long\_Range\_Max} = 116.7 because Beijing longitude is between 115.9230211 and 116.8609789.

To make sure that the randomly generated target points were not created in invalid places (e.g., inside a lake), we added a condition that accepts the generated point if it has 0.2 kilometers distance from any trajectory point of any taxi in the data set. Finally, we had to make sure that our graph formed by the generated target points is fully connected. An edge between any two points exists if and only if at least one taxi driver had these two points in its trajectory. We used BronKerboschClique [23] algorithm to find the maximum clique that can be formed from the target points. The result from the last step was the problem graph ready to be solved.

**GA Parameters.** The number of chromosomes in the GA was equal to the number of taxis in the dataset. Each chromosome was an array that holds the target points. The order of visiting the target points (i.e., a possible TSP solution) is the order of appearance of the target points in the chromosome. We used Single point crossover [24], bit string mutation [25], and Elitism selection [24]. We chose to stick with the standard GA in the way of crossover, mutation and selection to make sure that any changes in the final results would not be the result of changing the approach of generating initial chromosomes and calculating the fitness table.

![Figure 7](image-url)  
**Figure 7.** Quality of TSP solutions of studied TSP Problems for GA with chromosomes initialized from collected data (HeuristicIoT), GA with random initial chromosomes (RAND), and brute-force (BF). BF was feasible only for small problem sizes.

We got the time between every two target points using every taxi that passed between them. This was simply done by looping over every trajectory point of every taxi and checking if the distance between the taxi's trajectory point and any point \text{target}_i from the target points is less than 0.2 kilometers. Then, we calculated the travel time of the taxi starting from point \text{target}_i till reaching another target point \text{target}_j. This calculation was done by summing up the time between every two-trajectory points starting from \text{target}_i and ending with \text{target}_j. We added the calculated time to figure 6 indexed by the two target points, in which each cell contains the travel time between these two points by all Taxis that passed the two points. This 3D array is shown in Figure 5 The 3D array was used in the GA to provide chromosome fitness calculation.

**Solving Using Random Initial Population.** In the RAND approach, we got the initial chromosomes set randomly by generating random numbers between 0 and the total number of target points minus one. We also selected one value at random from the 3D array mentioned previously to represent the distance between two target points.
5.1 Results

We experimented with problems with different sizes (i.e., number of target points), whereby we had small problems (number of target nodes between 6 and 9) and bigger (23 and 25 target nodes). We ran every experiment ten times. We experimented with two ground truth tables, where the data of one table was collected between 13:00 and 14:00 and the other one was between 16:00 and 18:00. Figure 7 visualizes the results in terms of the best-obtained time in solving each problem. The proposed HeuristicIoT approach was the closest one to BF.

Figure 8 and 10 show the cumulative distribution function (CDF) for three studied algorithms in the first and third problem. As the figures show, HeuristicIoT was the closest one to the optimal solution generated by the Brut force Algorithm, it also shows in figure 8 that 0.8 of the framework solutions are under 1000s for the hole trip but for the RAND algorithm the best solution for it above 1000s.In figure 10 For HeuristicIoT the probability of having a solution less than 2000s is 0.9 although for RAND it is 100% for having 2000s trip which shows a great advantage for HeuristicIoT algorithm.

Figure 9 and 11 show the cumulative distribution function (CDF) for our two competitive algorithms in the second and forth problem. Although these problems have mush bigger search space the 2 figures shows that HeuristicIoT Line is steeper than RAND which means it gets a very good solutions in early time and it maintain this good performance unlike RAND which shows more distributed solutions. The reason for such behavior is because HeuristicIoT works on the minimum time for passing between two target nodes by any taxi driver unlike RAND which have to consider mush more possible paths, which reflects the real life decision search space. Figure 9 the longest trip for HeuristicIoT has time under 5000s while for RAND the shortest one starts from above 10000s.In Figure 11 it is 0.1 probability to have a solution above 13000s for the trip which is the worst for this algorithm but it's 0.7 in RAND to have a solution larger than or equal 13000s and the worst solution in this case is 19331s.

6. Conclusion and Future Work

This paper shows that using initial solutions captured from human heuristics had a potentially promising effect on the quality of the final solution.
The framework works on the Genetic Algorithm to modify the process of generating initial population based on data collected from taxi trajectory. To test the presented framework, we made a TSP with hidden costs on a real map. We used the TSP as an experimental test bed on a dataset generated from 10,357 taxis in a period of 18 months. We tested on different TSP problems with different sizes. Based on our experiments, our approach produced results that were an improvement over the traditional way of generating initial solutions (randomly). The final solutions generated by the framework had smaller total trip time.

Although we managed to use the available data to capture a partial image of the heuristics used by the taxi drivers, we think that merging data from other types of sensors can give us more insights. We would like to test HeuristicIoT on other NP-hard problems and also on other optimization algorithms, like Ant Colony Optimization, which also generates the initial population randomly. To have better insight into human heuristics we want to test our framework using data sets generated by drivers solving a TSP problem like delivery boys or delivery service companies. It is also a point of interest for us to test our framework on larger data sets, because the more data we have the more we would be able to build more accurate and more useful models of human heuristics.

References


[17] M. D. Lee, S. Zhang, M. Munro, and M. Steyvers, “Using heuristic models to understand human and optimal decision-making on bandit problems,” in Proceedings of


Mohammed Anas is a graduate student at the Department of Computer Science, Cairo University and a Teacher Assistant at October University for Modern Sciences and Arts (MSA). Before that, Mohammed received his B.Sc. in Computer Science in 2011 from Cairo University. His research interests include Internet Of Things (IoT), Computer vision, and Swarm intelligence. Presently, he is investigating approaches for understanding and using human heuristics to solve complex problems.

Sherif Khattab is an associate professor in the Department of Computer Science, Faculty of Computers and Information, Cairo University. His research focuses on the systems aspects of cybersecurity as applied to cloud computing, Internet-of-Things, electronic voting, and Big Data. Sherif has an h-index of 10 (Google Scholar Index) with 45 publications and he has successfully supervised 15 graduate students. He had interned in Google Inc., Ericsson Data Networks, and Bosch Research. He obtained his Ph.D. and M.Sc. in computer science from the Department of Computer Science, University of Pittsburgh in 2008 and 2004, respectively. Sherif has a Bachelor's degree in Computer Engineering from Cairo University in 1998.

Amr Badr is a Professor in the Department of Computer Science, Faculty of Computers and Information, Cairo University. He received his BS in Engineering with Honors in 1986. He received his MSc and PhD in Computer Science from Cairo University in 1995 and 1999. His research interests are Intelligent Systems, Bioinformatics, Medical Imaging and P-systems. He has published more than 150 journal research papers in these areas.